Dog Breed Classification Using Deep Neural Network
Texas State University
Spring 2019

Aniq Makhani
a_m1377
a_m1377@txstate.edu

Abstract

Dog breed classification fall under fine-grained image classification problem. What makes it so challenging is that the inter-class variations are so small that even a little part of image considered could effect the whole classification result. The images trained on ImageNet have large inter-class variation which makes it easier to classify an image correctly. My main aim here is to apply deep learning techniques to deal with this fine-grained image classification.

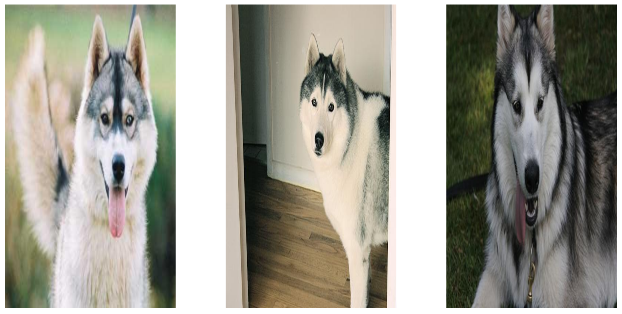
(a) Eskimo Dog (b) Husky (c) Malamute

Figure 1: Similar dog breeds

1. Introduction

CNN has been by far the most popular deep learning classifier used for images. CNN includes different deep layers which extract features both high level and low level. CNN also includes a final output layer which classifies the result into categories. CNN is very efficient algorithm in terms of classifying objects which have large inter-class variation. With small inter-class variation it becomes harder for the CNN classifier to extract features from the images and classify them into different categories.

Figure 1 shows three different breeds of dog, clearly they all look similar, but they belong to different breeds. This kinda of fine-grained classification problem makes it harder for classifiers to classify dog breeds. It will be an interesting task to see how well can CNN classify images of different dog breeds.

2. The Dataset

The dataset contain images of 120 breeds of dog and annotated with the label and bounding boxes. In total there are 20580 images of dogs. The dataset was collected from kaggle website where it was published as an open source data. This dataset is a specific example of fine-grained classification problem where dog breeds share similar features which makes it harder to classify. The images are saved in a particular folder for each breeds. The name of each breed folder is given i.e n02085620-Chihuahua. The number is kind of an identifier which is useless and will have to be removed. The images are all in different sizes which needs to be re-sized.

3. Related Work

While searching for fine-grained problems I saw many interesting and challenging datasets and some approaches that will help deal with this challenging task. There are so many other fine-grained dataset available such as Caltech UCSD Birds [6], Oxford flower [4], FGVC Aircraft datasets [3] e.t.c. If we look at these datasets individually we can see that inter-class variation is really small with little change in shapes and color across the same class.

Several approaches were explored for fine-grained image classification. A traditional method is to extract features using different descriptor extraction algorithm and train these features on a linear classifier. Khosla et. al[2] who created this stanford dog dataset were able to achieve 44% accuracy. They used SIFT descriptor for classification. The current highest accuracy was achieved by using selective pooling vectors, which encodes descriptors into vectors, and selects only those that are below a certain threshold of quantization error, with respect to the codebook which is used to approximate the nonlinear function f used to determine the classification likelihoods of various classes. One paper used R-CNN to categorize Caltech's UCSD birds dataset which makes use of CNN for both part localization and classification.

Also, gnostic field [1] were used to handle fine-grained image classification. Gnostic field was developed by Christopher Kanan in 2013. He implemented this algorithm on Stanford Dogs dataset and achieved really good accuracy. He developed a size and shape invariant model which doesnt suffer from bias when dealing with images in different size.

4. Methods

The data was first preprocessed and then I used data augmentation and transfer learning to train the classifier

4.1. Data Preprocessing

The first step in every Deep learning project is to preprocess data in a way that can be given to a classifier as an input. Before diving into the details about preprocessing let me explain how data was provided to me. The data had two parts one the actual images and second its annotation which includes class label and bounding boxes. Firstly, I used os library to import data into our python notebook. Each breed had its own folder. After importing all the data, a new directory was created to save cropped and re-sized images. All images were re-sized to 224 by 224 and were stored as RGB. Once all the files were stored I imported that directory into my notebook and stored it in a dictionary where class label was used as key and all the images as values. Furthermore, that dictionary was broken down into two arrays one for all the images and second for its label. The images were preprocessed in a way that it was re-sized to 74 by 74 and stored as RGB. The labels were also preprocessed to get the name of the breed from the string. The label were split with a '-' to separate the number with the breed name in the string, and that name was saved in the label array respective to its breed.

The labels were then changed into one hot encoding and then into categorical because machine learning algorithms cannot deal with categorical data and require input in numerical format.

Lastly, after getting both the images and its label, they were sent to a function called train_test_split which separates the data into training and testing data. first the data was split as 80-20 training and validation data and then from that 80% training data I split the data into 65-35 training and testing data.

4.2. Transfer Learning

Transfer learning is a machine learning method where a model trained for one task is used as a starting point for an another task.

This approach is really popular among fields related to computer vision and natural language processing where pretrained models are used as a starting point since training a model and all the computation require lots of resources and time to develop a model on large datasets.

There are two approaches to transfer learning:

1. Develop Model Approach
2. Pre-trained Model Approach

I decided to use pre-trained model approach because for deep learning task we need huge amount of data and pre-trained models are already trained on images so it will be really helpful for me to use pre-trained models as a starting point.

The research organization who develop models on a really large and challenging dataset usually open source their model for the whole community of machine learning to use. Without those models it would take days or weeks to train our model using current hardware resource.

Keras provides many pre-trained models which are already packaged into keras and can be easily accessible by just importing into the python environment. Some of the pre-trianed model I used are google inception model, xception model, resNet, VGG16, deseNet. All of these models are trained on a huge dataset of images provided in the ImageNet challenge which has over tens of millions of images.

4.3. Data Augmentation

Data Augmentation means to increase the number of images using several different methods. Why do we need data augmentation? The answer to this question is pretty simple, as we dont have enough amount of data to train a model we augment the data to make our dataset bigger.

There are so many ways we can manipulate an image to increase the size of the dataset. We can flip, rotate both horizontally and vertically, scale an image to make it smaller or bigger, we can also crop the image to focus on a particular sample point, translation can be done to move an object in an image, Gaussian noise is also very useful method which adds noise in an image.

Data augmentation helps in overcoming overfitting because as we generate more data the classifier will extract more useful features and improve the training of our model.

5. Architecture

5.1. Keras

Keras provides a neural network API, which runs on top of TensorFlow, Theano or CNTK. Keras can be run on both CPU and GPU. Keras provides user-friendly environment which enables user to create and experiment deep learning models very quickly and efficiently. Keras was build and maintained by Francois Chollet. Keras is really famous among machine learning community for its simplicity and easy to make models.

5.2. Pre-trained Model

Keras application provides us with the pre-trained models alongside their weights. These models can be used for prediction, feature-extraction and fine-tuning.

Some of the pre-trained models that I used in this projects are

1. InceptionV3 [5]
2. VGG16
3. ResNet50
4. Xception
5. DenseNet
6. InceptionResNetV2

5.3. Neural Network Model

To train a model we need a huge amount of data which is nearly impossible if you have limited amount of data available. In order to train the model we need a pre-trained model which is already trained on hundreds and thousands of images. We can then use that pre-trained model and add additional layer for out own classification.

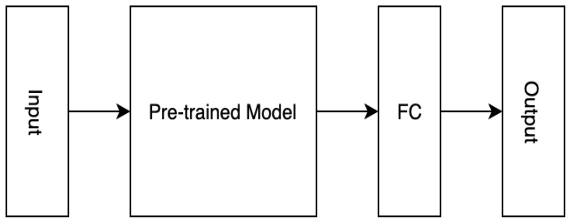

Figure 2: Architecture

Figure 2 shows my architecture of the model. The input is given to the backbone model which is the pre-trained model. In the next step I added three fully connected layers and an output layer to get the classification result. The three fully connected layer have 'relu' as an activation function and dropout of 0.5.

6. Training and Testing

The input was given to the model as 75 by 75 and with channel 3 which means RGB. The model was then then trained on the training data that we splitted. The learning rate was set to 1e-4 which was set after several trial and error. To train my model we used epochs to calculate the loss. Epoch is an approach where one epoch is equal to one forward pass and one backward pass. The epochs were set to 500. In this project batch size was set to 64.

Keras provides callback function if know how to use it can be really useful. Callbacks provides the view of internal state of the model and gives statistics during training.

I used callbacks for learning rate and early stop. Early stop callbacks monitor the val_loss score and if it increase or remains same for the given patience it stops the training. Learning rate plateau reduce the learning rate if the monitored value, which is val_loss, doesn't improve.

While training all the models I had to do several tuning of hyper parameter to get more accuracy. It was found that if we increase the learning rate the model was less accurate while keeping the learning rate 0.0001 the model performed its best. I had to spent several days finding this learning rate because the model takes time to train and it had to be compiled again to see the accuracy changed with respective to the change in learning rate.

When I started training the model on my laptop it kept going on for hours without producing any output.

After several days of failing to get any results, I found out that this problem won't run on my laptop due to huge number of computations. So I had to opt for GPUs which was freely available on kaggle and since this dataset was taken from kaggle it was easier for me to import the dataset into the kernel.

7. Experimental Result

Several pre-trained were imported from the keras library. These models are all pre-trained on huge image dataset provided by ImageNet. I'll discuss the result one by one for each pre-trained model and compare which model performed better on Stanford Dog dataset.

VGG16 VGG16 took approximately 10 min to train the model. After 20th epoch the model stopped training. The total evaluation accuracy was at 0.98% and loss was at 4.78.

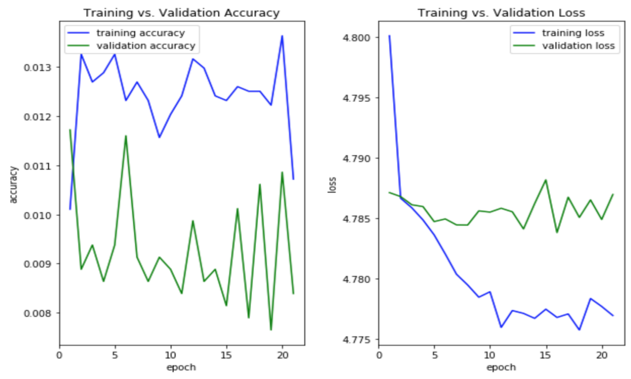

Figure 3: VGG16 curves

InceptionV3 Inception took approximately 20 mins to train the model and it stopped early as the error rate was not decreasing. The model stopped at 40 epochs and the accuracy was 32% with loss at 2.59.

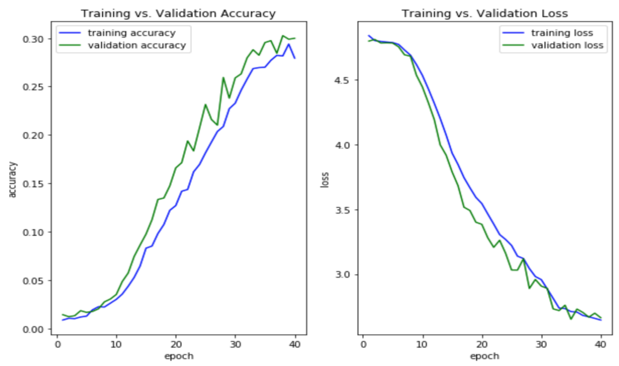

Figure 4: Inception curves

ResNet50 ResNet performed really well on the dataset provided. As shown in the Figure 5 the model stopped training at 43 epoch. The resNet50 gained an accuracy of 49% and loss at 2.06.

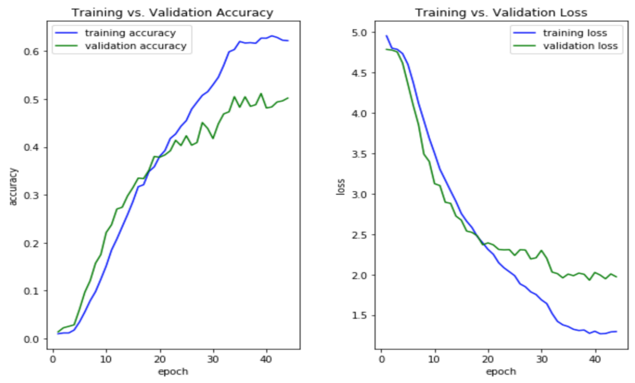

Figure 5: ResNet50 curves

Xception Xception is the best performing model among all of the other model that I trained. The model stopped after 38 epochs and achieved a really good accuracy of 54% and loss of 1.86.

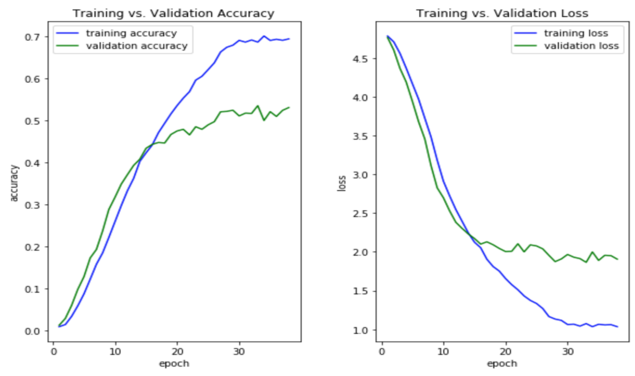

Figure 6: Xception curves

DenseNet121 DenseNet took total of 40 mins to train the model. The model was stopped at 49th epochs as there was no progress in accuracy. The final accuracy was 51% with loss of 1.90.

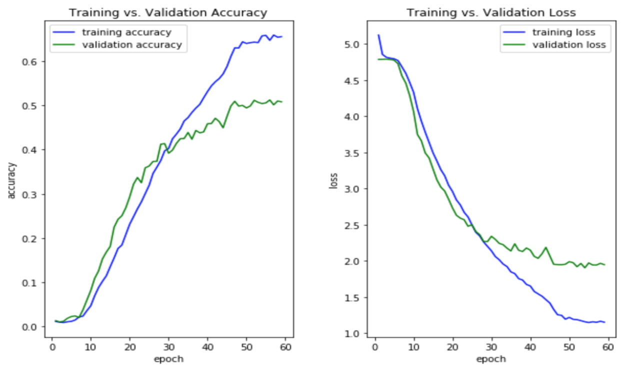

Figure 7: DenseNet curves

InceptionResNetV2 InceptionResNetV2 took 30 mins to train the model on Stanford Dog dataset. The model took only 35 epochs and then it stopped training. The accuracy I got on this model is 45% with loss at 2.12.

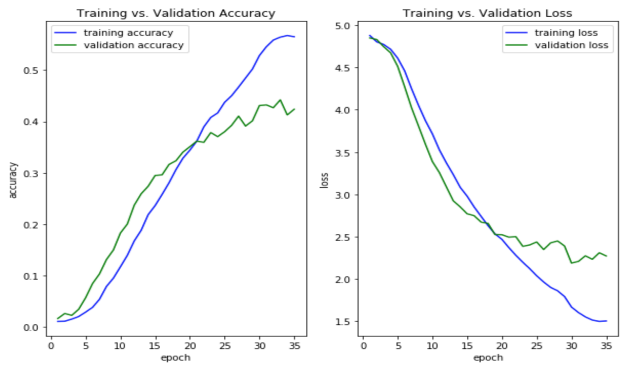

Figure 8: InceptionResNetV2 curves

Figure 8 shows the accuracy and loss with respect to the number of epochs. We can also observe that the accuracy started to go downwards after 35 epochs which is why it was unnecessary to do 500 epochs.

According to the Table 1 xception performed so much better than the other models except denseNet which scored second best accuracy. Interestingly VGG16 didn't perform at all and scored worse with only 1% accuracy.

Model	Accuracy	Loss
VGG16	0.98%	4.78
InceptionV3	32%	2.59
ResNet50	49%	2.06
Xception	54%	1.86
DenseNet121	51%	1.90
InceptionResNetV2	45%	2.12

Table 1: Results

8. Conclusion

Fine-grained image classification is a really challenging task because they have small inter-class variation which makes it harder for a classifier to extract distinguish feature and classify them separately to their respective category. Stanford Dog dataset is of the FGIC problem where different species of dogs have very similar features like skin color, face structure and many more. The main ain of of project was to use transfer learning and use pre-trained models and compare them to see which model performs better. Out of all the models Xception performed better which achieved an accuracy of 54%, but DenseNet is not very far behind which got an accuracy of 51%. One interesting thing I found is that VGG16 couldn't perform at all and got an accuracy of 1% which is very less than all the other models. This project showed that transfer learning helps gain much more accuracy than building our own classifier which will eat all of our resources.

The code can be found **here**.

References

[1] C. Kanan. Fine-grained object recognition with gnostic fields. In *IEEE Winter Conference on Applications of Computer Vision*, pages 23–30. IEEE, 2014.

[2] A. Khosla, N. Jayadevaprakash, B. Yao, and F.-F. Li. Novel dataset for fine-grained image categorization: Stanford dogs.

[3] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. 2013.

[4] M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008.

[5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions.

[6] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. the caltech-ucsd birds-200-2011 dataset. 2011.