

CE/CZ4041 Project: Dog Breed Identification (Kaggle)

Cao Hai Nam*
Nanyang Technological University
HAINAM001@e.ntu.edu.sg

Harold Teng Ze Chie*
Nanyang Technological University
HTENG001@e.ntu.edu.sg

Kok Zi Ming*
Nanyang Technological University
KOKZ0003@e.ntu.edu.sg

Manish Dhanetwal*
Nanyang Technological University
MANISH007@e.ntu.edu.sg

Pang Yu Shao*
Nanyang Technological University
C170134@e.ntu.edu.sg

1 INTRODUCTION

In this project, a Machine Learning model is implemented to solve the task of classifying a dog's breed based on an input image as part of a Kaggle competition, "**Dog Breed Identification**"¹.

Various methods are experimented and evaluated in order to obtain a final Implementation of a model which is able to predict the given test data with the lowest error and thus provide a higher score on the Kaggle competition's leaderboard.

2 DATASET

The dataset provided for this task is a canine subset of the ImageNet dataset [9]. Each of the images belong to one of 120 different classes (breeds).

2.1 Class Distribution

The training set was analysed to determine the distribution of classes for the training samples, the distribution can be seen in Figure 1.

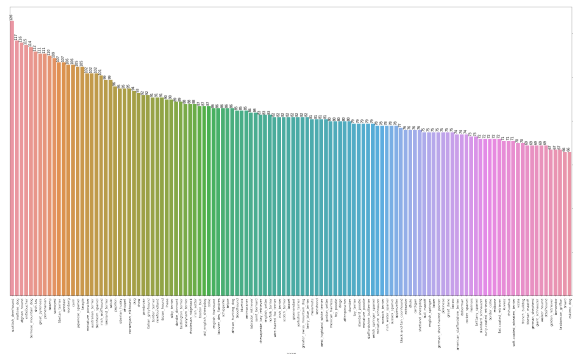


Figure 1: Distribution of Classes of Training Examples

From Figure 1, it can be seen that the dataset is imbalanced. This may result in biases being learnt by the model during the training stage. Therefore, various strategies such as oversampling and undersampling will be explored to allow the model to make more accurate predictions to unseen data.

3 IMPLEMENTATION

An Artificial Neural Network (ANN) was chosen for the implementation of the machine learning model with the use of **Transfer**

Learning by using other pre-trained Deep Learning models for extracting learned features from the images. This extraction of features is done as a "preprocessing" step and is done before training time. The dataset is then rebuilt with the extracted features as the inputs while the output target label is unchanged.

A simple ANN is then implemented as the "classification head", which learns to classify the breed of the dog in the input image based on the extracted features. Since the dimension of feature vectors at the output of the pre-trained models are large, a dropout layer is also applied to prevent over-fitting by the ANN. The architecture of the model is shown in Figure 2.

3.1 Transfer Learning

Transfer Learning is defined as given a source domain and learning task and a target domain and learning task, *transfer learning* aims to improve the learning of the target predictive function in the target domain using the knowledge in the source domain and the source task, whereby the source task and domain is not the same as that of the target task and domain [7].

With image datasets being publicly available such as ImageNet and COCO [6], challenges such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) as well as the COCO Challenge are held to spur research into tasks such as image classification or object detection.

From these challenges, many State-of-the-Art neural network architectures have been published and made available, such as: **Inception** [11] (ILSVRC2014) and **ResNet** [4] (ILSVRC2015). Many other architectures are further built upon these existing architectures with improved performance, such as the **Inception-ResNet** [10] architecture, which is built upon both the Inception and ResNet architectures.

As these pre-trained models are trained on the same dataset used in our task (i.e., ImageNet), the models' weights are well-trained to extract high-level features from the images. Therefore, the pre-trained models can be adapted for use as feature extractors by removing the classification layer. The layer before the classification layer would thus provide an image vector when an image is fed into the input, which is essentially a representation of high-level features of the input image.

In this case, the source and target domains are the same while the source and target tasks are different. This is also known as *Inductive transfer learning* which was introduced by Pan and Yang [7]. Specifically, this is a case of *Sequential transfer learning* (introduced by Ruder [8]), where the tasks are learnt in sequence and the

*All authors contributed equally to this project.

¹<https://www.kaggle.com/c/dog-breed-identification/overview>

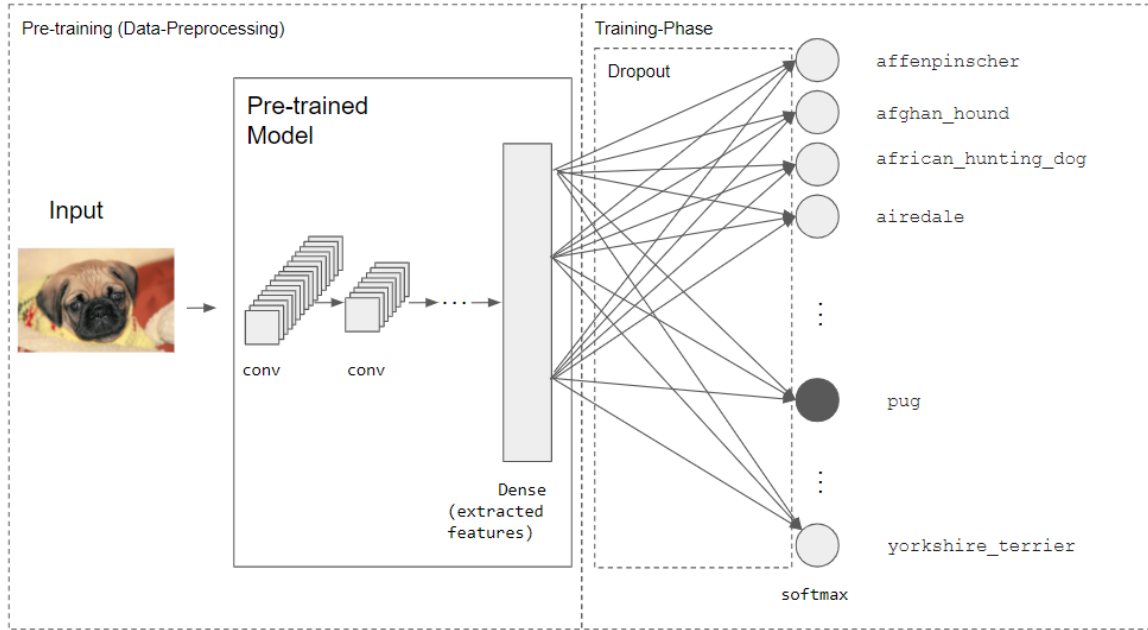


Figure 2: Architecture of the ANN built on top of the pre-trained model

knowledge of the model trained on the source task is transferred to the target task.

4 EXPERIMENT RESULTS & MODEL SELECTION

In this section, experiments are carried out and evaluated to arrive at a final model which would give the highest score which would be used to predict the classes of the unseen test data for submission to the Kaggle competition.

4.1 Experimental Set-up

In each of the experiments carried out, the models will be evaluated based on the loss on the **validation set**. The loss function would be the **multi-class log loss**, also known as the multi-class cross entropy. This loss function is also used in the judging of the submission by Kaggle.

The optimiser used for training the model is the **Adam** optimiser, with learning rate annealing and early stopping applied when the validation loss is detected to not improve over a number of training cycles (**epochs**). Each of the models were trained until they were stopped by the early stopping mechanism.

4.2 Pre-trained Model Selection

A few pre-trained models of varying architectures were used for feature extraction, and the extracted features were used to train the ANN. The following pre-trained models were used as feature extractors in our experiments:

- (1) InceptionV3 [12]
- (2) Xception [3]
- (3) Inception-ResNetV2 [10]

- (4) NASNetLarge [14]

- (5) BiT-S R101x3 [5]

Each of the models were evaluated by training the ANN with the same hyperparameters, with a learning rate of **0.0001**, and a drop probability of **50%** with stratified k-fold cross-validation. The following table shows the average validation loss from the model using each of the various pre-trained models as feature extractors:

Feature Extractor	Average Loss (Cross-Entropy)
InceptionV3	0.2614
Xception	0.2577
Inception-ResnetV2	0.2361
NASNetLarge	0.2018
BiT-S R101x3	0.2777

Table 1: Comparison of performance from utilising Feature Extractors

From Table 1, it can be seen that the features extracted by the NASNetLarge model yielded the best performance when used as the inputs to our ANN.

4.3 Combining Features from Different Feature Extractors

In an attempt to further increase the performance of the model, we experimented the concatenation of all features obtained from **all** pre-trained models stated in Section 4.2. This architecture is depicted in Figure 3. With the new concatenated features, the ANN was trained with the same hyperparameters as the models trained

from Section 4.2 and yielded a final average validation loss of **0.17** as shown in Table 2 below.

Feature Extractor	Average Loss (Cross-Entropy)
NASNetLarge (Best)	0.2018
Concatenated	0.1752

Table 2: Comparison of performance from utilising Feature Extractors

Therefore, the concatenated features will be used as inputs to our ANN as it yielded the best performance during our experiments.

4.4 Hyperparameters Tuning

With the feature extraction method decided, the next step for improving the performance of the model is to tune the hyperparameters used for the training of the model (i.e., **dropout** and **learning rate**). To accomplish this, **Grid Search** was performed. Grid search is performed by trying out all combinations of the parameters and to compare the performance of the models trained with the combination of the hyperparameters chosen. Stratified k-fold cross-validation was performed for each of the combination to obtain the average loss of the model trained with the combination of hyperparameters.

4.4.1 First Pass. For the first pass, the grid search was performed with the following variable values:

$$lr \in [0.0001, 0.0002, 0.0003, 0.0004, 0.0005]$$

$$dropout \in [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$$

After performing the grid search, the average validation loss of the model was recorded and plotted on a heatmap in Figure 4 below:

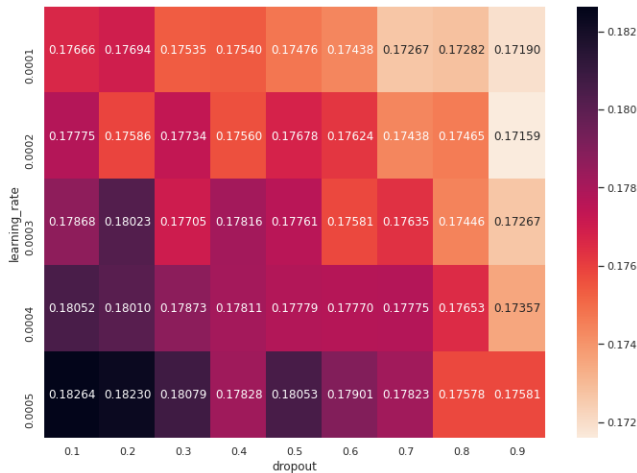


Figure 4: Results of Grid Search (First Pass)

From the results, it can be seen that the model trained with a learning rate of **0.0002** and a dropout rate of **0.9** achieved the highest performance by having the lowest average validation loss of **0.1716**.

4.4.2 Second Pass. The grid search was then repeated using a smaller resolution of values around the values obtained in the first pass of the grid search:

$$lr \in [0.000125, 0.00015, 0.000175, 0.0002, 0.000225]$$

$$dropout \in [0.86, 0.88, 0.90, 0.92, 0.94, 0.96]$$

After performing the grid search, the average validation loss of the model was recorded and plotted on a heatmap in Figure 5 below:



Figure 5: Results of Grid Search (Second Pass)

From the results, it can be seen that the model trained with a learning rate of **0.00015** and a dropout rate of **0.9** achieved the highest performance by having the lowest average validation loss of **0.1702**. These hyperparameters are thus selected for use for the rest of the experiments.

4.5 Overcoming Imbalanced Classes

From Section 2.1, we have identified that the dataset is slightly imbalanced where some classes have more training examples than others. While the imbalance is quite small as compared to other classification tasks / datasets (e.g., credit card fraud detection), we experiment various strategies used for classification on imbalanced data to determine if they are effective in improving the performance of our model.

4.5.1 Class Weights. A simple way of overcoming the problem of imbalanced classes is to use "class weights", which is to give a weighting for each class. The under-represented classes would have a higher weight and vice-versa. This information is then fed to the model during training which will cause the model to "focus"

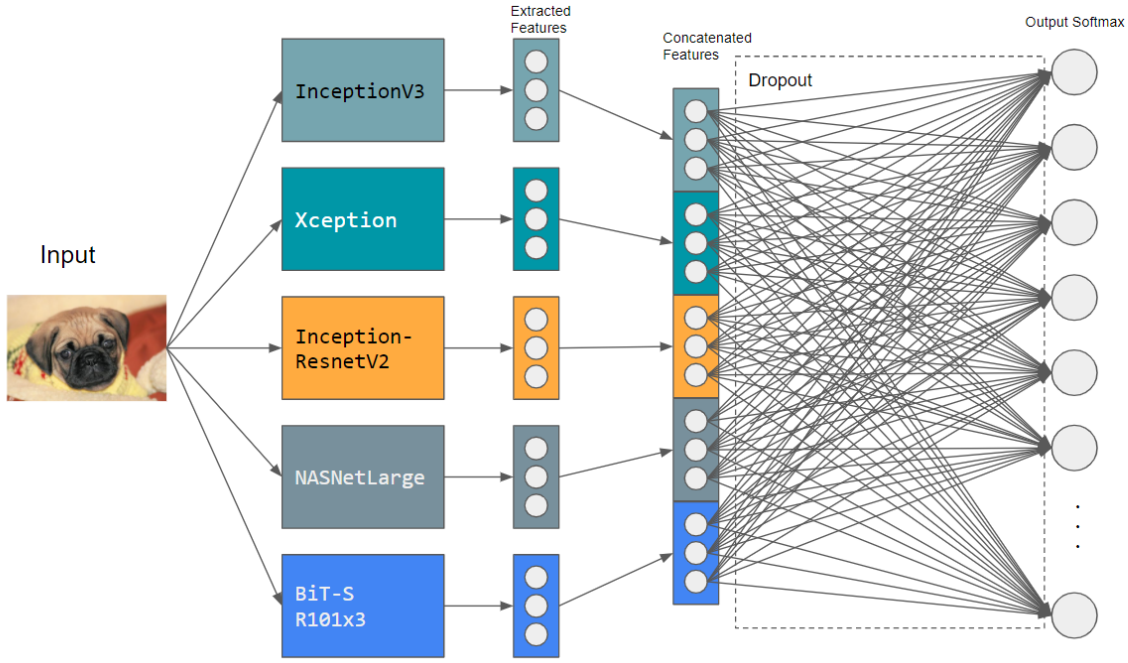


Figure 3: Architecture of the ANN built on top of the concatenated features

more on the examples from the under-represented classes. The class weights can be calculated using the following formula:

$$w_c = \frac{n_samples}{n_classes * n_samples_c}$$

This can be easily done without having to modify the training set by adding or removing training examples.

4.5.2 Random Under/Oversampling. The training dataset can also be balanced by performing Random Undersampling, where the over-represented training examples are removed at random such that there is an equal amount of training examples of each class. While the training examples would be balanced, valuable training information may be lost in this process. Similarly, Random Oversampling can also be performed on the training dataset to re-sample the under-represented training examples with replacement such that the number of training examples are the same.

4.5.3 Edited Nearest Neighbour Undersampling. The Edited Nearest Neighbours (ENN) is an approach of resampling proposed by Wilson [13] which aims to remove the majority class examples that are near the decision boundary. This is done by performing k-nearest neighbours classification on the examples of the majority classes and removing those that are misclassified as part of the minority class.

4.5.4 Synthetic Minority Over-sampling Technique (SMOTE). SMOTE is an approach of generating synthetic training examples of the minority classes proposed by Chawla, Bowyer, Hall, & Kegelmeyer [2]. The synthetic training examples are generated by joining a minority example to k neighbours belonging to the same class with

a straight line and generating the synthetic examples along those lines.

4.5.5 SMOTEENN. SMOTEENN is a resampling strategy proposed by Batista, Prati and Monard [1], which is essentially combination of SMOTE and ENN by first performing SMOTE then using ENN for data cleaning to produce better clusters of class examples.

4.5.6 Performance Evaluation. The strategies mentioned in Section 4.5.1 to 4.5.5 were implemented and trained with the model with the holdout cross-validation. This is such that the resampling strategies would be applied to the same training examples and the models trained would be evaluated with the same validation set. The validation losses and prediction accuracy were obtained and the performance of the models are shown in the table below:

Strategy	Average Loss (Cross-Entropy)	Prediction Accuracy
Base Model	0.1620	95.0%
Class Weights	0.1608	95.2%
Random Undersampling	0.1690	94.8%
ENN	0.2075	94.0%
Random Oversampling	0.1627	94.9%
SMOTE	0.1612	94.9%
SMOTEENN	0.1776	94.7%

Table 3: Comparison of performance of models trained from different resampling/ class weights

From Table 3, it can be seen that by applying information about the class weights to the model for training, the trained model yielded

the best performance as compared to the other resampling technique and the baseline model. Therefore, this strategy is implemented in favour of other resampling strategies for the implementation of our final model.

4.6 Model Selection

With the results obtained from performing the experiments conducted in Sections 4.1 - 4.5, the final model implemented for our task is the following:

- Feature Extractors: Concatenated features from pre-trained models (InceptionV3, Xception, Inception-ResNetV2, NAS-NetLarge, BiT-S R101x3)
- Dropout drop probability: 0.9
- Learning Rate: 0.00015 (with annealing)
- Optimiser: Adam
- Loss function: Multi-class log loss
- Training Epochs: 500 (With early termination)
- Class distribution information fed to model during training (Class Weights)

5 KAGGLE EVALUATION SCORE AND RANK

5.1 Evaluation Score

The model chosen in Section 4.6 was trained with the training dataset with holdout cross-validation (70% train, 30% validation) and the model was used to predict the breeds of the dog images in the test set provided by the Kaggle competition. The predictions of the test images are output to a .csv file and was uploaded to Kaggle for evalation and the score that was obtained was **0.16846**.

Submission and Description	Private Score	Public Score
submission.csv 8 minutes ago by yushao Class Weighted	0.16846	0.16846

Figure 6: Kaggle Evaluation Score

5.2 Evaluation Rank

With the score of 0.16846, our model's performance was ranked 182nd in the Kaggle competition's leaderboard out of 1281 participants, therefore our submission is within the **top 15%** of the submissions for this competition.




180	—	sda		0.16325
181	—	andy23		0.16420
182	—	Eugene Ware		0.16860

Figure 7: Kaggle Leaderboard Position

6 CONCLUSION

In this project, we have successfully implemented a classifier for predicting the breed of a dog given a photo for the Kaggle competition. We have also managed to learn about Transfer Learning

and have successfully applied it by utilising State-of-the-Art pre-trained models for feature extraction and further improved the performance of our model by tuning the hyperparameters used for the training of the model.

We have also managed research and learn about the different techniques used when performing classification on an imbalanced dataset and applied it to further improve the model's performance.

As a result, our implemented model was able to achieve top 15% in the competition's leaderboard.

REFERENCES

- [1] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. 2004. A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data. *SIGKDD Explor. Newsl.* 6, 1 (June 2004), 20–29. <https://doi.org/10.1145/1007730.1007735>
- [2] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research* 16 (Jun 2002), 321–357. <https://doi.org/10.1613/jair.953>
- [3] François Chollet. 2017. Xception: Deep Learning with Depthwise Separable Convolutions. arXiv:1610.02357 [cs.CV]
- [4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV]
- [5] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. 2020. Big Transfer (BiT): General Visual Representation Learning. arXiv:1912.11370 [cs.CV]
- [6] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. 2015. Microsoft COCO: Common Objects in Context. arXiv:1405.0312 [cs.CV]
- [7] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Trans. on Knowl. and Data Eng.* 22, 10 (Oct. 2010), 1345–1359. <https://doi.org/10.1109/TKDE.2009.191>
- [8] Sebastian Ruder. 2019. *Neural Transfer Learning for Natural Language Processing*. Ph.D. Dissertation. National University of Ireland, Galway.
- [9] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. <https://doi.org/10.1007/s11263-015-0816-y>
- [10] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. 2016. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. arXiv:1602.07261 [cs.CV]
- [11] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going Deeper with Convolutions. In *Computer Vision and Pattern Recognition (CVPR)*. <http://arxiv.org/abs/1409.4842>
- [12] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2015. Rethinking the Inception Architecture for Computer Vision. arXiv:1512.00567 [cs.CV]
- [13] D. L. Wilson. 1972. Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. *IEEE Transactions on Systems, Man, and Cybernetics SMC-2*, 3 (1972), 408–421. <https://doi.org/10.1109/TSMC.1972.4309137>
- [14] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning Transferable Architectures for Scalable Image Recognition. arXiv:1707.07012 [cs.CV]