ML in Healthcare

HW4: X-ray images classification

Leen Ileimi, 208809715

Taima Zoabi, 318476868

Part 1

Task 2: *Activation functions*

The activation function transforms the weighted sum of the inputs into the activation of the node, the proposed activation functions are nonlinear which means that they can learn complex mapping functions. The sigmoid is especially used for models where we have to predict the probability as an output, because the result is mapped in between 0 to 1. The tanh function is mainly used in classification between two classes. It's like sigmoid but better. The range of the tanh function is from (-1 to 1). The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph. Because we have multiclass situation with 4 different classes we thought that the LeakyRelu is the most suitable function cause it's like Relu yet better It is an attempt to solve the dying Relu problem, while Relu activation function turns the negative value into zero it turns it to a slightly negative values. Therefore, we think that we will get a better results, with lower loss and higher accuracy.

Task 3: *Number of epochs.*

As the number of epochs increases, more number of times the weight are changed in the neural network and the curve goes from underfitting (bad performance) to optimal (good performance) to overfitting (bad performance) curve. As we can see we got a better performance with a higher number of epochs but it does not necessarily mean the higher the number is the better the performance it becomes. If we compare the results with Relu model we can see that the performance didn't really improve like we had hoped for, this could be because the data isn't particularly good and we handled it as a multiclass situation and not multilabel.

Task 4: *Mini-batches.*

In Stochastic Gradient Descent (SGD), we consider just one example at a time to take a single step. It can be used for larger datasets. It converges faster when the dataset is large as it causes updates to the parameters more frequently. However, since we are considering just one example at a time the cost will fluctuate over the training examples and it will not necessarily decrease. In mini-batches, neither we use all the dataset all at once nor we use the single example at a time(SGD). We use a batch of a fixed number of training examples which is less than the actual dataset and call it a mini-batch. Doing this helps us achieve the advantages of SGD and still converge relatively faster to minima due to fewer

updates to the model which makes it more computationally efficient. As for the performance there is no much of a change.

Task 4: *Batch normalization.*

It consists of normalizing the mean and variance of each of the features at every level of representation during training. Which enables using higher learning rates thus enables to accelerate the learning process. As we can see in the results we got, the performance didn't really improve after adding the batch normalization, as we explained earlier it could be because the data is not so good and our models couldn't get the job done perfectly.

Part 2

Task 1: *2D CNN.*

- As can be seen in the code, the model has 8 layers. 5 of which are convolution layers, and 3 fully connected layers.
- In the convolution layers:
  first layer has 64 filters
  second layer: 128 filters
  third layer: 128 filters
  fourth layer: 256 filters
  fifth layer: 256 filters
- No, the number of parameters is not similar to a fully connected network. It is smaller. And basically, this is one of the main reasons in using CNN: it enables us to reduce the number of parameters by orders of magnitudes.
- Yes, this NN performs regularization in the convolution layers, of type L2.

Task 2: *Number of filters.*

As the number of filters was reduced, the loss was reduced and the accuracy almost did not change (reduced a bit). The reason may be that, as was mentioned before, the data is quite bad, so in both cases the performance was bad. The decrease in the loss may be explained by overfitting: using too many filters may have caused the model to overfit the data. So, reducing the number of filters reduced the loss.