Lab 8 Documentation: Convolutional Neural Network

Author: Jacqueline Arce

EE 104: Professor Pham

Overview:

In this lab students will use and modify a Convolutional Neural Network to achieve 90% accuracy or above. The same dataset will be used for a challenge in which students need to improve the base model to recognize at most 5 unrecognizable images. The final task is to add four tweeks to a ballon flight pygame.

Acknowledgment:

I would like to acknowledge San Jose State University, specifically Professor Pham as the basis of the code was provided in the class

CNN:

The base code for this CNN is given with a working code that produces over 70% accuracy. The assignment is to modify the code to improve the val_accuracy. The following imports are used where several needed to be added for the modifications that follow.

```
[ ]  import tensorflow as tf

     from tensorflow.keras import datasets, layers, models
     import matplotlib.pyplot as plt
     from keras.layers import Dropout, BatchNormalization, Flatten, Dense, Conv2D, MaxPooling2D
```

The cell below is where most of the changes were made. Here is the convolitional base. To improve the accuracy, BatchNormalization, MaxPooling, Dropout, and Cov2D were used. All combined the model was improved. The dropout is the most sensitive as 30% worked the best and kept consistent results.

```
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(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(0.3))
```

The dense layer is then modified with a flatten, dense, batchnormalization, and dropout. This portion completes the model

```
[ ]  model.add(Flatten())
     model.add(Dense(256, activation='relu', kernel_initializer='he_uniform'))
     model.add(BatchNormalization())
     model.add(Dropout(0.3))
     model.add(Dense(10, activation='softmax'))
```

The model is compiled and trained below using 45 epochs. The capture shows the fist 6 epochs executed followed by the last epoch showing the highest val_accuracy able tp be achieved at 87%.

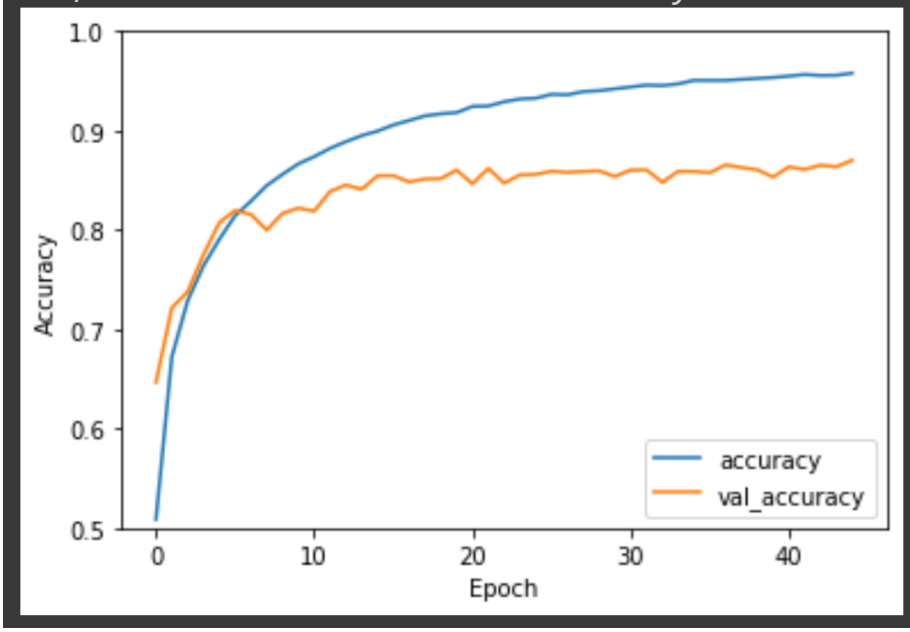
```
[ ]  model.compile(optimizer='adam',
                   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                   metrics=['accuracy'])

     history = model.fit(train_images, train_labels, epochs=45,
                         validation_data=(test_images, test_labels))
```

```
Epoch 1/45
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1082: UserWarning: "`sparse_categorical_crossentropy` receive
  return dispatch_target(*args, **kwargs)
1563/1563 [==============================] - 17s 8ms/step - loss: 1.4032 - accuracy: 0.5087 - val_loss: 0.9898 - val_accuracy: 0.6466
Epoch 2/45
1563/1563 [==============================] - 12s 7ms/step - loss: 0.9347 - accuracy: 0.6721 - val_loss: 0.8001 - val_accuracy: 0.7216
Epoch 3/45
1563/1563 [==============================] - 12s 8ms/step - loss: 0.7777 - accuracy: 0.7287 - val_loss: 0.7651 - val_accuracy: 0.7377
Epoch 4/45
1563/1563 [==============================] - 12s 8ms/step - loss: 0.6829 - accuracy: 0.7639 - val_loss: 0.6464 - val_accuracy: 0.7751
Epoch 5/45
1563/1563 [==============================] - 12s 8ms/step - loss: 0.6081 - accuracy: 0.7905 - val_loss: 0.5559 - val_accuracy: 0.8073
Epoch 6/45
1563/1563 [==============================] - 13s 8ms/step - loss: 0.5375 - accuracy: 0.8144 - val_loss: 0.5351 - val_accuracy: 0.8196
```

```
Epoch 45/45
1563/1563 [==============================] - 12s 8ms/step - loss: 0.1194 - accuracy: 0.9578 - val_loss: 0.5020 - val_accuracy: 0.8702
```

A visual of the data is shown below along with the accuracy printed out achieving at most an 87%

```
313/313 - 1s - loss: 0.5020 - accuracy: 0.8702 - 1s/epoch - 3ms/step
```




```
[ ]  print(test_acc)

     0.870199978351593
```

CNN Challenge:

The Same baseline code above is used to keep the same dataset. This dataset classifies ariplanes, automobiles, birds, cats, deers, dogs, frogs, horses, ships, and trucks.

The following cell is an example of the added cells at the end of the previous modified code to output the classification. The first line is the URL to the unrecognized images provided by the professor. The path uses the image name assigning it to the url. The image is then run through a target size of 32x32.

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

When running the code, all cells are executed one after another. The results are shown below where my modified code is able to recognize 6 of the 10 unrecognizable images. It was able to recognize one airplane, all three of the automobiles, one bird, and one cat.

```
[14] airplane_url = "https://www.zdnet.com/a/img/resize/071727877ee9884b60edd728253d2baadcb3985f/2021/0
     airplane_path = tf.keras.utils.get_file('bombardier-globaleye-jet', 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 23.20 percent confidence.
```

Automobile

```
[15] automobile_url = "https://images.all-free-download.com/images/graphiclarge/classic_jaguar_210354.jpg"
     automobile_path = tf.keras.utils.get_file('classic_jaguar_210354', 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))
     )
```

```
This image most likely belongs to automobile with a 23.20 percent confidence.
```

```
[16] 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', 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))
     )
```

```
This image most likely belongs to automobile with a 23.20 percent confidence.
```

```
[17] automobile_url = "https://amsc-prod-cd.azureedge.net/-/media/aston-martin/images/default-source/
     automobile_path = tf.keras.utils.get_file('valkyrie-spider_f02-169v2', 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))
     )
```

```
This image most likely belongs to automobile with a 23.20 percent confidence.
```

Birds

```
[18] bird_url = "https://ichef.bbci.co.uk/news/976/cpsprodpb/67CF/production/_108857562_mediaitem108857561.jpg "
     bird_path = tf.keras.utils.get_file('_108857562_mediaitem108857561', 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))
     )

     This image most likely belongs to bird with a 23.20 percent confidence.
```

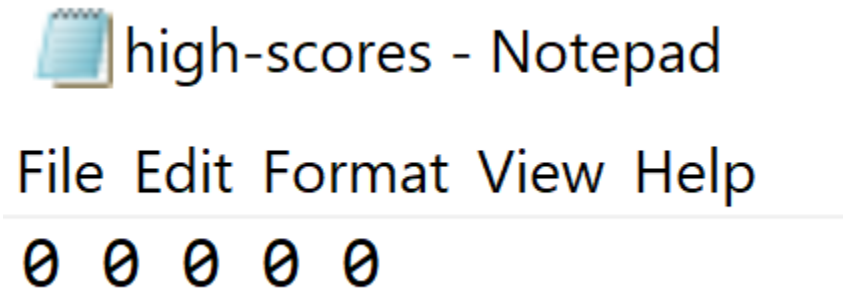
```
[22] cat_url = "https://static.toiimg.com/thumb/msid-67586673,width-1070,height-580,overlay-t
     cat_path = tf.keras.utils.get_file('67586673', origin=cat_url)

     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))
     )

     This image most likely belongs to cat with a 23.20 percent confidence.
```

If the accuracy in the first portion of the code was higher or at most past 90 when running this portion, I believe all images would have been recognized. Using more epochs when running the code may have also helped but because running several epochs takes longer it was more convenient to run with fewer to get over 4 recognitions. The confidence percentage all seem to be the same which leads me to believe there is some issue that could be explored for the root cause and fixed if time permitted.

Balloon Flight:

The four modifications made for the balloon flight game are more high scores, speed it up, different ways to score, and multiples of each obstacles.

Starting with more highscores, this was done by adding two more additional zeros in the notepad that stores the high scores. This allows the code to place for the 4th and 5th place scores.



Next, the speed it up was performed on one of the birds. This is done to have a faster bird going at a speed of 5. The speed is determined by how many placements it skips for the next frame.

```
126         if bird.x > 0:
127             bird.x -= 5
128             if number_of_updates == 9:
129                 flap()
130                 number_of_updates = 0
131             else:
132                 number_of_updates += 1
```

In different ways to score, the score goes up when the balloon passes a house. This is done by creating a new function to be called on to increase the score. Because the ballon is always at 400 in the x direction, at any point the left side of the house crosses the 400 mark, it would increase the score by one.

```
115 def add_score():
116     global score
117     if house.left == 400 or house2.left == 400:
118         score += 1
```

```
151         if house.right > 0:
152             house.x -= 2
153             add_score()
154         else:
155             house.x = randint(800, 1600)
156 
157         if house2.right > 0:
158             house2.x -= 2
159             add_score()
160         else:
161             house2.x = randint(800, 1600)
```

Lastly, multiples of each obstacles were added. The birds, the house, and the tree were all duplicated. To keep the code still working every every instance of the actors needed to be diblicated. The snapshots below show every section a second actor is added.

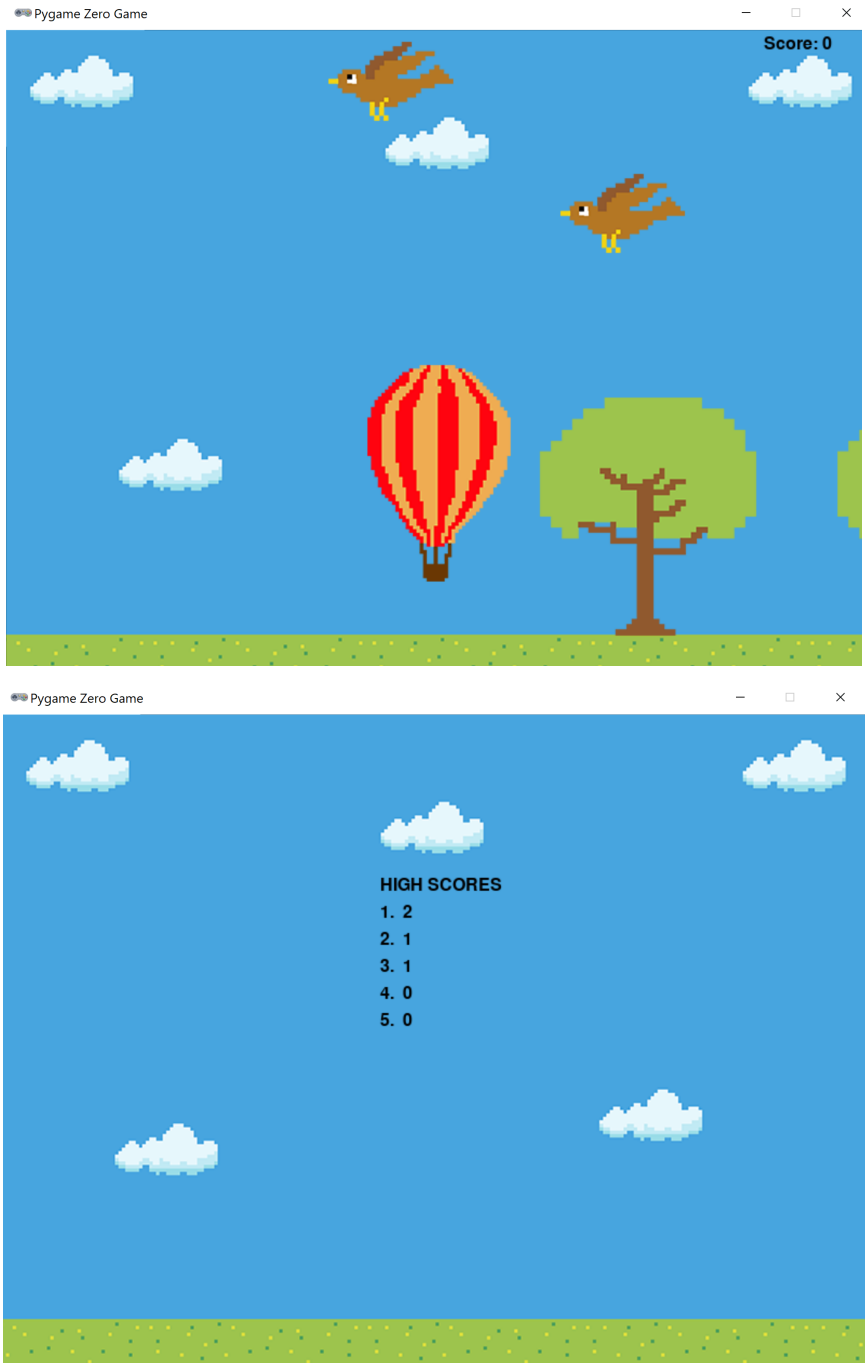
```
15  balloon = Actor('balloon')
16  balloon.pos = 400, 300
17  
18  bird = Actor('bird-up')
19  bird.pos = randint(800, 1600), randint(10, 200)
20  
21  bird2 = Actor('bird-up')
22  bird2.pos = randint(800, 1600), randint(10, 200)
23  
24  house = Actor('house')
25  house.pos = randint(800, 1600), 460
26  
27  house2 = Actor('house')
28  house2.pos = randint(800, 1600), 460
29  
30  tree = Actor('tree')
31  tree.pos = randint(800, 1600), 450
32  
33  tree2 = Actor('tree')
34  tree2.pos = randint(800, 1600), 450
35  
36  bird_up = True
37  bird2_up = True
```

```
75  def draw():
76      screen.blit('background', (0,0))
77      if not game_over:
78          balloon.draw()
79          bird.draw()
80          bird2.draw()
81          house.draw()
82          house2.draw()
83          tree.draw()
84          tree2.draw()
85          screen.draw.text('Score: ' + str(score), (700, 5), color='black')
86      else:
87          display_high_scores()
```

```
101 def flap():
102     global bird_up, bird2_up
103     if bird_up:
104         bird.image = 'bird-down'
105         bird_up = False
106     if bird2_up:
107         bird2.image = 'bird-down'
108         bird2_up = False
109     else:
110         bird.image = 'bird-up'
111         bird_up = True
112         bird2.image = 'bird-up'
113         bird2_up = True
```

```
def update():
    global game_over, score, number_of_updates
    if not game_over:
        if not up:
            balloon.y += GRAVITY_STRENGTH  # gravity
        if bird.x > 0:
            bird.x -= 5
            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 bird2.x > 0:
            bird2.x -= 3
            if number_of_updates == 9:
                flap()
                number_of_updates = 0
            else:
                number_of_updates += 1
        else:
            bird2.x = randint(800, 1600)
            bird2.y = randint(10, 200)
            number_of_updates = 0

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

        if house2.right > 0:
            house2.x -= 2
            add_score()
        else:
            house2.x = randint(800, 1600)

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

        if tree2.right > 0:
            tree2.x -= 2
        else:
            tree2.x = randint(800, 1600)

        if balloon.bottom < 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, tree.y) or
                balloon.collidepoint(bird2.x, bird2.y)):
            #game_over = True
            update_high_scores()
```

The snapshot below shows the game view followed by the end game score board.



https://github.com/jackiearce/EE104Lab8.git