

T'che Caver

Professor Christopher Pham

EE104, Sec. 01

7 May 2022

## Lab 8: Documentation

GitHub Link: [https://github.com/tlcaver/lab8\\_caver\\_tche](https://github.com/tlcaver/lab8_caver_tche)

### Convolutional Neural Network (CNN):

To train our CNN, we started by installing the required dependencies. We installed TensorFlow, Keras, h5py, Matplotlib, and Numpy.

```
!pip install tensorflow
Requirement already satisfied: tensorflow in /usr/local/lib/python3.7/dist-packages (2.8.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: gast>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.5.3)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.21.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.14.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: absl-py>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.0.0)
Requirement already satisfied: libclang>=9.0.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (14.0.1)
Requirement already satisfied: tensorboard>2.9.>=2.8 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.8.0)
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (4.1.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from tensorflow) (57.4.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: flatbuffers>=1.12 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.25.0)
Requirement already satisfied: keras>2.9.>=2.8.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.8.0)
Requirement already satisfied: grpcio>=0.1.28.3 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.44.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.17.3)
Requirement already satisfied: tensorflow>=1.14.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.14.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.15.0)
Requirement already satisfied: keras-preprocessing>1.1.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.1.2)
Collecting tf-estimator-nightly==2.8.0.dev202112109-py2.py3-none-any.whl (462 kB)
  Downloading tf-estimator-nightly==2.8.0.dev202112109-py2.py3-none-any.whl (462 kB)
409 kB 4.5 MB/s
```

```
[ ] !pip install keras
!pip install h5py
!pip install matplotlib
!pip install numpy
Requirement already satisfied: keras in /usr/local/lib/python3.7/dist-packages (2.8.0)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py) (1.5.2)
Requirement already satisfied: numpy>=1.14.2 in /usr/local/lib/python3.7/dist-packages (from h5py) (1.21.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (3.2.2)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (1.21.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (0.11.0)
Requirement already satisfied: pyparsing>=2.0.4,<=1.2,>=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (3.0.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (1.4.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from kiwisolver>=1.0.1->matplotlib) (4.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.1->matplotlib) (1.15.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (1.21.0)
```

Then, we imported the required packages from each of the dependencies we just installed.

```
[ ] 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, we downloaded the image training data, and normalized the pixels. This means that

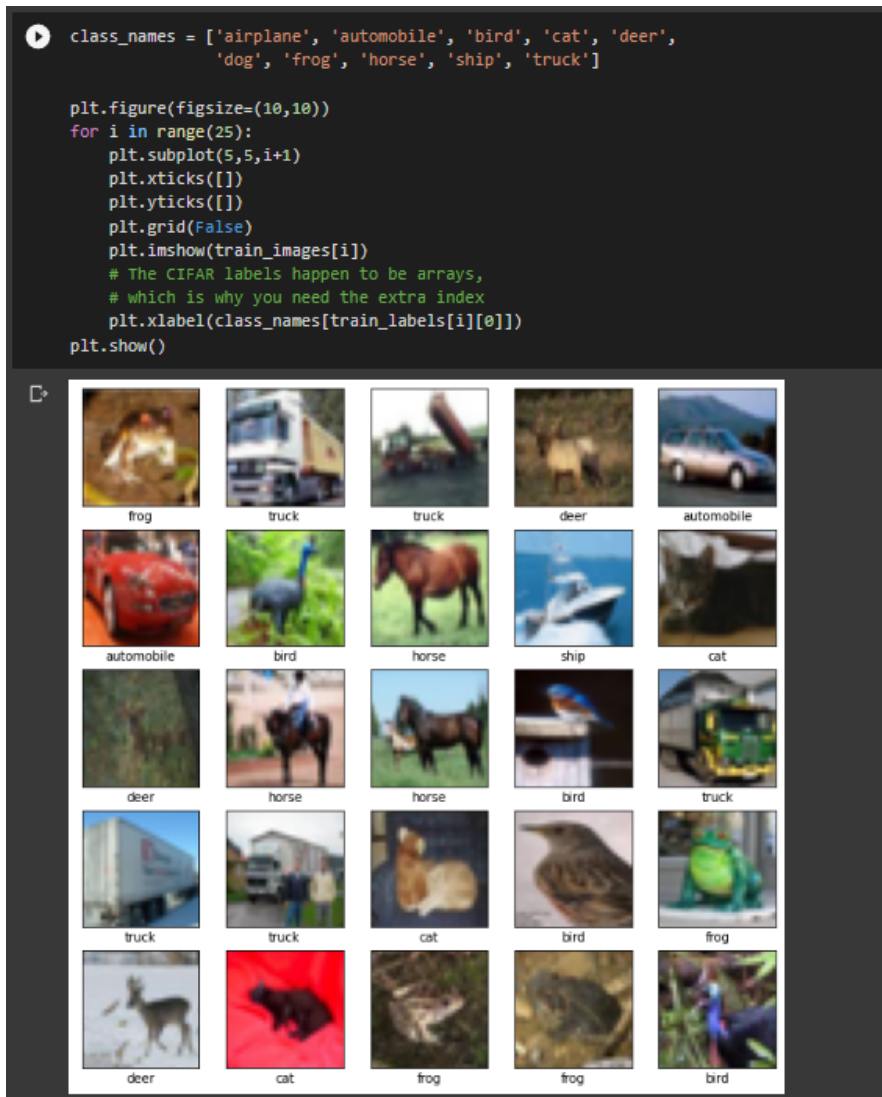
instead of being on a scale from 0 to 255, we divided the pixels by 255 to put them on a scale of 0 to 1.

```
[ ] [(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 [=====] - 2s 0us/step
```

We then displayed all of the classes, as well as an image of each that was labeled.



Now, we can augment the data. This slightly modifies the original pictures to look similar, but with different features. This will allow the neural network to train with a greater variety of variations of each image.

```
[ ] 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)
```

Now, we design our neural network model. To increase accuracy, we added far more convolutional layers than the example model included. They can be seen in the image below. This will take longer to train, but in exchange, will provide better results. We also implement dropout, to force the network to find new paths when training that will increase accuracy.

```
# define cnn model

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'))

# compile model
```

After flattening and performing some other transformations, we are ready to train the model.

```
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'])

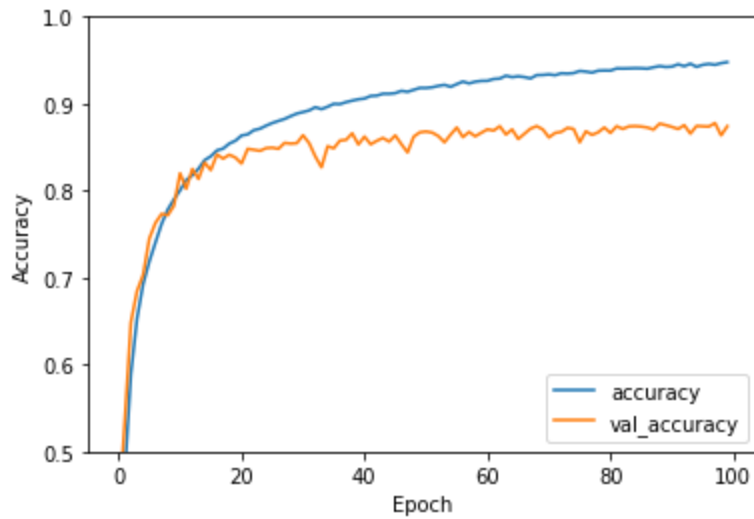
history = model.fit(train_images, train_labels, epochs=100, batch_size=64,
                    validation_data=(test_images, test_labels))
```

We set the epochs to 100, rather than 10, to train the model more extensively and attain better results. Again, this is more intensive and takes longer than training with only 10 epochs.

```
782/782 [=====] - 21s 27ms/step - loss: 0.2092 - accuracy: 0.9297 - val_loss: 0.4677 - val_accuracy: 0.8627
Epoch 68/100
...
782/782 [=====] - 21s 27ms/step - loss: 0.2027 - accuracy: 0.9316 - val_loss: 0.4672 - val_accuracy: 0.8662
Epoch 69/100
782/782 [=====] - 21s 27ms/step - loss: 0.2096 - accuracy: 0.9298 - val_loss: 0.4634 - val_accuracy: 0.8630
Epoch 70/100
782/782 [=====] - 21s 27ms/step - loss: 0.2014 - accuracy: 0.9320 - val_loss: 0.4687 - val_accuracy: 0.8655
Epoch 71/100
782/782 [=====] - 21s 27ms/step - loss: 0.1970 - accuracy: 0.9337 - val_loss: 0.4976 - val_accuracy: 0.8628
Epoch 72/100
782/782 [=====] - 21s 27ms/step - loss: 0.2007 - accuracy: 0.9311 - val_loss: 0.4844 - val_accuracy: 0.8677
Epoch 73/100
782/782 [=====] - 21s 27ms/step - loss: 0.1936 - accuracy: 0.9332 - val_loss: 0.4720 - val_accuracy: 0.8703
Epoch 74/100
782/782 [=====] - 21s 27ms/step - loss: 0.1938 - accuracy: 0.9348 - val_loss: 0.4755 - val_accuracy: 0.8628
Epoch 75/100
782/782 [=====] - 21s 27ms/step - loss: 0.1957 - accuracy: 0.9339 - val_loss: 0.4787 - val_accuracy: 0.8663
Epoch 76/100
782/782 [=====] - 21s 27ms/step - loss: 0.1931 - accuracy: 0.9364 - val_loss: 0.4851 - val_accuracy: 0.8646
Epoch 77/100
782/782 [=====] - 21s 27ms/step - loss: 0.1883 - accuracy: 0.9365 - val_loss: 0.4887 - val_accuracy: 0.8642
Epoch 78/100
782/782 [=====] - 21s 27ms/step - loss: 0.1896 - accuracy: 0.9364 - val_loss: 0.5104 - val_accuracy: 0.8632
Epoch 79/100
782/782 [=====] - 21s 27ms/step - loss: 0.1871 - accuracy: 0.9365 - val_loss: 0.4924 - val_accuracy: 0.8624
Epoch 80/100
782/782 [=====] - 22s 28ms/step - loss: 0.1879 - accuracy: 0.9355 - val_loss: 0.4765 - val_accuracy: 0.8669
Epoch 81/100
782/782 [=====] - 22s 28ms/step - loss: 0.1868 - accuracy: 0.9362 - val_loss: 0.4819 - val_accuracy: 0.8649
Epoch 82/100
782/782 [=====] - 22s 28ms/step - loss: 0.1796 - accuracy: 0.9397 - val_loss: 0.4717 - val_accuracy: 0.8653
Epoch 83/100
782/782 [=====] - 22s 28ms/step - loss: 0.1832 - accuracy: 0.9382 - val_loss: 0.5044 - val_accuracy: 0.8604
Epoch 84/100
782/782 [=====] - 22s 28ms/step - loss: 0.1818 - accuracy: 0.9379 - val_loss: 0.4886 - val_accuracy: 0.8676
Epoch 85/100
782/782 [=====] - 22s 28ms/step - loss: 0.1782 - accuracy: 0.9405 - val_loss: 0.5055 - val_accuracy: 0.8648
Epoch 86/100
782/782 [=====] - 22s 28ms/step - loss: 0.1773 - accuracy: 0.9416 - val_loss: 0.4721 - val_accuracy: 0.8686
Epoch 87/100
5/782 [.....] - ETA: 20s - loss: 0.1240 - accuracy: 0.9594
```

We then can generate a plot of the accuracy, and print out the final values.

Our accuracy plot can be seen in detail below.



The first training, I was not able to record a video. Therefore, I re-ran the code, and the values of the training recorded are slightly different than the original ones shown here.

We then tested the accuracy by recognizing unrecognizable images.

The images did have to be slightly rescaled before being fed into the CNN.

In this case, we can see that the airplane was recognized as an airplane.

```

airplane_url = "https://www.zdnet.com/a/img/resize/071727877ee9884b60edd728253d2baadcb3985f/2021/02/23/19631992-64df-4af9-a288-a8cb41126602/bombardier-globaleye-jet.jpg?width=1200&height=900&fit=crop&auto=webp"
airplane_path = tf.keras.utils.get_file('Airplane', origin=airplane_url)

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

width, height = img.size

img1 = img.crop((223, 0, 799, 575))
img2 = img1.resize((32, 32))

img_array = tf.keras.utils.img_to_array(img2)
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 12.42 percent confidence.

In the next case, we can see that the bird is recognized as a bird.

```

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= (1200, 900)
)

width, height = img.size


img1 = img.crop((149, 0, 1049, 900 ))
img2 = img1.resize((32, 32))

img_array = tf.keras.utils.img_to_array(img2)
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 bird with a 23.19 percent confidence.

The next one, a car, was recognized to be an automobile.

```

[ ] 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= (512, 349)
)

width, height = img.size

img2 = img.resize((32, 32))

img_array = tf.keras.utils.img_to_array(img2)
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.

I recognized one more automobile as well.

```

▶ automobile_url = "https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/devel-motors-sixteen-1540564064.jpg"
automobile_path = tf.keras.utils.get_file('Automobile', origin=automobile_url)

img = tf.keras.utils.load_img(
    automobile_path, target_size= (512, 349)
)

width, height = img.size

img2 = img.resize((32, 32))

img_array = tf.keras.utils.img_to_array(img2)
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.
```

Finally, a cat was recognized, however the model was slightly confused due to the accuracy.

```

[ ] 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= (512, 349)
)

width, height = img.size

img2 = img.resize((32, 32))

img_array = tf.keras.utils.img_to_array(img2)
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 bird with a 23.19 percent confidence.
```

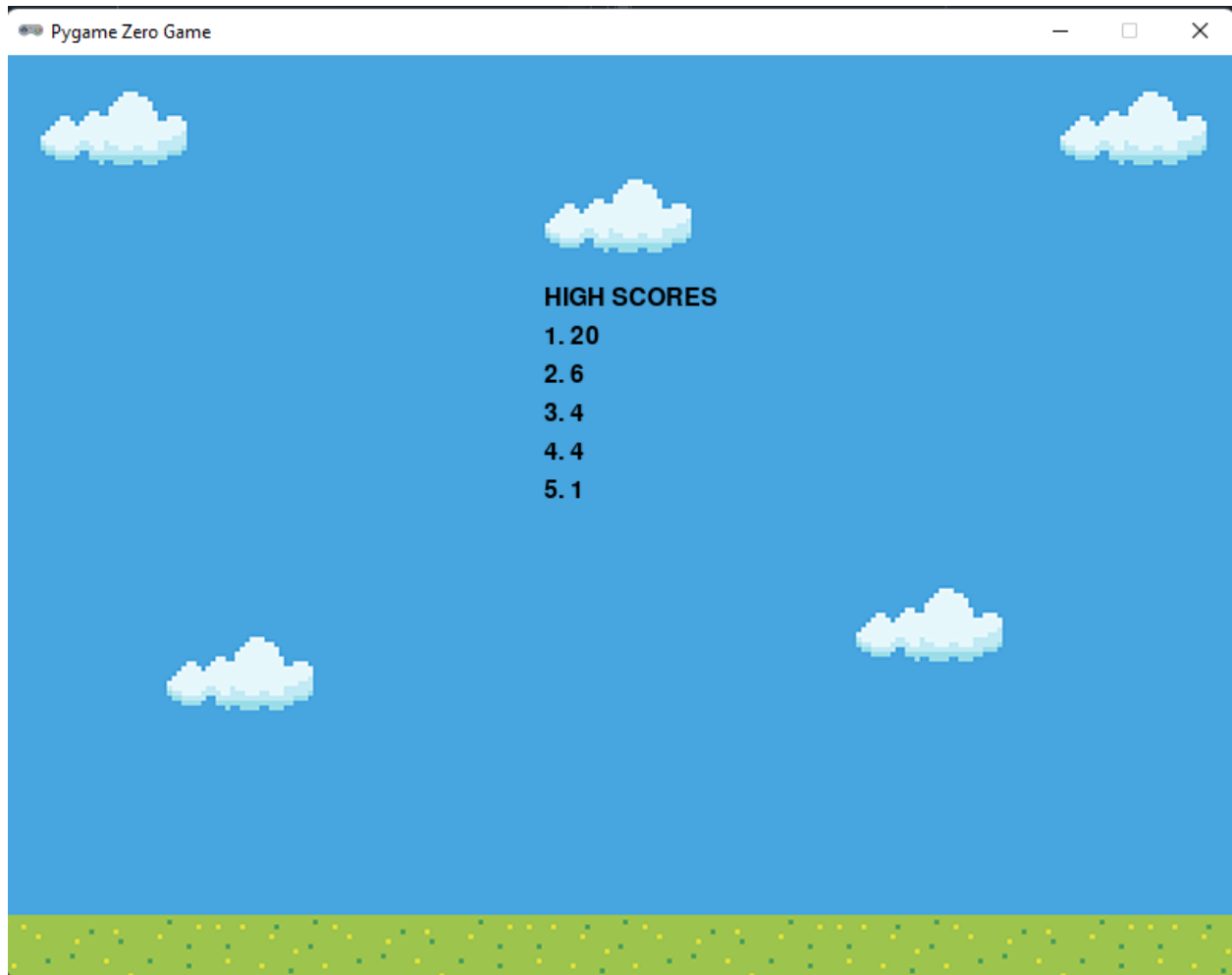
## **Game Development- Balloon Flight:**

For this portion of the lab, we implemented four different tweaks: More High Scores, Speed it Up, Level Up, and Space Out the Obstacles.

### **More High Scores:**

Looking at the code, the Python program uses the number of digits in the .txt file to determine how many high scores there are and print the scores out. To increase this, we can simply add more digits to the .txt file, and the number of high scores will increase as well as printing out all added high scores. In this case, we wanted to make the number of high scores saved 5. So we added two zeros to the text file, to have a total of 5 zeros and therefore 5 scores.





### Speed It Up:

We accomplished this by changing the number of pixels that each object moves for each loop iteration. The original code moves the bird 4 pixels, and the house and tree both move at 2 pixels. We changed the bird to move 6 pixels, and the house and tree move 3 pixels to keep the ratio the same. These changes are shown in the code below.

```

def update():
    global game_over, score, number_of_updates
    global speed, updatespeed

    if updatespeed != 1:
        if score != 0:
            if score % 10 == 0:
                speed = speed + 1
                updatespeed = 1
            if score % 10 != 0:
                updatespeed = 0

    if tree.x == house.x:
        tree.x = tree.x + 100;

    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 -= 3 * speed
        else:
            house.x = randint(800, 1600)
            score += 1
        if tree.right > 0:
            tree.x -= 3 * 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()

pgzrun.go()

```

Level Up:

For this portion of the lab, we added two new variables, one called speed, and one called updatespeed. The speed variable is a multiplier to the number of pixels each object moves per loop. Increasing this will cause the objects to move more pixels every loop, therefore speeding the game up. Each time the game score increases by a multiple of 10, and the score isn't 0, 1 is

added to the speed. The updatespeed variable is simply a flag to keep track of whether or not the speed has already updated for a certain score. Once the speed is updated on a multiple of 10, the flag goes to 1 so that the speed doesn't update again. Once the score goes to the next number, the flag resets to 0.

```
def update():
    global game_over, score, number_of_updates
    global speed, updatespeed

    if updatespeed != 1:
        if score != 0:
            if score % 10 == 0:
                speed = speed + 1
                updatespeed = 1
    if score % 10 != 0:
        updatespeed = 0
```

### Space Out the Obstacles:

This portion of the code is quite simple. This checks to see whether or not the house and tree are in the same spot. If so, it spaces them out by 100 pixels. In this case, since the objects are moving to the left (therefore subtracting pixels), adding 100 would put the tree behind the house by 100 pixels.

```
if tree.x == house.x:
    tree.x = tree.x + 100;
```