Machine Learning Engineer Nanodegree

Capstone Project

Xu Lin
February 8th, 2018

I. Definition

Project Overview

There are lots of things that is easy for humans, dogs, cats, but your computer will find it a bit more difficult. Web services are often protected with a challenge that's supposed to be easy for people to solve, but difficult for computers. Such a challenge is often called a CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) or HIP (Human Interactive Proof). HIPs are used for many purposes, such as to reduce email and blog spam and prevent brute-force attacks on web site passwords.

As for identifying photographs of cats and dogs, this task is difficult for computers, but studies have shown that people can accomplish it quickly and accurately. Kaggle has redux version for this. 1314 teams have been participated in this project and the leaderboard can be viewed here. Also there are some academic research about this project, like here

My personal motivation for this is improving my web crawler performance. Lots of time my web crawler will stop when it comes with CAPTCHA. I believe this project will help me have betting understanding how to deal with this situation.

Problem Statement

Our basic task is a supervised learning problem, we need to create an algorithm to classify whether an image contains a dog or a cat. The input for this task are images of dogs or cats from training dataset, while the output is the classification accuracy on test dataset. Keras or TFLearn might be applied as one of the solution method.

We will have a data set (images) with label indicated whether it is a dog or cat, we expect we can come up with an algorithm to calculate the probability that the image is a dog (1 = dog, 0 = cat).

We will use Log Loss to value the result.

Metrics

[The project is evaluated by log loss]

Submissions are scored on the log loss:

$$\text{LogLoss} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right],$$

where

- n is the number of images in the test set
- $\hat{y}_i$ is the predicted probability of the image being a dog
- y_i is 1 if the image is a dog, 0 if cat
- $log()$ is the natural (base e) logarithm

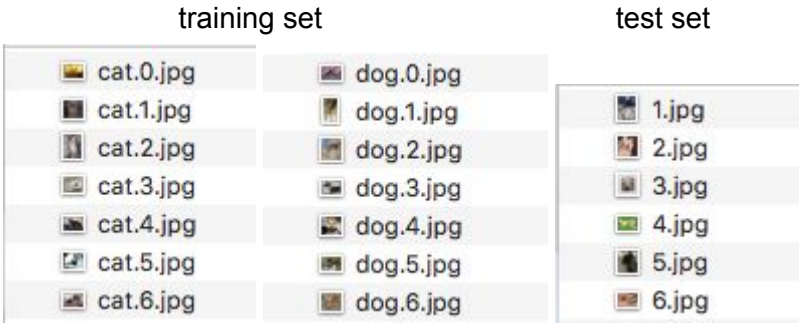
A smaller log loss is better.

II. Analysis

Data Exploration and Exploratory Visualization

[The data is coming from here] The train folder contains 25,000 images of dogs and cats. Each image in this folder has the label as part of the filename. The test folder contains 12,500 images, named according to a numeric id. For each image in the test set, you should predict a probability that the image is a dog (1 = dog, 0 = cat).

Files screenshots like this:



Regular dog and cat image will be like this:

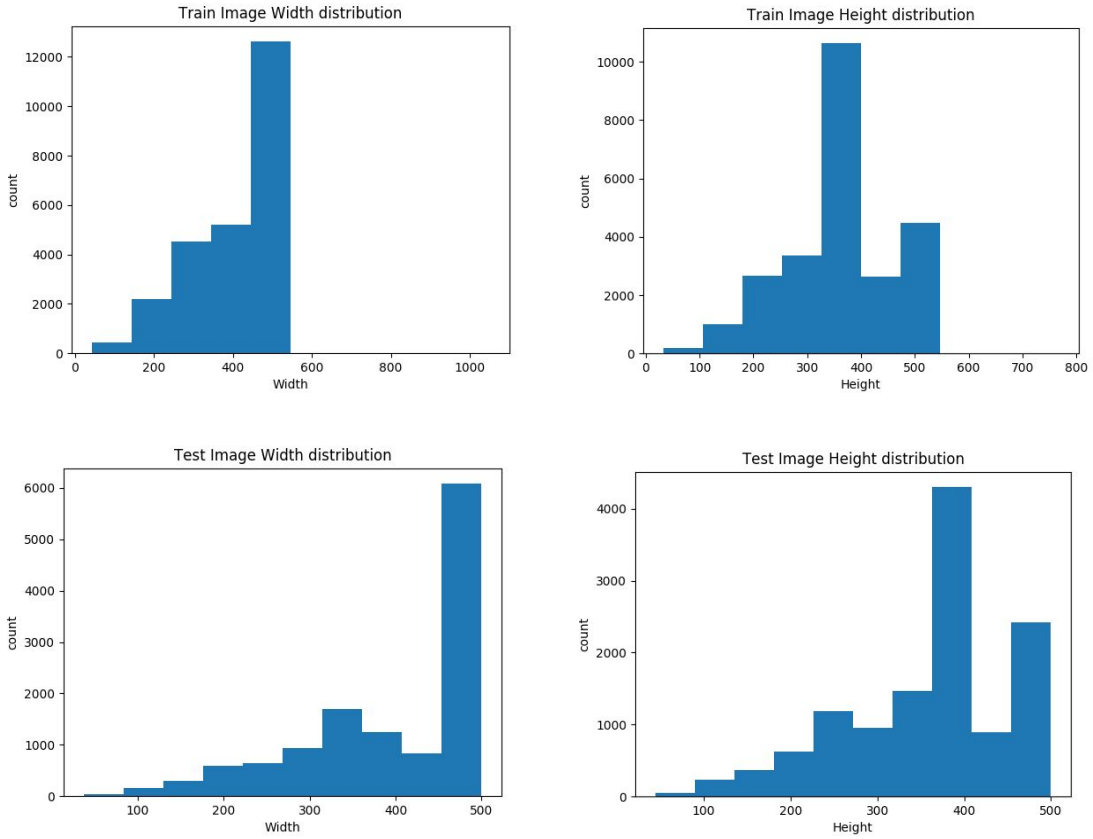
But you will find some special images, which make this classification exercise challenging:

1. Image has people or other objects
2. Some images have more than 1 dog/cat
3. Dog/cat is not always centered

4. Weight height ratio is different
5. Illumination amount is different

Input image data set width and height distribution

code can be found here



Algorithms and Techniques

Convolutional neural networks (CNNs) are widely used in pattern- and image-recognition problems as they have a number of advantages compared to other techniques. As for this project, I am trying to go with CNN as follow steps:

1. Train a small network from scratch, and see the result
2. Trying to improve it with add more layers
3. Trying to improve it with modifying input, like change input size or ImageDataGenerator parameter.
4. Trying to use some famous model like vgg16, vgg 19, etc

Benchmark

According to Kaggle Leaderboard, the first position of 1314 teams was taken by team "Cocostarcu", their Log Loss score is 0.03302. I hope I can be the top 20% of this project, which mean the Log Loss result should be lower than 0.08167. After the final result is submitted to Kaggle, it will show the final Log Loss.

III. Methodology

Data Preprocessing

Even though we know there are a few challenging “outliner”s, but since it is a Kaggle competition, we are not allowed to just remove them.
I compared the ImageAugmentation from TFlearn with ImageDataGenerator from Keras, it turns out ImageDataGenerator has more powerful features than ImageAugmentation. This class allows you to:

- Mapulate around 20 parameters to generate different images
- Generate batches of tensor image data with real-time data augmentation. The data will be looped over (in batches) indefinitely.
- Feed data into generator using multiple ways: like from a directory directly with method **flow_from_directory(directory)**

So I picked Keras as out project main training tool.

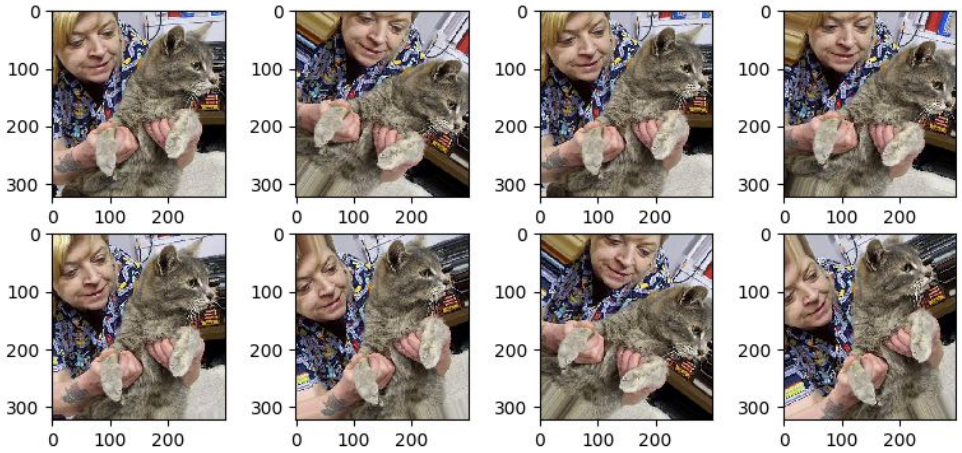
Let’s go through some parameters of ImageDataGenerator and see if they are useful for this project.

- rotation_range: Int. Degree range for random rotations.
- width_shift_range: Float (fraction of total width). Range for random horizontal shifts.
- height_shift_range: Float (fraction of total height). Range for random vertical shifts.
- shear_range: Float. Shear Intensity (Shear angle in counter-clockwise direction as radians)
- rescale: rescaling factor. Defaults to None. If None or 0, no rescaling is applied, otherwise we multiply the data by the value provided (before applying any other transformation).
- zoom_range: Float or [lower, upper]. Range for random zoom. If a float, [lower, upper] = [1-zoom_range, 1+zoom_range].
- horizontal_flip: Boolean. Randomly flip inputs horizontally.

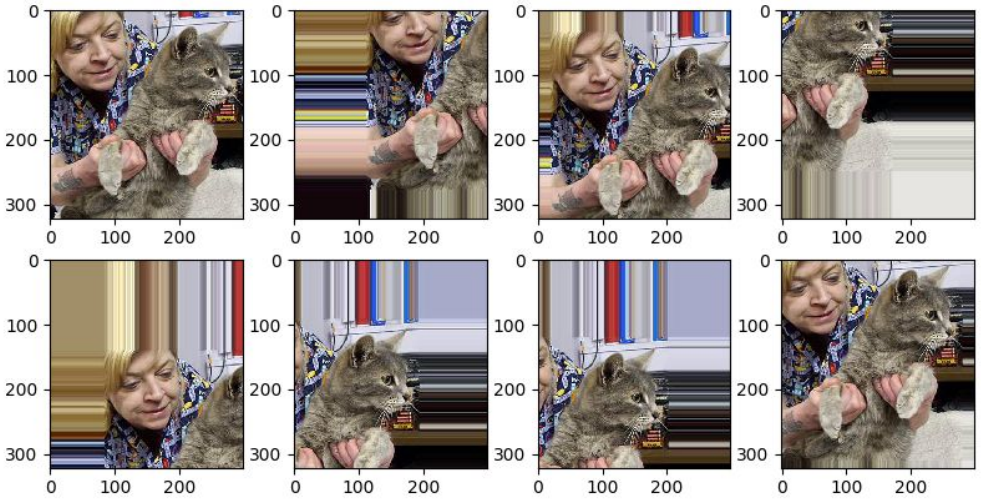
Real image after process example

Left top corner image is original one, code can be found here

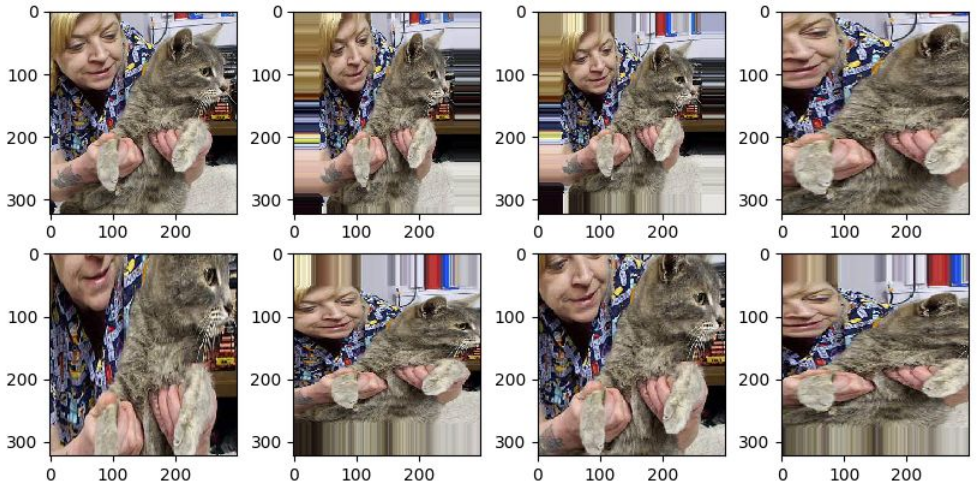
rotation_range = 40



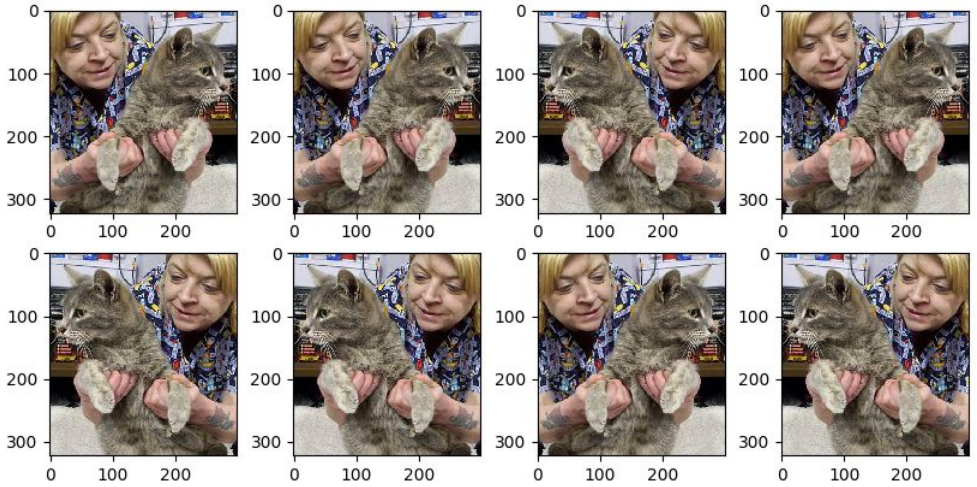
width_shift_range = 0.5, height_shift_range = 0.5 (0.5 seems too big, it shift half picture, 0.2 should be good option)



zoom_range = 0.5 (0.5 seems too big, 0.2 should be good option)



horizontal_flip = True



I would like to talk about **rescale** more, it works like this: it will made all the pixels in the image times this rescale number, this action should be done before other transformation. As for some models, directly input pixels might fall into “dead zone” of activate method, so if the rescale number is 1/255, the pixels gonna fall into [0, 1], which will be good to model convergence.

Since we have a big training set (25000 images), we don’t need to apply other parameters to augment the data. So In the project, I gonna just start with rescale = 1/225.

In order to use flow_from_directory function of ImageDataGenerator, I need to reorganise the folder structure. it will be like this:

```
.
├── test
│   └── test1 [12500 entries exceeds filelimit, not opening dir]
├── train
│   ├── cat [11500 entries exceeds filelimit, not opening dir]
│   └── dog [11500 entries exceeds filelimit, not opening dir]
└── valid
    ├── cat [1000 entries exceeds filelimit, not opening dir]
    └── dog [1000 entries exceeds filelimit, not opening dir]
```

12500 images of test1 folder are coming from original test folder. 12500 dog/cat training image are divided in to 11500 and 1000 and put into train and valid folder.

Implementation

GPU Prepare

I tried run training code on my Mac Mini, but it was very slow. So I have to go for cloud GPU instance. Compared with Google Cloud, I feel AWS is better, it has “ready to use” image, which contains Tensorflow GPU version, python 3, keras, etc installed, you can just launch it and start to work. I picked p2-xLarge E2 instance which contains a GPU K80, $0.9/hour, sometimes I use spot-instance, it gonna cost me like $0.3/hour.

Build From Scratch

Code can be found here.
Prepare data set:

```
img_width, img_height = 150, 150

train_data_dir = "input1/train"
validation_data_dir = "input1/valid"
nb_train_samples = 2000
nb_validation_samples = 800
epochs = 50
batch_size = 16

if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)
```

The screenshot below is our first model, 2 convolution layers with max-pooling layers, end with ReLu and Sigmoid activation. Dense(1) and Sigmoid Activation is perfect for a binary classification. After that we will try the binary_crossentropy loss to train the model.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 148, 148, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 72, 72, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 41472)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                2654272
_________________________________________________________________
activation_3 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_4 (Activation)    (None, 1)                 0
=================================================================
Total params: 2,664,481
Trainable params: 2,664,481
Non-trainable params: 0
_________________________________________________________________
```

Use .flow_from_directory() of ImageDataGenerator with rescale 1/255, it will generate image data directly from image folders.

```
datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary')

validation_generator = datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary')

model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size)
```

```
Epoch 50/50
125/125 [==============================] - 11s 89ms/step - loss: 0.4326 - acc: 0.8170 - val_loss: 0.4881 - val_acc:
0.7937
```

Note that the validation accuracy is 79.37%, because the model is too simple, I guess it loss some useful feature data. As you can see, it took 11s per epoch, so this run took around 9 minutes.

Refinement

In this section, I'm maining trying to change training model to achieve better result first, then I gonna try change the parameter in ImageDataGenerator.

Approach 1:

Code can be found here:
Add 1 more layer with relu activaition and followed by max-pooling layers.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 148, 148, 32)	896
activation_1 (Activation)	(None, 148, 148, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 74, 74, 32)	0
conv2d_2 (Conv2D)	(None, 72, 72, 32)	9248
activation_2 (Activation)	(None, 72, 72, 32)	0
max_pooling2d_2 (MaxPooling2	(None, 36, 36, 32)	0
conv2d_3 (Conv2D)	(None, 34, 34, 64)	18496
activation_3 (Activation)	(None, 34, 34, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 17, 17, 64)	0
flatten_1 (Flatten)	(None, 18496)	0
dense_1 (Dense)	(None, 64)	1183808
activation_4 (Activation)	(None, 64)	0
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 1)	65
activation_5 (Activation)	(None, 1)	0

Total params: 1,212,513
Trainable params: 1,212,513
Non-trainable params: 0

```
Epoch 50/50
125/125 [==============================] - 11s 90ms/step - loss: 0.4170 - acc: 0.8250 - val_loss: 0.3832 - val_acc:
0.8200
```

Note that the validation accuracy is 82.00%, it has around 3% improvement.

Approach 2:

Code can be found here:
Add 3 more layer with relu activaition and followed by max-pooling layers.

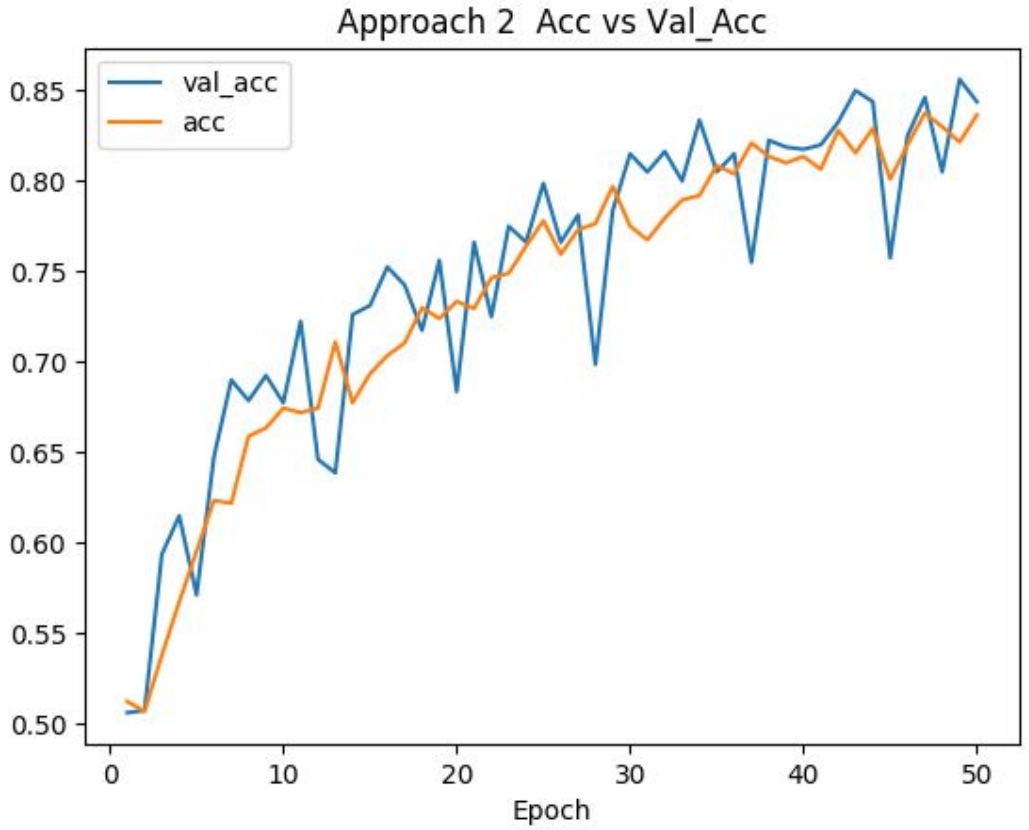
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 148, 148, 32)	896
activation_1 (Activation)	(None, 148, 148, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 74, 74, 32)	0
conv2d_2 (Conv2D)	(None, 72, 72, 32)	9248
activation_2 (Activation)	(None, 72, 72, 32)	0
max_pooling2d_2 (MaxPooling2	(None, 36, 36, 32)	0
conv2d_3 (Conv2D)	(None, 34, 34, 64)	18496
activation_3 (Activation)	(None, 34, 34, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 17, 17, 64)	0
conv2d_4 (Conv2D)	(None, 15, 15, 32)	18464
activation_4 (Activation)	(None, 15, 15, 32)	0
max_pooling2d_4 (MaxPooling2	(None, 7, 7, 32)	0
conv2d_5 (Conv2D)	(None, 5, 5, 64)	18496
activation_5 (Activation)	(None, 5, 5, 64)	0
max_pooling2d_5 (MaxPooling2	(None, 2, 2, 64)	0
flatten_1 (Flatten)	(None, 256)	0
dense_1 (Dense)	(None, 64)	16448
activation_6 (Activation)	(None, 64)	0
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 1)	65
activation_7 (Activation)	(None, 1)	0

Total params: 82,113
Trainable params: 82,113
Non-trainable params: 0

```
Epoch 50/50
125/125 [==============================] - 11s 91ms/step - loss: 0.3809 - acc: 0.8365 - val_loss: 0.3548 - val_acc:
0.8438
```

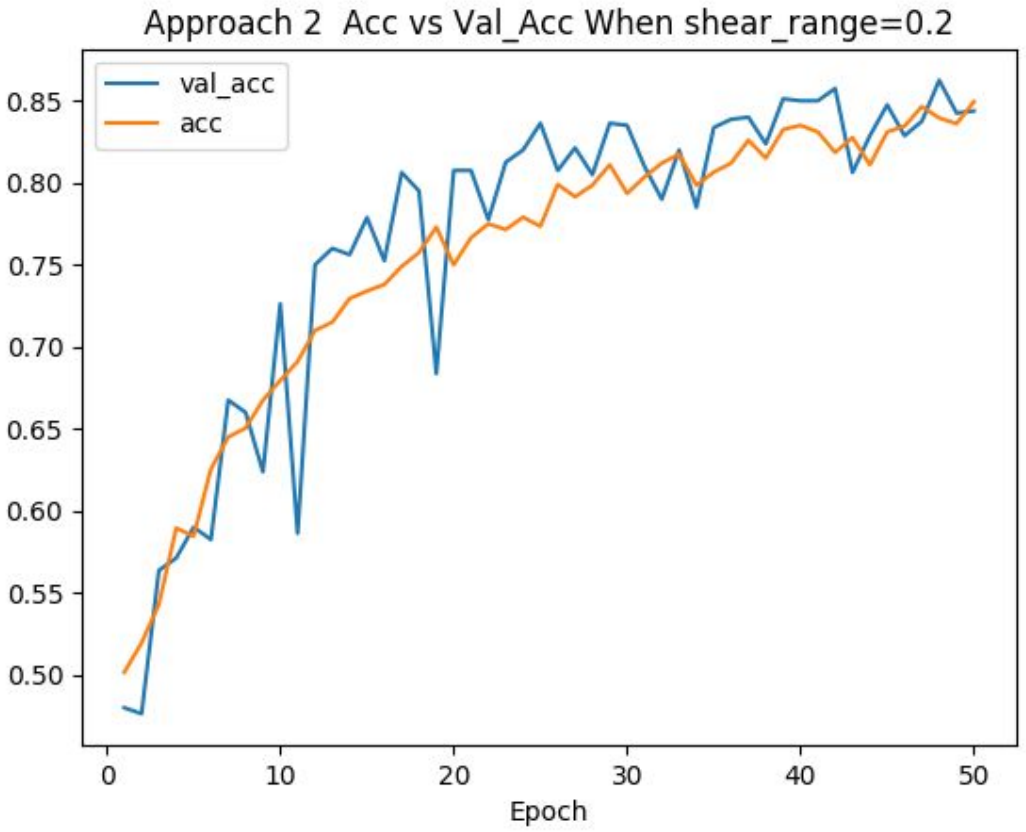
Note that the validation accuracy is 84.38%, it also has around 5% improvement.

But when I keep adding layers, the result is not improved a lot, which means adding layers blindly might not be a good solution.

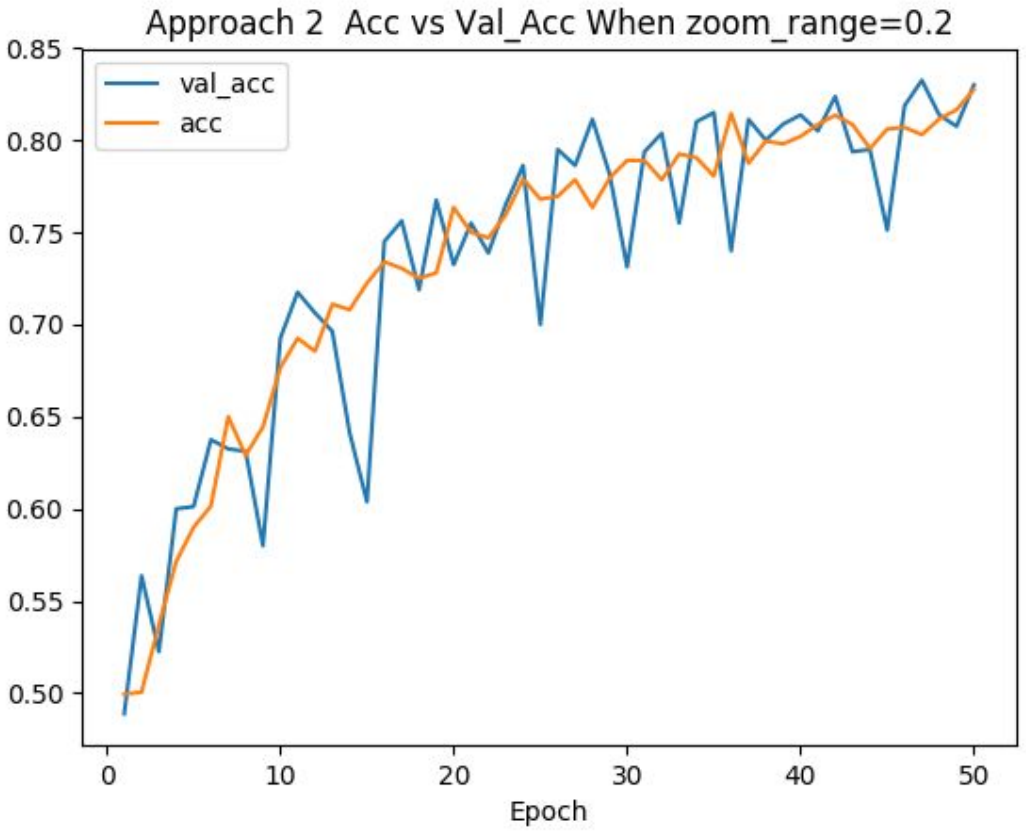


As you can see, training set accuracy and validation set accuracy both have some little up and down, but they are increasing overall. I guess it might have a little overfit on some places.

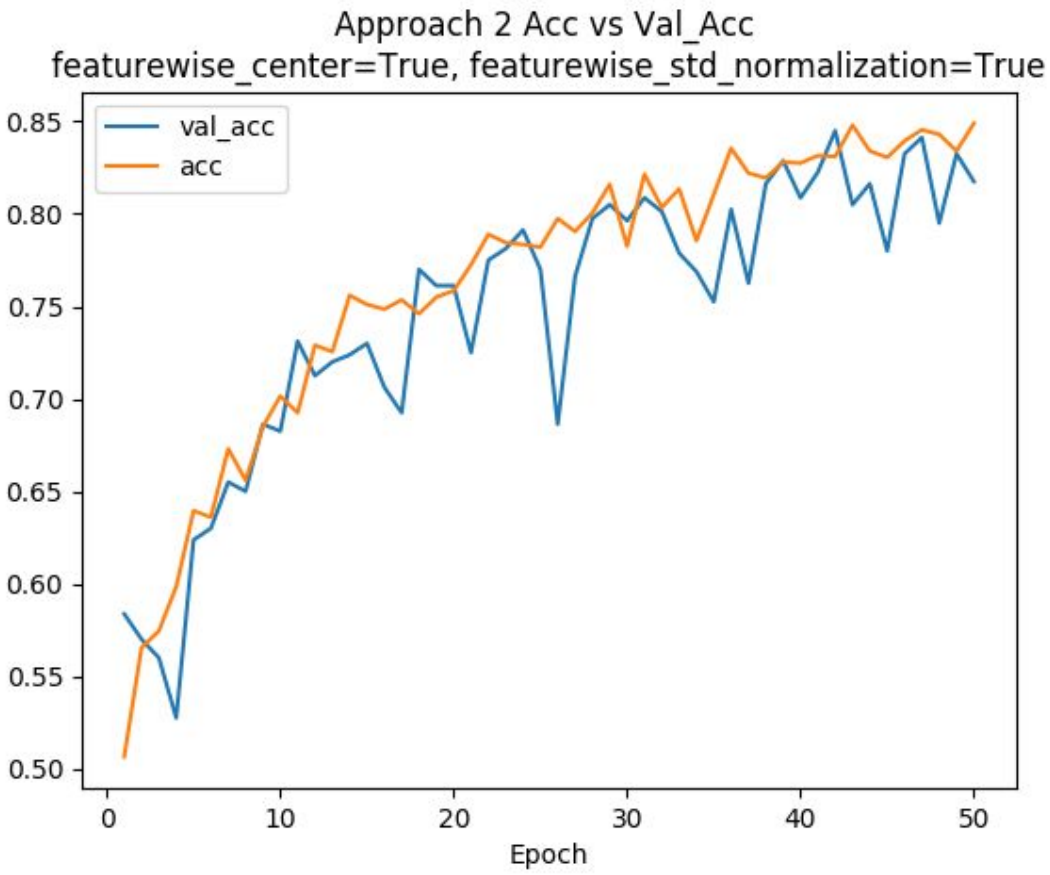
I would like to change the parameter of the ImageDataGenerator, and see if it could help to improve the performance.



As you can see, shear_range=0.2 does improve validation accuracy a little bit, but make it surged more, also double my training time, each epoch took around 22 second.



also zoom_range=0.2, performance doesn’t got improved, also double my training time, each epoch took around 22 second.



featurewise_center=True, featurewise_std_normalization=True, make validation accuracy even worse.

Approach 3:

It is hard to get improvement when I trying to change the parameter in ImageDataGenerator. So I do feel like the model is the bottleneck.
I decided to stand on shoulders of giants, I would like to try Transfer Learning using some pre-trained networks, like

- InceptionV3
- ResNet50
- VGG16
- VGG19
- Xception

Those models has been approved that they have great performance on ImageNet dataset. Since ImageNet dataset has many “dog” and “cat” class, so the models using ImageNet weight have learned their features, which will benefit this classification problem. That is the reason why we can use transfer learning here.

Thanks to PeiWen Yang share the idea.

Basic idea for this project is using those pre-trained network to predict our test/train images, get feature vectors and export to .h5 file. In this way, we can reuse those feature vectors to get better weight.

First, let's have a method to generate those feature vectors. Parameters will be pre-trained network name and their default input image size, corresponding image input tensor. All the weight for those pre-trained network gonna be “imagenet”.

```
def create_h5(MODEL, image_size, lambda_func=None):
    print(MODEL, 'start')
    width = image_size[0]
    height = image_size[1]
    input_tensor = Input((height, width, 3))
    x = input_tensor
    if lambda_func:
        x = Lambda(lambda_func)(x)
    base_model = MODEL(input_tensor=x, weights='imagenet', include_top=False)
    model = Model(base_model.input, GlobalAveragePooling2D()(base_model.output))

    gen = ImageDataGenerator()
    train_generator = gen.flow_from_directory("input/train", image_size, shuffle=False,
                                              batch_size=16)
    test_generator = gen.flow_from_directory("input/test1", image_size, shuffle=False,
                                             batch_size=16, class_mode=None)

    train = model.predict_generator(train_generator, train_generator.samples/16)
    test = model.predict_generator(test_generator, test_generator.samples/16)
    with h5py.File("feature_%s.h5"%MODEL.__name__) as h:
        h.create_dataset("train", data=train)
        h.create_dataset("test", data=test)
        h.create_dataset("label", data=train_generator.classes)


create_h5(ResNet50, (224, 224))
create_h5(Xception, (299, 299), xception.preprocess_input)
create_h5(InceptionV3, (299, 299), inception_v3.preprocess_input)
create_h5(VGG16, (224, 224))
create_h5(VGG19, (224, 224))
```

The features vector will contrain 3 numpy vector:

- train(25000, 2048)
- test(12500, 2048)
- label(25000,)

All .h5 files can be found here. Each one took me around 10-20 minutes on p2_xLarge.

We can get X_train from “train” vector, get Y_train from “label” vector, X_test from “test” vector.

```
X_train = []
X_test = []
filename = 'gap_ResNet50.h5'
with h5py.File(filename, 'r') as h:
    X_train.append(np.array(h['train']))
    X_test.append(np.array(h['test']))
    y_train = np.array(h['label'])

X_train = np.concatenate(X_train, axis=1)
X_test = np.concatenate(X_test, axis=1)

X_train, y_train = shuffle(X_train, y_train)
```

After get this X_train, we can just dropout and create this new model.

```
from keras.models import *
from keras.layers import *

input_tensor = Input(X_train.shape[1:])
x = input_tensor
x = Dropout(0.5)(x)
x = Dense(1, activation='sigmoid')(x)
model = Model(input_tensor, x)

model.compile(optimizer='adadelta',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Then use this model to fit X_train, Y_trian, we can set validation_split = 0.2, which means training set is 20000 image, validation is 5000 images.

```
model.fit(X_train, y_train, batch_size=128, nb_epoch=8, validation_split=0.2)
y_pred = model.predict(X_test, verbose=1)
```

InceptionV3

```
Epoch 8/8
20000/20000 [==============================] - 2s 111us/step - loss: 0.0261 - ac
c: 0.9925 - val_loss: 0.0226 - val_acc: 0.9934
```

ResNet50

```
Epoch 8/8
20000/20000 [==============================] - 2s 98us/step - loss: 0.0756 - acc
: 0.9710 - val_loss: 0.0621 - val_acc: 0.9762
```

VGG16

```
Epoch 8/8
20000/20000 [==============================] - 2s 98us/step - loss: 0.2551 - acc
: 0.9369 - val_loss: 0.0944 - val_acc: 0.9712
```

VGG19

```
Epoch 8/8
20000/20000 [==============================] - 1s 26us/step - loss: 0.2006 - acc
: 0.9433 - val_loss: 0.0912 - val_acc: 0.9708
```

Xception

```
Epoch 8/8
20000/20000 [==============================] - 2s 96us/step - loss: 0.0243 - acc
: 0.9930 - val_loss: 0.0268 - val_acc: 0.9928
```

We will see those pre-trained networks perform pretty good. Top 2 is Xception and InceptionV3. Use the model to predict test folder, generate csv file and upload to Kaggle, we got

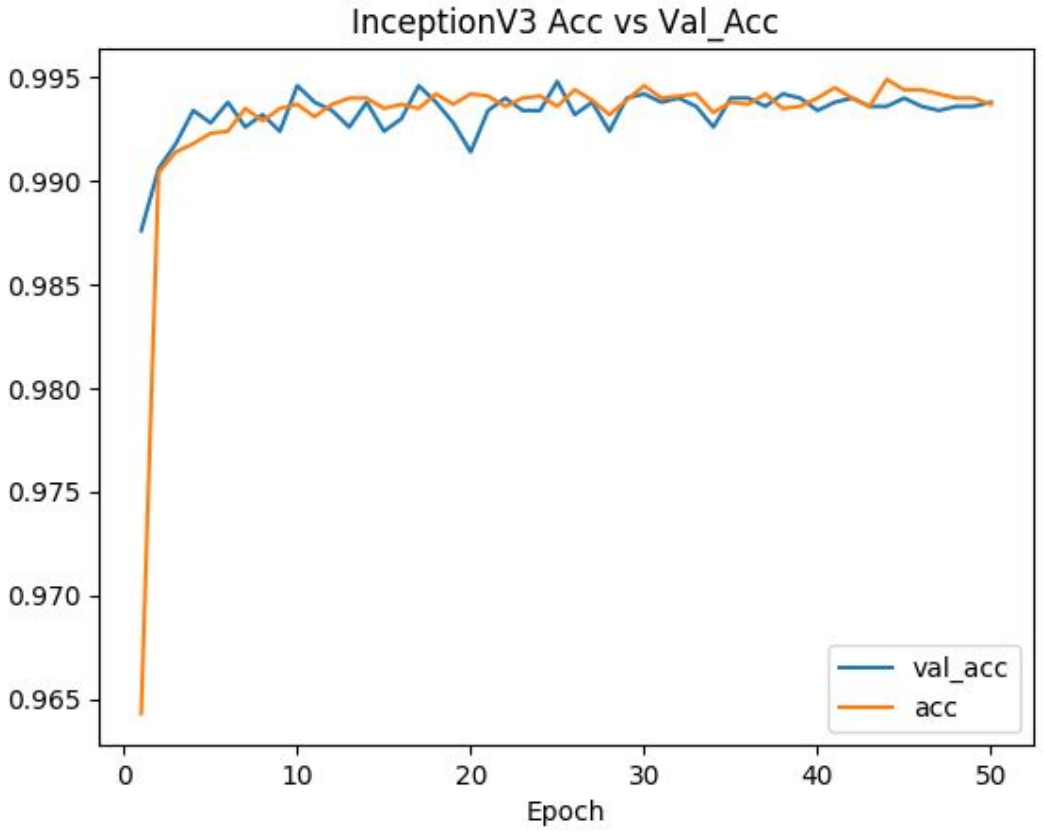
Submission and Description	Public Score	Use for Final Score
pred.csv a few seconds ago by Luke Lin feature_InceptionV3	0.04088	☐
pred.csv a minute ago by Luke Lin feature_Xception	0.04119	☐

You will see Xception got 0.04119, InceptionV3 got 0.04088

IV. Results

Model Evaluation and Validation

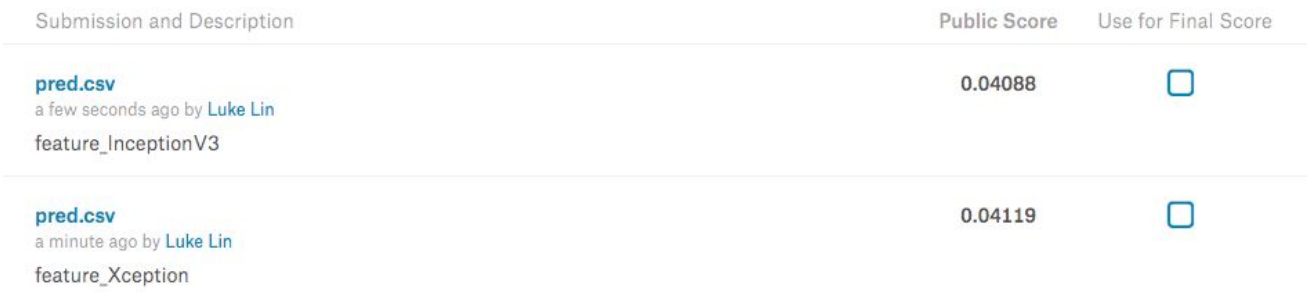
According to final result, the best model we got is InceptionV3. Weights are saved in “model.h5”. As you can see accuracy and validation accuracy are pretty stable, even epoch to 50, which can approve its robustness



Justification

```
y_pred = y_pred.clip(min=0.005, max=0.995)
```

There is a small trick here, we clip the prediction value to [0.005, 0.995]. The reason for that is Metrics for this project is LogLoss, so as for correct prediction, 0.995 is almost same with 1, but as for incorrect prediction, 0 is huge difference with 0.005, which is the difference between 15 and 2.



The final result 0.04088 is small the benchmark 0.08167 mentioned above. This solution is good enough to solve this problem. It might better than me.

V. Conclusion

Reflection

I learnt a lot as I worked through this project

- Understanding of deep learning, convolution neural networks and Tensorflow. Working on a real project will help understand those concept in higher level
- ImageDataGenerator is pretty powerful tool, especially you don't have enough data, it will help to enlarge your input data set
- Getting feature vectors will take unknown time on CPU. I compared AWS with Google Cloud to use GPU, I realized AWS is much better than Google Cloud in terms of transparency, documentation, support, resource online.
- Remember to save your work all the time, like saving data/model in to .h5 file. Since training is really time-consuming task, so save your work equals save time.
- Increase layer doesn't mean higher accuracy. It's good to set a stop function when you start to train.
- Don't hesitate to use well-known networks, they all well-trained, got a lots of prizes from different competitions. Since I'm not a researcher from famous lab or a phd who concentrate on building a model. I think the best way of machine learning for me is knowing how to use those existing models, not build on from scratch.

Improvement

- More aggresive data augmentation

- More aggresive dropout
- Combine different model weight and retry.

VI. References

- Francois Chollet, Building powerful image classification models using very little data, https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
- Peiwen Yang, https://ypw.io/dogs-vs-cats-2/
- CS231 Convolutional Neural Networks for Visual Recognition, http://cs231n.github.io/transfer-learning/