Langara College

CPSC- 4830

Instructor: Chinmaya Mahapatra

Student: Roberta Bukowski

ID: 100342032

Assignment 4

Comparison between Jupyter notebook CIFAR 10- Classification and Google Colab

The goal of this report is to implement a convnet to classify CIFAR10 images using Jupyter Notebook(CPU) and Google Colab (GPU) is going to be compared the its training time on local workstation against the time in Colab GPU. In both cases it will be used 50 Epochs.

Useful information

CIFAR 10 Dataset

Data Source: https://www.cs.toronto.edu/~kriz/cifar.html

The CIFAR(Canadian Institute For Advanced Research)-10 consists of several images divided into the following 10 classes:

- Airplanes
- Cars
- Birds
- Cats
- Deer
- Dogs
- Frogs
- Horses
- Ships
- Trucks

Equipment
CPU Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz, 1800 Mhz, 4 Core(s), 8 Logical Processor(s) (Acer-Aspire 515-52g)

* My notebook has a GPU NVidia GeForce 150MX, 2GB , in this particular report will be tested the CPU against google colab GPU's.

CIFAR 10 on Jupyter Notebook

Import the necessary Libraries

```
In [1]: #Libraries

        import tensorflow
        from tensorflow.keras.datasets import mnist
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten
        from tensorflow.keras.optimizers import RMSprop
        from keras.datasets import cifar10
        from keras.applications import resnet50
        from keras.preprocessing import image
        import matplotlib.pyplot as plt
        import pandas as pd
        import seaborn as sns
        import numpy as np
        sns.set()
        import autotime #! pip install ipython-autotime
        %load_ext autotime
        %matplotlib inline

        Using TensorFlow backend.

        time: 0 ns
```

Note that from keras.datasets we imported the cifar10 data. By using autotime all the outputs will show the time for execution for every cell. This code took 0ns to be executed.

```
In [5]: # Setting labels
        labels=['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck']

        time: 0 ns
```

It took zero seconds to set the labels

```
In [6]: #Loading the data
        (X_train,y_train),(X_test,y_test)=cifar10.load_data()

        time: 703 ms
```

The data loading took 703 milliseconds.

```
In [7]: #Shape of data
        print('X_train shape: {}, y_train_labels.shape: {}'
              .format(X_train.shape, y_train.shape))
        print('X_test shape: {}, y_test_labels.shape: {}'.format(X_test.shape, y_test.shape))

        X_train shape: (50000, 32, 32, 3), y_train_labels.shape: (50000, 1)
        X_test shape: (10000, 32, 32, 3), y_test_labels.shape: (10000, 1)
        time: 0 ns
```

To print the shape of the data took 0 ns.

```
In [8]: #Checking the images
        fig, axes = plt.subplots(ncols=7, nrows=3, figsize=(17, 8))
        index = 0
        for i in range(3):
            for j in range(7):
                axes[i,j].set_title(labels[y_train[index][0]])
                axes[i,j].imshow(X_train[index])
                axes[i,j].get_xaxis().set_visible(False)
                axes[i,j].get_yaxis().set_visible(False)
                index += 1
        plt.show()
```


```
time: 844 ms
```

To plot the images for checking took 844 ms

```
In [9]: #Label pre processing
        y_train = tensorflow.keras.utils.to_categorical(y_train, 10)
        y_test= tensorflow.keras.utils.to_categorical(y_test, 10)
        y_test.shape

Out[9]: (10000, 10)

        time: 16 ms
```

The label pre processing took 16 ms

```
In [10]: #Reshape
         from tensorflow.keras import backend as K

         if K.image_data_format() == 'channels_first':
             X_train = X_train.reshape(X_train.shape[0], 3, 32, 32)
             X_test = X_test.reshape(X_test.shape[0], 3, 32, 32)
             input_shape = (3, 32, 32)
         else:
             X_train = X_train.reshape(X_train.shape[0], 32, 32, 3)
             X_test = X_test.reshape(X_test.shape[0], 32,32 ,3)
             input_shape = (32,32, 3)

         X_train = X_train.astype('float32')
         X_test = X_test.astype('float32')
         X_train /= 255
         X_test /= 255

         time: 375 ms
```

To reshape the data , transform into float and divide by its RGB values took 375milliseconds still less than a second(1000 milliseconds = 1 second).

```
In [11]: #Building a model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
# 64 3x3 kernels
model.add(Conv2D(64, (3, 3), activation='relu'))
# Reduce by taking the max of each 2x2 block
model.add(MaxPooling2D(pool_size=(2, 2)))
# Flatten the results to one dimension for passing into our final layer
model.add(Flatten())
# A hidden layer to learn with
model.add(Dense(512, activation='relu'))
# Another dropout
model.add(Dropout(0.25))
# Final categorization from 0-9 with softmax
model.add(Dense(10, activation='softmax'))

time: 281 ms
```

To build the model, add a conv2D, MaxPooling2, flatten the results, add a hidden layer, add a Dropout and categorize the model softmax. Took 281ms.

```
In [12]: model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 12544)             0
_________________________________________________________________
dense (Dense)                (None, 512)               6423040
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 6,447,562
Trainable params: 6,447,562
Non-trainable params: 0
_________________________________________________________________
time: 0 ns
```

The model summary took 0ns.

```
In [13]: model.compile(loss='categorical_crossentropy',
              optimizer='RMSProp',
              metrics=['accuracy'])

time: 47 ms
```

The model compile took 47 ms

```
In [14]: history = model.fit(X_train, y_train,
                             batch_size=32,
                             epochs=50,
                             verbose=1,
                             validation_data=(X_test, y_test))
```

```
50000/50000 [==============================] - 164s 3ms/sample - loss: 1.2761 - accuracy: 0.6046 - val_loss: 2.4579 - val_ac
curacy: 0.4759
Epoch 45/50
50000/50000 [==============================] - 163s 3ms/sample - loss: 1.2907 - accuracy: 0.5996 - val_loss: 1.4145 - val_ac
curacy: 0.5293
Epoch 46/50
50000/50000 [==============================] - 166s 3ms/sample - loss: 1.3382 - accuracy: 0.5976 - val_loss: 1.5337 - val_ac
curacy: 0.5503
Epoch 47/50
50000/50000 [==============================] - 164s 3ms/sample - loss: 1.2792 - accuracy: 0.6032 - val_loss: 1.7177 - val_ac
curacy: 0.5691
Epoch 48/50
50000/50000 [==============================] - 165s 3ms/sample - loss: 1.2980 - accuracy: 0.5967 - val_loss: 1.5096 - val_ac
curacy: 0.5704
Epoch 49/50
50000/50000 [==============================] - 167s 3ms/sample - loss: 1.2801 - accuracy: 0.6031 - val_loss: 1.6507 - val_ac
curacy: 0.5083
Epoch 50/50
50000/50000 [==============================] - 166s 3ms/sample - loss: 1.2998 - accuracy: 0.5989 - val_loss: 1.4231 - val_ac
curacy: 0.5675
time: 4h 37min 34s
```

The training took 4h37min 34s. This is curious since when I ran for the first time took 2h and 18min.

```
In [15]: score = model.evaluate(X_test, y_test, verbose=0)
         print('Test loss:', score[0])
         print('Test accuracy:', score[1])
```

```
Test loss: 1.4231208263397217
Test accuracy: 0.5675
time: 5.24 s
```

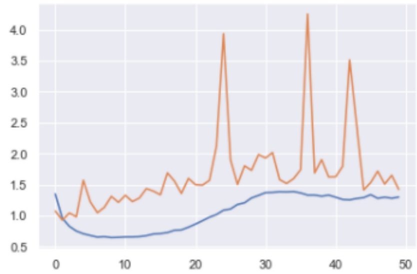
The model evaluation took 5.24s

```
In [16]: plt.plot(history.history['accuracy'])
         plt.plot(history.history['val_accuracy'])
         plt.show()
```


```
time: 312 ms
```

To plot the accuracy took 312ms.

In [17]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.show()
```




```
time: 140 ms
```

To plot the loss took 140 ms.

In [18]:
```
#Load images
truck = image.load_img("truck.jpg", target_size=(32, 32))
auto = image.load_img("automobile.jpg", target_size=(32, 32))
dog = image.load_img("dog.jpg", target_size=(32, 32))
cat = image.load_img("cat.jpg", target_size=(32, 32))
plane = image.load_img("airplane.jpg", target_size=(32, 32))
```

```
time: 125 ms
```

In [19]:
```
truck = image.img_to_array(truck)
auto = image.img_to_array(auto)
dog = image.img_to_array(dog)
cat = image.img_to_array(cat)
plane = image.img_to_array(plane)
```

```
time: 15 ms
```

In [20]:
```
truck= np.expand_dims(truck, axis=0)
auto = np.expand_dims(auto, axis=0)
dog = np.expand_dims(dog, axis=0)
cat = np.expand_dims(cat, axis=0)
plane = np.expand_dims(plane, axis=0)
```

```
time: 0 ns
```

In [21]:
```
predictions_truck = model.predict(truck)
predictions_auto = model.predict(auto)
predictions_dog = model.predict(dog)
predictions_cat = model.predict(cat)
predictions_plane = model.predict(plane)
```

```
time: 359 ms
```

To load images took 125ms, to pass the image to array took 15ms. To transform the images to array 15 ms. To expand the dimensions of the images, took 0ns

In [ ]:
```
#1: Airplane
#2: Car
#3: Bird
#4: Cat
#5: Deer
#6: Dog
#7: Frog
#8: Horse
#9: Ship
#10: Truck
```

In [22]:
```
print("Prediction truck")
predictions_truck
#Predicted: Truck
#Actual" Truck
```

```
Prediction truck
```

Out[22]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]], dtype=float32)
```

```
time: 16 ms
```

To print the array with the predictions of a truck took 16 ms, you can find bellow the time of all predictions.

```
In [23]: print("Prediction Auto")
         predictions_auto
         #Predicted: Car
         #Actual:Car

         Prediction Auto

Out[23]: array([[0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)

         time: 16 ms

In [24]: print("Prediction Dog")
         predictions_dog
         #Predicted:Car
         #Actual:Dog

         Prediction Dog

Out[24]: array([[0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)

         time: 0 ns

In [25]: print("Prediction Cat")
         predictions_cat
         #Predicted: Dog
         #Actual:Cat

         Prediction Cat

Out[25]: array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]], dtype=float32)

         time: 0 ns

In [26]: print("Prediction Airplaine")
         predictions_plane
         #predicted: Plane
         #Actual:Plane

         Prediction Airplaine

Out[26]: array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)

         time: 16 ms
```

Test Accuracy

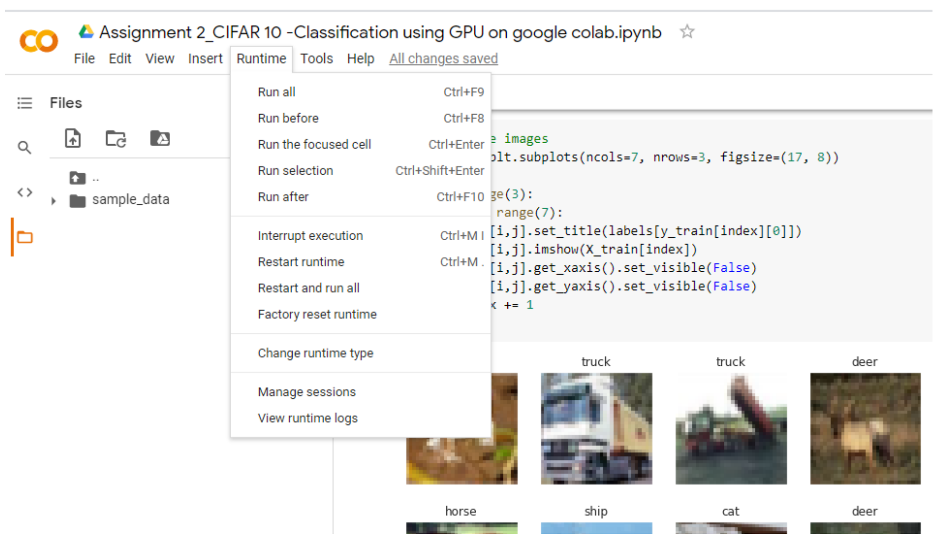
```
In [27]: #Test Accuracy
         from matplotlib import pyplot
         %matplotlib inline
         y_test = y_test.argmax(1)

         def plot_predictions(images, predictions, true_labels):
           n = images.shape[0]
           nc = int(np.ceil(n / 4))
           fig = pyplot.figure(figsize=(4,3))
           # axes = fig.add_subplot(nc, 4)
           f, axes = pyplot.subplots(nc, 4)
           f.tight_layout()
           for i in range(nc * 4):
             y = i // 4
             x = i % 4
             axes[x, y].axis('off')

             label = labels[np.argmax(predictions[i])]
             confidence = np.max(predictions[i])
             if i > n:
               continue
             axes[x, y].imshow(images[i])
             pred_label = np.argmax(predictions[i])
             axes[x, y].set_title("{} ({})\n {:.3f}".format(
               labels[pred_label],
               labels[true_labels[i]],
               confidence),
               color=("green" if true_labels[i] == pred_label else "red"))
           pyplot.gcf().set_size_inches(8, 8)

         plot_predictions(
             np.squeeze(X_test[:16]),
             model.predict(X_test[:16]),
             y_test[:16]
         )
```

```
<Figure size 288x216 with 0 Axes>
```

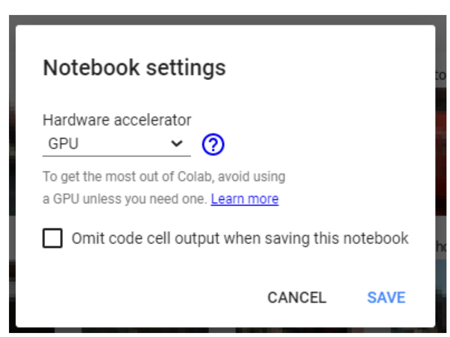

```
time: 1.88 s
```

The test accuracy with the plots code took 1.88s.

Google Colab with GPU



First step on google colab is certify that we are using GPU to do this click on Change runtime Type and select GPU.



```python
import tensorflow
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten
from tensorflow.keras.optimizers import RMSprop
from keras.datasets import cifar10
from keras.applications import resnet50
from keras.preprocessing import image
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import confusion_matrix
sns.set()
!pip install ipython-autotime
import autotime
%load_ext autotime
%matplotlib inline
```

```
Collecting ipython-autotime
  Downloading https://files.pythonhosted.org/packages/3f/58/a4a65efcce5c81a67b6893ade862736de355a3a718af5533d30c991831ce/ipython_autotime-0.2.0-p
Requirement already satisfied: ipython in /usr/local/lib/python3.6/dist-packages (from ipython-autotime) (5.5.0)
Requirement already satisfied: simplegeneric>0.8 in /usr/local/lib/python3.6/dist-packages (from ipython->ipython-autotime) (0.8.1)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.6/dist-packages (from ipython->ipython-autotime) (50.3.2)
Requirement already satisfied: pexpect; sys_platform != "win32" in /usr/local/lib/python3.6/dist-packages (from ipython->ipython-autotime) (4.8.0
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in /usr/local/lib/python3.6/dist-packages (from ipython->ipython-autotime) (1.0.18)
Requirement already satisfied: decorator in /usr/local/lib/python3.6/dist-packages (from ipython->ipython-autotime) (4.4.2)
Requirement already satisfied: pygments in /usr/local/lib/python3.6/dist-packages (from ipython->ipython-autotime) (2.6.1)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.6/dist-packages (from ipython->ipython-autotime) (4.3.3)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.6/dist-packages (from ipython->ipython-autotime) (0.7.5)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.6/dist-packages (from pexpect; sys_platform != "win32"->ipython->ipythor
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.6/dist-packages (from prompt-toolkit<2.0.0,>=1.0.4->ipython->ipython-autotime
Requirement already satisfied: wcwidth in /usr/local/lib/python3.6/dist-packages (from prompt-toolkit<2.0.0,>=1.0.4->ipython->ipython-autotime) (
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.6/dist-packages (from traitlets>=4.2->ipython->ipython-autotime) (0.2.0
Installing collected packages: ipython-autotime
Successfully installed ipython-autotime-0.2.0
time: 1.25 ms
```

To import the libraries it took 1.25ms

```python
[ ] #Loading the data
    (X_train,y_train),(X_test,y_test)=cifar10.load_data()
```

```
time: 672 ms
```

To load the data 672 ms

```python
[ ] #Shape of data
    print('X_train shape: {}, y_train_labels.shape: {}'
          .format(X_train.shape, y_train.shape))
    print('X_test shape: {}, y_test_labels.shape: {}'.format(X_test.shape, y_test.shape))
```

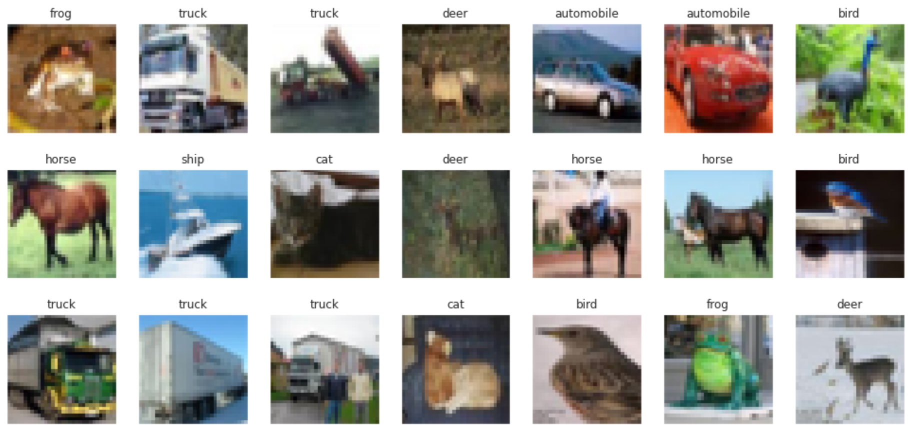
```
X_train shape: (50000, 32, 32, 3), y_train_labels.shape: (50000, 1)
X_test shape: (10000, 32, 32, 3), y_test_labels.shape: (10000, 1)
time: 1.91 ms
```

```python
[ ] labels=["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"]
```

```
time: 866 µs
```

To print the shape of data and make a list with labels 866 ms

```
[ ] #Checking the images
    fig, axes = plt.subplots(ncols=7, nrows=3, figsize=(17, 8))
    index = 0
    for i in range(3):
        for j in range(7):
            axes[i,j].set_title(labels[y_train[index][0]])
            axes[i,j].imshow(X_train[index])
            axes[i,j].get_xaxis().set_visible(False)
            axes[i,j].get_yaxis().set_visible(False)
            index += 1
    plt.show()
```




```
time: 746 ms
```

To plot the images 746 ms

```
[ ] #Label pre processing
    y_train = tensorflow.keras.utils.to_categorical(y_train, 10)
    y_test= tensorflow.keras.utils.to_categorical(y_test, 10)
    y_test.shape

    (10000, 10)time: 6.03 ms
```

The label pre processing took 6.03 ms

```
[ ] #Reshape
    from tensorflow.keras import backend as K

    if K.image_data_format() == 'channels_first':
        X_train = X_train.reshape(X_train.shape[0], 3, 32, 32)
        X_test = X_test.reshape(X_test.shape[0], 3, 32, 32)
        input_shape = (3, 32, 32)
    else:
        X_train = X_train.reshape(X_train.shape[0], 32, 32, 3)
        X_test = X_test.reshape(X_test.shape[0], 32,32 ,3)
        input_shape = (32,32, 3)

    X_train = X_train.astype('float32')
    X_test = X_test.astype('float32')
    X_train /= 255
    X_test /= 255

    time: 253 ms
```

To reshape, convert to float and divide by its RGB took 253ms

```
[ ] #Building a model
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3),
                     activation='relu',
                     input_shape=input_shape))
    # 64 3x3 kernels
    model.add(Conv2D(64, (3, 3), activation='relu'))
    # Reduce by taking the max of each 2x2 block
    model.add(MaxPooling2D(pool_size=(2, 2)))
    # Flatten the results to one dimension for passing into our final layer
    model.add(Flatten())
    # A hidden layer to learn with
    model.add(Dense(512, activation='relu'))
    # Another dropout
    model.add(Dropout(0.25))
    # Final categorization from 0-9 with softmax
    model.add(Dense(10, activation='softmax'))

    time: 822 ms
```

To build the model it took 822ms

```
[ ] model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 12544)             0
_________________________________________________________________
dense (Dense)                (None, 512)               6423040
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 6,447,562
Trainable params: 6,447,562
Non-trainable params: 0
_________________________________________________________________
time: 6.02 ms
```

To get the model summary took 6.02 ms

```
[ ] model.compile(loss='categorical_crossentropy',
                  optimizer='RMSProp',
                  metrics=['accuracy'])

    time: 16.6 ms
```

Model compile 16.6s

```
[ ] history = model.fit(X_train, y_train,
                        batch_size=32,
                        epochs=50,
                        verbose=1,
                        validation_data=(X_test, y_test))
```

```
1563/1563 [==============================] - 15s 10ms/step - loss: 0.8081 - accuracy: 0.7271 - val_loss: 1.0004 - val_accuracy: 0.6859
Epoch 4/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.7151 - accuracy: 0.7642 - val_loss: 1.1628 - val_accuracy: 0.6784
Epoch 5/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.6645 - accuracy: 0.7868 - val_loss: 1.2538 - val_accuracy: 0.6894
Epoch 6/50
1563/1563 [==============================] - 15s 9ms/step - loss: 0.6368 - accuracy: 0.7965 - val_loss: 1.0696 - val_accuracy: 0.6772
Epoch 7/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.6141 - accuracy: 0.8040 - val_loss: 1.2091 - val_accuracy: 0.6584
Epoch 8/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.6100 - accuracy: 0.8114 - val_loss: 1.1047 - val_accuracy: 0.6901
Epoch 9/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.6045 - accuracy: 0.8118 - val_loss: 3.3584 - val_accuracy: 0.5930
Epoch 10/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.5990 - accuracy: 0.8155 - val_loss: 1.6004 - val_accuracy: 0.6882
Epoch 11/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.6192 - accuracy: 0.8137 - val_loss: 2.0335 - val_accuracy: 0.6731
Epoch 12/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.6099 - accuracy: 0.8176 - val_loss: 1.8803 - val_accuracy: 0.6672
Epoch 13/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.6294 - accuracy: 0.8147 - val_loss: 1.6731 - val_accuracy: 0.6410
Epoch 14/50
1563/1563 [==============================] - 16s 10ms/step - loss: 0.6419 - accuracy: 0.8136 - val_loss: 1.9925 - val_accuracy: 0.6280
Epoch 15/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.6796 - accuracy: 0.8052 - val_loss: 2.2809 - val_accuracy: 0.6377
Epoch 16/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.7065 - accuracy: 0.7973 - val_loss: 1.5116 - val_accuracy: 0.6233
Epoch 17/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.7377 - accuracy: 0.7891 - val_loss: 1.7207 - val_accuracy: 0.6677
Epoch 18/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.8059 - accuracy: 0.7749 - val_loss: 1.6863 - val_accuracy: 0.5747
Epoch 19/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.8740 - accuracy: 0.7549 - val_loss: 1.4807 - val_accuracy: 0.5876
```

```
Epoch 20/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.9170 - accuracy: 0.7490 - val_loss: 1.6991 - val_accuracy: 0.5498
Epoch 21/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.9274 - accuracy: 0.7426 - val_loss: 1.8063 - val_accuracy: 0.6062
Epoch 22/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.9458 - accuracy: 0.7367 - val_loss: 1.6780 - val_accuracy: 0.6378
Epoch 23/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.9235 - accuracy: 0.7411 - val_loss: 1.5913 - val_accuracy: 0.6264
Epoch 24/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.9559 - accuracy: 0.7418 - val_loss: 1.7033 - val_accuracy: 0.6054
Epoch 25/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.9431 - accuracy: 0.7396 - val_loss: 1.8220 - val_accuracy: 0.5991
Epoch 26/50
1563/1563 [==============================] - 15s 10ms/step - loss: 0.9763 - accuracy: 0.7356 - val_loss: 2.0960 - val_accuracy: 0.6269
Epoch 27/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0006 - accuracy: 0.7294 - val_loss: 1.6410 - val_accuracy: 0.5691
Epoch 28/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0638 - accuracy: 0.7185 - val_loss: 1.8116 - val_accuracy: 0.5405
Epoch 29/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0550 - accuracy: 0.7174 - val_loss: 2.2497 - val_accuracy: 0.6026
Epoch 30/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0340 - accuracy: 0.7217 - val_loss: 1.8102 - val_accuracy: 0.6367
Epoch 31/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.1114 - accuracy: 0.7142 - val_loss: 2.0057 - val_accuracy: 0.6279
Epoch 32/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0947 - accuracy: 0.7152 - val_loss: 1.7513 - val_accuracy: 0.5168
Epoch 33/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0568 - accuracy: 0.7084 - val_loss: 2.1839 - val_accuracy: 0.6137
Epoch 34/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0771 - accuracy: 0.7008 - val_loss: 1.6077 - val_accuracy: 0.5769
Epoch 35/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0863 - accuracy: 0.6941 - val_loss: 3.4033 - val_accuracy: 0.5967
Epoch 36/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.1229 - accuracy: 0.6899 - val_loss: 2.5007 - val_accuracy: 0.5724
Epoch 37/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.1118 - accuracy: 0.6946 - val_loss: 1.5846 - val_accuracy: 0.5441
Epoch 38/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0743 - accuracy: 0.7028 - val_loss: 1.7678 - val_accuracy: 0.5827
Epoch 39/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0917 - accuracy: 0.6978 - val_loss: 1.6736 - val_accuracy: 0.5567
Epoch 40/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0970 - accuracy: 0.6993 - val_loss: 1.6848 - val_accuracy: 0.5564
Epoch 41/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0806 - accuracy: 0.7085 - val_loss: 1.9713 - val_accuracy: 0.6017
```

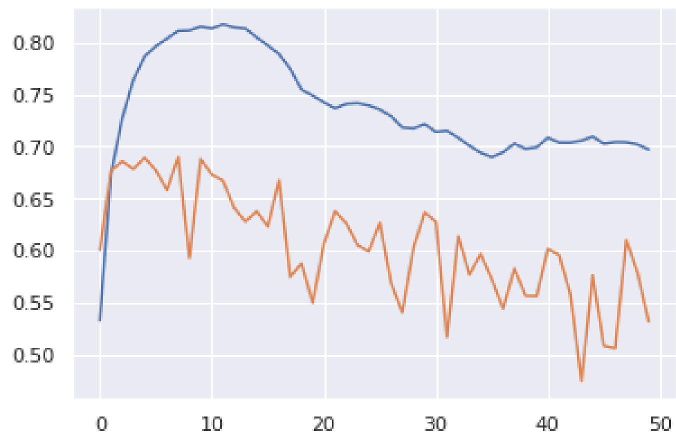
```
Epoch 41/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0806 - accuracy: 0.7085 - val_loss: 1.9713 - val_accuracy: 0.6017
Epoch 42/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0325 - accuracy: 0.7040 - val_loss: 2.0553 - val_accuracy: 0.5957
Epoch 43/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.2926 - accuracy: 0.7040 - val_loss: 1.7613 - val_accuracy: 0.5583
Epoch 44/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0566 - accuracy: 0.7056 - val_loss: 12.5122 - val_accuracy: 0.4744
Epoch 45/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0245 - accuracy: 0.7096 - val_loss: 2.2527 - val_accuracy: 0.5763
Epoch 46/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0449 - accuracy: 0.7028 - val_loss: 2.1493 - val_accuracy: 0.5085
Epoch 47/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0548 - accuracy: 0.7045 - val_loss: 1.9324 - val_accuracy: 0.5058
Epoch 48/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0338 - accuracy: 0.7043 - val_loss: 2.1617 - val_accuracy: 0.6102
Epoch 49/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.1159 - accuracy: 0.7020 - val_loss: 1.8190 - val_accuracy: 0.5788
Epoch 50/50
1563/1563 [==============================] - 15s 10ms/step - loss: 1.0808 - accuracy: 0.6972 - val_loss: 1.8511 - val_accuracy: 0.5316
time: 12min 42s
```

The whole train took 12min 42s in other attempts took only 9min and 18s considerably less then the Jupyter Notebook.

```
[ ] score = model.evaluate(X_test, y_test, verbose=0)
    print('Test loss:', score[0])
    print('Test accuracy:', score[1])
```

```
Test loss: 1.8511338233947754
Test accuracy: 0.5315999984741211
time: 905 ms
```

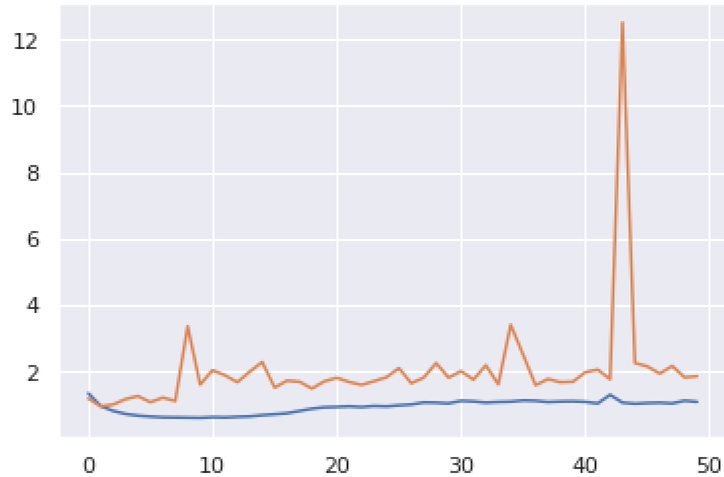
```
[ ] plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.show()
```




```
time: 216 ms
```

It took 905 ms to evaluate the model , and 216ms to plot the accuracy against validation accuracy.

```
[ ] plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.show()
```




```
time: 192 ms
```

192ms to plot the model loss.

```
[ ]
    #Load images
    truck = image.load_img("truck.jpg", target_size=(32, 32))
    auto = image.load_img("automobile.jpg", target_size=(32, 32))
    dog = image.load_img("dog.jpg", target_size=(32, 32))
    cat = image.load_img("cat.jpg", target_size=(32, 32))
    plane = image.load_img("airplane.jpg", target_size=(32, 32))
```

```
time: 101 ms
```

101ms to load the images

```
[ ] truck = image.img_to_array(truck)
    auto = image.img_to_array(auto)
    dog = image.img_to_array(dog)
    cat = image.img_to_array(cat)
    plane = image.img_to_array(plane)
```

```
time: 2.02 ms
```

```
[ ] truck= np.expand_dims(truck, axis=0)
    auto = np.expand_dims(auto, axis=0)
    dog = np.expand_dims(dog, axis=0)
    cat = np.expand_dims(cat, axis=0)
    plane = np.expand_dims(plane, axis=0)
```

```
time: 2.28 ms
```

```
[ ] predictions_truck = model.predict(truck)
    predictions_auto = model.predict(auto)
    predictions_dog = model.predict(dog)
    predictions_cat = model.predict(cat)
    predictions_plane = model.predict(plane)
```

```
time: 212 ms
```

Above the time to array the images, expand its dimensions make the predictions from model.

1: Airplane 2: Car 3: Bird 4: Cat 5: Deer 6: Dog 7: Frog 8: Horse 9: Ships 10: Truck

```
[ ] print("Prediction truck")
    predictions_truck
    #prediction: 1: airplane
    #actual: truck

    Prediction truck
    array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)time: 7.36 ms
```

```
[ ] print("Prediction Auto")
    predictions_auto
    #prediction: 1: Airplane
    #actual: Car

    Prediction Auto
    array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)time: 4.62 ms
```

```
[ ] print("Prediction Dog")
    predictions_dog
    #prediction: 7:frog
    #actual: dog

    Prediction Dog
    array([[0., 0., 0., 0., 0., 0., 1., 0., 0., 0.]], dtype=float32)time: 3.68 ms
```

```
[ ] print("Prediction Cat")
    predictions_cat
    #prediction: 3: bird
    #actual: cat

    Prediction Cat
    array([[0., 0., 1., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)time: 3.56 ms
```

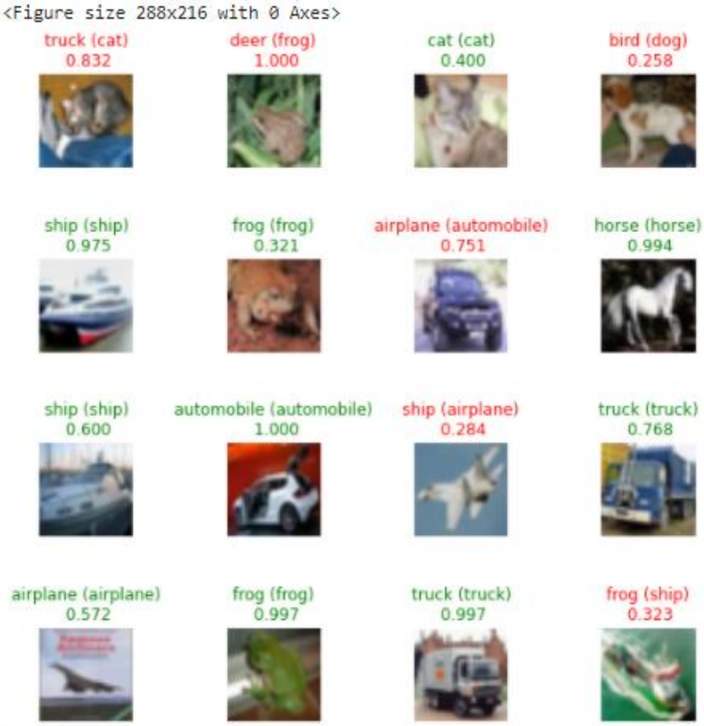
```
[ ] print("Prediction Airplaine")
    predictions_plane
    #prediction: 1: airplane
    #actual: airplane

    Prediction Airplaine
    array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)time: 4.25 ms
```

Predictions time above.

Lastly the model predictions with images

```
[ ] from matplotlib import pyplot
    %matplotlib inline
    y_test = y_test.argmax(1)

    def plot_predictions(images, predictions, true_labels):
      n = images.shape[0]
      nc = int(np.ceil(n / 4))
      fig = pyplot.figure(figsize=(4,3))
      # axes = fig.add_subplot(nc, 4)
      f, axes = pyplot.subplots(nc, 4)
      f.tight_layout()
      for i in range(nc * 4):
        y = i // 4
        x = i % 4
        axes[x, y].axis('off')

        label = labels[np.argmax(predictions[i])]
        confidence = np.max(predictions[i])
        if i > n:
          continue
        axes[x, y].imshow(images[i])
        pred_label = np.argmax(predictions[i])
        axes[x, y].set_title("{} ({})\n {:.3f}".format(
          labels[pred_label],
          labels[true_labels[i]],
          confidence),
          color=("green" if true_labels[i] == pred_label else "red"))
      pyplot.gcf().set_size_inches(8, 8)

    plot_predictions(
        np.squeeze(X_test[:16]),
        model.predict(X_test[:16]),
        y_test[:16]
    )
```

```
<Figure size 288x216 with 0 Axes>
```




```
time: 2.63 s
```

Conclusion

A hardware accelerator like GPU will improve the velocity during Deep Learning models, giving us the possibility to try different parameters faster . Training is time consuming , we can see this simply comparing the performance of a CPU vs the Google Colab GPU. Google Colaboratory is free this is also an advantage but some features are only available in the paid version(still in free version you can get instant access to a GPU or TPU). However, since we are professionals it make sense invest in better machines, but we should mind there's no such thing as a perfect machine to perform some tasks (image classification, text processing etc.) it always good use a server.