In [1]:
```
# ATTENTION: Please do not alter any of the provided code in the exercise. Only c
# ATTENTION: Please do not add or remove any cells in the exercise. The grader wi
# ATTENTION: Please use the provided epoch values when training.

import csv
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from os import getcwd
```

In [18]:
```
def get_data(filename):
  # You will need to write code that will read the file passed
  # into this function. The first line contains the column headers
  # so you should ignore it
  # Each successive line contians 785 comma separated values between 0 and 255
  # The first value is the label
  # The rest are the pixel values for that picture
  # The function will return 2 np.array types. One with all the labels
  # One with all the images
  #
  # Tips:
  # If you read a full line (as 'row') then row[0] has the label
  # and row[1:785] has the 784 pixel values
  # Take a look at np.array_split to turn the 784 pixels into 28x28
  # You are reading in strings, but need the values to be floats
  # Check out np.array().astype for a conversion
    with open(filename) as training_file:
      # Your code starts here
        lines_excluding_header = training_file.readlines()[1:]
        size = len(lines_excluding_header)
        labels = np.zeros(size)
        images = np.zeros((size, 28, 28))
        for index, line in enumerate(lines_excluding_header):
            individual_item = line.strip().split(",")
            if line.strip().split(","):
                #only first item is a label
                labels[index] = int(individual_item[0])
                #everthing else is an image
                image = np.asarray(individual_item[1:], dtype=np.float32)
                image = np.array_split(image, 28)
                images[index, :, :] = image
      # Your code ends here
    return images, labels

path_sign_mnist_train = f"{getcwd()}/../tmp2/sign_mnist_train.csv"
path_sign_mnist_test = f"{getcwd()}/../tmp2/sign_mnist_test.csv"
training_images, training_labels = get_data(path_sign_mnist_train)
testing_images, testing_labels = get_data(path_sign_mnist_test)

# Keep these
print(training_images.shape)
print(training_labels.shape)
print(testing_images.shape)
print(testing_labels.shape)

# Their output should be:
# (27455, 28, 28)
# (27455,)
# (7172, 28, 28)
# (7172,)
```

```
(27455, 28, 28)
(27455,)
(7172, 28, 28)
(7172,)
```

In [19]:
```
# In this section you will have to add another dimension to the data
# So, for example, if your array is (10000, 28, 28)
# You will need to make it (10000, 28, 28, 1)
# Hint: np.expand_dims

training_images = np.expand_dims(training_images, axis=3)
testing_images = np.expand_dims(testing_images, axis=3)

# Create an ImageDataGenerator and do Image Augmentation
train_datagen = ImageDataGenerator(
    rescale=1/255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

validation_datagen = ImageDataGenerator(
    rescale=1/255
)

# Keep These
print(training_images.shape)
print(testing_images.shape)

# Their output should be:
# (27455, 28, 28, 1)
# (7172, 28, 28, 1)
```

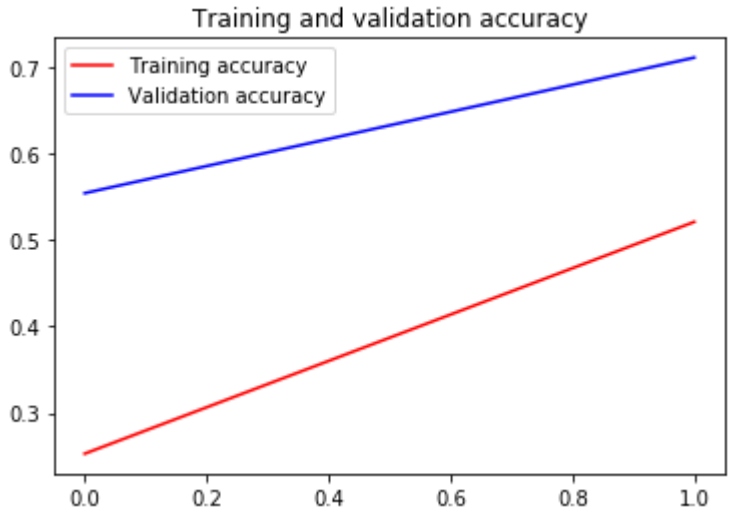
```
(27455, 28, 28, 1)
(7172, 28, 28, 1)
```

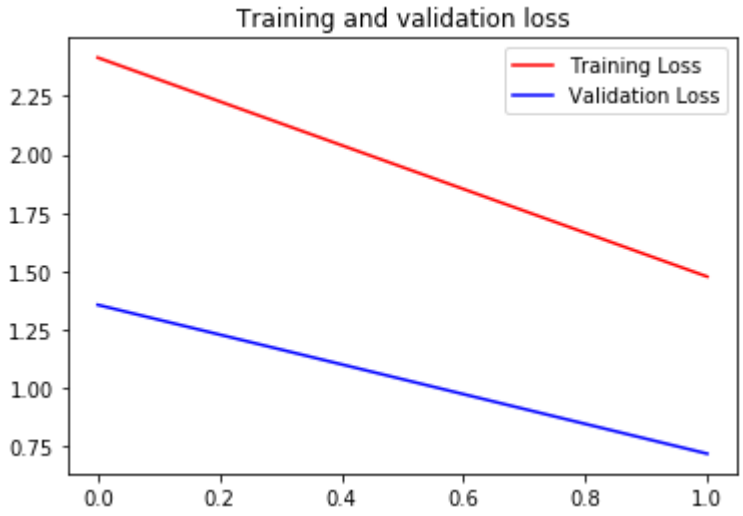
In [32]:
```
# Define the model
# Use no more than 2 Conv2D and 2 MaxPooling2D
from tensorflow.keras.optimizers import RMSprop

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(25, activation='softmax')
])

train_generator = train_datagen.flow(training_images, training_labels)

validation_generator = validation_datagen.flow(testing_images, testing_labels)

# Compile Model.
model.compile(
    optimizer=RMSprop(lr=0.002),
    loss='sparse_categorical_crossentropy',
    metrics=['acc']
)

# Train the Model
history = model.fit_generator(
    train_generator,
    epochs=2,
    validation_data=validation_generator,
    verbose=2
)

model.evaluate(testing_images, testing_labels, verbose=0)
```

```
Epoch 1/2
858/858 - 76s - loss: 2.4145 - acc: 0.2528 - val_loss: 1.3569 - val_acc: 0.5540
Epoch 2/2
858/858 - 77s - loss: 1.4776 - acc: 0.5208 - val_loss: 0.7201 - val_acc: 0.7107
```

Out[32]:
```
[190.11476199080408, 0.5064138]
```

In [34]:
```
# Plot the chart for accuracy and loss on both training and validation
%matplotlib inline
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()

plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





Submission Instructions

In [ ]:
```
# Now click the 'Submit Assignment' button above.
```

When you're done or would like to take a break, please run the two cells below to save your work and close the Notebook. This will free up resources for your fellow learners.

In [ ]:
```
%%javascript
<!-- Save the notebook -->
IPython.notebook.save_checkpoint();
```

In [ ]:
```
%%javascript
IPython.notebook.session.delete();
window.onbeforeunload = null
setTimeout(function() { window.close(); }, 1000);
```