Prentiss Beam
EE-104

Lab# 8 Documentation

Import the needed libraries used for Convolutional Neural Networks. The imports are shown below.

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, optimizers
from tensorflow.keras.layers import RandomFlip, RandomRotation, RandomZoom, Rescaling
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
import keras
from tensorflow.keras import preprocessing
from keras.callbacks import LearningRateScheduler
import matplotlib.pyplot as plt
```

Download the CIFAR-10 dataset

```
[3]  (train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

     # Normalize pixel values to be between 0 and 1
     train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 11s 0us/step
170508288/170498071 [==============================] - 11s 0us/step
```

Create Baseline model

This model includes convolutional and maxpooling layers. It also uses Dropout

```
model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',))
model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.4))


model.add(layers.Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',))
model.add(layers.Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.4))
```

Train and Test

In this part the model is trained using the training outputs and inputs and then calculates the classification accuracy and loss on the test data.

```python
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.002),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=15,
                    validation_data=(test_images, test_labels))
```

Increasing Dropout

To original model given resulted in the train and test accuracy shown below. As you can observe in the plot both the training accuracy and test accuracy increase dramatically at the beginning and then flattens out as the epochs increase.

```python
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 1s - loss: 0.8583 - accuracy: 0.7132 - 1s/epoch - 3ms/step
```

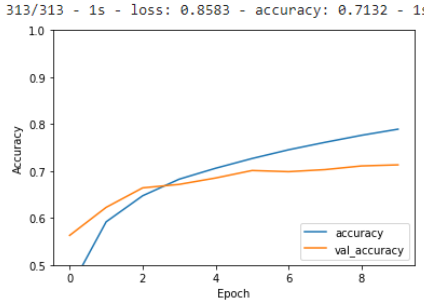


Figure 1. Test and train accuracy of the original NN model

```python
print(test_acc)
```

```
0.7131999731063843
```

Figure 2. Original NN test accuracy

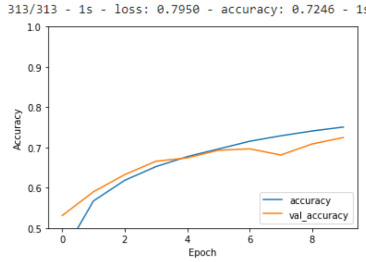
After testing the original Neural Network, I added one dropout and changed the parameter to see how the model would react. I found that increasing it over 0.3 reduces the overall accuracy, however 0.2 seemed to be the sweet spot as it increased as shown below.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Dropout(0.2))
```

Figure 3. Start to increase Dropout layer

The result from this model is shown below

```
313/313 - 1s - loss: 0.7950 - accuracy: 0.7246 - 1s/epoch - 4ms/step
```




```
print(test_acc)
```

```
0.7246000170707703
```

Figure 4. Results from model shown above

Adding a few parameters to the convolutional layer and the location of the dropout layer helped increase the accuracy.

```
[66] model = models.Sequential()
     model.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
     model.add(layers.MaxPooling2D((2, 2)))
     model.add(layers.Dropout(0.2))
     model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',))
     model.add(layers.MaxPooling2D((2, 2)))
     model.add(layers.Dropout(0.2))
     model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',))
```

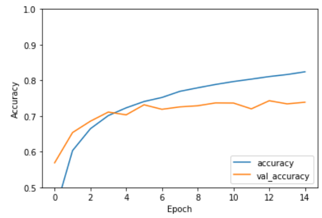
Figure 5. New NN model

The results are shown below.

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 1s - loss: 0.8315 - accuracy: 0.7387 - 1s/epoch - 4ms/step
```




```
print(test_acc)
```

```
0.7386999726295471
```

Figure 6. Results from new NN model

Final model >80% accuracy

For this model I included additional convolutional layers and more dropout layers. I increased the dropout parameter at every layer because it seemed the most effective.

The final model is shown below

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))
model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',))
model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))
model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',))
model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.4))
```

```
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu',kernel_initializer='he_uniform'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(10, activation='softmax'))
```

Figure 7. Final NN model with dropout

The summary final model is shown below

```
] model.summary()

Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_24 (Conv2D)          (None, 32, 32, 32)        896

 conv2d_25 (Conv2D)          (None, 32, 32, 32)        9248

 max_pooling2d_12 (MaxPoolin  (None, 16, 16, 32)       0
 g2D)

 dropout_16 (Dropout)        (None, 16, 16, 32)        0

 conv2d_26 (Conv2D)          (None, 16, 16, 64)        18496

 conv2d_27 (Conv2D)          (None, 16, 16, 64)        36928

 max_pooling2d_13 (MaxPoolin  (None, 8, 8, 64)         0
 g2D)

 dropout_17 (Dropout)        (None, 8, 8, 64)          0

 conv2d_28 (Conv2D)          (None, 8, 8, 128)         73856

 conv2d_29 (Conv2D)          (None, 8, 8, 128)         147584

 max_pooling2d_14 (MaxPoolin  (None, 4, 4, 128)        0
 g2D)

 dropout_18 (Dropout)        (None, 4, 4, 128)         0

 flatten_4 (Flatten)         (None, 2048)              0

 dense_8 (Dense)             (None, 128)               262272

 dropout_19 (Dropout)        (None, 128)               0

 dense_9 (Dense)             (None, 10)                1290

=================================================================
Total params: 550,570
Trainable params: 550,570
Non-trainable params: 0
_________________________________________________________________
```

Figure 8. Summary of model with dropout

A figure showing the compile and fit function, along with the last epochs to load are shown below.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=100, batch_size=128,
                    validation_data=(test_images, test_labels))
```

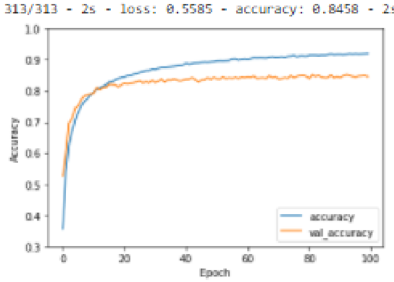
```
391/391 [==============================] - 12s 31ms/step - loss: 0.2610 - accuracy: 0.9079 - val_loss: 0.5290 - val_accuracy: 0.8380
Epoch 72/100
391/391 [==============================] - 12s 32ms/step - loss: 0.2568 - accuracy: 0.9096 - val_loss: 0.5389 - val_accuracy: 0.8389
Epoch 73/100
391/391 [==============================] - 12s 32ms/step - loss: 0.2584 - accuracy: 0.9077 - val_loss: 0.5461 - val_accuracy: 0.8424
Epoch 74/100
391/391 [==============================] - 12s 31ms/step - loss: 0.2542 - accuracy: 0.9109 - val_loss: 0.5453 - val_accuracy: 0.8400
Epoch 75/100
391/391 [==============================] - 13s 33ms/step - loss: 0.2527 - accuracy: 0.9113 - val_loss: 0.5171 - val_accuracy: 0.8494
Epoch 76/100
391/391 [==============================] - 12s 31ms/step - loss: 0.2479 - accuracy: 0.9137 - val_loss: 0.5612 - val_accuracy: 0.8423
Epoch 77/100
391/391 [==============================] - 12s 31ms/step - loss: 0.2579 - accuracy: 0.9092 - val_loss: 0.5119 - val_accuracy: 0.8467
Epoch 78/100
391/391 [==============================] - 12s 31ms/step - loss: 0.2482 - accuracy: 0.9123 - val_loss: 0.5217 - val_accuracy: 0.8492
Epoch 79/100
391/391 [==============================] - 12s 32ms/step - loss: 0.2502 - accuracy: 0.9121 - val_loss: 0.5186 - val_accuracy: 0.8506
Epoch 80/100
391/391 [==============================] - 12s 32ms/step - loss: 0.2555 - accuracy: 0.9108 - val_loss: 0.5354 - val_accuracy: 0.8470
Epoch 81/100
391/391 [==============================] - 12s 31ms/step - loss: 0.2441 - accuracy: 0.9146 - val_loss: 0.5413 - val_accuracy: 0.8396
Epoch 82/100
391/391 [==============================] - 12s 31ms/step - loss: 0.2447 - accuracy: 0.9149 - val_loss: 0.5400 - val_accuracy: 0.8526
Epoch 83/100
391/391 [==============================] - 12s 32ms/step - loss: 0.2462 - accuracy: 0.9147 - val_loss: 0.5406 - val_accuracy: 0.8463
Epoch 84/100
391/391 [==============================] - 12s 31ms/step - loss: 0.2468 - accuracy: 0.9134 - val_loss: 0.5375 - val_accuracy: 0.8499
Epoch 85/100
391/391 [==============================] - 12s 31ms/step - loss: 0.2454 - accuracy: 0.9150 - val_loss: 0.5453 - val_accuracy: 0.8438
Epoch 86/100
391/391 [==============================] - 13s 33ms/step - loss: 0.2436 - accuracy: 0.9148 - val_loss: 0.5488 - val_accuracy: 0.8419
Epoch 87/100
391/391 [==============================] - 12s 31ms/step - loss: 0.2420 - accuracy: 0.9161 - val_loss: 0.5598 - val_accuracy: 0.8422
Epoch 88/100
391/391 [==============================] - 12s 32ms/step - loss: 0.2341 - accuracy: 0.9189 - val_loss: 0.5352 - val_accuracy: 0.8513
Epoch 89/100
391/391 [==============================] - 12s 31ms/step - loss: 0.2406 - accuracy: 0.9161 - val_loss: 0.5667 - val_accuracy: 0.8418
Epoch 90/100
391/391 [==============================] - 12s 31ms/step - loss: 0.2371 - accuracy: 0.9172 - val_loss: 0.5593 - val_accuracy: 0.8459
Epoch 91/100
391/391 [==============================] - 12s 32ms/step - loss: 0.2354 - accuracy: 0.9174 - val_loss: 0.5588 - val_accuracy: 0.8428
Epoch 92/100
391/391 [==============================] - 12s 31ms/step - loss: 0.2439 - accuracy: 0.9161 - val_loss: 0.5489 - val_accuracy: 0.8482
Epoch 93/100
391/391 [==============================] - 12s 32ms/step - loss: 0.2366 - accuracy: 0.9175 - val_loss: 0.5335 - val_accuracy: 0.8527
Epoch 94/100
391/391 [==============================] - 12s 31ms/step - loss: 0.2329 - accuracy: 0.9191 - val_loss: 0.5351 - val_accuracy: 0.8502
Epoch 95/100
391/391 [==============================] - 12s 32ms/step - loss: 0.2328 - accuracy: 0.9181 - val_loss: 0.5572 - val_accuracy: 0.8463
Epoch 96/100
391/391 [==============================] - 12s 32ms/step - loss: 0.2316 - accuracy: 0.9189 - val_loss: 0.5669 - val_accuracy: 0.8444
Epoch 97/100
391/391 [==============================] - 12s 32ms/step - loss: 0.2395 - accuracy: 0.9182 - val_loss: 0.5284 - val_accuracy: 0.8474
Epoch 98/100
391/391 [==============================] - 12s 32ms/step - loss: 0.2347 - accuracy: 0.9186 - val_loss: 0.5601 - val_accuracy: 0.8483
Epoch 99/100
391/391 [==============================] - 12s 32ms/step - loss: 0.2333 - accuracy: 0.9190 - val_loss: 0.5499 - val_accuracy: 0.8498
Epoch 100/100
391/391 [==============================] - 12s 32ms/step - loss: 0.2328 - accuracy: 0.9196 - val_loss: 0.5585 - val_accuracy: 0.8458
```

Figure 9. Fit and epoch report

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.3, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 2s - loss: 0.5585 - accuracy: 0.8458 - 2s/epoch - 6ms/step
```




```
print(test_acc)
```

```
0.84579998254776
```

Adding Data Augmentation

We added data augmentation by apply three keras layers which flip, rotate, zoom or crop the input data to create more and unique data. This is also a method to reduce

```
model = models.Sequential([layers.RandomFlip("horizontal", input_shape=(32,32,3)),layers.RandomRotation(0.1),layers.RandomZoom(0.1)])
```

```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator( rotation_range=90,
                 width_shift_range=0.1, height_shift_range=0.1,
                 horizontal_flip=True)
datagen.fit(train_images)
```

Adding Batch Normalization

To add Batch normalization, we implement the command shown below within the neural network model.

```
model.add(layers.BatchNormalization())
```

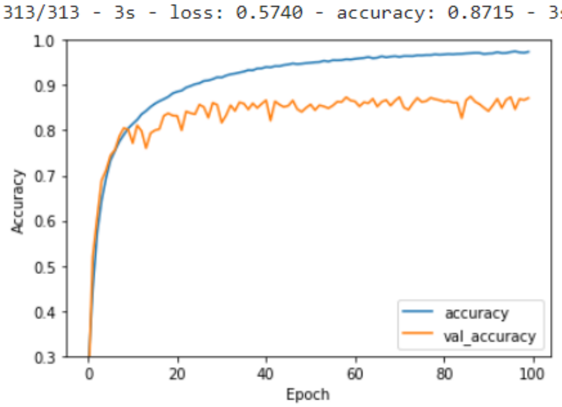
Figure 10. Batch Normalization

The methods shown above are ways to increase the classification accuracy and decrease the loss function. By using Dropout, Data Augmentation and Batch normalization we can achieve an accuracy close to 88%. The result from the final model is shown below.

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.3, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 3s - loss: 0.5740 - accuracy: 0.8715 - 3s/epoch - 9ms/step
```




```
[15] print(test_acc)
```

```
0.8715000152587891
```