First import required libraries

In [ ]:
```
import numpy as np
import keras as k
import tensorflow as tf
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Rescaling, Dropout,
from keras.utils import image_dataset_from_directory
from keras.applications.mobilenet_v2 import MobileNetV2
```

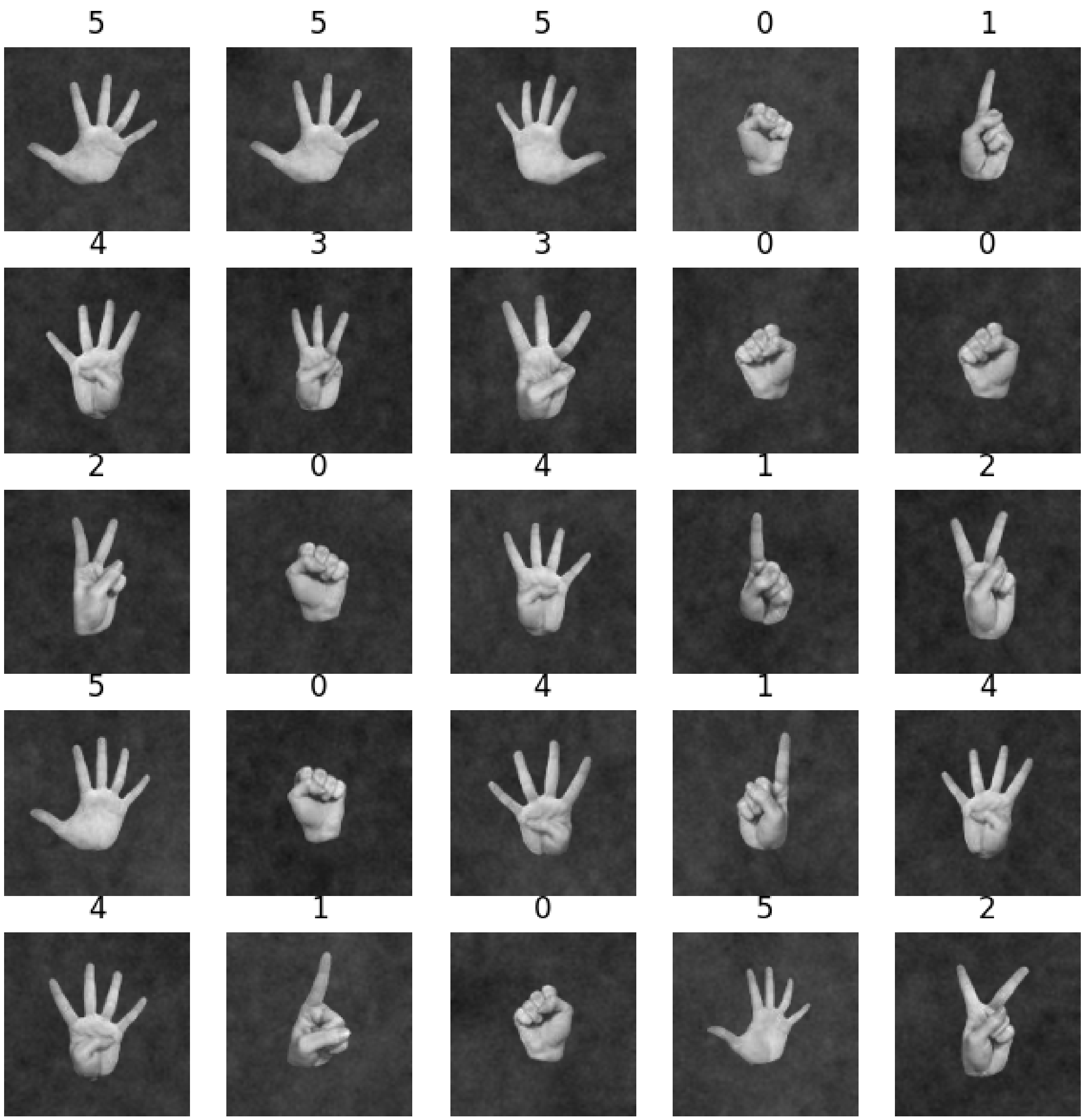
Set up parameters and create the data sets. This data set is a large set of image of hands holding up any nubmer of fingers. I reorganized all the files into folders for each class as well as ignoring the left vs right hand images. The dataset specifies the left vs right hand images are generated by mirroring the image and thus I dont care to differentiate between them and only use it to cut out a data augmentation step.

In [ ]:
```
train_dir = 'data/train'
test_dir = 'data/test'
image_width = 128
image_height = 128

train_ds = image_dataset_from_directory(train_dir, seed=0, image_size=(image_height
val_ds = image_dataset_from_directory(train_dir, seed=0, image_size=(image_height,
test_ds = image_dataset_from_directory(test_dir, seed=0, image_size=(image_height,

class_names = ['0','1','2','3','4','5']

train_ds = train_ds.cache().prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=tf.data.AUTOTUNE)
```

```
Found 18000 files belonging to 6 classes.
Using 14400 files for training.
Found 18000 files belonging to 6 classes.
Using 3600 files for validation.
Found 3600 files belonging to 6 classes.
```

Showing some examples from the training data set.

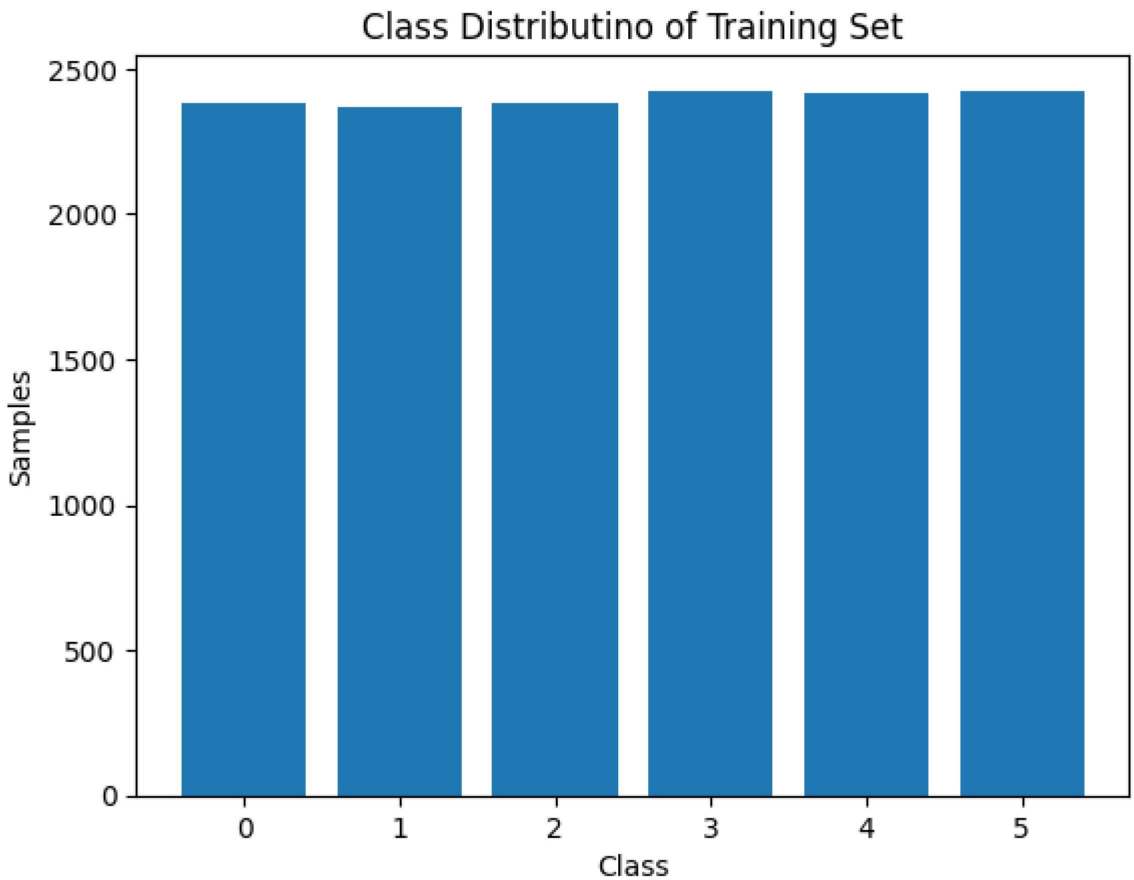
In [ ]:
```
plt.figure(figsize=(8, 8))
for images, labels in train_ds.take(1):
  for i in range(25):
    ax = plt.subplot(5, 5, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```



This graph shows the reletivly even distribution of the classes.

In [ ]:

```
train_counts = np.zeros(len(class_names))
for _,labels in train_ds:
    for label in labels:
        train_counts[label] += 1
plt.bar(class_names,train_counts)
plt.title('Class Distributino of Training Set')
plt.xlabel('Class')
plt.ylabel('Samples')
plt.show()
```



This first model is just a simple sequential model. The first few layers are for data augementation by applying random zoom and rotaion. I dont do a horizontal filp as the data set includes imagaes that are already fliped horizontaly to represent left and right hands. Then a rescaling layer. And finaly a flatten and 3 densly connected layers connected to the output layer.

In [ ]:
```
seq = Sequential([
    RandomRotation(0.2, input_shape=(image_width,image_height,3)),
    RandomZoom(0.2),
    Rescaling(1./255),
    Flatten(),
    Dense(units=128, activation='relu'),
    Dense(units=64, activation='relu'),
    Dense(units=32, activation='relu'),
    Dense(units=len(class_names)),
])
seq.build()
seq.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 random_rotation (RandomRota  (None, 128, 128, 3)      0         
 tion)                                                           
                                                                 
 random_zoom (RandomZoom)    (None, 128, 128, 3)       0         
                                                                 
 rescaling (Rescaling)       (None, 128, 128, 3)       0         
                                                                 
 flatten (Flatten)           (None, 49152)             0         
                                                                 
 dense (Dense)               (None, 128)               6291584   
                                                                 
 dense_1 (Dense)             (None, 64)                8256      
                                                                 
 dense_2 (Dense)             (None, 32)                2080      
                                                                 
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 random_rotation (RandomRota  (None, 128, 128, 3)      0         
 tion)                                                           
                                                                 
 random_zoom (RandomZoom)    (None, 128, 128, 3)       0         
                                                                 
 rescaling (Rescaling)       (None, 128, 128, 3)       0         
                                                                 
 flatten (Flatten)           (None, 49152)             0         
                                                                 
 dense (Dense)               (None, 128)               6291584   
                                                                 
 dense_1 (Dense)             (None, 64)                8256      
                                                                 
 dense_2 (Dense)             (None, 32)                2080      
                                                                 
 dense_3 (Dense)             (None, 6)                 198       
                                                                 
=================================================================
Total params: 6,302,118
Trainable params: 6,302,118
Non-trainable params: 0
_________________________________________________________________
```

This next model is the CNN model. It has the same data augementation and rescaling layers, followed by 2 sets of a 2D convolution layer and a 2D pooling layer. Then a flattening layer connected to a dense layer that goes to the output layer.

In [ ]:

```
cnn = Sequential([
    RandomRotation(0.2, input_shape=(image_width,image_height,3)),
    RandomZoom(0.2),
    Rescaling(1./255),
    Conv2D(16, kernel_size=(15, 15), activation='relu'),
    MaxPooling2D(pool_size=(5, 5)),
    Conv2D(32, kernel_size=(5, 5), activation='relu'),
```

```
    MaxPooling2D(pool_size=(3, 3)),
    Dropout(0.2),
    Flatten(),
    Dense(units=64, activation='relu'),
    Dense(units=len(class_names)),
])
cnn.build()
cnn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 random_rotation_1 (RandomRo  (None, 128, 128, 3)      0         
 tation)                                                         
                                                                 
 random_zoom_1 (RandomZoom)  (None, 128, 128, 3)       0         
                                                                 
 rescaling_1 (Rescaling)     (None, 128, 128, 3)       0         
                                                                 
 conv2d (Conv2D)             (None, 114, 114, 16)      10816     
                                                                 
 max_pooling2d (MaxPooling2D  (None, 22, 22, 16)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 18, 18, 32)        12832     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 6, 6, 32)         0         
 2D)                                                             
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 random_rotation_1 (RandomRo  (None, 128, 128, 3)      0         
 tation)                                                         
                                                                 
 random_zoom_1 (RandomZoom)  (None, 128, 128, 3)       0         
                                                                 
 rescaling_1 (Rescaling)     (None, 128, 128, 3)       0         
                                                                 
 conv2d (Conv2D)             (None, 114, 114, 16)      10816     
                                                                 
 max_pooling2d (MaxPooling2D  (None, 22, 22, 16)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 18, 18, 32)        12832     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 6, 6, 32)         0         
 2D)                                                             
                                                                 
 dropout (Dropout)           (None, 6, 6, 32)          0         
                                                                 
 flatten_1 (Flatten)         (None, 1152)              0         
                                                                 
 dense_4 (Dense)             (None, 64)                73792     
                                                                 
 dense_5 (Dense)             (None, 6)                 390       
                                                                 
=================================================================
Total params: 97,830
Trainable params: 97,830
Non-trainable params: 0
_________________________________________________________________
```

This final model uses the pretrained model MobilenNetV2. I add the same data augementation and rescaling layers as input to the pretrained model and then connect the output to a single dense layer connected to the output layer.

In [ ]:
```
mobilenet = MobileNetV2(weights="imagenet", pooling='avg', include_top=False, input
mobilenet.trainable = False

inputs = k.Input(shape=(image_width,image_height,3))
x = RandomRotation(0.2)(inputs)
x = RandomZoom(0.2)(x)
x = Rescaling(1./127.5, offset=-1)(inputs)
x = mobilenet(x, training=False)
x = Dense(units=64, activation='relu')(x)
outputs = x = Dense(units=len(class_names))(x)

pretrained = k.Model(inputs,outputs)
pretrained.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 128, 128, 3)]     0

 rescaling_3 (Rescaling)     (None, 128, 128, 3)       0

 mobilenetv2_1.00_128 (Funct  (None, 1280)             2257984
 ional)

 dense_8 (Dense)             (None, 64)                81984

 dense_9 (Dense)             (None, 6)                 390

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 128, 128, 3)]     0

 rescaling_3 (Rescaling)     (None, 128, 128, 3)       0

 mobilenetv2_1.00_128 (Funct  (None, 1280)             2257984
 ional)

 dense_8 (Dense)             (None, 64)                81984

 dense_9 (Dense)             (None, 6)                 390

=================================================================
Total params: 2,340,358
Trainable params: 82,374
Non-trainable params: 2,257,984
_________________________________________________________________
```

The training on the sequential model took about 4 minuets with 10 epochs.

In [ ]:
```
seq.compile(optimizer='adam', loss=k.losses.SparseCategoricalCrossentropy(from_logi
```

```
seq_history = seq.fit(train_ds, validation_data=val_ds, epochs=10)
```

```
Epoch 1/10
480/480 [==============================] - 24s 48ms/step - loss: 0.3960 - accuracy:
0.8471 - val_loss: 0.2199 - val_accuracy: 0.9222
Epoch 2/10
480/480 [==============================] - 23s 48ms/step - loss: 0.3299 - accuracy:
0.8737 - val_loss: 0.1125 - val_accuracy: 0.9633
Epoch 3/10
480/480 [==============================] - 23s 47ms/step - loss: 0.3081 - accuracy:
0.8807 - val_loss: 0.1910 - val_accuracy: 0.9175
Epoch 4/10
480/480 [==============================] - 22s 46ms/step - loss: 0.2550 - accuracy:
0.9027 - val_loss: 0.0661 - val_accuracy: 0.9808
Epoch 5/10
480/480 [==============================] - 22s 46ms/step - loss: 0.2492 - accuracy:
0.9057 - val_loss: 0.0695 - val_accuracy: 0.9814
Epoch 6/10
480/480 [==============================] - 22s 45ms/step - loss: 0.2137 - accuracy:
0.9198 - val_loss: 0.1912 - val_accuracy: 0.9117
Epoch 7/10
480/480 [==============================] - 22s 45ms/step - loss: 0.2282 - accuracy:
0.9180 - val_loss: 0.0827 - val_accuracy: 0.9722
Epoch 8/10
480/480 [==============================] - 22s 45ms/step - loss: 0.2005 - accuracy:
0.9251 - val_loss: 0.1058 - val_accuracy: 0.9594
Epoch 9/10
480/480 [==============================] - 22s 45ms/step - loss: 0.1890 - accuracy:
0.9300 - val_loss: 0.1030 - val_accuracy: 0.9628
Epoch 10/10
480/480 [==============================] - 22s 46ms/step - loss: 0.1728 - accuracy:
0.9351 - val_loss: 0.0616 - val_accuracy: 0.9811
```

The training on the cnn model took around 20 minuets with 5 epochs.

In [ ]:

```
cnn.compile(optimizer='adam', loss=k.losses.SparseCategoricalCrossentropy(from_logi

cnn_history = cnn.fit(train_ds, validation_data=val_ds, epochs=5)
```

```
Epoch 1/5
480/480 [==============================] - 199s 413ms/step - loss: 0.2408 - accurac
y: 0.9158 - val_loss: 0.0212 - val_accuracy: 0.9967
Epoch 2/5
480/480 [==============================] - 195s 406ms/step - loss: 0.0614 - accurac
y: 0.9801 - val_loss: 0.0059 - val_accuracy: 0.9992
Epoch 3/5
480/480 [==============================] - 194s 404ms/step - loss: 0.0380 - accurac
y: 0.9867 - val_loss: 0.0023 - val_accuracy: 0.9994
Epoch 4/5
480/480 [==============================] - 194s 405ms/step - loss: 0.0334 - accurac
y: 0.9892 - val_loss: 0.0107 - val_accuracy: 0.9958
Epoch 5/5
480/480 [==============================] - 194s 405ms/step - loss: 0.0291 - accurac
y: 0.9903 - val_loss: 0.0018 - val_accuracy: 1.0000
```

The training on the pretrained MobileNetV2 model took about 5 minuets with 5 epochs.

In [ ]:

```python
pretrained.compile(optimizer='adam', loss=k.losses.SparseCategoricalCrossentropy(),

pretrained_history = pretrained.fit(train_ds, validation_data=val_ds, epochs=5)
```

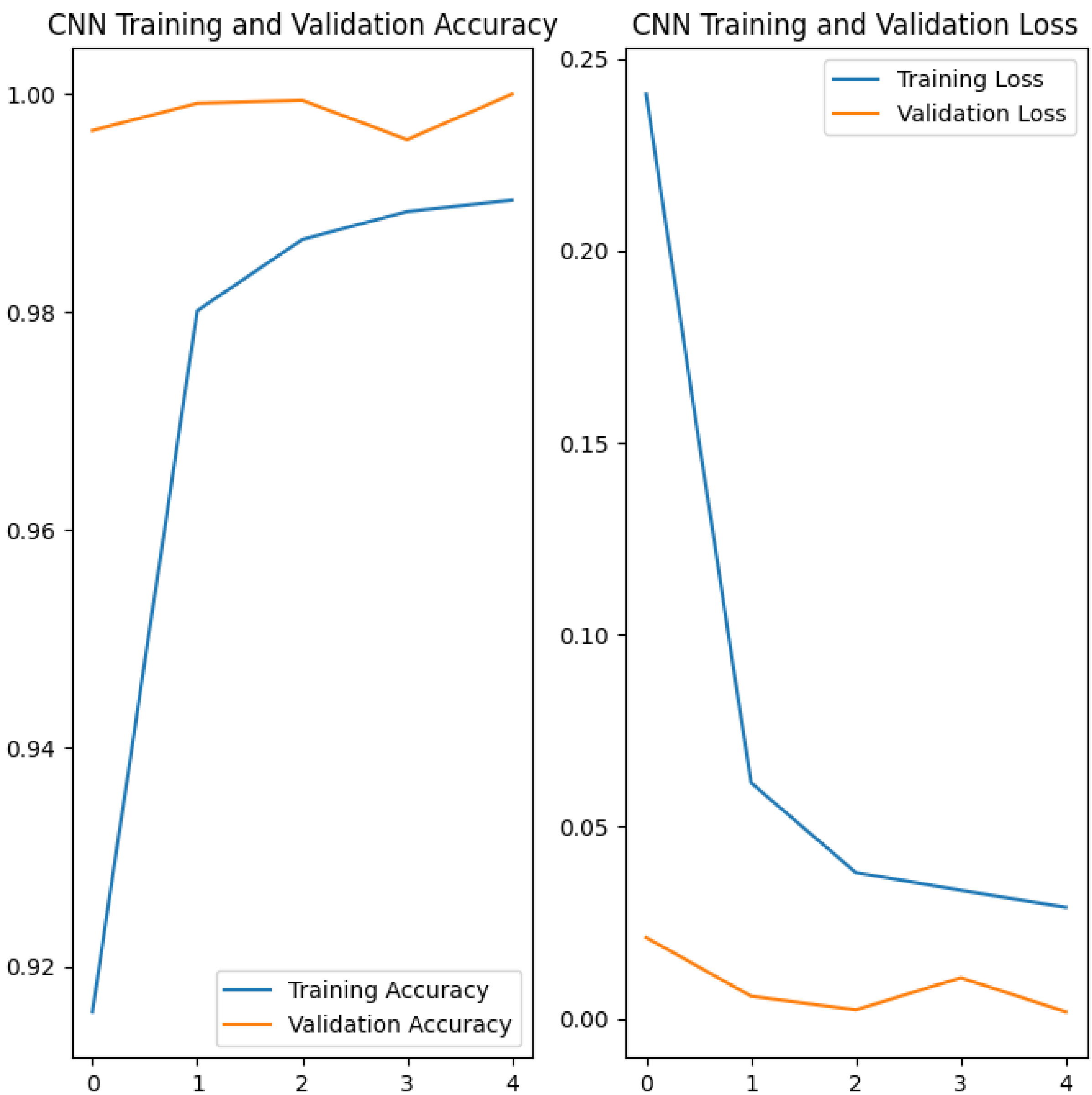
```
Epoch 1/5
480/480 [==============================] - 63s 126ms/step - loss: 1.8078 - accuracy:
0.2572 - val_loss: 1.7918 - val_accuracy: 0.2558
Epoch 2/5
480/480 [==============================] - 57s 118ms/step - loss: 1.7918 - accuracy:
0.2470 - val_loss: 1.7918 - val_accuracy: 0.2558
Epoch 3/5
480/480 [==============================] - 56s 117ms/step - loss: 1.7918 - accuracy:
0.2470 - val_loss: 1.7918 - val_accuracy: 0.2558
Epoch 4/5
480/480 [==============================] - 59s 122ms/step - loss: 1.7918 - accuracy:
0.2470 - val_loss: 1.7918 - val_accuracy: 0.2558
Epoch 5/5
480/480 [==============================] - 58s 121ms/step - loss: 1.7918 - accuracy:
0.2470 - val_loss: 1.7918 - val_accuracy: 0.2558
```

In [ ]:

```python
seq_acc = seq_history.history['accuracy']
seq_test_acc = seq_history.history['val_accuracy']

seq_loss = seq_history.history['loss']
seq_test_loss = seq_history.history['val_loss']
seq_erange = range(10)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(seq_erange, seq_acc, label='Training Accuracy')
plt.plot(seq_erange, seq_test_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Sequential Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(seq_erange, seq_loss, label='Training Loss')
plt.plot(seq_erange, seq_test_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Sequential Training and Validation Loss')
plt.show()
```

In [ ]:

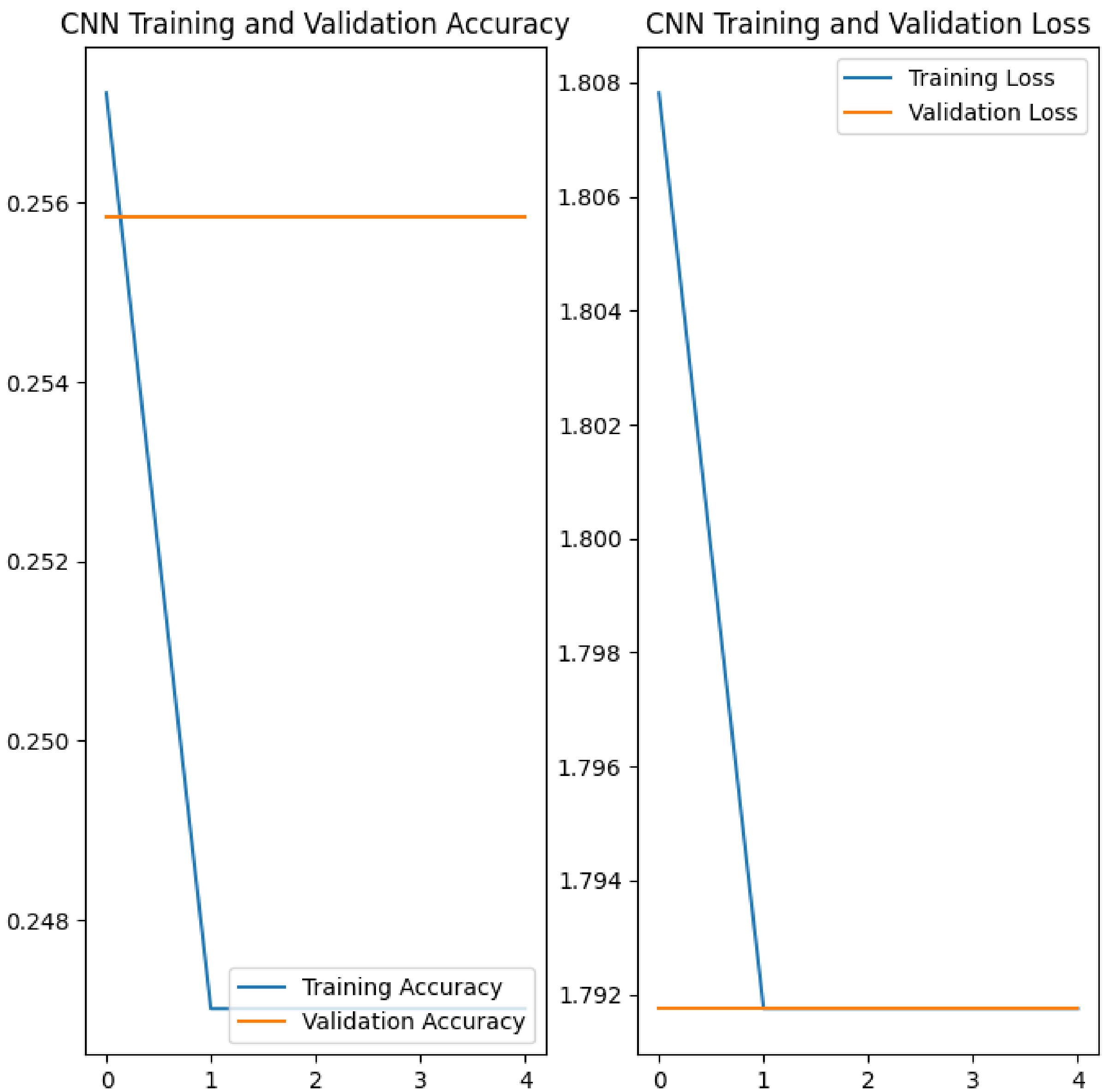
```
cnn_acc = cnn_history.history['accuracy']
cnn_test_acc = cnn_history.history['val_accuracy']

cnn_loss = cnn_history.history['loss']
cnn_test_loss = cnn_history.history['val_loss']
cnn_erange = range(5)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(cnn_erange, cnn_acc, label='Training Accuracy')
plt.plot(cnn_erange, cnn_test_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('CNN Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(cnn_erange, cnn_loss, label='Training Loss')
plt.plot(cnn_erange, cnn_test_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('CNN Training and Validation Loss')
plt.show()
```



In [ ]:

```
pretrained_acc = pretrained_history.history['accuracy']
pretrained_test_acc = pretrained_history.history['val_accuracy']

pretrained_loss = pretrained_history.history['loss']
pretrained_test_loss = pretrained_history.history['val_loss']
pretrained_erange = range(5)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(pretrained_erange, pretrained_acc, label='Training Accuracy')
plt.plot(pretrained_erange, pretrained_test_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('CNN Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(pretrained_erange, pretrained_loss, label='Training Loss')
plt.plot(pretrained_erange, pretrained_test_loss, label='Validation Loss')
plt.legend(loc='upper right')
```

```
plt.title('CNN Training and Validation Loss')
plt.show()
```



These are the evaluations of each model on the test set.

In [ ]:
```
seq_eval_loss, seq_eval_acc = seq.evaluate(test_ds)
print('Sequential Test Evaluation Loss:',seq_eval_loss)
print('Sequential Test Evaluation Accuracy:',seq_eval_acc)
```

```
120/120 [==============================] - 1s 4ms/step - loss: 0.0743 - accuracy: 0.
9717
Sequential Test Evaluation Loss: 0.07425636053085327
Sequential Test Evaluation Accuracy: 0.971666693687439
```

In [ ]:
```
cnn_eval_loss, cnn_eval_acc = cnn.evaluate(test_ds)
print('CNN Test Evaluation Loss:',cnn_eval_loss)
print('CNN Test Evaluation Accuracy:',cnn_eval_acc)
```

```
120/120 [==============================] - 4s 37ms/step - loss: 0.0012 - accuracy: 1.0000
CNN Test Evaluation Loss: 0.0011581633007153869
CNN Test Evaluation Accuracy: 1.0
```

In [ ]:
```
pretrained_eval_loss, pretrained_eval_acc = pretrained.evaluate(test_ds)
print('Pretrained Test Evaluation Loss:',pretrained_eval_loss)
print('Pretrained Test Evaluation Accuracy:',pretrained_eval_acc)
```

```
120/120 [==============================] - 13s 105ms/step - loss: 1.7918 - accuracy: 0.2536
Pretrained Test Evaluation Loss: 1.7917604446411133
Pretrained Test Evaluation Accuracy: 0.25361111760139465
```

As seen here, the best perfroming model was the cnn. this is not too surprising as this data set is meant for cnn image classification and is very clean. The sequential model performed much better than I thought it would though which is interesting. The pretrained model just would not perform well. I tried many differnt configurations and such and nothing I did could get it over the 25% accuracy. I dont know why it just didnt perform as well.

This is the final thing i did more for fun than anything. I created a single image of my hand with 2 fingers up as shown. Ironically, the pretrained model is the only one to predict correctly. I do not think this is a good test though as my image is really not in the same format as the images in the data set nor is it of the same quality. This was more for my own curiosity that actual evaluation.

In [ ]:
```
test_image = k.utils.load_img('data/my_hand_2.png', target_size=(image_width, image
test_image = k.utils.img_to_array(test_image)
test_image = tf.expand_dims(test_image, 0)

plt.figure(figsize=(3,3))
plt.imshow(test_image[0].numpy().astype("uint8"))
plt.title(class_names[2])
plt.axis("off")

seq_pred = seq.predict(test_image)
cnn_pred = cnn.predict(test_image)
pretrained_pred = pretrained.predict(test_image)

seq_score = tf.nn.softmax(seq_pred[0])
cnn_score = tf.nn.softmax(cnn_pred[0])
pretrained_score = tf.nn.softmax(pretrained_pred[0])

print("Sequential Model Predicted Class: {} ({:.2f})".format(class_names[np.argmax(
print("CNN Model Predicted Class: {} ({:.2f})".format(class_names[np.argmax(cnn_sco
print("Pretrained Model Predicted Class: {} ({:.2f})".format(class_names[np.argmax(
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 23ms/step
Sequential Model Predicted Class: 5 (98.02)
CNN Model Predicted Class: 3 (99.83)
Pretrained Model Predicted Class: 2 (74.51)
```

2