Detecting Coastal Litter with Neural Networks

Candidate Name: Linden Smith

This dissertation was submitted in part fulfilment of requirements for the degree of

MSc Advanced Computer Science with Big Data

Supervisor: Dr Marc Roper

Department of Computer and Information Sciences

University of Strathclyde

26/08/2019

Word Count: 22,000

This dissertation is submitted in part fulfilment of the requirements for the degree of MSc Advanced Computer Science with Big Data at the University of Strathclyde.

I declare that this dissertation embodies the results of my own work and that it has been composed by myself. Following normal academic conventions, I have made due acknowledgement of the work of others.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to provide copies of the dissertation, at cost, to those who may in the future request a copy of the dissertation for private study or research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to place a copy of the dissertation in a publicly available archive. (please tick) Yes [ ] No [ ]

I declare that the word count for this dissertation (excluding title page, declaration, abstract, acknowledgements, table of contents, list of illustrations, references and appendices is 22,000.

I confirm that I wish this to be assessed as a Type 3 Dissertation.

Signature:

Date:

Abstract

Coastal litter is a significant problem in Scotland, not only polluting and endangering wildlife but also harming the vital tourism sector. Aerial images of the coastlines, aswell as spreadsheets containing information for the majority of these images, were made available by Scrapbook, an organisation dedicated to combating this issue. The aim of this dissertation is to utilise this information in the design, application and evaluation of deep learning based systems for automatically classifying these aerial photographs by their level of litter accumulations, and to form recommendations for a fully automated system based on these results.

In the process, a variety of solutions to domain problems are explored, such as: insufficient samples, class imbalance, massive terrain variety, and the automated collection of features for both incorporation to training and the automatic presentation of findings. Additionally, significant processing of the supplied dataset was necessary, some utilities involved forming a part of the proposed automated system.

Ultimately the limited dataset prevented the development of a sufficiently effective model, performances poor for all classes other than the most numerous. However, the efficacy of proposed methods such as data augmentation and mixed input networks was validated, and it is proposed that through methodologies employed in this dissertation, when more images become available, an accurate system can be deployed.

Contents

List of Figures

List of Figures

List of Tables

Preface/Acknowledgements

I would like to acknowledge Dr Marc Roper for his role as project advisor, Sophie and the volunteers at Scrapbook for their time, input and data, and L3C for providing a remote server with a Tesla V100 16GB GPU.

Chapter 0. Preface/Acknowledgements

Chapter 1

Introduction

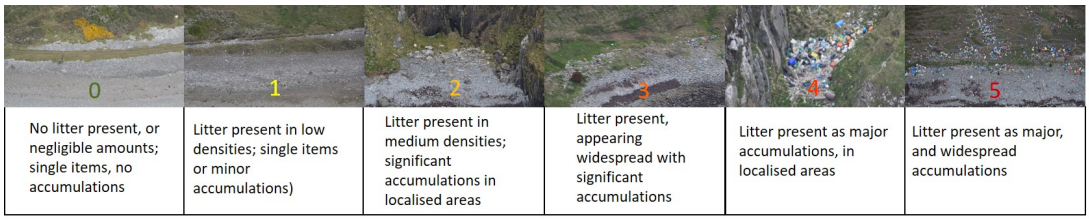


Figure 1.1: Litter Categories (credit: https://www.scrapbook.org.uk/

Litter on Scottish coastlines costs an estimated 16 million a year (Scotsman, 2013) to both third sector and governmental organisations, and poses a danger not only by polluting but also threating wildlife, and harming tourism (Society, 2018a).

SCRAPbook is a volunteer project targetting litter on Scotland's coastlines (Scrapbook, 2019). By managing volunteer aircraft crews, photographers and clean up operations, their aim is to monitor litter accumulations across the coastlines and coordinate clean ups in cases of significant (and ideally reachable) accumulations. Most of the supplied images were captured by a photographer, but they are increasingly moving towards automation, which involves an unmanned camera automatically capturing images as the aircraft passes a location. Currently, aerial photographs are examined by hand, a time consuming process; given their high resolution, it is necessary for volunteers to zoom around the image in search of litter, then fill in several columns of an excel spreadsheet document for analysis, including categorising the image as to the

level of litter accumulations — see Figure 1.1. Automating these classifications would allow a reallocation of resources that, for a third sector project, would be especially beneficial.

170GB of high resolution images are available, as-well as spreadsheets which contain information such as levels of accumulation for (most of) the 93 sectors they have segmented our coastline into. Therefore, with both images and labels, the opportunity to apply a machine learning algorithm presents itself. We therefore proceed towards the goal of reliably detecting litter accumulations in these high resolution images, as an exploration and potential proof of concept that an automated system could replace the current methods employed by Scrapbook; failing this, to understand the cause, and make recommendations on how this can be achieved in the future.

With only around $10,000$ unique, usable images in the dataset out of $14,000$ supplied, and a far smaller subset containing litter accumulations of any significant degree, training the network is difficult. Class Five accumulations only present 83 times in the entire dataset; class 0 accounts for 70% of images. Therefore the problem is one with significant class sample inequality, and of significantly low samples for litter positive images; the latter inducing a vulnerability to over-fitting. We also note there is a huge variety in terrain among the images, which are mostly of resolution 6000 by 4000. Towards the end of making most use of the data provided, we proceed with the image classification approach in mind rather than semantic segmentation, which our labels are not suited for; semantic segmentation would require an excess of labelling.

Many issues in building and training this system require address. From the data processing stage, the spreadsheets must be collated, resolving the many conflicts in notation, column names, the duplicate and missing entries, and filtering those rows which represent unusable data. We must also make best use of these sheets by, for example, utilising available features to whittle down the set to only images which represent the problem domain, sifting out, for example, images which do not depict coastlines, thereby preventing these images from introducing conflicting representations to the network, and weighing the cost of performing these operations — how will the network react when exposed to these in a potential deployment? More important than

these preprocessing steps, however, is the accurate association of images with labels, so that the supervised learning system can be trained. This not only relates to the design and creation of the prototype model, but also for the future if more images are to be gathered and labelled.

With many duplicate image names inside the spreadsheets, even quite significantly among sectors, associating image-label pairs stands a significant barrier to the creation of a prototype model. If a single sector were to be incorrectly labelled, conflicting information would prevent appropriate learning. Many images in the spreadsheets describe images that either do not exist or were not supplied; and many images in the folders supplied were not labelled in the spreadsheets. Further, many images appear to be labelled in a specific row of a spreadsheet, complete with its sector and image filename, sometimes even agreeing litter accumulation level and type, making identification difficult, but are in-fact separate, but similar, images. This can be caused by resetting the camera, clearing its memory, resulting in the images starting from 1 again. Even a handful of such images could harm learning through the provision of conflicting information — e.g., a sector 0 mistakenly identified a class 5, which has only $81 - 83$ images, depending on strictness of identification, out of all $10,000$ could seriously harm learning given it makes up more than 1% of class 5 samples. With folders full of images titled 'DSC_001.JPEG', and quite literally hundreds of some image titles, even those inside sectors; resolving this involves a more involved approach than simply assuming the sector on the folder name matches the spreadsheet with the same sector in its title, and collating this with image names.

In the end, a methodical approach is required involving detailed cross-reference of EXIF information.

Prior to building the network, structures are needed to pass batch sized subsections of the full data given it cannot fit in memory, even with the smallest resolution. Generators are then proposed.

The real issue however lies in training an adequate network given how few positive samples there are, and the class imbalance inherent in the data; low sample training sets leads quite easily to over-fitting, and class imbalances lead to sample dominance —

i.e., filters learned are mostly for the most numerous class as learning representations of this class leads to a higher accuracy than the others. We therefore must apply various sampling methods, creating utilities such as over-samplers and under-samplers, and weighting tools such as weighted loss functions. Additionally, we wish to express to the network that should a class 0 be mistaken for a class 5, this misclassification is *worse* than a class 0 for a class 1, so perhaps should be assigned a higher loss; so we carefully monitor the classifications via confusion matrices, attempting to solve the problem, if it presents, in the form of a weighted loss function.

Further to the above mentioned barriers of class imbalance and over-fitting, data augmentation is explored. This aids the imbalance by, in the additive approach, generating new samples. In the live approach, images are augmented by the generator, new transformations each time, before they are passed to the generator. Both approaches aid generalisation by the introduction of new images from, if applied correctly, the sample space from which future images should present, improving robustness to variation.

In tackling the huge variety of terrain and litter in the images, attempts at multiple input models are made, feeding not only images but also numeric features including the sector, latitude and longitude of an image. All of these factors can be automatically stripped from the EXIF data of future images, meaning the future application which contains the model can easily auto strip this data from each image exposed for prediction. Care is taken not to over-fit the data, which presents already due to the low number of samples, though when given more information of this kind, may lead to further over-fitting. For example, a prime concern is that the network may learn all coordinates where litter have appeared in the training set, over-fitting exactly, which will not generalise to all test samples if random split used, or none if split by sector. This is a potential flaw of splitting the data randomly versus by sector; however a by sector split means performance is underestimated, given training on a sector learns representations for said sector, which often have internal commonalities as-well as general similarities to the data as a whole.

Additionally, we search many architectures, configurations and hyper-parameter combinations to find the best model with the best performance possible; so the problem

is also one of optimisation.

In 'A Guide to Convolutional Neural Networks for Computer Vision' by Salman Khan et al (Khan et al., 2018, p. 1), Computer Vision is defined as:

> *''the science ... that seeks to develop methods which are able to replicate one of the most amazing capabilities of the human visual system, i.e., inferring characteristics of the 3D real world purely using the light reflected to the eyes from various objects.''*

Machine Learning algorithms are often used for image classification tasks. Prior to the advent of Deep Learning, methods such as the Support Vector Machine (Khan et al., 2018, p. 22) and handcrafted feature algorithms (Khan et al., 2018, p. 14) were used for such tasks, with the limitation that the engineer must build their own 'filters,' or 'feature descriptors.' Convolutional Neural Networks on the other hand are able to automatically learn such features. (Khan et al., 2018, p. 14, 21)

In recent years, Deep Learning, that is, deep layered neural networks, have surpassed prior methods, including shallow networks. Modern algorithms can reach very high levels of accuracy with the right approach and enough, good quality training data. In-fact, some neural networks approach and exceed human performance, such as with AlexNet on the famous ImageNet dataset (Krizhevsky et al., 2012, p. 1) (Khan et al., 2018, p. 102), shown in Figure 2.1. For image classification tasks, the Convolutional Neural Network, a variant inspired by biological processes in the visual cortex of humans and animals Fukushima (1980) (Khan et al., 2018, p. 31) (Géron, 2017, p. 354), is perhaps most appropriate (Khan et al., 2018, p. 43) (Géron, 2017, p. 355).

Chapter 2

Overview of Relevant Literature

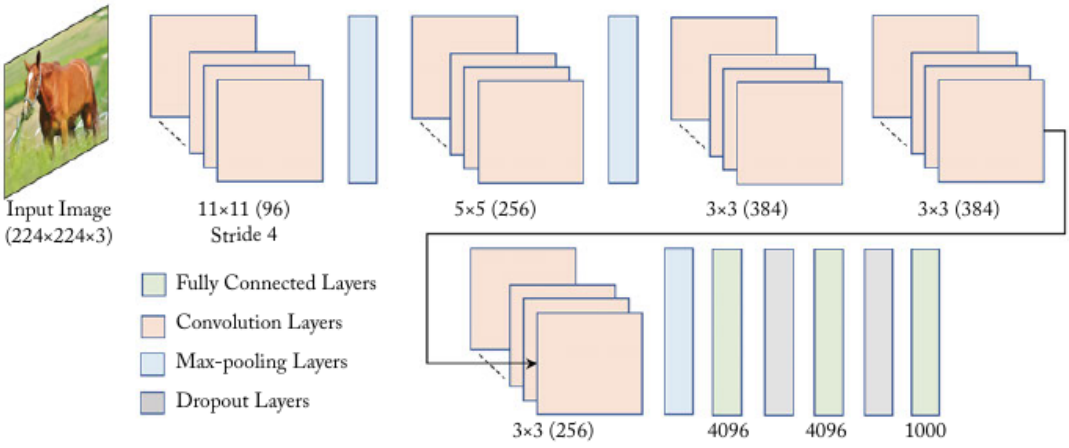


Figure 2.1: AlexNet Architecture (taken from ''A Guide to Convolutional Neural Networks for Computer Vision'' , (Khan et al., 2018, p. 103)

To understand convolutional neural networks, such as AlexNet in Figure 2.1, it is important to first discuss the origins of the convolutional neural network, and neural networks in general.

2.0.1 History of Neural Networks

In 1943, W. McCulloch and W. Pitts, inspired by developments in the study of mammal brains (Géron, 2017, p. 254) (Khan et al., 2018, p. 40), proposed a computational

model based on the synergy of neurons (McCulloch and Pitts, 1943). This proved to be the beginning of a wave of interest in Artificial Neural Networks, perhaps the next important advancement being Rosenblatts inception of the Perceptron in 1958 (Rosenblatt, 1958). Rosenblatt built on the neuron first proposed in 1943 to create his one-layer network of neurons based on the so called Linear Threshold Unit(Géron, 2017, p. 256) (Rosenblatt, 1958). Perceptrons are capable of performing binary classification only on datasets that are linearly separable, and are also unable to solve the so called XOR problem — the exclusive or that outputs true only when one input is true and the other is false. Years later, it was recognised that a Perceptron with more than one layer, called a Multi-Layer Perceptron, is far more effective (Géron, 2017, p. 250), being able to solve the XOR problem.

Further developments were the discovery of ''Backpropagation,'' short for ''the backwards propagation of errors,'' as an effective and viable method of training (Rumelhart et al., 1985), as-well as the increases in available computational power in recent years, particularly Graphics Processor Units (GPU), enabling the training of more complex networks (Khan et al., 2018, p. 8).

2.0.2 Origins of Convolutional Neural Networks

Feed-forward neural networks such as the Multi-Layer Perceptron are limited when it comes to high dimensional input data. The convolutional Neural Network is superior for image classification and recognition in that the MLP, and feed forward networks in general, are limited by the so called ''curse of dimensionality,'' (Géron, 2017, p. 479), due to full connection between nodes, and are therefore inappropriate for high-resolution images; which we are using. Geron writes, in Hands on Machine Learning with Scikit Learn and TensorFlow, on using CNNs over regular Deep Neural Networks (DNNs), ''though this works fine for small images it breaks down for larger images.'' In 'A guide to Convolutional Neural Networks for Computer Vision,' Khan et al state that, '[CNNs are]...essential for cases where we want to learn patterns from high-dimensional input media. (Khan et al., 2018, p. 43). In addition to requiring fewer parameters, CNNs carry the benefit of maintaining positional information (Khan et al., 2018, p. 43).

Fukushima and Miyake proposed the earliest form of the CNN (Fukushima, 1980). The ''Neocognitron,'' a model for pattern recognition, introduced the two main layer types used in modern day CNNs: convolutional, and downsampling. They write, in their original paper, ''The neocognitron can learn patterns and filters that would ordinarily require the architect to design.'' For example, in our case, we may wish to apply a filter that sharpens certain colour spectra such as blues, and dulls others. With a CNN, this is not necessary, as it will (ideally) learn a similar filter in training. Fukushima and Miyake go on to state, ''if a set of stimulus patterns are repeatedly presented to it, it gradually acquires the ability to recognise these patterns.'' This makes the process far more automated than a typical computer vision task, for example Support Vector Machines which necessitated manually engineered filters. Further to this, in 1988, Lecun et al applied training via the backpropagation algorithm to the neocognitron in ''Gradient-Based Learning Applied to Document Recognition,'' introducing the CNN largely as it is used today (LeCun et al., 1998) (Khan et al., 2018, p. 43). Their architecture, named, LeNet-5, was used to recognise handwritten check numbers.

2.0.3 Structure of CNNs

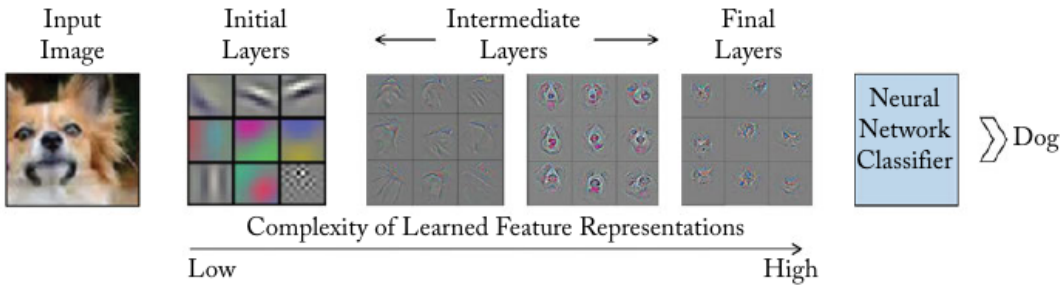


Figure 2.2: An Example of Feature Representations by Layers (credit:'A Guide to Convolutional Neural Networks for Computer Vision' by Salman Khan et al. (Khan et al., 2018, p. 44)

The eponymous layers in CNNs, convolutional layers, perform a mathematical operation called convolution on a set of learned filters — the kernel weights that are augmented in training, just as in other neural networks. (Khan et al., 2018, p. 46)

Figure 2.2 shows an example of feature representations from early, mid and later layers; later layers are combinations of the layers behind. These 'filters' are applied to the images to create 'feature maps.'

The convolutional layer aims to learn feature representations of the inputs. Each layer is comprised of several convolution kernels which are used to compute different feature maps. Specifically, each neuron of a feature map is connected to a region of neighbouring neurons in the previous layer. — 'Recent Advances in Convolutional Neural Networks' by J. Gu et al. (Gu et al., 2018)

Further, it is described that filters 'slide over' the image to create feature map.

Convolutional layers are followed by pooling layers, their purpose to reduce, or downsample, the resolution of feature maps, thus lowering computation cost. The most commonly used pooling operations are Max Pooling, where the maximum activations are used, and average pooling, where an average is used. Another, less widely used pooling method is Min Pooling, where smallest activations are used, for reasons such as the situation depicted in Figure 2.3. Pooling clearly has significant effects on models, and so should be considered in our methodology.

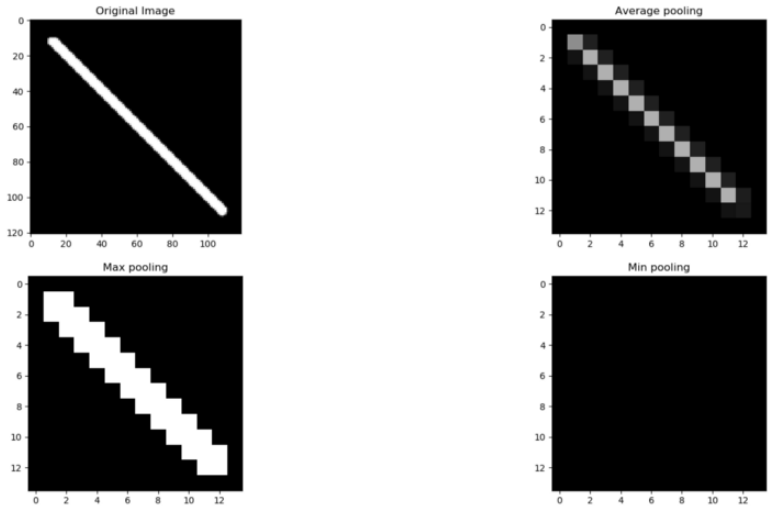


Figure 2.3: Min Pooling Example (credit: @bdhuma, https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9)

Following pairs of convolutional and pooling layers, there are typically one or more fully connected (dense) layers which perform high level reasoning on the feature maps. Finally, the dense layers connect to an output layer; for example, softmax for an image classification task (Khan et al., 2018, p. 56) (Gu et al., 2018).

As CNNs have come into popularity, new methods and architectures have been extensively studied.

2.0.4 Recent Developments in CNNs

In more recent years, developments such as, AlexNet, (Figure **??**) (Krizhevsky et al., 2012) winner of the ImageNet competition, have shone a spotlight on CNNs for their high performance(Gershgorn, 2018). Deep Convolutional Neural Networks are agreed by many as the dominant paradigm for machine learning with image data. (Scott et al., 2017) (Krizhevsky et al., 2012)

One alternative structure, the ResNet model (He et al., 2016), utilises a technique called 'Residual Learning,' intended to address the 'saturation' that very deep network suffer — where adding more layers can eventually lead to performance drops. Proposed by He et al (2015), they state that, ''when deeper networks are able to start converging, a *degredation* problem has been exposed: with the network depth increasing, accuracy gets saturated... then degrades rapidly.'' They posit that this is not caused by over-fitting due to the fact that excessive layering leads to worse training performance. Their new structure is able to stack many layers without suffering the problem of saturation due to the residual learning by use of shortcut connections. ResNet is available as a pre-trained model in keras, discussed in subsection 2.0.8. Another available model, and once ImageNet competition winner, is VGGNet, a 16-19 layer network that utilises stacked convolutional layers. (Simonyan and Zisserman, 2014)

Convolutional Neural Networks are growing deeper. By increasing depth, J. Gu et al say in 'Recent Developments in Convolutional Neural Networks' (Gu et al., 2018), the network can better approximate the target function with increased nonlinearity and get feature representations. As layers are added, the level of abstraction for features

learned increases. However, it also increases complexity, which makes the network more difficult to optimise and easier to overfit. This also means more computational power is required.

2.0.5 Activations, Initialisations

''A correct weight initialization is the key to stably train very deep networks.'' (Khan et al., 2018, p. 69).

In 'Activation Functions: Comparison of Trends in Practice and Research for Deep Learning,' C. Nwankpa et al state that, 'ReLU is the most widely used activation function for deep learning.' (Nwankpa et al., 2018) It is said to offer better performance and generalisation in deep learning as compared to sigmoid and tan, relating to it's near linearity, 'preserving properties of linaer models that [are] easy to optimise with gradient descent.' Considering backpropagation, derivatives are quick to compute. Additionally, they discover that ReLU and Softmax output layers are used in all top ten recent architectures tested on the ImageNet dataset.

Many alternatives to random and zero weight initialisations have been proposed. Initialisations such as the 'Glorot Normal' and the 'Xavier He' initialisations have been shown to outperform both zero and random initialisations (C. Nwankpa 2019.) (Nwankpa et al., 2018)

The 'Xavier Initialisation' initialises weights with a ''variance measure that is dependent on the number of ingoing and outgoing connections from a neuron,'' ((Khan et al., 2018, p. 70)), thus solving the problem that random initialisations cause for neurons: ''the variance of its output [is] directly proportional to the number of incoming connections.'' They go on to say that this initialisation leads to improved performance in practice.

2.0.6 Applications to Aerial Photographs

The question at this point is whether we wish to perform semantic segmentation, to isolate objects, accumulations, in an image, or to classify images unilaterally — one label for 24 million pixels. However, we must consider that our labels are not suited for

semantic segmentation, and so that would require likely hundreds of hours of precise labelling all images to end up with, most likely, insufficient data to train.

Referring back to Figure Figure 1.1, the image shown on their website to describe their classifications, it appears to indicate terrain classification would be easy. However these images not at all representative of the dataset; more on this in section 3.1.

Object recognition models can attain very high accuracy with Aerial photographs, (Radovic et al., 2017), with Radovic et al obtaining 97.5% targeting vehicles such as planes and buses, using the framework, 'YOLO' — you only look once. However, (Chung et al., 2018) found the YOLO algorithm to underperform alternative methods such as traditional CNN.

64% of identifiable Scottish beach litter has origins with the public, and 17% is classed as on the go litter, which contains, for example: drink cups or plastic bottles.(Society, 2018b) (Society, 2018a). The largest category of identifiable items are plastics. Around half overall is unidentifiable, mostly because its too tiny. Given that litter can accumulate in many shapes, it may be hard to use an object-based framework, but some objects, such as bottles, may be good predictors. Each instance or collection of littering may be of a unique shape, and it is likely that aspects of the images such as colour contrasts, for example blues and oranges in a background of natural colour spectra, e.g., browns and greens, will provide better predictive power; this based on Figure 1.1. However, it is still likely the Neural Network may learn some discriminatory object patterns.

CNNs have been successfully deployed for use of ground terrain classification with aerial photographs (Hu et al., 2018) (Sameen et al., 2018), and are especially prominent in the field of remote sensing (Hu et al., 2018).A 'spectral-spatial' model (Sameen et al., 2018) outperformed the traditional CNN by 4% in one case. This is of interest despite us not having access to hyper-spectral data, which is often used in remote sensing (Hu et al., 2018), but perhaps an appropriate camera/device could be applied on recommendation to improve future performance. It is more likely that colour contrasts will give some good results than learning a representational filter for common items. However, some recognisable objects, perhaps those in the TrashNet image repository

(Thung, 2017) or similar image dataset that could be used to pre-train a semantic segmentation model and serve as good discriminators of images that contain trash, and the quantity of those objects correspond to the level of accumulation.

2.0.7 Litter Detection with CNNs

The use of CNNs for litter detection has been proposed theoretically (Balchandani et al., 2017) as an overhaul to current litter prevention systems; and, in conjunction with UAVs, has been separately experimented with for real-time trash recognition at the street level (Chung et al., 2018). In the latter case, the dataset 'TrashNet' (Thung, 2017) was used for training. Given they use UAVs, they are closer to the trash, making object recognition easier than our case. Still, their results found the CNN underperformed the less evolved technique of Support Vector Machines, which they blamed on over-fitting. Early stoppage, regularisation and more can help with this (Géron, 2017, p. 272) (Khan et al., 2018, p. 79), but it is a significant barrier for applications with few training samples, some typical classes in TrashNet having around 500 samples; the magnitude of the present problem should become clear now, given they failed in their objective and with even more samples, and a drone to allow perfect repositioning.

Additionally, CNNs have been used successfully for underwater litter detection (Fulton et al., 2019), a case with relatively few samples. In most cases, there is agreement that, as put by G. Scott et al in *'Training Deep Convolutional Neural Networks for Land-Cover Classification of High-Resolution Imagery'*, 'acquiring a suitably large dataset for training DCNN is often a significant challenge.' (Hu et al., 2018)

2.0.8 Training and Transfer Learning

(Khan et al., 2018, p. 79) state that, In practical scenarios, it is desirable to train very deep networks, but we do not have a large amount of annotated data available for many problem settings. A very successful practice in such cases is to first train the neural network on a related but different problem, where a large amount of training data is already available. ImageNet is given as an example. For our purposes, if we included TrashNet in our pre-training, which includes images of common trash items such as

discarded bottles and cans, this may prove useful if the route of object identification is explored. For example, the NN may find that identifying one or several discarded bottles in the image is a good indicator of litter levels.

It has been proven that the weights of CNNs learned in the first layer are general (Azizpour et al., 2015), and therefore can be used for a separate purpose as that trained on. This is called Transfer Learning — the utilisation of pre-trained models for alternate purposes ((Khan et al., 2018, p. 72)). Khan et al state that, ''For this purpose, small learning rates are used ... so that the learning previously acquired on the previous dataset (e.g., ImageNet) is not lost.'' Further, Khan et al offer a second approach: to build your own architecture, then train on a large dataset that is available, then repurpose this model for the desired task with the initialisation of a successful model for that commonly available dataset.

Transfer learning can, when domains are similar, improve results — especially in cases of insufficient sample quantity. However, transfer learning is not allows appropriate. In the article 'A comprehensive hands on guide to transfer learning with real world applications in deep learning,' by (Sarkar, 2018), he states that:

'' There can be scenarios where transferring knowledge for the sake of it may make matters worse than improving anything (also known as negative transfer). We should aim at utilizing transfer learning to improve target task performance/results and not degrade them. We need to be careful about when to transfer and when not to.''

Therefore when applying transfer learning, the possibility of 'negative transfer' will be kept in mind.

Further to training optimisation, the consensus is that batch normalisation improves performance, with (**?**) stating it was integral to performance in their spectral-spatial terrain classification model, and (Khan et al., 2018, p. 76) stating it, provides robustness against bad weight initialisations [and] improves convergence rates. They go on to state that, it integrates the normalisation in the network by allowing back-propagation of errors through the normalisation layer, and therefore allows end-to-end training of deep networks. Similar ideas are expressed in (Géron, 2017, p. 282). It should therefore be used in the present case to optimise the training phase.

Unsupervised pre-training has been proven to improve deep learning models Erhan et al. (2010). However it is perhaps less often used in recent times due to more effective optimisation measures such as Transfer Learning, custom initialisations and Adam (along with its varieties) optimisers.

2.0.9 Multiple Input Models

Multiple image input deep convolutional networks have been applied in some domains, but mixed numeric/categorical and image input models less so. One particular example is 'House Price Estimation from Visual and Textual Features' Ahmed and Moustafa (2016).

They combine textual and numeric features such as neighbourhood area, number of rooms, with visual ones: images of the home and its interior. The combined features are fed to a fully connected mutli-layer Neural Network that estimates the house price as its single output, therefore being a regression task. 'To train and evaluate our network,' they say, 'we have collected the first houses dataset (to our knowledge) that combines both images and textual attributes.'

Ultimately, their model outperforms the best existing model already published for that dataset — 'Through experiments, it was shown that aggregating both visual and textual information yielded better estimation accuracy compared to textual features alone.'

Additionally, many articles such as 'Keras: Multiple Inputs and Mixed Data,' (Rosebrock, 2019), discuss the topic.

2.0.10 Data Augmentation and Insufficient Training Samples

''The problem with small datasets is that models trained with them do not generalise well from the validation and test set. Hence, these models suffer from the problem of over-fitting.'' (Perez and Wang, 2017)

In ''The Effectiveness of Data Augmentation in Image Classification using Deep Learning'' (Perez and Wang, 2017), two methods are proposed for augmentation: generating augmented data prior to training, meaning expanding the dataset, and alter-

nately to augment and classify simultaneously — through a method called 'Neural Augmentation,' where training samples were combined to form new representations. The Adam optimiser is used. They find that traditional augmentation outperforms in some cases, while their proposed approach outperforms by a small margin in others at the cost of much longer training times.

Data Augmentation has been extensively applied in deep learning image recognition tasks (Khan et al., 2018, p. 74), where the necessity of large datasets is a common issue. Salman Khan et al state that, '[Data Augmentation] is often a very easy way of enhancing the generalization power of CNN models. Especially for cases where the number of training examples is relatively low.' Khan goes on to state that datasets can be increased in size by orders of magnitude, generating new samples for each based on a set of operations.

Some simple types of data augmentation as applied to images are: cropping, rotating, flipping, horizontally and vertically, and adding random noise for example, single pixel changes (known as 'salt and pepper'), or shifting RGB colours slightly. These simple approaches are considered to be very successful in some cases; however, there are higher level methods of data augmentation, such as the creation of synthetic samples, for example by the use of a Generative Adversarial Network (Perez and Wang, 2017).

Real world deep learning applications are often faced with the problem of insufficiently numerous samples, with specialist domains in particular, such as the medical imaging domain, where the number of examples of a rare tumour types are small (Perez and Wang, 2017). A recent approach to the problem is the use of Generative Adversarial (Neural) Networks, which produces outputs similar to the data it has trained on, meaning they can be used to generate new training data.

In 'Conditional Infilling GANs for Data Augmentation in Mammogram Classification,' by (Wu et al., 2018), Generative Adversarial Networks (GANs) are applied to the domain of detecting lesions in Mammograms to screen for cancer where limited samples are available. The GAN is used to automatically generate lesions on an image that is without lesion, a negative sample, to create an image that is a realistic repre-

sentation of how a lesion would present in that negative sample. In our case, we could find the area of the image with trash, then superimpose that to a non trash image to create a realistic depiction of accumulations in that image. However these algorithms are extremely computationally costly, and cannot be applied adequately to such high resolution images with the hardware available.

Wu et al note that every 1000 iterations, ‘’we increase the relative proportion of real data used by 20%, such that the final iteration is trained on 90% real data. We observe that this regime helps prevent early over fitting and greater generalisation for later epochs.‘‘

Relating back to this project's purpose, this is an interesting idea to explore given our extreme class imbalance — varying the ratio of real to synthetic images so that by the last epochs only real images are seen and fine tuned on with lower learning rate, though our images will only be transformed through simple operations like mirroring and brightening/dulling due to the computation requirements for GANs on high resolution data. Augmenting every image, deigned 'live' augmentation in articles, may be too extreme. This dynamic ratio could improve generalisation not only to the test set, but outside of it, in the sample space in which future images reside, as representations of the sample space have been learned, then fine tuned to the real images. Also of note — to achieve better performance, they initiated with Image Net weights, which entailed training their network on the ImageNet dataset first.

The issue with applying transfer learning from a pre-trained (on the ImageNet repository) model to the present task is that our images are of much higher resolution and distinctly separate domain; considering the size features will be, and the size of filters/feature maps as applied to 224×224 images, this kind of transfer learning may be what's called a 'negative transfer.' (Sarkar, 2018) However, Sarkar also suggests this may aid in finding small features in large images (our case.)

In the article 'A comprehensive hands on guide to transfer learning with real world applications in deep learning,' (Sarkar, 2018) , he writes:

> ‘’There can be scenarios where transferring knowledge for the sake of it may make matters worse than improving anything (also known as negative transfer).‘‘

So it is possible for a weight transfer to harm performance, particularly when the domains are very different.

2.0.11 Sample Inequality

In 'Towards Effective Classification of Imbalanced Data with Convolutional Neural Networks,' (Raj et al., 2016), they discuss the tendency for real datasets, such as ours, to have overly numerous classes dominate the dataset, thereby having a 'strong bias towards the majority class.' Additionally, they discuss the weighting of mis-classifications, such that 'in medical applications, the cost of erroneously classifying a sick person as healthy can have a larger risk than wrongly classifying a healthy person as sick.' This relates to our problem also, as, up to a point, we would rather return the most positive classes we can at the cost of returning more images in total, and similarly would ideally weight a mis-classification of a class 0 for a class 5, say, litter accumulation, as having a higher cost than of a 0 for a 1.

As a solution to the class imbalance problem, two approaches are discussed: over or under-sampling the training data to a point of equality, or using a different algorithm. Ultimately they combine several approaches towards the end of 'cost-sensitive learning.'

While not exactly their approach, in the same general vein of ideas as cost, we could weight our cost function so that it values class 4 samples as $10\times$ that of a class 0, meaning the network will value better a filter which aids classification of 1 class 4 than 9 class 0s.

2.0.12 Multi-input Deep Networks

Multi-Input Models combining multiple sets of image data have been applied in a variety of image recognition tasks, from flower identification but combining numeric, categorical and image data is less common. (Ahmed and Moustafa, 2016) published one such approach.

(Ahmed and Moustafa, 2016) propose a mixed input network combining images and numerical data for house price estimation — that is, regression. Their model outperformed the numeric only network, but they did not compare to an image only

network. On the topic of rarity, they state:

''This paper announces the first dataset, to our knowledge, that combines both visual and textual features for house price estimation ... Through experiments, it was shown that aggregating both visual and textual information yielded better estimation accuracy compared to textual features alone.'

Therefore, we may see performance boosts with the inclusion of this data in our model.

In the article 'Keras: Multiple Inputs and Mixed Data,' (Rosebrock, 2019), another mixed numeric and image input network is applied to a model is applied to the same dataset discussed above.

Rosebrock states that:

''Developing machine learning systems capable of handling mixed data can be extremely challenging as each data type may require separate preprocessing steps, including scaling, normalization, and feature engineering.''

The preparation of numeric data in our case could include, for example, normalisation of the latitude and longitude columns, as-well as the consideration of our somewhat large Sector categorical variable; might it be better considered a continuous variable, the parabolic or circular nature of its relation to latitude and longitude providing better information to the network?

On the state of mixed input models in research, Rosebrock offers, ''working with mixed data is still very much an open area of research and is often heavily dependent on the specific task/end goal.''

Considering our end goal, the inclusion of location defining data seems appropriate given such a wide sample space from which each image is drawn exists. Any way to aid the network in this area should be explored.

Chapter 3

Methodology

The main topic of research is as follows: can an automated deep learning system be used to automatically classify aerial imagery of the Scottish coastline into levels of litter accumulation?

Towards this end, the following section lays out a series of research questions, each representing a facet of the many barriers, explorations and decisions involved in the development and demonstration of such a system, in particular towards the current validity with the limited data and resources available, and recommendations for future work based on methodologies proposed, as-well as the proposition of new practices at Scrapbook to facilitate the implementation of such a system in the future.

3.0.1 Research Questions

- In deep learning, the data is the most important factor in any system. How can we pre-process the data, including cleaning, compression and downsizing, to best prepare for model building while maintaining quantity and quality? Additionally, given there is some ambiguity in associating some images with labels, how can we accurately pair every image with a label where available, balancing certainty that no conflicting information is given to the network with the need for as much data as possible?

- With what architecture and configuration can the network be best built? This includes exploration of input shape, given the highest resolution inputs mean a

trade-off with depth and complexity of architecture. This necessitates the passing of data through generators: how will these be created, and later amended to suit different models and methods?

- Class imbalance is significant in the data. Firstly, does this harm performance? If so, of the many ways to address this, which best mitigates this issue? Consequently, positive samples are significantly low in number. Can methods such as alternate sampling approaches, data augmentation, initialisations and architecture improve the over-fitting associated with this problem? And which approach for each of these methods best improves generalisation?

- Would a binary system perform better than multi-class, and if so, is it appropriate to the purpose of the overall project? If such a binary system is trained, at what threshold should a class be considered positive to maintain the balance such that, given the problem domain, more false positives than false negatives may be desired, while maintaining a balance with the number of images returned?

- Significant variation in terrain among the images is present. Is there a way to address this through architecture or other means? If so, does this improve performance, in general but also in some cases where the variation may be causing issue, i.e., particular samples or regions with distinct representations?

- Given the model is intended for use to process images in bulk, if features other than images were to be passed to the model for training, only automatically collectable features could be collected. What, if any, meets this criteria, and does the inclusion of these features improve performance? If so, how these can be collected?

- Is the final model of strong enough performance that it could replace the current system at Scrapbook and be deployed? How feasible is the deployment of this model?

- Regardless of how effective the model is discovered to be, given at this stage we have limited data and more will be available in the future, what basic structure

should be deployed to generate results in such a fashion that it can be used for analysis by any interested party?

- Given analysis of the above methodologies, what recommendations from this analysis can be made so that the current practices at Scrapbook can align with the requirements of a deep learning system, and so that the best deep learning system could be deployed?

3.1 Data Processing

Considering the question of dataset generation and image-label pairing, the data is now discussed.

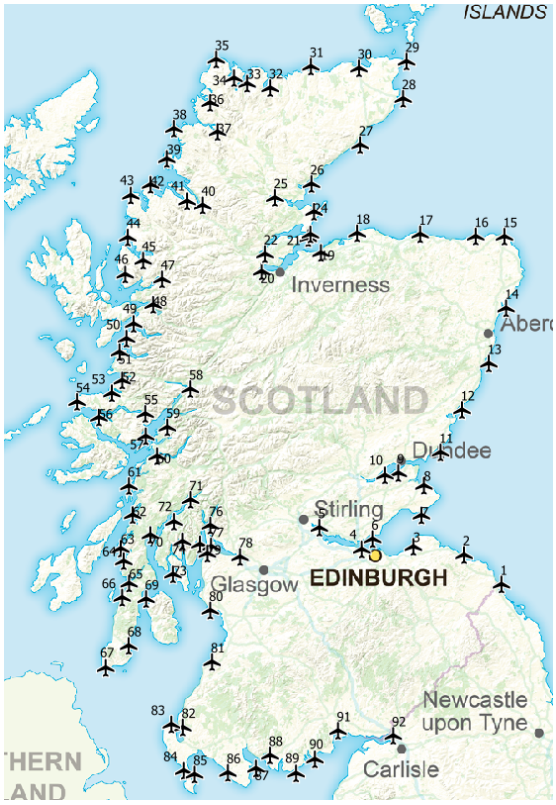


Figure 3.1: Scotland's Coastline Divided into Sectors

Scrapbook have supplied $13,880$ images segmented by sector and date of collection, each sector typically having an excel spreadsheet which contain information on the majority of these images. These are also segmented in some cases by 'parts', where there are mainland and island images, and in other cases some nearby groups of sectors are amalgamated in one spreadsheet. All images are from 2018 and from various dates, but typically one pass of a sector, segmented as shown in Figure 3.1.Due to the quantity of these images and spreadsheets, as-well as the inconsistencies to be discussed shortly, automatic processing is necessary. The methodology is then to amalgamate these spreadsheets to a master csv, and assign, accurately, a file path to every row where labels exist. The implementation of this is proposed in the following subsection 3.1.1.

3.1.1 Spreadsheets

Processing these spreadsheets entails the creation of several functions. To make best use of the columns available, while considering that to pass a feature to the network, given the methodology of modelling is supervised learning, it must be automatically collectable information; otherwise it is unusable for the purpose of an automatic system; a system which creates a data structure based on image EXIF and, perhaps as a later recommendation, information available online such as forecasted weather and tidal data, correlated with dates of collection, once more data has been collected. As the Scrapbook image analysis system stands, these sheets provide the tools for coordination of clean ups. We repurpose these to train a new system.

In discourse with Sophie from SCRAPbook, she stated the sheets are 'rudimentary,' having been filled by various volunteers at different times who 'zoom around' the images searching for accumulations, then filling in various columns. Consequently, while some basic (and varying) protocol has been followed, there are significant inconsistencies in features recorded, conventions used, as-well as overlap and redundancy between sheets, and some mistakes/missing values. However, on the positive, there are labels — notation: '$Litter\_Intensity$' $[0, 5]$ — for the majority of entries in the spreadsheets.

```
Index(['Survey', 'Sector', 'Image_Path', 'Image_Name', 'Lat', 'Long',
    'Image_Quality',
       'Confirm_ Litter_P/A', 'Litter_Intensity', 'Litter_Type',
       'Litter_Location', 'Coastal Character', 'Image_Quality', 'Other
           features', 'Unnamed: 12',
       'Unnamed: 13'],
      dtype='object')
```

Above we see the columns typically present in a volunteer filled sheet. See below for a brief description of each column alongside some notes.

- 'Survey' — The Date of Image Capture
- 'Sector' — A categorical for designated areas
- 'Image_Name' — The filename of the image in question; this became a significant issue later on as they are not unique
- 'Litter_Intensity' — the label, this varies from 0 to 5, a categorical measure for intensity of accumulations; element of subjectivity in many cases; refer to Figure **??**
- 'Litter_Type' — the nature of litter present; typically includes entries like, 'fish boxes,' 'plastic bags,' or, more rarely, 'literally everything.'
- 'Litter_Location' — a less subjective column, this features values from the range: ['foreshore', 'backshore'] and so forth
- 'Coastal_Character' — a description of the terrain, such as rocky, sandy or industrial
- 'Image_Quality' — a descriptor of the quality of the images, using various conventions, e.g., 'Good', 'Medium', 'N', 'Not Usable,' 'Mid.'

In considering feature selection of this text dataset, it is important to recall the purpose of the model; to automatically and efficiently analyse many images at once,

saving valuable volunteer time. It is therefore not appropriate to input, for example, a Coastal Character value for each image. We can consider the inputting of the value: sector; which is present in some spreadsheets but not most, and we have automatically stripped from the filenames of the excel spreadsheets (note there are discrepancies and missing sectors.) What is more desirable however is automatically detecting the sector based on the longitude and latitude. Nearby sectors typically have similar terrains and litter objects, so a mislabel of a sector or two over is not a serious issue.

From Figure 3.1, we can see the sectors adjoin each-other and run counter-clockwise around the coastlines. It's important to realise the longitude/latitude information refers to the GPS EXIF data of the photograph, and hence the location of the aeroplane, not the area itself. We then determine sector of future samples through nearness to the GPS coordinates.

Additionally, the spreadsheets must be concatenated and pre-processed via implementation through simple Python scripts. Application of this can be found in analysis, with much of the code available in the appendix due to size, the significant parts included in the main body.

3.1.2 Images

While the majority of images are of resolution 6000 by 4000, some, particularly from Sector 1, are only 12.1 Mega-pixels (as opposed to 24) and have resolution 4256 by 2832. The question is then, prior to feeding images to the networks, whether to upscale these to max resolution by use of an upscaling algorithm, or pad them with zeroes, or lastly to downscale the rest to this lower resolution. No upscaling algorithm has been shown to perform better in deep learning than downscaling all images to the lowest image size in the relevant dataset, so we rule out option 1; but zero padding is still an option. Given lower input shape means easier training, and training at high resolution takes a long, long time, we go for option 3 and downscale the images.

It's observed that to maintain sufficient model complexity and a batch size that promotes variety of class before weight augmentations, float16 data-type is a necessity, given the increases in memory usage with higher types. Given PIL reduces images to

8 bit colour depth anyhow, information loss is neglible.

Another noteworthy property of the data is the apparent extreme rarity of high accumulation samples, which we discuss in greater detail in subsection 3.1.1. Mitigating the effects of this may necessitate sampling techniques, or loss weightings.

3.1.3 Terrain Variety

Figure 3.2: Examples of Terrain Variation

As can be seen in Figure 3.2, there is a broad variety of terrain within the dataset. Among 'natural' backdrops, there is variety between grassy, rock, sand, cliff and dirt/-mud terrains, while the less natural areas are either industrial or pedestrian, containing the colours and geometric shapes which might serve as strong distinguishers, learned filters, of litter presence in natural images; features likely to trigger false positives in said areas. Additionally, some 'beachy' areas that are inhabited may trigger false positives as items that are not necessarily litter but could be (e.g., a picnic, people in bright

clothing, bicycles) will be present. Given most images do not contain people, and most are undeveloped, rural landscapes, the model may learn to identify litter more easily in rural, monotonous images, with low density and development.

Each image in the figure also has some variety in distance, elevation and angle. Ideally, our model would grow invariant to these factors, as-well as brightness, shadows — bearing in mind the sun (dependent on time of day also of-course) shines from slightly different angles as we travel around the coast from West to East. It's likely that data augmentation will help towards this end, exposing the model to synthetic images which approximate variation in some of these factors — for example, rotations for angle of aircraft, brightness for lighting changes, zooms for distance.

We should also note that among the sectors there is a great variance in the number of samples available, with some folders having as few as 40 images; however, ideally the network will learn that Sectors near each-other have (typically) similar terrain — instances where this does not hold are, for example, where country landscapes meet towns and tourist destinations, which can present as inter-Sector variance.

As for how to treat this issue, the multi input model is proposed (see subsection A.3.2.

3.1.4 What is Litter?

Understanding the items SCRAPbook are interested in is important moving forward. As Sophie stated in the interview, not all items constitute 'litter' in their eyes. For example, while manufactured, industrial wood such as in Figure 3.3, referred to as 'timber' in spreadsheets, do count as litter, natural wood items like driftwood do not, despite sometimes accumulating in masses at some sites.

Sophie, correspondent at Scrapbook, stated:

"They're natural items, so it doesn't take away from the view, and it's not polluting anything."

Timber presents very often in sectors 1 and 2; Sophie provided some clarity:

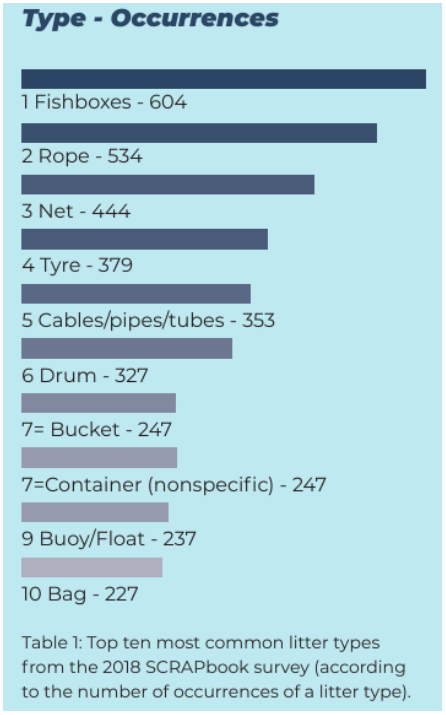
Figure 3.3: Timber at Section 1

‘’We think a shipping container may have been lost at sea near [sector 2], releasing a whole load of timber; it floats, so it’s just drifting up to shore’‘

Timber may prove troublesome for the network to represent, the colour spectrum similar to the backgrounds it typically presents at. The geometric pattern, a synthetically straight edged rectangle shape, may be adequate, but shapes like these — signposts, fences — are commonplace on many areas, particularly developed ones. But the filter would only apply to a relatively small variety of sectors, those on the south east coast; considering this situation, aiding the network in understand the location of samples via coordinates or a categorical sector feature could be beneficial. It is expected that incidence of litter types will correlate strongly with performance, in addition to samples per class.

Despite the strong presence of timber in sectors 1 and 2, timber is not one of the top ten items found overall:

While the items depicted in Figure 3.4 represent the majority of litter manifestations, there are also unique situations like strewn boat debris, classified as high accumulations, therefore an instance of interest in evaluating CNN performance, so we can later assess performance by recognition of litter types. Instances like these warrant discussion when considering applying, for example, a loss function which suggests to



(a) Credit: Scrapbook 2018 Report

(b) A Fishbox

(c) Timber

Figure 3.4: Common Items

the network that nearby classes have more in common each-other than those further away. Ideally, the network would detect the presence of a feature laying on natural terrain; examination of the final model's filters and feature maps may indicate, though

Figure 3.5: A Fish Box in Section 2

the expectation is for many indiscernible features more to do with colour bands, and some edges, than a feature map clearly defining such rare samples as boat wrecks.

Clean up crews would be dispatched to sites such as Figure 3.3 given the accumulation level — a class 2 + is severe, warranting action, Sophie has said. So a widespread variety of items is not necessary for an area, or more specifically an image, to be classed as a 2+ or even 5 (there is a class 5 instance which contains no plastics of any kind, instead a widespread shipwreck; which is an industrial item.) Learning could be difficult for such samples as a desirable scenario would be for the network to 'count' the likely presence of filters representing commonly occurring items — our labels can be thought of not as distinct, unique domains, but overlapping and sequential levelling of a more general space of filters; with also some features such as the mentioned shipwreck, perhaps representable as collections of timber, manufactured wood, or just by the presence of individual parts. Regardless, the rarity of occurrence for such items may consequently mean poor performance in those classifications.

Fish boxes (Figure 3.5) were the most common item found, often washing up from shipping vessels when dumped or lost at sea. Due to their bright blue colour and square edge shapes, and the significant quantity there are in the data, these should be easy to identify — that is, for our CNN to learn a representing filter.

Plastics, defined to be so if 'plastic component observed,' accounted for 59% of litter collected by Scrapbook in 2018 Society (2018b); which is the year of capture for our images. Another consideration is that of micro-plastics, hard to identify from such elevation and distance, though instances exist where small but visibly identifiable plas-

tics sporadically covered segments of terrain. In these cases, the colour and consistent shape should learn to adequate learning, if there are enough samples.

3.1.5 Unusable Images

Figure 3.6 is an example of an inappropriate image, the reason being the lack of coastline. The features, distance, terrain and general area of the photo do not share similarities with the rest of the data.

Figure 3.6: An Unusable Image

Among the images, there is a significant presence of such non coastal captures; perhaps taken by accident, or the photographer — all of these seemed to be manually captured, based on the images captured before and after (aiming) — looking for landscapes. The quantity of such images is so large that it raises an issue worth investing effort to solve; to class these as 0 could give conflicting information to the network,

which could struggle to find commonalities among the features of these often suburban towns and beaches, cliffs, all bordered by water on the side of the aircraft. It's unlikely that the network would learn anything valuable about the structure of the data as a whole from these, and so we desire to remove or except them from training. The approach proposed is discussed in subsection 3.1.1.

3.2 Addressing Sampling Issues

With the primary issues being the lack of samples for rare classes and the associated imbalanced distribution, the following methods are proposed as potential solutions.

Each method has drawbacks as-well as its intended outcome, to be discussed now. Solutions proposed here are then implemented and analysed in **??**.

3.2.1 Over and Under Sampling

Over and under sampling entails artificially re-sampling and removing samples so to artificially balance the data-set by distribution of class. Over sampling each class to the greatest number of samples, sector 0, means appending, with repeated values, each class until there are 6000 samples, then $30,000$ samples in total to be iterated through each epoch.

A number of drawbacks to this approach include a danger of skewing distribution of predictions, and the fact that extending our rare samples in quantity does not expand the sample space from which they are drawn, meaning it is not a full representation of the domain of the classes 1 and up. It is possible that this approach simply approximates iterating through the data in its naturally unbalanced form for many epochs, then unnecessarily processing samples repeatedly from which the same representations can be drawn.

Implementing this is simple, via the following function, a part of the created utilities module:

```
@staticmethod
def equalise_samples(df):
```

```
        """
        Over-Samples the rare classes until they are all the same n as the
            most numerous class
        """
        max_n = df.Litter_Intensity.value_counts()[0]

        return DFUtils.over_sample_df(df, max_n)
```

Where the over_sample function is as follows:

```
 @staticmethod
def over_sample_df(df, n=700):
    """
    over-samples the more numerous classes to n
    """

    df_1s, df_2s, df_3s, df_4s = df[df.Litter_Intensity == 1],
        df[df.Litter_Intensity == 2], \
                                df[df.Litter_Intensity == 3],
                                    df[df.Litter_Intensity == 4]

    df = df[df.Litter_Intensity < 1]


    for df_n in [df_1s, df_2s, df_3s, df_4s]:

        while df_n.shape[0] < n:
            df_n = df_n.append(df_n.sample(n=1))

    df = df.append([df_1s, df_2s, df_3s, df_4s])

    return df.sample(frac=1).reset_index(drop=True)
```

The addition of one sample per iteration, drawn randomly, includes re-placement.

This data-frame can then be passed to generators in the same way as the natural data-frame.

Similarly to under-sample, the following accomplishes this:

```
def under_sample_df(df, n=600):
    """
    under-samples the more numerous classes
    """

    for label in set(df.Litter_Intensity.values):
        df_m = df[df.Litter_Intensity == label]

        df = df[df.Litter_Intensity != label]

        df = df.append(df_m.sample(n=n))

    return df.sample(frac=1).reset_index(drop=True)
```

When implementing these methods, analysis of comparison to using the original data-frame should be made, thereby answering the questions of: does this improve by class performance, and performance in general.

It is possible that a system which passes through the data as is, with no weighting of loss and class, could achieve the same results. A ceiling of performance is hit while the network initially finds gains in fitting class 0 well, but continuing training past this point means the network can only improve by fitting the rest of the samples also.

Additionally, to answer the research problem pertaining to the nature of the classes themselves; their meaning is distinct levels of the same concept, and so were the network to understand a class 0 should be mistaken for a class 1 more often than a class 5, it may lead learning away from inappropriate, over-fitting filters such as representations of the terrain at which most or even all positive samples have in the training data. This raises a question of the number of filters used, as if we allow many, over-fitting by full or partial representation of samples can occur, rather than the ideally generalisable

features in these images. More generalisable filters may be learned with less filters and the penalising loss proposed.

To implement this, the following code is proposed for use as a loss function:

```
def custom_loss(y_true, y_pred, beta=0.03):
    """
    This loss adds a penalty to predictions that are significantly far away
        from the true value.
    """

    y_true_val = K.argmax(y_true) # get true class index
    y_pred_val = K.argmax(y_pred) # pred class index

    d = K.abs(y_true_val - y_pred_val) / 4 # 0 to 1

    mod = K.exp(beta * d) # is at-least one, at most

    return mod * K.categorical_crossentropy(y_pred, y_true)
```

This adds an additional cost of 3% to categorical cross-entropy with $\beta = 0.003$, which can be varied for analysis. The idea behind the small number is to not excessively damage the well researched and effective categorical cross-entropy function, but to lead away from representations of features that do not generalise.

3.2.2 Data Augmentation

With Data Augmentation, improvements are sought in the issue of sample and class imbalance, meaning both the rarity of samples of interest and the skewed distribution of classes; and to aid in reduction of over-fitting through both the introduction of these samples which are augmented within the sample space from which, theoretically, future images will come, and with live augmentation to prevent exact representations of the training data being learned by the introduction of uniquely augmented images for each sample, the network then seeing a better representation of the domain space and therefore giving more generalisable solutions.

To apply data augmentation, the following generator is created.

```
class ImageGenerators:

    aug_datagen = ImageDataGenerator(rotation_range=30,
                                     brightness_range=(-0.3, 0.3),
                                     width_shift_range=0.1,
                                     channel_shift_range=0.1,
                                     height_shift_range=0.1,
                                     shear_range=0.2,
                                     zoom_range=0.2,
                                     fill_mode='nearest',
                                     horizontal_flip=True,
                                     vertical_flip=True,
                                     rescale=1 / 255.0,
                                     data_format='channels_last',
                                         dtype=np.float16)
```

The operations proposed are: rotate in a range of 30°, which is intended to simulate the random variation in angle of camera/aircraft to the coastline, and zoom to simulate variations in distance and elevation. Shear transformations are also intended to improve robustness to angle, distance variations, altering perspective similar to how new images could appear.

Note that an advantage of the Image Data Generator approach is the option to fill blank areas caused by transformations such as rotations with the nearest pixels, which exist as a consequence of necessitating the same input shape as the input images for all images. Considering the alternative, filling with a single colour such as 'white' or 'black', which are the options using PIL to augment images, using the nearest pixel interpolation could prevent harmful features such as black or white corners from being picked up by filter maps — the pooling type used is a factor here, considering for example max pooling, white is 255 in RGB code, and therefore could overpower other features of these synthetic images; and similarly 0 is black, so inappropriate for min pooling — see Figure **??**.

Width and height shifts too are designed to simulate the variation among the data, in which there are typically several consecutive photographs of the same litter as they pass a location. Synthetically generating new ones from training images could give similar images to future, unseen test images; particularly given a series of images represent one instance due to the data capture process. This is a potential flaw, not wholely of data augmentation but splitting the data as a whole. However, real future images, not those of the test set, are likely to be very similar to those existing already; but with more data, particularly more class 1+ samples, a more complete sample space can be drawn from.

Horizontal flips are intended to aid in the recognition of litter as we can develop better representations of the litter when seen from different perspectives. Vertical flips however may not be appropriate given the skeletal structure of typical samples — sea, beach, land; shore; foreshore; back-shore, etc — is then inverted, placing the sea in the sky. This could harm learning, but also could improve by developing better representations of features; experimentation will indicate.

Below applies the image generator with the 'flow from data-frame' approach, where the data-frame holds image paths and labels.

```
ImageGenerators.aug_datagen.flow_from_dataframe(dataframe=train_df,
directory='/home/strychl2/DATA/scrap-1200x800-24/', class_mode='categorical',
    color_mode='rgb', target_size=(800, 1200), x_col='rel_fp',
    y_col='Litter_Intensity', batch_size=20, interpolation='nearest',
    shuffle=True)
```

Augmentation should only be applied to the training set for obvious reasons. Given our dataframe contains image paths, it is more appropriate than the alternative flow methods such as flow from directory, requiring images to the isolated by class in separate subdirectories.

We can use the same image generator for validation and testing, applying no transformations other than rescale. Alternatively we can use one of the many others created, as long as the same methodology is used — e.g., not under-sampling as we can't redefine the training data-frame on epoch end with the image generator; which may be

inappropriate for this approach.

Below is an image generator that applies only rescaling, to normalise.

```
no_aug_gen = ImageDataGenerator(rescale=1 / 255.0,
    data_format='channels_last', dtype=np.float16)
```

The subsequent generator, to 'flow' the data sequentially for training, is similar to the one identified above, just passing either a validation or test data-frame with all other parameters the same.

For the additive approach, we can generate images to a specified directory as below:

```
images = df.sample(n=10)

gen = ImageGenerators.aug_datagen.flow_from_dataframe(dataframe=images,
                                                      directory='/home/ulrich/PycharmProjects/scr
                                                  save_to_dir='/home/ulrich/Documents/Project
                                                      Stuff/augmented_images/',
                                                      class_mode='categorical',
                                                          color_mode='rgb',
                                                      target_size=(800, 1200),
                                                          save_prefix='aug',
                                                      x_col='rel_fp',
                                                          y_col='Litter_Intensity',
                                                          batch_size=10,
                                                      interpolation='nearest',
                                                          shuffle=True,
                                                          validate_filenames=False)
gen.next()
```

At the top, for demonstration, we just take a random sample. In application, we draw only from the training data-frame, and whether we draw randomly or from specified classes is a question that analysis can answer. Representations of class 0 could still be improved, and so generating some new class 0 samples may aid performance; but comparatively class 1/2 + are the prime interests so it's appropriate to generate pro-

portionately more of these, e.g., for class m , generate $n_{synthetic_m} = n_{samples_m}$ for each selected augmentation, for each desired class. The result is inflated samples to a degree of $3-6$, selecting appropriate transformations such as rotations, mirrors, sheering, brightness, zooms, height/width shifts. This is a significant increase, not equivalent to the same sample gains with non-synthetic data; but perhaps an improvement.

On applying this, a number of black images are discovered. Alternate normalisations affect the frequency, some standardise options formatting all pixels to 0; but regardless of method employed, some images returned black. Manually applying transformations is a possible solution.

A question then is, when applying additive augmentation, should the generated images be sampled randomly, shuffled throughout the data, or, as in some academic applications, start at high proportions and decrease as we near completion of training, aiding generalisation?

As for live augmentation, the question is what configurations, meaning operations and vale ranges, will provide best results; with a static augmented set, this is final, and so should be executed with the best discovered results from experimentation with live augmentation.

3.3 Models

Convolutional Neural Networks, which are typically applied with only one image input and one output layer, can be very powerful, learning high level abstractions of input data. It is highly possible that the image only input version of the CNN could attain the same performance as the multi-input, using the high level reasoning from later dense layers to overcome the issue of terrain variety; perhaps checking for the presence of all filters associated with litter, not requiring a feature to indicate what to expect/what filters may be more appropriate. Therefore more neurons in the dense layers, and similarly more dense layers, in addition to more filters in the convolutional layers, could approximate the proposed multi-input solution to the terrain variation present in the data. However, perhaps a less complex multi-input model could also achieve this; which is especially valuable considering how low batch size must be for some proposed

deep architectures on higher resolution data.

3.3.1 Generators

While several generators are defined for different purposes, the following is an example of a generator for the image only model which does not utilise over and under sampling; incorporating these simply means that, in addition to resetting n and breaking the loop once the data-frame is iterated through, a new random subset of class 0s and 1s are taken from the master train data-frame, and one is initialised at start. For over-sampling all to max value, the same generator can be used. Examples can be found in Appendix A.

Note that this generator cannot be used for keras live image augmentation, requiring an image generator discussed in 3.2.2.

```
def data_generator(df, batch_size, mode='train', weights=None):

    n = 0 # init n at 0

    while True: # eternal loop

        images, labels = [], [] # a list for images and labels

        while len(images) < batch_size: # loops until batch created

            if n == df.shape[0]:
                n = 0 # reset n
                df = df.sample(frac=1).reset_index(drop=True)
                break # last batch likely to not have batch size samples

            img = prep_image(df.loc[n, 'image_path'])
            images.append(img)
            labels.append(df.loc[n, 'Litter_Intensity']) # appends nth label
            n += 1 # index goes up each time we add to our batch lists
```

```
        yield (np.array(images, dtype=np.float16,
        tf.keras.utils.to_categorical(labels, num_classes=classes))
```

The generator runs through the data-frame iteratively until the last sample is reached, then resetting the index and shuffling the data-frame, appending batches of images and labels, pre-processed via 'prep_image,' which loads via PIL, converts to array and normalises, by methods such as division of max pixel, centre at mean and min-max, which were experimented with. Labels are one hot encoded to allow soft-max; for a binary system, one hot can still be used for example with binary cross-entropy loss, resulting in predictions such as: $[0.25, 0.75]$; which can equivalently be converted to just one value. Alternate losses like Area under the Receiver Operating Characteristic can be used in binary classification also, with one output, which can then be used as the threshold where we decide which class a sample belongs to — this can be altered, for example were it pushed to 0.3, samples with more than 0.3 estimated probability would be classified as belonging to class 1. This has a trade-off with samples returned, in the present case particularly extreme given the support (number of samples) of class 0.

Non-sequential generators are not thread-safe, such as the Image Data Generator shown in subsection 3.2.2, and so multi-processing must be disabled for use with this generator.

Both sequential and Image Data Generators (which allow augmentation through Keras library) were also written, but are mostly trivial as a base class and function is provided by keras, so they are available in the appendix.

Tweaking a typical generator for multiple inputs is quite simple; for example, referring back to the generator above, as-well as the image and label lists which are filled each batch, a third container is inserted, into which the desired numerical features are appended.

```
        labels = []
        batch_df = pd.DataFrame(columns=['Sector', 'Lat', 'Long']) # the 3
            numeric features
```

```
    imgs = []
```

This uses three features, but can be extended or reduced to any number of features. The appending is as below.

```
        batch_df = batch_df.append(df[['Sector', 'Lat', 'Long']].iloc[n])
            # append 3 numeric features
```

Yielding the batch then simply includes the numeric data-frame as converted to an array, with the two inputs passed as a list so the model can separate them from the labels, and sample weights used to weight costs — particularly useful when using class weighting as keras functionality currently only applies class weighting to training, so validation information is unweighted which makes comparison, especially early stopping, spurious.

```
        sample_weights = np.array([weights[litter_value] for litter_value
            in labels]) # weight the batch

        yield ([np.array(imgs, dtype=np.float16), np.array(batch_df,
            dtype=np.float16)],
              np.array(tf.keras.utils.to_categorical(labels,
                  num_classes=classes), dtype=np.uint8),
              sample_weights)
```

3.3.2 Image Only Networks

The following code is capable of creating a typical CNN architecture with the keras 'functional' API according to parameters passed in the function call.

As arguments, the model function takes: the pooling layer desired; whether batch normalisation should be applied; the number of filters for each layer as a tuple, which gives us the number of convolutional/pooling layer pairs; as-well as the number of dense neurons (and hence layers); and dropout.

```
def base_cnn(pooling=AveragePooling2D, batch_norm=True, filters=(32, 64, 128,
```

```
    160, 196, 256, 352, 712),
              dense_neurons=(8196, 2048, 1024), dropout=0.3):

    input_layer = layer = Input(shape=input_shape)

    for n_filters in filters:

        layer = Conv2D(filters=n_filters, kernel_initializer='he_normal',
            kernel_size=[3, 3], padding='same',
                      activation=tf.nn.relu)(layer)

        if batch_norm is True:
            layer = BatchNormalization()(layer)
        layer = pooling(pool_size=[2, 2], strides=2)(layer)

    layer = Flatten()(layer)

   if dropout != 0
     layer = Dropout(dropout)(layer)

    for neurons in dense_neurons:

        layer = Dense(neurons, activation=tf.nn.relu,
            kernel_initializer='he_normal')(layer)
        if batch_norm is True:
            layer = BatchNormalization()(layer)
    softmax_output = Dense(classes, activation='softmax',
        kernel_initializer='he_normal')(layer)

    model = Model(inputs=input_layer, outputs=softmax_output)

    print(model.summary())

    return model
```

By iterating through the filters passed, layers are added, batch normalisation applied if desired, and the selected pooling layer connected, until we have appended all required convolutional and pooling layers to the model structure.

Similarly, the dense layers are attached, following the flattening of the feature maps returned by the final pooling layer, which is necessary to connect with the dense, feed-forward component for high level reasoning prior to prediction.

This structure allows easy configuration of networks by passing arguments to the run file, which draws models, generators and utilities from separate modularised files. Only one function is then required for various models, easing experimentation.

We can then use this to explore the effects of architecture.

3.3.3 Transfer Learning

With transfer learning, we wish to optimise the network initialisation and improve model performance by importing the basic geometric shapes and edges learned in first layers of such networks. This optimises training and may aid in the problem of low samples, as perhaps less data will then be required to create representational filters. However, as the problem domains of ImageNet pre-trained models, close up and low resolution images of objects, and the present domain of interest are distinctly separate, only the earliest layers are likely to be of any use; if at all.

ResNet is a convolutional network utilising a technique called residual learning. This is unique in the 'layer skips' applied. Skipping, with double or triple layer skips, reuses weights from other layers and hence requires less computational power for complex networks. This allows deeper architectures to be created which do not suffer from divergence of performance caused by weight saturation.

Loading the pre-trained models available in keras is simple. Note the 'input_shape' parameter, augmenting the structure of the imported model to allow for our significantly larger images. For this to be effective on the highest resolution image approach, the features detected would likely have to be relatively small — this model was designed to identify features in 224×224 size ImageNet files, the features, for example, cats and

dogs which take up a large proportion of the frame.

To load only the first layers, that is, the first convolutional and pooling layers along with activations, padding and batch normalisation, the primary approach, the following model is created, shown segment by segment:

```
def first_layers_resnet(input_shape=(800, 1200, 3), pooling=AveragePooling2D,
    batch_norm=True,
               filters=(32, 64, 64, 96, 128, 128), dense_neurons=(2048, 1024),
                   dropout=None):

    res_model = ResNet50(include_top=False, weights='imagenet',
        pooling='avg', input_shape=input_shape,
                     classes=classes)

    input_layer = layer = res_model.input

    for resnet_layer in res_model.layers[1:7]: # adding res net layers (1st
        conv pool and actvns etc)
        layer = resnet_layer(layer)

    del res_model
```

Similar to the customisable functions generated for the other CNNs, the code here allows customisation of setup and architecture. The model is imported by calling the 'ResNet50' function, with the top (prediction layer) removed, the weights for the model trained on the ImageNet repository, which is the reason for using this model.

The input layer is isolated before iteratively appending the first 6 layers to the input, which are convolution up to pooling. The resnet model is then deleted to clear its memory usage.

Then, similar to the previous CNN, we append the convolutional and pooling rows along with batch normalisation and dropout, if desired; and the dense layers as above.

```
    for n_filters in filters: # adding own layers
```

```
        layer = Conv2D(filters=n_filters, kernel_initializer='he_normal',
            kernel_size=[3, 3], padding='same',
                      activation=tf.nn.relu)(layer)
        if batch_norm is True:
            layer = BatchNormalization()(layer)
        layer = pooling(pool_size=[2, 2], strides=2)(layer)

    layer = Flatten()(layer)

    if dropout is not None:
        layer = Dropout(dropout)(layer) # throwing in dropout after flatten

    for neurons in dense_neurons:
        layer = Dense(neurons, activation=tf.nn.relu,
            kernel_initializer='he_normal')(layer)
        if batch_norm is True:
            layer = BatchNormalization()(layer)

    output = Dense(classes, activation=tf.keras.activations.softmax,
        kernel_initializer='he_normal')(layer)

    model = Model(inputs=input_layer, outputs=output)

    for layer in model.layers[1:7]:
        layer.trainable = False

    model.summary()

    return model
```

the model takes, then, the input layer from ResNet, and connects this to the output of our soft-max model. We set these first layers untrainable so as not to undo the basic shapes and edges learned when training.

3.3.4 Multi-Input Models

A Multi-input model — numeric and image array — is possible for the problem domain. Given the application is intended to process large quantities of images without human effort, inputting most features present in the supplied volunteer sheets is inappropriate; for example, we do not wish to input the terrain type (referred to as coastal character) for every single image, and we cannot reliably auto-detect this — e.g., associate a gps location with a coastal character, given the inter-sector variety and that flight paths may change.

Recall that images contain the earlier discussed meta-data known as EXIF, and this can be automatically stripped from images for use in the model. Sector, to be autodetected via a function which compares the GPS coordinates of the plane identified as flying over/near the specified sector, may be a very useful attribute combining this with features learned by the CNN could lead to interesting inferences; for example, some sectors may have a particular item commonly occurring, e.g., the timber in sectors 1-10, and so the CNN may pay more attention to that particular feature: timber in the area that a shipping box full of timber was lost near. Additionally, the time of day may impact the CNN such that it devalues the importance of colour later/earlier in the day, given the de-saturation of darkness. Other examples may arise such that the black box model learns discrimination that a human observer may overlook. Similarly, the latitude/longitude features themselves, and the date (rationally one might conjecture we use the time of year, accounting for seasonal variation, rather than full date; however we have limited data-points. After a long time of data collection and recompiling the model, SCRAPbook could have a highly accurate system following methodology such as this.)

This approach could also increase likelihood of a positive prediction in sectors that tend to have more litter, which may not generalise to future data. A potential flaw though is that the model could under-perform when analysing images from sectors with few samples; on completion of this multi-input models training, it is then appropriate to analyse performance by sector, particularly on the test set as generalisability is a prime concern (and more information could promote overfitting in the network), to check for

a relationship with number of samples in said sector, and perhaps other patterns such as the backdrop of those areas affecting ability to learn.

?? in Appendix A shows the architecture of the proposed multi-input single output model; Figure 3.7 shows the summary, including parameter count and layer connections, input shapes.

The connection here is after the flatten layer, concatenating those inputs (our second input being just one feature, and adding just one neuron to the following layer.) They then both can be used for high level reasoning in the dense layers.

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_3 (InputLayer)         [(None, 800, 1200, 3 0
__________________________________________________________________________________________________
conv2d_1 (Conv2D)            (None, 800, 1200, 32 2432      input_3[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 400, 600, 32) 0         conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)            (None, 400, 600, 64) 18496     max_pooling2d[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 200, 300, 64) 0       conv2d_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)            (None, 200, 300, 96) 55392     max_pooling2d_1[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D) (None, 100, 150, 96) 0       conv2d_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)            (None, 100, 150, 128 110720    max_pooling2d_2[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D) (None, 50, 75, 128)  0       conv2d_4[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)            (None, 50, 75, 160)  184480    max_pooling2d_3[0][0]
__________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D) (None, 25, 37, 160)  0       conv2d_5[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)            (None, 25, 37, 192)  276672    max_pooling2d_4[0][0]
__________________________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D) (None, 12, 18, 192)  0       conv2d_6[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)            (None, 12, 18, 224)  387296    max_pooling2d_5[0][0]
__________________________________________________________________________________________________
max_pooling2d_6 (MaxPooling2D) (None, 6, 9, 224)    0       conv2d_7[0][0]
__________________________________________________________________________________________________
input_4 (InputLayer)         [(None, 3)]          0
__________________________________________________________________________________________________
flatten (Flatten)            (None, 12096)        0       max_pooling2d_6[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)    (None, 12099)        0       input_4[0][0]
                                                          flatten[0][0]
__________________________________________________________________________________________________
dense (Dense)                (None, 2048)         24780800  concatenate[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)              (None, 2048)         4196352   dense[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)              (None, 6)            12294     dense_1[0][0]
==================================================================================================
Total params: 30,024,934
Trainable params: 30,024,934
Non-trainable params: 0
```

Figure 3.7: Multi-Input Model Summary

From the model's summary, we can see the number of parameters is relatively similar to image only networks; therefore, including the numeric information is not computationally costly. As the number of features in the second, numeric data increase, this will increase also, but still remain a tiny proportion of parameters in the multi-input model. Therefore if an any associated performance improvement, even only training speed, is discovered, extending the configuration to include the numeric data is a worthy endeavour.

Chapter 4

Analysis

Implementing and evaluating the methodologies proposed in Methodology begins with the dataset, which we analyse and prepare based on the approaches posed; confronting issues such as the presence of unsuitable images, duplicates data-frame rows and images, and the creation of IDs for cross reference between images and spreadsheet entries, and image compression.

4.1 Building the Dataset

Refer to appendix for this implementation in full, not in the main body due to the length and the somewhat tangential relevance to the main problem; the following are presented as significant tools developed..

4.1.1 Tools Developed

Below is defined for consistence in the quality column of the master data-frame.

```
def fix_quality(qual):
    """
    Categorises 'quality' input to one of 4 values, or nan.
    """
    try:
        vals = ['h', 'm', 'l', 'n']
```

```
        if str.lower(qual[0]) in vals:
            return str.lower(qual[0])
        else:
            return np.nan
    except TypeError:
        return np.nan
```

This works as the notation variances ranged from the full words, e.g., 'higher,' 'Higher,' and capitalisation: 'm,' 'M', 'Mid,' Medium.' In some cases, notes were passed as the volunteer was unsure; we exempt these due to the ambiguity.

This is then used to filter the unusable images, which is trivial.

Removing the unsuitable images is intended to improve training of the network, preventing conflicting information from being passed to the network; for example, the classifying of 0 for these images could prevent commonalities with the typical representation for class 0 being discovered as the structure of the image changes from the typical: sea shore, foreshore, back-shore, land. It's worth noting however that the network in deployment will be passed these images, and the network may underperform on these samples. Were more data gathered, with consistent recording of the quality column, or a new column which more explicitly refers to the image's suitability, a better network can be trained, perhaps with two outputs: one for accumulations, and one for suitability.

Primarily the utilities developed strip EXIF data from images for cross-reference with the data-frame, the function then having an additional purpose to extract this kind of data for the proposed end application which generates a data-frame for each image through iteration.

Many tools were created, however the following presented represent the significant utilities.

The function below, which is half of a larger function to follow, requires the PIL Image and ExifTags modules. First, the function extracts EXIF data from a given image, where the function is passed the path to that image.

```
def open_image_get_lat_long(image_path):
    img = Image.open(image_path)
    exif = {ExifTags.TAGS[i]: j for i, j in img._getexif().items() if i in
        ExifTags.TAGS}

    try:
        info = exif['GPSInfo']

    except KeyError:
        lat, long = '', ''
        return lat, long
```

An image is opened via PIL Image by its path. Then a dictionary containing all available EXIF informations is created. 'Try' is used to handle the cases where the images do not have any EXIF data, these images often being from the subset which have lower resolution and were perhaps captured with a different camera, or at-least with different settings. The exception returns empty latitude and longitude information which we can then infer in ID creation that the next best alternative, sector and image name, be used for cross reference.

Following this, the latitude and longitude are extracted from the dictionary if available. Some images contain EXIF but not GPS data.

```
    try:

        lat = info[2][0][0] + ((info[2][1][0] / info[2][1][1]) / 60)
        long = info[4][0][0] + ((info[4][1][0] / info[4][1][1]) / 60)

        lat, long = round(lat, 5), -round(long, 5)
    except KeyError:
        lat, long = '', ''

    return lat, long
```

Indexes are selected based on exploring the dictionary returned, which contains further information such as date and time of capture. The longitude is rounded as a positive and then made negative to maintain consistency with the method used by Scrapbook in the spreadsheets, the rounding to 5 decimal places also for consistency. Division by 60 converts to decimal, the notation used in Scrapbook sheets.

To generate this data for each sample, this function is executed iteratively over a set of file paths. The returned data is then appended to a data-frame. Paths are gathered through functions such as the following, with more available in the appendix.

```
def get_paths(img_dir):'

    paths = []

    for path, dirs, files in os.walk(img_dir, topdown=True):

        paths.append(path)

    return paths[1:]
```

This collects subdirectories for later image search. Note that the first path returned is a null directory so the list is sliced from index 1 upward.

With these, the full paths to each image is gathered.

```
def get_file_paths():
    paths = get_paths()

    file_paths = []

    for path in paths: # iterate over all paths, including subdirectories

        for item in os.listdir(path): # only interested in files

            if os.path.isfile(os.path.join(path, item)) and
                (item.endswith('.JPG') or item.endswith('.JPEG')):
```

```
                file_paths.append(str(path) + '/' + str(item))

    return file_paths
```

IDs are then somewhat trivially generated, and used to assign a path to an image with a greater degree of certainty than the simpler method of sector and image name. For example, the following component of the ID creator:

```
    if lat != '' and long != '':

        id = str(sect) + '~' + str(file_name) + '~' + str(lat) + '~' +
            str(long)

    else:
        id = str(sect) + '~' + str(file_name)
```

In cross reference, only the filename and the GPS are used initially as some sectors are misplaced; but maintaining the sector in the ID allows for easy identification of a sample. Otherwise, sector and image name is a reasonable alternative.

4.2 Preliminary Analysis of Spreadsheet Features

Given the proposition of a multi-input model, determining whether Sector, latitude and longitude have a bearing on the levels of intensity is appropriate. While we will be throwing our data into various neural networks, in what's often called a 'black box' approach, it is also still valuable to understand any relationships present among the features/data.

Figure 4.1 shows samples by sector.

Images per sector vary from around 40 to over 600, but we note sectors 23 and 57 both carried some ambiguity in designation, some images in the sector 23 folder and

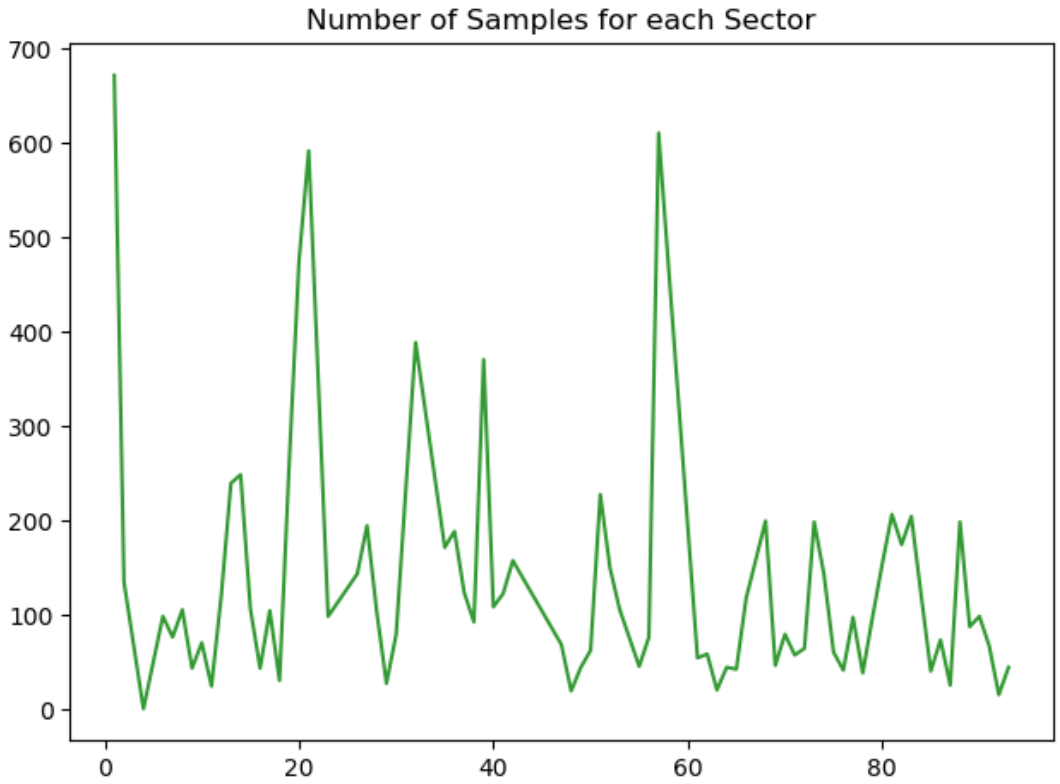


Figure 4.1: Latitude and Longitude vs Litter Accumulations

spreadsheet for example actually belonging to sectors one or two over (thus explaining the very few samples in sector 20), and similarly sectors 57-60 were lumped together. As they're similar locations, captured on the same flight, it doesn't affect analysis much. This may explain the high sample counts, that they're simply amalgamations of nearby sectors. Sector one at 600 samples, however, is an outlier in terms of number of samples; the rest typically lie between 50 and 200. The cause may be simply the length of the area, having more coastline than some other sectors at which the sea is bordered by cliffs and hence there is no area for which litter to wash up.

An initial idea was to train separate networks for each sector, however the impracticality of training 93 networks (76 practically due to missing sectors and amalgamations) coupled with the inter-sector variety suggest one network, perhaps with a multi-input segment providing a means to account for outer-sector variation. Terrains do change with sector, northern and higher elevation (rocky cliffs for example) coastlines being

typically colder and hence more barren; but there are commonalities in all, and learning from the images in other sectors should carry over, being hugely beneficial to all data as opposed to separate networks.

Figure 4.2 depicts accumulations by location (of the aircraft) by shade as shown in the colour bar on the right, lightest values being highest intensity instances. Each point represents an image captured. The path tracked along the coast outlines somewhat a map of Scotland, given the aircraft flies around all coastlines.

One anomylous picture appears in the centre of land, perhaps an image taken of the aircraft crew before flying out, an accidental capture, or a mistaken recording of coordinates in a spreadsheet; a sample that was not filtered by image quality or lack of a label.

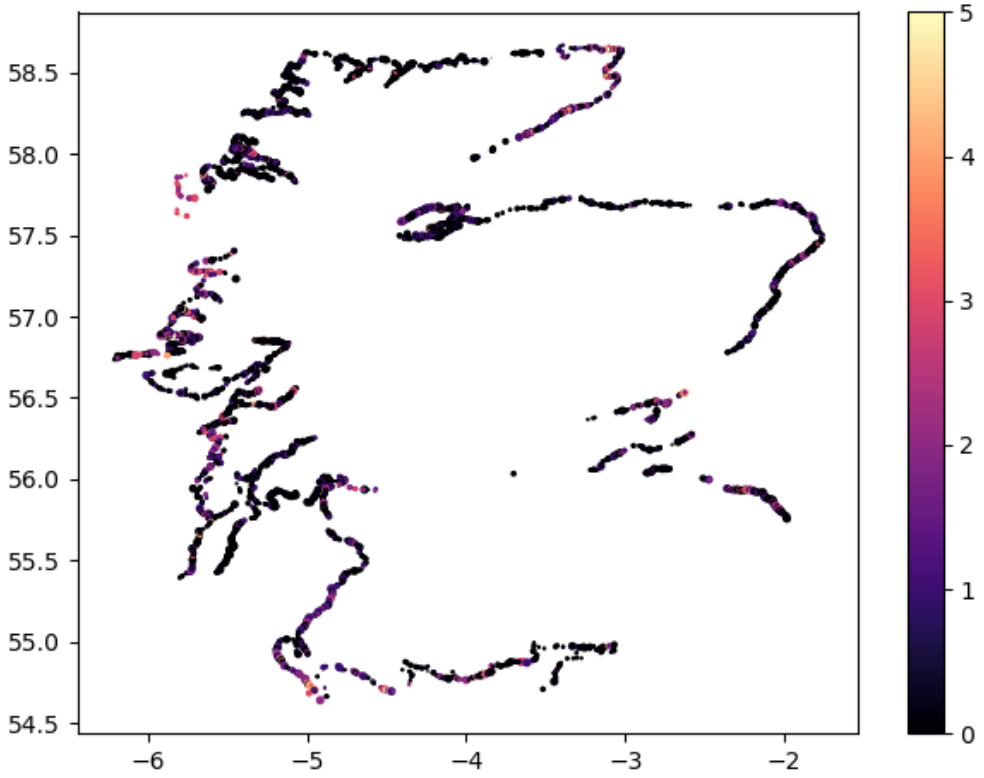


Figure 4.2: Latitude and Longitude vs Litter Accumulations

We can see the areas of dense accumulations, likely corresponding to ocean tides, perhaps sewage flows, rather than nearby population density. A particularly dense region appears on the top left, with a latitude of around 58° and longitude −5.7°. While there are many instances of litter dense areas on the east coast, the west coast appears worse, possibly relating to the ocean currents.

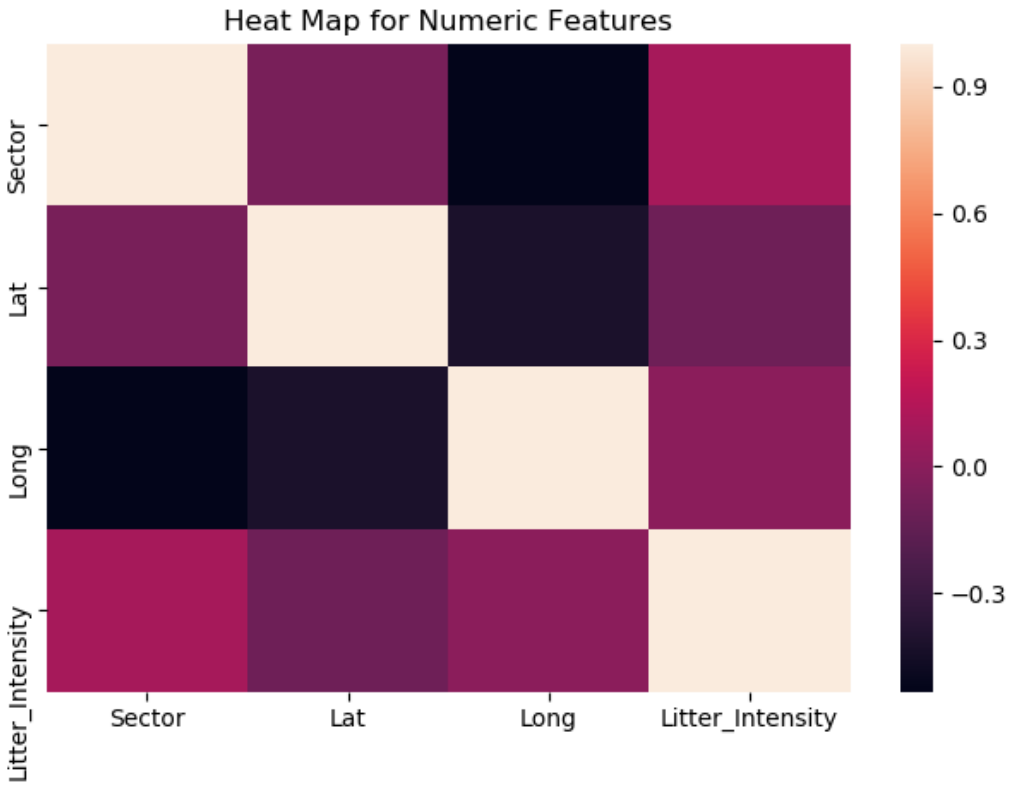


Figure 4.3: Heat Map of Numeric Data-Frame Features

From Figure 4.3, sector, longitude and latitude don't appear to correlate with litter intensity. Expectedly, longitude and Sector correlate, however, perhaps surprisingly, latitude does not, while it does correlate with longitude — referring to Figure 4.2 this is perhaps caused by the winding costs on the west, meaning several sectors are adjacent in terms of latitude, thus reducing the correlation.

Despite these factors having negligible correlations with litter intensity, they still

may aid the network in the high level reasoning of the dense layers, for example it may be determined that for areas which typically have low accumulations, a prediction of class 4 would be unlikely, so could require higher confidence to assign a higher soft-max value, or just decrease the soft-max prediction.

4.3 Initial Models

The first network, architecture available at subsection A.4.2, applied for the problem used images of size 512×356 to allow faster training and therefore more networks trained for exploration, as-well as easier transfer learning. There initial models were largely unsuccessful; primarily due to the limited ID association, using sector and image name for label association, causing some conflicting information by incorrect labels, and having far less data in total — some of which being, at the time, unidentified duplicates. Additionally, the lower resolution, observed to correlate strongly with performance, is a factor in the low performance.

	precision	recall	f1-score	support
0	0.32	0.21	0.26	164
1	0.35	0.24	0.29	184
2	0.21	0.38	0.27	120
3	0.22	0.36	0.27	110
4	0.00	0.00	0.00	32
5	0.00	0.00	0.00	20
avg / total	0.27	0.26	0.25	630

We can see there are only 630 samples in total for this test set, which is 0.2 of the whole set. All classes perform poorly, the ceiling observed through many runs with this resolution and shallower architectures — the above is only five convolutional layers. More layers led to inability to converge in training, suggesting over-complexity.

Shallow networks, which paradoxically can contain more parameters than deeper ones due to the gradual reduction of parameters in the CNN by creating filters and

then downsampling, tended to over-fit more so than deeper ones.

The model depicted here quickly reached a full test fit within a few epochs despite relatively low learning rates explored, the trade-off with very slow loss updates; further reductions appeared to simply delay the same over-fitting. A ceiling of up to 0.03 precision, recall and f1-score equilibrium was observed. Thus a generalisable, accurate network could not be drawn from this low-res data, with this architecture. Samples by class correlates strongly with performance, expectedly.

More layers and filters are necessary to achieve better test results, with at-least seven to eight convolutional/pooling layers appearing necessary. However, given the final architecture will require its own optimisation in this regard, explorations for smaller resolution datasets are prepared only for experimentation and generalised to the high resolution architectures.

Therefore we proceed to deeper architectures, and higher resolutions.

4.3.1 Deeper Network

Dropout of 0.3 and batch normalisation at all layers was used to mitigate over-fitting.

The architecture most complex, expectedly, over-fits the quickest. Within 800 samples, at the learning rate of 0.00001, a 0.96 training fit was reached. The presence of over-fitting is difficult to mitigate, particularly with deeper architectures. With shallower and less filter configurations, divergence in training and over-fitting represent two ends of the scale from which model's performed. Addressing this, for example with data augmentation, can be found in subsection 3.2.2.

In Figure 4.6, we can see the training accuracy quickly rise to a full fit despite attempts at regularisation and a low learning rate. Weighted accuracy, in orange, diverges from unweighted and is low at 0.35; the model classifies around 0.6 of samples as an aggregate correctly. This is caused by the network's only decent performance being for class 0, as seen below, and despite the class weighting which leads the network away from learning representations only for the dominant class. The network has learned to

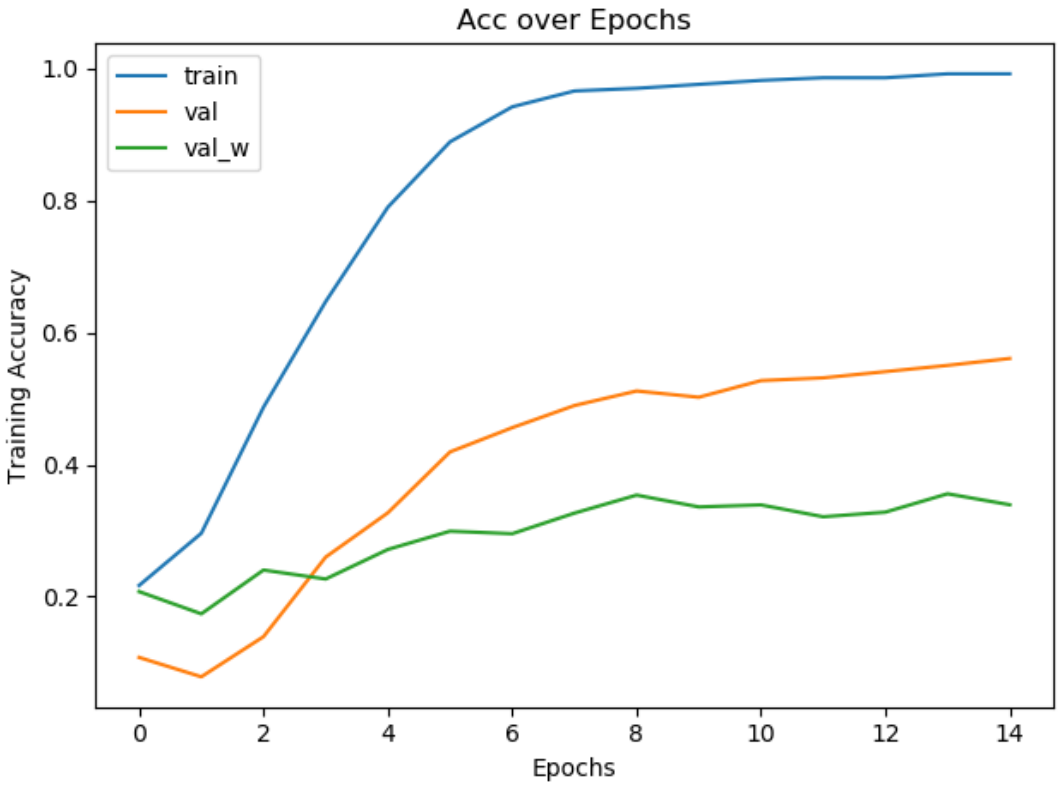


Figure 4.4: Deep CNN Training

fit the class 1 + samples in the training set, but has failed to generalise.

Here we can see the performance degrading as the number of samples in each class decreases. The cause is clear: lack of samples.

```
             precision   recall f1-score  support

          0       0.76     0.66     0.71     1174
          1       0.32     0.36     0.34      412
          2       0.16     0.17     0.17      152
          3       0.13     0.20     0.16      111
          4       0.10     0.12     0.11       60

avg / total       0.56     0.52     0.53     1909

Now doing confusion matrix
```

```
[[780 203 65  90  36]
 [164 149 46  36  17]
 [ 49  57  26  14   6]
 [ 27  42  16  22   4]
 [ 13  22   9   9   7]]
```

With the image only, He initialisation, batch normalisation, dropout, regardless of variation in complexity, number of neurons, the ceiling of performance appears to be around 0.35 weighted accuracy, meaning adjusted so that each class weighs an equal component. While the models can classify 0s with somewhat acceptable precision and recall, the rest of the classes were predicted poorly.

From the confusion matrix, we can see that the most common prediction for all classes is 0, meaning the kind of learning desired, that the network would mistake classes for their neighbours more often than farthest classes, is not being achieved. This is likely related to performance in general, as we can see that class 0 samples were most commonly mistaken for class 1, which is an acceptable error.

4.3.2 Effect of Sampling Methods

To illustrate the effect of sampling and loss weighting, a typical network configuration is ran with 5 convolutional/pooling layers rising from 16 filters to 60, and 2 dense layers of 2048 and 1024 neurons. No class weighting, and no sampling method is employed.

```
[[0.21140419 0.21032912 0.21088189 0.18906109 0.1783237 ]
 [0.21140419 0.21032912 0.21088189 0.18906109 0.1783237 ]
 [0.21140419 0.21032912 0.21088189 0.18906109 0.1783237 ]
 ...
 [0.21140419 0.21032912 0.21088189 0.18906109 0.1783237 ]
 [0.21140419 0.21032912 0.21088189 0.18906109 0.1783237 ]
 [0.21140419 0.21032912 0.21088189 0.18906109 0.1783237 ]]
```

The above are soft-max predictions for the test set, predicted by the model described above. Clearly each row is identical, meaning every estimation is identical and therefore

trivial; this is essentially the mean model.

As explained before, these relatively low complexity networks still over-fit quickly. But without an equalising sampling technique such as over or under-sampling, and the absence of a class weighting, the network predicts class 0 and 1 almost unilaterally, resulting in a misleading test accuracy of close to 60% — but only due to the dominance of that class. The precision, then, is low.

To introduce a weighting for loss and other metrics in training, the following function is created:

```
def get_class_weights(df):
    """
    Gets ratios of samples so that they're worth some value more than class
        0, most dominant
    """

    cw = [n for n in df.Litter_Intensity.value_counts()]

    return dict(zip(range(0, len(cw)), [max(cw)/n for n in cw]))
```

With this class weighting, (an equalised sampling technique was observed to produce similar results), the confusion matrix changes as follows:

```
             precision    recall  f1-score   support

          0       0.61      0.55      0.58      1172
          1       0.21      0.16      0.18       417
          2       0.11      0.03      0.04       150
          3       0.07      0.03      0.04       111
          4       0.05      0.36      0.08        59

avg / total       0.44      0.39      0.40      1909

Now doing confusion matrix
[[646 193 20  28 285]
```

```
[236  65   7   7 102]
[ 85  28   4   1  32]
[ 67  13   4   3  24]
[ 24   9   2   3  21]]
```

The results are poor still, but the distribution of estimates changes.

The cause of the poor performance here is over-fitting; so, were over-fitting to be solved, e.g., via data augmentation, the need for class weighting may dissipate.

When using under-sampling, we have the ability to randomly sub-sample class 0 and 1 at each epoch, using, for example, $n = 700$ samples for each class each epoch. The result on the distribution of predictions is similar, however leads to spikes at each re-sampling step.

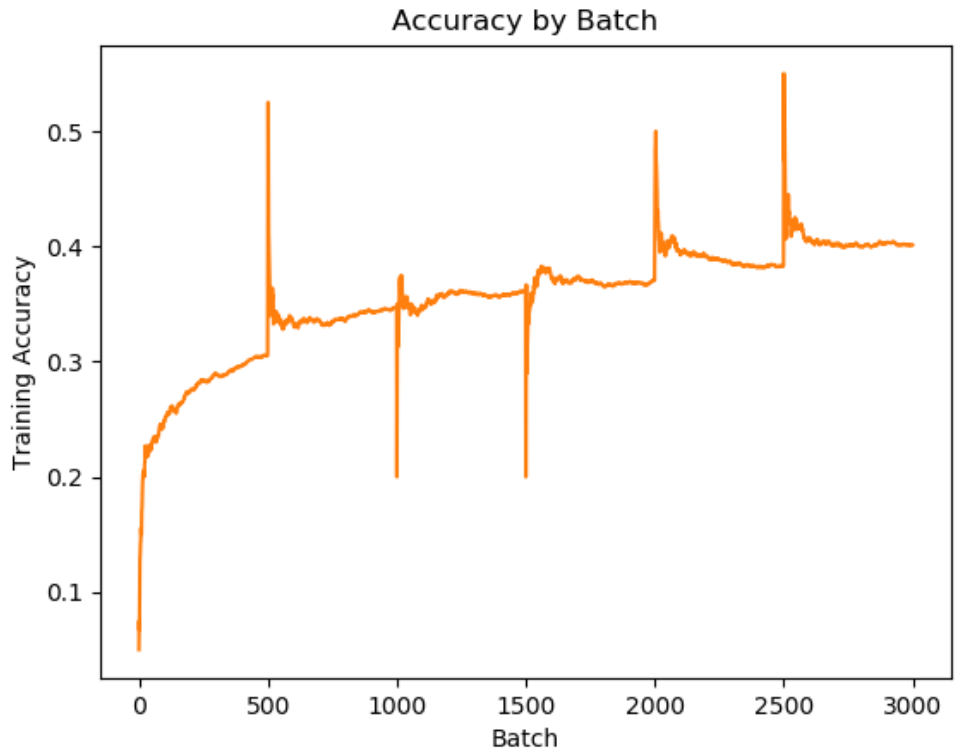


Figure 4.5: Training Performance for an Under-Sampled Approach

Given the number of samples per epoch changes according to over and under sampling, the rate of convergence changes in terms of epochs, but remains the same when considering iterations: there are still the same weight augmentations every $n =$ batch size iterations. However, with under-sampled data, a new random subset of class 0s and 1s at each epoch means often a performance spike, occurring in either direction often, due to learning from the random subset that sometimes does not generalise well; and sometimes better (to the new subset.) This spike is illustrated on Figure 4.5, which shows accuracy by batch rather than epoch to demonstrate this.

When balancing samples, each batch then contains a similar proportion of each class, and hence augments weights (filters) based on losses calculated for samples of each class at every augmentation. However, using class weighting and leaving the data as is can lead to more augmentations for class 0; the augmentations should be more slight as it is under-weighted as compared to other classes. This means that, for example, one class 4 sample has the same effect on loss calculations as 10 class 0s.

Over-sampling has a computational significance as-well, training repeatedly on the same images and therefore extending training time per epoch. Less epochs may be required then, since there is an approximation with more epochs of less samples each.

However, a benefit of balancing samples if done so for validation and training data also (with care not to over-sample the data-frame prior to segmentation but to each of the train, validation and testing data-frames) is the validation accuracy monitored gives a clearer picture of the model's performance as a whole, the equivalent being a weighted average of accuracy for the class weighting approach; this is utilised in training, and the effect of the weighting is clear as iterations pass and the weighted accuracy rises above the non-weighted accuracy.

All methods are observed not to preserve the inherent distribution of predictions in the data, the proportion of samples estimated as belonging to a class varying with accuracy, but not by approach. As seen above, there are several hundred samples (almost all incorrectly classified) as being class 4 despite only 59 samples of class 4 existing in the test set, and under 300 in the data as a whole.

Therefore, the sampling technique, including using a weighting for loss calculations,

have similar effects on the distribution of predictions. Similarly, each method is approximately effective at solving the class imbalance issue, variations mostly being the number of epochs required.

Attempts at using the custom weighted loss to penalise 'worse' errors resulted in divergence and difficulty training. This suggests it may be more appropriate to train for best performance by class and allow the effect of mislabelling classes as their neighbour more often to occur naturally as the model learns better representations; which did occur for class 1 of its own accord. It's unfortunate however that nudging this in the right direction, away from inappropriate learning, is unsuccessful.

4.3.3 Architecture Explorations

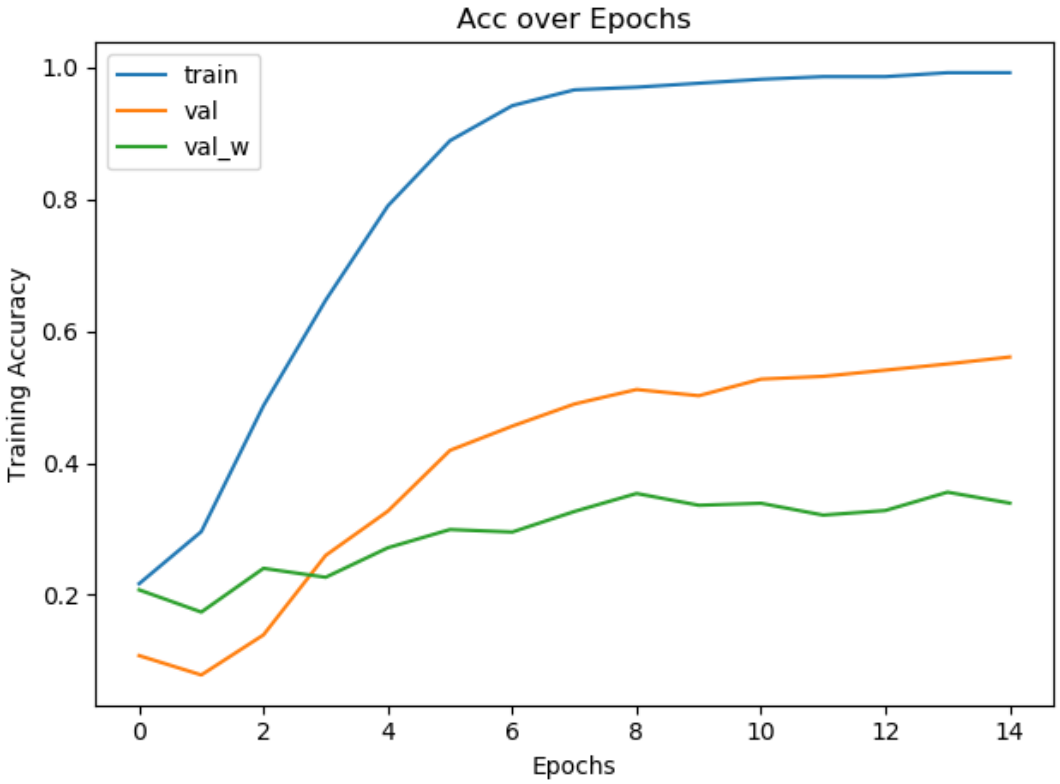


Figure 4.6: Deep CNN Training

Achieving a generalisable performance with such imbalanced data is difficult, nearly all configurations attempted either resulting in non-convergence or over-fitting, like Figure 4.6, and a val/test performance of half the training performance.

From Figure 4.6, the divergence of accuracy from weighted accuracy appears, coinciding with the network fitting class 0 well, but not others. This is not due to over-fitting but the lack of samples, most likely. Using a lower learning rate however draws out similar results, but over a longer time-frame.

To aid this, early stopping is applied, which works well with under-sampled data as there are more validation checks comparatively to weight augmentations. Deciding how many epochs, or to what value of loss/acc the network should arrive at before divergence of validation and training, or non-convergence, is a matter of trial and error. Broad fluctuations in performance sometimes meant epochs passed without improvement, sometimes even lower performance, which then recovered and passed previous levels. Therefore large patience values were necessary.

Batch normalisation in image only networks worked best when placed both after pooling layers, as opposed to convolutional layers, and in both cases additionally after dense layers. Average pooling was observed to perform better.

4.4 Binary System

While a multi-class system is desirable, a model which outputs 1, that is, the probability, of litter presence past some threshold and 0 otherwise would still be valuable. The issue is that with such vast information in the images, a label with only two possible values may be insufficient for the network to learn. However, as there is more data per class due to amalgamation, performance may improve; it may be analogous to the multi-class system trained with categorical cross-entropy and amalgamating predictions that way, or result in an independent new approach at learning and hence new results.

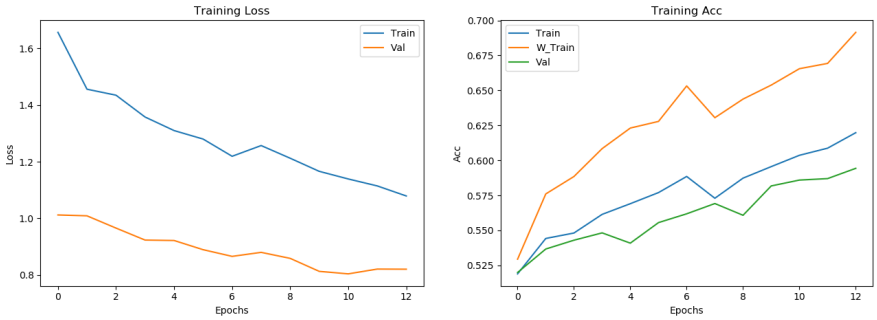
A benefit of this is due to the possibility of training metrics like area under the ROC curve, and the output of a single value, the threshold of which we can use to explore the balance between sensitivity and specificity, we can explore solutions which trade-off between False Negatives and True Positives. As for the threshold, recalling

the domain, it may be better to receive more False Positive and less False Negatives given missing an accumulation is worse than examining less images. However, this is only true up to a point, as returning half the dataset defeats the purpose.

The binary model is trained initially with binary cross-entropy; samples with a label of 2 or more are classed as 1, and classes 0 and 1 are classed 0. As for the distribution, this means that 5/6 of the data is of class 0. A class weighting, which, while in some applications may be unnecessary, is here used to prevent the model from outputting a modal result given that the modal result has been observed to achieve higher accuracy than best non-trivial solutions.

As with other comparisons, the network configuration is 8 convolutional and pooling layers with 4096, 2048 and 1024 dense neuron layers; the only difference is the output, which can be set to either 2 or 1 instead of the usual 5 — when using two, the ouputs are related in the form: $input\_1 = 1 - input\_2$. The full model summary can be seen in Appendix A subsection A.4.1 with the plot omitted as it is several pages long.

4.4.1 Results



(a) Loss (b) Accuracy

Figure 4.7: Binary Training Performance

After early stoppage at 12 epochs and the restoration of best weights, the model reaches a training loss of 1.0785, accuracy of 0.6198, weighted accuracy of 0.6915, and validation loss of 0.8208, non weighted accuracy 0.5943, and weighted accuracy (not pictured) of 0.708. The class weighting is calculated as before, but in this case weights class 1 samples around 5× that of class 0.

Interestingly, as can be seen on Figure 4.7, the validation loss is lower than the training loss, though training accuracy is higher than validation accuracy; behaviours like this can suggest over-regularisation, given dropout is not used in validation and testing. As the same structure and architecture was used with the other comparisons, this suggests the binary structure has less of a need for regularisation and thus suffers less from over-fitting. Perhaps the cause is the relative increase in samples by class. It was observed in experimentation that finding a configuration which maintained a similar training and validation accuracy was significantly easier with this model than the multi-class models.

The weighted accuracy however is only one part of the picture, the poor recall for class 0 and precision of class 1 completing this to give similar results as the equivalent multi-class results. Were we to condense the multi-class results from earlier sections, similar results would be obtained.

	precision	recall	f1-score	support
0	0.91	0.55	0.68	1586
1	0.24	0.72	0.36	323
avg / total	0.79	0.58	0.63	1909

The precision of class 0 is unimpressive when considering the poor recall value, the network only catching around half of the class 0 samples. That the network made only 867 predictions for the 1586 true samples (support) is interesting; the predictions do not conform to the distribution inherent in the data. Perhaps a cost function which penalised this could result in better test results; however there is no guarantee this distribution would continue in deployment. Over-fitting to a conformed expectation of distribution could prevent increases or decreases in litter from being detected. Therefore the system should remain as is, though requiring more data, given that the only goal is to reliably separate 0 and 1; of 0 to 5.

Class 1 was mistaken for class 2 more than three times its presence in the test set — a poor result.

Overall, the same pattern emerges: over-fitting, performance significantly related to samples by class. However, on condensing the classes, the recall of class 0 deteriorated, likely due to the fact that there can be no other misclassification of class 1.

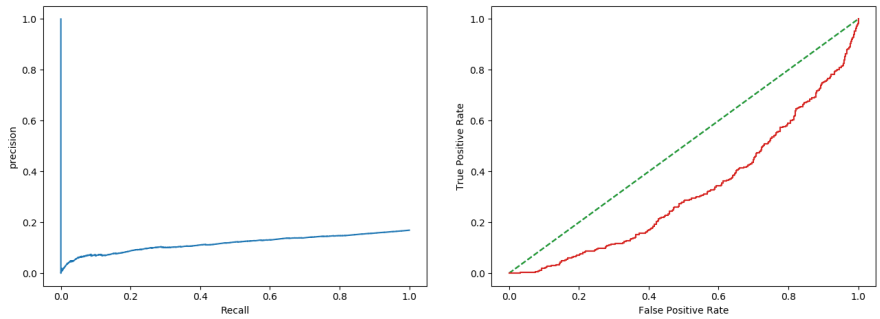
The performance, binary accounted for, is similar to the multi-class model. A significantly greater amount of class 0s were mistaken for class 1s than the converse, which correlates to the support — around half are mis-classified.

```
[[867 719]
 [ 91 232]]
```

The above was trained with binary cross entropy and thus had two ouputs for the soft-max confidences, which can be thought of as probabilities and is somewhat redundant as they should equal $1-$ the other. Training for another metric such as area under the ROC curve — the plot of sensitivity and specificity — with only one output, which we can apply a threshold to for our predictions, may be more appropriate as we can then alter the ratio of false positives to false negatives to result in more predictions but less false negatives; this aligns more with the problem domain, but given the poor performance it is not worth further investigation, of which the main point of experimentation would be the trade-off between images returned as positives and false positives. Were a reasonable number of images, including low confidence positives, returned alongside their probabilities, this may suit Scrapbook more in their analysis. As it stands, the number of false negatives is 91.

As to the question of whether the binary system is more appropriate, given that the errors are of similar magnitude when accounting for the difference in method, that, were we to condense the muli-class system and form two classes from that, a similar result would be returned, the multi-class classification system is recommended moving forward due to the increase in information it brings, and considering the eventual purpose of the model prototypes built in this dissertation. The binary system ultimately gives no advantage over the multi-class, and necessitates information loss; therefore it's concluded that the multi-class approach is superior.

Figure 4.8 demonstrates further the poor performance of the binary model, with



(a) Precision vs Recall (b) Receiver Operating Characteristic

Figure 4.8: Binary Training Performance

subfigure a demonstrating that high values of either type mean low values of the other, and thus the model does not trade off well with these factors; were the line tracking not from 0,0 but higher, and to the top right, this would indicate good performance of the classifier.

As for the ROC curve on subfigure b, which plots False Positives against True Positives, the ratio of false positives to true positives is below the identity line, which means the model performs worse than the random solution.

Considering the research question about selecting a threshold which returns predictions of more true positives at the cost of returning more false positives, and images in general, the sheer number of samples being from class 0 mean that this quickly becomes unfeasible. Based on this and the above analysis, a multi-class system is more appropriate, to be perfected when enough data is gathered.

4.5 Recycling Weights and Models

Transfer learning with ResNet, using imported models entirely and also only the early weights of those models, was unsuccessful, likely due to the huge domain differences, and the difference between the architecture and theory behind these models and our simple CNN.

The complexity of these models, designed for low resolution images, meant they could not be loaded in their entirety to memory while maintaining a resolution accept-

able for performance. However, the weights of the early layers are those of interest.

Additionally, VGGNet was attempted, code available in Appendix A subsection A.3.3, the results of which being extreme over-fitting due to the low batch size of 1 required, followed by crashes due to memory issues.

When applying transfer learning via the early layers of ResNet, an improvement in initialisation *training* accuracy was noted. However, this led to further over-fitting — faster than in the He initialised networks. Therefore, a lower learning rate is used, as-well as a reduction in steps per epoch so validation performance can be monitored more frequently; the over-fitting typically falling between epochs, and hence the best weights are missed.

An equal sampled split so that 600 of each class, over-sampled and under-sampled depending on the class, are seen each epoch, and the more numerous classes re-sampled for the next epoch.

With transfer learning, an accuracy similar to the 'He' initialised network could not be reached, nor exceeded. Accuracy, regardless of architecture, learning rate, seemed to diverge early on, with no generalisable representations learned. Given the imported layers are frozen, it may be that even the basic geometric features aren't as optimal as those learned by the previous CNN.

The predictions themselves oscillated, as can be seen in Figure 4.9, between different mean class predictions; for example, the 60+% non weighted accuracy coincides with the $y = 0$ model. As for the other values, they repeat, as the model switches between guesses.

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1174
1	0.00	0.00	0.00	412
2	0.08	1.00	0.15	152
3	0.00	0.00	0.00	111
4	0.00	0.00	0.00	60

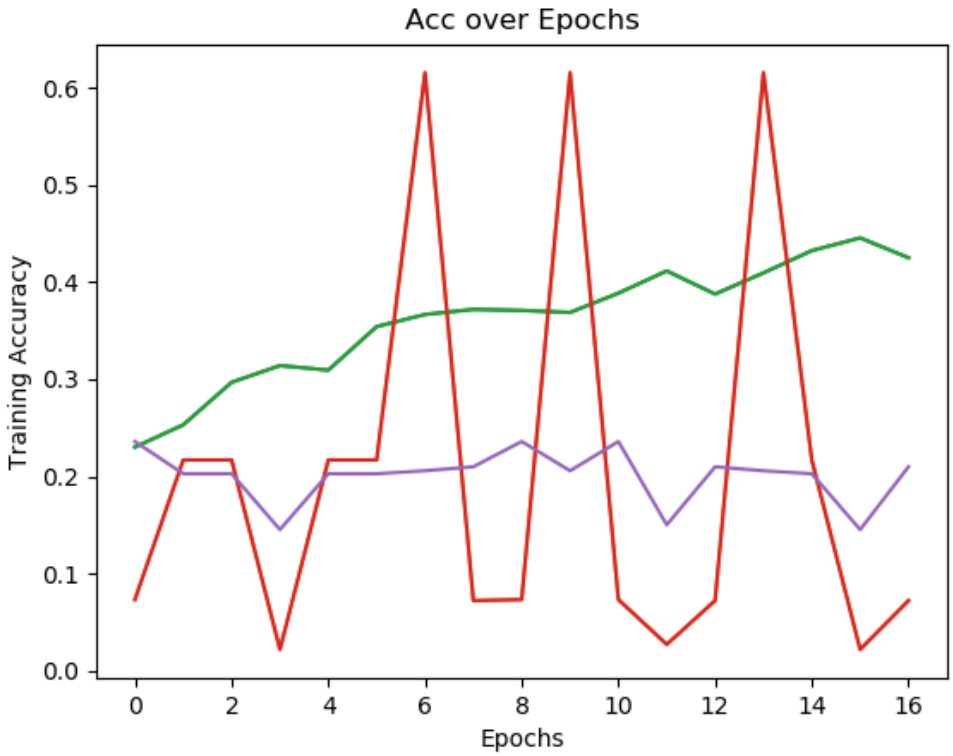


Figure 4.9: Transfer Learning Failure

```
avg / total      0.01      0.08      0.01      1909
```

Only class 2 is predicted. Based on Figure 4.10, the class it landed on is random; were another epoch to have passed, another class could've been selected. The class weighting prevented the selection of 0 unilaterally.

```
[[   0    0 1174    0    0]
 [   0    0  412    0    0]
 [   0    0  152    0    0]
 [   0    0  111    0    0]
 [   0    0   60    0    0]]
```

Therefore we will not employ transfer learning with ResNet in the final model

recommendations, given the low performance.

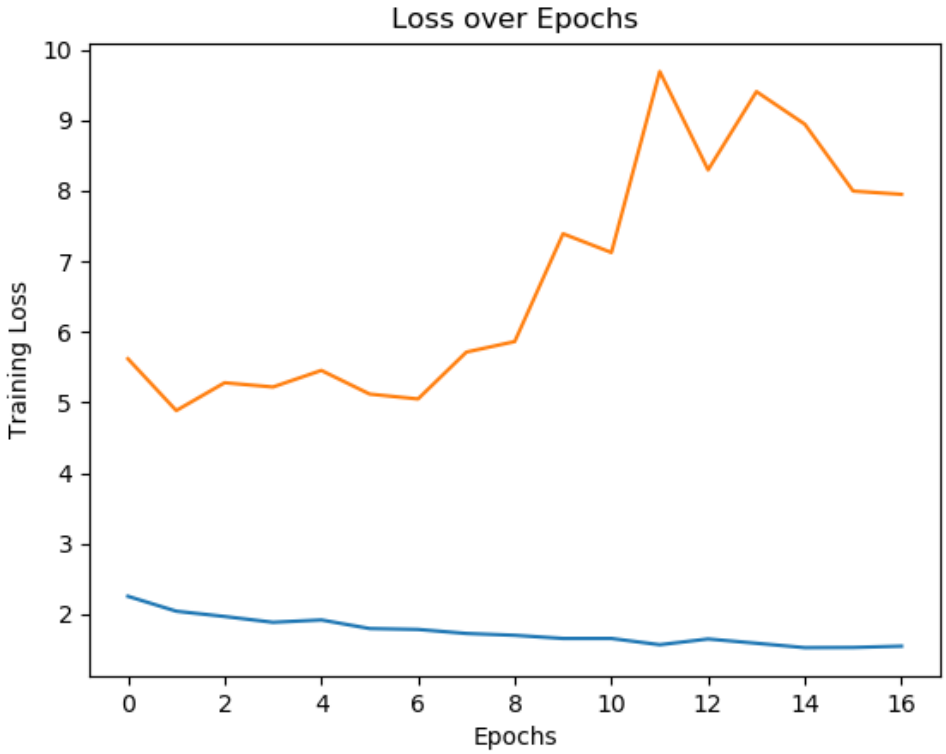


Figure 4.10: Transfer Learning Failure: Loss

The application employed may then be an example of ''negative transfer'' as discussed in chapter 2. To further explore transfer learning would then entail the discovery of a dataset with a domain more similar to the present, such as drone footage or some other high resolution aerially captured data — but perhaps not satellite data due to the overhead angle.

The problem of over-fitting remains, not just in this case but with all models applied. The clear divergence in directions is evidence of significant over-fitting. More interestingly, the (orange) validation does not tend initially to improve prior to a divergence, but tracks upward; as opposed to a parabolic shape where the initial loss improves to a point, where the network begins over-fitting and divergence presents.

When loading the weights from the prior model, trained on the same training set but with images downsized in the generator for speed, not all layers conform to the requirements; for example, the flatten layer expects a specific number of neurons which is different with different input shapes. However, having the weights of every convolutional layer is possible, and beneficial, if only for a stronger initialisation.

4.6 Multi Input Model

Three features are used for exploration of the architecture: sector, latitude and longitude.

The multiple input model, available in full in Appendix A subsection A.3.2, is implemented via the following simple amendments to the single input model, namely the creation of an additional input, where 'multi_features' is the number of features used, passed as an argument, so that the same function can be used to generate multi-input models with arbitrary numeric/categorical inputs.

```
image_input = layer = Input(shape=input_shape)
numeric_input = Input(shape=(multi_features, ))
```

The numeric input is connected to the flatten layer, which connects to the dense layers. This allows the consideration of these inputs in high level reasoning, ideally learning that some sectors are prone to both have more litter accumulations in genera (and vice versa) and also certain types of litter, which could mean that the application of a filter to the image would be given more confidence.

```
layer = concatenate([numeric_input, flat])
```

The inputs are concatenated, adding additional neurons for each input to the flatten layer. This then connects to the dense layers as normal. Connecting to convolutional layers is not appropriate, and inclusion at the earliest possible stage is likely the best approach as the network can consider this information along with the feature maps from the convolutional/pooling layers. It is possible to add this information as part of a single input, as another dimension to the array (meaning 4 dimensional), but this

increases computation time significantly.

Normalisation of latitude and longitude is achieved through min-max inside the generator.

The architecture used for control purposes is: 8 layers with up to 256 filters, and three dense layers with 4096, 2048 and 1024 neurons. Average pooling, 'He' initialisations for all layers are employed.

To maintain the distribution of classes, no over or under sampling technique is employed, however the use of class weighting in loss is employed to prevent the learning of 'mean/mode' solutions, where high accuracies and low losses can be obtained by predicting class 0 for the majority of classes, learning past that point being difficult due to the ceiling of performance observed for the lesser sample quantity classes.

Following presentation of results, comparisons with the single input is made via a table in terms of aggregate performance; and additionally the relation to performance by location and sector is explored as compared to the single input model.

Batch normalisation is applied after pooling layers based on results in prior sections, as-well as a dropout of 0.3 which is observed to balance over-fitting with the divergence and over regularisation associated with dropout, indicated in experimentation by validation results which were higher than training — given dropout is only applied during training.

4.6.1 Results

	precision	recall	f1-score	support
0	0.63	0.57	0.60	1174
1	0.21	0.03	0.05	412
2	0.09	0.12	0.10	152
3	0.05	0.25	0.09	111
4	0.03	0.03	0.03	60
avg / total	0.44	0.38	0.39	1909

```
[[667  32 126 322 27]
 [216  11  51 113 21]
 [ 83   5  19  39  6]
 [ 62   3  14  28  4]
 [ 28   2   8  20  2]]
```

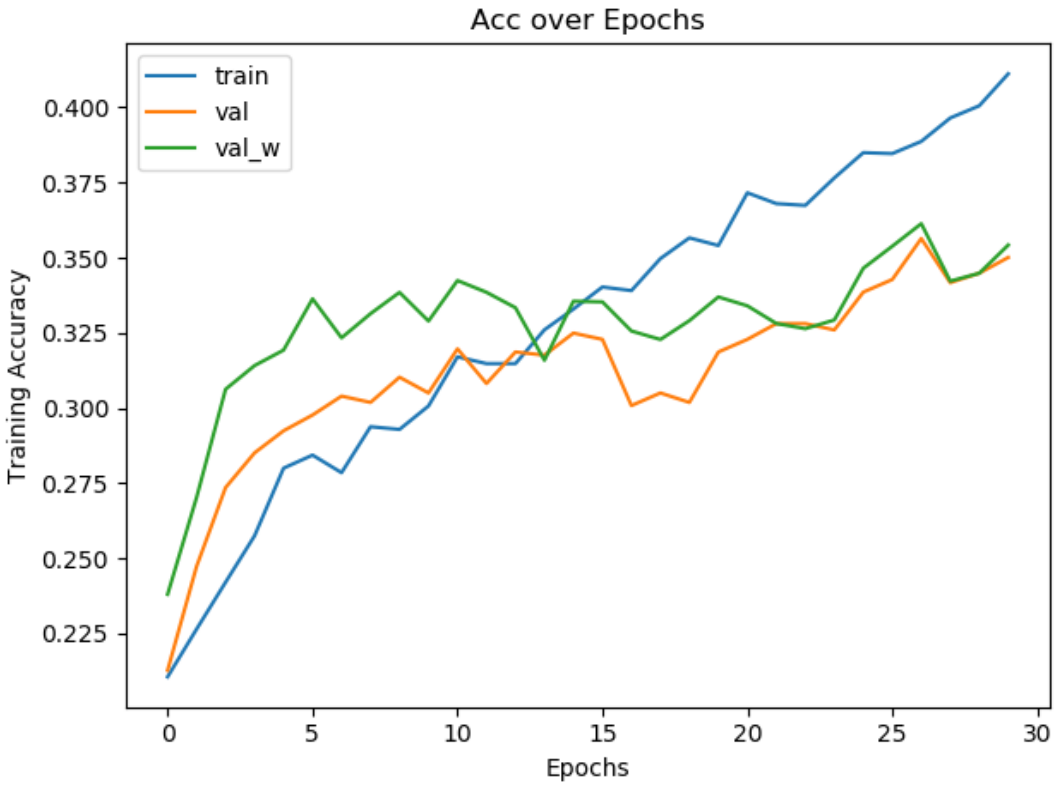


Figure 4.11: Multi Input Model: Training Accuracy

Figure 4.11 shows the training accuracy as-well as validation, and weighted validation accuracy. The model was restored to best performance due to consequent over-fitting. Over-fitting begins around epoch 15, while performance on validation still improves at a reduce rate. Following this, loss stagnated, which was the monitored training metric, and the model diverges leading to poor performance. This represented the best results found for the multi-input with three features. For the present task, this

appeared to represent an improved balance of over-fitting due to the very low learning rate and further epochs. The model trained for around two hours, diverging past epoch thirty to worse validation loss, hence the termination and restoration of best epoch.

It is worth noting that a simple architecture as compared to the similar, yet still subtly poorer, results from the single input architecture; the inclusion of numeric features may reduce necessity for neurons in the high level reasoning dense layers. The postulated reason for this is that the numeric features contribute to the application of filters to the images in a way that is mostly learned by the dense layers in the single input but at more cost to computations required and time.

The divergence is more clear in the loss diagram.

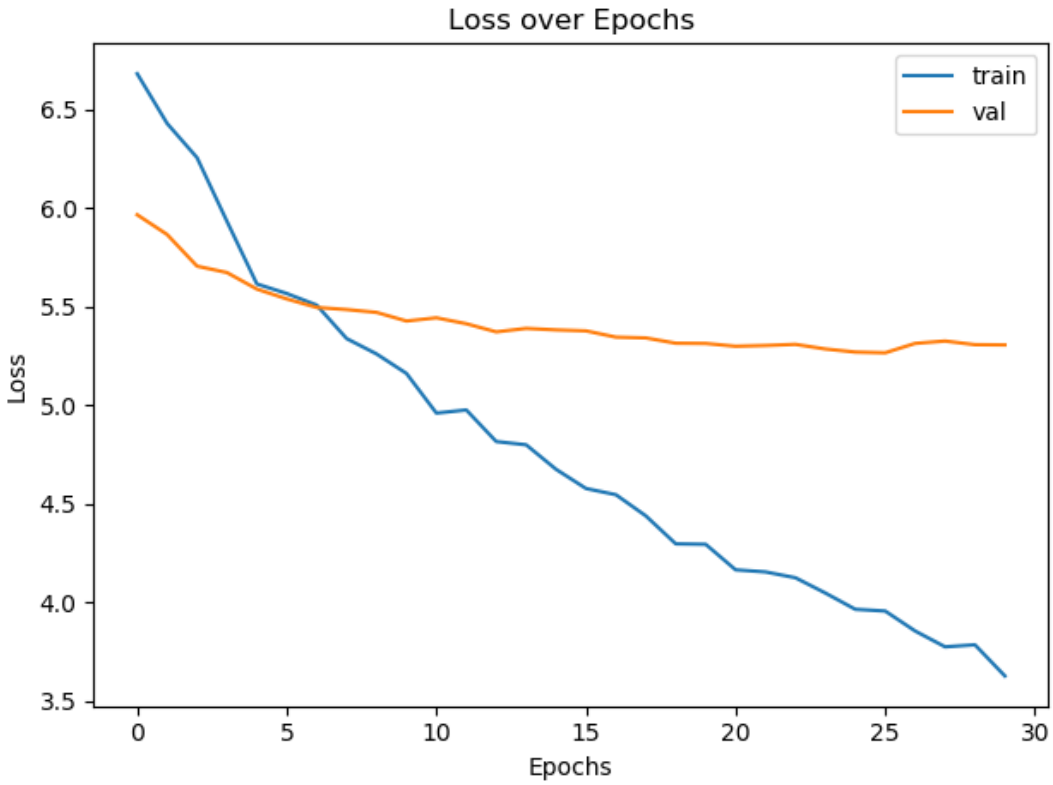


Figure 4.12: Multi Input Model: Training Loss

Where the accuracy appears to still trend upward in Figure 4.11, the divergence is

clearer in Figure 4.12, where validation has diverged significantly from training since the beginning, indicating over-fitting — as with all models attempted. However, that validation loss ceases to improve at a certain point, indicates the criteria for stoppage; which was automatically executed.

Despite the further over-fitting, when treated, the network outperforms the single input model, reaching a weighted validation accuracy of 0.3613, which is 0.01 higher than the best accuracy achieved with the single input model with the deepest architecture used. The aggregate precision increases by 0.04 and f1-score by 0.01.

Additionally, on experimentation with architecture and number of filters, it's discovered that the multi-input model is able to achieve similar or better performance than the single input with less dense layers: the high level reasoning. It's possible that the single input network's high level reasoning was able to treat the terrain variation with a similar effectiveness as the high resolution is with less neurons. Therefore using the multiple input architecture is valuable and perhaps the gathering of more numeric features will become appropriate with the expansion of the dataset.

Therefore the multi-input model improves over the single, image only input model, though exacerbating over-fitting, already a serious issue, and reduces the need for neurons in the dense high level reasoning layers. The final model should then utilise multiple inputs, including sector, latitude and longitude.

In analysing these results, attention should be paid not only to the general performance as compared to the single input model, but particularly to performance by sector and by location in coordinates.

Figure 4.13 shows performance by location, where the δ parameter is the difference from predicted class to actual, giving more information than simply whether the prediction was correct — one away is better than four away. The colour bar is the this delta factor, while the size of the plot symbol is the number of samples in that sector (that region). Darker crosses mean worse predictions. There are a few noticeable dark crosses which are quite small as compared to the rest, but overall the variance is too

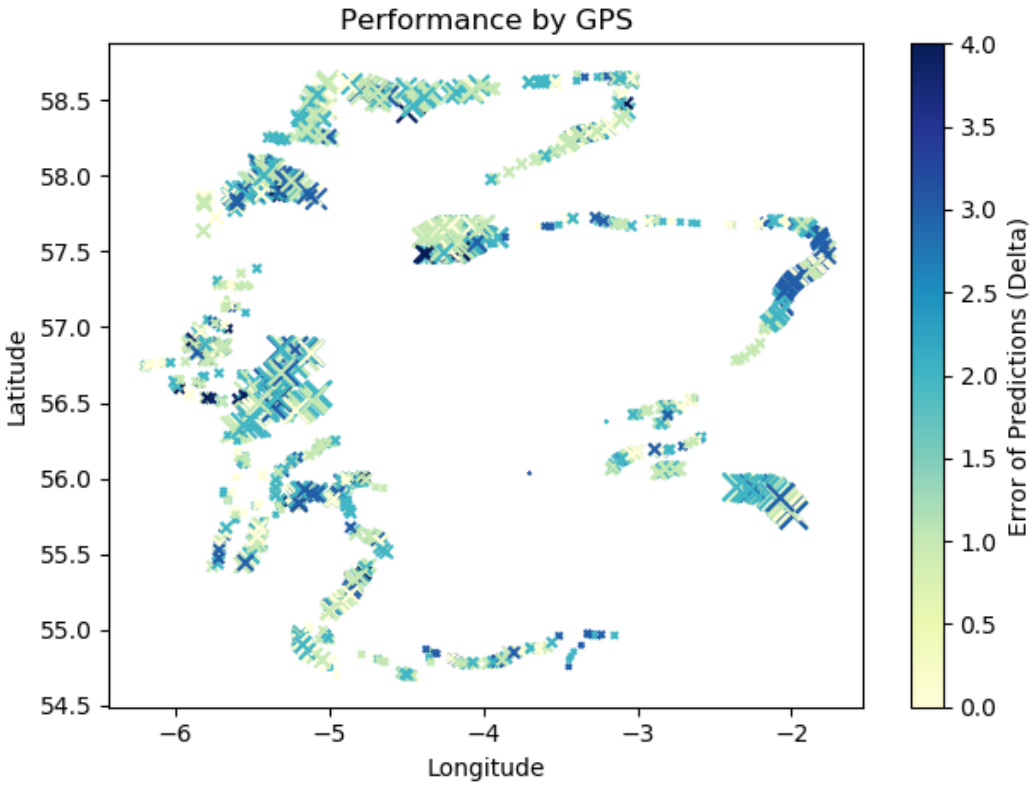


Figure 4.13: Multi Input Model: Performance Map

great to say whether samples by sector effects the accuracy of predictions. Therefore the samples by class may be a greater factor.

Predictions are worse in high intensity areas, given the model adequately predicts 0 while performing worse on classes 1+.

There doesn't appear to be much of a difference as compared to single input models, which supports the idea that the multi-input model allows less dense neurons in the final layers but does not significantly improve performance by location and sector over that model. However, with less filters, and simpler architecture required to achieve similar results, it is a worthy extension to the final model which is constrained by the hardware ceiling.

```
"Single Input Comparison"
```

```
              precision   recall f1-score  support
```

```
          0       0.77      0.28      0.41      1174
          1       0.23      0.15      0.18       412
          2       0.12      0.38      0.18       152
          3       0.11      0.35      0.17       111
          4       0.03      0.20      0.06        60

avg / total       0.54      0.26      0.32      1909

[[331 168 235 205 235]
 [ 65  63 129  75  80]
 [ 19  20  57  26  30]
 [  9  12  37  39  14]
 [  4   8  23  13  12]]
```

```
"Multi Input Comparison"
             precision   recall f1-score  support

          0       0.84      0.41      0.55      1174
          1       0.28      0.26      0.27       412
          2       0.15      0.34      0.21       152
          3       0.13      0.47      0.21       111
          4       0.07      0.23      0.11        60

avg / total       0.60      0.37      0.43      1909

[[481 222 178 180 113]
 [ 67 109 91  99  46]
 [ 11  33  52  39  17]
 [  8  21  17  52  13]
 [  4   9  13  20  14]]
```

The final model will then include an additional input with Sector, latitude and longitude. Further improvements on this in later research could involve the inclusion of

further factors such as date, which is inappropriate at this stage with only one date per sector, gathered on one pass. Cycling observations such as seasoning could be learned, the model eventually becoming invariant.

4.7 Data Augmentation

Insufficiency of samples has been a prime factor in the performances of models thus far, with a clear observed correlation between samples per class and performance of that class. Data Augmentation is one proposed approach to mitigate this, with several unique implementations discussed in chapter 3.

4.7.1 The Augmented Images

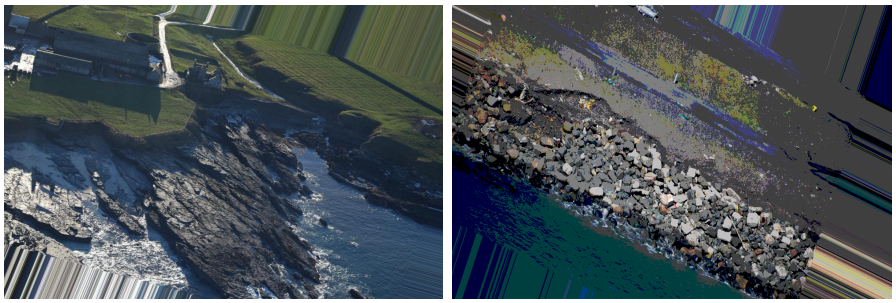
While applying the augmentations manually produced very simple, realistic transformations, they were perhaps flawed by, for example, no fill by nearest pixel for blank regions after rotations. Whether it is better to pad with zeros or nearest pixels is unclear, given both can give harmful information to the network, but use of the Image Data Generator

Here are some examples of those generated by the functions created manually for this project.

On utilising the Image Data Generator keras utility with typical initialisation setups, many images appeared significantly distorted — to the point of blackness in some images; many appeared useless for analysis, the features being obscured and thus providing harmful information to the network.

Sub-figure 1 of Figure Figure 4.14 is heavily effected by shear, and the fills of nearest colour around the regions that would be blank. It is not exactly an ideal representation of future images, the coastal region being partially obscured.

Sub-figure 2 shows channel shift, damaging the information, though the beach feature remains. The distortion appears to place more significance on the centre, as these centre-wise operations should, preserving, in this case, the feature; however in many other images the features are spread throughout. Overall, these augmentations may be



(a) A sheered and Shifted Image (b) Channel and Various Shifts

Figure 4.14: Poor Augmentations

too extreme. A more subtle range both of transformations and their degree of variation will be compared.

Based on the above, it appears that for the present purpose, augmentation is vulnerable to information loss in cases where the important features, litter accumulations, are not central but instead spread out. Therefore transformations that shift should be slight, set to a maximum of 0.05 based on visual examinations.

These are highly unlikely representations of future images, having been augmented to a severe degree. Therefore we proceed with a smaller range of transformations, each of lesser degree.

Channel shifts are then removed, as-well as ranges tightened on remaining operations other than brightness which was increased to (-0.6, 0.6) towards network adaption to lighting variation. Vertical flips, while causing boats to fly, are kept in as they may improve robustness to angle of objects, allowing better filter representations; though logically similar images are not going to appear in future data.

Most interesting of Figure Figure 4.15 is sub-figure c, appearing almost as a whole new image; it's a simple mirror: a flip around the horizontal axis. The robustness gains from such augmentations as Figure Figure 4.15 are worthy of the process, aiding the network in learning filter representations, perhaps like the red box, by exposure to the feature from new angles, lightings and distances.

Overall, the augmentation is not perfect, though it is automatic.

(a) Original (b) Brightness and Standardised

(c) Mirrored, slight Shifts (d) Sheering, Rotation

Figure 4.15: Augmentation Examples

4.7.2 Live Augmentation: Keras Image Data Generator

First, live augmentation was employed via the Keras ImageDataGenerator object, which performs selected augmentations from defined ranges on every image.

Initial attempts, particularly keras live image generator methods, did not improve performance by class, resulting in similar to slightly worse overall performances. One posited reason is the skewed information that can be generated by over-augmentation, which the image data generator can do by applying many augmentations at once, even resulting in black images every so often.

Manual augmentation performed similarly to the baseline, and so for space reasons, expansive augmentation is presented.

4.7.3 Expansive and Dynamic Augmentation

Application of dynamic and expansive augmentation, meaning the dataset is expanded and not augmented in place as with the live approach, requires more adjustments to the generator than for live augmentation. The following is the main methodology, with

the full structure available in the appendix at subsection A.2.1:

First, introduce a variable for how many augmented images are desired per batch. This starts at 0.8, meaning 80% of the batch will be augmented, drawn randomly from the training data-frame.

```
n_augs = round(0.8 * batch_size)
```

Then as the empty containers for the data of each batch are generated, on each iteration, a new random sample of the training data is selected for augmentation.

```
while True: # eternal loop

    augs = df.sample(n=n_augs).reset_index(drop=True)

    images = [] # a list for image arrays
    labels = [] # a list for labels
```

The augmented portion of the batch is appended first, the idea being that the weight augmentations may generalise better to see true samples before the weight updates.

At the end of each epoch, the proportion of augs to be initialised at each batch changes as follows:

```
n_augs = round(n_augs - (n_augs/total_epochs))
```

4.7.4 Overall Best

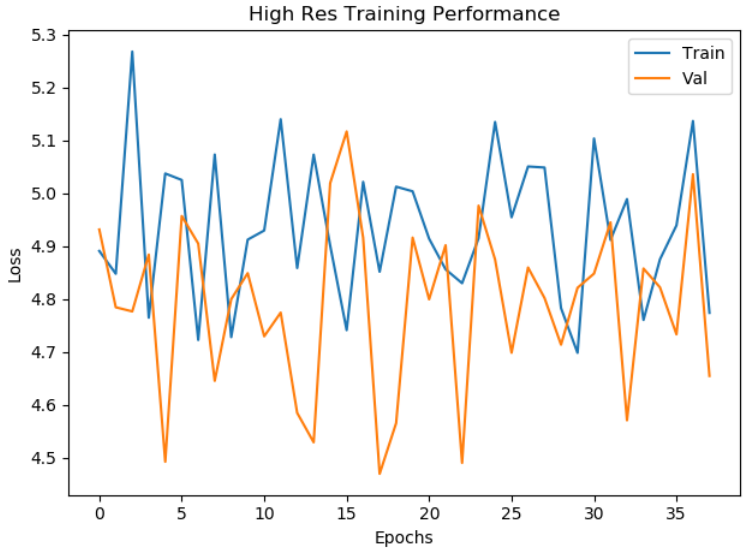
Of the above defined approaches, the manual augmentations with dynamic ratio performed best.

	precision	recall	f1-score	support
0	0.83	0.27	0.40	1174
1	0.24	0.16	0.20	412
2	0.09	0.29	0.14	152
3	0.10	0.42	0.16	111

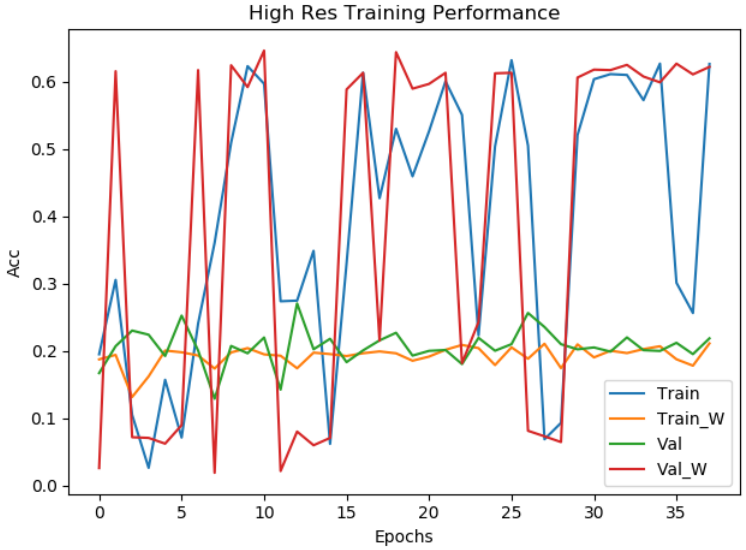
```
          4       0.04      0.20      0.06       60

avg / total       0.58      0.25      0.31     1909

[[314 161 291 244 164]
 [ 44  67  98 112  91]
 [  7  29  44  36  36]
 [  8  13  19  47  24]
 [  4   5  15  24  12]]
```

Performance by class was still

4.8 Attempts at Higher Resolution Input Network



(a) Training Loss



(b) Weighted Training Accuracy

Figure 4.16: Hectic High Resolution Training

Figure 4.16 shows a typical implementation of a high resolution input model, this particular example an 8 layer convolutional/pooling with from 16 up to 256 filters and two dense layers at 2048 and 1024 with numeric inputs of Sector, Latitude and

Longitude as-well as images at resolution 2400×1600 and live augmentation of every sample throughout.

As with other attempts at high resolution inputs, the necessary complexity could not be achieved without running out of memory, at times during training but typically on loading the model itself. The learning rate for this sample was $1 \exp -10$, but various were explored, the following observed: with a batch size of 1, preventing the network from instantly reaching training accuracies of 1 for most samples meant vast reductions in learning rate, middle grounds allowing the network some gains in accuracy but to a ceiling of around 0.23 for 2400×1600, which is a little over half the dimensions of the smallest images provided,the network then unable to improve further and simply oscillating between different solutions that accurately predict class 0 and no others. The batch size of 1 is postulated to be a prime factor in this. With 4256×2832, further reductions in complexity are required and the effect is exaggerated; the balances found between the extreme over-fitting and non-divergence to present solutions which more complex architectures, particularly with many dense layer neurons and using downsampled data, outperformed. However, there is likely a resolution size which performs better than the as yet best demonstrated results, but the observation is the same; some level of downsampling is necessary unless access to more GPUs is available.

The initial goal was to use mid resolution downsampled images to train, then import the convolutional layer weights (the filters), to the higher resolution architectures which can then fine tune and improve performance further. From visual examination of the images, some level of downgrading is acceptable to maintain object and feature clarity, but of-course using higher resolution allows more defined features and hence more accurate filters and feature maps as they're applied to the image, while necessitating deeper and more complex architectures to facilitate the learning of said filters.

In application, a major difficulty presented: the reduction in complexity required to accommodate such high input shapes, meaning the low to mid downsized sample architecture from which we recycle the weights — same data splits — is necessarily smaller, and observed to result in weighted accuracy ceilings of around 0.28 weighted accuracy of the model to be recycled, the typical adequate class 0 performance, sometimes low

recall but high precision, and descending but unilaterally poor predictions as samples per class decreases to class 4, being at best, with augmentation, a significantly poor $f1$ score of 0.16, which is the harmonic mean of precision and recall to give a trade-off between the two as a measure of overall effectiveness for the classifier.

Given to adequately learn from high resolution images, the filters must necessarily be both larger and more complex, meaning combinations of more filters learned in earlier layers, so more layers and filters are required in general.

Computational requirements played a role here also, with some models requiring 5 hours for one pass through the data, which typically resulted in either drastic overfitting or, more commonly, a near complete fit of class 0, for which most of batches (of one sample in extreme cases) augmented weight based on learning from only one class.

Therefore the following results present an unfortunate poor solution, the hardware ceiling being a large factor, and the other related factor being the small batch size required which leads to less generalisable weight augmentations — much literature on the topic suggests each batch, at-least in general, should contain at-least one sample per class, which is not possible with batch sizes of 1. Depending on just how high the input shape is — experimentation ranged from the maximum 6000×4000 with rescaled 12 megapixel images up to the 24 that the majority of the data conforms to, to 4256×2832, the smallest resolution of the image supplied as is, and 2400×1600, then to the 1200×800 which was most successful, and the 600×400 used in the very beginning.

Overall, this indicates some level of downsampling is acceptable, but depends greatly on the available machines for training. With more power, deeper architectures can be trained on the highest resolution images for very long times, likely weeks rather than days and hours as it has been in the present case with the mid range resolutions.

Given that with the mid range resolution model formulations adequate class 0 performances were achieved, though poorer otherwise, it is probable that with enough data, downsampling to this or a similar level is acceptable for future models. The effect of live downsampling is a consideration also; considering continuous recompiling, were a cloud environment used, with plenty of storage space, for the future application, data

could be automatically compressed and scaled down via the data generation script. Additive data augmentation could be applied then too, given increasing the dataset in a desktop application where storage space may be a concern is inappropriate.

4.9 Combining Approaches: Final Efforts

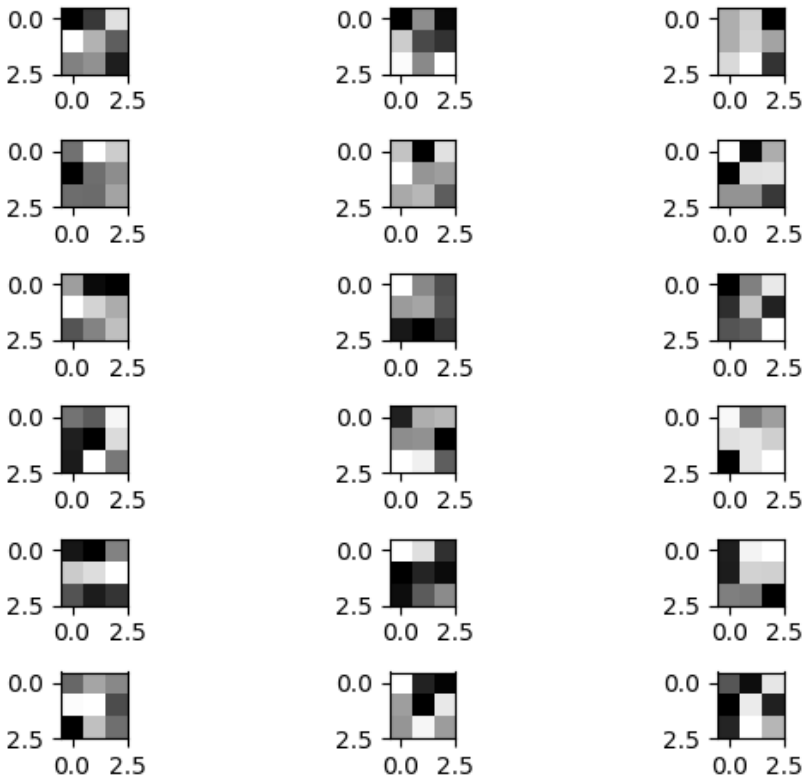


Figure 4.17: The First Layer Filters of the Multi-Input Network (with Live Augmentation)

Combining the findings of earlier sections, namely that the following are observed to improve generalisation performance: incorporating numeric and categorical features (which can be automatically gathered), utilisation of live data augmentation, passing a class weighting while training to mitigate the effects of imbalanced data, and using sample wise weighting with generators to monitor for early stoppage without bias

towards more numerous samples, and iterating through the entire dataset each epoch without under-sampling due to the observed instability, and without over-sampling due to the increase in computation costs with no observed benefit over class weighting.

The ultimate performance is:

```
             precision   recall f1-score  support

          0       0.84     0.41     0.55     1174
          1       0.28     0.26     0.27      412
          2       0.15     0.34     0.21      152
          3       0.13     0.47     0.21      111
          4       0.07     0.23     0.11       60

avg / total       0.60     0.37     0.43     1909

[[481 222 178 180 113]
 [ 67 109 91  99  46]
 [ 11  33  52  39  17]
 [  8  21  17  52  13]
 [  4   9  13  20  14]]
```

This is an overall improvement over best efforts from single input networks and multi input without augmentation, but still not accurate enough to be deployed.

4.10 Proposed Application

While the model is not yet accurate enough for deployment, were it to be so, the model can be loaded via keras in a python script, the architecture saved as a function and the weight saved as '.hdf5'. It's then loaded and the user defined directory is searched for images, which are exposed for prediction iteratively and the results output to a csv for analysis.

An interface allows the user to open directories containing the new images. They are then iteratively exposed for prediction, the results collected and displayed in the

form of a dashboard — showing particularly areas, by sector, that have changed in litter accumulations, as-well as generally displaying lists of all high accumulation predictions and their locations by Sector and GPS of image capture.

Prior to compile via PyInstaller, the application consists of modularised python files, as-well as a keras model file containing the weights. In the modularised python scripts, there are pre-processing tools for the images, as-well as utilities to automatically gather EXIF data which is used both as numeric features and for the presentation of results, given returning an image, which are zoomed in and impossible to identify, for every positive sample would flood the interface. Therefore the results are presented in a data structure format and saved as an excel spreadsheet for Scrapbook to then coordinate their clean ups. Additionally, the application should have a validation interface, where a small portion of the predicted data can be validated/invalidated, then recompiling the model with up to date data, maintaining generalisability.

When compiled to an executable by PyInstaller, all packages, such as Tensorflow and Keras, will be stored in the application, which means a significant amount of space. The application is intended for use on desktop machines; or later as a cloud application.

One of the greatest concerns in providing an application to Scrapbook to analyse directories full of images is that, in order to predict, the model must be loaded and batches, necessarily of size one, provided to the model for prediction. This can consume a significant amount of memory, the models having millions of parameters. A cloud based application would be ideal, but otherwise, a simpler model

4.10.1 Demonstration of Concept: Automatically Gathering Data

For a given directory of images, utilising some functions defined throughout **??** and Appendix A, a data structure can be generated which contains:

- **image name** — the filename of each image
- **file path** — where to find the image, both for passing to the generator to gather predictions, and so users can check the images when analysed, and assess the situation. In the future, were a highly accurate model developed, a GUI could

display images when clicked on; this is more user friendly than the generation of an excel or csv document.

- **Sector** — the projected sector of the image. If folders are still labelled by sector in the future, which may not be the case as Scrapbook move further towards automation, the utilities defined in master csv creation can detect this. However, the main proposition is the use of the sector detection function, which computes nearness to all defined Sectors and selects the closest one.

- **Date** — date of capture. At present, with so few collection dates, this is of limited utility, and could cause severe over-fitting, and also may not generalise outside of the test set; considering an image in the test set may be of the same date as a training sample, with similar coordinates, the network could learn this exactly and achieve a low loss. This wouldn't generalise as accumulations change over time. However in the future, this could be highly valuable. Instead of the date as a categorical feature, to communicate the cyclical nature of seasonal variation, the feature could be converted to days since year start — the range $(1, 365)$, and normalised by division of 365, and used then as a numeric float variable. The network can then learn seasonal representations. As the number of samples increase, with many images captured over a long time-frame, the network should then grow invariant to seasonal change.

- **Time of Day** — this is useful factor given it accounts for some proportion of lighting variations, weather being a factor also. With enough samples, incorporating this feature into the numeric portion of the model, the network could learn invariability to brightness.

- **Latitude** — the use of GPS has already been discussed, so it is appropriate only to add that, with enough samples, the network could associate certain image representations with particular areas, thus better detecting fluctuations such as the presence of new litter.

- **Longitude** — see above. Note that for both longitude and latitude, conversion to

decimal is used, with consistent method (e.g., of rounding to five decimal places), and normalised only as part of the model's pre-processing, either in the generator or by a data-frame prepare function, so as not to skew the information which will be dumped to csv.

- **Prediction** — Lastly, after the generation of the data structure with all the above for each image provided, the data-frame is exposed for prediction and the results appended. Predictions will be of classes either 0 to 4 or 0 to 5, depending on how much class 5 data is gathered; in the early stages, only 0 to 4 is recommended.

The data-frame is then sorted by forecasted litter intensity and saved to both excel and csv for Scrapbook to then analyse.

Producing infographics is another possibility. Maintaining a data structure with (ideally validated) accumulations from the last coastline pass will allow the generation of graphics which convey accumulations by area and other simple comparisons, but more importantly showing increases — this is especially valuable as the detection of *new* accumulations is the prime purpose of the analysis. It is likely that accumulations from earlier passes will remain, likely with some variation, but exacerbation of a region, by GPS and by sector, and the presentation of new accumulations , are the main points of interest.

Chapter 5

Conclusions and Recommendations

In general, the goal of implementing an accurate prototype model, as proof of concept that automation can overhaul the current method of analysis, was not achieved. With such a limited dataset, particularly with so few samples, despite the extensive measures taken, no methodology was able to produce a viable model; therefore an automated system cannot be implemented at this stage. However, at a later stage where more data is available, and perhaps more computing power to train deeper networks on the highest resolution data, a similar methodology to that proposed may be employed towards the end of automating these classifications. Multiple input models, as-well as extensive data augmentation, have been established as performance improving methods, and such it is recommended they be employed in future attempts towards the same ends as this dissertation. Additionally, the system is able to classify samples of class 0 with up to 0.9 accuracy, and class 1 with 0.4, a factor directly related to the number of samples, demonstrating that, in all likelihood, if more samples of the rare classes were gathered, such a system could be applied. Perhaps with 6000 samples per class, meaning periodic flybys by Scrapbook over a period which allows time for new accumulations to form, such a system may then be possible.

5.1 Reflections on Models Applied

In general, building an accurate system is out of reach with these approaches and the current data limitations, particularly the low sample sizes. Generalisable patterns cannot be learned for any class other than 0, which has 6000 samples — the connection is clear. With high resolution images like these, we require more images per class than, say, the 700-1000 often recommended for ImageNet, where objects comprise most of the 224×224 frame.

Higher resolution images, and hence input shape, meant higher accuracies, but also required more computational power, accordingly slowing training and necessitating architecture changes and very low batch sizes — the former limiting performance, and the latter possibly causing problems with imbalanced data. Perhaps to optimise this, future attempts at working with this data could apply progressive resizing, where early stages are trained on downsampled input images to learn the base structure, then, as time training continues, the resolution scaled up to coincide with the increasing level of abstraction for features learned by the network.

All networks, regardless of depth and complexity, had a tendency to quickly overfit the training data even with regularisation, data augmentation and a low learning rate. This reflects the dataset, with its significant class imbalance and sample insufficiency. Varying the structure aids with this, such as the use of batch normalisation and dropout, but these add only small gains, and further strain to computational power, making remaining below the hardware ceiling difficult. Ultimately, the optimisation is in creating the best network that will run — without crashing due to memory allocations over what's available. Even a slow training network is adequate as the solution is worth days, a week of time even if, while monitoring, it continues to improve in validation.

Architectures with greater depth and more filters performed slightly better in terms of generalisation than shallower ones, perhaps indicating a high level of abstraction is necessary for this problem domain. Average pooling proved more effective than 'Max,' and the 'He Normal' initialisation outperformed random; the ImageNet first

layer initialisation from ResNet outperforming both.

Transfer learning was limited in this domain due to the dissimilarity between available pre-trained models, typically trained on low resolution, object based datasets such as 'ImageNet,' and the present domain. The filters from the first convolutional and pooling layers performed similarly to the Xavier initialisation with an equivalent setup. This could, however, be due to the particular choice of model; in the future, alternate pre-trained models could be explored — perhaps inquiries could be made to publishers of scientific papers on high resolution aerial data.

Learning rate decay was essential in preventing late stage divergence of validation and training loss, at the stage where base representations of the images were learned but, likely related to the low number of samples, struggling to find generalisable filters and instead over-fitting.

Clearly the model struggled to train, most likely caused by the low sample sizes for classes 1 +, as evidenced by the stronger performance of class 0 in all models attempted.

Samples per sector seemed to be an insignificant factor as compared to samples per class; the true variation is throughout the data as a whole, not isolated by class.

The basic CNNs over-fit the data quite easily even with batch normalisation and dropout. A likely reason is the low sample size, not having enough data to learn a generalisable solution and hence learning too closely the representation of the training samples.

Multi-input models perform similarly to slightly better than the single input CNNs, their extra numeric features appearing to allow less complex dense structures prior to soft-max output, due perhaps to learning a simpler relationship between filters and GPS, sectors and so forth than the complex one learned in a three layer 8196 neuron max dense construct.

The multi-input model is more computationally efficient as the equivalently performing single layer CNNs, reaching similar performances without the need for as many neurons in the dense layers and filters in the convolutional layers. Adding numeric features to a CNN barely increases the number of parameters, and so in that regard is optimal. However, adding numeric features can also increase the tendency for over-

fitting, so caution should be taken.

Data augmentation improved results, aiding generalisability and improving end accuracy/loss. An operations list of the following was observed to be most effective: rotation, sheering, brightness, shifts and mirrors. Augmentation particularly improves performance on low sample classes, and so is especially useful for this problem domain.

5.2 Recommendations

If the current Scrapbook data gathering and volunteer analysis system is to continue while more images are gathered, by the same methodology as described and applied in subsection 3.1.1, more labelled data can be gathered, pre-processed and collated, which is useful in any case; thus demonstrating the concept of the application.

Otherwise, their methodology may change to better suit an application of deep learning on recommendations; for example, the following: adopt a rigid set of standards, for features recorded and notation, units and so forth; then we could perform multiple outputs models, not possible presently as only half the data has a categorial 'litter location' and 'litter type' with already too few samples. These would allow object detection, a multi-label classification system, and relative object isolation in the image, further easing analysis by Scrapbook in a final system through the provision of information additional information.

In any case, with the system proposed, an interface can provide a dashboard showing areas, and particular samples, estimated as containing accumulations and at what levels; then images can be accessed to assess the situation and proceed as their current system dictates — dispatching cleaning volunteers.

Given the primary issue throughout this project has been the low number of samples for classes other than 0 and perhaps 1, gathering more data, over many dates of collection, is a strong recommendation. With the huge amount of information contained in these high resolution images, it is likely that a substantially greater amount of data is required to train a model with high accuracy across classes, a model that becomes invariant to the many variations future data will come with; e.g., litter against a background that has never presented in the dataset thus far.

In Analysis, we've shown the number of samples per class is a stronger indicator of accuracy than the number of samples in a sector; indicating there is only so much the network can learn from the class 0 samples. More class 2+ samples are necessary, so that future models may under-sample classes 0 and 1 to an equality with adequate litter positive samples, while still having enough data to accurately train the model, simultaneously solving the class inequality and loss issue.

While high resolution images offered a greater accuracy, they also necessitate more computational power. To achieve the highest accuracy, it may be a worthy investment to utilise more GPUs, training in parallel — which is possible in keras and tensorflow. This would allow deeper, more complex architectures to be trained, learning higher level abstractions of the data and allowing higher batch sizes which may be beneficial in imbalanced datasets such as this; given the weights, our filters, are augmented at the end of each batch, with the huge variety in terrain, types of litter, and the still existing presence of non-representational images, it is perhaps more appropriate to augment based on the loss of a batch, likely to then learn a more generalisable filter representation with a random selection of images. Additionally, a future network may benefit from the creation of a new category, not likely in the litter accumulations feature but instead a categorical similar to 'image quality' in some columns of the spreadsheets: image relevant; the network could then predict this also, isolating these cases, which are a small minority, but then would be able to offer better predictions to these images which will still be passed to a future model when tasked with automatically generating predictions for an entire, unprocessed dataset — necessarily unprocessed given that is the proposed system's purpose. If such a network could effectively isolate these images, performance could improve. This is a good alternative to classifying them a 0, which could provide conflicting information for the network, of which enough exists already due to the huge varieties in class 0. Therefore it is recommended that volunteers filling future spreadsheets include a column such as this, or at-least unilaterally fill in the image quality column rather than sporadically as it stands.

A further recommendation in this regard is to adopt clearer standards of notation, preventing issues such as the conflicts in latitude and longitude form: Cartesian vs

decimal. Also, widespread recording of the litter's location could allow a multiple input or ensemble approach that would in effect perform semantic segmentation, producing a numeric estimation of litter location, while not being able to physically identify the feature on the image. This could be integrated with a semantic segmentation approach, but would require many, many labels.

Additionally, a system of file name generation for supplied images is recommended to prevent issues such as those discussed with the supplied dataset: duplicate image names. Alternately, the GPS correlation approach proposed and implemented could be employed as standard; but this may be unnecessary.

Semantic segmentation in general should be attempted if through some effort the images could be labelled as required. Given the huge amount of information in a 24 MegaPixel image, 24 million pixels, one label is not ideal for the network to learn; especially with so few samples. Following a semantic segmentation approach may be better as the network could be trained with external datasets for these features, such as the TrashNet repository: a dataset containing images of commonly found trash items. The network could then learn to recognise these features in the high resolution images, isolating the regions and classifying the images as positive, and expressing the location in an interactive dashboard for Scrapbook to analyse.

In future, when more data is gathered, a new model should be attempted (of high resolution input shape, with at-least 9 layers, data augmentation and multiple mixed type inputs); this model should utilise additional numeric features, not only those available in the spreadsheets, of which more are required to label new images, but also other automatically collectable attributes such as weather, in a time series or an average of a (debatable) period, and tidal formations, indicating when sea litter could wash up — a weighted ensemble combining the CNN, feed forward networks and tidal models could prove highly effective.

Data augmentation has been shown to substantially improve results. As the dataset grows in quantity in the future, the effect may be less marked; but it is still recommended, particularly for the rare classes. Simple types of augmentation such as rotations and mirrors represent well the kind of variation space future samples will originate.

Live augmentation over static is recommended due to the fact that the network is not likely to see the exact same image twice, meaning better generalisation as a result of improved robustness to the variation inherent in the images. Data augmentation is particularly useful in this case as simple augmentations such as rotation give new synthetic data within the space of likely future images.

Further to simple augmentation methods, Generative Adversarial Networks could be explored to create new images in general, but more valuably synthetic class 2-5 accumulation images, further to the end of solving the low samples and class imbalance issues; however a great deal of computational power is necessary to generate realistic synthetic images of such high resolution. Therefore further GPUs to train in parallel are necessary. If successful, though, a synthetic image could be effectively an automated superimposing of trash features onto class 0 images, creating a realistic representation of how litter could present on that image's coastline. The converse is also possible — remove trash from class 2+ images, but is perhaps less important, unless future surveys note the accumulations rarely change over time, in which case it would be useful to have these synthetic varieties in the dataset as we could then better detect if these coasts were cleaned, meaning they are now class 0s.

As the data grows, the expectation of distribution may be as follows, assuming similar proportions of litter present and a similar number of photographs are captured:

Therefore an estimate of $5-10$ more passes, which does not necessitate one per year — in-fact, were more passes taken at periods of say, per month, an accurate system could be developed then.

Finally, on following the suggestions above, if an accurate system is created, the model should then be wrapped in the cloud or a remote server, continually retraining on data gathered at specified intervals, towards the end of maintaining generalisability and staying up to date — particularly relevant given litter can present at new locations which the network could struggle to learn if not exposed to positive predictions before in these terrains; the model should then be connected to a desktop application with a user friendly interface, as Scrapbook have to technical knowledge, with which they will open folders of images for automatic processing. EXIF information would be automatically

scraped, as-well as weather and tidal predictions tied to sectors via GPS, a numeric dataset then auto-created, the features of which all automatically gathered. A loading bar with a time estimate then would present before a dashboard indicates the findings: areas with litter *that were litter free last survey date* shown first, as a location on a map, sector and prediction, then clickable to view the image itself and perhaps an isolation of the feature. Optionally, the user could then navigate to a validation/invalidation interface where a portion of predictions will be assessed, particularly high accumulation classes, then training the model further with these new image-label pairs. Over time, the system will improve in accuracy such that it may replace current methods.

Appendix A

Appendix

A.0.1 Packages

IDE: PyCharm Professional 18.2; Python version: 3.7.1

IDE used to write code on personal machines which was then uploaded to Github, cloned on the server, and executed remotely via SSH; in terminal. As new files were written and others updated, Git pulls were used to update the files remotely.

- Pandas: used for processing the excel spreadsheets and csv files, and additionally for the data-frame structure in data generation.
- Numpy: used for mathematical operations such as 'ceiling' in calculating number of batches required to iterate through all available data once (per epoch.) Also used to convert data to arrays before 'yielding' with the data generator.
- Tensorflow: a machine learning library, we use tensorflow as the backend for Keras (described below) to utilise the GPU for 'graphical calculations;' additionally use TensorBoard for model structure and performance visualisation
- Keras: a deep learning library available as part of the Tensorflow package and independently; includes pre-defined convolutional, pooling and dense layers, as-well as support for multi-input type models via the 'functional' API. Additionally provides support for creation of data generators to provide batch sized increments of input arrays, allowing us to stay beneath the hardware ceiling. Provides function-

ality to simultaneously generate batches with the CPU while training is executed on the GPU via the 'train_on_batch' and 'fit_generator' approaches. We use both the 'sequential' and 'functional' APIs dependent on the model.

- sklearn: a library with many machine learning tools and metrics to evaluate sets of predictions
- PIL: an imaging library used to process, save, compress, resize and augment images; contains the library Image and ImageEnhance, the latter used for example in brightness enhancement while the former is the basic functionality (e.g., save images.)
- os: used for searching directories, enabling us to strip information from spreadsheets and collate many image folders. Primarily used 'list_dir' to list all subdirectories and files in a directory.
- random: used for shuffling lists
- datetime: used for saving log files; attaches the date of execution.

A.1 Data Processing and Creation

The following was written to describe the time consuming process of building a dataset from the spreadsheets and images, initially loosely associated; it is not necessary to read in entirety. Some was used in the main body to describe the overall methodology.

The supplied spreadsheets have many inconsistencies in manner of value recording for many features, and variety of columns included also. For example, many sheets have no Sector column, others no Survey column, which holds the date of collection, and some have categorical Litter_Present(Y/N). Extensive data preparation was applied to make best use of all supplied.

Of the features, Litter_Intensity is most appropriate for use as our label. A single label, a numeric category from 0 to 5, for an entire image may be insufficient information for the network to learn. However, we make the best use of the information available semantic segmentation would be a far too costly, in terms of time, approach: the

column Litter_Location indicates the location of accumulations, typically holding values such as backshore, foreshore, and more rarely, literally everywhere; were it possible to accurately identify the foreshore and the backshore in future unlabelled images, an approach utilising this information could be useful. However, this would require an external training set/model, such as edge set detection semantic segmentation networks; something which could be explored in further work. Given we wish to automate the en-masse processing of images, this is a column of little use to us in terms of model building, as we cannot provide a label of this value automatically for future images, but may be of interest in understanding the problem. However, were it to have more samples, to have been recorded in all data-frames, it could be used as a second output, in effect a form of semantic segmentation.

Image_Name holds the filename of the image a volunteer has examined. Note there are duplicates even inside of single spreadsheets in some cases; while in the image folders, exact duplicate filenames cannot exist.

Image_Quality as a column is of particular utility in filtering out unusable images, of which there are a great many photographs of towns and villages, other random things which are not coastlines.

Only around a third of the images have a recorded quality, but we can use this to automatically sift those designated unusable. Many of these are those unrelated, non-coastal images mentioned, while others are exceptionally blurry or otherwise degraded, therefore the column is of use. As with most features, a variety of notations were used by the volunteers, such as the use of N, n, Not Usable for unusable images the following function catches these varieties and cleans the column (when applied to the dataframe.)

```
def fix_quality(qual):
    """
    Categorises 'quality' input to one of 4 values, or nan.
    :param qual:
    :return:
    """
    try:
```

```
        out = str(qual)

        if str.lower(qual[0]) == 'h':
            return 'h'
        if str.lower(qual[0]) == 'm':
            return 'm'
        if str.lower(qual[0]) == 'l':
            return 'l'
        if str.lower(qual[0]) == 'n':
            return 'n'
        else:
            return np.nan
    except TypeError:
        return np.nan
```

Note that entries which do not ascribe to any of the desired forms, commonly in the form of a textual comment where a volunteer was unsure, are converted to nan, to give us a clear picture of how many values are missing, and allow conversion to an appropriate replacement when utilising the column for analysis. Whether the images should be designated as medium, high or low is largely irrelevant, as it is the unusable ones of interest. Therefore we fill these entries with 'm', the medium value, which is not the mean or mode — high.

The function works by detecting the first letter of the string, in lower case to catch case differences, and designating the associated value. Images of low quality, on a cursory examination, tend to be blurred or from an usually far distance, and so we keep these given the random noise they represent, thus improving the models robustness to variation.

It is a worthy consideration that while these images are likely to appear in future data which is processed by the model, and the model may not predict well on these images. But adding another category of classification, for example, 'irrelevant' is perhaps appropriate given so many images were not classified in this regard. The proposed category could be introduced in one of two ways: as an extra feature of the accumula-

tion feature, or as a new feature entirely; the latter is more appropriate, particularly if employing a loss function which suggests non-independence of classes, expressing the sequentially increasing levels from 0 to 5. This approach is similar to regression in a sense. Utilising regression, where a float could be returned as an output, could indicate a confidence between two classes. The flaw in this, however, is that some higher accumulation classifications do not share great commonalities with lower ones, for example the presence of widespread shipwreck debris, distinct from the characteristics of moderate, $1-2$ classes, often with small items such as bottles and plastic bags strewn throughout the coastline.

However, the assumption could be made that those not classified be designated as relevant. This is spurious on examination, but with the limited information, may still improve prediction on these images.

Other columns which required cleaning are the latitude and longitude, which contain some entries using the Cartesian coordinate system, while the dominant convention is decimal (a negative for a westward longitude, for example, which all of Scotland falls under.) The following function, when applied to the dataframe, corrects these entries (note that ° replaces the "\degree" in the actual code.

```
def fix_lat_long(value):
    """ checks for degrees and converts if so; also rounds to 5 decimal
        places"""

    value_original = str(value)

    if 'N' in value_original:

        low = value.split('\degree')[0].split('N')[0]

        value = value.split('\degree')[1].split('N')[0]
        value = filter(lambda x: x in "0123456789.", value)
        value = ''.join(value)

        value = float(low) + (float(value) / 60)
```

```
    if 'W' in value_original:

        low = value.split('\degree')[0].split('N')[0]

        value = value.split('\degree')[1].split('N')[0]
        value = filter(lambda x: x in "0123456789.", value)
        value = ''.join(value)

        value = -(float(low) + (float(value) / 60))

    else:
        try:

            if float(value_original) < 0:

                value = -round(-float(value_original), 5)

            else:

                value = round(float(value_original), 5)

        except ValueError:
            return np.nan

    return value
```

This works by detecting the N and W present in a Cartesian coordinate systems. Division by 60 converts to decimal. The end value is rounded to 5 decimal places for consistency. Note the bottom section if statement detecting a negative value (the case with longitude), this ensures consistency in rounding as the round function rounds up, which in the case of a negative number rounds, well, still up but the opposite of if it were a positive number (the program they've used to extract the coordinates follows

this convention.)

Towards automation, this code can be used to fill future spreadsheets or data structures, ensuring a consistent form and simpler input than the previous extraction, requiring manual input of information.

The dataframe is created by iterating through all spreadsheets, appending each sheet's relevant columns to the main dataframe.

```
for sheet in spreadsheets:

    df = pd.read_excel(sheet, sheet_name='SCRAPbook_image_assessment_form')

    if 'Sector' not in df.columns: # sector not in some sheets

        sect = get_sector(sheet) # get from excel filename if so, reasonable
            alternative
```

If the spreadsheet does not contain a Sector column, we use the next best alternative: auto detecting it by reading the name of the spreadsheet via the following functions:

```
def get_number(num):
    """
    gets number from mixed char input, returns nan for any unwanted input
    :param num:
    :return:
    """
    try:
        num = str(num)
    except TypeError:
        return np.nan

    name = filter(lambda x: x in '0123456789', num) # filters all not in the
        string
    name = ''.join(name) # converts generator to string
    try:
```

```
        return int(name)
    except ValueError or TypeError:
        return np.nan

def get_sector(sheet):

    sect = get_number(sheet.split('/')[-1])

    if np.isnan(sect) is True:
        return np.nan
    else:
        sect = str(sect)[0:2]
        return sect
```

Cases where this approach is problematic are, for example, the spreadsheet titled 57-60, which we detect as 57. A possible solution is to detect sector by GPS, however this is vulnerable to changes in flightpath. Therefore this approach is the best alternative in cases where latitude and longitude are unavailable.

With the creation of unique IDs for all images, we have the ability to detect exact duplicates. Many of these exist in the spreadsheets cases where perhaps sectors overlap, and so an image is recorded in several sheets. This reduces the total number of entries from 14 thousand to ten thousand, in addition to the removal of unusable images, showing the significant redundancy. Outside of this, we also drop columns with no litter intensity label, as without a label we cant use these rows for training also considering the cases where a litter intensity value is not recorded due to the image being unsuitable for analysis, catching some of the values which do not have an Image_Quality value but are unusable. We also clean other columns via this function, which unifies data type (e.g., float/integer) for columns:

```
def column_cleaner(df, col, type_data='integer'):

    df[col] = pd.to_numeric(df[col], errors='coerce', downcast=type_data)
    df = df[df[col].notna()]
```

```
    if type_data is 'integer':
        df[col] = df[col].astype(int)
        return df, df[col]
    else:
        df[col] = df[col].astype(float)
        return df, df[col]
```

The end dataframe, with path added, has the following columns:

```
Index(['index', 'Image_Name', 'Sector', 'Lat', 'Long', 'Image_Quality',
       'Litter_Intensity', 'Litter_Type', 'id', 'path'],
      dtype='object')
```

Considering the filtered images, using these to pre-train a network via unsupervised learning so that it learns representations of the images could be explored, but given many of these do not conform to the usual image structure, and most of these were likely unlabelled for a reason, it is risky to do so, perhaps leading the network away from a better initialisation.

A.1.1 Collating Spreadsheets

In all, the spreadsheets concatenate to form a dataframe with 14755 rows; however, many are duplicates, likely due to the overlap among 'Sectors;' a geographical boundary which segments the coastlines into 93 areas, 'tracking anti-clockwise around Scotland from 1,' Sophie stated.

Below we illustrate the variation in features volunteers have recorded values for:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14755 entries, 0 to 14754
Data columns (total 7 columns):
Image_Name        14075 non-null object
Image_Quality     11582 non-null object
Lat               13993 non-null object
Litter_Intensity  11650 non-null object
Litter_Type       5567 non-null object
```

```
Long              13992 non-null object
sect              14755 non-null object
dtypes: object(7)
memory usage: 807.0+ KB
```

Note that many 'sector' values were automatically assigned by reading/parsing the '.xlsx' spreadsheet filenames for the numeric contents, which could be thought of as a case of 'weak' labelling — there are inconsistencies. For this reason, later we use sector as an identifier *if and only if* there are no latitude and longitude values for cross-reference; meaning, only for those rows where no GPS info was inserted, and for images which do not contain any EXIF GPS information (of which there are a subset of several hundred images, probably captured via a camera either without the capability or with the function disabled, or lost in some sort of pre-processing); we observe these images begin typically with 'IMG_' rather than 'DSC_'.

Image Quality is of great utility for us, bearing in mind the presence of 'stray' images in the data which do not represent the problem domain: e.g., images of villages (not coastlines), land artefacts. With this column, we can auto-sift the images from the data-frame and accordingly not compress and re-save these images prior to sever upload and model execution. We bear in mind the flaw, however, that future images will not be able to auto-detect poor quality; given SCRAPbook are moving towards automation photography, this issue may solve itself: no more photographers capturing images of interesting things, or their homes!

For later cross-reference, we add a column: 'origin;' this holds the sheet from which all labels and other information was taken, allowing examination in cases where perhaps two images are associated with a label; in most cases this is the result of duplicate images, while in others it is caused by the small 'sector + img_name' ID images, 23 samples which could not be uniquely identified as they lacked EXIF data; and in some others, cases where no images were found — the origin sheet then manually checked to observe this.

Most images in the dataset contain little to negligible litter accumulations, an inequality present in many real world machine learning applications. Following full data

preparation and cleansing we have the following distribution:

```
Out[48]:
0    6302
1    2217
2     684
3     600
4     206
5      81
Name: Litter_Intensity, dtype: int64
df.shape
Out[47]: (10090, 8)
```

Note how few samples there are of class 5; if the network fails to discriminate these well we may amalgamate the class 4 and 5 SCRAPbook have indicated that a forecasting of the exact categories, while desirable, could be satisfactorily replaced with a model that returns instead those greater or equal to a class 2 situation, and those predicted to be a class 0 or 1: a binary system. Accordingly, a combination of 4 and 5 to mean most serious accumulations beforehand would be acceptable, in hopes this provides an adequate solution. Following completion of the first networks training phase, the results of the classification matrix will illuminate the issue. If, for example, the model most commonly misclassifies a class 5 as a class 4, or even 3, this is less problematic than to mistake a class 5 for a class 0. An approach which takes into account what could be abstractly seen as a nesting of classes is sought; a class five is a class 2,3,4 but to a higher degree: a greater scale. In a sense, the problem is similar to a regression task. Perhaps then a custom loss function, based on categorical cross-entropy but which exponentially (or conversely logarithmically), or some other way, assigns a weight misclassifications of contrast (0 for 5) as being of higher loss than a 2 for 3.

Training the network to improve the accuracy, for example, is then problematic as the model could achieve over 60% accuracy simply by predicting 0 for every image. It is appropriate then to consider other metrics by which we should train the network and monitor the progress of the model.

A.1.2 Unique ID System

Matching an image with the corresponding label is the utmost priority at this stage. A single section with incorrect labels, for example due to volunteer error or inconsistency in input of sector feature in the spreadsheets, provides conflicting information for the network and thus damages ability to learn the inherent patterns in the data. Creating a new ID to address the faults of the prior approach, we utilise latitude and longitude to uniquely identify each image, with the added bonus of identifying duplicate cases (where the same image is labelled on multiple spreadsheets, often with subtle variation preventing a duplicate identification); this approach is the most accurate way of identifying an image due to inconsistencies in recorded sector by volunteers and duplicate image filenames.

Given IDs have already been assigned for each row in the dataframe, comprised of the information from the spreadsheets, we now create IDs for every image supplied to the end of cross referencing. The general approach is to first search the directory containing the images for each subdirectory, detect the sector through stripping numeric information from the folder name, then search these subdirectories for images, open these images using Image package from PIL (Pillow), and using ExifTags from PIL we open the images to gather the necessary GPS information. To do this, first a function is made to list all full paths to images in the given directory which contains all images.

```
def get_paths():

    img_dir = '/media/ulrich/Seagate Expansion
        Drive/scrapbookimages2018/2018_SCRAPbook_Images/'

    paths = []

    for path, dirs, files in os.walk(img_dir, topdown=True):

        paths.append(path)

    return paths[1:]
```

```
def get_file_paths():
    paths = get_paths()

    file_paths = []
    # file_list = []

    for path in paths: # iterate over all paths, including subdirectories

        for item in os.listdir(path): # only interested in files

            if os.path.isfile(os.path.join(path, item)) and
                (item.endswith('.JPG') or item.endswith('.JPEG')):
                file_paths.append(str(path) + '/' + str(item))

    return file_paths
```

Then we create a function to open an image from its path and extract the latitude and longitude, if available.

```
def open_image_get_lat_long(image_path):

    img = Image.open(image_path)

    exif = {ExifTags.TAGS[k]: v for k, v in img._getexif().items() if k in
        ExifTags.TAGS}

    try:

        info = exif['GPSInfo']

    except KeyError:
        lat, long = '', ''
        return lat, long
```

```
    try:

        lat = info[2][0][0] + ((info[2][1][0] / info[2][1][1]) / 60)

        long = info[4][0][0] + ((info[4][1][0] / info[4][1][1]) / 60)

        lat, long = round(lat, 5), -round(long, 5)
    except KeyError:
        lat, long = '', ''

    return lat, long
```

The above code extracts the information from the EXIF dictionary returned from the ExifTags object, the indexes referring to the numeric values held in the dictionary, which includes values such as N for north and W for west (note this makes longitude negative) and converting from the cartesian coordinate system to decimal by division with 60. Note that we round the negative value of longitude as the behaviour of the Python round function always rounds up, and so would essentially round down our negative number if we rounded negative longitude instead of rounding positive longitude and taking the negative of this.

Then IDs can be created with this information via the following function:

```
def create_id_lat_long(image_path):

    folder = image_path.split('/')[-2]

    file_name = image_path.split('/')[-1]

    try:
        sect = str(get_number(folder))[0:2]
    except IndexError or TypeError or ValueError:
        sect = '0'
```

```
    if np.isnan(int(sect)) is True:
        return np.nan

    else:

        lat, long = open_image_get_lat_long(image_path)

    if lat != '' and long != '':

        id = str(sect) + '~' + str(file_name) + '~' + str(lat) + '~' +
            str(long)

    else:
        id = str(sect) + '~' + str(file_name)

    return id
```

Note that we create an id with sector and image filename if no gps information is available from an image; and if we cant detect sector from the folder name (a case of this is the folder supplied named JPG, containing images of unclear origin however if these images contain gps info, we will be able to identify them, another positive to this new extensive approach to image/label association.) Following this, we simply iterate through the file paths to assign an ID to each path.

```
def assign_ids():

    file_paths = get_file_paths()

    ids = []
    # n = 0
    for path in file_paths:
        # n += 1
        # print(path, n)
```

```
        id = create_id_lat_long(path)

        ids.append(id)

    return file_paths, ids
```

The output is then two lists in the desired order.

There are two cases in which latitude and longitude are not available for cross reference between the excel spreadsheets and the images themselves: volunteer failed to input available GPS information to spreadsheets, according ID does not incorporate that information, and image itself has no GPS information different cameras were used to take photographs on different occasions, learned via exploring EXIF data, and some have GPS disabled or possibly no GPS ability.

To match an image with the correct label we then have to account for those eventualities: The necessary code must then detect the type of ID it is cross referencing to be one of two: uses lat and long, or uses only sector and image name, the best alternative. Initially, a simple approach was attempted: to check if the full IDs match, excluding sector to catch those cases where sector is incorrectly input to spreadsheets or was not entered in spreadsheets and so we use the alternative: detecting sector by reading the spreadsheet filenames, which carries some uncertainty (for example in the case of spreadsheet 57-60).

This approach originally failed to identify an image path (the function of the ID, cross referencing) for around 20% of the data. On examination, the cause in most cases appeared to be small incremental differences in latitude and longitude caused by differences in behaviour of the Python round function, which was used to round each latitude and longitude value for both IDs to five decimal places for consistency image set and spreadsheet set and also in behaviour of conversion from cartesian coordinate system to decimal; SCRAPbook informed in the interview that freeware was used in most cases to strip the GPS information from images and input to the spreadsheets, however some sheets used the cartesian system, which we converted for consistency

in the spreadsheet dataset creation. To account for this, a delta parameter is created which is the absolute difference of the latitude and longitude. Care must be taken to ensure a delta parameter threshold too large is not selected as differences in coordinates can be minute. To address this, instead of breaking the loop (which iterates over the full list/series of IDs) when a match is found, the loop continues and adds 1 to a new variable n_matches that is first initialised at 0. A warning is then given if multiple image paths are found for a pair of dataframe-image set IDs. We can then alter the delta threshold until there is only one *unique* match for each image, and identify duplicates by examination.

In the end, we assign a path to each image in the dataframe, exported as csv, and our data generation process is no longer vulnerable to incorrect labels outside the case of volunteer error in assigning a label to that image, which is very possible as zooming the images is required to satisfactorily examine for the presence of litter.

A.1.3 Matching IDs and Creating Compressed Datasets

Given IDs have already been assigned for each row in the dataframe, comprised of the information from the spreadsheets, we now create IDs for every image supplied to the end of cross referencing. The general approach is to first search the directory containing the images for each subdirectory, detect the sector through stripping numeric information from the folder name, then search these subdirectories for images, open these images using Image package from PIL (Pillow), and using ExifTags from PIL we open the images to gather the necessary GPS information. To do this, first a function is made to list all full paths to images in the given directory which contains all images.

```
def get_paths():

    img_dir = '/media/ulrich/Seagate Expansion
        Drive/scrapbookimages2018/2018_SCRAPbook_Images/'

    paths = []

    for path, dirs, files in os.walk(img_dir, topdown=True):
```

```
        paths.append(path)

    return paths[1:]

def get_file_paths():
    paths = get_paths()

    file_paths = []
    # file_list = []

    for path in paths: # iterate over all paths, including subdirectories

        for item in os.listdir(path): # only interested in files

            if os.path.isfile(os.path.join(path, item)) and
                (item.endswith('.JPG') or item.endswith('.JPEG')):
                file_paths.append(str(path) + '/' + str(item))

    return file_paths
```

Then we create a function open an image path and extract the latitude and longitude, if available.

```
def open_image_get_lat_long(image_path):

    img = Image.open(image_path)

    exif = {ExifTags.TAGS[k]: v for k, v in img._getexif().items() if k in
        ExifTags.TAGS}

    try:
```

```
        info = exif['GPSInfo']

    except KeyError:
        lat, long = '', ''
        return lat, long

    try:

        lat = info[2][0][0] + ((info[2][1][0] / info[2][1][1]) / 60)

        long = info[4][0][0] + ((info[4][1][0] / info[4][1][1]) / 60)

        lat, long = round(lat, 5), -round(long, 5)
    except KeyError:
        lat, long = '', ''

    return lat, long
```

The above code extracts the information from the EXIF dictionary returned from the ExifTags object, the indexes referring to the numeric values held in the dictionary, which includes values such as N for north and W for west (note this makes longitude negative) and converting from the cartesian coordinate system to decimal by division with 60. Note that we round the negative value of longitude as the behaviour of the Python round function always rounds up, and so would essentially round down our negative number if we rounded negative longitude instead of rounding positive longitude and taking the negative of this.

Considering the point of the potential future application would be to expose for prediction images which have never been processed, functions including the above will instead be used to automatically gather this information. A great deal of information can be stripped from EXIF and automatically passed to excel spreadsheets or csv files, which Scrapbook or another interested party can then view, in effect an auto-generated report.

Towards correlating images and labels, though, IDs can be created for images with this information via the following function:

```
def create_id_lat_long(image_path):

    folder = image_path.split('/')[-2]

    file_name = image_path.split('/')[-1]

    try:
        sect = str(get_number(folder))[0:2]
    except IndexError or TypeError or ValueError:
        sect = '0'

    if np.isnan(int(sect)) is True:
        return np.nan

    else:

        lat, long = open_image_get_lat_long(image_path)

    if lat != '' and long != '':

        id = str(sect) + '~' + str(file_name) + '~' + str(lat) + '~' +
            str(long)

    else:
        id = str(sect) + '~' + str(file_name)

    return id
```

Note that we create an id with sector and image filename if no gps information is available from an image; and if we cant detect sector from the folder name (a case of this is the folder supplied named JPG, containing images of unclear origin, later

revealed as from sectors $20 - 23$ however if these images contain gps info, we will be able to identify them, another positive to this new extensive approach to image/label association.) Following this, we simply iterate through the file paths to assign an ID to each path.

```
def assign_ids():

    file_paths = get_file_paths()

    ids = []
    # n = 0
    for path in file_paths:
        # n += 1
        # print(path, n)

        id = create_id_lat_long(path)

        ids.append(id)

    return file_paths, ids
```

The output is then two lists in the desired order, which we use to append file paths to the data-frame; thus having exact matches.

When adding paths to the images, accurately associating a label with an image, a significant amount of images are lost; however, the cause primarily is the images dont exist in the supplied dataset. For example, a large number of images, over one thousand four hundred, from sector 23 are described in the associated spreadsheet, but only four hundred exist in the supplied folder. Searching by latitude and longitude where available and excluding sector in pairing of labels with images solves the issue of images being labelled in the wrong sector spreadsheet or included in the wrong sector folder, and allow us to definitively say an image does not exist (in the set supplied, images either in SCRAPbook possession or failed to transfer properly.) The only situation in which we cannot positively identify a pairing is when latitude/longitude is not available in the

spreadsheets or the image EXIF; these cases however make up only a small minority in the null path subset of the dataframe, which has 1414 rows, as opposed to the prior ID's 2400.

The subset of null images for sector 57, with the last ID approach over 1000 strong, notably had the convention of not including the file extension, .JPG or .JPEG, in the Image_Name column. So we augment the dataframe to automatically add this extension if neither one is present, and use this information together with GPS, ignoring sector, to identify; this reduced the number of null paths by over 700.

```
null.Sector.value_counts().iloc[0:5]
Out[14]:
22.0   279
3.0    164
24.0   160
7.0    145
2.0    128
Name: Sector, dtype: int64
```

Unidentifiable images are now spread out through the data, and are unidentifiable as no label with their GPS, or name + sector, exist in the data-frame. Quite possibly, they weren't supplied, or were lost in data transfer. Particularly, several sectors surrounding sector 23 are missing many images — these are referred to in a spreadsheet, but are not present in the supplied images.

For comparison of these approaches, the following table compares the samples returned by using the ID type 'Sector' + 'Image_Name' in column 1 with the stricter EXIF approach described above.

Clearly a substantial improvement over the last system, and with the added benefit of identifying duplicates and preventing, more adequately, any incorrect labelling.

class	Samples (original ID)	Samples (EXIF ID)
0	5048	6201
1	1915	2320
2	630	762
3	549	645
4	197	239
5	70	72

Table A.1: Samples in Both ID Approaches

A.2 Generators

A.2.1 Expansive Augmentation Generator

```
def expansive_aug_gen(df, batch_size, mode='train', weights=None,
    total_epochs=20):

    df_main = df

    n_augs = round(0.8 * batch_size)
    n = 0 # init n at 0
    n_epochs = 0

    while True: # eternal loop

        augs = df.sample(n=n_augs).reset_index(drop=True)

        images = [] # a list for image arrays
        batch_df = pd.DataFrame(columns=['Sector', 'Lat', 'Long']) # the 3
            numeric features
        labels = [] # a list for labels

        if mode == 'train': # only augment if training
            for n_b in augs.index: # putting augmented images at the start of
                the batch so generalises better
                try:
                    img = aug_image(augs.loc[n_b, 'image_path'])
```

```
            except IOError:
                continue

                images.append(img)
                batch_df = batch_df.append(augs[['Sector', 'Lat',
                    'Long']].iloc[n_b])
                labels.append(augs.loc[n_b, 'Litter_Intensity'])

        while len(images) < batch_size: # loops until batch created

            if n == df.shape[0]: # reach end of list; take new subsample of
                train set and reset n, end batch

                if mode == 'train': # only under-sample if we are training; new
                    random subset
                    df = undersample_df(df_main) # under-samples the training
                        set

                n_epochs += 1
                n_augs = round(n_augs - (n_augs/total_epochs))
                n = 0 # reset n
                break # last batch likely to not have batch size samples

            try:
                img = prep_image(df.loc[n, 'image_path']) # in-case image can't
                    be loaded for any reason
            except IOError: # missing files
                n += 1
                continue

            # Now we append the relevant data to fill the batch

            images.append(img) # append array to list
```

```
            batch_df.append(df[['Sector', 'Lat', 'Long']].iloc[n]) # append 3
                numeric features
            labels.append(df.loc[n, 'Litter_Intensity']) # appends nth label
            n += 1 # index goes up each time we add to our batch lists

        # now the batch is created.

        labels = tf.keras.utils.to_categorical(labels, num_classes=classes) #
            to one hot encoded

        if weights is None:

            yield ([np.array(images, dtype=np.float16), np.array(batch_df,
                dtype=np.float16)],
                  np.array(labels, dtype=np.uint8))

        else:

            yield ([np.array(images, dtype=np.float16), np.array(batch_df,
                dtype=np.float16)],
                  np.array(labels, dtype=np.uint8), weights)
```

A.3 Model Scripts

A.3.1 Variable CNN

```
def cnn(input_shape=(800, 1200, 3), pooling=AveragePooling2D,
    batch_norm=(True, True),
              filters=(32, 64, 128, 160, 196, 256, 352, 712),
              dense_neurons=(8196, 2048, 1024), dropout=0, stride=2,
                  classes=5):
    """
```

```
    To alter the number of layers, simply pass a different length tuple to
        filters and dense neurons. Also takes
    pooling argument.
    """
    input_layer = layer = Input(shape=input_shape)

    for n_filters in filters: # add convolutional and pooling layers with
        batch norm before pooling

        layer = Conv2D(filters=n_filters, kernel_initializer='he_normal',
            kernel_size=[3, 3], padding='same',
                      activation=tf.nn.relu)(layer)

        if batch_norm[0] is True:
            layer = BatchNormalization()(layer)
        layer = pooling(pool_size=[stride, stride], strides=2)(layer)

    layer = Flatten()(layer) # flatten the pooling output

    if dropout != 0:
        layer = Dropout(dropout)(layer) # dropout before dense layers

    for neurons in dense_neurons: # add the dense layers, optional batch norm

        layer = Dense(neurons, activation=tf.nn.relu,
            kernel_initializer='he_normal')(layer)
        if batch_norm[0] is True:
            layer = BatchNormalization()(layer)

    softmax_output = Dense(classes, activation='softmax',
        kernel_initializer='he_normal')(layer)

    model = Model(inputs=input_layer, outputs=softmax_output)
```

```
    print(model.summary())

    return model
```

A.3.2 Multi-Input

```
def multi_input(input_shape=(800, 1200, 3), pooling=AveragePooling2D,
    multi_features=3, batch_norm=(True, True),
                filters=(16, 32, 64, 128, 160, 196, 256, 352),
                    dense_neurons=(8196, 2048, 1024), dropout=0,
                classes=5):

    image_input = layer = Input(shape=input_shape)
    numeric_input = Input(shape=(multi_features, ))

    for n_filters in filters:

        layer = Conv2D(filters=n_filters, kernel_initializer='he_normal',
            kernel_size=[3, 3], padding='same',
                      activation=tf.nn.relu)(layer)
        layer = pooling(pool_size=[2, 2], strides=2)(layer)
        if batch_norm[0] is True:
            layer = BatchNormalization()(layer)

    flat = Flatten()(layer)

    layer = concatenate([numeric_input, flat])

    if dropout != 0:
        layer = Dropout(dropout)(layer)

    for neurons in dense_neurons:
```

```
        layer = Dense(neurons, activation=tf.nn.relu,
            kernel_initializer='he_normal')(layer)
        if batch_norm[1] is True:
            layer = BatchNormalization()(layer)
    softmax_output = Dense(classes, activation='softmax',
        kernel_initializer='he_normal')(layer)

    model = Model(inputs=[image_input, numeric_input], outputs=softmax_output)

    print(model.summary())

    # plot_model(model, to_file='mutli_input_model.png')

    return model
```

A.3.3 VGG Pre-Trained Model

```
def vgg16(input_shape):

    vgg_model = VGG16(include_top=False, classes=classes, weights='imagenet',
        pooling='avg', input_shape=input_shape)

    layer = Flatten()(vgg_model.output)
    layer = BatchNormalization()(layer)

    layer = Dense(2048, activation=tf.keras.activations.relu,
        kernel_initializer='he_normal')(layer)
    layer = BatchNormalization()(layer)
```

```
    layer = Dense(1024, activation=tf.keras.activations.relu,
        kernel_initializer='he_normal')(layer)
    layer = BatchNormalization()(layer)
    output = Dense(classes, activation=tf.keras.activations.softmax,
        kernel_initializer='he_normal')(layer)

    model = Model(inputs=vgg_model.input, outputs=output)

    model.summary()

    return model
```

A.3.4 Full ResNet

```
def resnet(input_shape=(800, 1200, 3)):
    from tensorflow.python.keras.applications.resnet50 import ResNet50

    model = Sequential()

    model = ResNet50(include_top=False, classes=classes, weights='imagenet',
        pooling='avg', input_shape=input_shape)

    model.add(Flatten())
    model.add(BatchNormalization)

    model.add(Dense(128, activation=tf.keras.activations.relu,
        kernel_initializer='he_normal'))
    model.add(BatchNormalization())

    # dense_2 = Dense(1024, activation=tf.keras.activations.relu,
        kernel_initializer='he_normal')(bn1)
    # bn2 = BatchNormalization()(dense_2)
```

```
    model.add(Dense(classes, activation=tf.keras.activations.softmax,
        kernel_initializer='he_normal'))

    model.summary()

    return model
```

A.3.5 First Layers Transfer Learn for ResNet

```
def first_layers_resnet(input_shape=(800, 1200, 3), pooling=AveragePooling2D,
    batch_norm=(True, True),
              filters=(32, 64, 128, 160, 196, 256), dense_neurons=(2048,
                  1024), dropout=0):

    res_model = ResNet50(include_top=False, weights='imagenet',
        pooling='avg', input_shape=input_shape, classes=classes)

    input_layer = layer = res_model.input

    for resnet_layer in res_model.layers[1:7]: # adding res net layers (1st
        conv pool and actvns etc)
        layer = resnet_layer(layer)

    # del res_model

    for n_filters in filters: # adding own layers
        layer = Conv2D(filters=n_filters, kernel_initializer='he_normal',
            kernel_size=[3, 3], padding='same',
                      activation=tf.nn.relu)(layer)
        if batch_norm[0] is True:
            layer = BatchNormalization()(layer)
        layer = pooling(pool_size=[2, 2], strides=2)(layer)
```

```
    layer = Flatten()(layer)

    for neurons in dense_neurons:
        layer = Dense(neurons, activation=tf.nn.relu,
            kernel_initializer='he_normal')(layer)
        if batch_norm[1] is True:
            layer = BatchNormalization()(layer)

    if dropout != 0:
        layer = Dropout(dropout)(layer)

    output = Dense(classes, activation=tf.keras.activations.softmax,
        kernel_initializer='he_normal')(layer)

    model = Model(inputs=input_layer, outputs=output)

    for layer in model.layers[0:7]:
        layer.trainable = False

    model.summary()

    # plot_model(model, to_file='transfer_learning_network.png')

    return model
```

A.4 Model Plots and Summaries

A.4.1 Binary Script

```
Layer (type)                 Output Shape              Param #
=================================================================
```

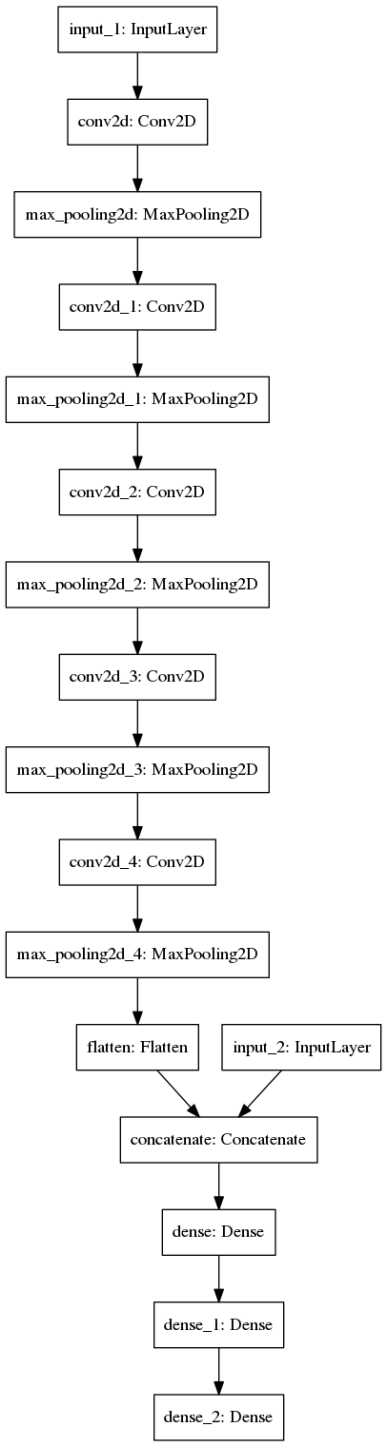


Figure A.1: Multi-Input Model

```
input_1 (InputLayer)      [(None, 800, 1200, 3)] 0
_________________________________________________________________
conv2d (Conv2D)           (None, 800, 1200, 16)  448
_________________________________________________________________
batch_normalization (BatchNo (None, 800, 1200, 16) 64
_________________________________________________________________
average_pooling2d (AveragePo (None, 400, 600, 16) 0
_________________________________________________________________
conv2d_1 (Conv2D)         (None, 400, 600, 32)   4640
_________________________________________________________________
batch_normalization_1 (Batch (None, 400, 600, 32) 128
_________________________________________________________________
average_pooling2d_1 (Average (None, 200, 300, 32) 0
_________________________________________________________________
conv2d_2 (Conv2D)         (None, 200, 300, 64)   18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 200, 300, 64) 256
_________________________________________________________________
average_pooling2d_2 (Average (None, 100, 150, 64) 0
_________________________________________________________________
conv2d_3 (Conv2D)         (None, 100, 150, 128)  73856
_________________________________________________________________
batch_normalization_3 (Batch (None, 100, 150, 128) 512
_________________________________________________________________
average_pooling2d_3 (Average (None, 50, 75, 128)  0
_________________________________________________________________
conv2d_4 (Conv2D)         (None, 50, 75, 128)    147584
_________________________________________________________________
batch_normalization_4 (Batch (None, 50, 75, 128)  512
_________________________________________________________________
average_pooling2d_4 (Average (None, 25, 37, 128)  0
_________________________________________________________________
conv2d_5 (Conv2D)         (None, 25, 37, 256)    295168
_________________________________________________________________
```

```
batch_normalization_5 (Batch (None, 25, 37, 256)  1024
_________________________________________________________________
average_pooling2d_5 (Average (None, 12, 18, 256)  0
_________________________________________________________________
conv2d_6 (Conv2D)         (None, 12, 18, 256)    590080
_________________________________________________________________
batch_normalization_6 (Batch (None, 12, 18, 256)  1024
_________________________________________________________________
average_pooling2d_6 (Average (None, 6, 9, 256)    0
_________________________________________________________________
conv2d_7 (Conv2D)         (None, 6, 9, 256)      590080
_________________________________________________________________
batch_normalization_7 (Batch (None, 6, 9, 256)    1024
_________________________________________________________________
average_pooling2d_7 (Average (None, 3, 4, 256)    0
_________________________________________________________________
flatten (Flatten)         (None, 3072)           0
_________________________________________________________________
dropout (Dropout)         (None, 3072)           0
_________________________________________________________________
dense (Dense)             (None, 4096)           12587008
_________________________________________________________________
batch_normalization_8 (Batch (None, 4096)         16384
_________________________________________________________________
dense_1 (Dense)           (None, 2048)           8390656
_________________________________________________________________
batch_normalization_9 (Batch (None, 2048)         8192
_________________________________________________________________
dense_2 (Dense)           (None, 1024)           2098176
_________________________________________________________________
batch_normalization_10 (Batc (None, 1024)         4096
_________________________________________________________________
dense_3 (Dense)           (None, 2)              2050
=================================================================
```

```
Total params: 24,831,458
Trainable params: 24,814,850
Non-trainable params: 16,608
_________________________________________________________________
```

A.4.2 Low Res Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 356, 512, 32)      11648
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 178, 256, 32) 0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 178, 256, 64)      51264
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 89, 128, 64)  0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 89, 128, 128)      204928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 44, 64, 128)  0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 44, 64, 256)       819456
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 22, 32, 256)  0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 22, 32, 512)       1180160
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 11, 16, 512)  0
_________________________________________________________________
flatten (Flatten)            (None, 90112)             0
_________________________________________________________________
batch_normalization (BatchNo (None, 90112)             360448
_________________________________________________________________
```

```
dense (Dense)                (None, 4096)              369102848
_________________________________________________________________
dense_1 (Dense)              (None, 2048)              8390656
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              2098176
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 6150
=================================================================
Total params: 382,225,734
Trainable params: 382,045,510
Non-trainable params: 180,224
_________________________________
```

A.4.3 High Res Model Summary

```
Layer (type)                 Output Shape       Param #    Connected to
==================================================================================================
input_1 (InputLayer)         [(None, 1600, 2400, 0
__________________________________________________________________________________________________
conv2d (Conv2D)              (None, 1600, 2400, 1 448      input_1[0][0]
__________________________________________________________________________________________________
average_pooling2d (AveragePooli (None, 800, 1200, 16 0     conv2d[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 800, 1200, 16 64
    average_pooling2d[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)            (None, 800, 1200, 32 4640
    batch_normalization[0][0]
__________________________________________________________________________________________________
average_pooling2d_1 (AveragePoo (None, 400, 600, 32) 0     conv2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 400, 600, 32) 128
    average_pooling2d_1[0][0]
```

```
__________________________________________________________________________________________________
conv2d_2 (Conv2D)            (None, 400, 600, 64) 18496
    batch_normalization_1[0][0]
__________________________________________________________________________________________________
average_pooling2d_2 (AveragePoo (None, 200, 300, 64) 0     conv2d_2[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 200, 300, 64) 256
    average_pooling2d_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)            (None, 200, 300, 64) 36928
    batch_normalization_2[0][0]
__________________________________________________________________________________________________
average_pooling2d_3 (AveragePoo (None, 100, 150, 64) 0     conv2d_3[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 100, 150, 64) 256
    average_pooling2d_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)            (None, 100, 150, 64) 36928
    batch_normalization_3[0][0]
__________________________________________________________________________________________________
average_pooling2d_4 (AveragePoo (None, 50, 75, 64) 0       conv2d_4[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 50, 75, 64) 256
    average_pooling2d_4[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)            (None, 50, 75, 128) 73856
    batch_normalization_4[0][0]
__________________________________________________________________________________________________
average_pooling2d_5 (AveragePoo (None, 25, 37, 128) 0      conv2d_5[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 25, 37, 128) 512
    average_pooling2d_5[0][0]
__________________________________________________________________________________________________
```

```
conv2d_6 (Conv2D)            (None, 25, 37, 128) 147584
    batch_normalization_5[0][0]
__________________________________________________________________________________________________
average_pooling2d_6 (AveragePoo (None, 12, 18, 128) 0      conv2d_6[0][0]
__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 12, 18, 128) 512
    average_pooling2d_6[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)            (None, 12, 18, 256) 295168
    batch_normalization_6[0][0]
__________________________________________________________________________________________________
average_pooling2d_7 (AveragePoo (None, 6, 9, 256) 0        conv2d_7[0][0]
__________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 6, 9, 256) 1024
    average_pooling2d_7[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)            (None, 6, 9, 256)  590080
    batch_normalization_7[0][0]
__________________________________________________________________________________________________
average_pooling2d_8 (AveragePoo (None, 3, 4, 256) 0        conv2d_8[0][0]
__________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 3, 4, 256) 1024
    average_pooling2d_8[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)         [(None, 3)]        0
__________________________________________________________________________________________________
flatten (Flatten)            (None, 3072)       0
    batch_normalization_8[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)    (None, 3075)       0          input_2[0][0]
                                                           flatten[0][0]
__________________________________________________________________________________________________
dropout (Dropout)            (None, 3075)       0          concatenate[0][0]
__________________________________________________________________________________________________
```

```
dense (Dense)                (None, 2048)       6299648    dropout[0][0]
--------------------------------------------------------------------------------------------------
batch_normalization_9 (BatchNor (None, 2048)    8192       dense[0][0]
--------------------------------------------------------------------------------------------------
dense_1 (Dense)              (None, 1024)       2098176
    batch_normalization_9[0][0]
--------------------------------------------------------------------------------------------------
batch_normalization_10 (BatchNo (None, 1024)    4096       dense_1[0][0]
--------------------------------------------------------------------------------------------------
dense_2 (Dense)              (None, 5)          5125
    batch_normalization_10[0][0]
==================================================================================================
Total params: 9,623,397
Trainable params: 9,615,237
Non-trainable params: 8,160
```

A.4.4 Figures

A.4.5 Tables

Bibliography

Ahmed, E., Moustafa, M., 2016. House price estimation from visual and textual features. arXiv preprint arXiv:1609.08399 .

Azizpour, H., Razavian, A.S., Sullivan, J., Maki, A., Carlsson, S., 2015. Factors of transferability for a generic convnet representation. IEEE transactions on pattern analysis and machine intelligence 38, 1790–1802.

Balchandani, C., Hatwar, R.K., Makkar, P., Shah, Y., Yelure, P., Eirinaki, M., 2017. A deep learning framework for smart street cleaning, in: 2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService), IEEE. pp. 112–117.

Chung, A., Kim, S., Kwok, E., Ryan, M., Tan, E., Gamadia, R., 2018. Cloud computed machine learning based real-time litter detection using micro-uav surveillance .

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.A., Vincent, P., Bengio, S., 2010. Why does unsupervised pre-training help deep learning? Journal of Machine Learning Research 11, 625–660.

Fukushima, K., 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological cybernetics 36, 193–202.

Fulton, M., Hong, J., Islam, M.J., Sattar, J., 2019. Robotic detection of marine litter using deep visual detection models, in: 2019 International Conference on Robotics and Automation (ICRA), IEEE. pp. 5752–5758.

Géron, A., 2017. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".

Gershgorn, G., 2018. The inside story of how ai got good enough to dominate sillicon valley. URL: https://qz.com/1307091/the-inside-story-of-how-ai-got-good-enough-to-dominate-silicon-valley/.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al., 2018. Recent advances in convolutional neural networks. Pattern Recognition 77, 354–377.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.

Hu, Y., Zhang, Q., Zhang, Y., Yan, H., 2018. A deep convolution neural network method for land cover mapping: A case study of qinhuangdao, china. Remote Sensing 10, 2053.

Khan, S., Rahmani, H., Shah, S.A.A., Bennamoun, M., 2018. A guide to convolutional neural networks for computer vision. Synthesis Lectures on Computer Vision 8, 1–207.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, pp. 1097–1105.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 2278–2324.

McCulloch, W.S., Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics 5, 115–133.

Nwankpa, C., Ijomah, W., Gachagan, A., Marshall, S., 2018. Activation functions: Comparison of trends in practice and research for deep learning. arXiv preprint arXiv:1811.03378 .

Perez, L., Wang, J., 2017. The effectiveness of data augmentation in image classification using deep learning. arXiv preprint arXiv:1712.04621 .

Radovic, M., Adarkwa, O., Wang, Q., 2017. Object recognition in aerial images using convolutional neural networks. Journal of Imaging 3, 21.

Raj, V., Magg, S., Wermter, S., 2016. Towards effective classification of imbalanced data with convolutional neural networks, in: IAPR Workshop on Artificial Neural Networks in Pattern Recognition, Springer. pp. 150–162.

Rosebrock, A., 2019. Keras: Multiple inputs and mixed data. URL: https://github.com/garythung/trashnet.

Rosenblatt, F., 1958. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review 65, 386.

Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1985. Learning internal representations by error propagation. Technical Report. California Univ San Diego La Jolla Inst for Cognitive Science.

Sameen, M.I., Pradhan, B., Aziz, O.S., 2018. Classification of very high resolution aerial photos using spectral-spatial convolutional neural networks. Journal of Sensors 2018.

Sarkar, D., 2018. A comprehensive hands on guide to transfer learning with real world applications in deep learning. URL: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-

Scotsman, 2013. Cost of scotland's litter problem revealed as 53m. URL: https://www.scotsman.com/news/environment/cost-of-scotland-s-litter-problem-revealed-as-53m-1-2988577.

Scott, G.J., England, M.R., Starms, W.A., Marcum, R.A., Davis, C.H., 2017. Training deep convolutional neural networks for land–cover classification of high-resolution imagery. IEEE Geoscience and Remote Sensing Letters 14, 549–553.

Scrapbook, 2019. Scrapbok website. URL: https://www.scrapbook.org.uk/.

Simonyan, K., Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 .

Society, M.C., 2018a. 25th great british beach clean report. Marine Conservation Society (MCS) 2018 .

Society, M.C., 2018b. Be part of the biggest ever scotland-wide beach litter pick. Marine Conservation Society (MCS) 2018 .

Thung, G., 2017. Trashnet repository. URL: `https://github.com/garythung/trashnet`.

Wu, E., Wu, K., Cox, D., Lotter, W., 2018. Conditional infilling gans for data augmentation in mammogram classification, in: Image Analysis for Moving Organ, Breast, and Thoracic Images. Springer, pp. 98–106.