DATA.ML.200 Pattern Recognition and Machine Learning

Exercise Set 3: Convolutional Neural Network (CNN)

Miska Romppainen
H274426

March 2023

1 Count the form and number of parameters in different layers

The first task was to calculate all the parameters in the following neural network:

• The first layer is 2D convolution layer of 10 filters of the size 3 * 3 with stride 2 and ReLU activation function
• The first layer is followed by a 2 * 2 max pooling layer.
• The max pooling layer is followed by another convolutional layer with the same parameters as the first.
• The second convolutional layer is followed by another max pooling layer of the same parameters.
• The second max pooling layer is "Flattened" and followed by a full-connected (dense) layer of two neurons with sigmoid activation function.

The output size of each layer in the alternative convolutional structure for the traffic signs classification problem is as follows:
1. Input layer: (64, 64, 3)
2. First convolutional layer:
Output $= [(64-3)/2+1] * [(64-3)/2+1] * 10 = 31 * 31 * 10 = 9\ 610$
3. First max pooling layer:
Output $= 15 * 15 * 10$
4. Second convolutional layer:
Output $= [(15-3)/2+1] * [(15-3)/2+1] * 10 = 7 * 7 * 10 = 490$
5. Second max pooling layer:
Output $= 3 * 3 * 10 = 90$
6. Flattening layer:
Output $= 3 * 3 * 10 = 90$
7. Fully connected (dense) layer with 2 neurons and sigmoid activation function:
Output $= 2$

The form and number of parameters (weights) in each layer are as follows:
1. First convolutional layer:
3(kernel height)$*$3(kernel width)$*$3(number of input channels)$*$10(number of filters)+10(bias terms)= 280 parameters
2. First max pooling layer: no parameters
3. Second convolutional layer:
3(kernel height)$*$3(kernel width)$*$10(number of input channels)$*$10(number of filters)+10(bias terms)= 910 parameters
4. Second max pooling layer: no parameters
5. Flattening layer: no parameters
6. Fully connected layer:
90(number of inputs)x2(number of neurons)+2(bias terms)= 182 parameters

Therefore, the total number of parameters (weights) in the model is $280 + 910 + 182 = 1372$. This can be checked in Keras using *summary*-function. This can be seen in the image 1.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 31, 31, 10)        280

 max_pooling2d (MaxPooling2D  (None, 15, 15, 10)       0
 )

 conv2d_1 (Conv2D)           (None, 7, 7, 10)          910

 max_pooling2d_1 (MaxPooling  (None, 3, 3, 10)         0
 2D)

 flatten (Flatten)           (None, 90)                0

 dense (Dense)               (None, 2)                 182

=================================================================
Total params: 1,372
Trainable params: 1,372
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Checking the summary of the network in Keras

2 Define the network in Keras

Using the previous weeks solution of the data preparation we can save some time, since we use the same dataset. Only thing we need to do is to define the network with given attributes.

```
# Simple Sequential structure
model = tf.keras.models.Sequential()

# The first layer is 2D convolution layer of 10 filters of
# the size 3 * 3 with stride 2 and ReLU activation function.
model.add(tf.keras.layers.Conv2D(filters=10, kernel_size=(3,3),
    strides=2, activation='relu', input_shape=(64,64,3)))

# The first layer is followed by a 2 * 2 max pooling layer.
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))

# The max pooling layer is followed by another
# convolutional layer with the same parameters as the first.
model.add(tf.keras.layers.Conv2D(filters=10, kernel_size=(3,3),
    strides=2, activation='relu'))

# The second convolutional layer is followed
# by another max pooling layer of the same parameters.
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))

# The second max pooling layer is 'Flattened' and followed
# by a fullconnected (dense) layer of two neurons
# with sigmoid activation function.
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units=2, activation='softmax'))
```

Listing 1: Defining the network in Keras

From the *model.summary*(), we get an output presented in the figure 1.

3 Compile and train the net.

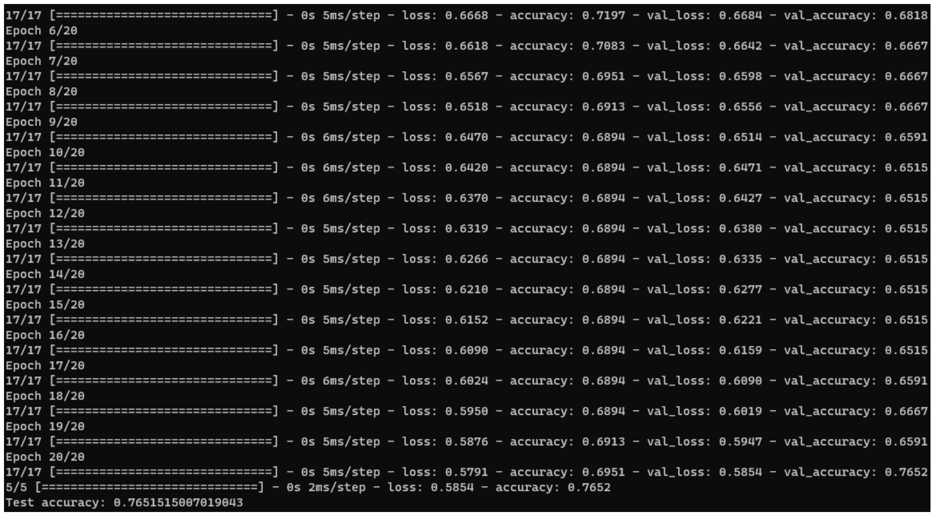
To compile and train the network we can simply use the predetermined functions *model.compile* and *model.fit*. After training the model, we can print out plots and accuracies from the training, with the presented code we get the output presented in images 2 and 3. The test accuracy is around 78%. We can see from the figure 3, we could train the network even more, and get even better accuracy, since the figures curve is not settling into any values yet.

```
# Compile the model
model.compile(optimizer='SGD', loss='binary_crossentropy', metrics
    =['accuracy'])

# Train the model
history = model.fit(x_train, y_train, batch_size=32, epochs=20,
    validation_data=(x_test, y_test))
plt.plot(history.history['loss'])

# Evaluate the model on test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
plt.show()
```

Listing 2: Compling and evaluating the network



```
17/17 [==============================] - 0s 5ms/step - loss: 0.6668 - accuracy: 0.7197 - val_loss: 0.6684 - val_accuracy: 0.6818
Epoch 6/20
17/17 [==============================] - 0s 5ms/step - loss: 0.6618 - accuracy: 0.7083 - val_loss: 0.6642 - val_accuracy: 0.6667
Epoch 7/20
17/17 [==============================] - 0s 5ms/step - loss: 0.6567 - accuracy: 0.6951 - val_loss: 0.6598 - val_accuracy: 0.6667
Epoch 8/20
17/17 [==============================] - 0s 5ms/step - loss: 0.6518 - accuracy: 0.6913 - val_loss: 0.6556 - val_accuracy: 0.6667
Epoch 9/20
17/17 [==============================] - 0s 6ms/step - loss: 0.6470 - accuracy: 0.6894 - val_loss: 0.6514 - val_accuracy: 0.6591
Epoch 10/20
17/17 [==============================] - 0s 6ms/step - loss: 0.6420 - accuracy: 0.6894 - val_loss: 0.6471 - val_accuracy: 0.6515
Epoch 11/20
17/17 [==============================] - 0s 6ms/step - loss: 0.6370 - accuracy: 0.6894 - val_loss: 0.6427 - val_accuracy: 0.6515
Epoch 12/20
17/17 [==============================] - 0s 5ms/step - loss: 0.6319 - accuracy: 0.6894 - val_loss: 0.6380 - val_accuracy: 0.6515
Epoch 13/20
17/17 [==============================] - 0s 5ms/step - loss: 0.6266 - accuracy: 0.6894 - val_loss: 0.6335 - val_accuracy: 0.6515
Epoch 14/20
17/17 [==============================] - 0s 5ms/step - loss: 0.6210 - accuracy: 0.6894 - val_loss: 0.6277 - val_accuracy: 0.6515
Epoch 15/20
17/17 [==============================] - 0s 5ms/step - loss: 0.6152 - accuracy: 0.6894 - val_loss: 0.6221 - val_accuracy: 0.6515
Epoch 16/20
17/17 [==============================] - 0s 5ms/step - loss: 0.6090 - accuracy: 0.6894 - val_loss: 0.6159 - val_accuracy: 0.6515
Epoch 17/20
17/17 [==============================] - 0s 6ms/step - loss: 0.6024 - accuracy: 0.6894 - val_loss: 0.6090 - val_accuracy: 0.6591
Epoch 18/20
17/17 [==============================] - 0s 5ms/step - loss: 0.5950 - accuracy: 0.6894 - val_loss: 0.6019 - val_accuracy: 0.6667
Epoch 19/20
17/17 [==============================] - 0s 5ms/step - loss: 0.5876 - accuracy: 0.6913 - val_loss: 0.5947 - val_accuracy: 0.6591
Epoch 20/20
17/17 [==============================] - 0s 5ms/step - loss: 0.5791 - accuracy: 0.6951 - val_loss: 0.5854 - val_accuracy: 0.7652
5/5 [==============================] - 0s 2ms/step - loss: 0.5854 - accuracy: 0.7652
Test accuracy: 0.7651515007019043
```

Figure 2: Evaluation of the training and test accuracy

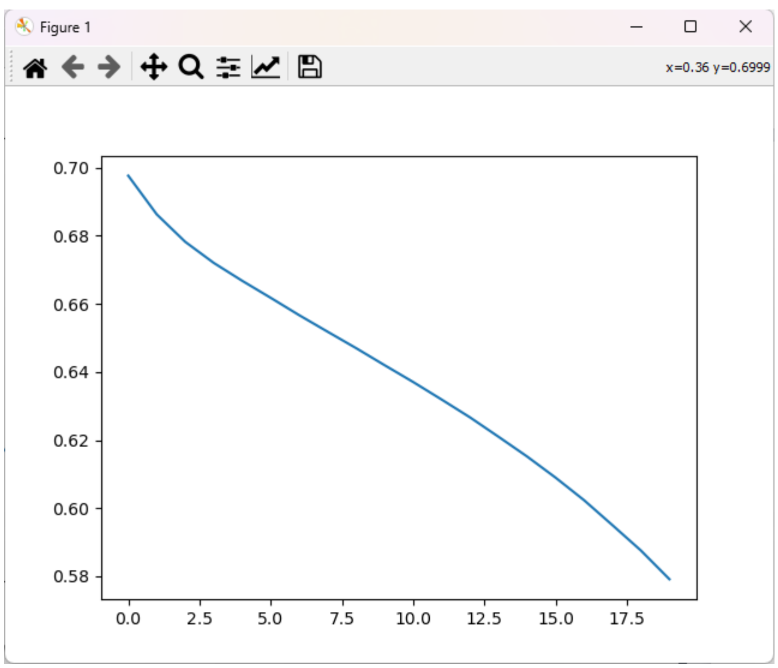


Figure 3: Plot of the "loss" from the training