General report of image detection model

Introduction

Computer vision is one of the most research fields in machine learning and is the study of helping computers to see like humans. Object classification is one of the most popular computer vision research areas.

The goal of this report is to outline the steps taken to develop an end-to-end machine learning pipeline used to develop a machine learning model which can categorise fine grained images of dogs into their subcategories (breeds). The Stanford Dog Dataset was used for this analysis.

The report will explore:

- The dataset and outline the key considerations when starting the analysis.

- How the group prepared the dataset, so it was ready for modelling.
- The modelling process, looking at several different well-known neural network architectures.
- How we tried to mitigate errors throughout the process.
- The results of our best model.

In this report we present analysis on pre trained models looking at vgg16, ResNet50 and Xception. The analysis showed that the Xception was the best model outperformed the other two pre trained models, so we used this to explore further providing some good results.

Description of the dataset

Dataset summary

The Stanford Dogs dataset contains 20580 images of 120 breeds of dogs from around the world and each breed has around 170 images on average.

- **"Images" directory**: Images files (JPG) of different breeds are stored in a separate folder which named by its breed id and name (e.g. n02085620-Chihuahua). Images in each folder are named by breed id followed by image id.
- **"Annotation" directory**: Same structure with "Images" directory that each image has a corresponding annotation. It contains the class label of image, image size and the coordinate of bounding box which indicates the location of dog in the image.
- **"list" directory**: Including file_list.mat, train_list.mat and test_list.mat which is the list of all files, training images and test images in the datasets respectively.

Dataset analysis

First, we imported train and test lists (train_list.mat, test_list.mat) as pandas data frames. For each breed, there are 100 images been assigned to training set and the rest are in test set. The figure below shows the number of images in each breed. As we can see, Redbone breed has the minimum number of images at 148, while Maltese contains the greatest number of images at 252.The training sets have been split such that there are 100 images for each of the dog breeds and the remainder are in the test set. This means that the model will not be biased to any breed.

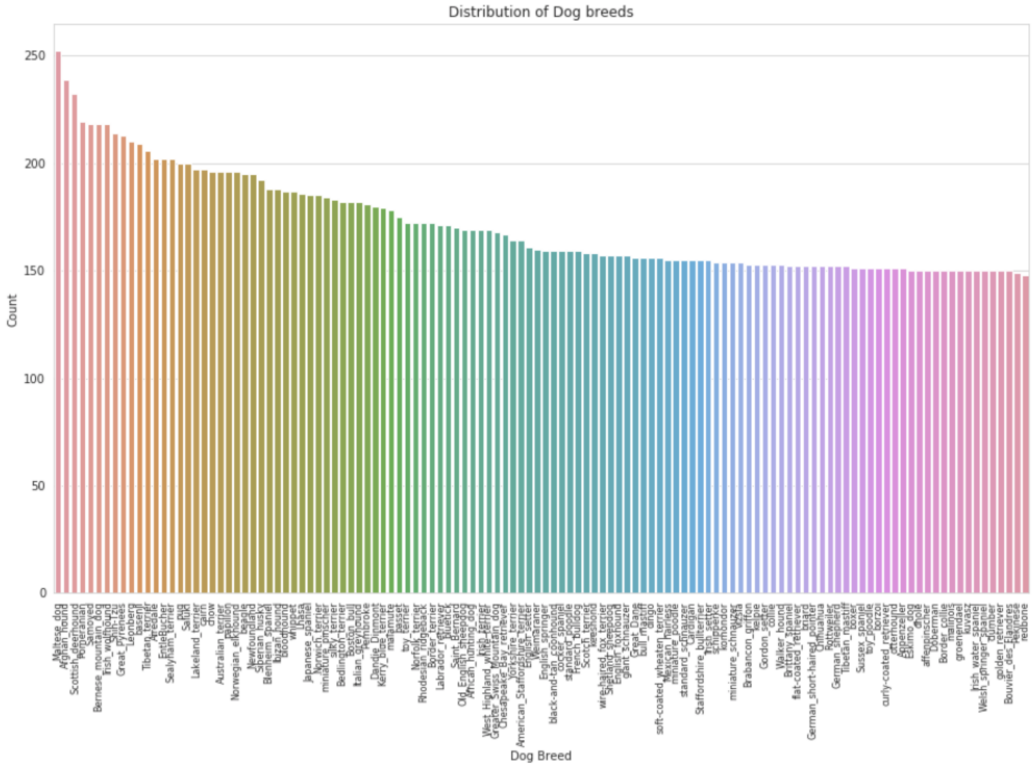

Figure 1. Distribution of Dog breeds

Pandas function **describe()** function is used to view some basic statistical detail of dataset (see Table 1). It returns the number of data instances which determines there are 120 breeds in the dataset, the mean number of dog images among all breed is around 170. std indicates the standard deviation of data which is a measure of the amount of variation in the dataset. Also, the minimum of dog images of breeds and the percentiles located in 25th, 50th and 75th places.

Table 1. Descriptive statistics of image number of Dog breeds

	count	mean	std	min	25%	50%	75%	max
Count	120.0	171.5	23.220898	148.0	152.75	159.5	186.25	252.0

In order to figure out the appropriate size of images for all data and how image size will affect the experiment. We used the describe() function on the width and height of all images in the dataset (20,580 images). The Results (Table 2) Show that the majority of images are between a width of 361 and height of 333 to a width of 500 and a height of 453. We have a large distribution of image sizes (Fig. 2) which means that we will have to compress some images and enlarge others which will affect the visual information of each image. These images are very complex, and his information will be very important what trying to classify dog breads. (Kannojia, 2018) concludes that performance of the classifier is mainly depended upon visual information and resolution of images so this will have to be considered in our analysis.

Table 2. Descriptive Statistics of image sizes.

	Width	*Height*
mean	442.5	385.9
std	142.8	124.9
min	97	4
25%	361	333
50%	500	375
75%	500	453
max	3264	2562

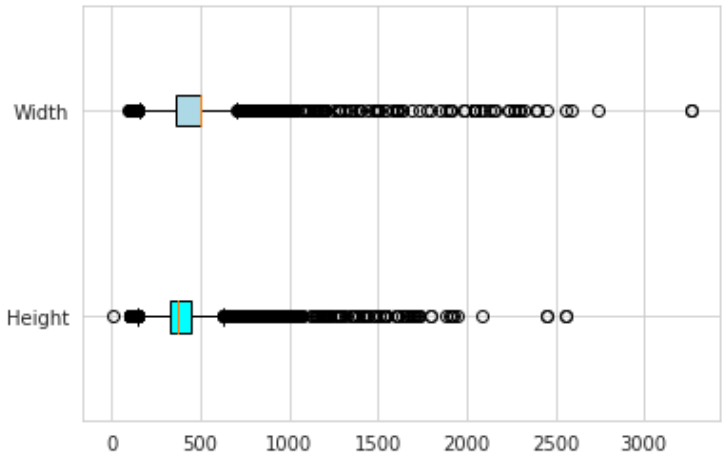

Figure 2. Distribution of image sizes

We have selected random example images from datasets to observe significant pose variation and background. Additionally, there is large variation in appearance of the dogs between classes. For example, the same breed doesn't share a consistent colour of fur and puppy and fully-grown dog have different sizes because of age. Furthermore, their appearance is often modified by humans by placing clothing or cutting/growing the fur (Khosla et al. 2012).

Data Preparation

Dataset Split

Stanford Dogs data frame consist of 2,580 images, illustrating 120 different breeds of dogs from all over the world. The images and the corresponding annotations have been drawn from ImageNet for the purpose of fine-grained categorisation. This dataset contains the following files:

- **I**mages (757MB)
- Annotations (21MB)
- Lists, with train/test splits (0.5MB)
- Train Features (1.2GB), Test Features (850MB)

Annotation folder consist of 120 sub-folders each one of which has a unique serial number along with the actual breed name. Image names are allocated inside these folders and expressed by the corresponding breed serial number plus a shorter number appended at the end. The 'File_list' contains the annotation path of every image as well as the label, that is a number ranging from 1 to 120 which refers to the breed category. Using this file, we were able to obtain the paths as strings and extract the breed names. After

separating the image names from the rest of the path, we created a data frame that has two columns referring to image IDs and the analogous breed names.

In any modelling process, regardless of the application, there is a need of validating and testing the model. In anticipation of this the initial data frame was already split into *test* and *train* sets using a 60:40 ratio, that is 12,000 images for the *train* and the remaining 8,580 for the *test* set. A brief literature review showed that there is no global or ideal ratio of splitting. In fact, if the data-set is large enough one can consider ratios like 70:30 or 75:25, while in the case of small data-sets it is recommended to use ratios such as 90 : 10. Above all, it is important how well the training and validation sets describe the feature space.

During the training process we realised that the number of training images was not enough for achieving the desired performance. Thus, it was decided to ignore the provided split and choose a different one using a 90:10 *train:test* ratio. In addition, we further extract a 10% validation set, ending up with three data partitions, which are train, validation and test samples with proportions of 80%, 10% and 10% respectively. These proportions reflect to 16,418 images for the train, 2,009 images for the validation and 2,153 images for the test partition. The validation set was used to experiment on improving model's accuracy, while the overall performance was evaluated on the test sample.

Having decided the data frame split ratios, we proceed to create a new directory that contains three folders, one for each of our data partitions. The images are copied and allocated in these three folders with respect to the proportions mentioned above.

Image Pre-processing

When importing images, we used the Keras ImageDataGenerator class. This class provides the configuration for image data preparation and augmentation (Brownlee, 2020). This API allows us to normalise and augment the images we will use to train our model.

Normalisation of the images is important because it makes sure that each pixel has a similar distribution which will allow faster convergence when training the model. (Image Data Pre-Processing for Neural Networks, 2020)

Image augmentation is important because it allows us to generate more training data by using our existing training data sets by transforming the original images providing more data for each class (Brownlee, 2020).

When generating images for model training the first step of the image generator is to rescale the image. We use 1./255 because 255 is the maximum pixel value and using this rescale will normalise the pixels to [0,1] rather than [0,255]. This will allow us to treat all images the same (Keras Image Preprocessing: scaling image pixels for training, 2020). We augmented the image in several different ways.

- Rotation of 45-degrees to generate the images of the dog at a 45-degree angle.
- Width and height shift that randomly shifts the images left and right and up and down.
- Slanting the image.
- Zooming out
- Horizontally and vertically flipping the image
- Setting the fill mode to nearest which autofill's any empty pixels with the nearest pixel value.

(Exploring Image Data Augmentation with Keras and Tensorflow, 2020)

For the validation and test sets we use an image generator with rescaling only to make sure the images are normalised the same as the training set.

Implementation of Model

Deep learning is thought of as learning a hierarchical set of representations, such that it learns low, mid and high-level features. In images this analogous to learning edges, shapes and finally objects. For the purpose of this image classifier we implemented several models in order to achieve the highest possible performance. The attempted models are the following:

- Vgg16
- ResNet50
- Xception

These Convolutional Neural Networks have been pre-trained on ImageNet dataset and are included in Python's Keras core library. They represent some of the highest performing networks and demonstrate strong generalisation ability to images outside the ImageNet dataset, via transfer learning such as feature extraction and fine-tuning. ImageNet is a project focused on labeling and categorizing objects into 22,000 separate categories for the purpose of computer vision research. In the context of deep learning it usually refers to ImageNet Visual Large-Scale Recognition Challenge or ILVSRC. The models are trained on 1.2 million images, validated on 50,000 more and tested on 100,000 images. The images categories correspond to daily life object classes, such as dogs, cats, vehicle types etc. After evaluating the performance of every model, it was found that the Xception slightly outperforms ResNet and significantly outperforms Vgg16 network. This is due to the fact that Xception is a deeper network that also contains residual skip connections among the layers (like ResNet) and at the same time is considerably lighter than VggNet, which contains a large number of fully-connected nodes (approximately 138 million parameters).

Xception

Xception stands for the Extreme version of Inception and replaces the standard Inception modules with depth-wise separable convolutions (a point-wise convolution followed by a depth-wise convolution).[] This network slightly outperforms Inception V2 and significantly outperforms Inception V3 on the ImageNet data-set. Since the Xception architecture has the same number of parameters as Inception V3, the performance gains are not due to increased capacity, but rather to a more efficient use of model parameters. The number of connections is smaller which leads to lighter model.

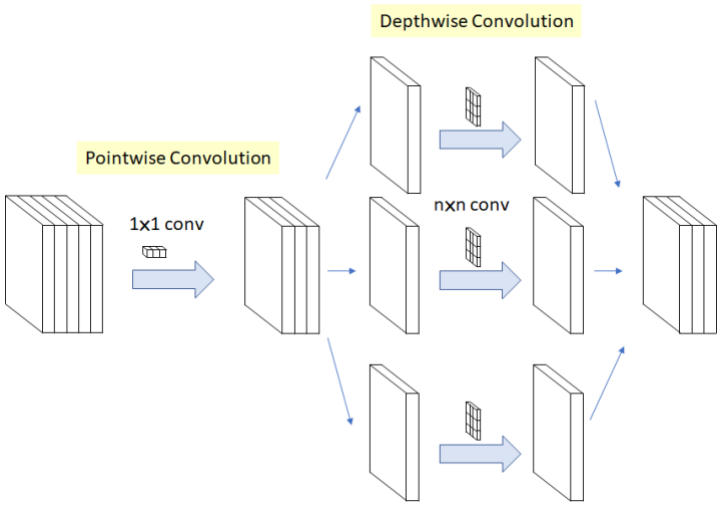

Figure 3: Xception Architecture

Depth-wise convolution is a channel-wise network, in which the initial number of channels is translated to n×n spatial convolutions. Unlike Inception V3, in this case the 1×1 convolutional is done first, before any spatial convolutions. However, this is claimed to be unimportant because when it is used in stack setting, there are only small differences appeared at the beginning and at the end of the chained Inception model. Another difference between Inception and Xception is the presence/absence of non-linearity. While in the original Inception module there is non-linearity after the first operation, in Xception there is no intermediate ReLU non-linearity.

For the purpose of this model, different activation units were tested. It was found that the Xception without any intermediate activation has the highest accuracy compared with the ones using either ELU or ReLU.

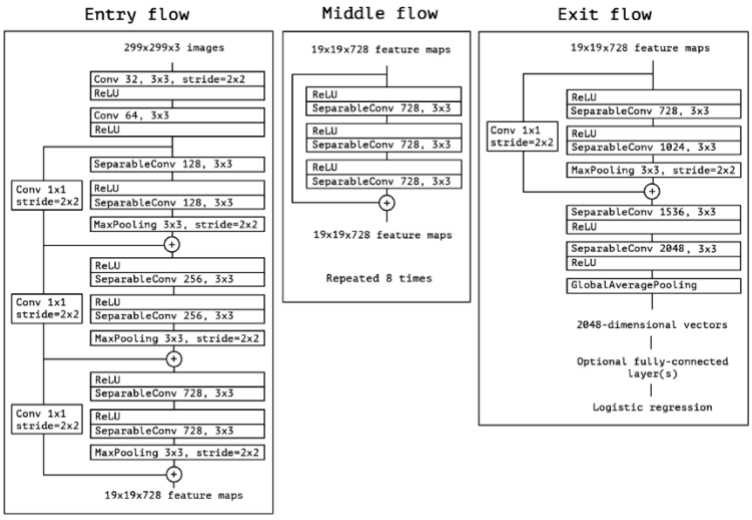

Figure 4: Overall Xception Architecture

As we can see in figure 4, the separable convolutions are treated as Inception modules and placed throughout the whole deep learning architecture. In addition, there are residual skip connections, originally proposed by ResNet, placed for all flows. After testing the model with and without residual connections, it was found that the accuracy is much higher when using them. Xception is claimed to have similar model size with Inception V3.

Model build

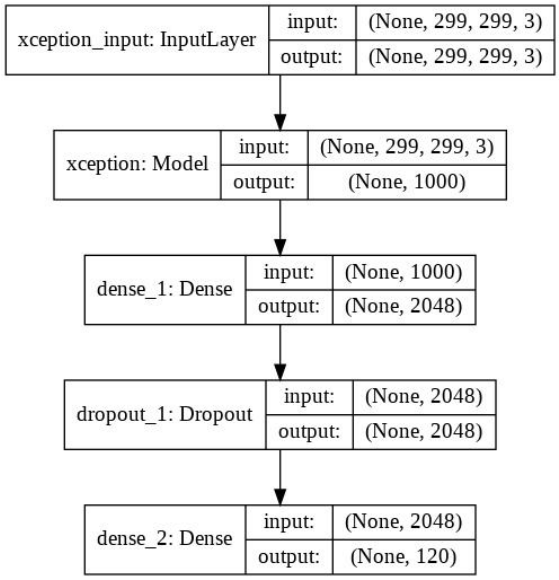

Figure 5: Part of Model Architecture

At the beginning of the function, define inputs as a custom layer with a size of (size, size, 3).

Create a lightweight convolutional neural network MobileNetV2 model structure 'input_tensor' is the inputs used as the model's image input. Moreover, set the trainable property to True. Then set the layers x. X is assigned to the MobileNetV2 model structure 'backbone.output'.

GAP (Global Average Pooling) can use for fine-tuning the model, which is better when doing image classification competitions. So, use GlobalAveragePooling2D () (x) to fine-tune the x model.

Use Dropout (0.2) (x) for x. This operation is equivalent to adding noise to the neural network so that it cannot over-rely on certain features to reduce the over-fitting of the model. After adding dropout, the first neural network can be regarded as a bagging model of multiple sub-networks. Then use Dense () to perform a fully connected layer operation on x processed by GlobalAveragePooling2D (). This operation will allow them to alleviate the problem of disappearing gradients, enhance feature propagation, encourage functional reuse, and significantly reduce the number of parameters. Dense Nets is a significant improvement over the latest technology in most respects while requiring less computing to achieve high performance.

Main Body

Callback Customize a callback, pass the original model into the callback class through ModelCheckpoint, and use ReduceLROnPlateau to control the model training and learning indicators. When the indicator stops rising, it will reduce the learning rate and focus on measuring significant changes in the new optimized threshold, which can optimize the final result of the model. The model uses the fit method to train the data 10 times that Controlled by epochs. Using the data set, API can extend to large data sets or multi-device training. Will generate by build_model. The tf.data. Dataset instance passed to the fit method Use the Datasets API to scale to large data sets or multi-device training. The fit method uses the steps_per_epoch parameter-this is the number of training steps the model runs before moving to the next epoch. Because the data set generates batch data, this code snippet does not require batch_size.

Results

After running the Xception model for 30 epochs we obtained the accuracy and loss values for both the train and validation sets. More specifically, in the first epoch the model has 4.7 loss and 0.1 accuracy on the train set, while the loss and the accuracy on the validation set is at 4.3 and 0.3 respectively. At the end of the model fitting process, the results of the final epoch were ready to be reviewed. In fact, the loss of the train set decreased to 2.1 and the accuracy rose at 0.5. Regarding the validation set the loss dropped considerably, at 0.4 and the accuracy soared at 0.9, which is a highly acceptable value.

Having gone through the fitting process, we implement keras' *predict_generator* function in order to retrieve the predictions of the test set in the form of *numpy arrays* with shape (2153,120), where the first number refers to the length of the test set and the second one to the total number of classes-breeds. In addition, we create a second array that contains the labels of the test set, as numbers ranging from 0 to 119. Using the *to_categorical* function of keras library, we manage to reshape this array so that it matches the predictions shape. Finally, with these two ingredients, plus the *evaluate* function of keras, we obtain the accuracy of the network on the test sample which peaks at 92%.

Error Analysis

Misclassified Breeds

For these fine-grained images of dog breeds there is sometimes no clear line for classification. Some breeds look very different e.g. Doberman and a Chihuahua, some breeds have lots of similarities e.g. Whippet and an Italian greyhound and some breeds have in class variability which could make them difficult to classify. These breeds would be difficult to classify for a human, so it would also be difficult for a machine learning algorithm to tell the difference. A machine learning algorithm would have to find subtle differences in these breeds in order to classify accurately, e.g. eyes, ears, tails or colour (Wei, X.S., Wu, J. and Cui, Q., 2019). These Images would also have to be classified correctly by a human before an algorithm can be trained correctly. We must accept the fact that our datasets will contain images which are miss classified.

We also need to consider images where there contains more than one dog breed. This example (Fig.6) shows a Pitbull terrier and an Irish Greyhound in the same image. This image is from the training set and is categorised as an Italian Greyhound. This would cause issues when trying to train an algorithm to predict the correct dog breed.

Figure 6: Image example of two breeds.

Bias and Variance

Bias can present itself in several ways in the modelling process. Initially there could be bias within the datasets. From the exploratory data analysis, we saw that some dog breeds have much more images than others. This would cause bias in the model because the model would perform better with these breeds. When we split the datasets, we made sure that the distributions for each breed stayed this should help mitigate the risk of bias between breeds.

To try and reduce the variance in the model we increased the size of the training set so there was more data available to the model for each breed. We also used a Sparse model which gives a good combination of a better model with less variance. (Error Analysis in Neural Networks, 2020).

Error analysis on Models

The system performance is analysed in detail and the changes in the accuracy of the train set and validation set under different conditions are considered in experiments. The first step was to split the data into training and validation sets. With the data split, we then fit the training set to the model and run through the validation set in order to obtain the accuracy value. In our model, it attempts to find which set of neurons would provide us with the best model to predict dog breeds.

A pre-trained convolutional neural network with depthwise separable convolutions (Xception) was shown to perform best for this dog breeds image classification task. Xception is an extension of the Inception architecture which replaces the standard inception modules with depthwise separable convolutions (Chollet 2017). In our model, it includes a dropout layer of rate 0.3 before the dense layer, with batch size set as 20 and stochastic gradient descent with Nesterov momentum (SGDN) as optimizer, the accuracy of train set is 52.13% and reached 90.04% for validation sets after 30 epochs.

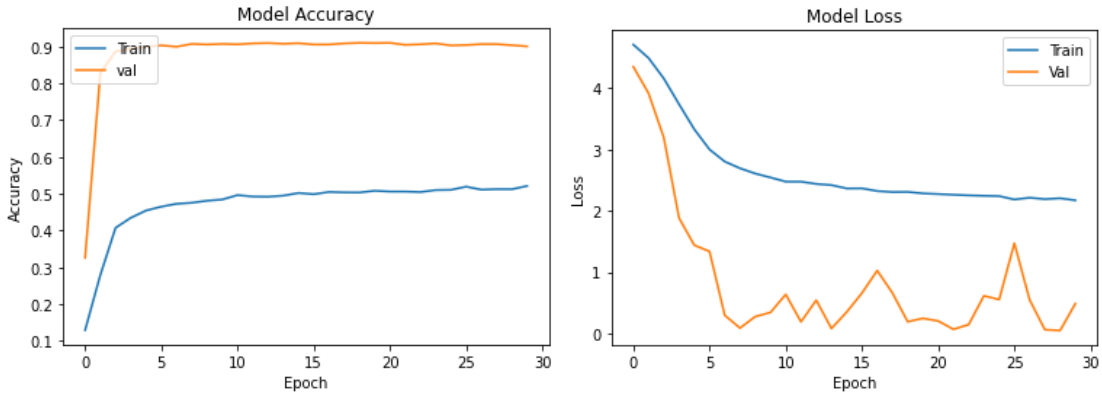

Figure 7. Accuracy of xception algorithm changes in 30 epochs

Meanwhile, we have also implemented VGG16 in experiments, but it does not perform as good as xception. From its definition, the limitation which may cause the wrong classification including the VGG16 encoder requires a large amount of training data and relies heavily on the data augmentation (Balakrishna et al. 2018). However, the total number of images for each breed are only around 150 before splitting into train and testing set. In addition, the implementation of dropout layer. The dropout layer disables 30% of the neurons from the previous dense layer. Although it helps to make the network more robust as every time the disable neurons are chosen randomly. While if some neurons which includes essential information for classification been removed from consideration in the following layers. In this case, it may have an impact on its performance and resulting in low accuracy.

Our final Xception model shows the validation accuracy much higher than the training accuracy (Fig.7). This is because we are using dropout. Dropout is when the model randomly ignores selected neurons when training. This results in a model that is capable of better generalization and is less likely to overfit the training data. Making the model more robust on the validation set because it is using all the features.

Literature Review

Choice of Technology

In the field of image classification, neural network and support vector machine (SVM) are two commonly used technologies. In this modeling, we chose the convolutional neural network (CNN) as the framework of our model. The main reason is that support vector machines (SVM) are mainly used to process data sets with small samples. When processing large samples and multi-classification datasets, its efficiency will decrease obviously. At the same time, with their self-learning ability and ability to find optimal solutions quickly, neural networks have become popular in completing such tasks. In the convolutional neural network, CNN can imitate the cell properties in mammalian visual cortex by alternating the convolutional layer and the maximum convergence layer. Hence, compared with traditional support vector machines and deep neural networks, CNN shows a better performance in vision-related fields (Ciresan et al., 2011 and Hu et al., 2015). In addition, neural codes which are the features of the upper layers of the convolutional neural network have excellent performance, as visual descriptors, and their

performance can be improved significantly when CNN is trained to similar datasets (Babenko, Slesarev, Chigorin and Lempitsky, 2014).

Image Augmentation

In the field of image classification, the lack of training set has always been a major problem affecting the model accuracy, thus making it necessary to transform the original training image to create new images to expand the training set, which is called image augmentation. In our model, several common image transformation methods were used, including rotation, scaling and noise. Based on these common variations, the researchers improved the quality and efficiency of the images they produced with more sophisticated methods. For example, in the training image, many sub-windows whose positions and sizes may overlap randomly are extracted and trained after adjusting them to a fixed size (Maree, Geurts, Piater and Wehenkel, 2005). However, the accuracy obtained with this completely random sampling method is based on a large training set, thus making it difficult to maintain high accuracy and efficiency at the same time. In addition, the content of the basic image is combined with the appearance of other images through style transformation, to generate new images of high perceived quality for pre-training.(Mikolajczyk and Grochowski, 2018). Besides, inspired by the game theory, the generative adversarial Networks (GANs) is also widely used in the field of computer vision. In GANs, there are two networks which are trained in an adversarial process are used in the model, in which, a network is responsible for generating the false image and the other is responsible for repeatedly distinguishing true images from false ones, to generate high-quality synthetic image. (Frid-Adar et al., 2018) Compared with the first method, the quality and relevance of the images generated with the latter two methods are much higher, which can effectively improve the model accuracy.

At present, the basic image transformation has been implemented under the mainstream deep learning framework, while the complex image augmentation usually requires the help of image augmentation tools. Currently, common image enhancement tools include Imgaug, Augmentor, and CLoDSA. In addition, an open as a source Python library for fast and flexible image augmentations, Albumentatuins can effectively transform images and provide a powerful image augmentation interface for different computer vision tasks (Buslaev et al., 2020).

Model contrast

Two machine learning models in which CNN is used to classify dog breeds, serving as a good example for our model. In the model built by Hsu in 2015, caffe was used as a deep learning framework in the model to compare lenet and geoglenet which is common CNN architectures in terms of the efficiency. Besides, when extracting the training image from the original image, he used the annotated border to clip the original image, and then selected two dimensions with pixels larger than 256 to adjust the dimension of the smaller pixels to 256, and took [1:256] rows and columns finally. Eventually, he got 5,678 training images and 4,007 test images. However, clearly, there are some defects in his modeling process. Firstly, only two CNN architectures, namely, lenet and geoglenet, were used in the model, so examples for comparison are relatively less comprehensive. In addition, there is no data augmentation in this model, and the proportion of data set partition as a test is too high, which may affect the accuracy.

In Ayanzadeh and Vahidnia's (2018) model, the model on imagenet dateset was built and more architectures were used for testing. Furthermore, in order to prevent the conventional model of imagenet from being unable to classify effectively due to its high complexity, they used the pre-trained model to extract features. About image augmentation, a random per-pixel mean subtraction and some traditional

augmentation techniques were used in the model, with the ratio of the training set to the testing set being set to 7:3.

Conclusion

As demonstrated in what is discussed above, in the field of vision, CNN shows a higher performance than SVM and other neural networks. Besides, with data augmentation, the data set can be effectively expanded, and the model accuracy can be improved. Furthermore, when modeling, multiple CNN architectures can be selected for comparison to select the most accurate architecture

Conclusion

Future work

To further improve this analysis, we could have explored:

- Deeper Network Topology: this would involve exploring deeper models allowing us to explore and learn features at a much lower level.
- Explore how other algorithm tuning methods could affect the accuracy of the model such as early stopping or using different batch types and epochs.
- Try different resampling methods such as K-folds cross validation.

Problems encountered

There were several problems face when completing this project. These include:

- Separations of tasks made it difficult to work on some parts. For example, data preparation needed to be completed before the modelling which meant that some people were waiting for others to complete their section before you could continue.
- Training a model on images is very time consuming. Code would be left to run overnight using colab but we would encounter errors such as the session being terminated due to inactivity while our models were running which meant they would not save. This made it very challenging.
- Time differences with some members of the team moving back to other countries due to the Covid-19 pandemic meant communication was very difficult.

References

Brownlee, J., 2020. Image Augmentation For Deep Learning With Keras. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/image-augmentation-deep-learning-keras/> [Accessed 28 April 2020].

Medium. 2020. Image Data Pre-Processing For Neural Networks. [online] Available at: <https://becominghuman.ai/image-data-pre-processing-for-neural-networks-498289068258> [Accessed 28 April 2020].

Linkedin.com. 2020. Keras Image Preprocessing: Scaling Image Pixels For Training. [online] Available at: <https://www.linkedin.com/pulse/keras-image-preprocessing-scaling-pixels-training-adwin-jahn> [Accessed 28 April 2020].

Medium. 2020. Exploring Image Data Augmentation With Keras And Tensorflow. [online] Available at: <https://towardsdatascience.com/exploring-image-data-augmentation-with-keras-and-tensorflow-a8162d89b844> [Accessed 28 April 2020].

Ayanzadeh, A. and Vahidnia, S. 2018. Modified Deep Neural Networks for Dog Breeds Identification. doi: 10.20944/preprints201812.0232.v1.

Babenko, A. et al. 2014. Neural Codes for Image Retrieval. Computer Vision – ECCV 2014 , pp. 584-599. doi: 10.1007/978-3-319-10590-1_38.

Buslaev, A. et al. 2020. Albumentations: Fast and Flexible Image Augmentations. Information 11(2), p. 125. doi: 10.3390/info11020125.

Chapelle, O. et al. 1999. Support vector machines for histogram-based image classification. IEEE Transactions on Neural Networks 10(5), pp. 1055-1064. doi: 10.1109/72.788646.

Ciresan, D. et al. 2011. Flexible, High Performance Convolutional Neural Networks for Image Classification. Available at: https://www.aaai.org/ocs/index.php/IJCAI/IJCAI11/paper/viewPaper/3098 [Accessed: 27 April 2020].

Dollar, P. et al. 2007. Feature Mining for Image Classification. 2007 IEEE Conference on Computer Vision and Pattern Recognition . doi: 10.1109/cvpr.2007.383046.

Frid-Adar, M. et al. 2018. GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification. Neurocomputing 321, pp. 321-331. doi: 10.1016/j.neucom.2018.09.013.

Hu, W. et al. 2015. Deep Convolutional Neural Networks for Hyperspectral Image Classification. Journal of Sensors 2015, pp. 1-12. doi: 10.1155/2015/258619.

Giacinto, G. and Roli, F. 2001. Design of effective neural network ensembles for image classification purposes. Image and Vision Computing 19(9-10), pp. 699-707. doi: 10.1016/s0262-8856(01)00045-2.

Maree, R. et al. [no date]. Random Subwindows for Robust Image Classification. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) . doi: 10.1109/cvpr.2005.287.

Mikolajczyk, A. and Grochowski, M. 2018. Data augmentation for improving deep learning in image classification problem. 2018 International Interdisciplinary PhD Workshop (IIPhDW) . doi: 10.1109/iiphdw.2018.8388338.

Perez, L. and Wang, J. 2017. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. Available at: https://arxiv.org/abs/1712.04621 [Accessed: 27 April 2020].

Thai, L. et al. 2012. Image Classification using Support Vector Machine and Artificial Neural Network. International Journal of Information Technology and Computer Science 4(5), pp. 32-38. doi: 10.5815/ijitcs.2012.05.05.

Tzotsos, A. and Argialas, D. [no date]. Support Vector Machine Classification for Object-Based Image Analysis. Lecture Notes in Geoinformation and Cartography , pp. 663-677. doi: 10.1007/978-3-540-77058-9_36.

Xie, S. et al. 2015. Hyper-class augmented and regularized deep learning for fine-grained image classification. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) . doi: 10.1109/cvpr.2015.7298880.