Machine Learning in Healthcare

Homework 4
Deep Learning

Theoretical question

Hadassa Malka and Sarah Buzaglo

PART 1: Fully connected layers

ReLU vs. tanh activation function

Activation functions are applied after each node (neuron, sometimes a group of neurons, from a specific layer) of a neural network to determine what will be the input of the nodes (neurons) of the next layer.

We first used the ReLU activation function. Although it has the advantage of allowing a fast convergence of the network as well as having a depth and allowing backpropagation to learn (unlike linear activations), when we have a close-to-zero/negative inputs, the gradient of the function tends/is equal to zero and the learning process is small/not possible.

We then chose to use the **tanh function**. It has the advantage of being non-linear and zero-centered, with values between -1 and 1 (therefore it is easier to map inputs with high negative/positive/neutral values), allowing the training process to be less ‘stuck’ when small or negative values are received compared to ReLU. By contrast, tanh is more expensive computationally. Also, it is subject to the ‘vanishing gradient’ issue (since the output is bounded between -1 and 1, with the optimization process the gradient will be smaller and smaller, making the convergence slower and slower if it converges at all), to which the ReLU activation is less sensitive.

Source to support the explanation: https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/

With 25 epochs, we received the following results with the different activation functions:

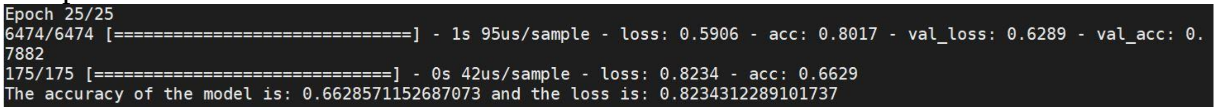
	ReLU	tanh
Accuracy	0.6914	0.6629
Loss	0.8160	0.8234

We can notice that we received better performance scores using the ReLU activation function.

25 vs. 40 epochs

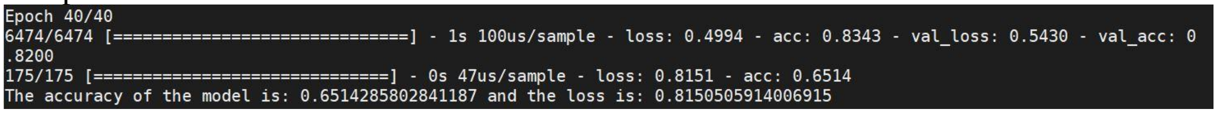
The number of epochs is defined by the number of times the learning algorithm will run through the whole train set. The more we train, the more we will improve the performance **but** a too high number of training steps can lead to an overfit (on the training set) and a too low number of training steps can lead to an underfit.
In our case, 25 epochs give better results. Adding 15 other epochs leads to a reduction of the performance scores (due to overfitting, it has increased the performance on the training set but reduced it on the test set).

25 epochs:



```
Epoch 25/25
6474/6474 [==============================] - 1s 95us/sample - loss: 0.5906 - acc: 0.8017 - val_loss: 0.6289 - val_acc: 0.
7882
175/175 [==============================] - 0s 42us/sample - loss: 0.8234 - acc: 0.6629
The accuracy of the model is: 0.6628571152687073 and the loss is: 0.8234312289101737
```

40 epochs:



```
Epoch 40/40
6474/6474 [==============================] - 1s 100us/sample - loss: 0.4994 - acc: 0.8343 - val_loss: 0.5430 - val_acc: 0
.8200
175/175 [==============================] - 0s 47us/sample - loss: 0.8151 - acc: 0.6514
The accuracy of the model is: 0.6514285802841187 and the loss is: 0.8150505914006915
```

	25 epochs	40 epochs
Accuracy	0.6628	0.6514
Loss	0.8234	0.8151

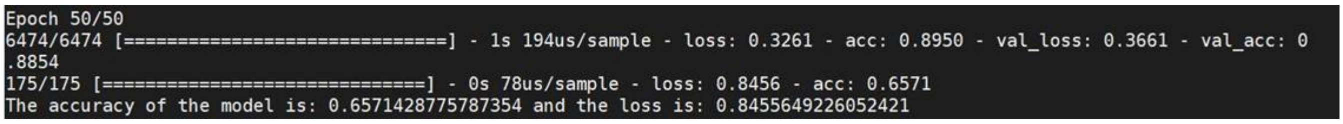
Mini-batch vs. SGD

Stochastic gradient descent (SGD) is an algorithm similar to the gradient descent algorithm with the particularity that it updates the model **for each example** from the training dataset (instead of using all the training set at once for training).
Mini-batch is also an algorithm similar to the gradient descent algorithm, but his particularity is to update the model **using batches** (group of examples) from the training dataset (instead of using all the training set at once for training).

The main advantages of mini-batch over SGD are:

- Better computational efficiency: for very large dataset, the training process with SGD will be very slow due the very frequent updates of the model (at each example), and will be more efficient by grouping examples.
- Less noisy learning process: since the updates are less frequent, we have a smallest variance (less ‘jumps’) in the parameters/errors updates other the training process, and the convergence is therefore more stable than in SGD (avoid local minima).

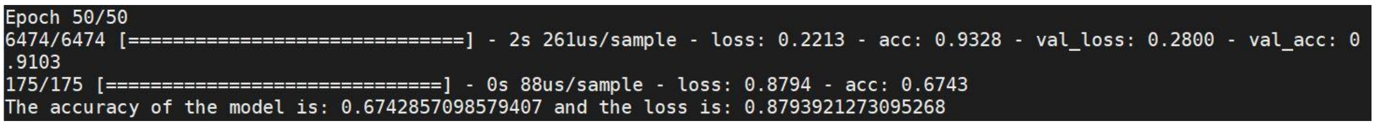
In this section, we needed to reduce the batch size from 64 to 32 for the ReLU model, and we got the following results:



```
Epoch 50/50
6474/6474 [==============================] - 1s 194us/sample - loss: 0.3261 - acc: 0.8950 - val_loss: 0.3661 - val_acc: 0
.8854
175/175 [==============================] - 0s 78us/sample - loss: 0.8456 - acc: 0.6571
The accuracy of the model is: 0.6571428775787354 and the loss is: 0.8455649226052421
```

Batch normalization layers vs. none

After adding the batch normalization layers (we also reduced the batch size and increased the number of epochs for new_a_model, compared to its first run), it seems that the batch normalization improved a little bit the model's performance: the accuracy increased from 0.66 to 0.67. Since the quality of the data is low, we can deduce that even batch normalization cannot significantly if at all increase the model performance.



```
Epoch 50/50
6474/6474 [==============================] - 2s 261us/sample - loss: 0.2213 - acc: 0.9328 - val_loss: 0.2800 - val_acc: 0
.9103
175/175 [==============================] - 0s 88us/sample - loss: 0.8794 - acc: 0.6743
The accuracy of the model is: 0.6742857098579407 and the loss is: 0.8793921273095268
```

PART 2: CNN

Task 1: 2D CNN

1) The Convolutional Neural Network has 8 layers: 5 convolutional layers, 2 fully connected layers and one output (fully connected) layer.

2) The convolutional layers have respectively 64, 128, 128, 256, 256 filters.

3) The number of parameters of a Convolutional Neural Network (CNN) would not be similar to a Fully Connected Neural Network. In a Fully Connected Neural Network, every neuron is connected to every other neuron in the previous layer which is very expensive in term of quantity of parameters (weights). In a CNN, each neuron is connected only to a few nearby (local) neurons from the previous layers. So, a CNN allows us to significantly reduce the number of parameters.

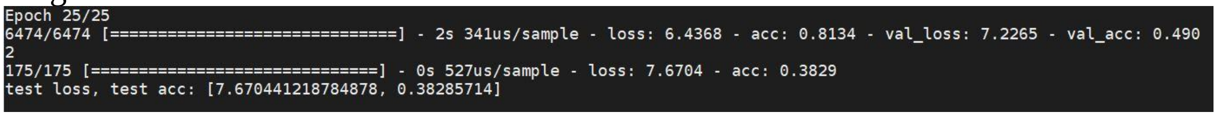
4) This specific NN does perform regularization: using L2 and Dropout.

Task 2: Reducing the number of filters

	Original Network	New Network (less filters)
Accuracy	0.3829	0.2686
Loss	7.6704	4.8235

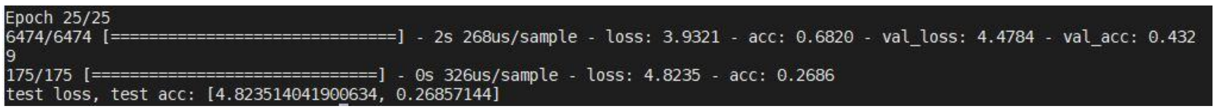
We can notice that reducing by half the number of filters in the convolutional layers decreased the loss but also decreased the accuracy of the model (after running again the model a few times, we obtain various results due to the random initialization of the weights, but overall we always get very low performance scores. We can attribute that to the low quality of the images received in our data set, illustrating the quote ‘garbage in – garbage out’).

Original:



```
Epoch 25/25
6474/6474 [==============================] - 2s 341us/sample - loss: 6.4368 - acc: 0.8134 - val_loss: 7.2265 - val_acc: 0.490
2
175/175 [==============================] - 0s 527us/sample - loss: 7.6704 - acc: 0.3829
test loss, test acc: [7.670441218784878, 0.38285714]
```

New:



```
Epoch 25/25
6474/6474 [==============================] - 2s 268us/sample - loss: 3.9321 - acc: 0.6820 - val_loss: 4.4784 - val_acc: 0.432
9
175/175 [==============================] - 0s 326us/sample - loss: 4.8235 - acc: 0.2686
test loss, test acc: [4.823514041900634, 0.26857144]
```