Cats and Dogs Image Recognition Using Convolutional Neural Network

Ming Jin, Xin Luo

Abstract

In this project, our task is to develop an algorithm to classify images of dogs and cats. It is a common topic for doing image recognition. There are some methods to solve this problem. But we want to compare some typical model according to the training accuracy and test accuracy and find the best fit. We used Deep Convolutional Neural Networks (CNN) to learn features of images. We tried various experiments to improve our performance on the test dataset, and finally got the best accuracy of 98.47% by the second approach.

Cats vs. Dogs is a Kaggle big data competition in a certain year. It uses a given data set to realize the recognition of Cats and Dogs by algorithms. We saw all the kernels found there are lots of methods to solves it, such as Keras and Tensorflow. Also, we noticed that the accuracy of training data and test data in Tensorflow function is the highest one. But not too many group use Tensorflow. Our task is to figure out which methods is the most significant. Also, the which hyper parameter can fit well. One of the most popular deep learning techniques is a Convolutional Neural Network which is a class of deep, feed-forward artificial neural networks. CNNs are commonly used for solving problems related to Computer Vision.

Our basic task is to create an algorithm to classify whether an image contains a dog or a cat. The input for this task is images of dogs or cats from training dataset, while the output is the classification accuracy on test dataset. Our learning task is to learn a classification model to determine the decision boundary for the training dataset. The whole process is illustrated in Figure 1, from which we can see the input for the learning task is images from the training dataset, while the output is the learned classification model. Our performance task is to apply the learned classification model to classify images from the test dataset, and then evaluate the classification accuracy.

Introduction

A. Background

In this project, our task is to develop an algorithm to classify images of dogs and cats. It is a common topic for doing image recognition. There are some methods to solve this problem. But we want to compare some typical model according to the training accuracy and test accuracy and find the best fit. We used Deep Convolutional Neural Networks (CNN) to learn features of images. We tried various experiments to improve our performance on the test dataset, and finally got the best accuracy of 98.47% by the second approach.

Cats vs. Dogs is a Kaggle big data competition in a certain year. It uses a given data set to realize the recognition of Cats and Dogs by algorithms. We saw all the kernels found there are lots of methods to solves it, such as MLP, Keras and Tensorflow. Also, we noticed that the accuracy of training data and test data in Tensorflow function is the highest one. But not too many group use Tensorflow. Our task is to figure out which methods is the most significant. Also, the which hyper parameter can fit well.

B. Algorithms

We plan to use the algorithms below:

A convolutional neural network (CNN) is a class of deep neural networks, most commonly applied to analyzing visual imagery. Compared with other image classification algorithms, CNN USES relatively less preprocessing. This means that the network learns to manually design filters in traditional algorithms. This independence from prior knowledge and manpower in feature design is a major advantage.

Based on CNN, we use Tensorflow(TFLearn) and Keras to compare the accuracy. TensorFlow is flexible and difficult to learn. Keras is capable of running on top of TensorFlow. It is also flexible and easy to learn. Keras can be seen as an API encapsulated by Tensorflow.

In our solution, we mainly tried two different approaches. The first method is a traditional pattern recognition model, by which we learned the classification model from some human-crafted features, mainly including color feature, Dense-SIFT feature, and a combination of the two kinds of features. The second method is a trainable model, by which we applied a CNN to learn features. In terms of classifiers, we mainly chose SVMs and BP Neural Networks, considering the high dimensional feature space for images. We tried various experiments to achieve high accuracy on the test dataset, with different algorithms and parameter settings.

TensorFlow, Theano and Keras are deep learning frameworks. TensorFlow and Theano are flexible and difficult to learn. They are actually differentiators. Keras is actually the interface between TensorFlow and Keras. It is also flexible and easy to learn. Keras can be seen as an API encapsulated by Tensorflow.

METHODS

A. Dataset

We use 10 thousand cats and dogs images from Kaggle website, which has separated by two files. One as training set, the other as test set.

The link has attached. https://www.kaggle.com/tongpython/cat-and-dog

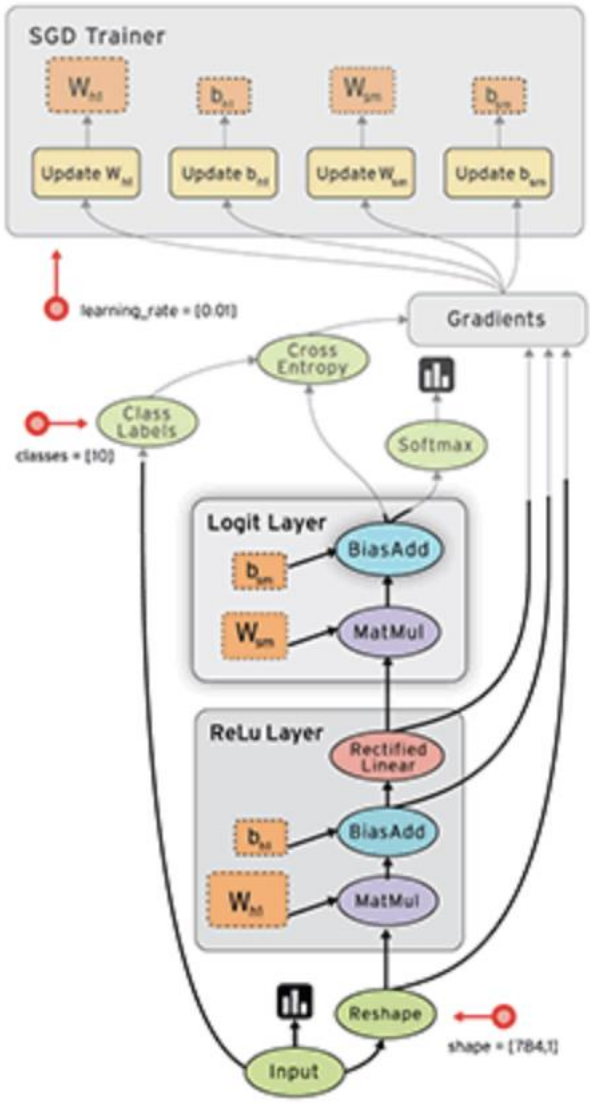
The dataset has the following distribution:

1. Train set contains roughly 80% of the total images
2. Test set contains roughly 20% of the total images

B. Model Selection

1) Tensorflow:

TensorFlow™ is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.



Figure— Graph's data stream

2) TFLearn:

TFLearn is a modular and transparent deep learning library built on top of Tensorflow. It was designed to provide a higher-level API to TensorFlow in order to facilitate and speed-up experimentations, while remaining fully transparent and compatible with it.

Those of you who have used TensorFlow should know that this deep learning library is a bit difficult to use and requires us to define all the compute nodes (op) ourselves. So, if we want to define a full connectivity layer, we need to first define the parameters weight(W) and bias (B) of the full connectivity layer, and then define the output Tensor equal to the input Tensor times W plus B and then take the activation function.

TFLearn is a modular and transparent deep learning library based on TensorFlow. It provides a higher level of API than TensorFlow, allowing us to experiment more quickly.

The TFLearn library has the following features

I. It is easy-to-use and understand high-level API for implementing deep neural networks, with tutorial and examples.
II. Fast prototyping through modular built-in neural network layers, regularizes, optimizers, metrics...
III. Full transparency over Tensorflow. All functions are built over tensors and can be used independently of TFLearn.
IV. Powerful helper functions to train any TensorFlow graph, with support of multiple inputs, outputs and optimizers.
V. Easy and beautiful graph visualization, with details about weights, gradients, activations and more...
VI. Effortless device placement for using multiple CPU/GPU

3) Keras:

Keras provides a simple and modular API to create and train Neural Networks, hiding most of the complicated details under the hood. This makes it easy to get you started on your Deep Learning journey.

4) Convolutional neural network (CNN)

A Convolutional Neural Network (CNN) is a multilayered neural network with a special architecture to detect complex features in data. CNNs have been used in image recognition, powering vision in robots, and for self-driving vehicles. In this article, we're going to build a CNN capable of classifying images. An image classifier CNN can be used in myriad ways, to classify cats and dogs.

Convolutional neural network is a feedforward neural network. Its artificial neurons can respond to some surrounding units within the coverage range, and it has excellent performance for large image processing. Convolutional neural network consists of one or more convolutional layers and the fully connected layer at the top, as well as the associated weight and pooling layer. This structure enables convolutional neural networks to take advantage of the two-dimensional structure of input data. Compared with other deep learning structures, convolutional neural networks can give better results in image and speech

recognition. Compared with other depth and feedforward neural networks, convolutional neural networks require fewer parameters to be estimated, making them an attractive structure for deep learning.

C. CNN Architecture

A convolution is a combined integration of two functions that shows you how one function modifies the other. There are three important items to mention in this process: the input image, the feature detector, and the feature map. The input image is the image being detected. The aim of this step is to reduce the size of the image and make processing faster and easier. Some of the features of the image are lost in this step. However, the main features of the image that are important in image detection are retained. These features are the ones that are unique to identifying that specific object.

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau$$
$$= \int_{-\infty}^{\infty} f(t - \tau) g(\tau)\, d\tau.$$

Figure— The convolution function

Then, in next step we apply the rectifier function to increase non-linearity in the CNN. Images are made of different objects that are not linear to each other. Without applying this function, the image classification will be treated as a linear problem while it is actually a non-linear one.

Pooling enables CNN to detect features in different images, regardless of the difference in illumination and the angle of the image in the image. There are different types of pools, such as the largest pool and the smallest pool. The Max pool works by placing a 2x2 matrix on the feature map and selecting the largest value in the box. Move the 2x2 matrix from left to right, traversing the entire feature map, and selecting the largest value in each traversal. These values then form a new matrix called the pool function map. The largest pool works to maintain the main function while also reducing the size of the image. This helps to reduce overfitting, and if the CNN gets too much information, especially when the information is not related to the image classification, overfitting will occur.

Once the aggregated feature map is obtained, the next step is to flatten it. Flattening involves converting the entire set of feature map matrices into a single column, which is then input into the neural network for processing.

After flattening, the flattened feature map is passed through the neural network. This step consists of an input layer, a fully connected layer, and an output layer. The fully connected layer is similar to the hidden layer in ANNs, but in this case it is fully connected. The output layer is where we get the prediction class. The information is passed through the network to calculate the prediction error. The error is then propagated back through the system to improve prediction accuracy.

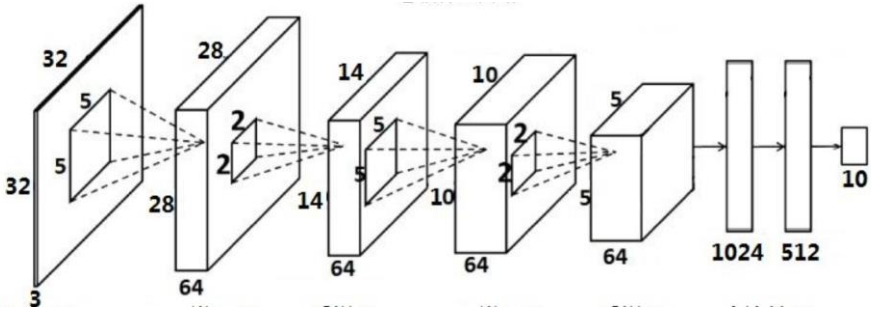
The final result produced by a neural network is usually not equal to one. However, it is important that these numbers fall between 0 and 1, which represents the probability of each class. This is what the Softmax function does.

$$\sigma : \mathbb{R}^K \to (0,1)^K$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Figure—The Softmax function

D. Step of CNN



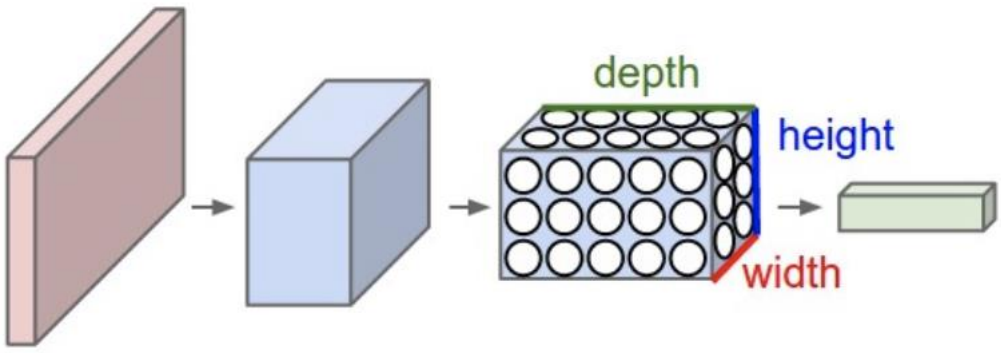
a. Input image
b. Convolutional layer
c. Pooling layer
d. Convolutional layer

e. Pooling layer
f. Fully connected layer

E. Convnet

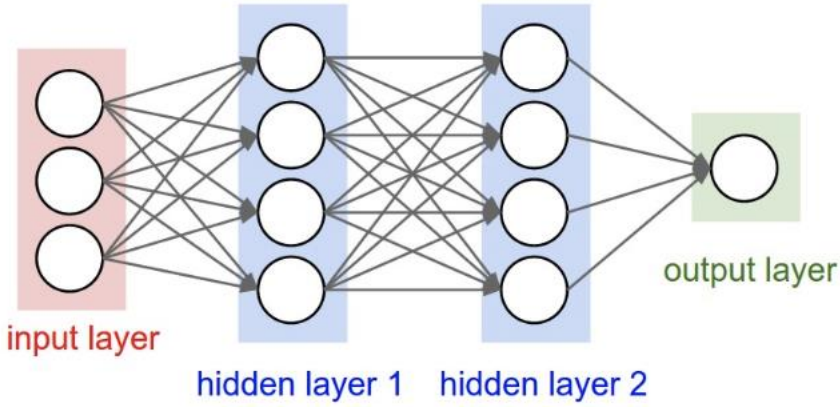
A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer (exactly as seen in regular Neural Networks). We will stack these layers to form a full ConvNet architecture. ConvNets transform the original image layer by layer from the original pixel values to the final class scores. Note that some layers contain parameters and other don't. In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image. A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume. There are a few distinct types of Layers

Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function. A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3.



Figure— A ConvNet arranges its neurons in three

The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting. The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height) but extends through the full depth of the input volume. The size of our convolutional layer filter is (5,5).

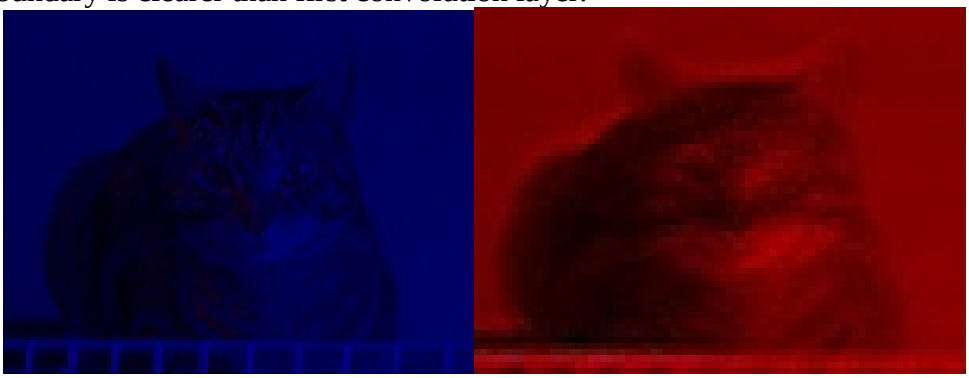


Figure— A regular 3-layer Neural Network

F. Our Step

a. Load data

b. Use one image to show how image change after image has been do the convolution and max pooling. First convolution layer and final convolution layer. Obviously, the boundary is clearer than first convolution layer.



Figure— First convolution layer

Figure— Final convolution layer

c. Count the probability of label change into cat or dogs

d. Train training data

e. Process test data

f. Use TensorFlow to design 2D CNN and use Max pooling

g. Confirm the input layer

h. Set dropout avoid overfit and design fully connected Feedforward Network

Discussion

The accuracy for TFlearn is 97.62 with 50 epochs, but for Keras is just 0.53. Time for each epoch is also different. For TFlearn, 24~25s for each epoch, but for Keras, at least 90s for one epoch. For this output we can see that TFlearn perform better than Keras not only in accuracy, but also in effectiveness. What's more, this is an interesting thing. In order to make a better comparation, I need to build the same CNN model for all the libraries. But if I change the model settings in Keras, like using activation function as sigmoid rather than softmax as the fully connected layer or reduce the number of convolution and max pooling layer to three, the accuracy of Keras will increase to 0.72 or so. Why this phenomenon happened?

First is effectiveness aspect. As we all know the purpose for creating Keras is just for operating Theano. Keras support Tensorflow to run after Tensorflow published. Thus, Keras is compatible with two backends: Tensorflow and Theano. Of course, its effectiveness is highly affected. However, TFlearn just have one backend: Tensorflow. Even though it is little slower than TensorLayer, but as least fast than Keras.

Second is accuracy aspect. The analyze for this can also answer that phenomenon above. Keras is really easy to use, it's friendly for the students who is a freshman in machine learning and deep learning. You don't need to define the structure and set each parameter on your own. You just need to call that function and set some parameters. It's easy to use, but not good to get a better accuracy. The reason why is that Keras is highly packaged, so many parameters and hyper-parameters have already packaged inside the model, you can't change all of them. Maybe some parameters are not fit in your dataset, but you are not allowing to change it. Thus, sometimes you can't increase your accuracy more. But for TFlearn, even though it's a higher API, some of parameters you can't

change, it is more freedom than Keras. For example, inside Keras, they don't support 'categorical_crossentropy' as loss function.

Last is the way I use to process the data. In TFLearn, I define a function on my own to process the image, like resize, gray scale and so on. But in keras, I just use Image Data Generator function which has already defined in keras. I think the way I process image is another important reason lead to high accuracy difference. That’s the part we still need to improve for this project.

We evaluate the success from two ways:

- According to the accuracy rate, we can roughly predict the result.
- In the output, we can judge whether the output result is correct or not based on the photo and the output text

```
In [17]: model.fit({'input': X}, {'targets': y},
                   n_epoch=50,
                   validation_set=({'input': Xtest}, {'targets': ytest}),
                   snapshot_step=500,
                   show_metric=True,
                   run_id='model' )

Training Step: 5899  | total loss: 0.04024 | time: 15.503s
| Adam | epoch: 050 | loss: 0.04024 - acc: 0.9848 -- iter: 7488/7505
Training Step: 5900  | total loss: 0.04226 | time: 16.643s
| Adam | epoch: 050 | loss: 0.04226 - acc: 0.9847 | val_loss: 1.59294 - val_acc: 0.6880 -- iter: 7505/7505
--
```

Figure— evaluate success1

Figure— evaluate success2

Results

A. Process data

In this part, first we define a function to create training data and create a list to save training data. We use a for cycle to remove the trash file which is not image. Then we read image's label. Using CV2 to do gray scaling for each image to make all images in a same size (50*50). Then putting gray scaled image and label in the training data list and shuffle the training data list.

```
def create_train_data():                          # Define a function to create training data.
    training_data = []                            # Create a list to save training data.
    for img in tqdm(os.listdir(train_dir)):       # Read each image from the training file.
        if (not img.endswith('.jpg')):            # Remove the trash file which is not image.
            continue
        label = label_img(img)                    # Read image's label.
        path = os.path.join(train_dir, img)
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)  # Use CV2 to do gray scaling for each image.
        img = cv2.resize(img, (img_size, img_size) )  # Make all images in a same size: 50*50, which as already defined above.
        training_data.append([np.array(img), np.array(label)])  # Put gray scaled image and label in the training data list.
    shuffle(training_data)                        # Shuffle the training data list.
    return training_data
```

B. Convolution & max pooling

Also, we have 5 times convolution and 5 times max pooling.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_1 (Conv2D)              (None, 384, 498, 3)       84
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 128, 166, 3)       0
_________________________________________________________________
activation_1 (Activation)    (None, 128, 166, 3)       0
_________________________________________________________________
conv_2 (Conv2D)              (None, 126, 164, 3)       84
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 63, 82, 3)         0
_________________________________________________________________
activation_2 (Activation)    (None, 63, 82, 3)         0
_________________________________________________________________
conv_3 (Conv2D)              (None, 61, 80, 3)         84
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 30, 40, 3)         0
_________________________________________________________________
activation_3 (Activation)    (None, 30, 40, 3)         0
_________________________________________________________________
conv_4 (Conv2D)              (None, 28, 38, 3)         84
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 19, 3)         0
_________________________________________________________________
activation_4 (Activation)    (None, 14, 19, 3)         0
=================================================================
Total params: 336
Trainable params: 336
```

Figure— Convolution &max pooling

C. Process training data

```
In [17]: model.fit({'input': X}, {'targets': y},
                   n_epoch=50,
                   validation_set=({'input': Xtest}, {'targets': ytest}),
                   snapshot_step=500,
                   show_metric=True,
                   run_id='model' )
```

Figure— training data

D. Epochs on TFlearn

We got 0.99 accuracy for test data.

```
Training Step: 5899  | total loss: 0.02690 | time: 14.076s
| Adam | epoch: 050 | loss: 0.02690 - acc: 0.9926 -- iter: 7488/7505
Training Step: 5900  | total loss: 0.02622 | time: 15.207s
| Adam | epoch: 050 | loss: 0.02622 - acc: 0.9918 | val_loss: 1.58480 - val_acc: 0.7180 -- iter: 7505/7505
--
```

E. CNN Model on Keras

This is our CNN model and parameters on every level.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_23 (Conv2D)           (None, 298, 298, 32)      896
_________________________________________________________________
activation_33 (Activation)   (None, 298, 298, 32)      0
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 149, 149, 32)      0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 147, 147, 32)      9248
_________________________________________________________________
activation_34 (Activation)   (None, 147, 147, 32)      0
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 73, 73, 32)        0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 71, 71, 64)        18496
_________________________________________________________________
activation_35 (Activation)   (None, 71, 71, 64)        0
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 35, 35, 64)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 78400)             0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                5017664
_________________________________________________________________
activation_36 (Activation)   (None, 64)                0
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 65
_________________________________________________________________
activation_37 (Activation)   (None, 1)                 0
=================================================================
Total params: 5,046,369
Trainable params: 5,046,369
Non-trainable params: 0
_________________________________________________________________
```

F. Training in Keras

Using ImageDataGenerator to process training data.

```
In [45]: batch_size = 128

         # Augmentation configuration for training data.
         train_data_generate = ImageDataGenerator(
                 rescale=1./255,
                 shear_range=0.2,
                 zoom_range=0.2,
                 horizontal_flip=True)

         # Rescaling
         test_data_generate = ImageDataGenerator(rescale=1./255)

         # Batches of augmented training data.
         train_generator = train_data_generate.flow_from_directory(
                 'training_set1',          # Training set address
                 target_size=(300, 300),  # Resize all images to 300*300
                 batch_size=batch_size,   # Set batch_size = 128
                 class_mode='binary')

         # Augmentation configuration for test data.
         validation_generator = test_data_generate.flow_from_directory(
                 'test_set1',
                 target_size=(300, 300),
                 batch_size=batch_size,
                 class_mode='binary')

         Found 8005 images belonging to 2 classes.
         Found 2023 images belonging to 2 classes.
```

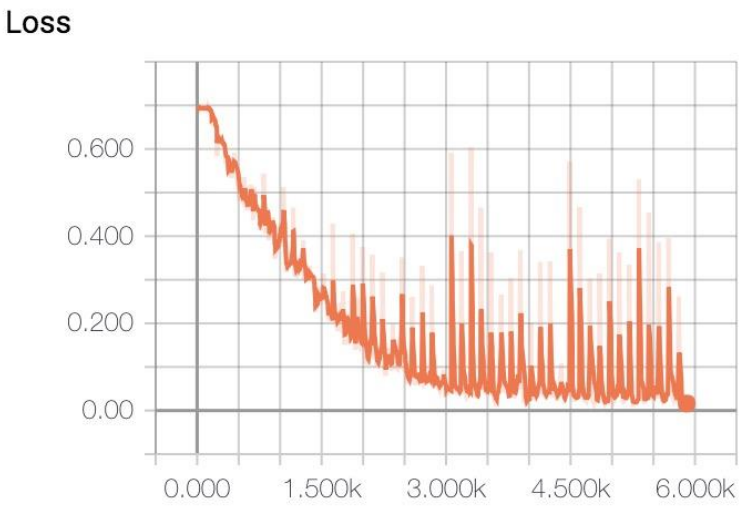
G. Epochs on Kears

From this picture, we can see the how the epochs run and how loss and accuracy changes.

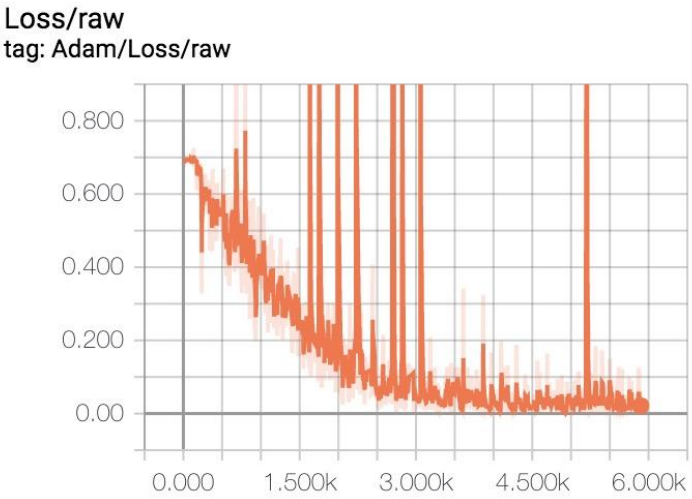
```
Epoch 45/50
3/3 [==============================] - ETA: 48s - loss: 0.5444 - acc: 0.74 - ETA: 23s - loss: 0.5765 - acc: 0.69 - 76s 25s/st
ep - loss: 0.5792 - acc: 0.7083 - val_loss: 0.5830 - val_acc: 0.6719
Epoch 46/50
3/3 [==============================] - ETA: 42s - loss: 0.5965 - acc: 0.65 - ETA: 20s - loss: 0.6173 - acc: 0.66 - 68s 23s/st
ep - loss: 0.6135 - acc: 0.6693 - val_loss: 0.5606 - val_acc: 0.7031
Epoch 47/50
3/3 [==============================] - ETA: 19s - loss: 0.6066 - acc: 0.65 - ETA: 16s - loss: 0.5693 - acc: 0.70 - 61s 20s/st
ep - loss: 0.5851 - acc: 0.6877 - val_loss: 0.5899 - val_acc: 0.6953
Epoch 48/50
3/3 [==============================] - ETA: 46s - loss: 0.5634 - acc: 0.67 - ETA: 24s - loss: 0.6517 - acc: 0.62 - 78s 26s/st
ep - loss: 0.6405 - acc: 0.6328 - val_loss: 0.5889 - val_acc: 0.6719
Epoch 49/50
3/3 [==============================] - ETA: 49s - loss: 0.6280 - acc: 0.69 - ETA: 25s - loss: 0.6134 - acc: 0.69 - 83s 28s/st
ep - loss: 0.6149 - acc: 0.6901 - val_loss: 0.5645 - val_acc: 0.7188
Epoch 50/50
3/3 [==============================] - ETA: 46s - loss: 0.6104 - acc: 0.67 - ETA: 24s - loss: 0.5911 - acc: 0.67 - 79s 26s/st
ep - loss: 0.5993 - acc: 0.6667 - val_loss: 0.5540 - val_acc: 0.7266

Out[47]: <tensorflow.python.keras.callbacks.History at 0x1aa52014ba8>
```

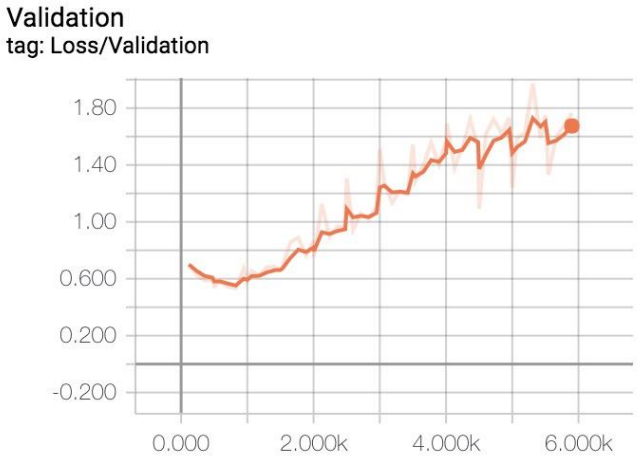
H. Tensorboard(TFlearm)



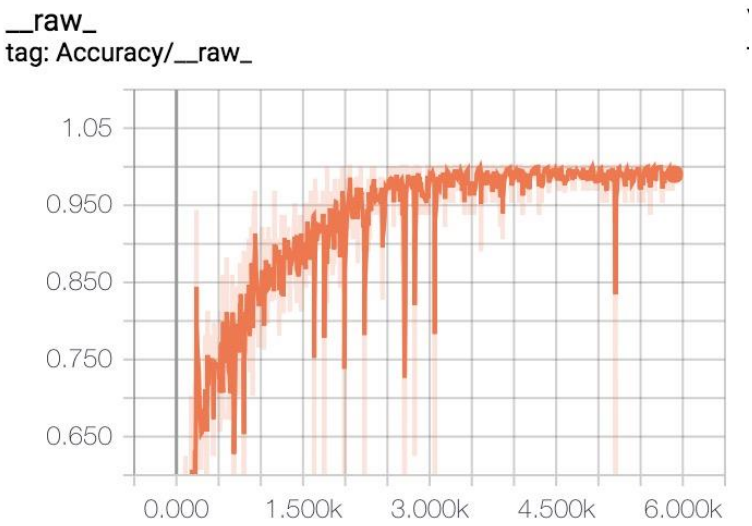
Figure—Loss of TFlearn



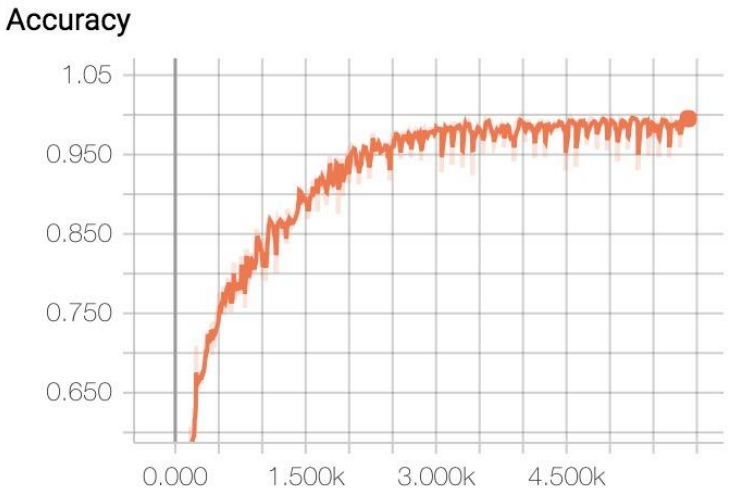
Figure—Loss of Adam Opt



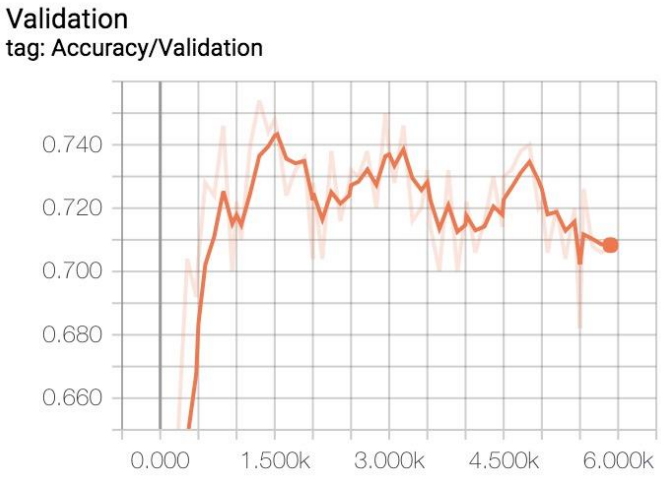
Figure—Loss / Validation



Figure—Accuracy/raw

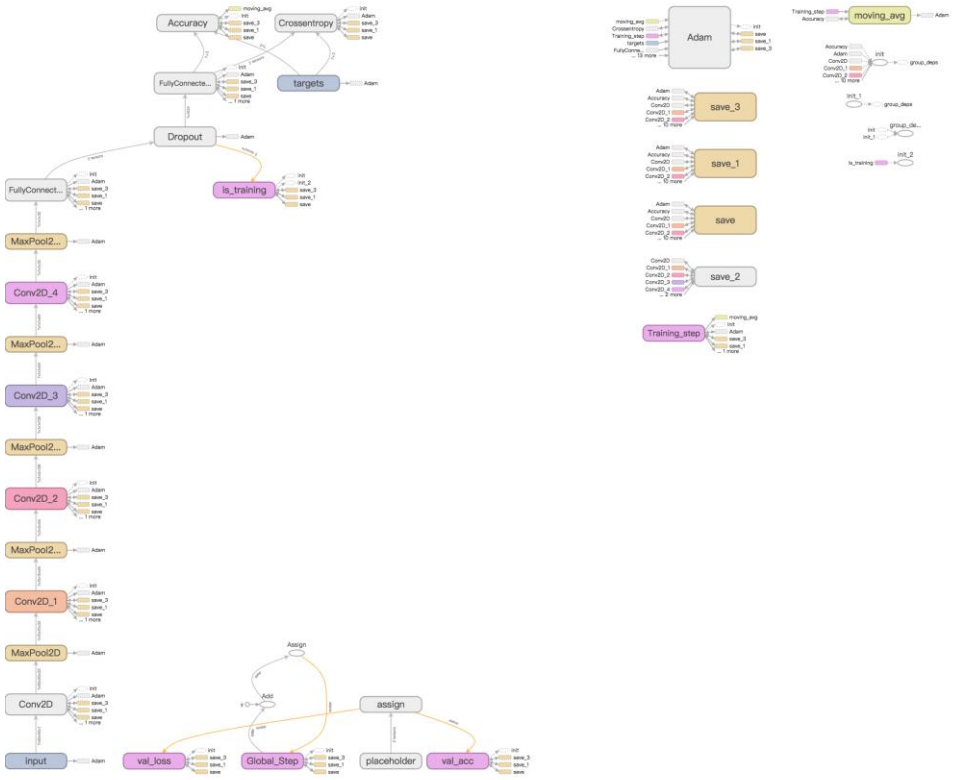


Figure—Accuracy of TFlearn



Figure—Accuracy /Validation

I. Create model



Figure—Create model

Í

References

[1] Bang, Liu. Yan, Liu. Kai, Zhou. Image Classification for Dogs and Cats

[2] “TFlearn: Deep learning library featuring a higher-level API for Tensorflow.” http://tflearn.org/

[3] "Convolutional Neural Networks (LeNet) – DeepLearning 0.1 documentation". DeepLearning 0.1. LISA Lab. Retrieved 31 August 2013.

[4] Graham, Benjamin (2014-12-18). "Fractional Max-Pooling". arXiv:1412.6071 [cs.CV].

[5] "Image Classifier using CNN". Nature. 521 (7553): 436–444. Bibcode:2015Natur.521..436L. doi:10.1038/nature14539. PMID 26017442. https://www.geeksforgeeks.org/image-classifier-using-cnn/

[6] Rock, Irvin. "The frame of reference." The legacy of Solomon Asch: Essays in cognition and social psychology (1990): 243–268..

[7] https://en.wikipedia.org/wiki/Convolutional_neural_network

[8] J. Wang, “Fundamentals of erbium-doped fiber amplifiers arrays (Periodical style—Submitted for publication),” *IEEE J. Quantum Electron.*, submitted for publication.

[9] C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.

[10] Haralick, Robert M., and Karthikeyan Shanmugam. "Textural features for image classification." IEEE Transactions on systems, man, and cybernetics 6 (1973): 610-621

[11] “TFLearn - Deep Learning with Neural Networks and TensorFlow p. 14” https://www.youtube.com/watch?v=NMd7WjZiCzc

[12] “How to create convolutional neural network || dog-cat classifier using tflearn and python”. https://www.youtube.com/watch?v=uDSKgSEh4NM