Computational Methods for Data Scientists – Identifying CIFAR – 10 Image Data Set using Convolutional Neural Networks in Deep Learning

Stephen Pohl
Universtiy of Texas at Dallas
Data Science Undergraduate
sjp170330@utdallas.edu

Jaemin Lee
Universtiy of Texas at Dallas
Data Science Undergraduate
jxl142430@utdallas.edu

***Abstract*—The term deep learning refers to artificial neural networks with multi-layer perceptrons. Over the past decades, deep learning has been considered to be one of the most powerful tools. It's widely implemented for handling a huge amount of data. The implementation of deeper hidden layers has recently begun to surpass classical methods performance in different fields; especially in pattern recognition. The purpose of this document is to perform image classification using CNNs on CIFAR-10 data set.**

***Key words* – Convolutional Neural Networks, Deep Learning, Computer Vision, CIFAR – 10 Data Set, TensorFlow, Keras**

I. Introduction

CNN stands for Convolutional Neural Networks, a type of deep-learning model used in computer vision applications. CNNs are used widely in image classification problems. They are regularized versions of multilayer perceptrons. Usually, multilayer perceptrons are fully connected networks, which means that each neuron in a layer is connected to all the neurons in the following layer. The reason why CNNs are powerful is due to it being fully-connected, so it prevents the network from overfitting the data [1]. Generally, regularization includes adding some kind of magnitude measurement of weights to the loss function. It is a technique which makes slight modifications to the learning method such that the model generalizes better This in turn improves the model's performance on the unseen data as well. However, CNNs uses regularization a bit differently. They implement hierarchical pattern in data and make complex patterns simpler using smaller patterns. So, in terms of complexity, CNNs have high advantages.

The background of convolutional networks comes from biological neurons. It implements connectivity between neurons of living things. Individual neurons respond to stimuli only in a restricted region of the visual field known as the receptive field [1]. The receptive fields of different neurons partially overlap so that they cover the entire visual field.

CNNs implement feature maps to help identify patterns in like images. Feature maps have two axes, the first axis is for the height and width of the feature map [1]. The second axis is a depth axis, the depth of a feature map is decided by the colors within the image that the feature map is being applied too, for an RGB image the depth would be 3 for each of the colors: red, green and blue. For a black and white image, the depth would be 1 as it is only measuring the levels of gray.

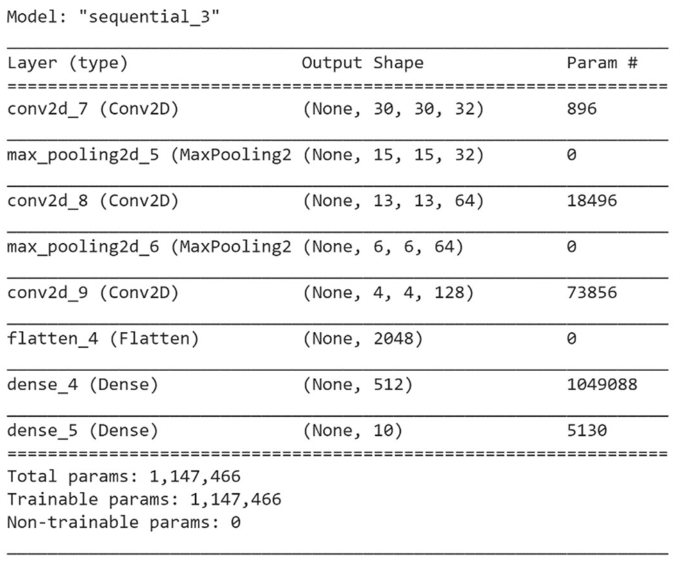
II. Our Task and Data Set

A. *About Data Set*

The data set we attempted to work on is called the "CIFAR – 10" data set. It consists of total number of 60,000 32 x 32 color images in 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck), with 6000 images per class. Out of 60,000 images, 50,000 images are training data and the remaining images (10,000) are used as test data. The data set is divided into five training batches and one test batch, each with 10,000 images. The test batch contains exactly 1000 randomly chosen images from each class. the training batches contain the remaining images in random order, but some training batches may contain more images from one class than another [2]. Between them, the training batches contain exactly 5000 images from each class.

Fig. 1: One of the training images (i.e. frog)

Fig. 2: One of the test images (i.e. cat)



```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 4, 4, 128)         73856
_________________________________________________________________
flatten_4 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               1049088
_________________________________________________________________
dense_5 (Dense)              (None, 10)                5130
=================================================================
Total params: 1,147,466
Trainable params: 1,147,466
Non-trainable params: 0
_________________________________________________________________
```

Fig. 3: Model Summary of Simple CNN

B. *Our Task*

Our task is to classify test images into 10 classes by building CNN using 50,000 training images. We want to see how well a CNN created from the ground up can be improved by implementing a pre-trained network. Before applying the pre-trained network, we will need to get our initial CNN working, once that stipulation is satisfied, we can simply add the pretrained network onto the existing CNN. We want to compare the accuracy between a simple CNN model and the CNN after the addition of the pre-trained network. This experiment should serve as a strong showing of the power of implementing a pre-trained network onto an existing CNN.

III. Experiments and Results

A. *TensorFlow with Keras*

For this particular experiment, we implemented TensorFlow with Keras. TensorFlow focuses on simplicity so it helps users to easily implement with intuitive higher-level APIs. Thus, it's very flexible in terms of model building on any platform. Also, we used Google Colab – a hosted notebook environment doesn't require any setup to use TensorFlow. Keras is a high-level API that builds and train models. It makes TensorFlow easier to implement without sacrificing flexibility and performance [3].

B. *Simple CNN*

This part of the experiment accounts for the simple convolutional neural network on CIFAR-10 data set. We used the splits given by the built-in data set. 50,000 images are training images and 10,000 images are test images. Our network design contains three Conv2D layers, two MaxPooling layers, one Flatten layer, and two Dense layers. The first dense layer has the ReLu activation function and the last dense layer accounts for SoftMax.

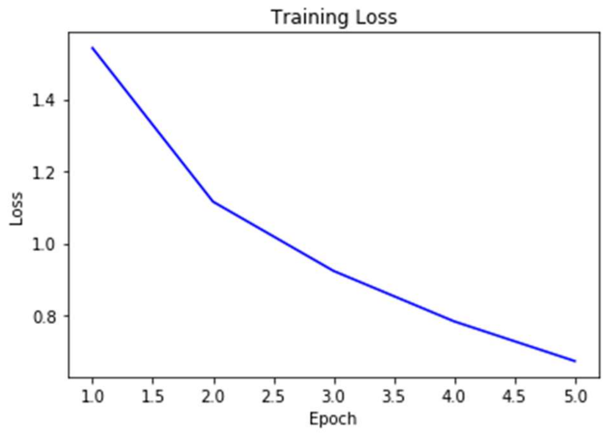


Fig. 4: Plot of Training Loss

Based on figure 4, it is apparent that training loss decreases for each epoch. This is expected as each epoch uses previous epochs to train itself and adjust its weights to help minimize the loss.

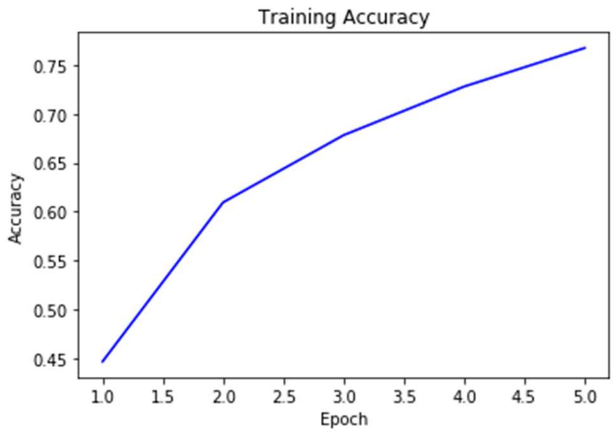


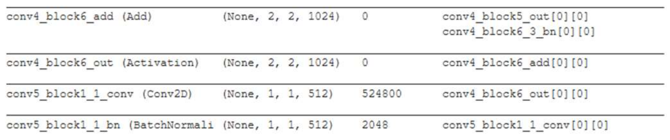
Fig. 5: Plot of Training Accuracy for Simple CNN

Figure 5 represents how training accuracy increases for each epoch. In relation to Figure 4, training accuracy increases as traing loss decreases. This shows us that the model is learning each epoch.

After the fifth epoch, we ran the current CNN against the test data. The test accuarcy ended up with 64.8%. While not great, this is expected as our CNN is not deep and it was only run for five epochs.

C. *Pre-trained CNN*

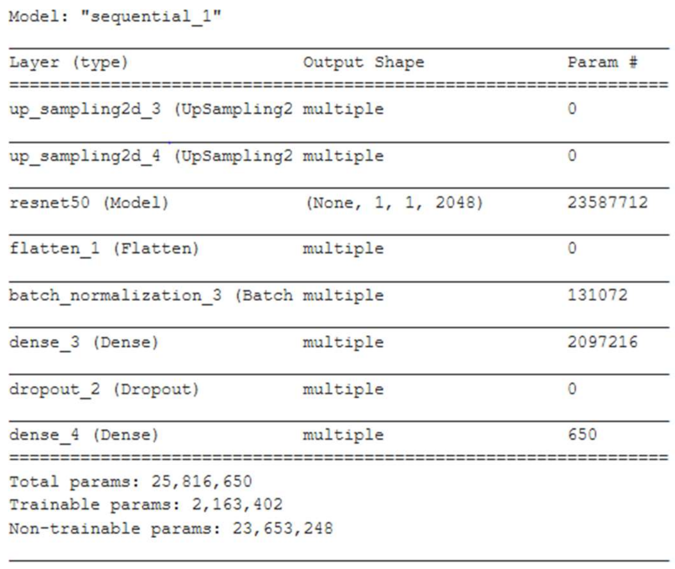
Next objective was to run and use a pretrained network on the CIFAR-10 data. We want to show that a pre-trained network will perform better than our simple CNN network. We chose to use the ResNet50 pre-trained CNN since it is a network of more than a million images from ImageNet database. It is 50 layers deep and has 1000 object categories [4]. While our objective is to only categorize the images into 10 object categories ResNet50 image recognition will be more refined and will do a much better job than our simple CNN.



```
conv4_block6_add (Add)          (None, 2, 2, 1024)   0          conv4_block5_out[0][0]
                                                                conv4_block6_3_bn[0][0]
____________________________________________________________________________________________
conv4_block6_out (Activation)   (None, 2, 2, 1024)   0          conv4_block6_add[0][0]
____________________________________________________________________________________________
conv5_block1_1_conv (Conv2D)    (None, 1, 1, 512)    524800     conv4_block6_out[0][0]
____________________________________________________________________________________________
conv5_block1_1_bn (BatchNormali (None, 1, 1, 512)    2048       conv5_block1_1_conv[0][0]
```

Fig. 6: Design of ResNet50

One problem in using ResNet50 as seen in figure 6 is that the image will end in a 1 by 1 state, while we want to shrink the image to find the filters, we don't want to make it into an unusable size. We will fix this issue by using upsampling to increase the dimensions of the image so that it will have a shape bigger than 1 by 1 after being ran through the ResNet50 pre-trained network.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
up_sampling2d_3 (UpSampling2 multiple                  0
_________________________________________________________________
up_sampling2d_4 (UpSampling2 multiple                  0
_________________________________________________________________
resnet50 (Model)             (None, 1, 1, 2048)        23587712
_________________________________________________________________
flatten_1 (Flatten)          multiple                  0
_________________________________________________________________
batch_normalization_3 (Batch multiple                  131072
_________________________________________________________________
dense_3 (Dense)              multiple                  2097216
_________________________________________________________________
dropout_2 (Dropout)          multiple                  0
_________________________________________________________________
dense_4 (Dense)              multiple                  650
=================================================================
Total params: 25,816,650
Trainable params: 2,163,402
Non-trainable params: 23,653,248
_________________________________________________________________
```

Fig 7: Model Summary of Pre-trained CNN

We can see that the parameter size of the CNN in figure 7 is much larger than that of the one in figure 3, this is to be expected as ResNet50 is a much larger and more robust CNN than our original simple CNN.

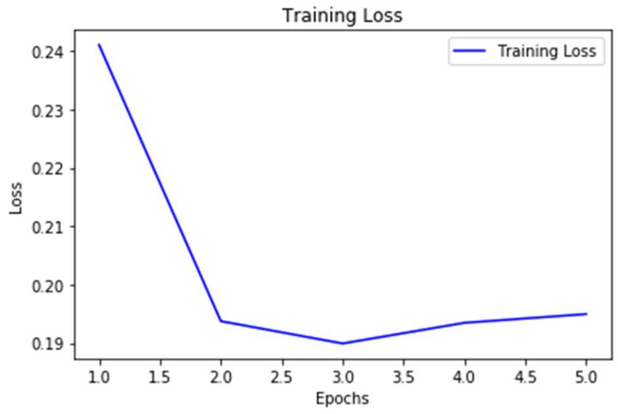


Fig. 8: Plot of Training Loss for Pre-trained CNN

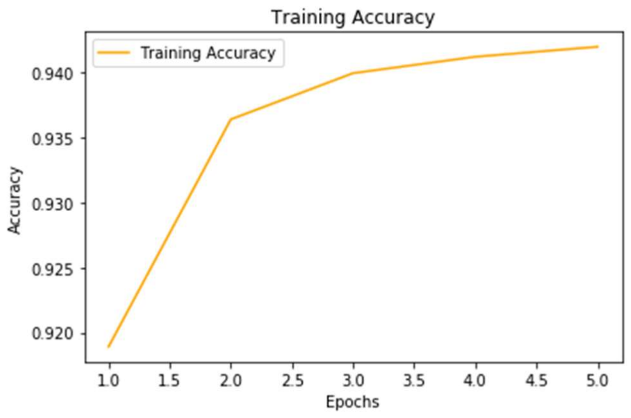


Fig. 9: Plot of Training Accuracy for Pre-trained CNN

The results of using a pre-trained network is immediately noticeable since after just the first epoch the training accuracy is already 91.9% much higher than any number achieved in the simple CNN. Test accuracy is much better as well, ending at 90% this result is 25.2% better than the simple CNN.

IV. Conclusion

We observed that pre-trained outperforms simple CNN. The only tradeoff in using pretrained over simple CNN is that pre-trained takes much longer to run. However, results are well worth the extra run time. We learned how much better pre-trained performs better compared to simple CNN. Also, we learned how to set up simple CNN ground up. If we had more time, we would like to be able to import the images ourselves rather than using the built-in images. We believe that there is a room for an improvement for our approach in building simple CNN and pre-trained network. With better execution, we could achieve better results.

References

[1] Chollet François, Deep learning with Python. Shelter Island, NY: Manning Publications Co., 2018, ch. 5

[2] Cs.toronto.edu. (2019). CIFAR-10 and CIFAR-100 datasets. [online] Available at: https://www.cs.toronto.edu/~kriz/cifar.html [Accessed 01 Dec. 2019].

[3] “Keras overview : TensorFlow Core,” TensorFlow. [Online]. Available: https://www.tensorflow.org/guide/keras/overview. [Accessed: 01-Dec-2019].

[4] “DAGNetwork,” Pretrained ResNet-50 convolutional neural network - MATLAB. [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/resnet50.html. [Accessed: 01-Dec-2019]