
Transfer Learning for Dog Breed Identification

Yisheng Ji
y3ji@ucsd.edu

Zhongke Ma
z2ma@ucsd.edu

Qichao Zheng
q5zheng@ucsd.edu

Yizhou Chen
yichen@ucsd.edu

Abstract

In this project, we achieve classification of dog breeds from Stanford Dog Dataset which contains images of 120 dog breeds. DenseNet, VGG19 and InceptionV3 were used to extract the features. To get a better accuracy, we apply transfer learning on different networks such as VGG19 and InceptionV3. These networks were already built on Keras, which is a deep learning library and provides some popular neural networks. The result of this project can be used in identification of dogs in images.

1 Introduction

1.1 Motivation

Dog have become more and more important in our daily life, they appear as pets, transportation, drug detector and so on. Since different breed of dogs can help people in different way, the identification on breed of dogs have become commonly mentioned nowadays in many fields. Thus, we need a more accurate and efficiency way to classify the breed of dogs.

1.2 Convolutional Neural Network

Deep learning models have achieved remarkable results in computer vision in recent years. [1] As one of the deep learning networks, Convolutional Neural Network has widely used around the world for image classification. The CNN consist of neurons and the weights and biases of the neurons can be trained. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. Convolutional layer, pooling layer, and fully-connected layer are the three main types of layers of Convolutional Neural Network.

In order to improve the performance of the network, CNN model nowadays has become deeper. However, as CNN become deeper, the cost of time and computing resource also become larger. Thus, to save time and make our calculation more efficient, we will use a pre-trained model in our project. Pre-trained model is a model already trained for some other task, we can re-train them in order to fit our task.

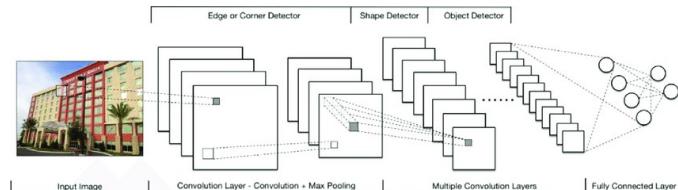


Figure 1: Basic structure of CNN.

2 Description of methods

In this project, we will adopt several CNN models for the purpose of image classification. The past ten years have witnessed great success in the performance of deep network handling the problem of image classification. And there have been several great framework or architectures successful achieve below the 5% top five error rate. Such as the **DenseNet**, **ResNet** and **Inception**. The former two methods adopt the concepts of *shortcut connections*, while the latter one utilize the manner of *factorizing convolutions*. The purpose of these methods is intuitive, by reducing the parameter of deep network so as to allow for the network to grow deeper and deeper.

Therefore, in this project, we wish to explore the difference between these methods. Namely, we wish to see the difference of *a) shortcut connections* and *b) factorizing convolutions* in performing large scale, multi-labeled image classifications. Unlike the basic classification problems, where the objects being classified varied in a wide range. In this project, we wish to examine how well the deep network can handle the classification of similar objects. Given that those target objects of the same category can somehow varied in a wide range.

Further, we compare the performance of **DenseNet** and **Inception V3**, together with an earlier framework **VGG19**, to examine the performance under different deep network.

2.1 VGG19

VGG is a network of increasing depth using an architecture with very small convolution filters. With small convolution filters, the number of parameters can be reduced. The input of VGG is 224*224 RGB images. Then the input image will go through the convolutional layers, where filters with a small receptive field will be applied. Each convolutional layer has different depth, some of the convolutional layers will have a max-pooling layer followed, and the max-pooling layer has a 2×2 pixels window with stride 2. With max-pooling layers, VGG can achieve spatial pooling. After the stack of convolutional layers three fully connected layers are followed. The first two layers have 4096 channels each, the third have 1000 channels. And the last layer of the network is a soft max layer. After the input goes through the whole network, the output will be classification contains 1000 channels. VGG16 and VGG19 are the two most common models of VGG. VGG16 has 16 weight layers which has about 138 million of parameters in total, while VGG19 has 19 weight layers which has about 144 million of parameters in total. [4] The model we will use in our project is VGG19, it can achieve 7.3% top 5 error in ILSVRC.

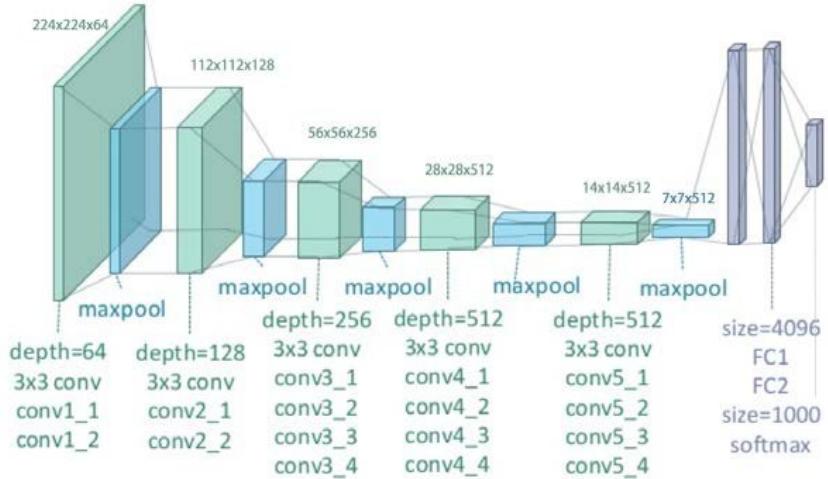


Figure 2: VGG19 architecture.

2.2 DenseNet

More and more models have shown that convolutional neural networks can become substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close

to the input and those close to the output. Thus, Dense Convolutional Networks (DenseNet) was introduced. DenseNet consists of dense block, within each dense block, each layer is connected to every other layer in a feed-forward fashion. [3] In DenseNet, every layer is connected together by dense connectivity in order to have max information flow between the layers which means every layer have multiple inputs from other layers.

In DenseNet, fewer parameters is needed, which means the parameters in DenseNet can be more efficiency than traditional neural networks. Besides, DenseNet can be deep while the layers can be slim. With all these advantages, DenseNet is easily to be trained.

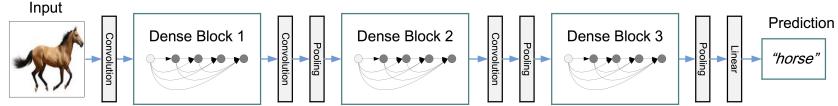


Figure 3: A deep DenseNet with three dense blocks.

2.3 InceptionV3

Inception is a deep convolutional neural network which provides a new state of the art for classification and detection. It makes a better use of the computing resources inside the network and makes the network deeper and wider while the computing budget is not affected. Inception networks consist of Inception modules. The main idea of the Inception architecture is to consider how an optimal local sparse structure of a convolutional vision network can be approximated and covered by readily available dense components. [5] To find the optimal local sparse structure, we assume that each unit of the previous layer refers to some part of the input image, which can find the clusters that focused on some particular region, thus, these clusters can be summed up by a 1×1 convolution in the following layer. And to avoid patch-alignment problem, the filter size of Inception has only 3 options: 1×1 , 3×3 and 5×5 . To keep the computing complexity stable, Inception network uses dimensionality reduction which can reduce the feature maps from 192 to 16.

InceptionV3, as the improved model of Inception networks, has 42 layers, which is much deeper than GoogLeNet, the first model of Inception networks. The improvement InceptionV3 make is that they applied factorization of convolutions and improved normalization in the networks. [6] Comparing with GoogLeNet, the top 5 error of InceptionV3 is improved from 6.67% to 4.49%.

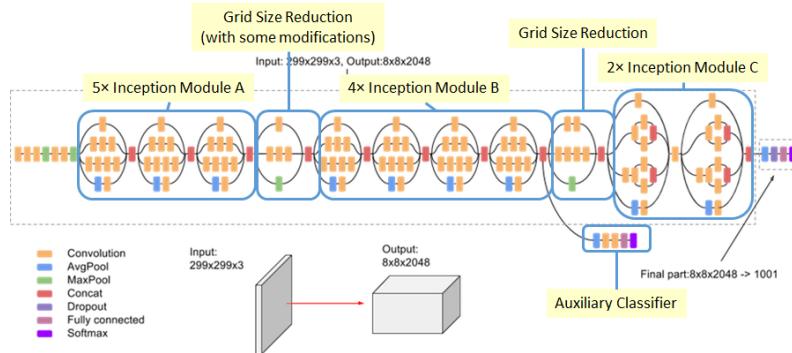


Figure 4: InceptionV3 architecture.

2.4 Transfer learning

Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task. In traditional CNN for image classification, usually the earlier layers are to detect edges, while deeper layers will try to detect some specific features. With transfer learning, we can use the earlier layers to re-train the deeper layers. The main benefit transfer learning is saving training time without hurting the performance. Also, with transfer learning, we can use less data to train the neural networks.

3 Implementation Details

In this section, we will briefly introduce how we implemented and modified these three deep network mentioned above.

3.1 Models

Basically, we first examine the performance of these three pretrained network on our testing set; then we remove the final layers (top layers) within the pretrained models, plugged a new fully-connected layers as the new classifier. During training, we first frozen the parameters of previous layers, since they all have been properly trained and fine-tuned, and tend to achieve optimal performance on general classification problems. These previous layers, then functioned as feature extractor, extracting the features from the input image tensors, generating a group of feature vectors, passing through the last fully connected layer, and fire on some specified output nodes indicating the predicted categories of the input images.

Table 1 demonstrates one of the prototype of the deep network that will be adopted in our project.

Table 1: A typical modified framework for our project.

layer	function	input dims	output dims
0	<i>input images</i>	(n, 3, 224, 224)	–
...	–	–	–
n – 1	<i>extract deep features</i>	–	(n, n, n, 2048)
n	<i>global average pooling</i>	(n, n, n, 2048)	(n, 2048)
n + 1	<i>drop out</i>	(n, 2048)	(n, 2048)
n + 2	<i>fully connected</i>	(n, 2048)	(n, 120)

3.2 Dataset

As for the dataset, we have 12,000 samples for training and 8,580 samples for testing. We further adopted the method of data augmentation to enhance the numbers of the training set. More specifically, we adopt some random *shifts*, *zooms*, *rotates* and *flips* to transform the origin image, so as to obtain the augmented data for training. This method, does somehow improve the performance of our model.

3.3 Losses

As for the loss function, we basically use both the *categorical_crossentropy* and *sparse_categorical_crossentropy* that are typically adopted in classification problem. Also, by some literature review, we have also tried to adopt the *center loss* \mathcal{L}_C , as a regularized term for the training of our model, which is found to be significantly useful in classifying similar targets. Therefore, we define the loss of our model as

$$\mathcal{L} = \mathcal{L}_S + \lambda \mathcal{L}_C \quad (1)$$

where \mathcal{L}_S is the softmax loss and \mathcal{L}_C is the center loss defined by

$$\mathcal{L}_S = - \sum_{i=1}^m \log \frac{e^{W^T x_i + b_i}}{\sum_{j=1}^n e^{W^T x_j + b_j}} \quad (2)$$

$$\mathcal{L}_C = \frac{1}{2} \sum_{i=1}^m \|x_i - c_i\|_2^2 \quad (3)$$

4 Experimental Settings

4.1 Datasets

The dataset we used in this project is Stanford Dogs Breed Dataset, which contains images of 120 breeds of dogs from around the world, such as Chihuahua, Japanese spaniel and Rhodesian ridgeback.

This dataset has been built using images and annotation from ImageNet for the task of fine-grained image categorization. There are totally 20,580 images in this dataset, with 12,000 images in the training set and 8,580 in the testing set.

For training-validation split rate, we use 1/6 and introduce data augmentation methods including left/right shifting ± 0.2 , scaling 1 ± 0.2 , rotation ± 30 and horizontal flip. The training, testing and validation data in a batch scale of 32 samples per batch, and in total we have 315 batches for training, 60 batches for validation and 269 batches for testing. Figure 5 shows 10 sample images from the dataset with class label of *Border Collie*.



Figure 5: Sample Images from the Stanford Dogs Breed Dataset. Category =*Border Collie*

4.2 Training Parameters

Undoubtedly, the most important and most difficult part of training a deep network, is how to properly choose the 1) layers, 2) losses and 3) optimizers. All these staffs are characterized by parameters and closely related with the performance of the whole network. Table 2 demonstrates some of the parameters that we adopted for training the plugged layers.

Table 2: A typical modified framework for our project.

plugged layer	activations	optimizer	step / epoch	loss	early stopping
fc64, fc120	ReLU, Softmax	adam + sgd	120	\mathcal{L}_S	<i>true</i>
d0.2, fc120	–, Softmax	sgd	80	\mathcal{L}_S	<i>true</i>
d0.4, fc120×2	–, Softmax	adam + sgd	120	\mathcal{L}_S	<i>true</i>
fc64, fc120	GLU, Softmax	adam + sgd	120	$\mathcal{L}_S + \mathcal{L}_C$	<i>true</i>

As for finetuning, we adopt a two-step finetune manner. Namely, we fine-tune the network for two stages, while in the first stage, we use the less augmented data, with only left/right shifting ± 0.1 and scaling 1 ± 0.05 . And for the second stage, we use the non-augmented data, simply set all augment manner to *false*. As for the optimizer, we again, utilize sgd, with a learning rate of 0.0001 and momentum of 0.9. For each stage, we run for 200 epochs with a step size of 120 batches per epoch, together with early stopping if the validation loss start to increase.

4.3 Hardware & Framework

For the whole project, we use the *Keras* along with the *Tensorflow* backend, running on both the UCSD's MLDSP Clusters equipped Nividia GeForce 1080Ti and our both PC with GeForce 1060. To avoid the problem of packages' version compatibility, we adopted the official *tensorflow*'s docker.

5 Results

5.1 VGG19

As the forerunner of deep network in image classification, *VGG19* does somehow introduce some very important concepts, such as *going deeper*, *reduce parameters*. It is very basic model in understanding

Figure 6: Results of Top- k Accuracy Verses k index

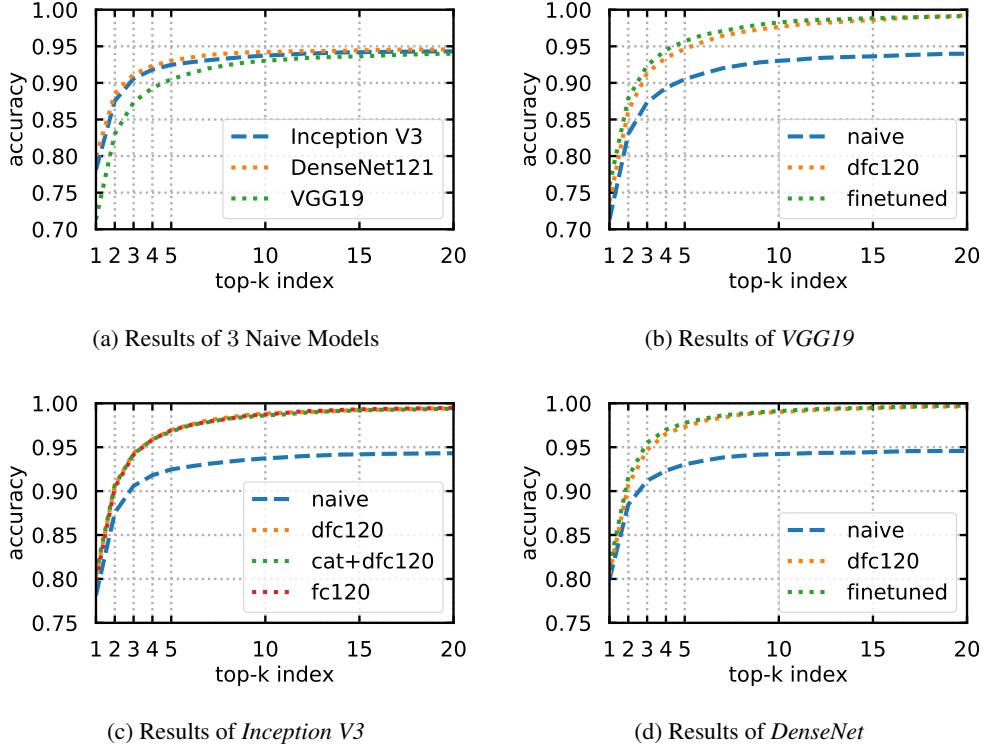


Table 3: *VGG19* Plugged Layers

layer	input dims	output dims	comments
<i>fc</i>	$(n, 25088)$	$(n, 4096)$	<i>fixed</i>
<i>dropout</i>	—	—	$r = 0.5$
<i>fc</i>	$(n, 4096)$	$(n, 120)$	<i>predictions</i>

how *convolutional neural network* can be experted in classifying images. Therefore, we first examine the performance of *VGG19* under our experimental settings as described in Section 4.

The orginal *VGG19* network have 19 weighted layers and with the last flatten layers given a output of $(1, 25088)$ feature vector. Followed by 3 fully connected layers with 4096, 4096 and 1000 outputs each. This however, for our dataset with onlt 120 classes, seems a litte trival. Table 3 described the details of our modified *VGG19* network.

5.2 InceptionV3

We further explored the effects different plugged layers based on *Google’s Inception V3* model. Results of those methods are ploted in the form of *top- k accuracy verse k index*, as in Figure 6c

Noted that, by implementing *fully-connected* layers with / without *dropout* layers can results in significant difference, this is because the neural networks are, by natural, likely to be overfitted, especially in case of multi-layer classifier. Also, we further noted that introducing the *center loss*, as described in Section 3.3, canot enhance the performance of the network very much, while significant increase the difficult of both implementation and trainning, therefore we will not include the results here.

The performance of our modified version of *Inception V3*, have significant improvement, as shown in Figure 6c, increase the top-5 accuracy from approx. 0.92 to approx. 0.97.

5.3 DenseNet121

Bsed on the conclusions from *InceptionV3*, we modified and trained the *DenseNet121* network with only plugged *dfc120* layers. Cause it can amlost achieve the same performance among the other architecture without introducing more parameters. The final *Dense Block* outputs a tensor with only 1024 dimensions,

Figure 6d shows the performance of our modified *DenseNet* model.

5.4 Comparision

In Figure 6, we evaluate the performance of different models by accuracy curves. When we use the naive pre-trained models to predict the dogs' breeds, the *Inception V3* model and *DenseNet121* model's accuracy are close to each other and both of them perform better than the *VGG19* model.

After we change the classification layers of each model, all of their performance increase. For each model, the accuracy at top-5 increases about five percentage. We also fine tuned each model to achieve higher accuracy as much as possible. We can see from Figure 6b, 6c and 6d that after fine tuning, the *DenseNet121* model has the highest accuracy, InceptionV3 follows behind and VGG19 performs worst among the three models.

5.5 Success Cases

In order to visualize the prediction result of the model, we show some successful cases. Some of these dogs' appearance are unique, which leads to high prediction accuracy. Results of these visulaizations are shown in Figure 7.

5.6 Failures

In fact, some of the dog images of different classes within the dataset have almost the "same" appearance, even a human without specified knowledge of the thoes dogs cannot discern between them. This, however, is the major cause of our deep network making mistakes. Typically, under such cases, the outputs of the final layer of our network tend vote for multi-classes with almost equal scores. This is the resson why the top-1 result is much low than top-2, 3 and larger. As shown in Figure 8, we illustrate this problem by showing 3 misclassified pairs in columns.

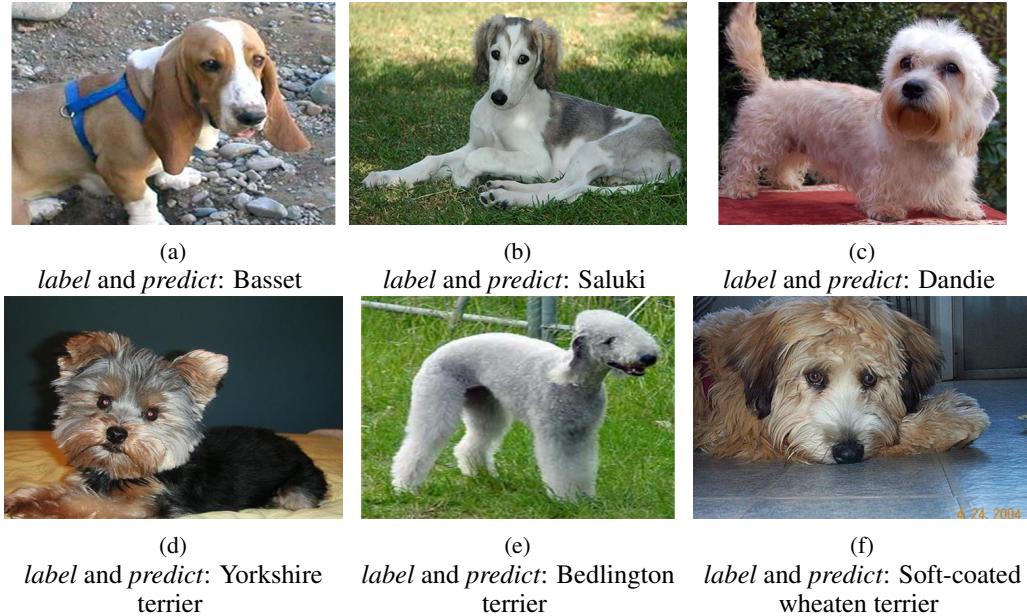


Figure 7: Examples of Success Cases shown in Columns.

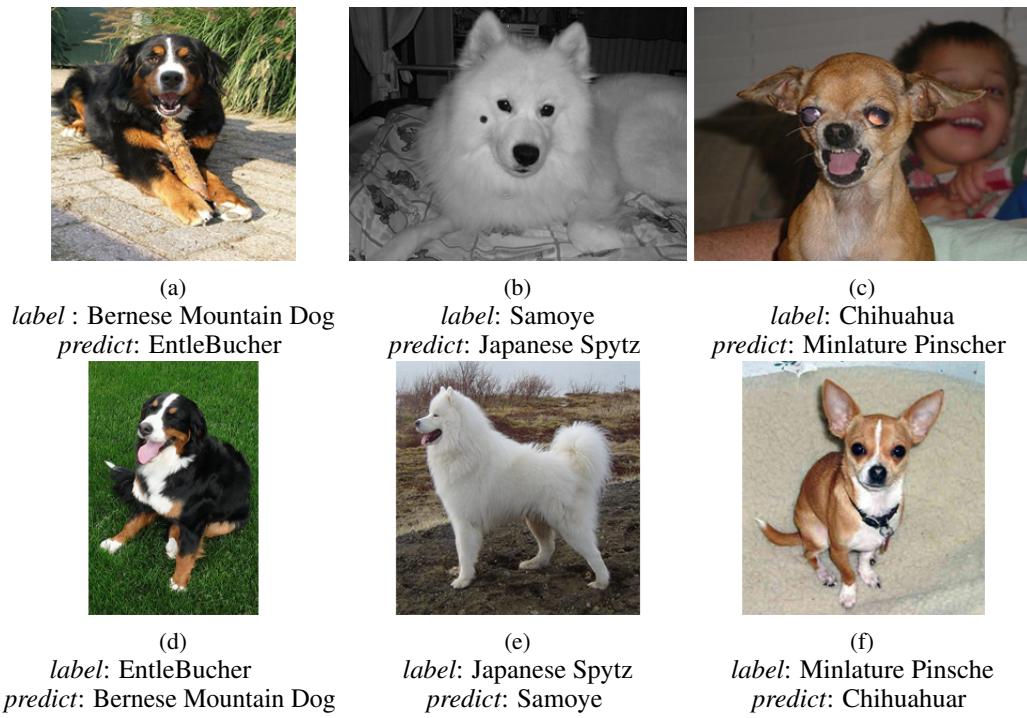


Figure 8: Examples of Failure Pairs shown in Columns.

6 Discussion

6.1 Inference

Before we implement the VGG19 model, Densenet121 model and InceptionV3 model, we have a approximate estimation of the performance of these three models. Taking into consideration the architecture of the three models, we predicted that Densenet121 had the best performance and VGG model might behave worse than the other two models.

6.2 Conclusion

From Figure 6 we can draw a conclusion that in naive conditions (Figure 6a), the VGG19 model has the lowest accuracy compared with the other two models. And after we change the last layer of the three models by a classifier with 120 classifications, the order of the accuracy remains the same. To be detail, DenseNet121 has the best performance, InceptionV3 just similar to DenseNet121 and VGG19 behaves worst in the three models. The results of the experiment just proved what we have predicted before we implemented the project.

We can also notice from the pictures Figure 6b, 6c and 6d. that after we change the classifier and fine tune the last few layers, the performance of each model increases to some extent. This is because that the model is more specialized in dogs' breeds classification and therefore, it will have better performance than the more generalized one.

6.3 Learning

During the process of doing project, we have learnt much that we can never learn from books. Before we did the project, we just had an abstract concept of what machine learning is doing and why it was so powerful. We looked forward to put what we have learnt into practice and the project was just a good opportunity. We learnt much about TensorFlow, Keras and many other packages that can greatly improve the efficiency of establishing a convolutional neural network. We also learnt how to designed a classifier. And the most important thing is that we had a deeper understanding of the architecture of different models. We learnt step by step and found that the world of machine learning is so interesting, so powerful that we are willing to discover the fields of machine learning in the future.

6.4 Challenges

For the purpose of classifying the breed of dogs properly, several challenges exist to be solved:

- 1 there are many breeds of dogs in the world, which means the computer vision algorithm needs to take into consideration a great number of categories. And based on No-Free-Lunch-Theorem, with the number of dogs' breeds increasing, the accuracy of dog classification will decrease to some extent;
- 2 some breeds of dogs have very similar appearance, such as Samoye and Japanese Spitz, Bichon Frise and Teddy dog, Chihuahua and Minlature Pinscher, etc. Some of these dogs are even too similar for human beings to distinguish from one another. Therefore, how to accurately classify these similar dogs is a big challenge;
- 3 the third lies in the fact that we only have access to limited number of training images per class, which means we need to make the best use of the images to extract the features of different kinds of dogs. Therefore, we need to find a good way to process the original images to get as much information as possible.

6.5 Future Work

In this project, we have tried three different models and evaluated their performance on the data set, from which we have the result of which model is good at recognizing different kinds of dogs and which is not. However, there are still many things that we can do in the future to improve the efficiency and accuracy of classifing dogs. The first thing we can do in the future is that we can combine the prediction result of the three models together and give out an 'mean' prediction. Since

different models may extract different features of the same image, we can choose the best result by 'majority rule' or just choose the classification with the largest probability. Another thing we can do is to collect more images of dogs to enlarge the training set. Since the number of training data is limited, the accuracy of all three models are not very satisfying. With more data, we are sure we can improve the accuracy much more.

7 Appendix

7.1 Github Repo Link

Github Repo (https://github.com/acetanil/ECE285_Final.git)

7.2 Github Repo Readme

ECE285_FInal

Description

=====

In this project, we achieve classification of dog breeds from Stanford Dog Dataset which contains images of 120 dog breeds. DenseNet, VGG19 and InceptionV3 were used to extract the features. To get a better accuracy, we apply transfer learning on different networks such as VGG19 and InceptionV3. These networks were already built on Keras, which is a deep learning library and provides some popular neural networks. The result of this project can be used in identification of dogs in images.

Requirements

=====

TensorFlow version: 1.12.0 keras version: 2.1.6-ft

Run Time

=====

launch-tf-gpu.sh

Code organization

=====

src/evaluate_naive.ipynb	-evaluate the naive conditions of three models
src/inceptionV3.ipynb	-train, finetune the InceptionV3 model and evaluate its performance
src/Densenet121.ipynb	-train, finetune the Densenet121 model and evaluate its performance
src/VGG19.ipynb	-train, finetune the InceptionV3 model and evaluate its performance
src/main.ipynb	-try different losses and find the most proper one
Stanford_Dog_Breed	-The data set we used to train and evaluate

References

- [1] Kim Y. Convolutional neural networks for sentence classification[J]. arXiv preprint arXiv:1408.5882, 2014.
- [2] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.
- [3] Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks[C]//CVPR. 2017, 1(2): 3.
- [4] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.
- [5] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 1-9.
- [6] Szegedy C, Vanhoucke V, Ioffe S, et al. Rethinking the inception architecture for computer vision[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 2818-2826.