DSC 521 Cifar10 Models

Nolan Stelzner

Table of Contents

Summary:

This project was a fun but tedious task of finding a good performing model as a result of fine tuning the multiple layers, dropouts, activation functions, and optimizers. I found that the 'Adam' optimizer seemed to provide the best results for all my models. After some research it seems that this is common unless you are adjusting the learning rate of each layer, something I did not do. The two most significant performance functions that helped the most were in max pooling and batch normalization. I found that adjusting the number of layers & having a variety of activation functions minimally helped performance. The same goes for the dropout percentages. However, I attempted to use a smaller dropout in my early layers and then increase them throughout the network as I wanted to somewhat smooth out the number of neurons lost throughout the models. I started my attempts of finding the best model by using the hyperparameter tuning module in TensorFlow but got bad results. Most of my models were in the 15-35% accuracy range. This was a mix of dropouts, relu & sigmoid activation functions, sgd and adam optimizers, a variety of dense layers, and multiple different epochs. I think the turning point was including some max pooling into the models; however, I was not able to figure out how to include that into the hyperparameter tuning functions, so I moved on to manually adjusting and running these models. The GPU really sped up the training time of my models, but I could only run my models once and up to 15 epochs because I ran out of RAM on the Colab platform. I did include a couple of examples of the GPU performance in this document so you can see how significant the training times per epoch were improved on the exact same model.

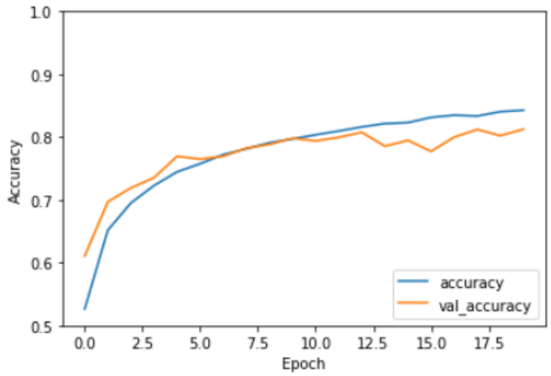
Model 1:

My number one performing model was a multi-layer CNN that consisted of different activation's, dropouts, some max pooling, and batch normalization. While batch normalization was not something listed on the project proposal, after some research and many adjustments I found that adding some normalization into the layers helped the accuracy much more than only having max pooling but did not overexert the amount of time required to train, therefor I wanted to keep it included in the model. This specific model trained much faster and had a better performance without too much overfitting or underfitting error. I trained it at 20 epochs on an Adam optimizer that took about 2 hours and resulted in 81% accuracy on the Cifar10 test set. I assume that If I could have used a GPU to train it on more epochs that accuracy would improve significantly.

```
313/313 - 8s - loss: 0.5565 - accuracy: 0.8124
[0.5564824342727661, 0.8123999834060669]
```

```
CPU times: user 1h 51min 31s, sys: 6min 20s, total: 1h 57min 52s
Wall time: 1h 8min 9s
```

```
313/313 - 8s - loss: 0.5565 - accuracy: 0.8124
```




```python
model8 = models.Sequential()

model8.add(layers.Conv2D(32, (3, 3), activation='elu', input_shape=(32, 32, 3)))
model8.add(layers.BatchNormalization())

model8.add(layers.Conv2D(32, (2, 2)))
model8.add(layers.Activation('relu'))
model8.add(layers.BatchNormalization())
model8.add(layers.MaxPooling2D((2, 2)))
model8.add(layers.Dropout(0.15))

model8.add(layers.Conv2D(64, (3, 3)))
model8.add(layers.Activation('elu'))
model8.add(layers.BatchNormalization())


model8.add(layers.Conv2D(64, (2, 2)))
model8.add(layers.Activation('relu'))
model8.add(layers.BatchNormalization())
model8.add(layers.MaxPooling2D((2, 2)))
model8.add(layers.Dropout(0.20))

model8.add(layers.Conv2D(128, (3, 3)))
model8.add(layers.Activation('relu'))
model8.add(layers.Dropout(0.20))

model8.add(layers.Conv2D(128, (3, 3)))
model8.add(layers.Activation('relu'))
model8.add(layers.BatchNormalization())
#model8.add(layers.MaxPooling2D((2, 2)))
model8.add(layers.Dropout(0.30))

model8.add(layers.Flatten())
model8.add(layers.Dense(10, activation='softmax'))
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
batch_normalization_2 (Batch (None, 30, 30, 32)        128
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 29, 29, 32)        4128
_________________________________________________________________
activation_13 (Activation)   (None, 29, 29, 32)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 29, 29, 32)        128
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_10 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 12, 12, 64)        18496
_________________________________________________________________
activation_14 (Activation)   (None, 12, 12, 64)        0
_________________________________________________________________
batch_normalization_4 (Batch (None, 12, 12, 64)        256
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 11, 11, 64)        16448
_________________________________________________________________
activation_15 (Activation)   (None, 11, 11, 64)        0
_________________________________________________________________
batch_normalization_5 (Batch (None, 11, 11, 64)        256
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
dropout_11 (Dropout)         (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 3, 3, 128)         73856
_________________________________________________________________
activation_16 (Activation)   (None, 3, 3, 128)         0
_________________________________________________________________
dropout_12 (Dropout)         (None, 3, 3, 128)         0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 1, 1, 128)         147584
_________________________________________________________________
activation_17 (Activation)   (None, 1, 1, 128)         0
_________________________________________________________________
batch_normalization_6 (Batch (None, 1, 1, 128)         512
_________________________________________________________________
dropout_13 (Dropout)         (None, 1, 1, 128)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 263,978
Trainable params: 263,338
Non-trainable params: 640
_________________________________________________________________
```

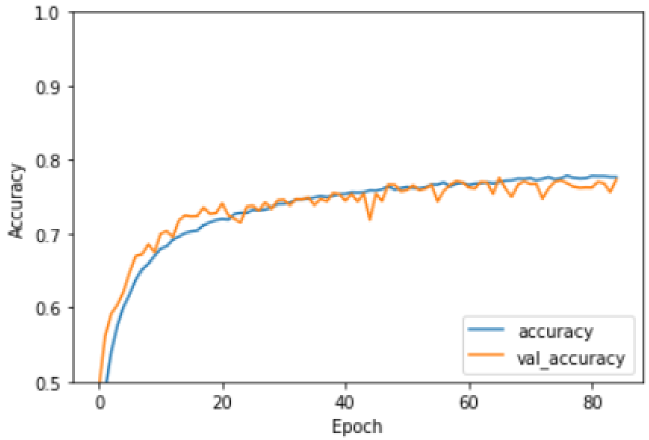
Model 2:

My second top performing model achieved a decent accuracy but at the expense of training time. This original model was run at 100 epochs for a 78% accuracy and took nearly 8 hours. The refined model below was reduced to 85 epochs resulting in 77% accuracy and only 5 hours and 40 mins. That is still a long time to train, especially compared the model number 1. For this one I found that using sigmoid activation functions and increasing the dense layers caused it to perform worse. Again, I stuck to the relatively smaller dropouts early on and increased it throughout resulting in a somewhat close accuracy error between both data sets as explained by the chart below. This model is the one I tried to speed up by applying a GPU to but kept running out of RAM. The next section will provide a comparison of speed.

```
CPU times: user 5h 25min 31s, sys: 15min 20s, total: 5h 40min 51s
Wall time: 3h 11min 58s
```

```
313/313 - 5s - loss: 0.6808 - accuracy: 0.7733
[0.6807695031166077, 0.7732999920845032]
```

313/313 - 5s - loss: 0.6808 - accuracy: 0.7733




```python
model6 = models.Sequential()
model6.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model6.add(layers.MaxPooling2D((2, 2)))
model6.add(layers.Dropout(0.10))
model6.add(layers.Conv2D(64, (2, 2)))
model6.add(layers.Activation('relu'))
model6.add(layers.Dropout(0.10))
model6.add(layers.Conv2D(32, (3, 3)))
model6.add(layers.Activation('relu'))
model6.add(layers.Dropout(0.15))
model6.add(layers.Conv2D(32, (3, 3)))
model6.add(layers.Activation('relu'))
model6.add(layers.MaxPooling2D((2, 2)))
model6.add(layers.Dropout(0.20))
model6.add(layers.Flatten())
model6.add(layers.Dense(64))
model6.add(layers.Activation('relu'))
model6.add(layers.Dropout(0.20))
model6.add(layers.Dense(64))
model6.add(layers.Activation('relu'))
model6.add(layers.Dropout(0.25))
model6.add(layers.Dense(10, activation='softmax'))
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 64)        1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        16448
_________________________________________________________________
activation (Activation)      (None, 14, 14, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 32)        18464
_________________________________________________________________
activation_1 (Activation)    (None, 12, 12, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 10, 10, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 32)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 5, 5, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 800)               0
_________________________________________________________________
dense (Dense)                (None, 64)                51264
_________________________________________________________________
activation_3 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
activation_4 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 102,026
Trainable params: 102,026
Non-trainable params: 0
```

Model 2 GPU [Bonus 1]:

Below is a screenshot comparison of Model #2 being run on a GPU compared to my CPU. The GPU was able to run each epoch in just under 10 seconds while my CPU ran each epoch in about 2 minutes and 10 -15 seconds. Multiply that by 100, 200, or 300 epochs; that is a significant difference in training time. The only issue I ran into was RAM space allowed on the Google Colab platform. They allow for about 12GB of RAM. Running the model for longer than 15 epochs result in an overuse of the allowed RAM and timed me out. I am not entirely sure why this is the case on the platform and not my CPU as I only have 8GB of RAM.

```
Epoch 1/15
1563/1563 [==============================] - 9s 5ms/step - loss: 1.9186 - accuracy: 0.2819 - val_loss: 1.3596 - val_accuracy: 0.5056
Epoch 2/15
1563/1563 [==============================] - 8s 5ms/step - loss: 1.3909 - accuracy: 0.4961 - val_loss: 1.2673 - val_accuracy: 0.5367
Epoch 3/15
1563/1563 [==============================] - 9s 6ms/step - loss: 1.2265 - accuracy: 0.5592 - val_loss: 1.1338 - val_accuracy: 0.6001
Epoch 4/15
1563/1563 [==============================] - 9s 6ms/step - loss: 1.1482 - accuracy: 0.5900 - val_loss: 1.0602 - val_accuracy: 0.6229
Epoch 5/15
1563/1563 [==============================] - 8s 5ms/step - loss: 1.0752 - accuracy: 0.6192 - val_loss: 1.0007 - val_accuracy: 0.6513
Epoch 6/15
1563/1563 [==============================] - 9s 6ms/step - loss: 1.0301 - accuracy: 0.6334 - val_loss: 1.0032 - val_accuracy: 0.6463
Epoch 7/15
1563/1563 [==============================] - 9s 6ms/step - loss: 0.9854 - accuracy: 0.6518 - val_loss: 0.9140 - val_accuracy: 0.6739
Epoch 8/15
1563/1563 [==============================] - 8s 5ms/step - loss: 0.9653 - accuracy: 0.6571 - val_loss: 0.9111 - val_accuracy: 0.6848
Epoch 9/15
1563/1563 [==============================] - 8s 5ms/step - loss: 0.9331 - accuracy: 0.6723 - val_loss: 0.8829 - val_accuracy: 0.6918
Epoch 10/15
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8936 - accuracy: 0.6839 - val_loss: 0.9218 - val_accuracy: 0.6783
Epoch 11/15
1563/1563 [==============================] - 9s 6ms/step - loss: 0.8899 - accuracy: 0.6812 - val_loss: 0.8911 - val_accuracy: 0.6853
Epoch 12/15
1563/1563 [==============================] - 9s 6ms/step - loss: 0.8711 - accuracy: 0.6915 - val_loss: 0.8410 - val_accuracy: 0.7057
Epoch 13/15
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8525 - accuracy: 0.6982 - val_loss: 0.8320 - val_accuracy: 0.7119
Epoch 14/15
1563/1563 [==============================] - 9s 6ms/step - loss: 0.8373 - accuracy: 0.7037 - val_loss: 0.8334 - val_accuracy: 0.7128
Epoch 15/15
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8139 - accuracy: 0.7135 - val_loss: 0.8117 - val_accuracy: 0.7178
CPU times: user 2min 16s, sys: 21.4 s, total: 2min 38s
Wall time: 2min 9s
```

Opposed to:

```
Epoch 1/85
1563/1563 [==============================] - 132s 84ms/step - loss: 1.9779 - accuracy: 0.2506 - val_loss: 1.4058 - val_accuracy: 0.4918
Epoch 2/85
1563/1563 [==============================] - 136s 87ms/step - loss: 1.4561 - accuracy: 0.4712 - val_loss: 1.2367 - val_accuracy: 0.5614
Epoch 3/85
1563/1563 [==============================] - 134s 86ms/step - loss: 1.3137 - accuracy: 0.5322 - val_loss: 1.1406 - val_accuracy: 0.5914
Epoch 4/85
1563/1563 [==============================] - 133s 85ms/step - loss: 1.2101 - accuracy: 0.5723 - val_loss: 1.1227 - val_accuracy: 0.6033
Epoch 5/85
1563/1563 [==============================] - 134s 86ms/step - loss: 1.1389 - accuracy: 0.5975 - val_loss: 1.0687 - val_accuracy: 0.6211
Epoch 6/85
1563/1563 [==============================] - 134s 86ms/step - loss: 1.0868 - accuracy: 0.6150 - val_loss: 1.0034 - val_accuracy: 0.6477
Epoch 7/85
1563/1563 [==============================] - 133s 85ms/step - loss: 1.0521 - accuracy: 0.6348 - val_loss: 0.9456 - val_accuracy: 0.6694
Epoch 8/85
1563/1563 [==============================] - 136s 87ms/step - loss: 0.9988 - accuracy: 0.6524 - val_loss: 0.9251 - val_accuracy: 0.6721
Epoch 9/85
1563/1563 [==============================] - 135s 86ms/step - loss: 0.9805 - accuracy: 0.6585 - val_loss: 0.8992 - val_accuracy: 0.6856
Epoch 10/85
1563/1563 [==============================] - 136s 87ms/step - loss: 0.9411 - accuracy: 0.6712 - val_loss: 0.9372 - val_accuracy: 0.6747
Epoch 11/85
1563/1563 [==============================] - 136s 87ms/step - loss: 0.9239 - accuracy: 0.6794 - val_loss: 0.8677 - val_accuracy: 0.6999
Epoch 12/85
1563/1563 [==============================] - 136s 87ms/step - loss: 0.9121 - accuracy: 0.6836 - val_loss: 0.8472 - val_accuracy: 0.7036
Epoch 13/85
1563/1563 [==============================] - 135s 87ms/step - loss: 0.8779 - accuracy: 0.6964 - val_loss: 0.8768 - val_accuracy: 0.6954
Epoch 14/85
1563/1563 [==============================] - 135s 87ms/step - loss: 0.8811 - accuracy: 0.6960 - val_loss: 0.8096 - val_accuracy: 0.7183
Epoch 15/85
1563/1563 [==============================] - 135s 87ms/step - loss: 0.8575 - accuracy: 0.7030 - val_loss: 0.7962 - val_accuracy: 0.7250
```

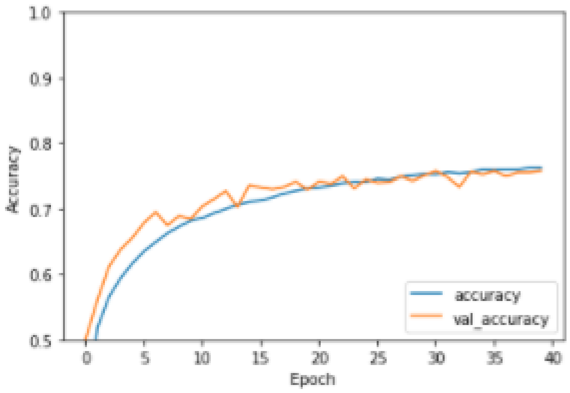
Model 3:

Model 3 is very similar to model 2. Although I have ordered the model performance by accuracy, I think model 3 performs slightly better than model 2 though. The accuracy is very close to model 2 in nearly half of the training time of model 3. I slightly adjusted the number of dense layers toward the end of the network along with some of the activation functions. I kept the max pooling layers, dropouts, and optimizer the same though as they seem to work just fine and not alter the performance of this model too much if adjusted. I think this model could get close to model 2 in performance if trained longer but based on the chart below it looks like it would take more than 100 epochs as it starts to plateau the longer it is trained.

```
313/313 - 6s - loss: 0.7140 - accuracy: 0.7572
[0.7139893770217896, 0.7572000026702881]
```

```
CPU times: user 2h 50min 17s, sys: 5min 59s, total: 2h 56min 17s
Wall time: 1h 37min 55s
```

```
313/313 - 6s - loss: 0.7140 - accuracy: 0.7572
```




```python
model5 = models.Sequential()
model5.add(layers.Conv2D(64, (3, 3), activation='elu', input_shape=(32, 32, 3)))
model5.add(layers.MaxPooling2D((2, 2)))
model5.add(layers.Dropout(0.10))
model5.add(layers.Conv2D(64, (2, 2)))
model5.add(layers.Activation('relu'))
model5.add(layers.Dropout(0.10))
model5.add(layers.Conv2D(32, (3, 3)))
model5.add(layers.Activation('relu'))
model5.add(layers.Dropout(0.15))
model5.add(layers.Conv2D(32, (3, 3)))
model5.add(layers.Activation('relu'))
model5.add(layers.MaxPooling2D((2, 2)))
model5.add(layers.Dropout(0.20))
model5.add(layers.Flatten())
model5.add(layers.Dense(64))
model5.add(layers.Activation('relu'))
model5.add(layers.Dropout(0.20))
model5.add(layers.Dense(128))
model5.add(layers.Activation('elu'))
model5.add(layers.Dropout(0.20))
model5.add(layers.Dense(10, activation='softmax'))
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_10 (Conv2D)           (None, 30, 30, 64)        1792      
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 15, 15, 64)        0         
_________________________________________________________________
dropout_10 (Dropout)         (None, 15, 15, 64)        0         
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 14, 14, 64)        16448     
_________________________________________________________________
activation_10 (Activation)   (None, 14, 14, 64)        0         
_________________________________________________________________
dropout_11 (Dropout)         (None, 14, 14, 64)        0         
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 12, 12, 32)        18464     
_________________________________________________________________
activation_11 (Activation)   (None, 12, 12, 32)        0         
_________________________________________________________________
dropout_12 (Dropout)         (None, 12, 12, 32)        0         
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 10, 10, 32)        9248      
_________________________________________________________________
activation_12 (Activation)   (None, 10, 10, 32)        0         
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 5, 5, 32)          0         
_________________________________________________________________
dropout_13 (Dropout)         (None, 5, 5, 32)          0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 800)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 64)                51264     
_________________________________________________________________
activation_13 (Activation)   (None, 64)                0         
_________________________________________________________________
dropout_14 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_5 (Dense)              (None, 128)               8320      
_________________________________________________________________
activation_14 (Activation)   (None, 128)               0         
_________________________________________________________________
dropout_15 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290      
=================================================================
Total params: 106,826
Trainable params: 106,826
Non-trainable params: 0
_________________________________________________________________
```