In [12]:
```
import os
import random
from imutils import paths
import matplotlib.pyplot as plt
import argparse
import cv2
import tensorflow as tf
from tensorflow.keras.models import load_model
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
import numpy as np
from tensorflow import keras
```

In [13]:
```
data = []
labels = []
imagePaths = list(paths.list_images("gesture"))
for imagePath in imagePaths:
    image = cv2.imread(imagePath)
    data.append(image)
    # extract the label from the image path and update the labels list
    label = imagePath.split(os.path.sep)[-1][0]

    labels.append(int(label))
```

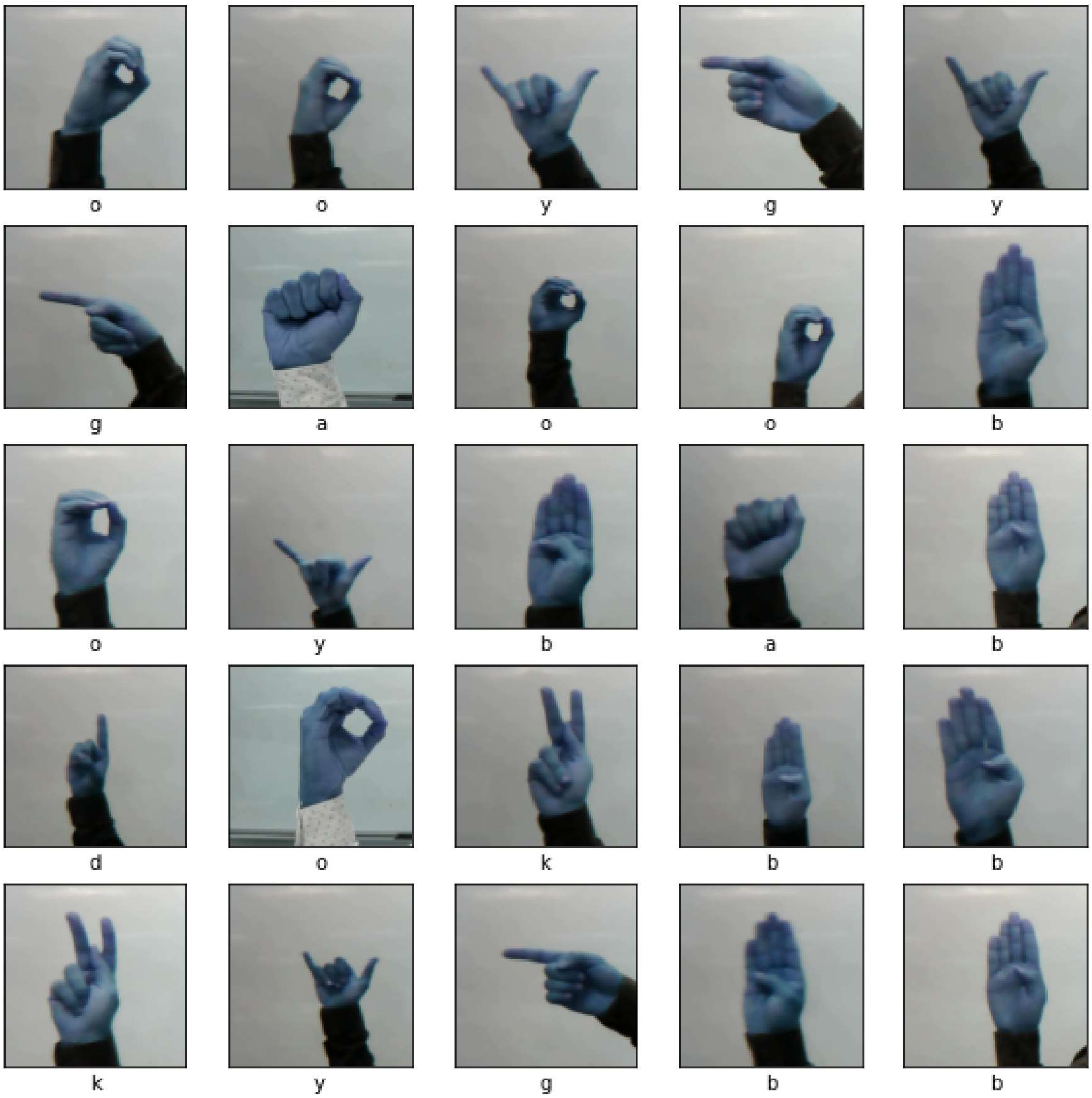
In [3]:
```
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)
```

In [4]:
```
labels
```

Out[4]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6,
       6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
       6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7,
       7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
       7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7])
```

In [5]:
```
(trainX, testX, trainY, testY) = train_test_split(data,
    labels, test_size=0.15, random_state=22)
```

In [48]:
```
plt.figure(figsize=(10,10))
for i in range(25):
  plt.subplot(5,5,i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(trainX[i], cmap=plt.cm.gray)
  plt.xlabel(trainY[i])
plt.show()
```



In [7]:
```
model = keras.Sequential([
  keras.layers.InputLayer(input_shape=(400, 400, 3)),
  keras.layers.Reshape(target_shape=(400, 400, 3)),
  keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation=tf.nn.relu),
  keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation=tf.nn.relu),
  keras.layers.MaxPooling2D(pool_size=(2, 2)),
  keras.layers.Dropout(0.25),
  keras.layers.Flatten(),
  keras.layers.Dense(10)
])

# Define how to train the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True
              metrics=['accuracy'])

# Train the digit classification model
model.fit(trainX, trainY, epochs=5)
```

```
Epoch 1/5
8/8 [==============================] - 140s 18s/step - loss: 31.4515 - accurac
y: 0.1030
Epoch 2/5
8/8 [==============================] - 121s 15s/step - loss: 2.8156 - accuracy:
0.3305
Epoch 3/5
8/8 [==============================] - 117s 15s/step - loss: 0.9184 - accuracy:
0.6953
Epoch 4/5
8/8 [==============================] - 117s 15s/step - loss: 0.4348 - accuracy:
0.8412
Epoch 5/5
8/8 [==============================] - 117s 15s/step - loss: 0.1656 - accuracy:
0.9700
```

Out[7]:
```
<tensorflow.python.keras.callbacks.History at 0x28568e57908>
```

In [8]:
```
model.summary()
```

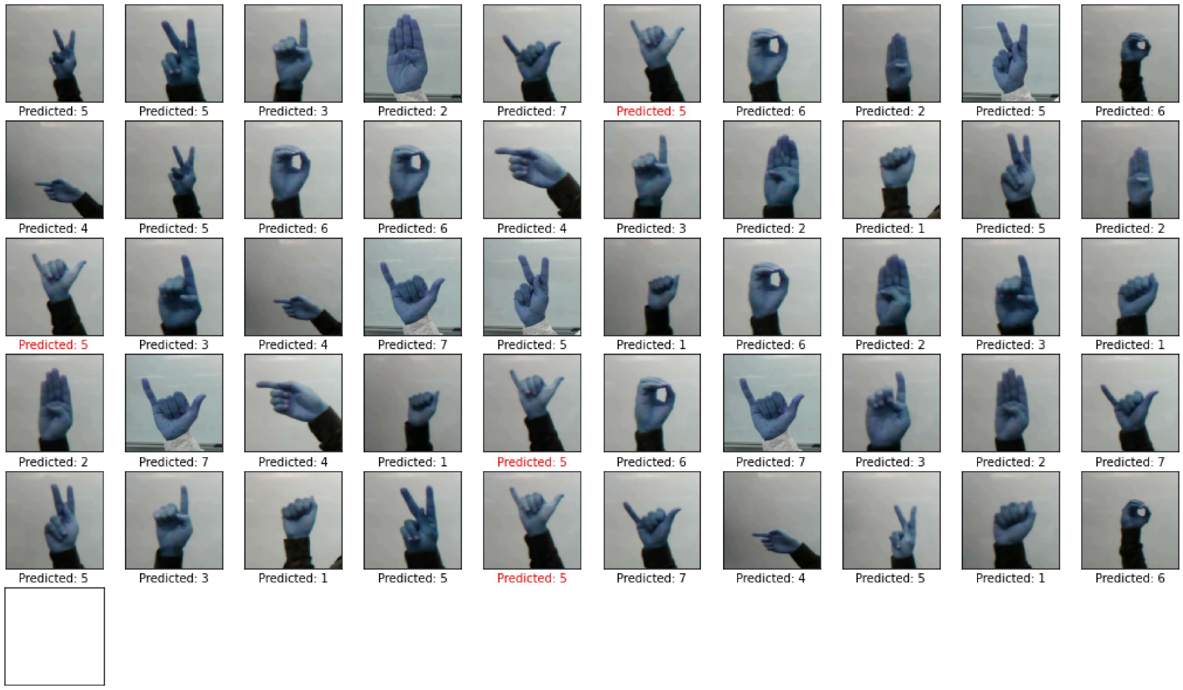
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_1 (Reshape)          (None, 400, 400, 3)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 398, 398, 32)      896
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 396, 396, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 198, 198, 64)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 198, 198, 64)      0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2509056)           0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                25090570
=================================================================
Total params: 25,109,962
Trainable params: 25,109,962
Non-trainable params: 0
_________________________________________________________________
```

In [10]:
```
test_loss, test_acc = model.evaluate(testX, testY)

print('Test accuracy:', test_acc)
```

```
2/2 [==============================] - 2s 815ms/step - loss: 0.3660 - accuracy:
0.9048
Test accuracy: 0.9047619104385376
```

In [15]:
```
def get_label_color(val1, val2):
  if val1 == val2:
    return 'black'
  else:
    return 'red'

# Predict the labels of digit images in our test dataset.
predictions = model.predict(testX)


prediction_digits = np.argmax(predictions, axis=1)


# dataset, we will highlight it in red color.
plt.figure(figsize=(18, 18))
for i in range(100):
  ax = plt.subplot(10, 10, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  image_index = random.randint(0, len(prediction_digits))
  plt.imshow(testX[image_index], cmap=plt.cm.gray)
  ax.xaxis.label.set_color(get_label_color(prediction_digits[image_index],\
                                           testY[image_index]))
  plt.xlabel('Predicted: %d' % prediction_digits[image_index])
plt.show()
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-15-ec4c7ded4c22> in <module>
     20   plt.grid(False)
     21   image_index = random.randint(0, len(prediction_digits))
---> 22   plt.imshow(testX[image_index], cmap=plt.cm.gray)
     23   ax.xaxis.label.set_color(get_label_color(prediction_digits[image_inde
x],\
     24                                            testY[image_index]))

IndexError: index 42 is out of bounds for axis 0 with size 42
```



In [17]:
```
# Convert Keras model to TF Lite format.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_float_model = converter.convert()

# Show model size in KBs.
float_model_size = len(tflite_float_model) / 1024
print('Float model size = %dKBs.' % float_model_size)
```

```
Float model size = 98087KBs.
```

In [18]:
```
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quantized_model = converter.convert()

# Show model size in KBs.
quantized_model_size = len(tflite_quantized_model) / 1024
print('Quantized model size = %dKBs,' % quantized_model_size)
print('which is about %d%% of the float model size.'\
      % (quantized_model_size * 100 / float_model_size))
```

```
Quantized model size = 24526KBs,
which is about 25% of the float model size.
```

In [27]:
```
def evaluate_tflite_model(tflite_model):
  # Initialize TFLite interpreter using the model.
  interpreter = tf.lite.Interpreter(model_content=tflite_model)
  interpreter.allocate_tensors()
  input_tensor_index = interpreter.get_input_details()[0]["index"]
  output = interpreter.tensor(interpreter.get_output_details()[0]["index"])

  # Run predictions on every image in the "test" dataset.
  prediction_digits = []
  for test_image in trainX:
    # Pre-processing: add batch dimension and convert to float32 to match with
    # the model's input data format.
    test_image = np.expand_dims(test_image, axis=0).astype(np.float32)
    interpreter.set_tensor(input_tensor_index, test_image)

    # Run inference.
    interpreter.invoke()

    # Post-processing: remove batch dimension and find the digit with highest
    # probability.
    digit = np.argmax(output()[0])
    prediction_digits.append(digit)

  # Compare prediction results with ground truth labels to calculate accuracy.
  accurate_count = 0
  for index in range(len(prediction_digits)):
    if prediction_digits[index] == trainY[index]:
      accurate_count += 1
  accuracy = accurate_count * 1.0 / len(prediction_digits)

  return accuracy

# Evaluate the TF Lite float model. You'll find that its accurary is identical
# to the original TF (Keras) model because they are essentially the same model
# stored in different format.
float_accuracy = evaluate_tflite_model(tflite_float_model)
print('Float model accuracy = %.4f' % float_accuracy)

# Evalualte the TF Lite quantized model.
# Don't be surprised if you see quantized model accuracy is higher than
# the original float model. It happens sometimes :)
quantized_accuracy = evaluate_tflite_model(tflite_quantized_model)
print('Quantized model accuracy = %.4f' % quantized_accuracy)
print('Accuracy drop = %.4f' % (float_accuracy - quantized_accuracy))
```

```
Float model accuracy = 0.9957
Quantized model accuracy = 0.9957
Accuracy drop = 0.0000
```

In [28]:
```
f = open('gesture.tflite', "wb")
f.write(tflite_quantized_model)
f.close()
```

In [ ]:

```
```