Lab 8 Documentation

Name: Crystal Low

Class: EE 104

April 22, 2022

Professor Pham

Abstract: This lab consists of three parts, the first part is to explore the CNN code on google collaboration and achieve at least 90% accuracy and recognize 90% of the images provided. Then there is a challenge test with CNN to recognize five of the provided unrecognizable images from the test images. The last part of this lab is to make modifications to the balloon flight game.

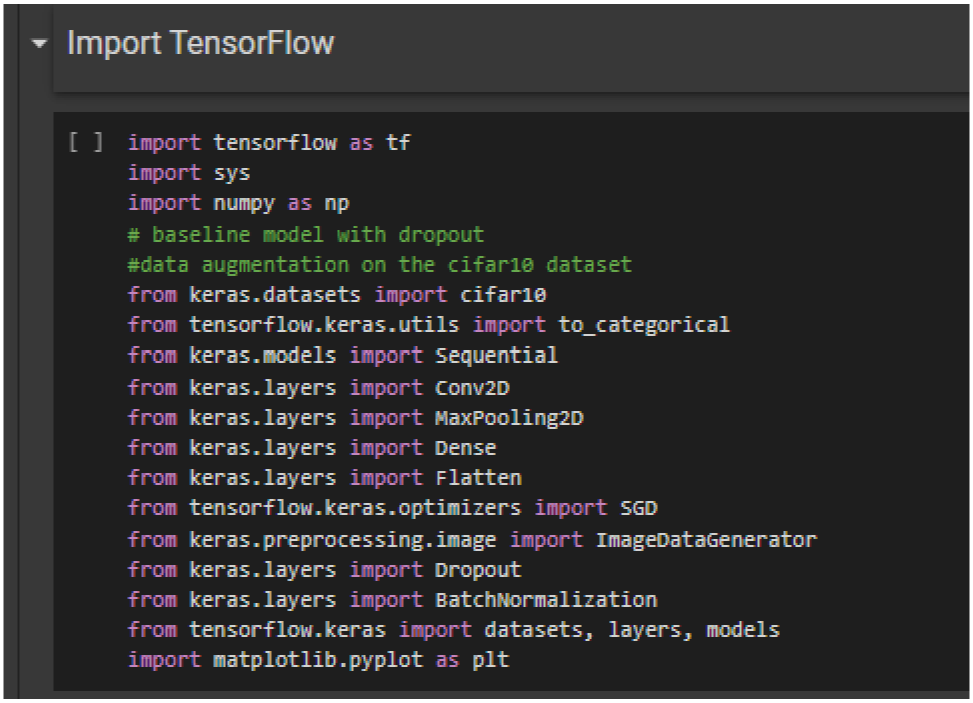
Objective:

The objective of this lab is to make modifications to the CNN code that is provided to improve the accuracy of the recognition of the given images. The achievement can be done using any method including baseline, increasing dropout, data augmentation, batch normalization, or any method that the developer would like. In the challenge test part of this lab, the goal is to recognize at least five unrecognizable images. The lab objective is to modify the balloon flight game by either, adding more high score spots, having multiple lives, speeding up the object, adding more objects to make it harder for the user, file handling, different ways to score, having a level up, or space out the obstacles. The goal is to complete four out of eight of the provided tweaks to the game.

Instructions:

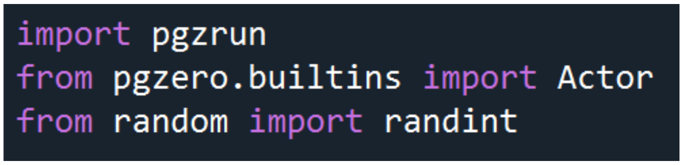
Import Packages:

CNN:



```
import tensorflow as tf
import sys
import numpy as np
# baseline model with dropout
#data augmentation on the cifar10 dataset
from keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from tensorflow.keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dropout
from keras.layers import BatchNormalization
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Balloon Flight:



```
import pgzrun
from pgzero.builtins import Actor
from random import randint
```

References:

Module 8 provided by Professor Pham

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

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

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/classification.ipynb#scrollTo=dC40sRITBSsQ

https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/

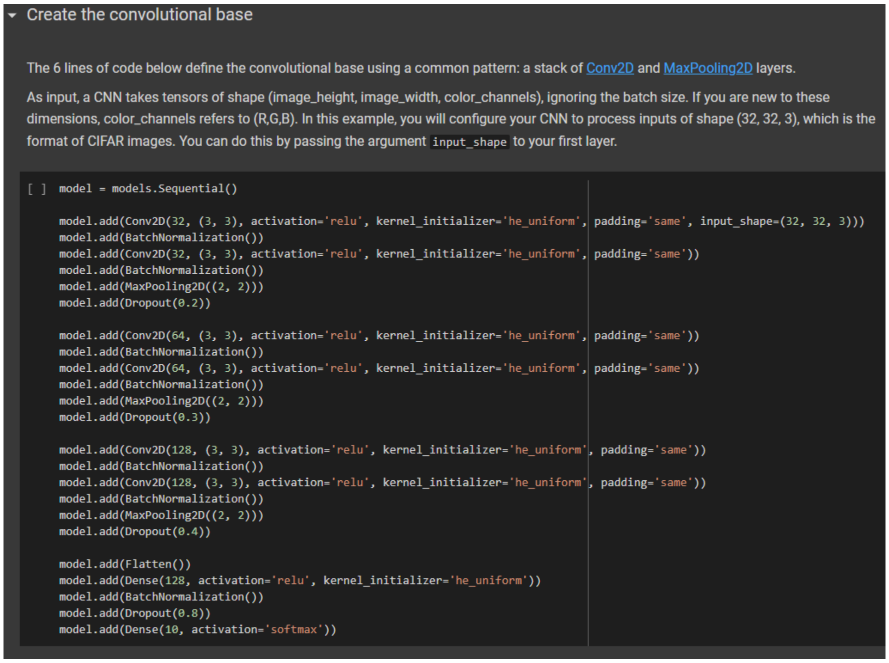
<u>CNN:</u>

In this code, the goal was to improve the accuracy of the images that were provided. The layers were fixed to accommodate the accuracy. The results will show the final accuracy that was captured. This code was running multiple times to get these results. It was tweaked and modified many times however 85% was the highest it has got.

1) In the layers section, the baseline model was created to help improve the accuracy.

   Referenced this website:

   https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/



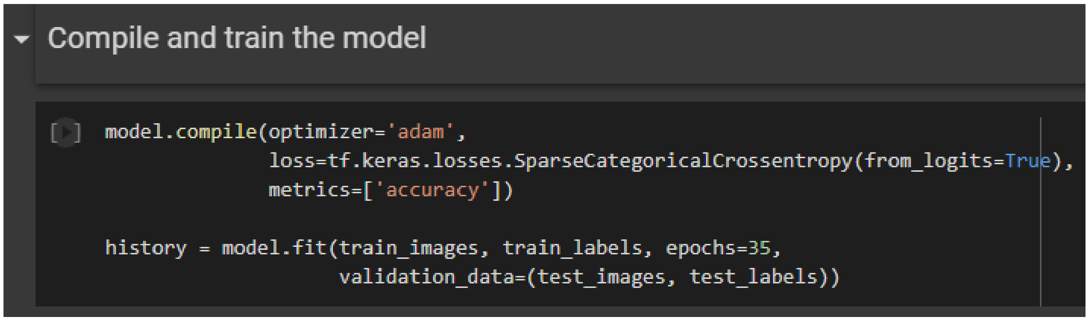
Create the convolutional base

The 6 lines of code below define the convolutional base using a common pattern: a stack of Conv2D and MaxPooling2D layers.

As input, a CNN takes tensors of shape (image_height, image_width, color_channels), ignoring the batch size. If you are new to these dimensions, color_channels refers to (R,G,B). In this example, you will configure your CNN to process inputs of shape (32, 32, 3), which is the format of CIFAR images. You can do this by passing the argument `input_shape` to your first layer.

```
[ ] model = models.Sequential()

    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
    model.add(BatchNormalization())
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))

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

    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(BatchNormalization())
    model.add(Dropout(0.8))
    model.add(Dense(10, activation='softmax'))
```

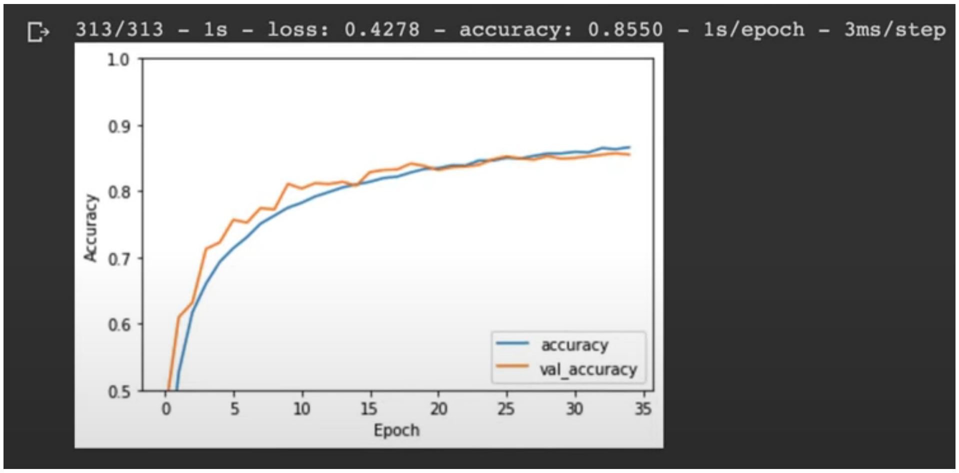
2) Added more epochs to obtain a better range of accuracy



Compile and train the model

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

history = model.fit(train_images, train_labels, epochs=35,
                    validation_data=(test_images, test_labels))
```

3) The results with the accuracy

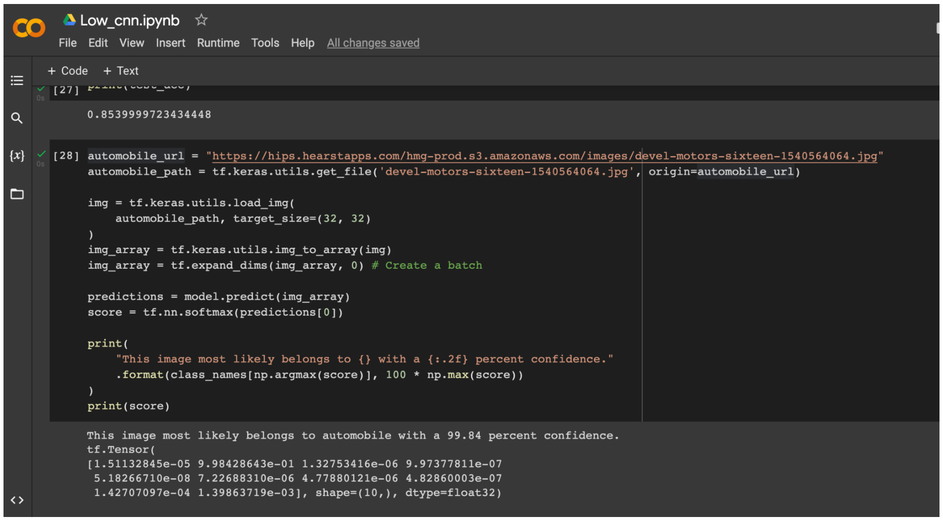


```
313/313 - 1s - loss: 0.4278 - accuracy: 0.8550 - 1s/epoch - 3ms/step
```

CNN Challenge Test

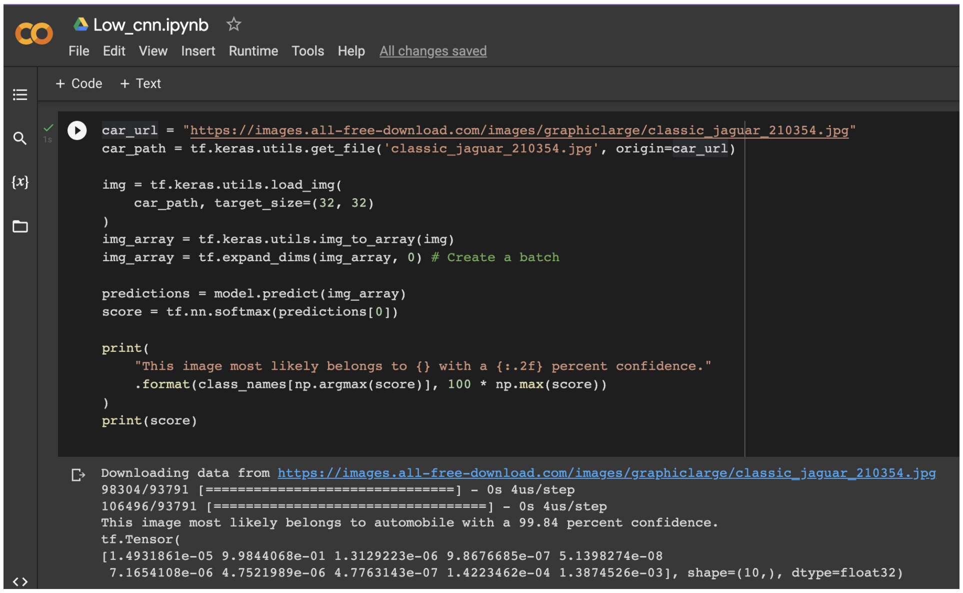
The goal of this lab is to recognize the unrecognizable images from the test images section provided in the lab instructions.

1) Added a cell to recognize the automobile, figured the frame size that will cater to the image put in the link of the image, and classify under an automobile.



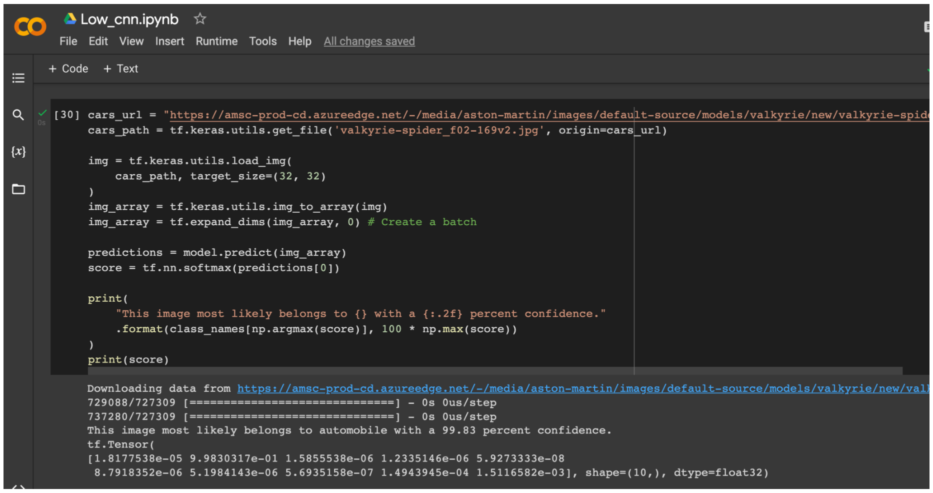
```
[27] print(test_acc)

     0.8539999723434448

[28] automobile_url = "https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/devel-motors-sixteen-1540564064.jpg"
     automobile_path = tf.keras.utils.get_file('devel-motors-sixteen-1540564064.jpg', 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))
     )
     print(score)

     This image most likely belongs to automobile with a 99.84 percent confidence.
     tf.Tensor(
     [1.51132845e-05 9.98428643e-01 1.32753416e-06 9.97377811e-07
      5.18266710e-08 7.22688310e-06 4.77880121e-06 4.82860003e-07
      1.42707097e-04 1.39863719e-03], shape=(10,), dtype=float32)
```

2) Second image is classified as an automobile as it is an automobile



```
car_url = "https://images.all-free-download.com/images/graphiclarge/classic_jaguar_210354.jpg"
car_path = tf.keras.utils.get_file('classic_jaguar_210354.jpg', origin=car_url)

img = tf.keras.utils.load_img(
    car_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))
)
print(score)

Downloading data from https://images.all-free-download.com/images/graphiclarge/classic_jaguar_210354.jpg
98304/93791 [===============================] - 0s 4us/step
106496/93791 [==================================] - 0s 4us/step
This image most likely belongs to automobile with a 99.84 percent confidence.
tf.Tensor(
[1.4931861e-05 9.9844068e-01 1.3129223e-06 9.8676685e-07 5.1398274e-08
 7.1654108e-06 4.7521989e-06 4.7763143e-07 1.4223462e-04 1.3874526e-03], shape=(10,), dtype=float32)
```

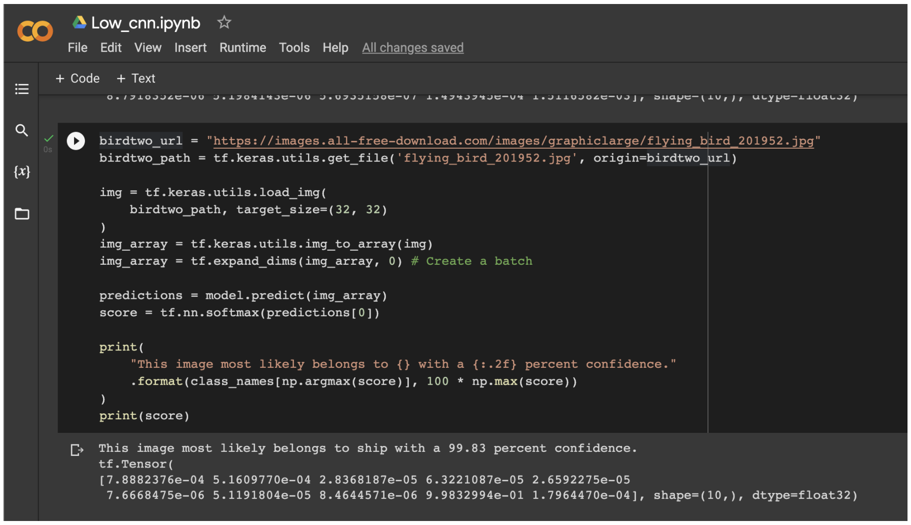
3) Third image classified as an automobile as it is an automobile



```
[30] cars_url = "https://amsc-prod-cd.azureedge.net/-/media/aston-martin/images/default-source/models/valkyrie/new/valkyrie-spide
     cars_path = tf.keras.utils.get_file('valkyrie-spider_f02-169v2.jpg', origin=cars_url)

     img = tf.keras.utils.load_img(
         cars_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))
     )
     print(score)
```

```
Downloading data from https://amsc-prod-cd.azureedge.net/-/media/aston-martin/images/default-source/models/valkyrie/new/valk
729088/727309 [==============================] - 0s 0us/step
737280/727309 [==============================] - 0s 0us/step
This image most likely belongs to automobile with a 99.83 percent confidence.
tf.Tensor(
[1.8177538e-05 9.9830317e-01 1.5855538e-06 1.2335146e-06 5.9273333e-08
 8.7918352e-06 5.1984143e-06 5.6935158e-07 1.4943945e-04 1.5116582e-03], shape=(10,), dtype=float32)
```

4) The fourth image was classified as a ship when it was a bird

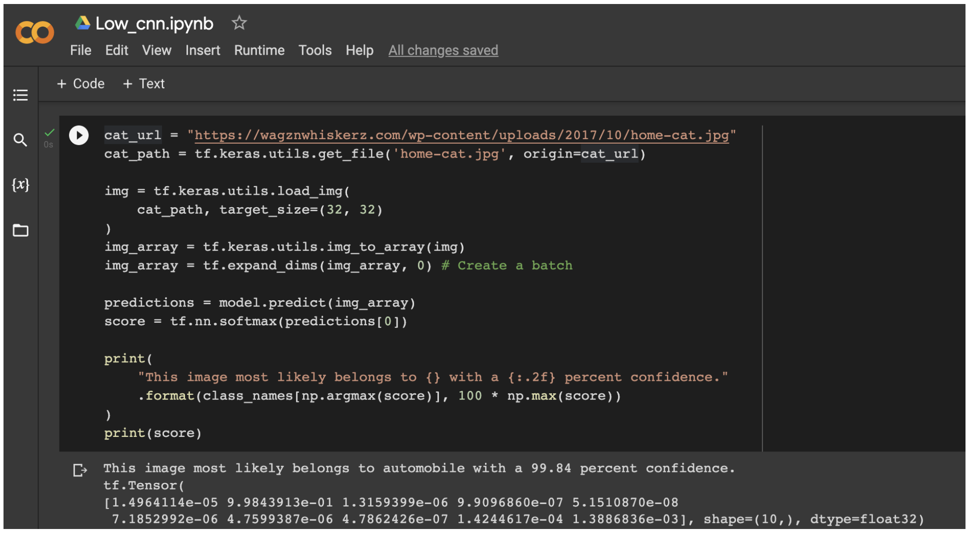


```
 8.7918352e-06 5.1984143e-06 5.6935158e-07 1.4943945e-04 1.5116582e-03], shape=(10,), dtype=float32)
```

```
birdtwo_url = "https://images.all-free-download.com/images/graphiclarge/flying_bird_201952.jpg"
birdtwo_path = tf.keras.utils.get_file('flying_bird_201952.jpg', origin=birdtwo_url)

img = tf.keras.utils.load_img(
    birdtwo_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))
)
print(score)
```

```
This image most likely belongs to ship with a 99.83 percent confidence.
tf.Tensor(
[7.8882376e-04 5.1609770e-04 2.8368187e-05 6.3221087e-05 2.6592275e-05
 7.6668475e-06 5.1191804e-05 8.4644571e-06 9.9832994e-01 1.7964470e-04], shape=(10,), dtype=float32)
```

5) The fifth image was classified as an automobile but it was a cat image



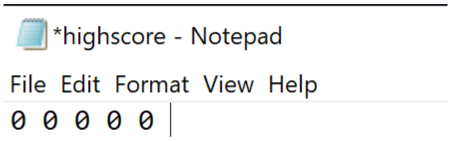
```
cat_url = "https://wagznwhiskerz.com/wp-content/uploads/2017/10/home-cat.jpg"
cat_path = tf.keras.utils.get_file('home-cat.jpg', origin=cat_url)

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))
)
print(score)
```

```
This image most likely belongs to automobile with a 99.84 percent confidence.
tf.Tensor(
[1.4964114e-05 9.9843913e-01 1.3159399e-06 9.9096860e-07 5.1510870e-08
 7.1852992e-06 4.7599387e-06 4.7862426e-07 1.4244617e-04 1.3886836e-03], shape=(10,), dtype=float32)
```

Balloon Flight

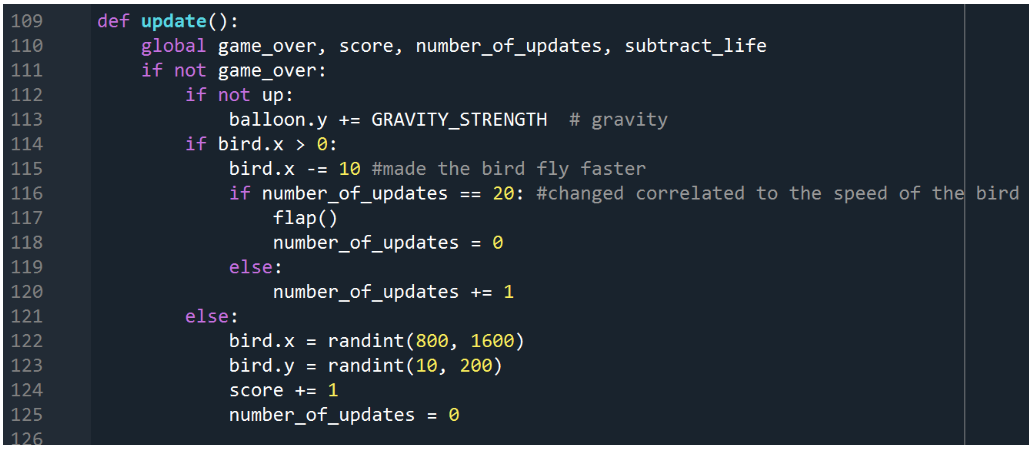
The goal of this lab is to incorporate four minor tweaks of the developer's choice. The first tweak that was made was adding more lines for the high score. The second tweak that was made was to speed up the bird to fly faster across the screen. The third modification made was to add another actor to the game to make it harder for the user to score. The fourth adjustment that was made was to create a different way to score so that when the hot air balloon passes an object then the score will change accordingly.

1) Changing the high score to prompt more slots was completed by adding more to the text file created for the high score to prompt.



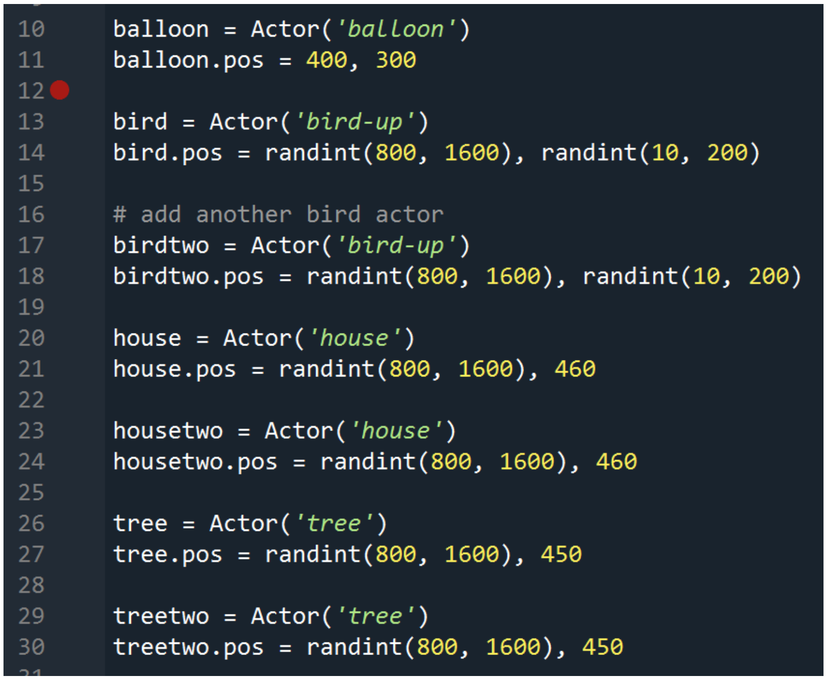
```
0 0 0 0 0
```

2) To make the bird fly across the screen faster, the number has to increase at Line in the def update structure.



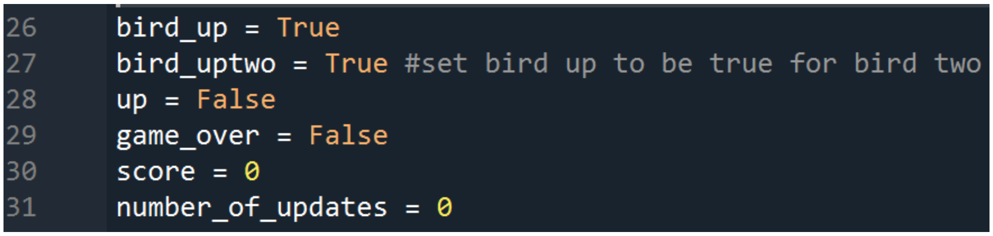
```
def update():
    global game_over, score, number_of_updates, subtract_life
    if not game_over:
        if not up:
            balloon.y += GRAVITY_STRENGTH  # gravity
        if bird.x > 0:
            bird.x -= 10 #made the bird fly faster
            if number_of_updates == 20: #changed correlated to the speed of the bird
                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
```

3) Add new actors for the tree, house and the bird so they can be called throughout the code



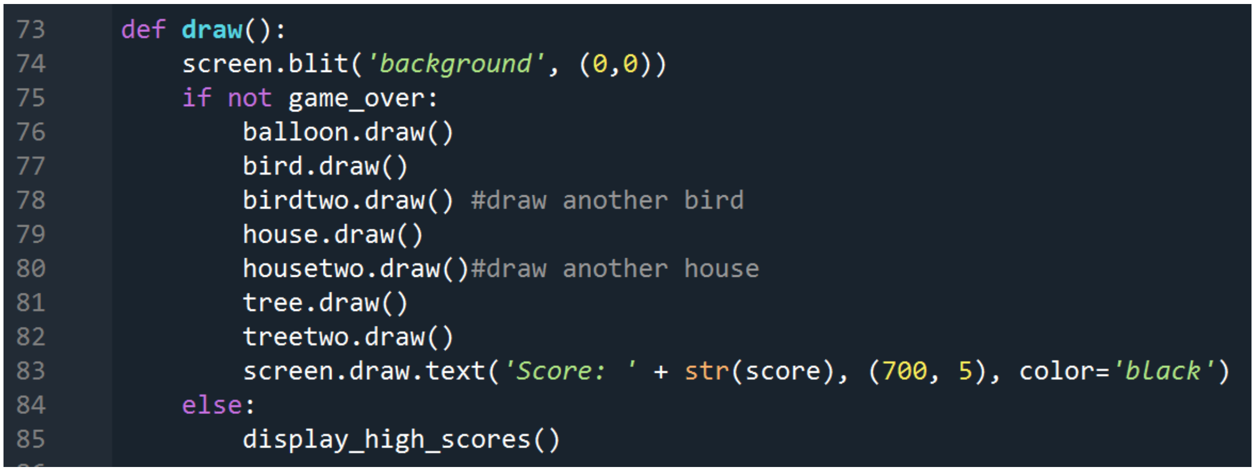
```
balloon = Actor('balloon')
balloon.pos = 400, 300

bird = Actor('bird-up')
bird.pos = randint(800, 1600), randint(10, 200)

# add another bird actor
birdtwo = Actor('bird-up')
birdtwo.pos = randint(800, 1600), randint(10, 200)

house = Actor('house')
house.pos = randint(800, 1600), 460

housetwo = Actor('house')
housetwo.pos = randint(800, 1600), 460

tree = Actor('tree')
tree.pos = randint(800, 1600), 450

treetwo = Actor('tree')
treetwo.pos = randint(800, 1600), 450
```

4) Make sure the set the bird up to be true for bird two so it can be called later on



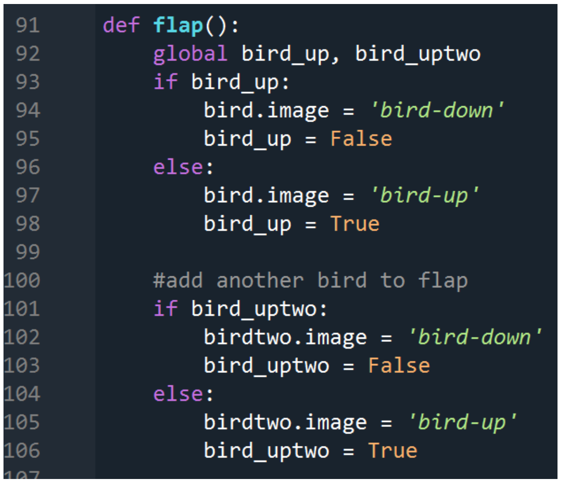
```
bird_up = True
bird_uptwo = True #set bird up to be true for bird two
up = False
game_over = False
score = 0
number_of_updates = 0
```

5) Make sure to draw the duplicated characters so it can be seen in the screen



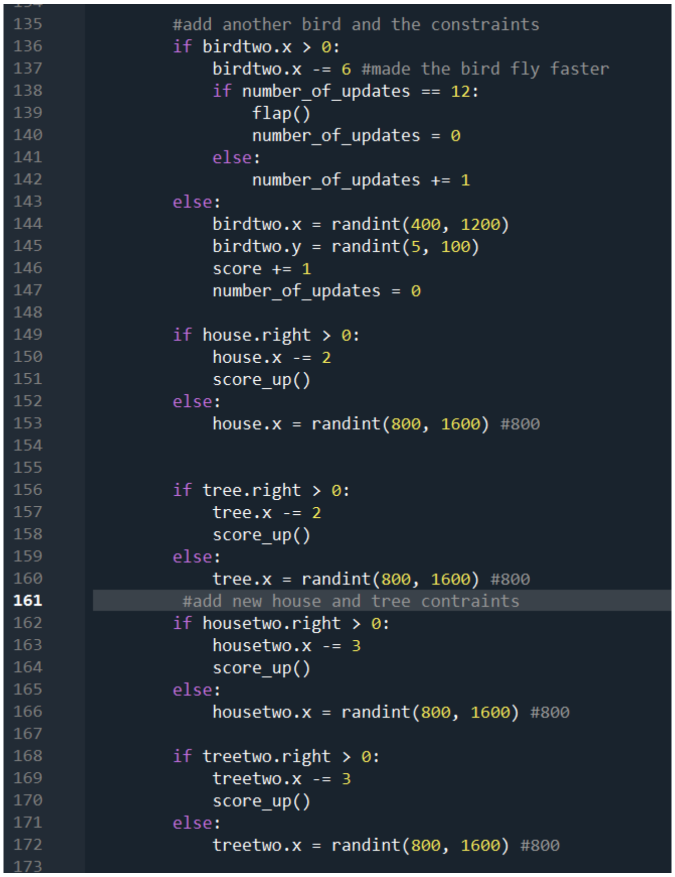
```
def draw():
    screen.blit('background', (0,0))
    if not game_over:
        balloon.draw()
        bird.draw()
        birdtwo.draw() #draw another bird
        house.draw()
        housetwo.draw()#draw another house
        tree.draw()
        treetwo.draw()
        screen.draw.text('Score: ' + str(score), (700, 5), color='black')
    else:
        display_high_scores()
```

6) The def flap includes the second bird so that it can flap



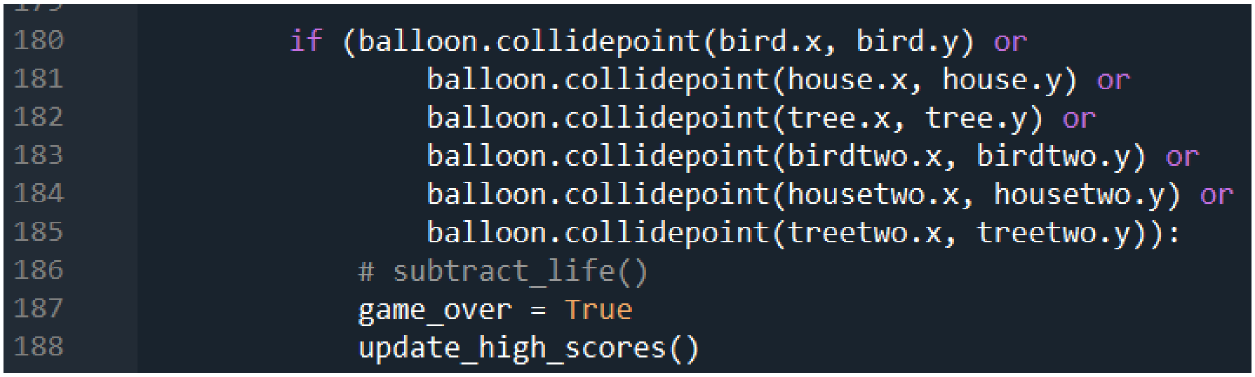
```
def flap():
    global bird_up, bird_uptwo
    if bird_up:
        bird.image = 'bird-down'
        bird_up = False
    else:
        bird.image = 'bird-up'
        bird_up = True

    #add another bird to flap
    if bird_uptwo:
        birdtwo.image = 'bird-down'
        bird_uptwo = False
    else:
        birdtwo.image = 'bird-up'
        bird_uptwo = True
```

7) Make sure there are constraints for the other obstacles with how they score.



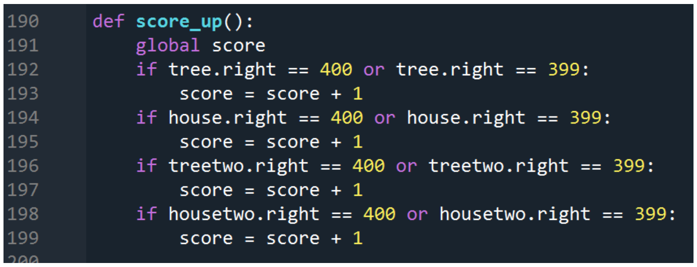
```
        #add another bird and the constraints
        if birdtwo.x > 0:
            birdtwo.x -= 6 #made the bird fly faster
            if number_of_updates == 12:
                flap()
                number_of_updates = 0
            else:
                number_of_updates += 1
        else:
            birdtwo.x = randint(400, 1200)
            birdtwo.y = randint(5, 100)
            score += 1
            number_of_updates = 0

        if house.right > 0:
            house.x -= 2
            score_up()
        else:
            house.x = randint(800, 1600) #800


        if tree.right > 0:
            tree.x -= 2
            score_up()
        else:
            tree.x = randint(800, 1600) #800
         #add new house and tree contraints
        if housetwo.right > 0:
            housetwo.x -= 3
            score_up()
        else:
            housetwo.x = randint(800, 1600) #800

        if treetwo.right > 0:
            treetwo.x -= 3
            score_up()
        else:
            treetwo.x = randint(800, 1600) #800
```

8) Add new collide points for the new obstacles added



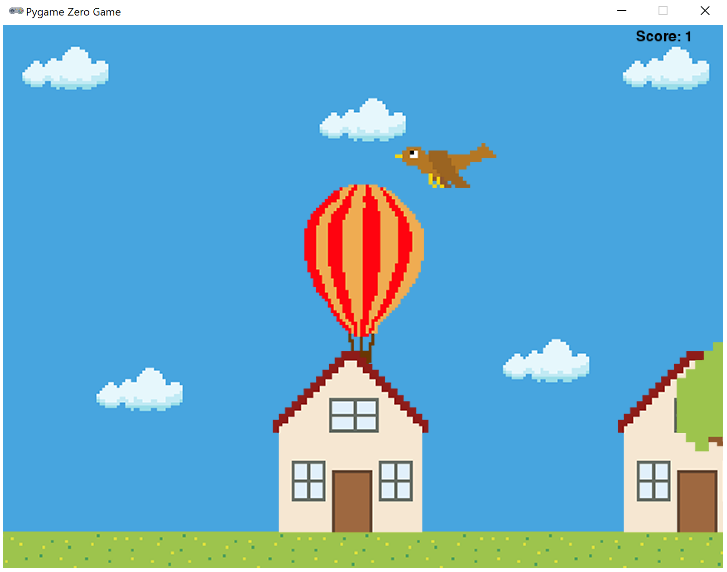
```
        if (balloon.collidepoint(bird.x, bird.y) or
                balloon.collidepoint(house.x, house.y) or
                balloon.collidepoint(tree.x, tree.y) or
                balloon.collidepoint(birdtwo.x, birdtwo.y) or
                balloon.collidepoint(housetwo.x, housetwo.y) or
                balloon.collidepoint(treetwo.x, treetwo.y)):
            # subtract_life()
            game_over = True
            update_high_scores()
```

9) Adding a new def structure for score_up for every time the balloon passes the tree and house the points will add to the score instead of when the objects move out of the screen.

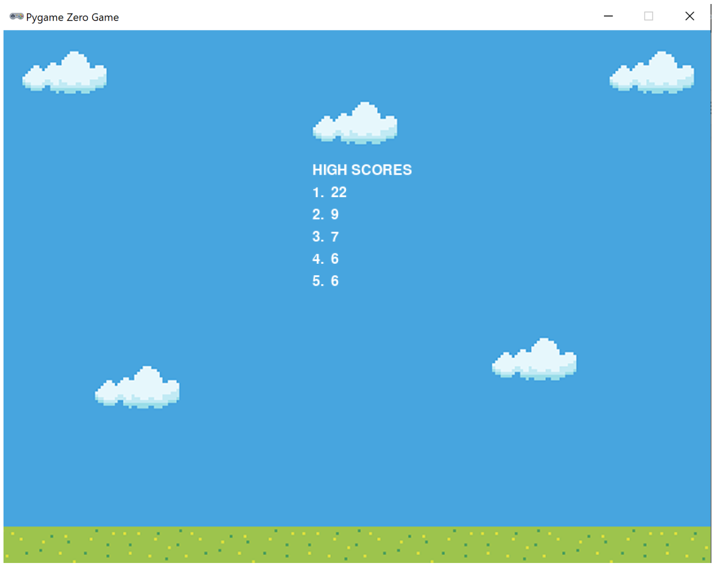


```
def score_up():
    global score
    if tree.right == 400 or tree.right == 399:
        score = score + 1
    if house.right == 400 or house.right == 399:
        score = score + 1
    if treetwo.right == 400 or treetwo.right == 399:
        score = score + 1
    if housetwo.right == 400 or housetwo.right == 399:
        score = score + 1
```

10) Results after the modifications



Score: 1



```
HIGH SCORES
1. 22
2. 9
3. 7
4. 6
5. 6
```