BME 336546 - HW4

X-ray images classification

Dekel & Nathan

March 3, 2021

1 Fully Connected Layers

1.1 NN with fully connected layers

The accuracy and loss over both the training and testing sets for the first model with ReLu activation are plotted below. The appropriate loss for this multi-class task is categorical cross-entropy.

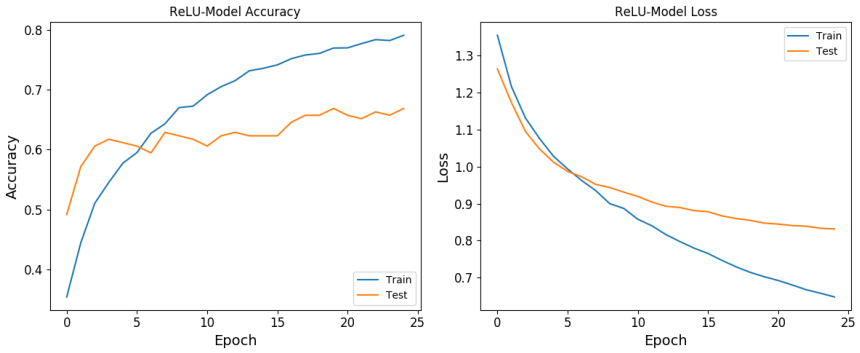

Figure 1: Accuracy and loss against the number of epochs

Results of the evaluation on the test set:

Loss	Accuracy
0.8316	0.6686

Table 1: Accuracy and loss over the test set

1.2 Activation Functions

The activation function determines the output of a neuron given its weighted sum of inputs. In our case, ReLU (Rectified Linear Unit) outputs 0 for any non-positive input, and outputs the input for any positive input, as shown by the following graph:

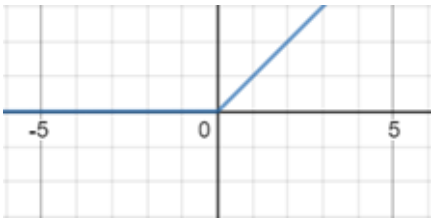

Figure 2: The ReLU function

We decided to change the activation function to a tanh activation function (hyperbolic tangent), which maps the input between -1 and 1, where very negative inputs are (asymptotically) output as -1 and very positive inputs are output as 1, as shown by the following graph:

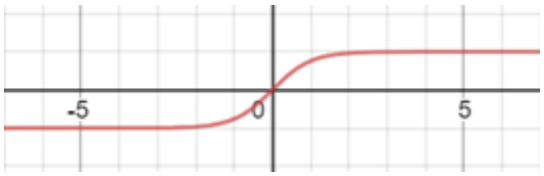

Figure 3: The Tanh function

Tanh is preferred over sigmoid due to its property of centering the data at zero. Nowadays, ReLU is the most extensively used, as both tanh and sigmoid saturate for large inputs, preventing the weights from being updated. Our intuition is that with our normalized data (values between 0 and 1), this shortcoming of tanh should not be an important limitation, as its inputs will be of small magnitude, avoiding saturation.

1.3 Number of epochs

An epoch refers to a single pass through the whole training dataset. As we increase the number of epochs, we reduce the error on the training data, but increasing the number of epochs too much can result in overfitting. Now we checked for the effect the number of epochs has on our NN. We used the NN with the tanh activation function and checked for 25 epochs vs. 40 epochs.

25 epochs

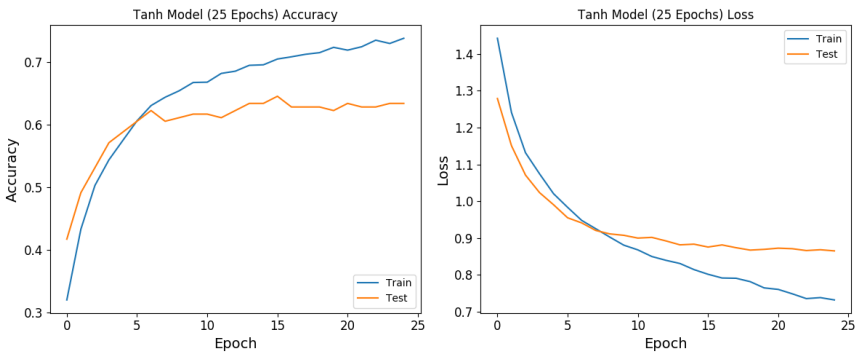

Figure 4: Accuracy and loss against the number of epochs

Results of the evaluation on the test set:

Loss	Accuracy
0.8491	0.6629

Table 2: Accuracy and loss over the test set

40 epochs

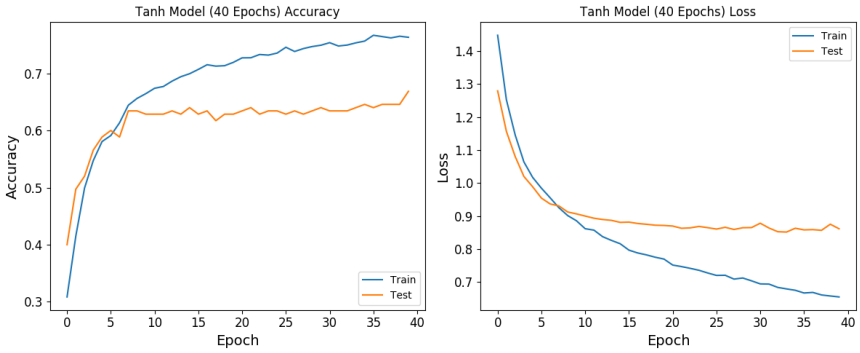

Figure 5: Accuracy and loss against the number of epochs

Results of the evaluation on the test set:

Loss	Accuracy
0.8411	0.6457

Table 3: Accuracy and lost over the test set

First, we don't see saturation effects on the figures above: the algorithm keeps learning.
We can notice that both the accuracy and loss over the testing set had better results for 40 epochs, but not by a lot. In Figure 3, we see that the curves for the test set reach a plateau. The small change in results (and the generally low accuracy) probably has to do with the low resolution of our sample images, leading to a low ceiling for performance.

Comparing Table 1 (NN with ReLU activation, 25 epochs) and Table 2 (NN with tanh activation, 25 epochs) shows a difference of accuracy of 0.6%.

1.4 Mini-Batches

There are three variants of gradient descent: BGD, SGD, and Mini-Batch Gradient Descent. They differ in how many examples are used to compute the gradient of the cost function.
Let's compare mini-batch (the gradient of the cost function is computed for a subset of the training set) with SGD (the gradient of the cost function is computed for each training example). The advantages of SGD over mini-batch gradient descent is that SGD has a better computational speed and uses less memory than mini-batch. The advantages of mini-batch is that the weights updates are more accurate. Moreover, in the long run, mini-batch will converge to a more refined estimate of the minimum, while SGD may fluctuate around it.

Therefore, we can see that there may be a "sweet spot" where we pick the "best" batch size for our data that can be fast computationally, but also provide us with accurate enough results.

Now we checked the performance of the model after reducing the batch size to 32 (from 64 in the first model we built – Task I) and doubling the number of epochs. Here are our results:

50 epochs

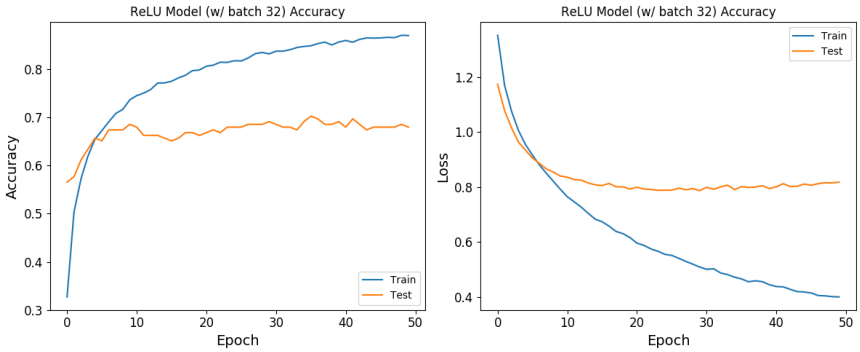

Figure 6: Accuracy and loss against the number of epochs

Results of the evaluation on the test set:

Loss	Accuracy
0.8213	0.6743

Table 4: Accuracy and loss over the test set

Again, results from Table 5 are very similar of those of our first model shown in Table 1.

25 epochs

We also show the results for a same number of epochs as in Task I (25) and a batch size of 32 examples:

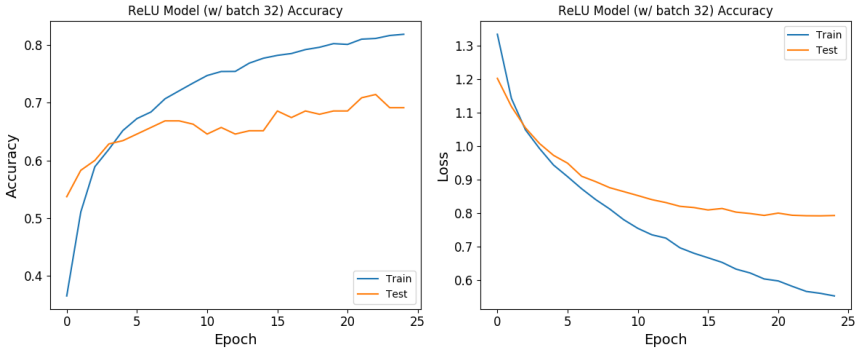

Figure 7: Accuracy and loss against the number of epochs

Comparing the graph of the training loss in Figure 7 and Figure 1, we observe that dividing by two the batch size does make the convergence slightly faster.

Results of the evaluation on the test set:

Loss	Accuracy
0.7927	0.6914

Table 5: Accuracy and loss over the test set

1.5 Batch Normalization

Batch normalization is the process in which we normalize the activation at each layer. Very often it improves the learning speed (as gradient descent is sensitive to scaling) and outcome. Now, we checked the performance of the tanh activation function model with batch normalization.

50 epochs

Here are our results for 50 epochs:

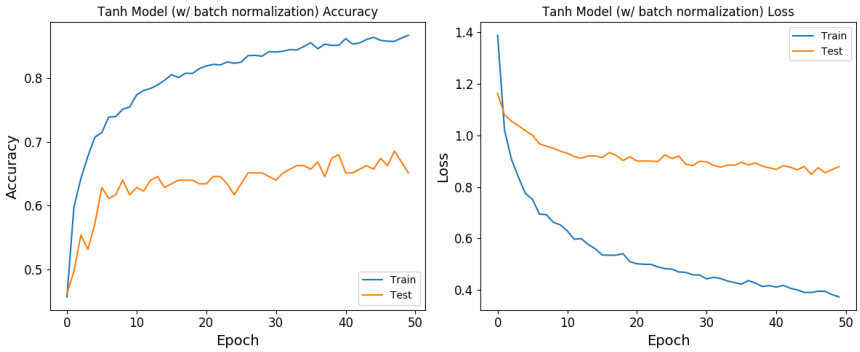

Figure 8: Accuracy and loss against the number of epochs

Results of the evaluation on the test set:

Loss	Accuracy
0.8789	0.6514

Table 6: Accuracy and loss over the test set

25 epochs

Let's zoom again on the first 25 epochs to see whether the learning speed has been improved by batch normalization.

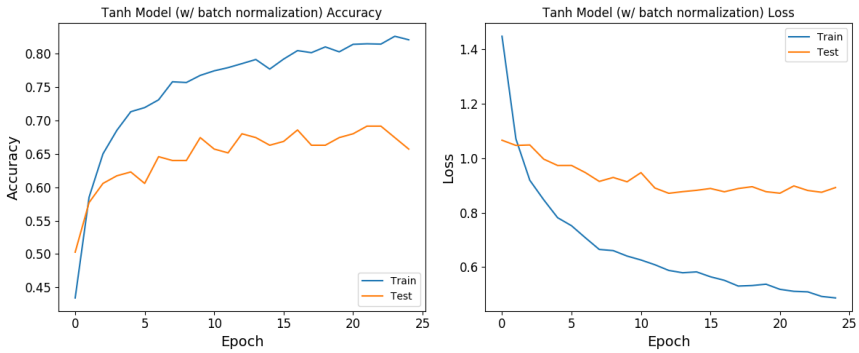

Figure 9: Accuracy and loss against the number of epochs

Comparing Figure 9 and Figure 4 shows that the decrease of the train loss is faster when we perform batch normalization. But we see no improve in the final performance on the test set after 25 epochs.

Results of the evaluation on the test set:

Loss	Accuracy
0.8919	0.6571

Table 7: Accuracy and loss over the test set

Conclusion

To sum up, we investigated the influence of the activation function, the batch size, the number of epochs and of the batch normalization on the performances of our fully-connected neural network. Both ReLU and tanh activations lead to equivalent performances on the test set. Increasing the number of epochs proved to be useless. Although reducing the batch size and performing batch normalization speeds up the decrease of the train loss, and therefore the learning, it leads to similar performances after 25 epochs. It leads us to believe that due to the nature of the samples (low resolution images) there is a low ceiling for accuracy and loss (around 0.65-0.70 for accuracy, and 0.8-0.9 for loss).

2 Convolutional Neural Network (CNN)

2.1 2D CNN

How many layers?

If we define a layer like the method *summary* does, there are:

- 5 2D convolutional layers,
- 3 pooling layers,
- 5 batch normalization layers,
- 6 dropout layers,

- 1 permutation layer,
- 1 flatten layer,
- 3 fully connected layers.

Overall, 24 layers.

If we talk about the hidden layers, there are 5 convolutional layers and 3 fully-connected layers, so **8 layers**.

How many filter in each layer?

The filters are in the 2D convolution layers:

- $Conv2D - 1$: 64 filters,
- $Conv2D - 2$: 128 filters,
- $Conv2D - 3$: 128 filters,
- $Conv2D - 4$: 256 filters,
- $Conv2D - 5$: 256 filters,

Would the number of parameters be similar to a fully connected NN?

No. We usually use CNN to reduce the number of parameters to learn compared to a fully connected NN. The 5 convolutional layers require to learn 1.1M parameters, as there are many filters at each layer. For instance, if instead of this, we use 5 fully-connected layers of 512 neurons each, we would have to learn 1.6M parameters.

Is this specific NN performing regularization?

Yes. In every 2D convolution layer there is kernel regularization (L2). Its purpose is to regularize the kernel's weight matrix (using the L2 norm in our case). And optionally, *dropout* can be added. In such a case, at each iteration during training, only a fraction of the weights is learned.

2.2 Number of filters

After changing the number of filters (halving them in each Conv2D layer), we compared the results:

Number of filters	Loss	Accuracy
[64, 128, 128, 256, 256]	7.96	0.38
[32, 64, 64, 128, 128]	5.03	0.25

Table 8: Accuracy and loss over the test set, after 25 epochs

There is a net decrease in accuracy. We should not halve the number of filters.