In [1]:
```
import tensorflow as tf
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, models
from sklearn.metrics import confusion_matrix
import pandas as pd
import cv2
import os
```

Loading in the data

In [2]:
```
# Important information
image_sizes = (256,256)
base_path = "C:\\School\\EECS 605\\Project\\Data\\"
image_paths = [base_path + "Ones\\Right\\", base_path + "Ones\\Left\\",
               base_path + "Twos\\Right\\", base_path + "Twos\\Left\\",
               base_path + "Threes\\Right\\", base_path + "Threes\\Left\\",
               base_path + "Fours\\Right\\", base_path + "Fours\\Left\\",
               base_path + "Fives\\Right\\", base_path + "Fives\\Left\\",]
num_images = []
for i in range(int(len(image_paths))):
    num_images.append(len(os.listdir(image_paths[i][:-1])))
```

In [3]:
```
print(f"The total number of train/valid images is {sum(num_images)}")
```

```
The total number of train/valid images is 3065
```

In [4]:
```
# Reading in data and creating dataset
X = np.zeros((sum(num_images), image_sizes[0], image_sizes[1]))
Y = np.zeros(sum(num_images))
summed = 0
for i in range(int(len(image_paths))):
    for j in range(num_images[i]):
        X[summed + j, : :] = np.asarray(Image.open(image_paths[i] + f"image{j}.jpg")) / 255
        Y[summed + j] = i // 2
    summed += num_images[i]
X_train, X_valid, y_train, y_valid = train_test_split(np.expand_dims(X, axis = 3),
                                                      Y, test_size=0.2, random_state=42)
```

Training the model

In [106]:
```
# Deep Learning Model
model = models.Sequential()
model.add(layers.Conv2D(16, (3, 3), activation='relu', input_shape=(256, 256, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(16, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(5, activation='softmax'))
```

In [107]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 254, 254, 16)      160
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 127, 127, 16)      0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 125, 125, 16)      2320
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 62, 62, 16)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 60, 60, 32)        4640
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 30, 30, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 28, 28, 32)        9248
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               802944
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 5)                 645
=================================================================
Total params: 819,957
Trainable params: 819,957
Non-trainable params: 0
_________________________________________________________________
```

In [108]:
```
early_stop = tf.keras.callbacks.EarlyStopping(monitor="val_loss",
                                              patience=10,
                                              restore_best_weights=True)
opt = tf.keras.optimizers.Adam(learning_rate=1e-3)
model.compile(optimizer = opt, loss = 'sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=100, batch_size=64,
                    validation_data=(X_valid, y_valid),
                    callbacks = [early_stop])
```

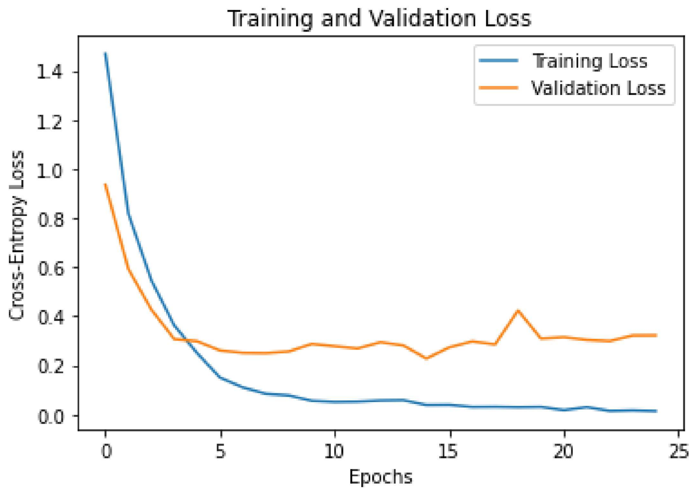
```
Train on 2452 samples, validate on 613 samples
Epoch 1/100
2452/2452 [==============================] - 36s 14ms/sample - loss: 1.4684 - accuracy: 0.3434 - va
l_loss: 0.9350 - val_accuracy: 0.6786
Epoch 2/100
2452/2452 [==============================] - 35s 14ms/sample - loss: 0.8185 - accuracy: 0.6794 - va
l_loss: 0.5925 - val_accuracy: 0.8042
Epoch 3/100
2452/2452 [==============================] - 36s 15ms/sample - loss: 0.5474 - accuracy: 0.8002 - va
l_loss: 0.4279 - val_accuracy: 0.8499
Epoch 4/100
2452/2452 [==============================] - 35s 14ms/sample - loss: 0.3630 - accuracy: 0.8670 - va
l_loss: 0.3071 - val_accuracy: 0.8940
Epoch 5/100
2452/2452 [==============================] - 36s 15ms/sample - loss: 0.2499 - accuracy: 0.9066 - va
l_loss: 0.2982 - val_accuracy: 0.8972
Epoch 6/100
2452/2452 [==============================] - 35s 14ms/sample - loss: 0.1502 - accuracy: 0.9462 - va
l_loss: 0.2604 - val_accuracy: 0.9184
Epoch 7/100
2452/2452 [==============================] - 37s 15ms/sample - loss: 0.1100 - accuracy: 0.9580 - va
l_loss: 0.2509 - val_accuracy: 0.9152
Epoch 8/100
2452/2452 [==============================] - 33s 14ms/sample - loss: 0.0843 - accuracy: 0.9723 - va
l_loss: 0.2498 - val_accuracy: 0.9250
Epoch 9/100
2452/2452 [==============================] - 34s 14ms/sample - loss: 0.0776 - accuracy: 0.9710 - va
l_loss: 0.2562 - val_accuracy: 0.9413
Epoch 10/100
2452/2452 [==============================] - 34s 14ms/sample - loss: 0.0557 - accuracy: 0.9763 - va
l_loss: 0.2874 - val_accuracy: 0.9331
Epoch 11/100
2452/2452 [==============================] - 33s 14ms/sample - loss: 0.0511 - accuracy: 0.9853 - va
l_loss: 0.2778 - val_accuracy: 0.9299
Epoch 12/100
2452/2452 [==============================] - 35s 14ms/sample - loss: 0.0523 - accuracy: 0.9816 - va
l_loss: 0.2687 - val_accuracy: 0.9331
Epoch 13/100
2452/2452 [==============================] - 37s 15ms/sample - loss: 0.0568 - accuracy: 0.9837 - va
l_loss: 0.2943 - val_accuracy: 0.9299
Epoch 14/100
2452/2452 [==============================] - 36s 15ms/sample - loss: 0.0580 - accuracy: 0.9808 - va
l_loss: 0.2810 - val_accuracy: 0.9364
Epoch 15/100
2452/2452 [==============================] - 33s 14ms/sample - loss: 0.0385 - accuracy: 0.9878 - va
l_loss: 0.2277 - val_accuracy: 0.9494
Epoch 16/100
2452/2452 [==============================] - 34s 14ms/sample - loss: 0.0391 - accuracy: 0.9865 - va
l_loss: 0.2732 - val_accuracy: 0.9511
Epoch 17/100
2452/2452 [==============================] - 33s 14ms/sample - loss: 0.0306 - accuracy: 0.9902 - va
l_loss: 0.2975 - val_accuracy: 0.9364
Epoch 18/100
2452/2452 [==============================] - 34s 14ms/sample - loss: 0.0312 - accuracy: 0.9886 - va
l_loss: 0.2853 - val_accuracy: 0.9511
Epoch 19/100
2452/2452 [==============================] - 33s 14ms/sample - loss: 0.0298 - accuracy: 0.9902 - va
l_loss: 0.4234 - val_accuracy: 0.9266
Epoch 20/100
2452/2452 [==============================] - 34s 14ms/sample - loss: 0.0308 - accuracy: 0.9894 - va
l_loss: 0.3091 - val_accuracy: 0.9299
Epoch 21/100
2452/2452 [==============================] - 33s 14ms/sample - loss: 0.0177 - accuracy: 0.9951 - va
l_loss: 0.3147 - val_accuracy: 0.9429
Epoch 22/100
2452/2452 [==============================] - 33s 13ms/sample - loss: 0.0301 - accuracy: 0.9898 - va
l_loss: 0.3038 - val_accuracy: 0.9347
Epoch 23/100
```

```
2452/2452 [==============================] - 33s 14ms/sample - loss: 0.0145 - accuracy: 0.9947 - va
l_loss: 0.2990 - val_accuracy: 0.9429
Epoch 24/100
2452/2452 [==============================] - 35s 14ms/sample - loss: 0.0167 - accuracy: 0.9943 - va
l_loss: 0.3216 - val_accuracy: 0.9478
Epoch 25/100
2452/2452 [==============================] - 35s 14ms/sample - loss: 0.0138 - accuracy: 0.9967 - va
l_loss: 0.3220 - val_accuracy: 0.9396
```

Analyzing the model

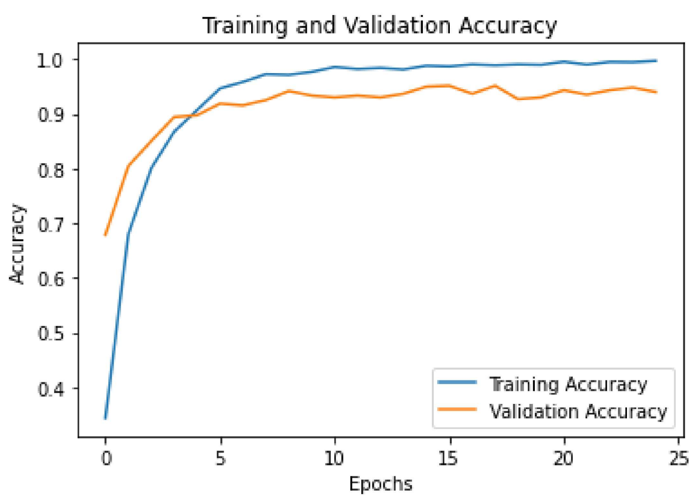
In [109]:

```
plt.plot(history.history['loss'], label = 'Training Loss')
plt.plot(history.history['val_loss'], label = 'Validation Loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Cross-Entropy Loss')
plt.title('Training and Validation Loss')
plt.savefig('Training Loss Plot.png')
```



In [110]:

```
plt.plot(history.history['accuracy'], label = 'Training Accuracy')
plt.plot(history.history['val_accuracy'], label = 'Validation Accuracy')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.savefig('Training Accuracy Plot.png')
```



Analyzing on Test Data

In [111]:
```
# Creating the test dataset
image_paths = [base_path + "zTest_Data\\Ones\\Right\\", base_path + "zTest_Data\\Ones\\Left\\",
               base_path + "zTest_Data\\Twos\\Right\\", base_path + "zTest_Data\\Twos\\Left\\",
               base_path + "zTest_Data\\Threes\\Right\\", base_path + "zTest_Data\\Threes\\Left\\",
               base_path + "zTest_Data\\Fours\\Right\\", base_path + "zTest_Data\\Fours\\Left\\",
               base_path + "zTest_Data\\Fives\\Right\\", base_path + "zTest_Data\\Fives\\Left\\",]
num_images = []
for i in range(int(len(image_paths))):
    num_images.append(len(os.listdir(image_paths[i][:-1])))

X_test = np.zeros((sum(num_images), image_sizes[0], image_sizes[1]))
y_test = np.zeros(sum(num_images))
summed = 0
for i in range(int(len(image_paths))):
    for j in range(num_images[i]):
        X_test[summed + j, : :] = np.asarray(Image.open(image_paths[i] + f"image{j}.jpg")) / 255
        y_test[summed + j] = i // 2
    summed += num_images[i]
```

In [112]:
```
predictions = model.predict(np.expand_dims(X_test, axis=3))
y_pred = np.argmax(predictions, axis=1)
```

In [113]:
```
print(f"The test accuracy was {sum(y_pred == y_test) / y_test.shape[0]}")
```

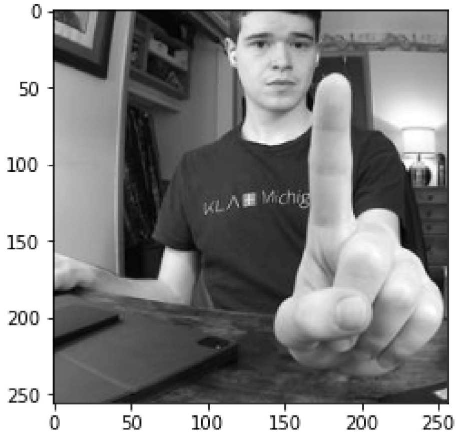
```
The test accuracy was 0.9161111111111111
```

In [114]:
```
pd.DataFrame(confusion_matrix(y_test, y_pred),
             columns = ["Predicted 1 Finger", "Predicted 2 Fingers", "Predicted 3 Fingers",
                        "Predicted 4 Fingers", "Predicted 5 Fingers"],
             index = ["Actually 1 Finger", "Actually 2 Fingers", "Actually 3 Fingers",
                      "Actually 4 Fingers", "Actually 5 Fingers"])
```

Out[114]:

	Predicted 1 Finger	Predicted 2 Fingers	Predicted 3 Fingers	Predicted 4 Fingers	Predicted 5 Fingers
Actually 1 Finger	350	10	0	0	0
Actually 2 Fingers	7	349	4	0	0
Actually 3 Fingers	4	18	331	7	0
Actually 4 Fingers	5	3	33	298	21
Actually 5 Fingers	6	0	1	32	321

In [200]:
```
# Saving Examples of Missclassified Images
image_path = "C:\\School\\EECS 605\\Project\\Data\\zMissclassified_Examples\\"
missclassified = np.where(y_pred != y_test)[0]
for i in range(12):
    missclassified_image = np.random.choice(missclassified)
    plt.imshow(X_test[missclassified_image,:,:], cmap='gray')
    plt.savefig(f"{image_path}" +
                f"Missclassified {i}. Predicted {y_pred[missclassified_image]+1}.jpg")
```



Saving the model as an ONNX model

In [115]:
```
import keras2onnx
onnx_model = keras2onnx.convert_keras(model)
keras2onnx.save_model(onnx_model, "my_model.onnx")
```

```
tf executing eager_mode: True
tf.keras model eager_mode: False
The ONNX operator number change on the optimization: 44 -> 21
```

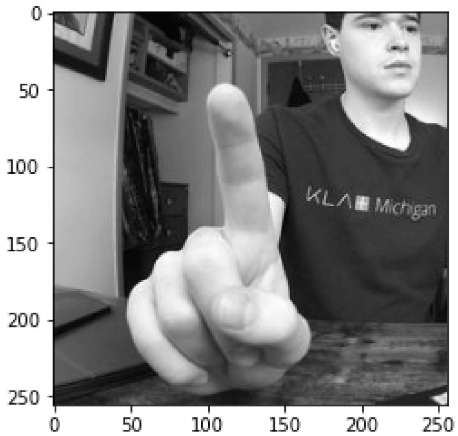
Testing that the ONNX model works

In [116]:
```
import onnxruntime
sess = onnxruntime.InferenceSession("my_model.onnx")
```

In [117]:
```
def makeInferences(sess, input_img):
    input_name = sess.get_inputs()[0].name
    output_name = sess.get_outputs()[0].name
    pred_onx = sess.run([output_name], {input_name: input_img})[0]
    return pred_onx
```

In [118]:
```
image_test_num = 17
input_img = np.reshape(X_test[image_test_num,:,:], (1,256,256,1))
input_img = input_img.astype(np.float32)
plt.imshow(np.reshape(input_img, (256,256)), cmap='gray')
```

Out[118]: <matplotlib.image.AxesImage at 0x262c59bb1c8>



In [119]:
```
scores = makeInferences(sess, input_img)
print(f"The predicted number of fingers was {np.argmax(scores) + 1}")
```

The predicted number of fingers was 1

Testing on my test image

In [120]:
```
def preprocess(image):
    grayImage = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    midx, midy = int(grayImage.shape[1]/2), int(grayImage.shape[0]/2)
    crop_img = grayImage[:, midx-midy:midx+midy]

    img = Image.fromarray(crop_img)
    img = img.resize((256, 256), Image.ANTIALIAS)

    return img
```

In [181]:

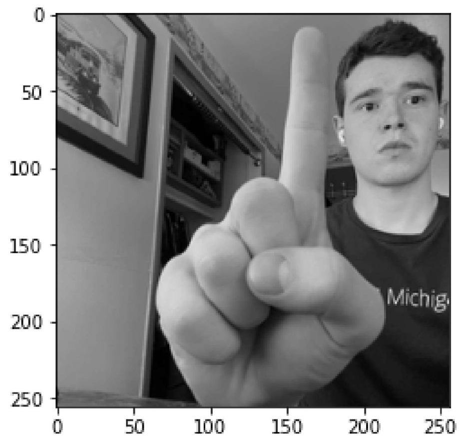
```
image_path = base_path + "zPhone_Test_Data\\"
image_name = "IMG_3566.jpg"
image_path = image_path + image_name

img = np.asarray(Image.open(image_path), dtype=np.float32)
img = preprocess(img)

plt.imshow(img)
```

Out[181]:

```
<matplotlib.image.AxesImage at 0x262bf7d0448>
```



In [182]:

```
img = np.asarray(img, dtype=np.float32) / 255
img = np.reshape(img, (1,256,256,1))
scores = makeInferences(sess, img)
scores
```

Out[182]:

```
array([[7.6353335e-01, 1.8507418e-01, 5.1378593e-02, 1.3874003e-05,
        1.2649862e-08]], dtype=float32)
```

In [183]:

```
np.argmax(scores) + 1
```

Out[183]:

```
1
```