Big Data Final Group Report

Group Members and each one's works

Zhihuan Xue (Input Data part, the last 3 Models(except LeNet) part, Model Ensemble part)
Lei Kan (Charts drawing, Data Augmentation part, the model LeNet part)
Tianyuan Zheng (Charts drawing, Report draft writing, Theory accumulation of the models,)

1.Introduction

In face recognition, programmers write a complete set of identification system in advance to tell the computer what faces should be. Computer only need to capture data by the camera, and then mark the images which contain similar identification information so as to achieve the result of "recognition".

Nowadays, there are many ways of identifying human face. Due to multiple species, the process of identifying faces of animals, such as cats and dogs is much more complex.

In our project, we are trying to find the method to **judge whether the image is the face of a cat or the face of a dog.**

This technique can be implied to animal insurance field. By referring to the database of animals, the insurance company can judge the uniqueness of certain animals to avoid the possibility of insurance fraud. What's more, the identification system can also provide some parameters of environment where animals live thus gives the technical support to insurance companies, governments and banks in diversified risk- preventing area.

2.Data

We downloaded the original dataset from Kaggle. The data set contains 12,500 images of dogs and 12,500 images of cats separately. The dataset is created by Microsoft Research which is used in a kaggle competition focused on Asirra(Animal Species Image Recognition for Restricting Access).

Subject to the hardware level, we randomly chose one fifth of the original data set, which means 2500 images of dogs and 2500 images of cats. Then we chose 2000 of them separately as the training set and the remaining 500 of them as our testing set. This method also guarantees the equal proportion of images of these two animals.

Then we use some **data augmentation** methods, such as rotation, shift, flip, shear, and zoom to create some new images based on the original training set. In order to keep the balance of number of dog images and cat images in the training set, we generate 10,000 new images of dogs and 10,000 new images of cats. So now we have 22,500 images of dogs and 22,500 images of cats as our training data.

After determining our training set and testing set, we label the images: "1" for dogs and "0" for cats. And we can use the mean pixel values to generate our average cat and dog photo.

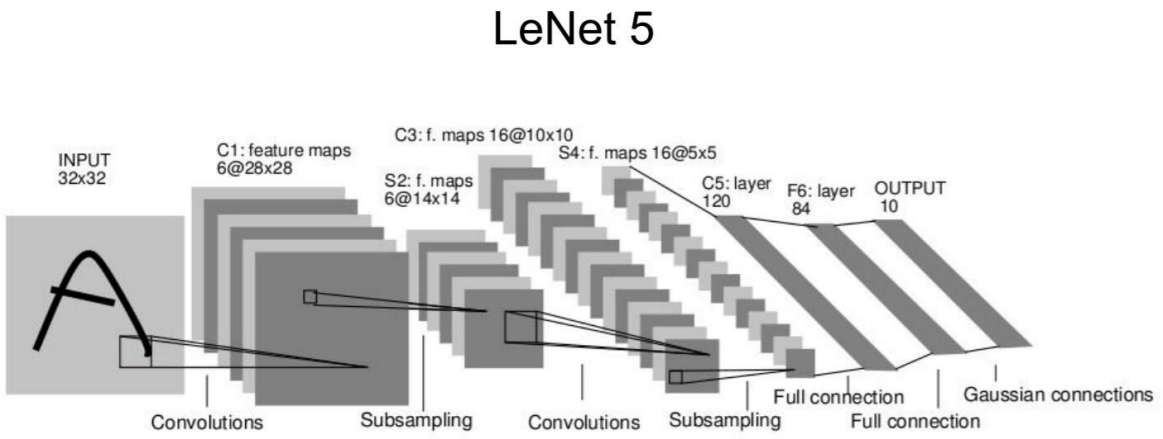
3.Approach：

We together use five neural network models to do the classification, including two kinds of VGGNet.

- LeNet

LeNet was created by Yan LeCun, a famous computer scientist focused on machine learning, who is also known as a founding father of convolutional nets in 1998. LeNet is a basic neural network with simple but useful structures. LeNet is widely used in commercial banks to distinguish handwriting on checks. Also, in our experiment, LeNet has obvious advantages both in time-costing and accuracy.

Traditional LeNet has the structure below. It has 2 convolution layers, 2 pooling layers and 2 connection layers, totally 7 layers. Convolutional filters are 5x5, applied at stride 1. Subsampling (Pooling) layers were 2x2 applied at stride 2. Architecture is **[CONV-POOL-CONV-POOL-FC-FC]**.

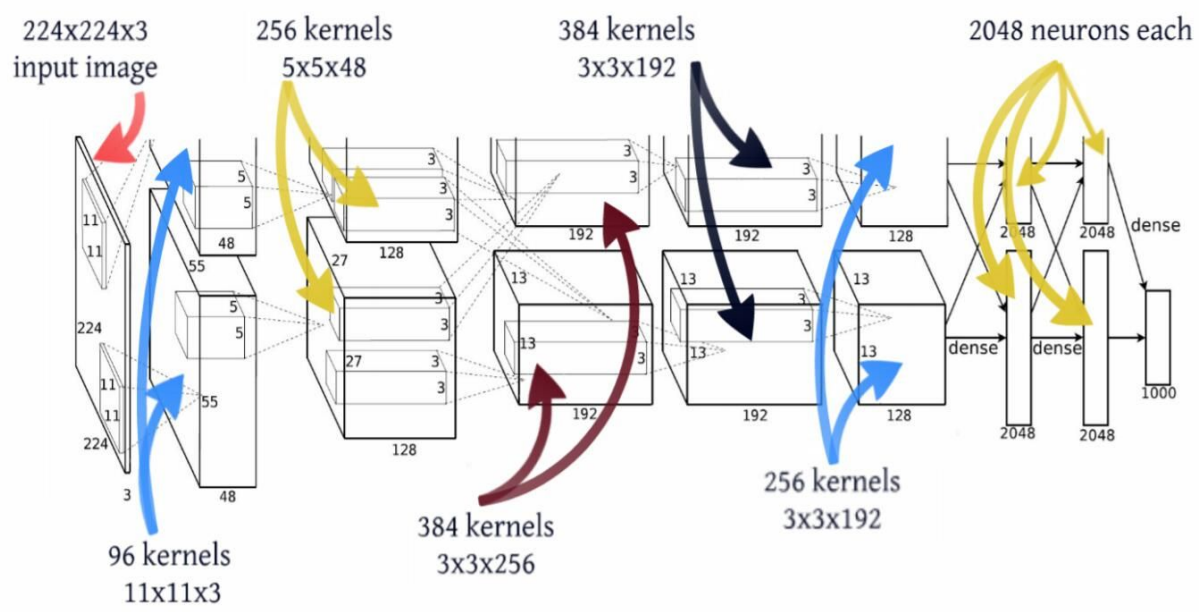


Source: <Gradient-based learning applied to document recognition>,Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner

- AlexNet

AlexNet was put forward by the SuperVision group, consisting of Alex Krizhevsky, etc to compete in the ImageNet Large Scale Visual Recognition Challenge in 2012[1]. They had very clearly advantage over the second team with 15.3% vs 26.2% error. The paper has been cited a total of 17,955 times and is widely regarded as one of the most influential publications in the field.This model is a pioneer of coming numerous CNN networks in the computer vision community. This was the first time a model performed so well on a historically difficult ImageNet dataset. AlexNet uses many effective methods to have more accuracy without overfitting, like data augmentation and dropout.

AlexNet has a structure like the below chart shows. The network was made up of 5 conv layers, max-pooling layers, dropout layers, and 3 fully connected layers. Architect is **[CONV1 MAX POOL1 NORM1 CONV2 MAX POOL2 NORM2 CONV3 CONV4 CONV5 Max POOL3 FC6 FC7 FC8]**. The network they designed was used for classification with 1000 possible categories. Compared with the simple LeNet, AlexNet has far more parameters to estimate. For example, for the first layer, it has 55*55*96 = 290,400 neurons, each has 11*11*3 = 363 weights and 1 bias, together has 290400 * 364 = 105,705,600 paramaters on the first layer alone.



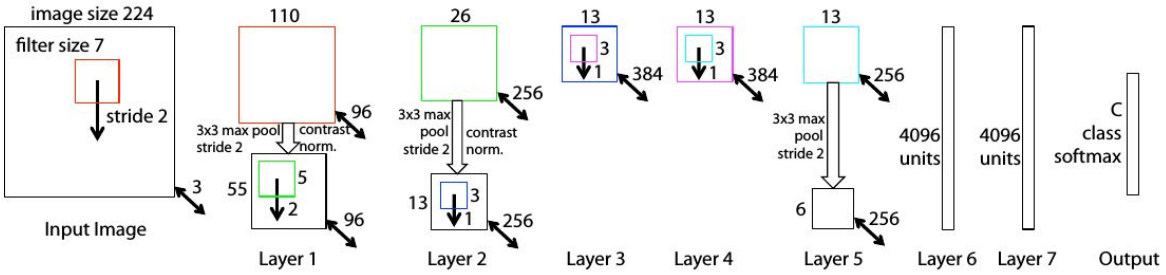
source: Imanol Schlag[2]

Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Source: cs231n[3]

- ZF Net

Compared with AlexNet, the architecture of ZFNet makes a fine tuning, but still developed some very keys ideas about improving performance. (1)In convolution layer 1, instead of using 11x11 sized filters, ZF Net uses filters of size 7x7 and a decreased stride value from 4 to 2 ;(2) In convolution 3,4,5, instead of 384, 384, 256 filters, using 512, 1024, 512. This result in decreasing error from 15.4% to 14.8%. The reasoning behind this modification is that a smaller filter size in the first conv layer helps retain a lot of original pixel information in the input volume. A filtering of size 11x11 proved to be skipping a lot of relevant information, especially as this is the first conv layer.ZF Net has the following architecture.



source:<Visualizing and Understanding Convolutional Networks>, M. Zeiler and R. Fergus

- VGGNet

VGGNet uses only 3x3 sized filters which is quite different from AlexNet's 11x11 filters in the first layer and ZF Net's 7x7 filters. The combination of two 3x3 conv layers has an effective receptive field of 5x5.VGG Net is one of the most influential model because it reinforced the notion that convolutional neural networks have to have a deep network of layers in order for this hierarchical representation of visual data to work.

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					

Source:<Very Deep Convolutional Networks for Large-Scale Image Recognition> , K. Simonyan and A. Zisserman

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

- Ensemble

Ensemble model combines multiple ‘individual’ models together to achieve superior prediction power. Basically, an ensemble is a supervised learning technique for combining multiple weak learners/ models to produce a strong learner.

- Data Augmentation

Have been mentioned in the **Data** part.

- Other Possible Approaches

To follow those famous models, we try our best to follow their structures. But some of the following may not be suitable for our question here, which is a binomial classification question. For example, the activation function of the last Dense Layer may be changed to “Sigmoid” and the loss function “binomial_crossentropy can be used to replace the loss function now.

Besides these possible adjustments of parameters. We can also use some other more complicated models like GoogleNet, ResNet and so on. But because of the limited time and poor computation ability of our computers, we don’t try them at the same time.

4.Data Analysis

As stated above, we decided to try our dataset on some very famous models in the past, which are LeNet, AlexNet, ZFNet and VGG13.

- Data Inputting Method

The first part of our codes are about data inputting. We didn’t use all of the data downloaded from Kaggle to train and test the performance of the 5 models because that may take too much time. We only use 20% percent from all the pictures(5000) and split 80%(4000) of it to be the train and 20%(1000) to be the test. We ensure we have same percent of cats and dogs (which means both 50%) in both the train and test.

- Model Structure Details and Performance Evaluation

To make it specific, you can see the exact structures of our 4 models below:

1.Lenet

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 60, 60, 32)	2432
max_pooling2d_1 (MaxPooling2	(None, 30, 30, 32)	0
conv2d_2 (Conv2D)	(None, 26, 26, 64)	51264
max_pooling2d_2 (MaxPooling2	(None, 13, 13, 64)	0
flatten_1 (Flatten)	(None, 10816)	0
dense_1 (Dense)	(None, 100)	1081700
dense_2 (Dense)	(None, 2)	202

Total params: 1,135,598
Trainable params: 1,135,598
Non-trainable params: 0

2.AlexNet

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 27, 27, 96)	34944
max_pooling2d_3 (MaxPooling2	(None, 13, 13, 96)	0
conv2d_4 (Conv2D)	(None, 13, 13, 256)	614656
max_pooling2d_4 (MaxPooling2	(None, 6, 6, 256)	0
conv2d_5 (Conv2D)	(None, 6, 6, 384)	885120
conv2d_6 (Conv2D)	(None, 6, 6, 384)	1327488
conv2d_7 (Conv2D)	(None, 6, 6, 256)	884992
max_pooling2d_5 (MaxPooling2	(None, 2, 2, 256)	0
flatten_2 (Flatten)	(None, 1024)	0
dense_3 (Dense)	(None, 4096)	4198400
dropout_1 (Dropout)	(None, 4096)	0
dense_4 (Dense)	(None, 4096)	16781312
dropout_2 (Dropout)	(None, 4096)	0
dense_5 (Dense)	(None, 2)	8194

Total params: 24,735,106
Trainable params: 24,735,106
Non-trainable params: 0

3.ZFNet

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 29, 29, 96)	14208
max_pooling2d_6 (MaxPooling2	(None, 14, 14, 96)	0
conv2d_9 (Conv2D)	(None, 7, 7, 256)	614656
max_pooling2d_7 (MaxPooling2	(None, 3, 3, 256)	0
conv2d_10 (Conv2D)	(None, 3, 3, 384)	885120
conv2d_11 (Conv2D)	(None, 3, 3, 384)	1327488
conv2d_12 (Conv2D)	(None, 3, 3, 256)	884992
max_pooling2d_8 (MaxPooling2	(None, 1, 1, 256)	0
flatten_3 (Flatten)	(None, 256)	0
dense_6 (Dense)	(None, 4096)	1052672
dropout_3 (Dropout)	(None, 4096)	0
dense_7 (Dense)	(None, 4096)	16781312
dropout_4 (Dropout)	(None, 4096)	0
dense_8 (Dense)	(None, 2)	8194

Total params: 21,568,642
Trainable params: 21,568,642
Non-trainable params: 0

4. VGG13

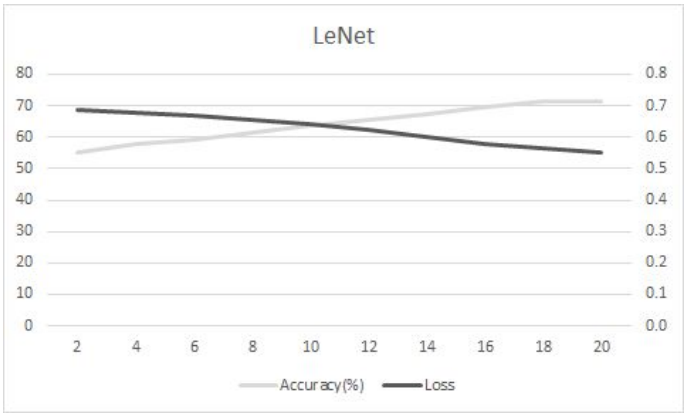
Layer (type)	Output Shape	Param #
conv2d_23 (Conv2D)	(None, 64, 64, 64)	1792
conv2d_24 (Conv2D)	(None, 64, 64, 64)	36928
max_pooling2d_14 (MaxPooling	(None, 32, 32, 64)	0
conv2d_25 (Conv2D)	(None, 32, 32, 128)	49280
conv2d_26 (Conv2D)	(None, 32, 32, 128)	147584
max_pooling2d_15 (MaxPooling	(None, 16, 16, 128)	0
conv2d_27 (Conv2D)	(None, 16, 16, 256)	295168
conv2d_28 (Conv2D)	(None, 16, 16, 256)	590080
max_pooling2d_16 (MaxPooling	(None, 8, 8, 256)	0
conv2d_29 (Conv2D)	(None, 8, 8, 512)	1180160
conv2d_30 (Conv2D)	(None, 8, 8, 512)	2359808
max_pooling2d_17 (MaxPooling	(None, 4, 4, 512)	0
conv2d_31 (Conv2D)	(None, 4, 4, 512)	2359808
conv2d_32 (Conv2D)	(None, 4, 4, 512)	2359808
max_pooling2d_18 (MaxPooling	(None, 2, 2, 512)	0
flatten_5 (Flatten)	(None, 2048)	0
dense_12 (Dense)	(None, 4096)	8392704
dropout_7 (Dropout)	(None, 4096)	0
dense_13 (Dense)	(None, 4096)	16781312
dropout_8 (Dropout)	(None, 4096)	0
dense_14 (Dense)	(None, 2)	8194

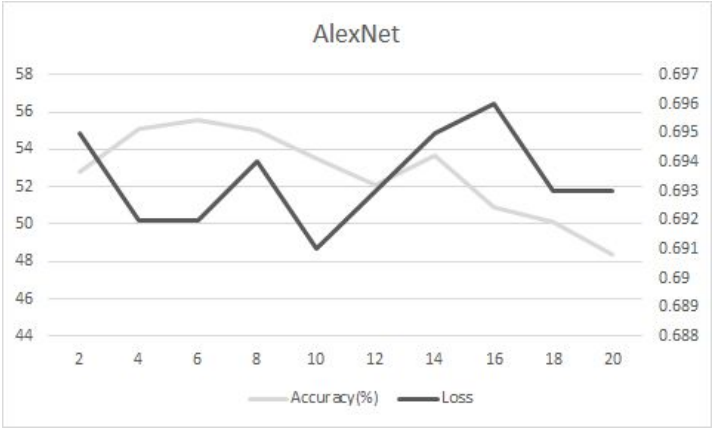
Total params: 34,562,626
Trainable params: 34,562,626
Non-trainable params: 0

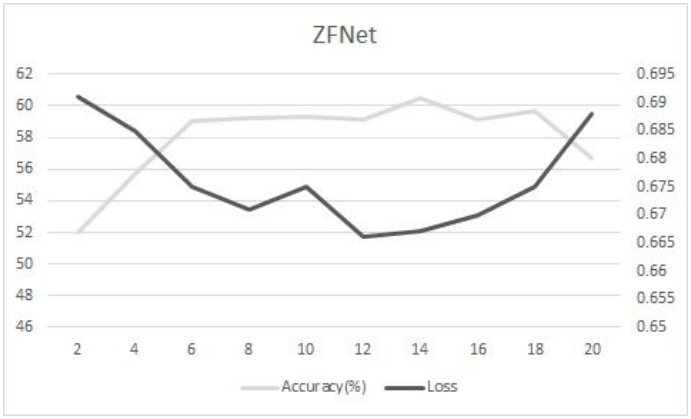
As shown above, our 4 models are very complicated. Besides LeNet with a total params of 1,135,598, the other three models are all have total params more than 10,000,000. And all of them has something new at their time.

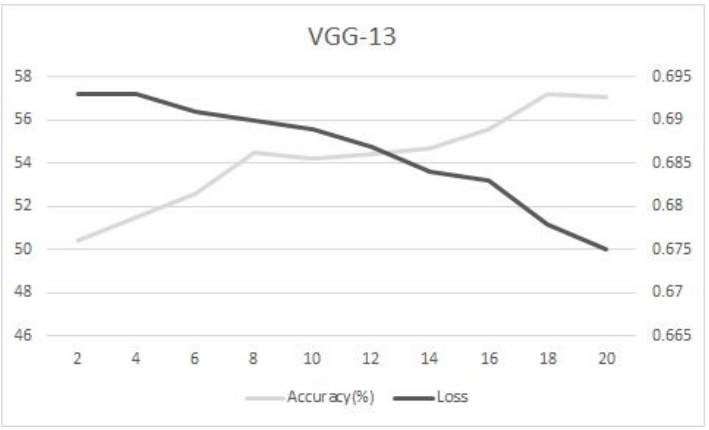
We fit out models on the same train and test data, all with an epoch of 20 and a batch size of 50. We recorded 1) train loss, 2) train accuracy, 3) test loss 4) test accuracy
after every epoch training.

- Training Accuracy and Loss:

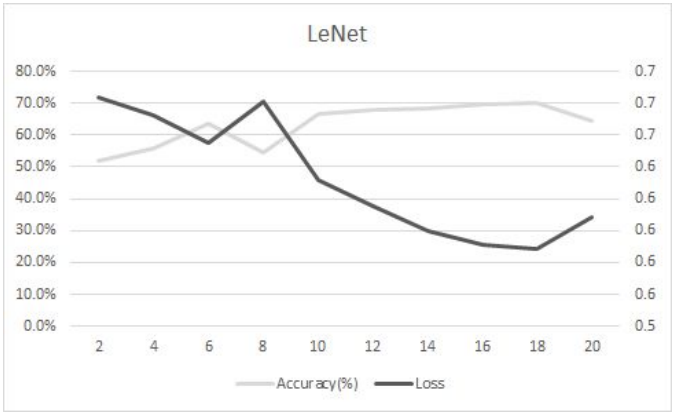


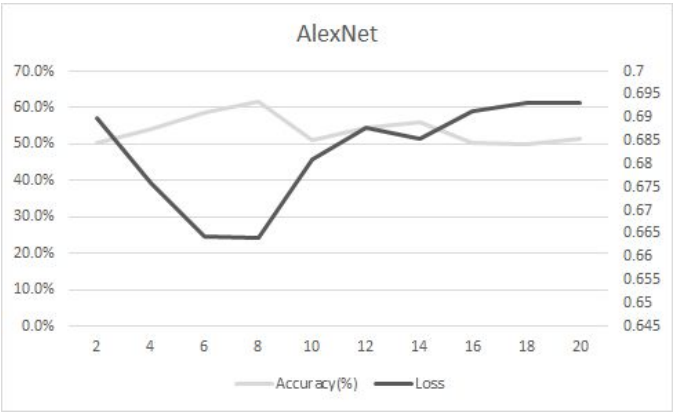


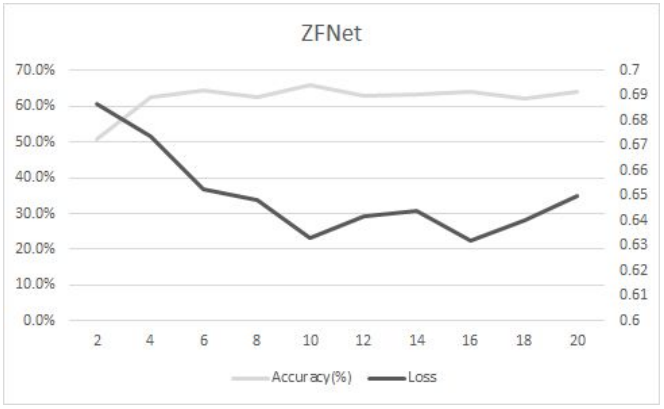


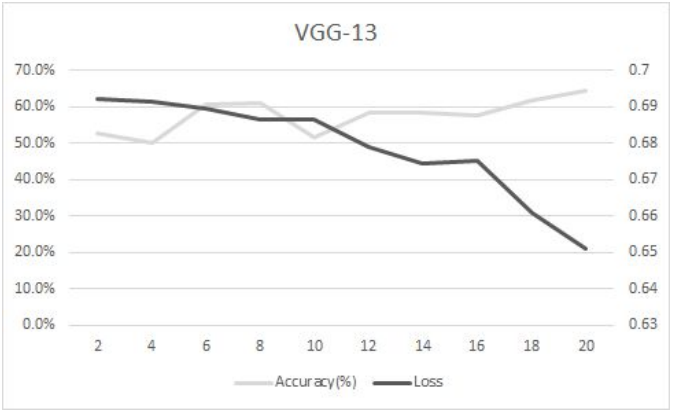


- Testing Accuracy and Loss:









As shown above, the results are kind of wired. LeNet and VGG-13 give good results in the first 20 epoches. They keep achieving better results and seem to have a potential to get better results if train more epoches. But for AlexNet and ZFNet, their performances are very frustrating, they seem to lose their way on the training, even for training data, there is not a decreasing trend in training loss and an increasing trend in training accuracy.
Why this situation happens? Is it because the structure is too complicated and is not applicable for this binomial classification question? But why VGG-13 works?

And by comparing VGG-13 and LeNet, though they both have a clear trend in loss and accuracy, the speed of improving of VGG-13 is much slower than LeNet. LeNet got to an accuracy rate of around 70% after 20 epoch training, but VGG-13 only got to around 57%. We think this is partly caused by the different of the amount of params. For a model with more then 10 million params like VGG-13, it's very slow to fit better to the dataset so it will takes more epoches to train comparing with a simple model. But is there any other reasons. This may be our next study direction.

- Model Ensemble Conclusion

Then we tried to ensemble the five models using a voting approch. We try to improve our classification accuracy into a new level but failed.

Because of some reasons, we don't test the effect of model ensemble on the original training and testing data which we use to train the 4 models, but on the whole original data, which is consist of 12500 cats and 12500 dogs.

The original accuracy and loss we calculated for the 4 models are:

```
The results for model1 is:
     loss = 0.6027353811836242, acc = 0.66272

The results for model2 is:
     loss = 0.6930758922576904, acc = 0.51116

The results for model3 is:
     loss = 0.6582265794754029, acc = 0.62364

The results for model4 is:
     loss = 0.6563407592010498, acc = 0.63092
```

Then we use a voting strategy for model ensemble. Because model1 has the best performance, we give model1 a weight more than others. The weights we have tried are [2,1,1,1], [3,1,1,1],[4,1,1,1] and get following result:

```
when weights =  [2, 1, 1, 1]
ensemble accuracy =  0.66196

when weights =  [3, 1, 1, 1]
ensemble accuracy =  0.66292

when weights =  [4, 1, 1, 1]
ensemble accuracy =  0.66272
```

As shown above, for different weights, we will get different ensemble accuracy. And ensemble approach do help us to get a higher accuracy of 0.66292.

However, there is still a weight setting problem for us. As shown above, if we give model LeNet a smaller weight like 2, other models will pull down the total accuracy. If we give it a higher weight of 4, the ensemble accuracy will be the same as Model LeNex still decides all the situation by itself. Only if we give a proper weight like 3 can the ensemble accuracy be improved.

So how to decide a proper weights? Is there a normal way to do that? We will study it deeper in the future.

- Data Augmentation Conclusion

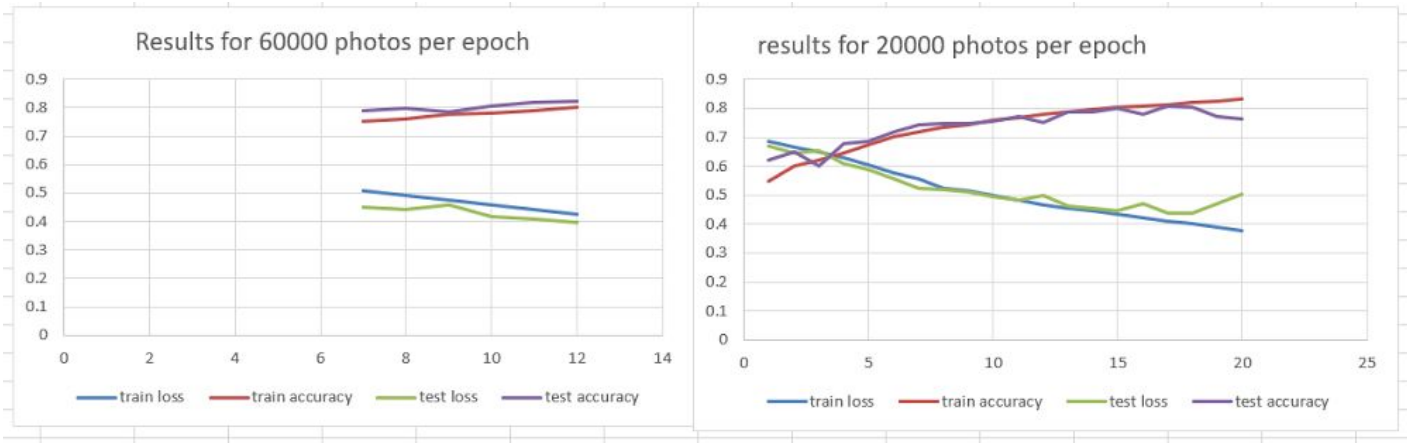
After trying model ensemble, we want to see what will the effect of data augmentation be. We trained LeNex on two kinds of dataset :

(1) train----10000 dogs and 10000 cats (Total 20000)
test----2500 dogs and 2500 cats (Total 5000)

(2)
train----10000 dogs and 10000 cats and 2 more photos got from each of the dogs and cats, so total 80000 (Total 60000, the same 20000+new 40000)
test---- 2500 dogs and 2500 cats (Total 5000, the same)

We trained LeNex on the two kinds of dataset separately. Former with a 20 epoch and a 50 batch size, Later with a 6 epoch and a 50 batch size. Both start from a random weight condition without training before.

Finally we got the results below:



As shown above, after training for 18 epochs, the result of the model training on 20000 photos gives an obvious trend of overfitting. The test loss goes up and test accuracy began to go down, the difference between test and train loss or between train and test accuracy becomes bigger.

But for the models trained on our dataset after using data augmentation is not severe. In fact, after the training of most epoches, the test loss is lower than train loss and testing accuracy is higher than train accuracy. Though the epoch number seems not fit each other, but the test accuracy and test loss has achieved the same level. And if considered into the training picture number perspective. 12 epoches of a 60000-photo dataset means 720000 photos, and 20 epoches of a 20000-photo dataset is 400000 photos. So the model has been trained on nearly on more time of photos but still not have a trend of overfitting.

So, because of the results from the following pictures, we get to the conclusion that these two results can show the power of data augmentation to avoid overfitting and improve test accuracy of the model.

5.References：

[1] http://image-net.org/challenges/LSVRC/2012/results.html
[2] http://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
[3] http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf
[4] http://blog.csdn.net/wang1127248268/article/details/77258055
[5] http://blog.csdn.net/cyh_24/article/details/51440344
[6] https://www.kaggle.com/jeffd23/catdognet-keras-convnet-starter