Convolutional Neural Networks and Image Classification

Bolei Chen

Yilin Hou

Yitong Zhou

Professor:

Teaching Assistant:

Date Submitted: 05/14/2020

TABLE OF CONTENTS

ABSTRACT

This report introduces the team's understanding of convolutional neural network(CNN) and image classification. The CNN was constructed with Tensorflow for image classification. Users can get image classification results by uploading image paths in the user interface. The team got about 80 percent right. And summary and analysis of the work were provided at the end.

INTRODUCTION

Convolutional neural network (CNN) has achieved unprecedented success in the field of computer vision. At present, many traditional computer vision algorithms have been replaced by deep learning. Due to its huge commercial value, deep learning and convolutional neural network have become the research hotspot. Image recognition is the most important and basic problem in the field of computer vision. Other tasks, such as target detection, image segmentation, image generation and video comprehension, are highly dependent on the ability of feature expression in image recognition. The latest development of convolutional neural network in image recognition can directly affect the performance of all computer vision tasks based on deep learning, so it is particularly important to understand this progress in depth.

In order to understand CNN and image classification better, the team used the TensorFlow environment and Keras to build a model to identify cats and dogs.The classification of cats and dogs is significant. In order to protect the Web service, it will prevent some computers from malicious access or information crawling, and then set up some authentication problems, which are easy for people to do, but difficult for computers. For example, Asirra (animal image recognition used to restrict access) allows users to identify image information, such as whether it is a cat or a dog. Random guesses are generally the easiest way to attack a computer to break into. It is not easy to identify the pictures, because there are huge differences in the background, angle, posture, brightness and so on between the pictures, so it is difficult to identify.

Through this project, team members can get familiar with the basic usage and use environment of Keras and lay a foundation for the later implementation of more in-depth learning networks by using Keras and TensorFlow

PROCEDURES

- **Preparatory work:**

  dataset: Dogs and Cats

  Programming language: Python 3

  Machine learning libraries: Numpy, sklearn, Keras, Tensorflow, matplotlib

  Applications: jupyter notebook

- **Data preparation:**

  Since the computer can't directly manipulate images, videos, text, etc., the first thing to do is to turn the images into a numerical matrix, and make sure the images you train are the same size. The image was scaled to find the optimal value of image size, a standard size that is smaller than the actual image resolution, that not only shows. So the images was narrowed all down to 70 by 70 for further processing which can also take the less memory.

  After all the cat and dog images have been processed in this way, save them into a data array, ignore the damaged photos and resize the array. And splitting the training set to the feature set and the label set finally. The data were shuffled before splitting for better accurracy.

- **Model building and training:**

  To begin by defining the model format, Keras provides several different formats to build the model. Sequential is the most commonly used and therefore needs to be imported

from Keras. The first layer of the model is the convolution layer, which will receive the input images and perform a convolution filter on them. When implementing in Keras, the number of channels/filters (32 in this case), the filter size (in this case 3×3), the shape of the tensor (when creating the first layer), the activation function, and the fill should be specified. Relu is the most common activation function. The number of layers can vary according to individual needs, but each additional layer adds computation. There are 2 convolution layers in the model. Then there is the pooling layer, which helps make the image classifier more robust so that it can learn related patterns. After processing the convolutional layer, the Flatten data is required. And use Dense to create the first Dense layer. The model contains two dense layers.

Now that we've designed the model we want to use, we just need to compile it. Firstly, specify the number of epochs (in this case, 3) to train and the optimizer to use. The optimizer will adjust the weights in the network to get closer to the lowest loss point. Among them, Adam algorithm is one of the most commonly used optimizers, because it shows excellent performance on most problems. Then, compile the model with the selected parameters and specify the metrics (accuracy in this case) to use.

Now the model can be training, which fit() function can be implemented and passing the relevant parameters that were set. In addition, Tensorflow visualization is also implemented. TensorBoard is used to clearly see the variation trend of loss function and accuracy.

- **Created user interface**

  We built the graphical user interface using the Tkinter module. Firstly, we created an entry that stores the input data (in this case, the path of the testing image). Then we built a button which will run the classification function using the model we trained before when it is clicked. Lastly, we programmed a label showing the output data (in this case, the class predicted).

RESULT

Through the above methods, the accuracy of the model is about 80% after training and testing, which means when users upload images to do the classification using this model, the probability of that they will get correct classification is eighty percent.

CALCULATIONS AND GRAPHS

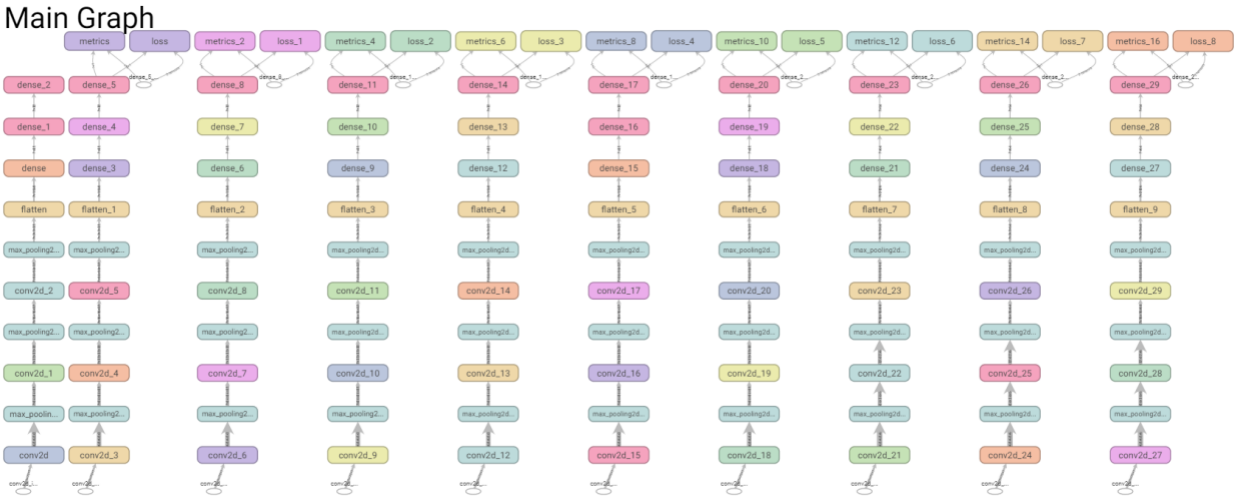

Figure 1.1 Op-Level Graph

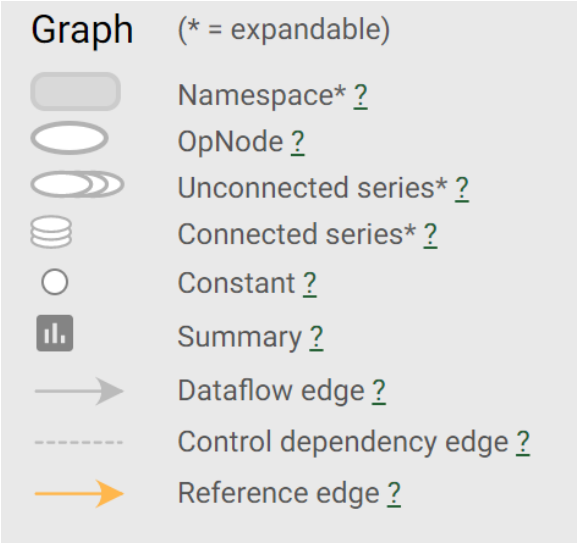

Figure 1.1.1 Interpretation of the characters

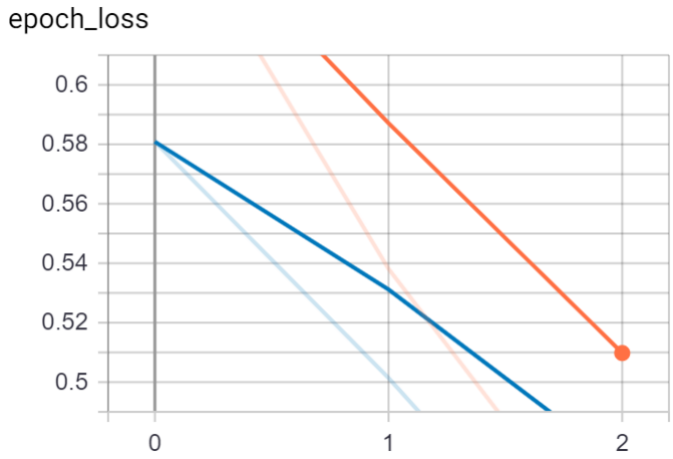

Figure 1.2 epoch loss

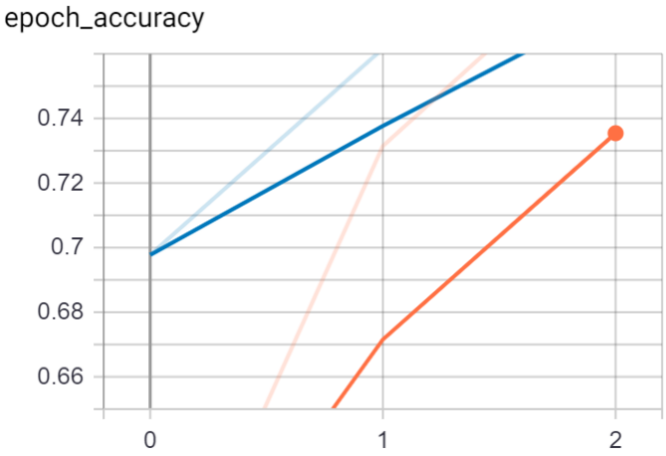

Figure 1.3 epoch accuracy

```
Train on 17462 samples, validate on 7484 samples
Epoch 1/3
17462/17462 [==============================] - 124s 7ms/sample - loss: 0.6688 - accuracy: 0.5717 - val_loss: 0.5809 - val_
accuracy: 0.6978
Epoch 2/3
17462/17462 [==============================] - 126s 7ms/sample - loss: 0.5381 - accuracy: 0.7314 - val_loss: 0.5014 - val_
accuracy: 0.7616
Epoch 3/3
17462/17462 [==============================] - 126s 7ms/sample - loss: 0.4355 - accuracy: 0.7968 - val_loss: 0.4143 - val_
accuracy: 0.8112
```

Figure 1.4 Loss and Accuracy Metrics

DISCUSSION

The results, 80% accuracy, are good for the first run, but the model structure and parameters can be adjusted like removing or adding convolutional layers, changing the filter size, or even changing the activation functions to achieve better performance.

To get higher accuracy, more aggressive data augmentation like Neural Style Transfer or Generative Adversarial Networks and dropout will be helpful. Several parameters such as the number and the type of layers can also be changed to modify and improve the algorithm. Moreover, the model may have the problem of overfitting. Overfitting can be solved by adding a dropout layer or simplifying the network architecture. However, it is difficult to calculate exactly how to reduce overfitting without trial and error.

CONCLUSION

In this project, the image classification network is implemented through Keras, but it is better to repeatedly study the model and observe the impact of parameter adjustment on its performance. This will provide intuition about the best choice of different model parameters. When doing this, the programmer should also be aware of the different parameters and hyperparameter options. And then you can try to implement your own image classifier on different data sets.

BIBLIOGRAPH

Chollet, F. (n.d.). The Keras Blog. Retrieved from https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Google Colaboratory. (n.d.). Retrieved from https://colab.research.google.com/github/tensorflow/tensorboard/blob/master/docs/graphs.ipynb

Shorten, C. (2019, April 22). Image Classification Python/Keras Tutorial: Kaggle Challenge. Retrieved from https://towardsdatascience.com/image-classification-python-keras-tutorial-kaggle-challenge-45a6332a58b8