Multiple Deep Learning Method Comparison on Dog Breed Identification Problem

Sihan Wang
Department of Electrical and Computer Engineering
Boston University
Boston, MA 02215
`shwang95@bu.edu`

Yutong Gao
Department of Electrical and Computer Engineering
Boston University
Boston, MA 02215
`gyt@bu.edu`

Tianheng Hu
Department of Electrical and Computer Engineering
Boston University
Boston, MA 02215
`tianheng@bu.edu`

Abstract

In this project, we trained four models to identify the breed of a dog from a given image. Our training was based on Stanford Dogs Dataset and ImageNet Dataset. We implemented ResNet-50, VGG-16/19, Inception V3 and Xception to find an optimal solution to identify the breed of the dog. The model comparison is discussed in the result part.

1 Introduction

When we meet a cute dog on the street, one question usually comes to our mind: what kind of dog is that? We leverage machine learning skills to help people solve this question.

We choose Stanford Dogs Dataset to train our model. The Stanford Dogs dataset contains images of 120 breeds of dogs from around the world. This dataset has been built using images and annotation from ImageNet for the task of fine-grained image categorization. This dataset contains 20,580 images with class labels and bound boxes.

Small networks like CNN fall short for data with large variations. The number of neurons becomes enormous. In that case, we need deep networks. Since we do not have GPU to train with and the Stanford Dogs Dataset is kind of small, we decided to pretrain the model by training it on a different task than the goal task. We choose ImageNet Dataset to perform the pretraining.

ImageNet is an image dataset organized according to the WordNet hierarchy. Each meaningful concept in WordNet, possibly described by multiple words or word phrases, is called a *synonym set* or *synset*. There are 21841 synsets in WordNet, majority of them are nouns (>80,000). In ImageNet, on average 650 images are provided to illustrate each synset. Images of each concept are quality-controlled and human-annotated. Many researchers use ImageNet to pretrain their models and extract features from images.

In order to find an optimal solution to identify the breed of the dog, we trained four neural networks, ResNet-50, VGG-16/19, Inception V3 and Xception.

2 Models

In the project we tried several different models such as ResNet, VGG, Inception and Xception. Based on the pretrained models in Keras, we were able to efficiently train the network and compare the results of them.

2.1 ResNet-50

ResNet is short for deep residual network which is introduced in paper *Deep Residual Learning for Image Recognition*. (He et al., 2015) In 2014, VGG-16 and VGG-19 is already considered as very deep network but it is not deep enough. In the research based on ImageNet, people find that the depth of the network is crucial point for better results. However, with the network depth increasing, accuracy gets saturate and then degrades rapidly. Microsoft research addressed the degradation problem by introducing a deep residual learning framework. In Figure 2, it shows a residual block of the network. Each block can can be expressed as: $\mathbf{y} = \mathcal{H}(\mathbf{x}) + \mathcal{F}(\mathbf{x}, W_l), \mathbf{x} + 1 = \mathcal{F}(\mathbf{y})$.

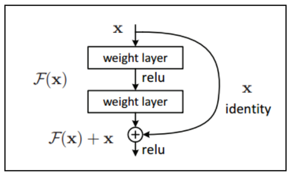
x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 1: Residual Block.

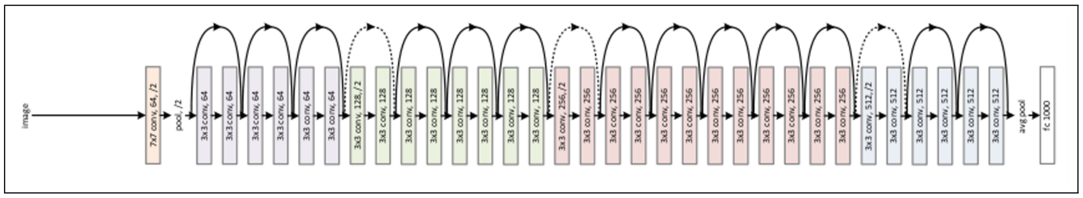

Figure 2: Architecture of ResNet-50.

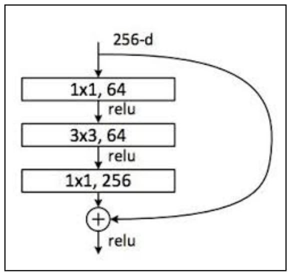

Figure 3: A *Bottleneck* Building Block for ResNet-50.

In figure 3, it shows the original structure of ResNet-50. It uses a 3-layer block instead of normal 2-layer block. The three layers are 1×1, 3×3, and 1×1 convolutions, where the 1×1 layers are responsible for reducing and then increasing dimensions, leaving the 3×3 layer a bottleneck with smaller input/output dimensions. The identity shortcuts $\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$ is directly used when the input and output are of the same dimensions. When the dimensions increase, the projection shortcut in $\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_{sx}$ is used to match dimensions. When the shortcuts go across feature maps of two sizes, they are performed with a stride of 2.

What we need to do is to resize the input data to 224×224 and we need to transfer the last 1000 fully connected layer to a 16 fully connected layer.

2.2 VGG-16/19

VGG is short for the *Very Deep Convolutional NetWorks* which is suitable for processing large-scale image dataset. The dog dataset we used is a subset of ImageNet which is a famous image dataset contains more than 100,000 synsets in WordNet and each synset provide on average 1,000 images. Download the dataset from Kaggle and we can find each image is a colorful image with hundred-pixel-width and length. So VGG should be a good solution to do the dog breed identification which is actually image classification problem.

In Very Deep Convolutional Networks for Large Scale Image Recognition, we can learn the detailed structure of VGG network. (Simonyan and Zisserman, 2014) Figure 1 is a visualization of the VGG architecture.

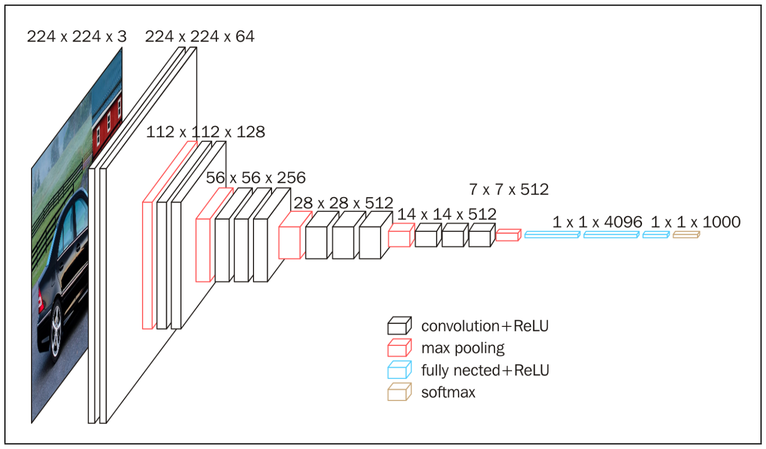

Figure 4: A Visualization of the VGG Architecture.

The training dataset is in a fixed size of $224 \times 224 \times 3$. 3 stands for the three channels of RGB image. So we need to preprocessing the dataset, what we need to do is to resize the image to 224×224 before we use it. Instead having so many hyper parameters, VGG network is a much simpler network which focuses on just having convolutional layers that are 3×3 filters with stride 1 and always use the same padding, and make all the max pooling layers 2×2 with a strid of 2. VGG-16 network has a total of about 138 million parameters and VGG-19 has a total of about 144 million parameters. A stack of convolutional layers which has a different depth in VGG-19 and VGG-16 is followed by three fully-connected layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels. We used the pre-trained model in Keras but the final classes of our problem is 16 rather than 1000 so we need to change the network. We removed final fully-connected layers from that model then use the remaining portion of the model as a feature extractor for our smaller dataset. Extracted features are called "Bottleneck Features" and we use these features to train a new small fully-connected network to assign the input image to 16 classes.

In Table 1 we can find the structure of different depth VGG network. D stands for VGG-16 and E stands for VGG-19. The main difference between VGG-16 and VGG-19 is the number of weight layers and the performance of them are similar.

Table 1: Comparison of VGG Family.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

2.3 Inception

In 2014, a 22-layer deep network (Szegedy et al., 2014) - *GoogleLeNet* won the ILSVRC challenge, bringing down the error rate to 6.67%. The name seems a tribute to LeNet.

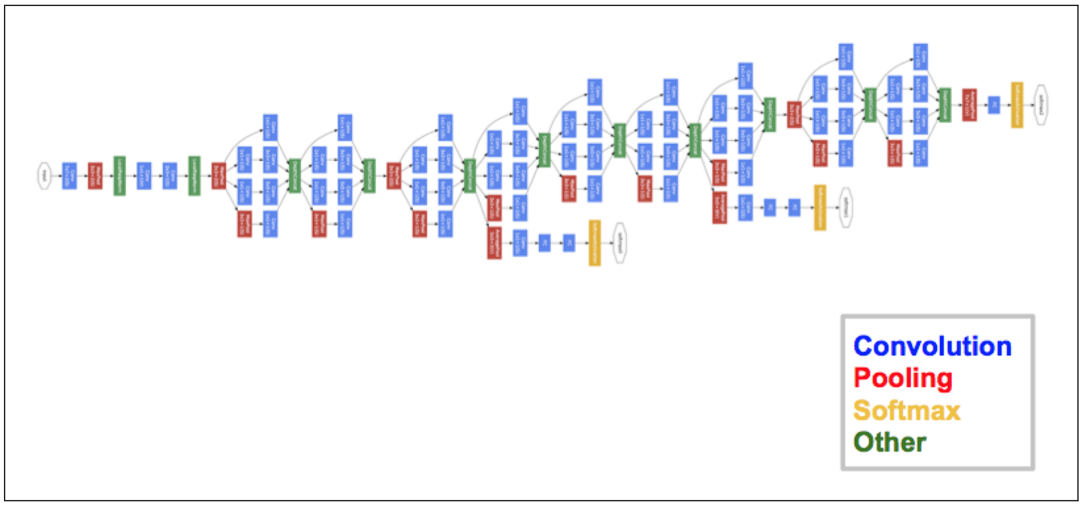

Figure 5: Architecture of GoogleLeNet.

Before GoogleLeNet, the main development trend of network architecture is going deeper, i.e., more layers, and going wider, i.e., more neurons. But there are several cons of these methods:

1. When the training set is limited and many parameters are provided, overfitting will occur;
2. The larger the network is, the more complex the computations are, the harder the design will be;
3. While the amount of layers increases, the gradient starts to disappear.

Considering that the pursuit of accuracy is partly due to the fact that the design of the feature extraction module has not been able to extract the features of the image very well, if it can do some optimization on the basic feature extraction unit, then the optimized feature extraction module can be used to construct the network, and it will be beneficial to the final recognition. Thus, the Inception model is created.

2.3.1 Inception V1

The general convolution layer only increases the depth of the convolutional layer, while there is only one kind of convolution kernel on the single layer. For example, for VGG, the convolution kernel of a single layer has only a size of 3×3, so the feature extraction may be rather weak. The intention of GoogLeNet is to increase the width of a single convolutional layer, i.e., using a convolution kernel with different scales on a single layer convolution layer, and GoogLeNet constructs the basic unit of Inception module. The basic Inception module has a 1×1 convolution kernel, a 3×3 convolution kernel, a 5x5 convolution kernel and a 3×3 downsampling, this is named *Inception V1* model.

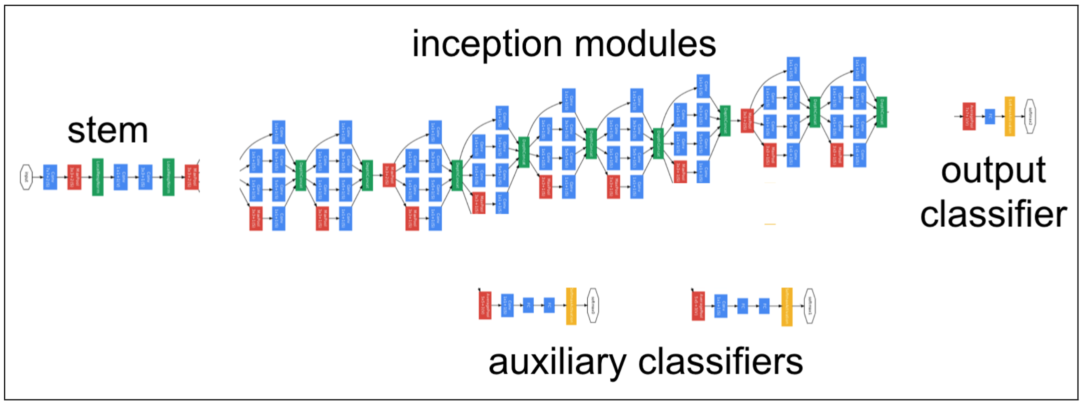

Figure 6: Architecture of Inception V1.

Thus, convolution kernels with different sizes can extract features of different sizes, and the feature extraction capability of a single layer is enhanced. There is a problem in the above Inception module architecture, the output of the previous layer is directly inputted into the next convolutional layer without any processing, so if the number of features of the previous layer is very large, the amount of computation will become very large. Thus, the Inception V2 model is designed.

2.3.2 Inception V2

The Inception V2 network has been improved on the basis of V1. First, adding the batch norm (BN) layer to reducing the Internal Covariate Shift, i.e., the changes in the data distribution of the internal neuron, and standardizing the output of each layer to a Gaussian of $N(0,1)$. Second, using two 3×3 convolutions instead of 5×5 in the module, not only reduces the number of parameters but also accelerates the calculation. (Szegedy et al., 2015)

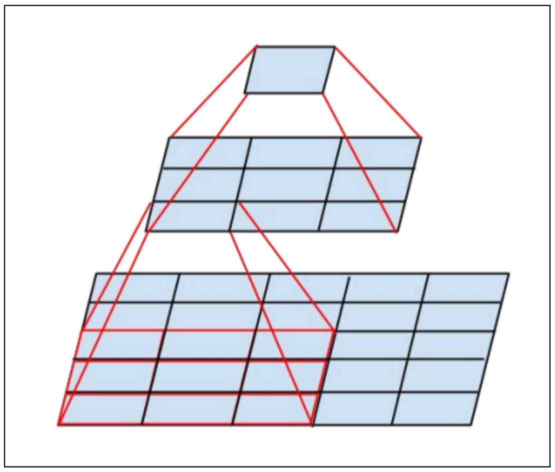

Figure 7: Changes of Inception V2.

So, after the correction of Inception module, before the output to the next layer of convolution, the data will be reduced by using the 1×1 convolution kernel, e.g., an original input with 256 feature maps will first do linear combination on the feature map using the 1×1 convolution, so that the output will only have 96 feature maps. In this way, no information loss happens, and the computation amount of the next convolutional layer is reduced. The revised Inception module is shown in Figure 8.

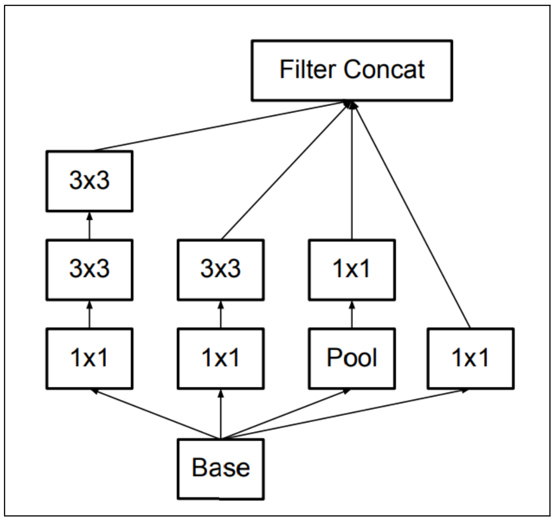

Figure 8: Architecture of Inception V2.

2.3.3 Inception V3

Design guidelines Inception V3 follows the following guidelines:

1. Avoiding to represent the bottleneck, i.e., the sizes of the feature maps should be reduced slowly, and the information flow obviously should not pass through highly compressed layers in the forward propagation process, i.e., the expression bottleneck. From input to output, the width and height of feature map will gradually decrease, but it can not become small at once. In addition, the output dimension channel will increase gradually, otherwise, the network will be hard to train. (The feature dimension does not represent the amount of information but as a means of estimation.)

2. High-dimensional representation can be replaced by the network and high-dimensional features are easier to handle. High dimensional features are more distinguishable, and training will be accelerated.

3. Space aggregation can be done on low-dimensional embedding without having to worry about losing a lot of information. For example, before 3×3 convolution, the dimensions of input can be reduced without serious consequences. Assuming that information can be compressed simply, training will speed up.

4. Balance the width and depth of the network.

Convolution decomposition with large size filters The accuracy gain of GoogleNet is mainly derived from dimensionality reduction, which can be regarded as a special case of convolution decomposition. We can reduce dimensionality before integration. The Inception module is fully convolutional, each weight value corresponds to a multiplication, and fast training can be done with the number of parameters reduced after the convolution decomposition. This can increase the size of the filter group to improve the precision.

Decomposing large convolutions into small convolutions 5×5 convolutions can be replaced by two layers of 3×3 convolutions, and the saving time can be used to increase the number of filters. For a region convolved by 5×5 convolution kernel, a 3×3 can be used to convolve in the first, then using a 3×3 convolution kernel to convolve the result, this is equivalent to 5×5 kernel. This reduces the number of convolution calculations, the original 5×5 kernel needs to do 25 additions and multiplications, and now the two 3×3 kernels only need to do 18 additions and multiplications. The lager the convolution kernel is, the greater the computation reduces.

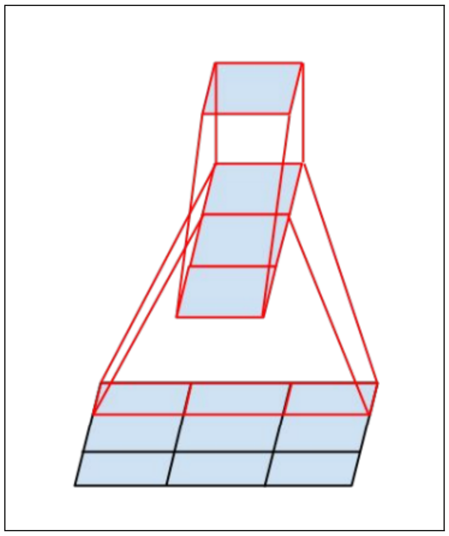
Figure 9: Changes of Inception V3.

Inception V3 also proposed another way of decomposition, a 3×3 convolution can first be convolved with a 1×3 kernel, followed by a convolution of a 3×1 kernel, this can also achieve the same output, which can also be processed for 5×5 and 7×7. This method can also clearly reduce the convolution kernel parameters and the amount of computation.

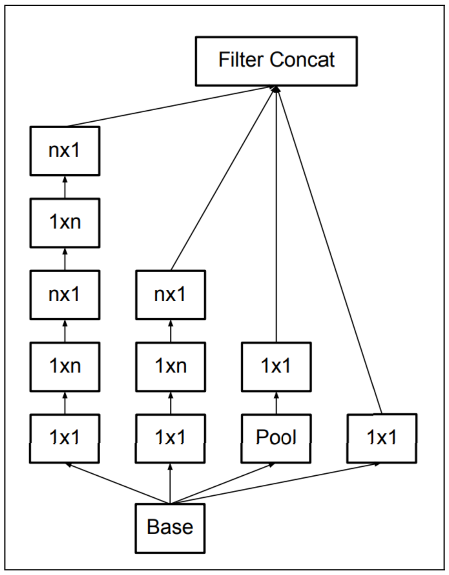

Figure 10: Architecture of Inception V3.

Note this the architecture is not very effective in the previous layers, but the effect is obvious to the middle layer of the size of 12~20.

Auxiliary classifiers The auxiliary classifier with BN layer has a better effect and gets its normalized role, similar work is done in GoogLeNet. The conventional network calculates the loss on the last layer, then do gradient direction propagation, but if the network layer is large, this leads to the phenomenon of gradient disappearance. An Auxiliary classifier is a classifier in the middle layer of the network. It calculates the auxiliary distortion and gradient, and then add the gradient to the gradient of the normal gradient. In this way the gradient disappearance is effectively solved. In fact, this approach echoes the method of solving the problem of gradient disappearance in ResNet, although the ResNet surface looks like the loss is calculated on the last layer, essentially the last layer of loss is actually in the subnetwork of ResNet. The calculation is equivalent to the way of adding loss layer to the middle layer in GoogLeNet.

Reducing the size of the feature map Pooling is used to reduce the size of the feature map, in order to avoid the bottleneck, i.e., to save the image information more effectively, the number of filters should be increased before the pooling. The previous network is adding a pooling layer to the convolutional layer to reduce the size of the feature map, but this way of reducing the size inevitably has the loss of information. Another way to reduce the size of feature maps is illustrated in the Figure 11.

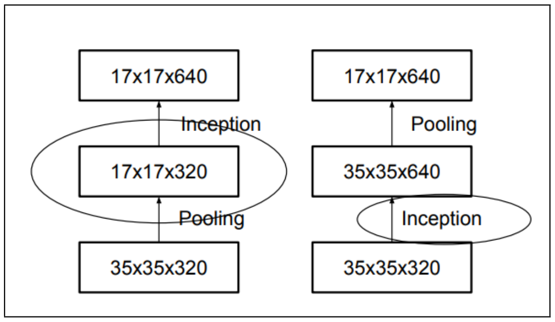

Figure 11: Way to Reduce the Size of Feature Maps.

Right part is a normal reduction, but the amount of calculation is very large. Left part performing pooling first leads to a bottleneck in feature characterization, violating the first rule mentioned above. In order to simultaneously achieve no rule violation and reduce the amount of computation, the network is changed into architecture shown in the following figure.

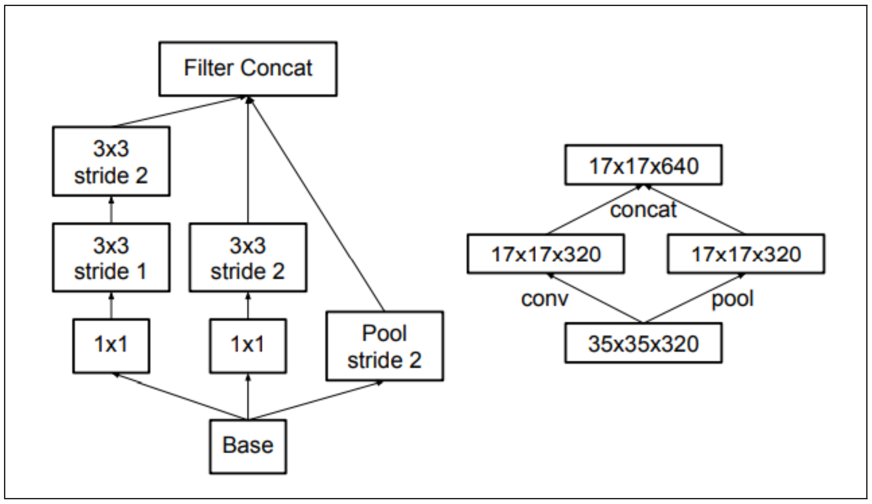

Figure 12: Architecture of Inception V3.

This method does convolution and pooling of length 2 at the same time, and then concatenate. From the figure, it can be seen that the reduction of size is realized with over convolution and the appropriate stride, and of course a branch is realized by pooling, which can effectively avoid the loss of information.

2.4 Xception

The Inception module is an intermediate state between ordinary convolution and deep separable convolution operation (convolution by point convolution). Based on this, the deep separable convolution can be understood as the largest number of tower Inception modules. Inspired by Inception, a new deep convolutional neural network structure called *Xception* (*extreme Inception*) is proposed, which uses depth separable convolution instead of Inception module. Xception is slightly better than Inception V3 on the ImageNet dataset and is obviously better in the larger picture classification dataset including 350 million pictures and 17,000 classes. Considering that the Xception structure and Inception V3 have the same number of parameters, this performance enhancement comes from the more efficient use of model parameters rather than increasing capacity. (Chollet, 2016)

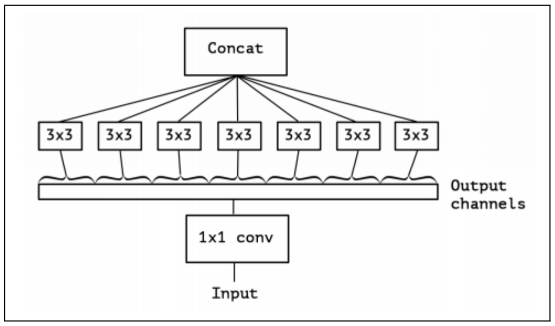

Figure 13: Architecture of Xception.

Figure 13 shows the extreme version of Inception module, cross-correlation is plotted using 1×1 convolutions, and then the spatial correlation of each output channel is drawn independently.

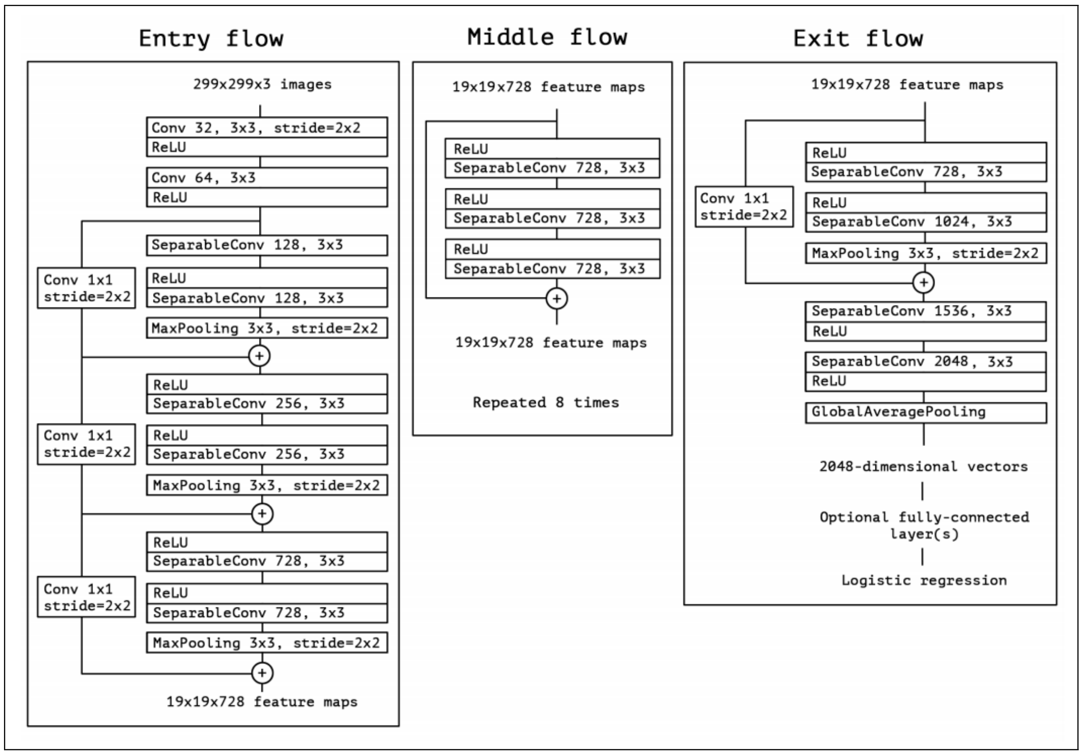

Figure 14: Full Architecture of Xception.

The Xception structure is composed of 36 convolutional layers, which form the basis of feature extraction. Our evaluation experiments only take image classification, so we use logical regression layer after convolution. Alternatively, a fully connected layer can be added before the logical regression layer. The 36 convolutional layers are divided into 14 modules. Except for the last one, there are linear residual connections among modules.

In short, the Xception structure is a linear stacking of depthwise separable convolutional layer with residual join. This is the structure that is very easy to define and modify; using a high-level library such as Keras or TensorFlow-Slim requires only 30~40 lines of codes, similar to VGG-16, but different from the complex and undefined structures of Inception V2 and Inception V3. Under the MIT certificate, Keras provides an open source implementation of Xception.

3 Result

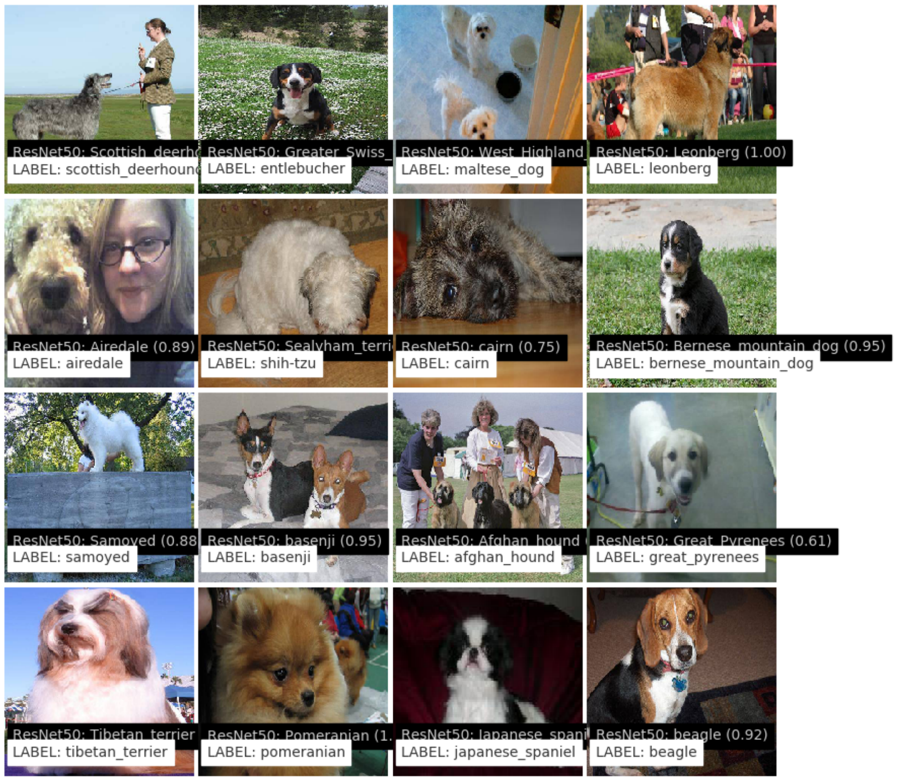

Figure 15: Sample Outputs of Our Models, ResNet-50 Method.

Figure 15 is a demonstration of the output of our models. The words in the black boxes shows what method is used, what breed is predicted, and the confidence of the prediction. The words in the write box shows the ground truth label.

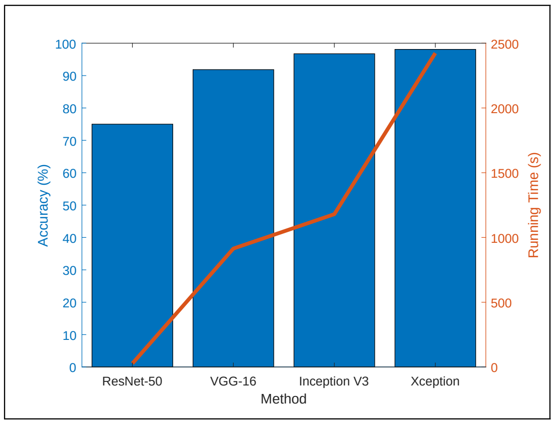

Figure 16: Accuracy and Running Time of Each Method.

Figure 16 shows the accuracy and running time of each method we use. Judging from the result we obtained, Xception has the largest accuracy with longest running time upon the Stanford Dogs Dataset. At the same time, the performance of VGG-16 and Inception V3 are both quite satisfactory. ResNet-50 achieves the fastest but least accuracy among all the models.

4 Conclusions

In this project, we leveraged our knowledge in machine learning to find an optimal solution to identify the breed of a dog from a given image. Among the four neural networks we trained, Xception performed best both in accuracy and efficiency. We achieved an overall accuracy of 97% on 16 categories of the Stanford Dogs Dataset, and the average processing period is 0.5s per image.

Acknowledgments

We appreciate Prof. Chin for his classes about machine learning, we learnt a lot from he. We also thanks to all of the teaching assistants of the course: Ken, Kieran and Gavin, their enthusiastic spirit was of great help.

References

[1] Simonyan, K. and Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. [online] Arxiv.org. Available at: https://arxiv.org/abs/1409.1556 [Accessed 10 May 2018].

[2] He, K., Zhang, X., Ren, S. and Sun, J. (2015). *Deep Residual Learning for Image Recognition*. [online] Arxiv.org. Available at: https://arxiv.org/abs/1512.03385 [Accessed 10 May 2018].

[3] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (2014). *Going Deeper with Convolutions*. [online] Arxiv.org. Available at: https://arxiv.org/abs/1409.4842 [Accessed 10 May 2018].

[4] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z. (2015). *Rethinking the Inception Architecture for Computer Vision*. [online] Arxiv.org. Available at: https://arxiv.org/abs/1512.00567 [Accessed 10 May 2018].

[5] Chollet, F. (2016). *Xception: Deep Learning with Depthwise Separable Convolutions*. [online] Arxiv.org. Available at: https://arxiv.org/abs/1610.02357 [Accessed 10 May 2018].