Documentation

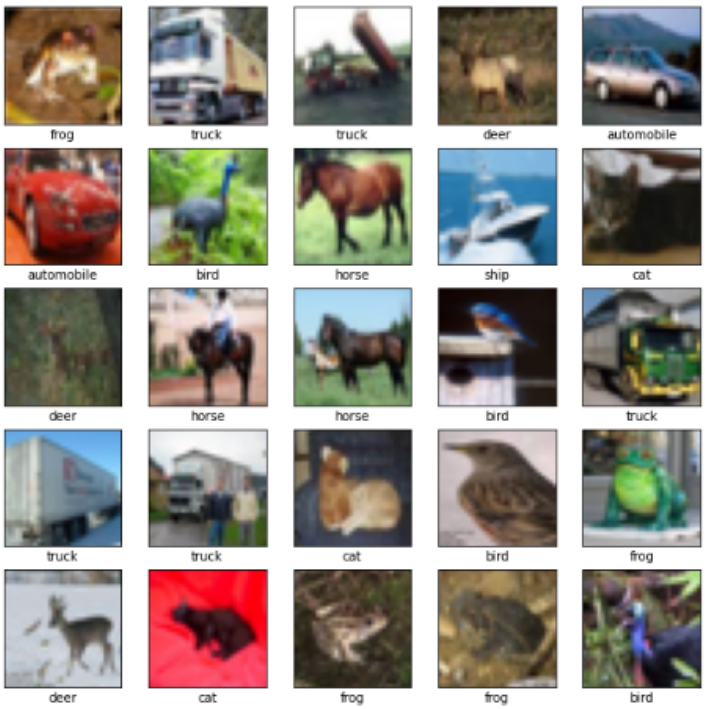
For this lab, google collab was used to train a model in order to recognize certain images. A CNN template was run using the training with modified code in order to receive a higher accuracy for the model after training it. A tutorial was used from the following website to begin the lab:
https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/cnn.ipynb#scrollTo=WRzW5xSDDbNF

This website provided a working CNN mode with an accuracy of around 70% which needed to be modified in order to increase the accuracy. For this lab, several installations must be made using the pip install command which were tensorflow, keras, h5py, Matplotlib, and numpy. After this, the libraries required were imported as well.

```
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
```

After this, the CIFAR 10 dataset is used to train the model and then verified by plotting the images with the correct classification of the image.

```
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 involves increasing the accuracy of the program using keras for data augmentation. All of the images were resized to better train the model for higher accuracy.

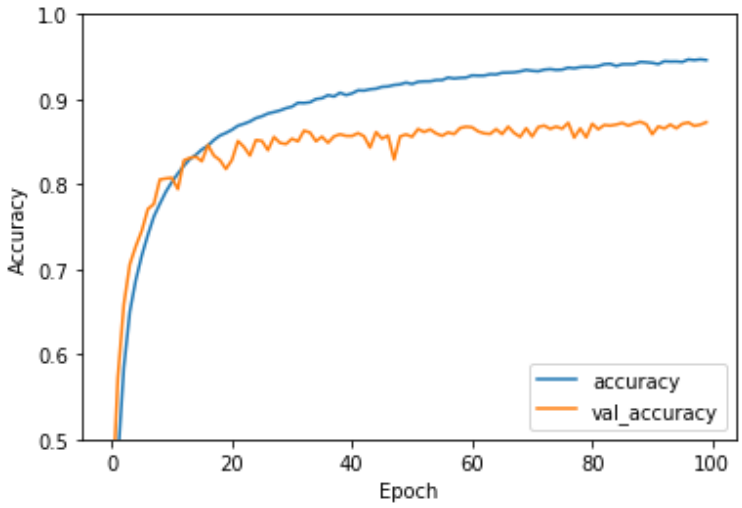
```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator( rotation_range=90,
                width_shift_range=0.1, height_shift_range=0.1,
                horizontal_flip=True)
datagen.fit(train_images)
```

After this, batch normalization was done as well which takes the image height, width and color to add layers to the convolution base to increase 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'))
```

After this, a model summary was printed and a dense layer was added to perform the classification which essentially flattens a 3D layer to 1D. Once this change was implemented, the model was run with the training lasting for 100 epochs in order to have more time for the model to train and provide a better accuracy over time. Once the model was done running, the accuracy that was received was 0.8771 which is around 87.71%.

```
782/782 [==============================] - 20s 26ms/step - loss: 0.1674 - accuracy: 0.9441 - val_loss: 0.4618 - val_accuracy: 0.8752
Epoch 94/100
782/782 [==============================] - 20s 26ms/step - loss: 0.1645 - accuracy: 0.9453 - val_loss: 0.4907 - val_accuracy: 0.8638
Epoch 95/100
782/782 [==============================] - 20s 26ms/step - loss: 0.1581 - accuracy: 0.9474 - val_loss: 0.4758 - val_accuracy: 0.8737
Epoch 96/100
782/782 [==============================] - 20s 26ms/step - loss: 0.1675 - accuracy: 0.9444 - val_loss: 0.4578 - val_accuracy: 0.8765
Epoch 97/100
782/782 [==============================] - 21s 26ms/step - loss: 0.1599 - accuracy: 0.9458 - val_loss: 0.4946 - val_accuracy: 0.8691
Epoch 98/100
782/782 [==============================] - 20s 26ms/step - loss: 0.1624 - accuracy: 0.9466 - val_loss: 0.5095 - val_accuracy: 0.8599
Epoch 99/100
782/782 [==============================] - 20s 26ms/step - loss: 0.1582 - accuracy: 0.9464 - val_loss: 0.4553 - val_accuracy: 0.8751
Epoch 100/100
782/782 [==============================] - 20s 26ms/step - loss: 0.1572 - accuracy: 0.9471 - val_loss: 0.4771 - val_accuracy: 0.8771
```



Next, we tested the accuracy of the model by recognizing unrecognizable images that were rescued and fed into the CNN. Several images were correctly identified with others being incorrectly identified at times.

```
[ ]  automobile_url = "https://images.all-free-download.com/images/graphiclarge/classic_jaguar_210354.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://images.all-free-download.com/images/graphiclarge/classic_jaguar_210354.jpg
98304/93791 [==============================] - 0s 0us/step
106496/93791 [================================] - 0s 0us/step
This image most likely belongs to automobile with a 23.19 percent confidence.
```

```
airplane_url = "https://static.wikia.nocookie.net/lostpedia/images/f/f7/Kateandplane.jpg/revision/latest/top-crop/width/360/height/450?cb=20061109012445"
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 automobile with a 23.19 percent confidence.
```

```
bird_url = "https://upload.wikimedia.org/wikipedia/commons/5/53/Weaver_bird.jpg"
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://upload.wikimedia.org/wikipedia/commons/5/53/Weaver_bird.jpg
319488/313992 [==============================] - 0s 0us/step
327680/313992 [==============================] - 0s 0us/step
This image most likely belongs to bird with a 23.19 percent confidence.
```

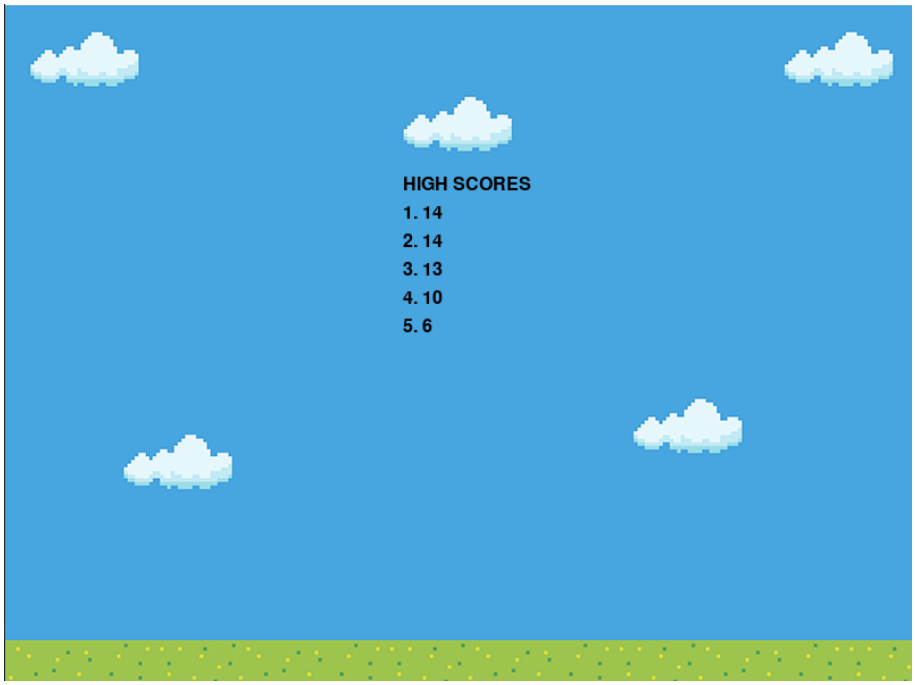
```
cat_url = "https://static.toiimg.com/thumb/msid-67586673,width-1070,height-580,overlay-toi_sw,pt-32,y_pad-40,resizemode-75,imgsize-3918697/67586673.jpg"
cat_path = tf.keras.utils.get_file('cat', origin=cat_url)

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://static.toiimg.com/thumb/msid-67586673,width-1070,height-580,overlay-toi_sw,pt-32,y_pad-40,resizemode-75,imgsize-3918697/67586673.jpg
65536/60396 [================================] - 0s 0us/step
73728/60396 [==================================] - 0s 0us/step
This image most likely belongs to truck with a 23.19 percent confidence.
```

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

The last part of the lab involves modifying 4 things in the Ballonflight game. The changes that were made were having more scores displayed, level up, speeding up the game, and spacing out the obstacles.

The first change was simple, which was to have higher scores. To do this, the text file where the high scores are saved added two more columns so that more scores could be saved. It went from 3 scores saved to 5.



To speed up the game, some changes were made to the position of the obstacles which were modified to become faster.

```
    if not game_over:
        if not up:
            balloon.y += 1
        if bird.x > 0:
            bird.x -= 4*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 -= 2*speed
        else:
            house.x = randint(800, 1600)
            score += 1
        if tree.right > 0:
            tree.x -= 2*speed
        else:
            tree.x = randint(800, 1600)
            score += 1
        if balloon.top < 0 or balloon.bottom > 560:
            game_over = True
            update_high_scores()
        if balloon.collidepoint(bird.x, bird.y) or balloon.collidepoint(house.x, house.y) or balloon.collidepoint(tree.x
            game_over = True
            update_high_scores()
        scoreupdate = score
pgzrun.go()
```

The next feature was the level up speed so that each time the score had a multiple of 10, the speed of the game would increase by 1.

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

The last change was spacing out the obstacles so that they would not overlap each other during the game.

```
  if(house.x == tree.x):
          tree.x = tree.x + 20
```

The demonstration video includes all of these things.