Madhav Sharma

EE104

Professor Pham

7 May 2022

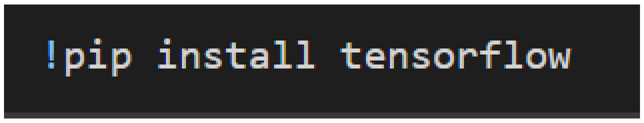
Lab 8 Documentation

For this lab, we used google collab to train the model in order to recognize images. We used CNN template to run the training and modified the code in order to receive better accuracy for the model when we trained it. To begin with we followed the tutorial from the following website to start the lab:

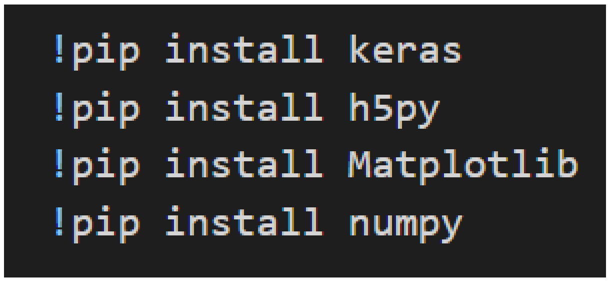
https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/cnn.ipynb#scrollTo=WRzW5xSDDbNF

The website gave us a working model for a CNN neural network model but the accuracy was around 70%, therefore we need to modify the code to increase accuracy.

The first thing we needed to do was to make sure that we had all of the correct installations:

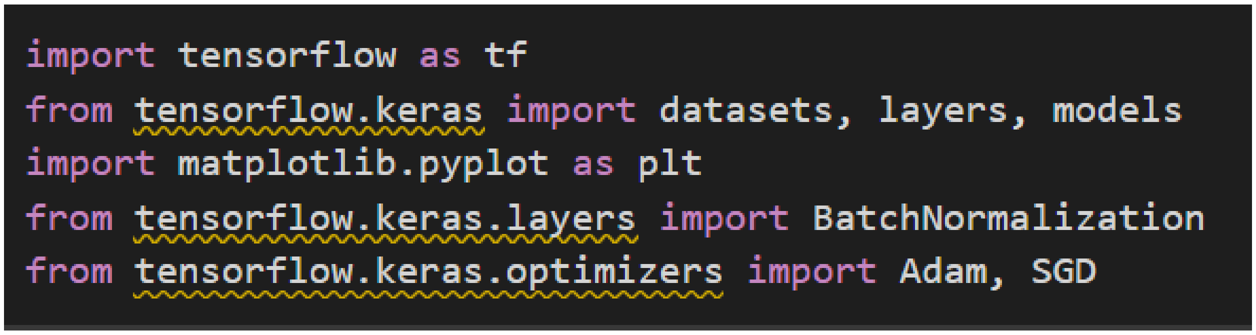


```
!pip install tensorflow
```



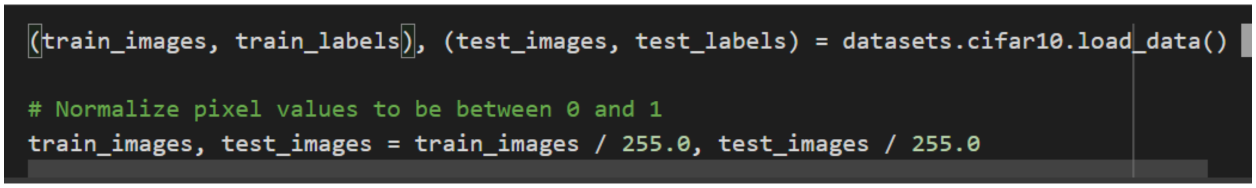
```
!pip install keras
!pip install h5py
!pip install Matplotlib
!pip install numpy
```

Once these were installed we began importing the correct libraries needed for the training.



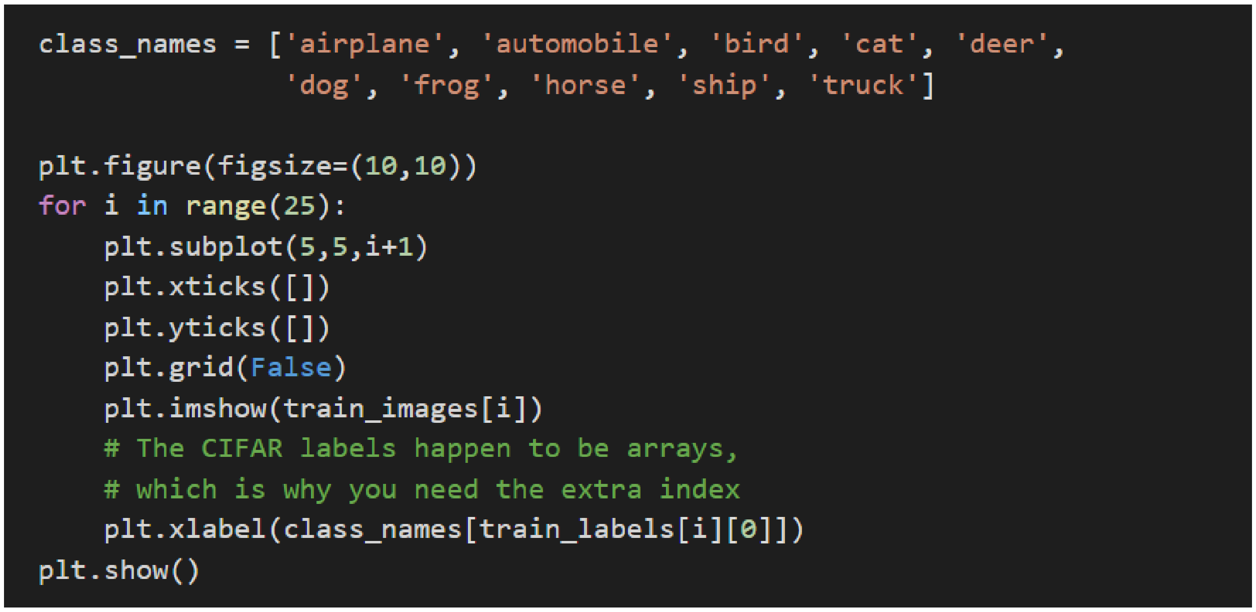
```python
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.optimizers import Adam, SGD
```

Next we prepared the CIFAR 10 data set used to train the model.

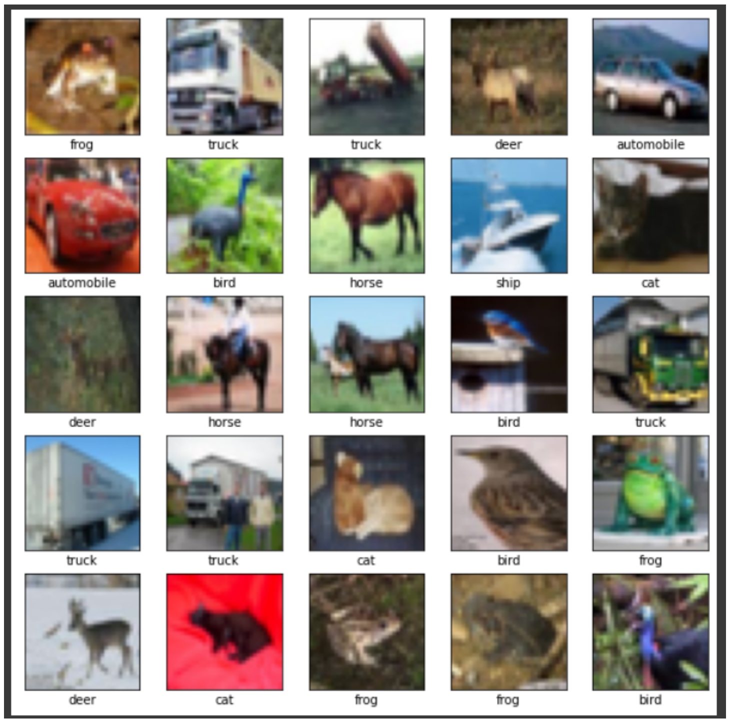


```python
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

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

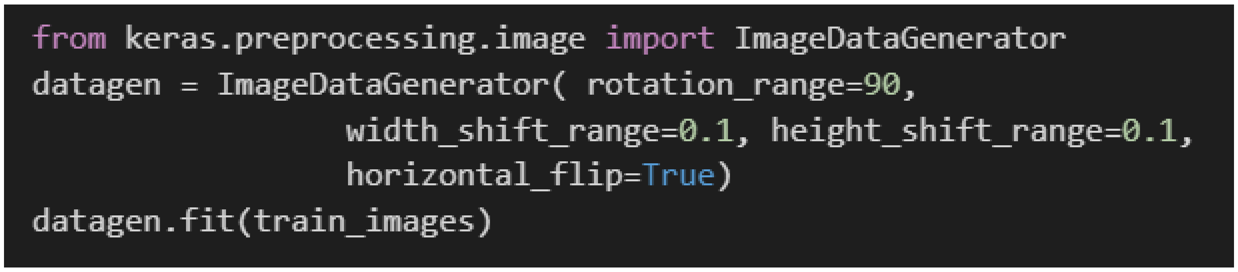
Then we verified the dataset by plotting the images along with the classification of the images:



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

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

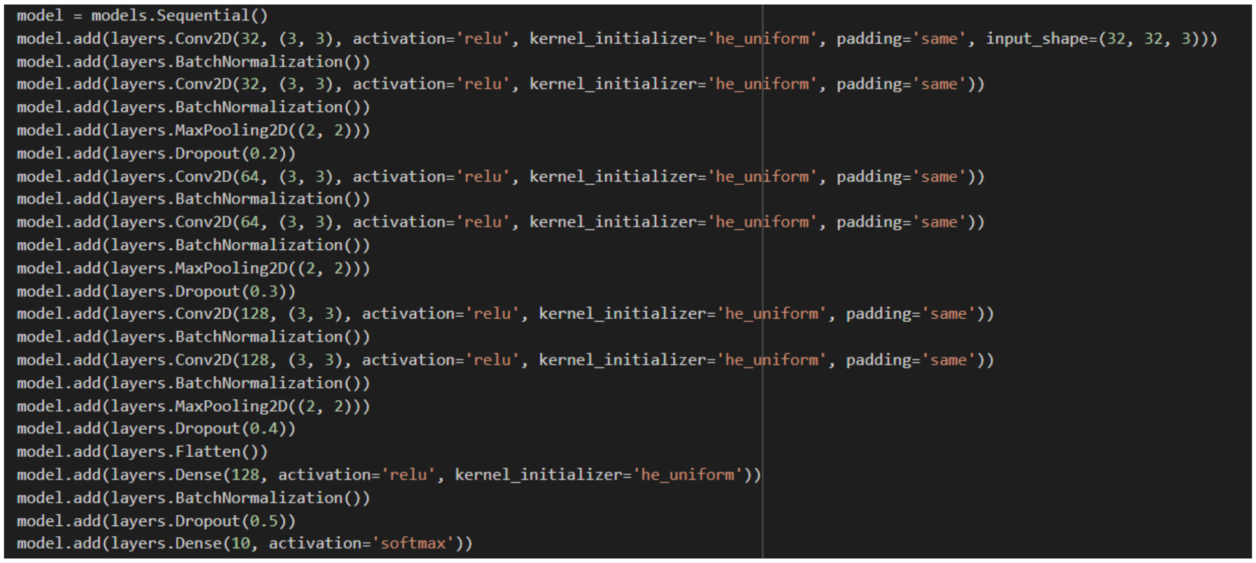


The next part is where we began optimizing the program to increase the accuracy. In order to this we used keras for data augmentation. The images were essentially resized for better accuracy to train the model.



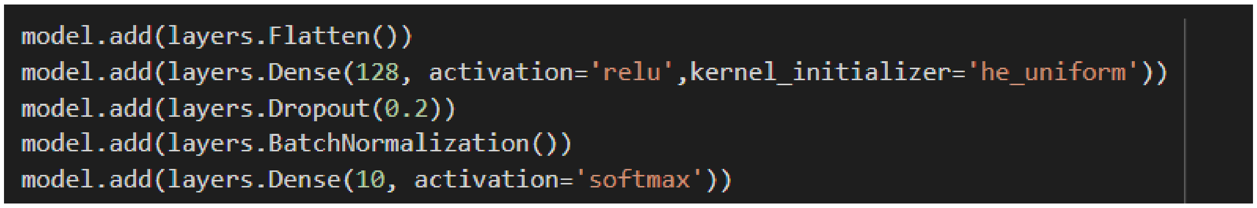
```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator( rotation_range=90,
                 width_shift_range=0.1, height_shift_range=0.1,
                 horizontal_flip=True)
datagen.fit(train_images)
```

Next, we used batch normalization in order to get better accuracy. This was done on the convolutional base. The input takes (image_height, image_width, and color). In our case, we added multiple layers to the convolution base to increase the accuracy.



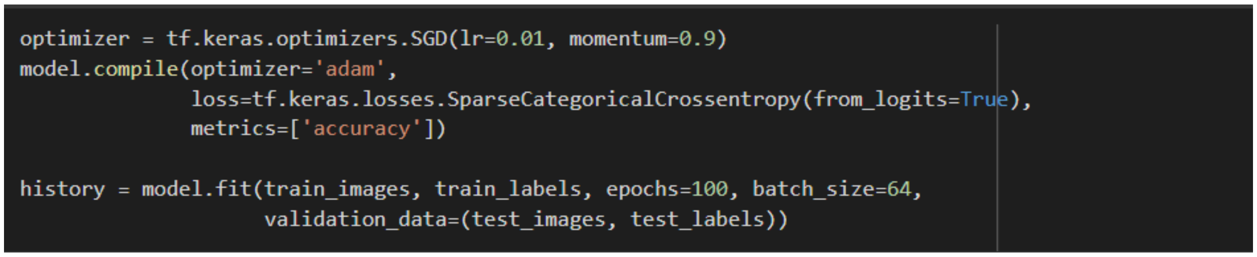
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(layers.BatchNormalization())
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.BatchNormalization())
model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(layers.BatchNormalization())
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.BatchNormalization())
model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(layers.BatchNormalization())
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.BatchNormalization())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10, activation='softmax'))
```

We printed a model summary and added a dense layer to the model to perform classification. It flattens a 3D layer to 1D and then adds a dense layer to the top. Once we made that change, we again printed out the model summary to see the changes in the training.



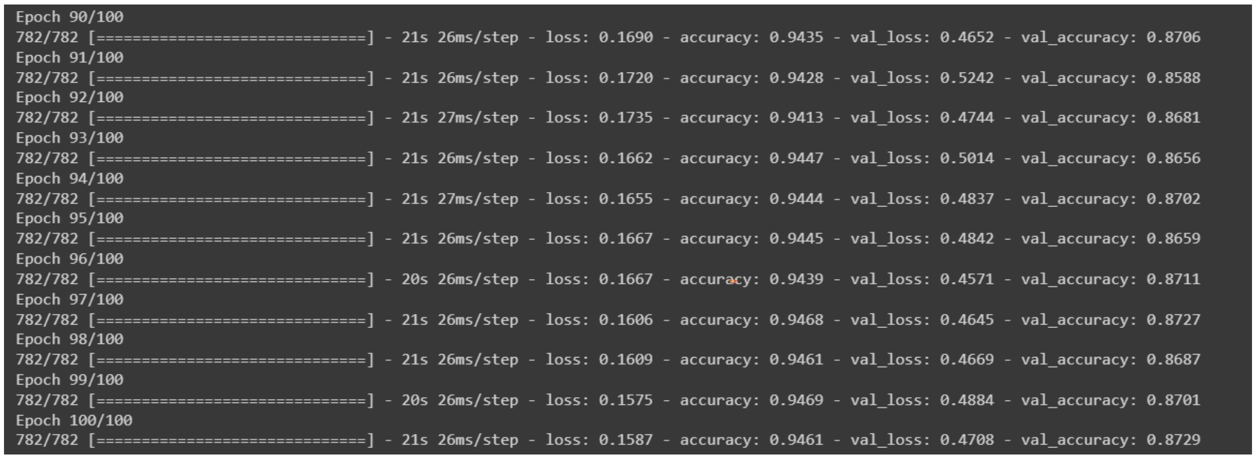
```
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu',kernel_initializer='he_uniform'))
model.add(layers.Dropout(0.2))
model.add(layers.BatchNormalization())
model.add(layers.Dense(10, activation='softmax'))
```

Lastly, we compiled and trained the model. Originally the training would last for 10 epochs, but we changed the epochs to 100, so the model would have more time to train and reach a better accuracy over time.



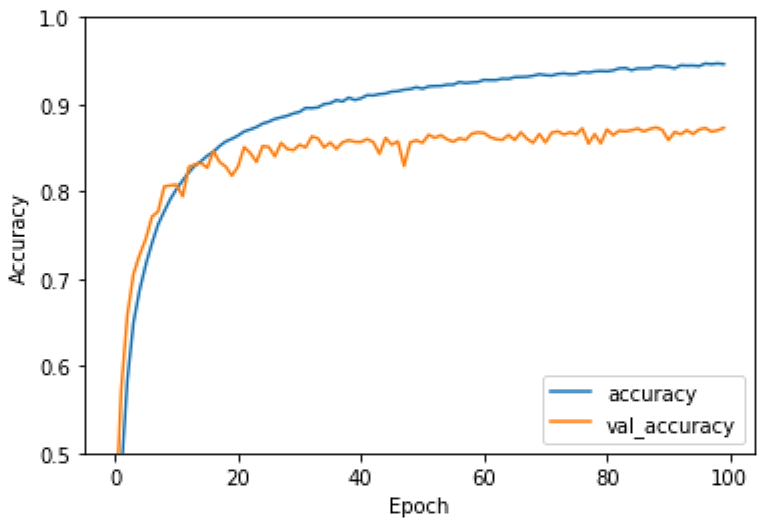
```
optimizer = tf.keras.optimizers.SGD(lr=0.01, momentum=0.9)
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=100, batch_size=64,
                    validation_data=(test_images, test_labels))
```

Once we clicked play the model began training and it took about 21 seconds to get through each epoch and since we ran it for 100 epochs it took a while. Once it was done running we got an accuracy of 0.8729 which is approximately 87.29%.

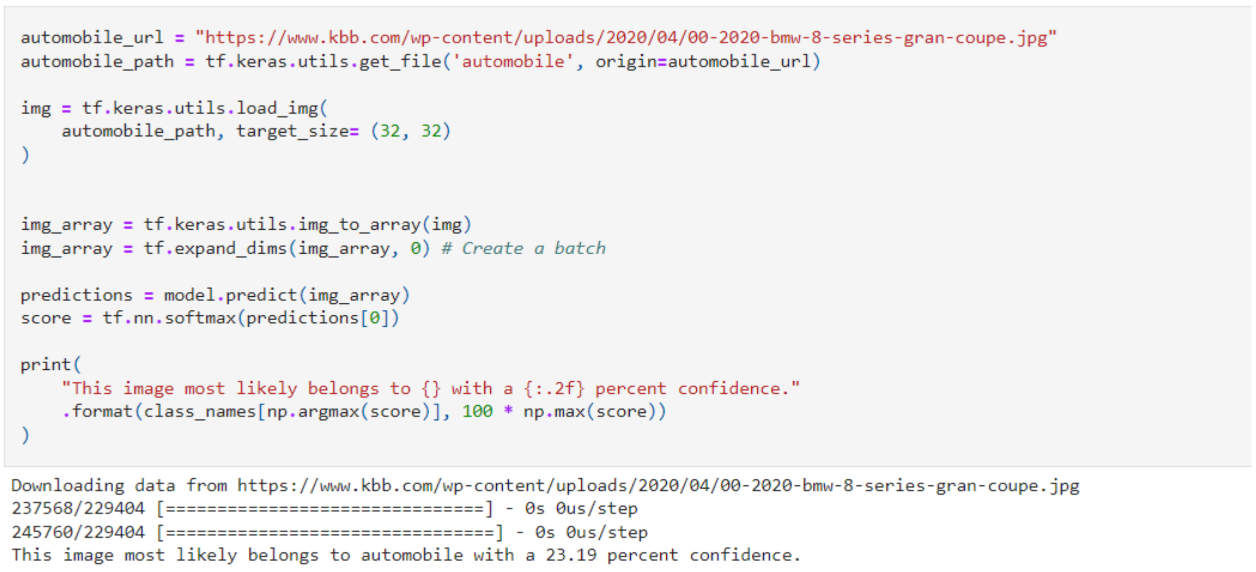


```
Epoch 90/100
782/782 [==============================] - 21s 26ms/step - loss: 0.1690 - accuracy: 0.9435 - val_loss: 0.4652 - val_accuracy: 0.8706
Epoch 91/100
782/782 [==============================] - 21s 26ms/step - loss: 0.1720 - accuracy: 0.9428 - val_loss: 0.5242 - val_accuracy: 0.8588
Epoch 92/100
782/782 [==============================] - 21s 27ms/step - loss: 0.1735 - accuracy: 0.9413 - val_loss: 0.4744 - val_accuracy: 0.8681
Epoch 93/100
782/782 [==============================] - 21s 26ms/step - loss: 0.1662 - accuracy: 0.9447 - val_loss: 0.5014 - val_accuracy: 0.8656
Epoch 94/100
782/782 [==============================] - 21s 27ms/step - loss: 0.1655 - accuracy: 0.9444 - val_loss: 0.4837 - val_accuracy: 0.8702
Epoch 95/100
782/782 [==============================] - 21s 26ms/step - loss: 0.1667 - accuracy: 0.9445 - val_loss: 0.4842 - val_accuracy: 0.8659
Epoch 96/100
782/782 [==============================] - 20s 26ms/step - loss: 0.1667 - accuracy: 0.9439 - val_loss: 0.4571 - val_accuracy: 0.8711
Epoch 97/100
782/782 [==============================] - 21s 26ms/step - loss: 0.1606 - accuracy: 0.9468 - val_loss: 0.4645 - val_accuracy: 0.8727
Epoch 98/100
782/782 [==============================] - 21s 26ms/step - loss: 0.1609 - accuracy: 0.9461 - val_loss: 0.4669 - val_accuracy: 0.8687
Epoch 99/100
782/782 [==============================] - 20s 26ms/step - loss: 0.1575 - accuracy: 0.9469 - val_loss: 0.4884 - val_accuracy: 0.8701
Epoch 100/100
782/782 [==============================] - 21s 26ms/step - loss: 0.1587 - accuracy: 0.9461 - val_loss: 0.4708 - val_accuracy: 0.8729
```

We then plotted our results and printed the accuracy.

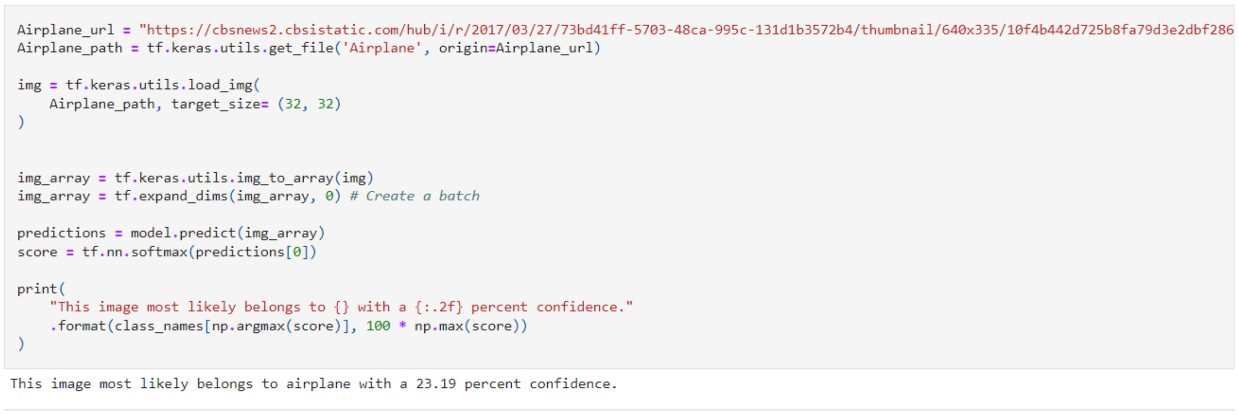


The next part of the lab was to test the accuracy by checking if it could recognize the images. Below are the results and even though it was able to recognize a few images, so of them were a bit inaccurate.



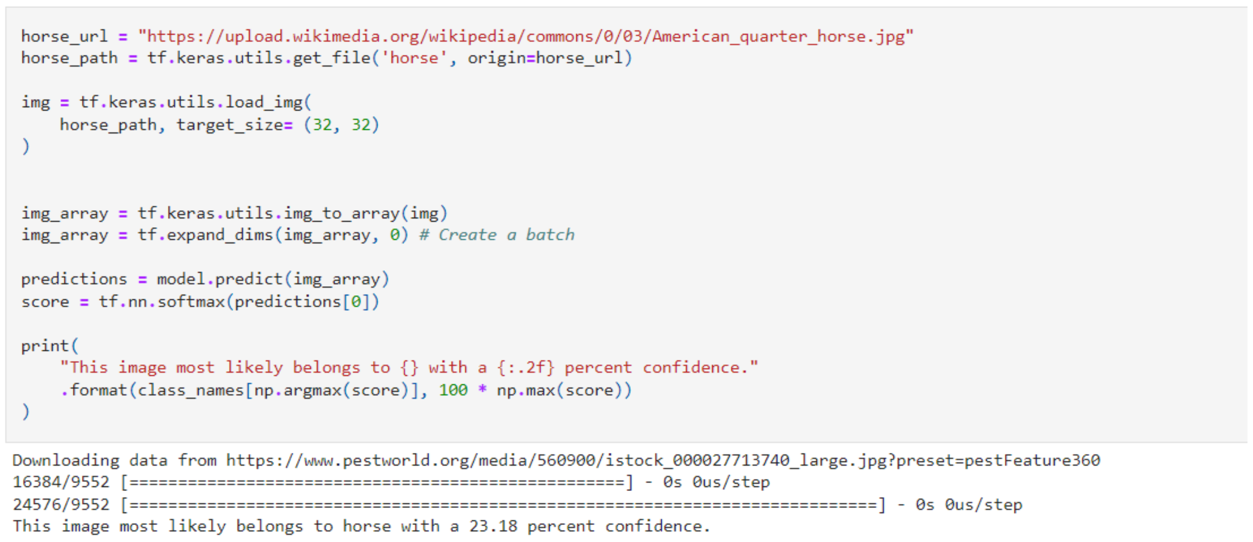
```
automobile_url = "https://www.kbb.com/wp-content/uploads/2020/04/00-2020-bmw-8-series-gran-coupe.jpg"
automobile_path = tf.keras.utils.get_file('automobile', origin=automobile_url)

img = tf.keras.utils.load_img(
    automobile_path, target_size= (32, 32)
)


img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
Downloading data from https://www.kbb.com/wp-content/uploads/2020/04/00-2020-bmw-8-series-gran-coupe.jpg
237568/229404 [===============================] - 0s 0us/step
245760/229404 [================================] - 0s 0us/step
This image most likely belongs to automobile with a 23.19 percent confidence.
```



```
Airplane_url = "https://cbsnews2.cbsistatic.com/hub/i/r/2017/03/27/73bd41ff-5703-48ca-995c-131d1b3572b4/thumbnail/640x335/10f4b442d725b8fa79d3e2dbf286
Airplane_path = tf.keras.utils.get_file('Airplane', origin=Airplane_url)

img = tf.keras.utils.load_img(
    Airplane_path, target_size= (32, 32)
)


img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

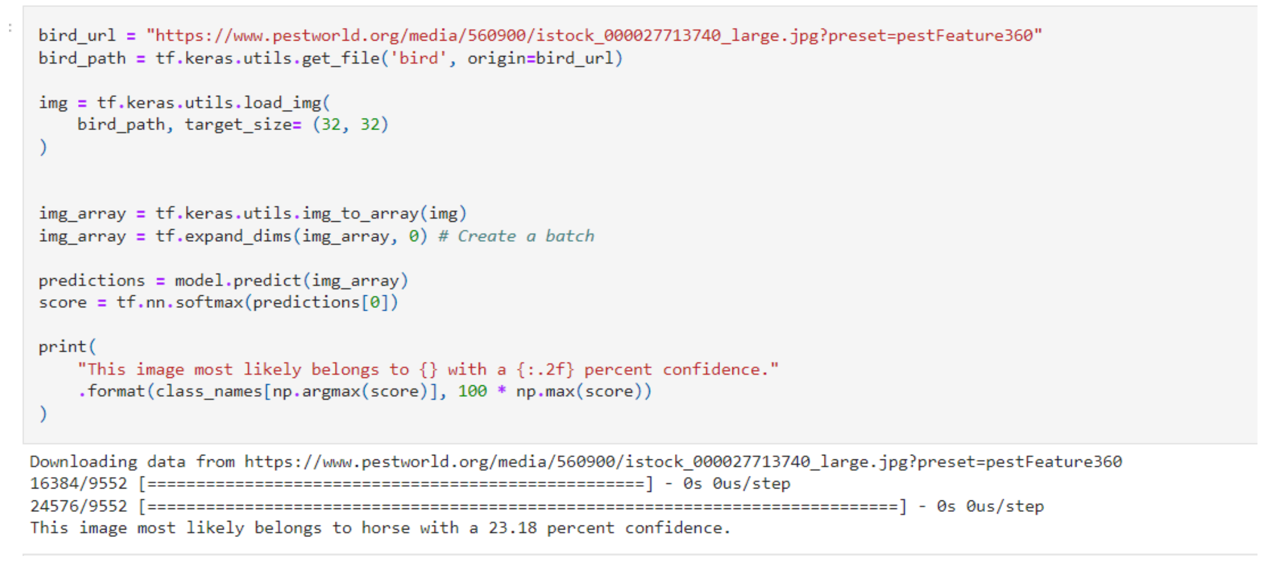
```
This image most likely belongs to airplane with a 23.19 percent confidence.
```



```
horse_url = "https://upload.wikimedia.org/wikipedia/commons/0/03/American_quarter_horse.jpg"
horse_path = tf.keras.utils.get_file('horse', origin=horse_url)

img = tf.keras.utils.load_img(
    horse_path, target_size= (32, 32)
)


img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

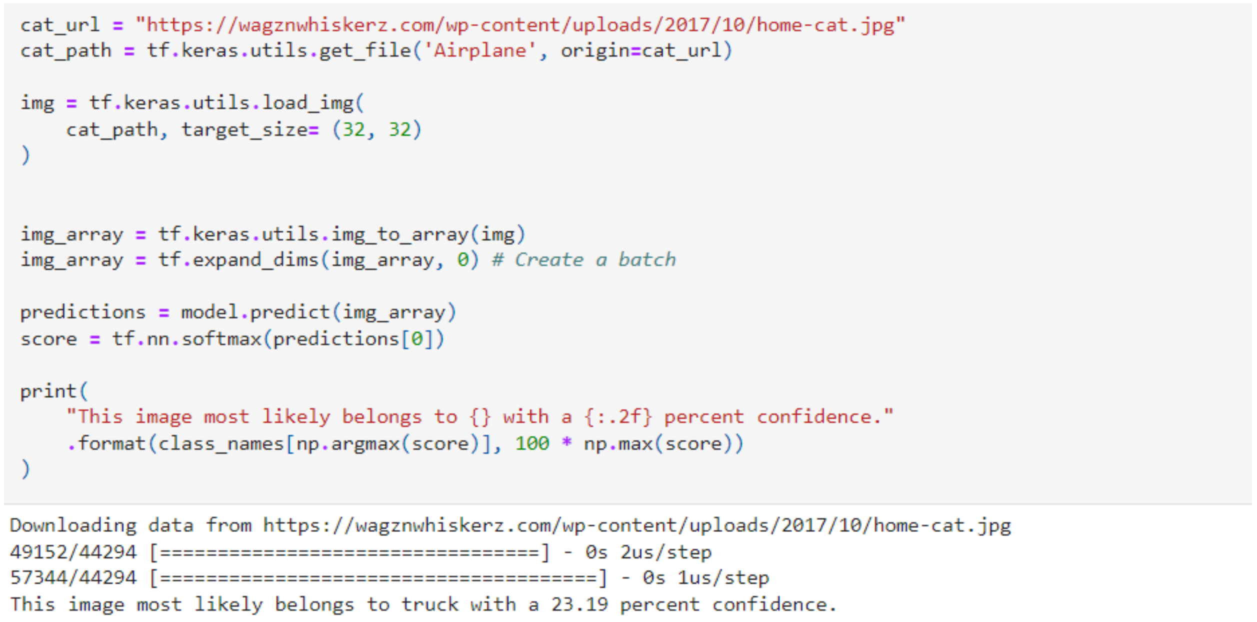
```
Downloading data from https://www.pestworld.org/media/560900/istock_000027713740_large.jpg?preset=pestFeature360
16384/9552 [==================================================] - 0s 0us/step
24576/9552 [===========================================================================] - 0s 0us/step
This image most likely belongs to horse with a 23.18 percent confidence.
```

As you can see it is able to recognize the objects but with low confidence, here are some of the non-recognizable images.



```
bird_url = "https://www.pestworld.org/media/560900/istock_000027713740_large.jpg?preset=pestFeature360"
bird_path = tf.keras.utils.get_file('bird', origin=bird_url)

img = tf.keras.utils.load_img(
    bird_path, target_size= (32, 32)
)


img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
Downloading data from https://www.pestworld.org/media/560900/istock_000027713740_large.jpg?preset=pestFeature360
16384/9552 [==================================================] - 0s 0us/step
24576/9552 [===========================================================================] - 0s 0us/step
This image most likely belongs to horse with a 23.18 percent confidence.
```

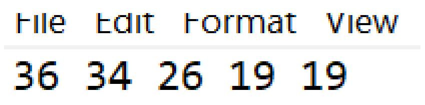


```
cat_url = "https://wagznwhiskerz.com/wp-content/uploads/2017/10/home-cat.jpg"
cat_path = tf.keras.utils.get_file('Airplane', origin=cat_url)

img = tf.keras.utils.load_img(
    cat_path, target_size= (32, 32)
)


img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

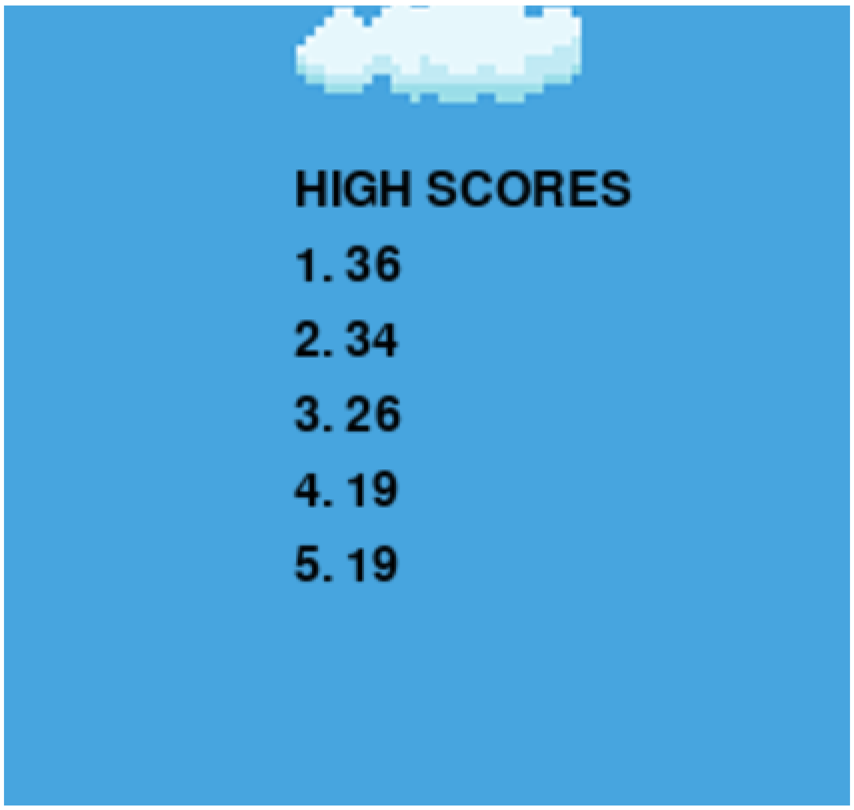
```
Downloading data from https://wagznwhiskerz.com/wp-content/uploads/2017/10/home-cat.jpg
49152/44294 [=================================] - 0s 2us/step
57344/44294 [======================================] - 0s 1us/step
This image most likely belongs to truck with a 23.19 percent confidence.
```

The last part of the lab was to modify the Ballonflight game. Once we got the base code for the lab we needed to make 4 changes to the code. In our case the changes we made were more high scores, speed it up, level up, and space out obstacles.

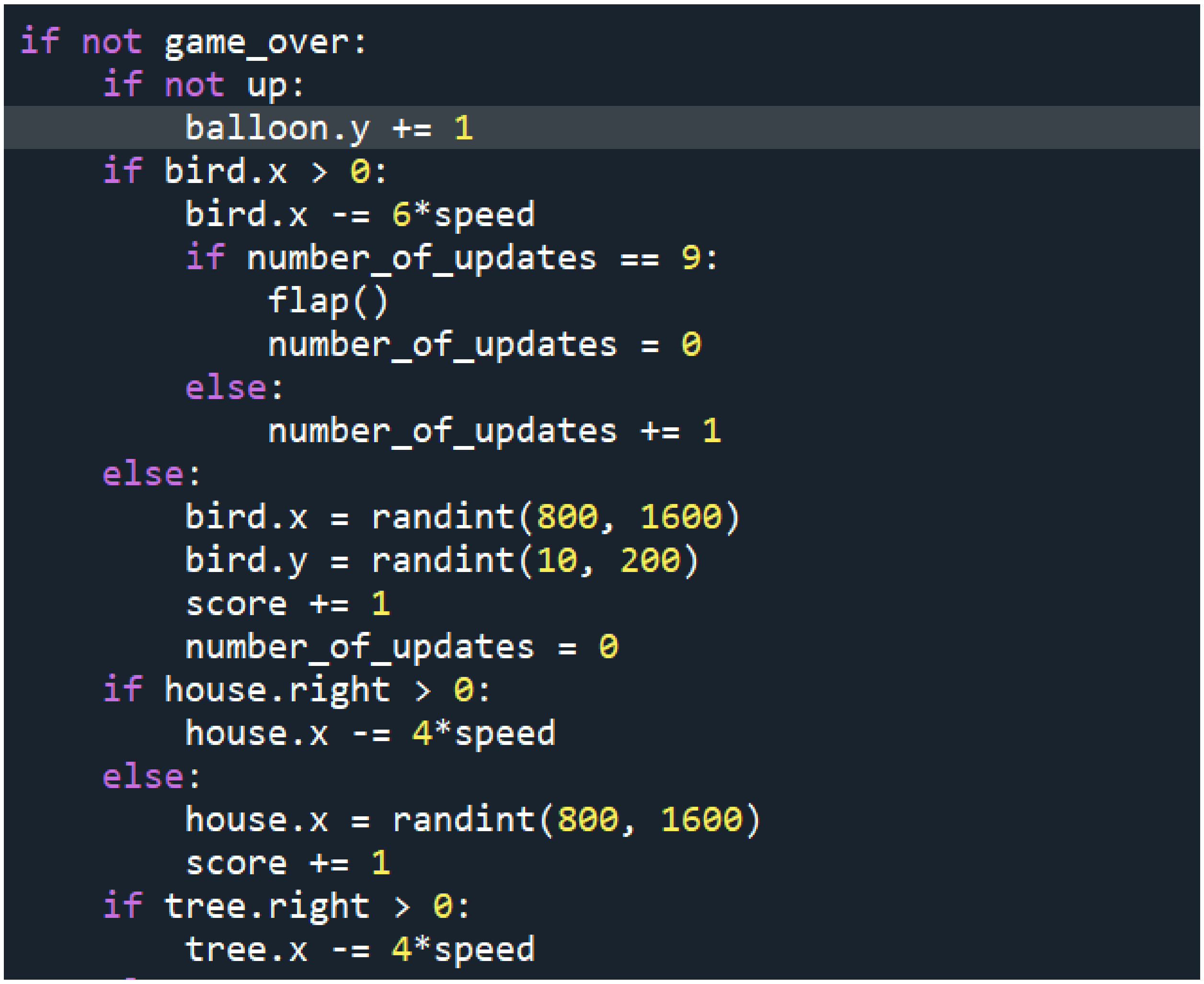
The first change we made was more high scores which was very simple. In the text file where the high scores were being saved, we added two more columns to the text files so that more high scores can be saved. Originally it was 3, but we changed it to 5.



```
36 34 26 19 19
```

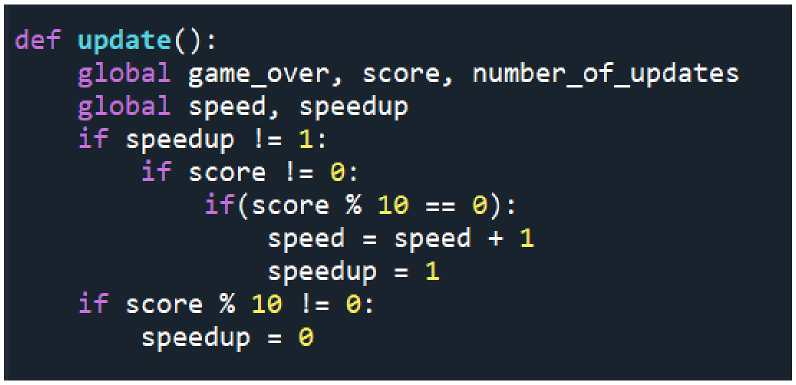

HIGH SCORES
1. 36
2. 34
3. 26
4. 19
5. 19

In order to speed up the game, we made changes in the position of the obstacles quicker by increasing their x value. The following program shows the changes



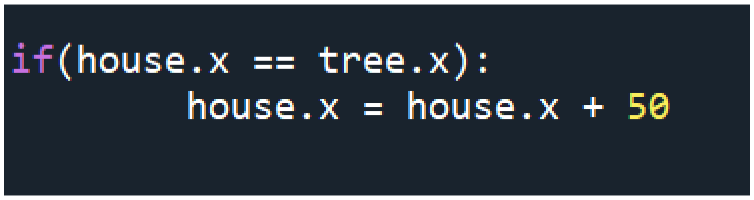
```
if not game_over:
    if not up:
        balloon.y += 1
    if bird.x > 0:
        bird.x -= 6*speed
        if number_of_updates == 9:
            flap()
            number_of_updates = 0
        else:
            number_of_updates += 1
    else:
        bird.x = randint(800, 1600)
        bird.y = randint(10, 200)
        score += 1
        number_of_updates = 0
    if house.right > 0:
        house.x -= 4*speed
    else:
        house.x = randint(800, 1600)
        score += 1
    if tree.right > 0:
        tree.x -= 4*speed
```

In the demonstration video, you will see that the game starts off at a quicker speed than the default code. The next feature we made was to level up, and for our code we programmed it so that each time the user had a score which a multiple of 10, the game speed would increase by 1. The conditional statement below shows how we implemented it in the game.



```
def update():
    global game_over, score, number_of_updates
    global speed, speedup
    if speedup != 1:
        if score != 0:
            if(score % 10 == 0):
                speed = speed + 1
                speedup = 1
    if score % 10 != 0:
        speedup = 0
```

The last thing we added to the game was spacing out the images so that the objects would not appear over each other. Since the house and the tree were mostly overlapping, those were the two obstacles that consistently overlapped. The following code shows the conditional statement which changes the position of the tree if it spawns in the same location as the house.



```
if(house.x == tree.x):
        house.x = house.x + 50
```