CSE5523 Final Project Report

Cat or Dog Image Classification

Rita Tong
Tai-Yu Pan
Aaron Rubenacker

Abstract:

The image classification is an important field in the machine learning area. The project is based on serval deep learning models to classify whether an image is a dog or cat.

Project Overview

For this project, the group uses two different methods of training to create a network to classify images. Specifically, the group trained the neural network to classify images based on whether they contained a dog or a cat. The data that the group used is currently available on the Kaggle website and is 25,000 training images labeled dogs or cats. The group separates these images into two groups: the training data set and validation data set (4:1). There are also 12,500 test images without labels. One method used was looking directly at pixels and using only that information. It was not transformed in any way and used a weight matrix to evaluate each picture. The first method accuracy is around 50% which needs to improve. Then the group implements a second method. Our second method was using a convolutional neural network.

Method

Softmax:

Softmax regression is a general format of logistic regression. The function contains a K-dimension vector R whose entries add up to 1. The standard softmax function is calculated by an exponential function on each coordinate, normalizing the format by divided by the sum of all the coordinates. Thus the sum of the output coordinates is 1.

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

The softmax stands for the smooth approximation to the argmax function. The softmax function could be used in multiclass classification. It will calculate different classes' probability and return the highest-probability class. It could be used in multiple classification logistic regression model. The group also use this method in the last neural network layer to output the result.

Convolutional Neural Networks:

The Convolutional Neural Networks are the special type of neural networks. The network takes the raw image pixels as input data. The data will go through several different layers and the result will be the target classes score vector. There are four basic steps:
Step - 1: Convolution layer
Step - 2: ReLU function
Step - 3: Pool layer
Step - 4: Full connection

The Convolutional layer will compute the output neuron by doing the dot product between weights and each small region add the bias value. RELU layer is an activation function layer.

For example, it will use f(x)=max(0,x) to add nonlinear function on the layer. The Pooling layer is also a non-linear downsampling function. Last, the Full connection layer will compute the class score which is in form of a K-dimension vector in which K is the number of class types.

Environments and Parameters

Our model is built in the following environment(main packages).
Python == 3.5.6
tensorflow == 1.10.0
tflearn == 0.3.2
image size == 32x32

Softmax

```
# Define the classifier's result
logits = tf.matmul(images_placeholder, weights) + biases

# Define the loss function
loss =
tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits=log
its, labels=labels_placeholder))

# Define the training operation
train_step =
tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

# Operation comparing prediction with true label
correct_prediction = tf.equal(tf.argmax(logits, 1), labels_placeholder)

# Operation calculating the accuracy of our predictions
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

First, we define a variable called logits to perform a dot product with all input and weights and then add a bias term. By applying the softmax function, we can transform these values into probabilities (see Method for more information). These probabilities will compare to the true probability distribution, where 1 is for the correct class and 0 for all other classes. We use a method called cross-entropy to compare two distributions and this is exactly our loss function. We apply the gradient descent method to minimize the loss function by using a learning rate of 0.005. A max step is set to 1000 to see a preliminary result.

Convolutional neural network

```
def create_model():
    convnet = input_data(shape=[None, IMG_SIZE, IMG_SIZE, 3],
```

```
name='input')

    conv1 = conv_2d(convnet, 32, 2, activation='relu', name='conv1')
    conv1_ = max_pool_2d(conv1, 2)

    conv2 = conv_2d(conv1_, 64, 2, activation='relu', name='conv2')
    conv2_ = max_pool_2d(conv2, 2)

    conv3 = conv_2d(conv2_, 32, 2, activation='relu', name='conv3')
    conv3_ = max_pool_2d(conv3, 2)

    conv4 = conv_2d(conv3_, 64, 2, activation='relu', name='conv4')
    conv4_ = max_pool_2d(conv4, 2)

    conv5 = conv_2d(conv4_, 32, 2, activation='relu', name='conv5')
    conv5_ = max_pool_2d(conv5, 2)

    conv6 = conv_2d(conv5_, 64, 2, activation='relu', name='conv6')
    conv6_ = max_pool_2d(conv6, 2)

    fc1 = fully_connected(conv6_, 1024, activation='relu', name='fc1')
    fc1_ = dropout(fc1, 0.8)

    fc2 = fully_connected(fc1_, 2, activation='softmax', name='fc2')
    fc2_ = regression(fc2, optimizer='adam', learning_rate=LR,
loss='categorical_crossentropy', name='targets')

    model = tflearn.DNN(fc2_, tensorboard_dir='log')
    return model
```

In this model, the shape of an input image is (32, 32, 3). In other words, the input data didn't flatten as we did in the Softmax model. Refer to LeNet and AlexNet, We applied 6 convolutional layers to extract features from images in our model. In each convolutional layer, we apply ReLU function as the activation function to reach nonlinearity and use max pooling method as the downsampling function. In addition, the filter size of (2, 2) is used. There are two fully connected layers in our CNN model. The first FC layer flattens the output from the convolutional layer into a 1-D array of size of 1024. And then we apply a dropout function to prevent from overfitting by applying a possibility of 0.8. In the second FC layer, we apply Softmax function again to get the probability of each class. Adam is a commonly used optimizer of the loss function and a learning rate of 0.001 is set.

Result and Conclusion

Softmax

```
Step     0: training accuracy 0.5
Step   100: training accuracy 0.49
Step   200: training accuracy 0.6
Step   300: training accuracy 0.45
Step   400: training accuracy 0.47
Step   500: training accuracy 0.5
Step   600: training accuracy 0.47
Step   700: training accuracy 0.57
Step   800: training accuracy 0.45
Step   900: training accuracy 0.51
Test accuracy 0.5554
Total time: 36.22s
```

In the Softmax model, we get an accuracy of 55.54% in the test set.

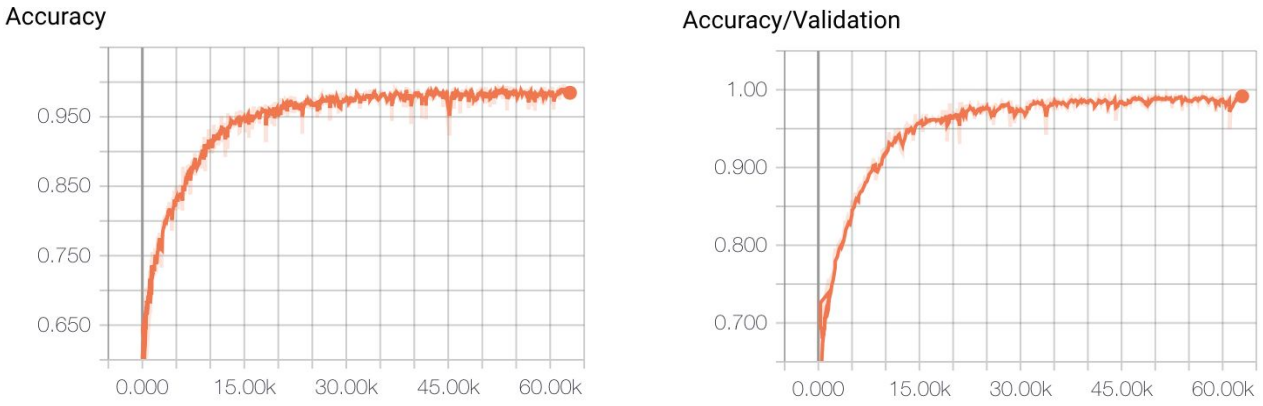
Convolutional neural network

Computer Environment:
model == MacBook Pro (15-inch, 2016)
processor == 2.9 GHz Intel Core i7
Graphics == Radeon Pro 460 4096 MB
It takes in total 16 hrs 33 mins to train the model for 200 epoch containing 62913 steps.

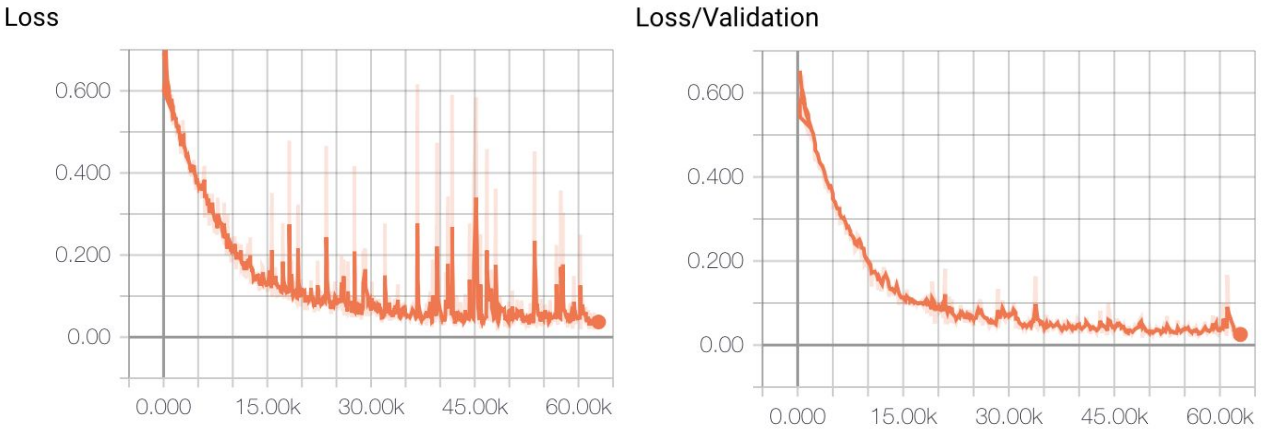
```
Training Step: 61500  | total loss: 0.06957 | time: 88.467s
| Adam | epoch: 196 | loss: 0.06957 - acc: 0.9804 | val_loss: 0.05519 - val_acc: 0.9812 -- iter: 09728/20000
--
Training Step: 61661  | total loss: 0.03424 | time: 179.967s
| Adam | epoch: 196 | loss: 0.03424 - acc: 0.9900 | val_loss: 0.04190 - val_acc: 0.9888 -- iter: 20000/20000
--
Training Step: 61974  | total loss: 0.05505 | time: 145.492s
| Adam | epoch: 197 | loss: 0.05505 - acc: 0.9888 | val_loss: 0.02407 - val_acc: 0.9910 -- iter: 20000/20000
--
Training Step: 62000  | total loss: 0.02000 | time: 43.913s
| Adam | epoch: 198 | loss: 0.02000 - acc: 0.9942 | val_loss: 0.02151 - val_acc: 0.9922 -- iter: 01664/20000
--
Training Step: 62287  | total loss: 0.03045 | time: 180.257s
| Adam | epoch: 198 | loss: 0.03045 - acc: 0.9914 | val_loss: 0.02636 - val_acc: 0.9919 -- iter: 20000/20000
--
Training Step: 62500  | total loss: 0.03250 | time: 109.991s
| Adam | epoch: 199 | loss: 0.03250 - acc: 0.9860 | val_loss: 0.03269 - val_acc: 0.9849 -- iter: 13632/20000
--
Training Step: 62600  | total loss: 0.01942 | time: 180.209s
| Adam | epoch: 199 | loss: 0.01942 - acc: 0.9927 | val_loss: 0.02158 - val_acc: 0.9936 -- iter: 20000/20000
--
Training Step: 62913  | total loss: 0.03722 | time: 145.902s
| Adam | epoch: 200 | loss: 0.03722 - acc: 0.9833 | val_loss: 0.02176 - val_acc: 0.9935 -- iter: 20000/20000
```

Snapshot for last 5 epoch

Accuracy vs steps on the training set and test set respectively:

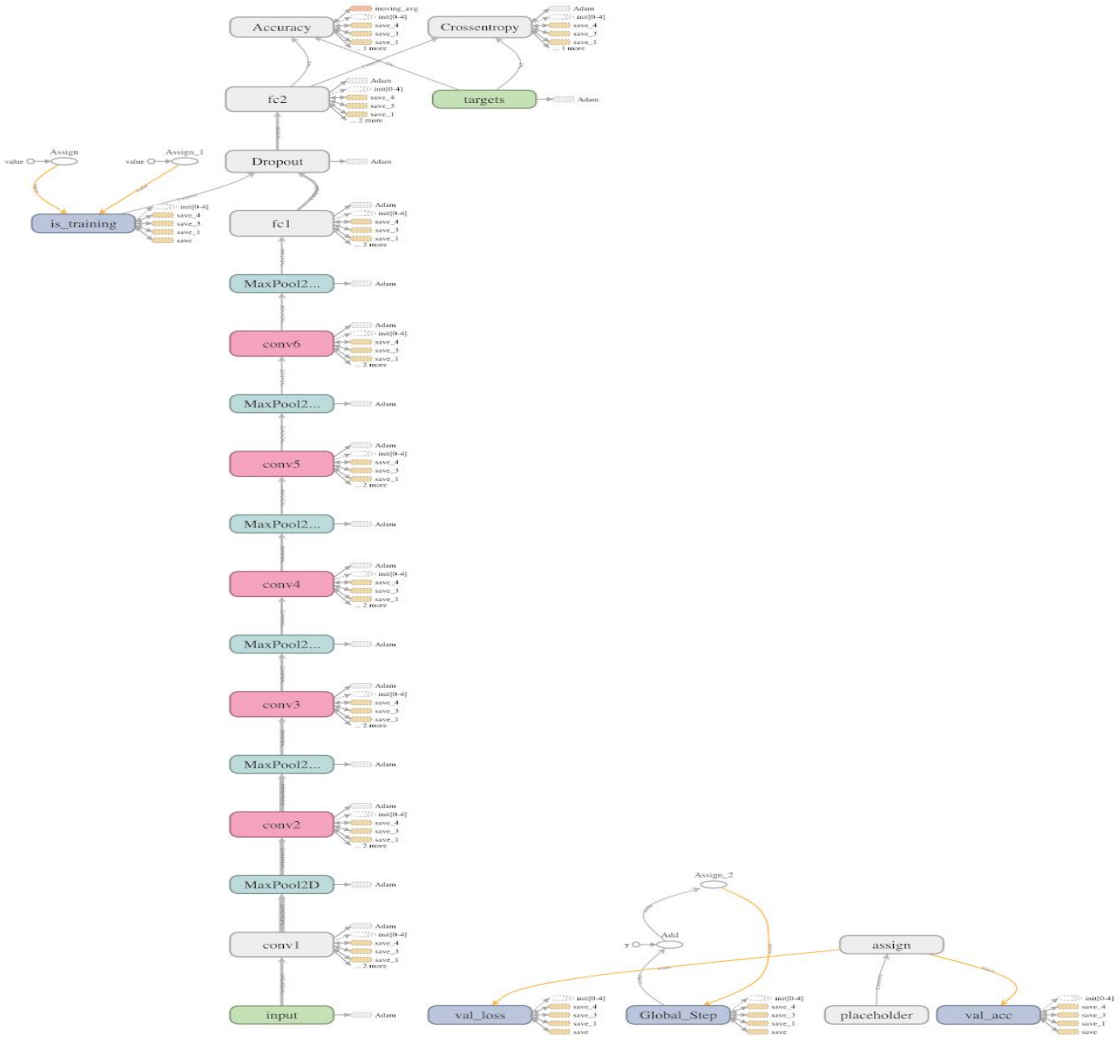


Loss values vs steps on training set and test set respectively:



A sample of the output:

Overall CNN Structure:



Discussion

Softmax is a great logistic classifier. Actually, in the data set of handwriting numbers, it can reach an accuracy of 97%. At the first step of our project, we want to do a quick test on the data set by applying Softmax. Unfortunately, we got an accuracy of 55.54% which is very low because we can even reach 50% by assigning all to dogs. Here are some reasons that Softmax didn't work on this data set. First, in handwriting numbers, color doesn't matter, which means it can be transferred to black and white image. However, in our case, dogs and cats have many different colors corresponding to different kinds of species. We should take this feature into consideration. Second, handwriting numbers have much fewer features. In handwriting numbers, features may be some straight lines, angles, and curves. In our case, the appearance of dogs and cats are totally different. Features may contain colors, as stated previously, eyes, tails, ears, noses, mouths, etc. Third, the most import reason, in Softmax, the place of the pixels matters. However, for example, when we look at an image, we will

know it’s a dog or cat no matter it appears on the bottom left or top right. Thus, according to these three reasons, we need a further technique to classify dogs and cats.

Convolutional Neural Network perfectly solves the problems we met in Softmax. In convolutional layers, we can input an image with 3 channels (corresponding to RGB). In addition, we try to extract features from the input and make them into new input. By doing this multiple times, we can extract some high-level features. At the last step, we use these features to feed into Softmax again. Amazingly, we got an accuracy of 99% on both training set and test set after 17-hour training. If we take a further look at incorrect predictions, some of them even couldn’t be recognized by humans.

Another interesting point is that the image we use as input is only 32*32 pixels. This size is even too small for humans eyes. However, surprisingly, it doesn’t matter too much in our model. We still got a very high accuracy.

Reference:

1. Data: https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data
2. TensorFlow: https://www.tensorflow.org/
3. tflearn: https://github.com/tflearn/tflearn