

DATA.ML.200 Pattern Recognition and Machine Learning

Exercise Set 2: Multi-layer Perceptron (MLP)

Miska Romppainen
H274426

March 2023

1 Count the number of parameters in a neural network

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

- The input is “flattened” $3 \times 64 \times 64$ -dimensional
- On the 1st layer there are 100 nodes
- On the 2nd layer there are 100 nodes
- On the 3rd (output) layer there are 10 nodes

The second part of the task was to use the old rule of thumb that states that the number of training samples should be at least 5 times the number of coefficients and compute the desired sample size based on this rule for the first specified neural network. The task 1 written by hand can be seen in the image 2.

First, we can define connections for each node in the first layer. The first layer gets ($3 \times 64 \times 64 = 12888$) flattened image as input and the flattened image is fully connected to each node, therefore each node has $12888 + \textit{bias} = 12889$ weights, so the layer has $12889 \times 100 = 1288900$ parameters in total.

The second layer has 100 nodes fully connected to the first 100 nodes, therefore each node in the second layer has $100 + \textit{bias} = 101$ weights, so the layer has $101 \times 100 = 10100$ weights in total.

The third (final) layer has 10 nodes which are fully connected to the second layer, so each node has $100 + \textit{bias} = 101$ weights, and the layer has $101 \times 10 = 1010$ weights in total.

By summing all the total weights from each layer we get the total amount of the

parameters in the neural network. This comes out to be $1288900 + 10100 + 1010 = 1240010$ parameters in total. To sanity-check, we can use Keras to check if our output was correct, image 1.

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 12288)	0
dense (Dense)	(None, 100)	1228900
dense_1 (Dense)	(None, 100)	10100
dense_2 (Dense)	(None, 10)	1010

```

Total params: 1,240,010
Trainable params: 1,240,010
Non-trainable params: 0

```

Figure 1: Model in keras

The rule of thumb states that the number of training samples should be at least 5 times the number of coefficients. However, the coefficients can be seen as the number of nodes in each layer, since a coefficient is defined as "a numerical or constant quantity placed before and multiplying the variable in an algebraic expression" and the variable can be seen as the amount of weights in each node - using this the sum of coefficients comes out to be $100 + 100 + 10 = 210$, by applying the rule of thumb, we get $210 * 5 = 1050$ training samples. Which seems to be extremely little training points for a 1240010 parameter neural network. In the question, with the word "coefficients", what I think was meant was the amount of "parameters", if we apply the rule of thumb to the amount of parameters we get $1240010 * 5 = 6200050$.

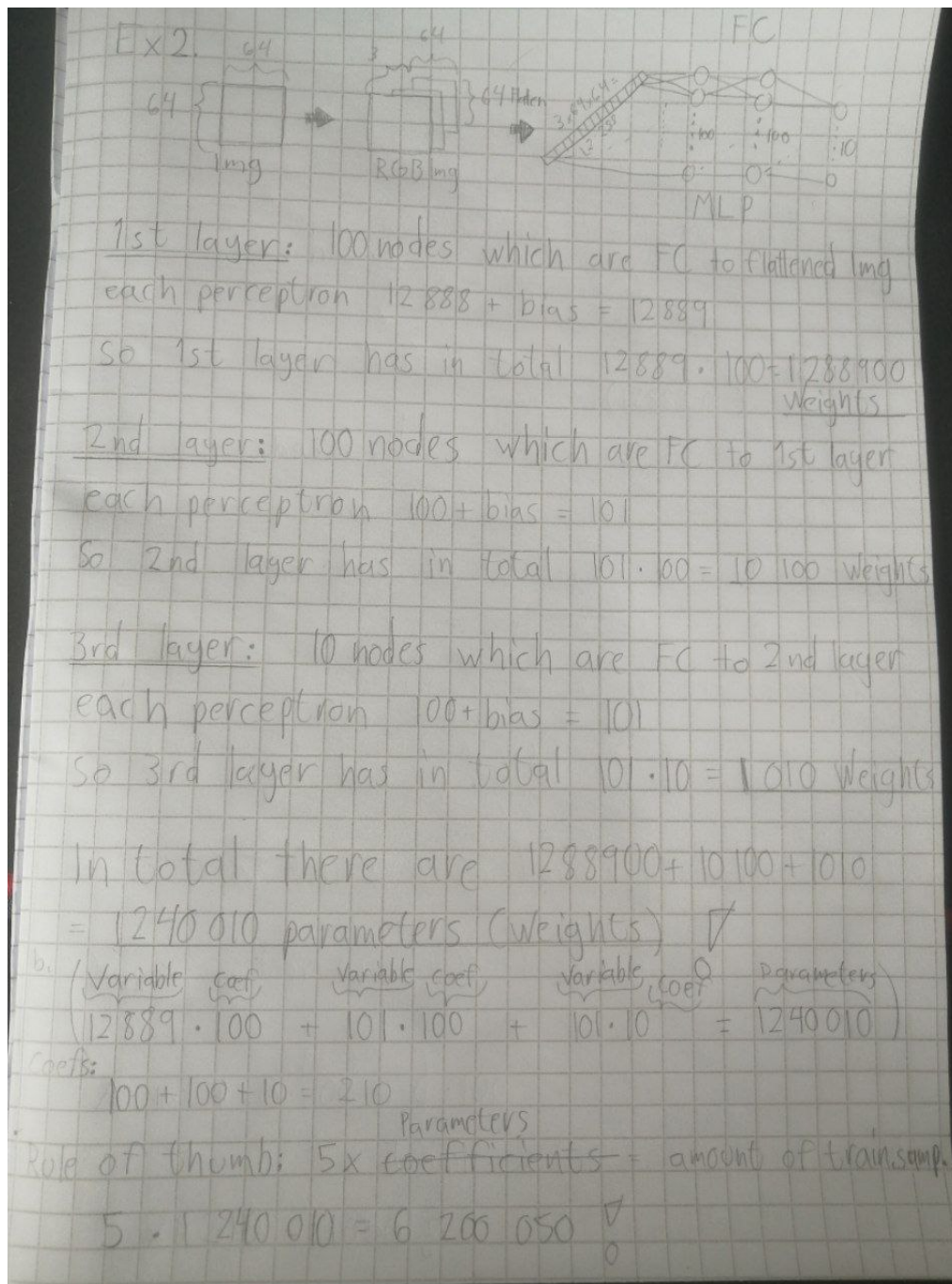


Figure 2: Task 1 on pen and paper

2 Load Traffic sign data for deep neural network processing

We start by downloading the dataset from the moodle page (Image 3).

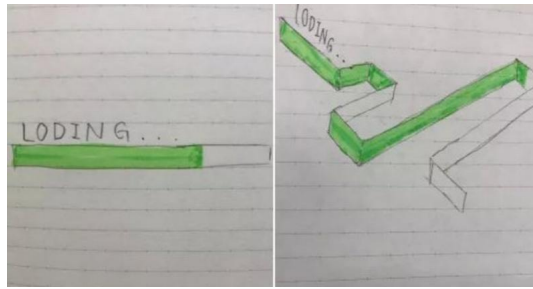


Figure 3: Downloading

After downloading the data we need to prepare the data in the program. First we need to give each image in the folders "class1" and "class2" labels to represent their specific class. We fetch the images from the folders, label them and put the image data in the numpy array and label array to the corresponding order. After preparing the datasets, we normalize the brightness values between (0,1) to make them more suitable for neural networks (same as in the lecture notebook). By using *train_test_split* function from *sklearn* we can split the dataset 80% for training and 20% testing, after splitting we shuffle the datasets.

```
1 # Load the dataset
2 data = []; labels = []; classes = 2
3 for i in range(classes):
4     class_path = f'class{i+1}'
5     img_path = os.path.join(images, class_path)
6     for img in os.listdir(img_path):
7         im = Image.open(img_path + '/' + img)
8         im = np.array(im)
9         data.append(im)
10        labels.append(i)
11
12 data = np.array(data); labels = np.array(labels)
13
14 x = data.astype('float32') / 255 # Normalizing data
15 y = np.array(labels)
16
17 # Split data into two parts - 80% for training and 20% for testing
18 x_train, x_test, y_train, y_test = train_test_split(x, y,
19                                                    train_size=0.8, test_size=0.2, shuffle=True)
20
21 # Preprocess the data
22 y_train = tf.keras.utils.to_categorical(y_train)
23 y_test = tf.keras.utils.to_categorical(y_test)
```

Listing 1: Loading the data set and splitting the dataset 80/20.

After defining the splits, we get *shape* outputs of:

```
Training data  
(528, 64, 64, 3)
```

Figure 4: Training set output in console (*x_train.shape*)

```
Test data  
(132, 64, 64, 3)
```

Figure 5: Testing set output in console (*x_test.shape*)

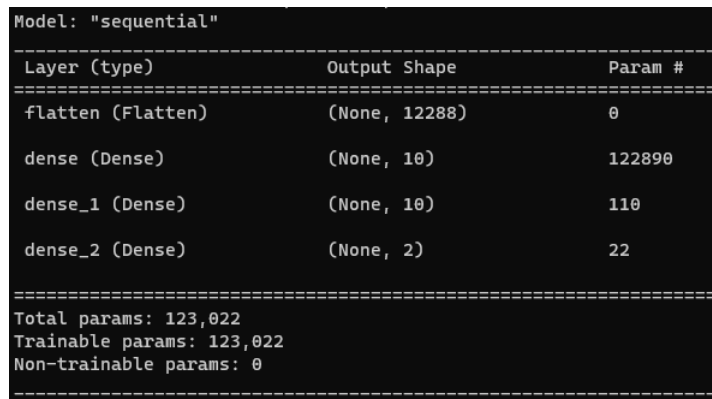
3 Define the network in Keras

To define the network from task 1 in Keras, we first have to dictate the structure of the network, I chose to use the simple Sequential structure, from which Joni provided an example on lecture. In the assignment paper, it is defined that "You may in the beginning reduce the number of neurons from 100 to 10 in the two layers", which is why there are only 10 neurons in each layer, instead of 100. And since there are only two classes to classify between in the dataset, there cannot be 10 possible outputs, but instead 2. The functions used were "ReLU" and "softmax" since its an classification problem.

```
1 # Simple Sequential structure
2 model = tf.keras.models.Sequential()
3
4 # Flatten 2D input image to a 1D vector; size of 12288
5 model.add(tf.keras.layers.Flatten(input_shape=(64,64,3)))
6 print(model.output_shape)
7
8 # Add a single layer of 10 neurons each connected to each input
9 model.add(tf.keras.layers.Dense(10,activation='relu'))
10 print(model.output_shape)
11
12 # Add a single layer of 10 neurons each connected to each input
13 model.add(tf.keras.layers.Dense(10,activation='relu'))
14 print(model.output_shape)
15
16 # Add a single layer of 2 neurons each connected to each input
17 model.add(tf.keras.layers.Dense(2,activation='softmax'))
18 print(model.output_shape)
19
20 print(model.summary())
```

Listing 2: Defining the network in Keras

From the *model.summary()*, we get an output presented in the figure 6.



Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 12288)	0
dense (Dense)	(None, 10)	122890
dense_1 (Dense)	(None, 10)	110
dense_2 (Dense)	(None, 2)	22

=====
Total params: 123,022
Trainable params: 123,022
Non-trainable params: 0
=====

Figure 6: Downloading

4 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 7 and 8. The test accuracy is almost 98%.

```
1 # Compile the model
2 model.compile(optimizer='sgd', loss='categorical_crossentropy',
3               metrics=['accuracy'])
4
5 # Train the model
6 history = model.fit(x_train, y_train, epochs=10)
7 plt.plot(history.history['loss'])
8
9 # Evaluate the model on the test set
10 test_loss, test_acc = model.evaluate(x_test, y_test)
11 print('Test accuracy:', test_acc)
12 plt.show()
```

Listing 3: Compiling and evaluating the network

```

Epoch 1/10
17/17 [=====] - 0s 2ms/step - loss: 0.6040 - accuracy: 0.6894
Epoch 2/10
17/17 [=====] - 0s 2ms/step - loss: 0.4420 - accuracy: 0.8561
Epoch 3/10
17/17 [=====] - 0s 2ms/step - loss: 0.3317 - accuracy: 0.9261
Epoch 4/10
17/17 [=====] - 0s 2ms/step - loss: 0.2356 - accuracy: 0.9640
Epoch 5/10
17/17 [=====] - 0s 2ms/step - loss: 0.1803 - accuracy: 0.9792
Epoch 6/10
17/17 [=====] - 0s 2ms/step - loss: 0.1403 - accuracy: 0.9943
Epoch 7/10
17/17 [=====] - 0s 2ms/step - loss: 0.1123 - accuracy: 0.9962
Epoch 8/10
17/17 [=====] - 0s 2ms/step - loss: 0.0961 - accuracy: 0.9943
Epoch 9/10
17/17 [=====] - 0s 1ms/step - loss: 0.0777 - accuracy: 0.9981
Epoch 10/10
17/17 [=====] - 0s 1ms/step - loss: 0.0686 - accuracy: 0.9981
5/5 [=====] - 0s 1ms/step - loss: 0.0757 - accuracy: 0.9773
Test accuracy: 0.9772727489471436

```

Figure 7: Evaluation of the training and test accuracy

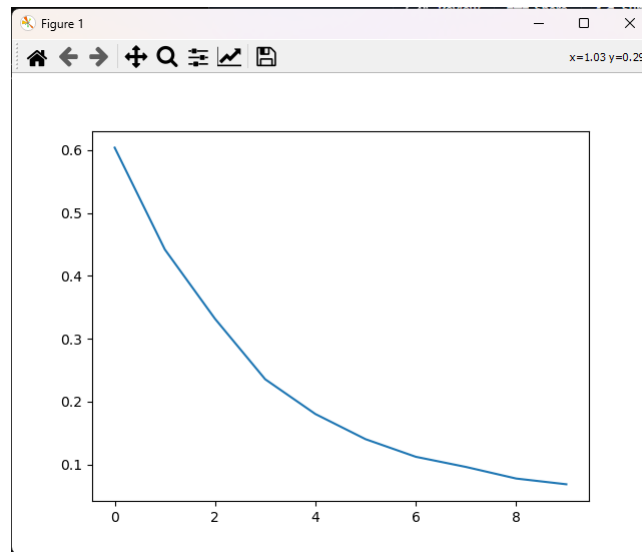


Figure 8: Plot of the "loss" from the training

The reason why each "Epoch" only processes 17 out of 17, is because since there are only 528 input images, each batch tries to be as close to 32 as possible, since its the default batch size. $528/32 = 16, 5\ 17$ which is why on the last epoch it processes extra 5 sets.

And here to conclude the second exercise...

My model on training data



My model on test dataset



Figure 9: We all been there...