In [1]:

```
# ATTENTION: Please do not alter any of the provided code in the exercise. Only add your own code where indicated
# ATTENTION: Please do not add or remove any cells in the exercise. The grader will check specific cells based on the cell position.
# 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 [2]:

```
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:
        csv_reader = csv.reader(training_file, delimiter=',')
        first_line = True
        temp_images = []
        temp_labels = []
        for row in csv_reader:
            if first_line:
                first_line = False
            else:
                temp_labels.append(row[0])
                image_data = row[1:785]
                image_data_as_array = np.array_split(image_data, 28)
                temp_images.append(image_data_as_array)
        images = np.array(temp_images).astype('float')
        labels = np.array(temp_labels).astype('float')

    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 [3]:

```
# 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 [4]:

```
# Define the model
# Use no more than 2 Conv2D and 2 MaxPooling2D
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64, (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(128, activation='relu'),
    tf.keras.layers.Dense(26, activation='softmax')
])

# Compile Model.
model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy'])

# Train the Model
history = model.fit_generator(train_datagen.flow(training_images, training_labels, batch_size=32),
                    steps_per_epoch=len(training_images) / 32,
                    epochs=15,
                    validation_data=validation_datagen.flow(testing_images, testing_labels,
batch_size=32),
                    validation_steps=len(testing_images) / 32)

model.evaluate(testing_images, testing_labels, verbose=2)
```

```
Epoch 1/15
858/857 [==============================] - 58s 68ms/step - loss: 2.8340 - accuracy: 0.1396 - val_l
oss: 2.3132 - val_accuracy: 0.2867
Epoch 2/15
858/857 [==============================] - 50s 59ms/step - loss: 2.1575 - accuracy: 0.3127 - val_l
oss: 1.4174 - val_accuracy: 0.5022
Epoch 3/15
858/857 [==============================] - 48s 56ms/step - loss: 1.8175 - accuracy: 0.4156 - val_l
oss: 1.1923 - val_accuracy: 0.5956
Epoch 4/15
```

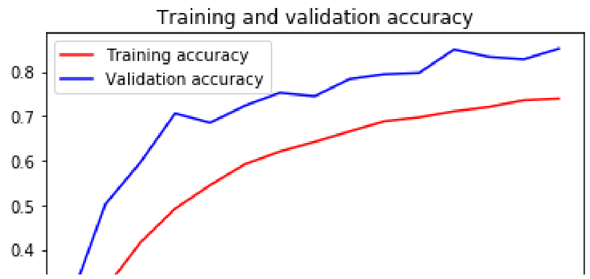
```
858/857 [==============================] - 53s 61ms/step - loss: 1.5625 - accuracy: 0.4919 - val_l
oss: 0.9312 - val_accuracy: 0.7061
Epoch 5/15
858/857 [==============================] - 51s 59ms/step - loss: 1.3909 - accuracy: 0.5447 - val_l
oss: 0.8800 - val_accuracy: 0.6850
Epoch 6/15
858/857 [==============================] - 49s 57ms/step - loss: 1.2411 - accuracy: 0.5916 - val_l
oss: 0.7438 - val_accuracy: 0.7232
Epoch 7/15
858/857 [==============================] - 52s 60ms/step - loss: 1.1463 - accuracy: 0.6203 - val_l
oss: 0.7054 - val_accuracy: 0.7522
Epoch 8/15
858/857 [==============================] - 53s 62ms/step - loss: 1.0739 - accuracy: 0.6420 - val_l
oss: 0.7162 - val_accuracy: 0.7444
Epoch 9/15
858/857 [==============================] - 51s 59ms/step - loss: 0.9999 - accuracy: 0.6655 - val_l
oss: 0.5828 - val_accuracy: 0.7832
Epoch 10/15
858/857 [==============================] - 50s 58ms/step - loss: 0.9305 - accuracy: 0.6884 - val_l
oss: 0.5676 - val_accuracy: 0.7941
Epoch 11/15
858/857 [==============================] - 49s 57ms/step - loss: 0.8995 - accuracy: 0.6972 - val_l
oss: 0.6060 - val_accuracy: 0.7971
Epoch 12/15
858/857 [==============================] - 51s 59ms/step - loss: 0.8581 - accuracy: 0.7108 - val_l
oss: 0.4588 - val_accuracy: 0.8498
Epoch 13/15
858/857 [==============================] - 52s 60ms/step - loss: 0.8288 - accuracy: 0.7206 - val_l
oss: 0.4620 - val_accuracy: 0.8331
Epoch 14/15
858/857 [==============================] - 50s 58ms/step - loss: 0.7902 - accuracy: 0.7355 - val_l
oss: 0.4901 - val_accuracy: 0.8277
Epoch 15/15
858/857 [==============================] - 51s 59ms/step - loss: 0.7693 - accuracy: 0.7392 - val_l
oss: 0.4185 - val_accuracy: 0.8515
```

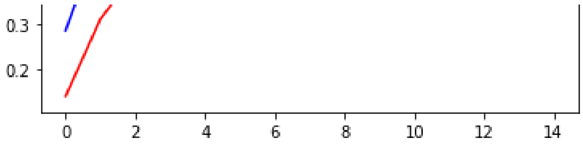
Out[4]:

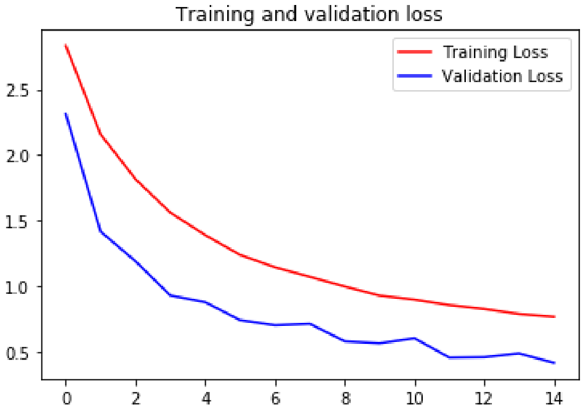
```
[189.2520849309816, 0.64891243]
```

In [5]:

```python
# Plot the chart for accuracy and loss on both training and validation
%matplotlib inline
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
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);
```