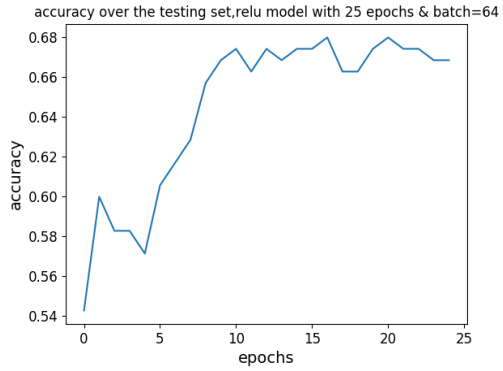
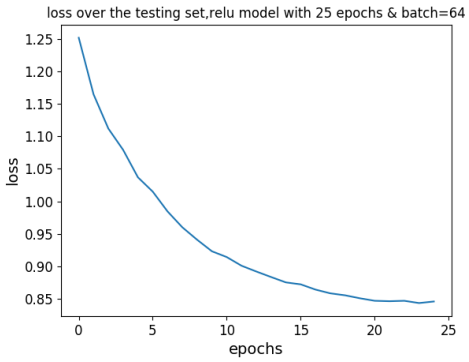
HW4, ML in Healthcare – 336546

Edited by **Hadas Ben-Atya & Nathan Berdugo**

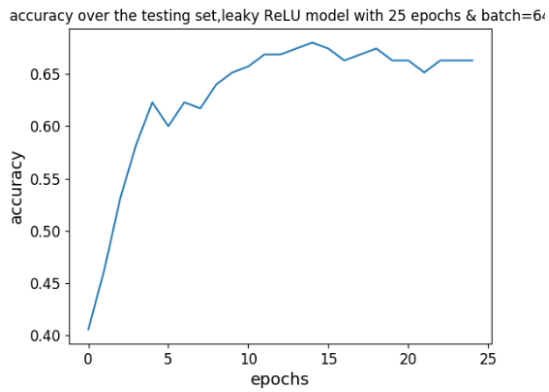
Task 1:

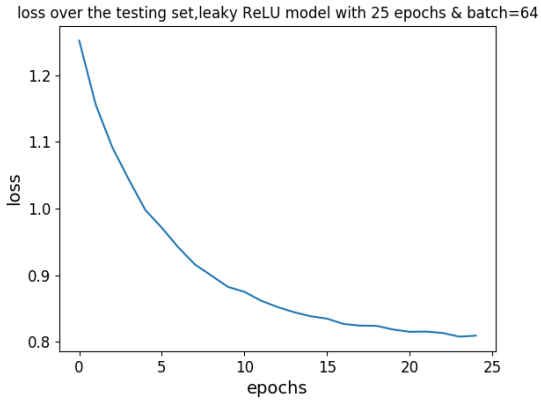




Task 2: *Activation functions.*

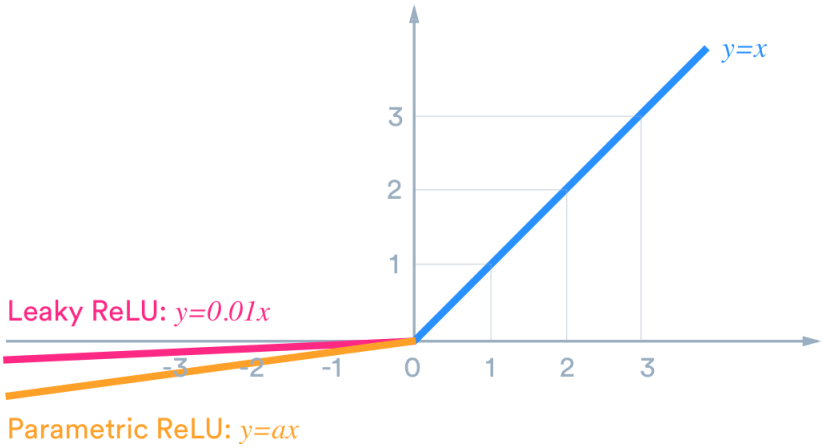
Change the activation functions to LeakyRelu or tanh or sigmoid. Name the new model new_a_model. Explain how it can affect the model.*





We changed the activation function to Leaky- ReLU:

$$LRelu = \begin{cases} x\,; & if\ x > 0 \\ \alpha * x; & if\ x < 0 \end{cases}$$



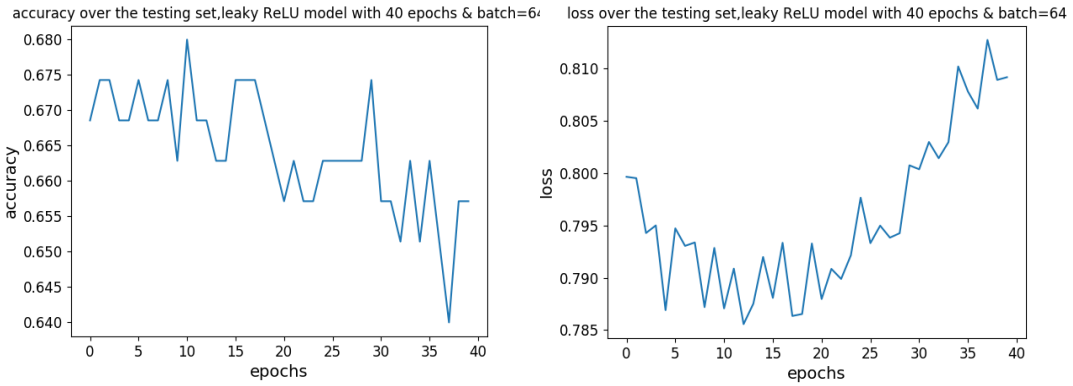
This change will affect the model's non-linearity method.

The Leaky ReLU has some benefits over the ReLU[1]:

1. It does not have a zero-slope parts.
2. It speeds up the training because unlike ReLU, leaky ReLU is more "balanced," and may therefore learn faster.

Task 3: *Number of epochs.*

Train the new model using 25 and 40 epochs. What difference does it makes in term of performance? Remember to save the compiled model for having initialized weights for every run as we did in tutorial 12. Evaluate each trained model on the test set



When the number of epochs is more than the necessary, the model will learn patterns that are specific to the training dataset, and we may reach an **overfitted model**. This model, then, will become incapable to perform well on new set of data (the test dataset in our case)

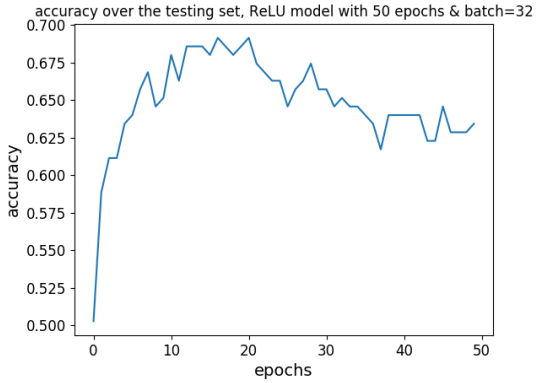
– the accuracy on the training data will be high but the accuracy for the testing data will be low, and will imply for a poor performance[2].

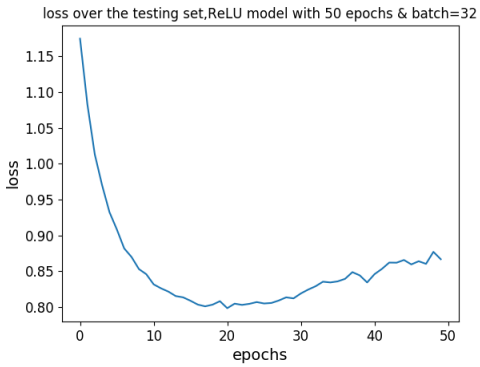
On the other hand, too small number of epochs will cause **underfitting**; the model will not learn enough features before the end of the training, and our results will be poor for both the training and testing data.

As a result, the model should be trained for an optimal number of epochs. Regarding our model, one can notice that the model which was trained with 25 epochs achieved better accuracy score than the model with 40 epochs. So, we conclude that in our case, 25 is indeed the optimal number of epochs needed, while by using 40 epochs we reach **overfitting** and poor performance.

Task 4.1: *Mini-batches.*

Build the model_relu again and run it with a batch size of 32 instead of 64. What are the advantages of the mini-batch vs. SGD?*





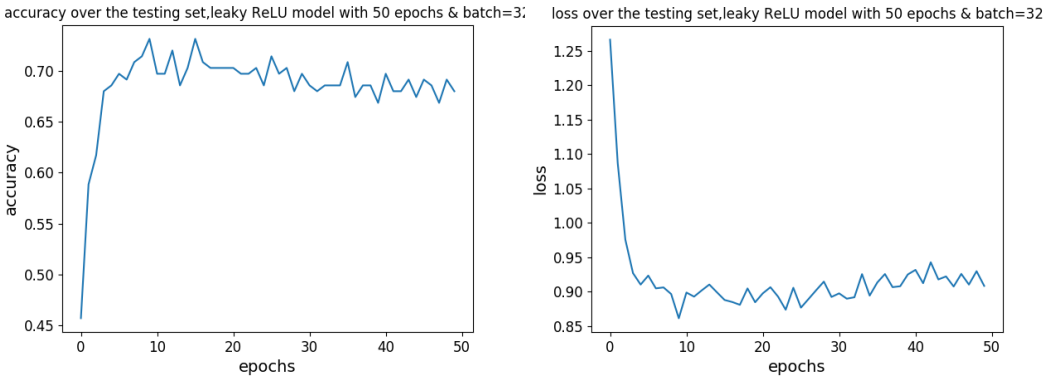
As learned in the lectures, there is a trade-off between the accuracy of the parameter update and the computational time to perform an update.
By using SGD, we compute the gradient of the cost function for one example in each step. This will lead to high fluctuation due to high variance in each update and therefore we end with low accuracy. However, the SGD is considered to be quick because we upload only 1 example each time. In this model we take the mini-batch to the extremity by practically choosing the batch size to be 1, thus the step size will not be averaged and **the noise will not be reduced**.

	Accuracy of parameter update	Computational time to perform an update
BGD	+	-
SGD	-	+
Mini-batch $1<p<m$	+	+

In mini-batch gradient descent, we compute the gradient of the cost function for p examples, when $1 < p < m$, and m is the entire training set. Regarding the trade-off described above, mini-batch is considered to be the compromise which leads to high accuracy and fast computational time compared to the SGD.

Task 4.2: *Batch normalization.*

Build the new_a_model again and add batch normalization layers. How does it impact your results?*



Batch normalization means that the input layer is normalized (i.e. re-centered and re-scaled[3]). Once the input is *normalized*, it is *standardized* to the mean of 0 (zero), and STD of 1. **By using batch normalization, we optimize the NN**: The model will now run faster, because of achieving convergence quicker. Thus, by using this method, one could be satisfied with smaller epoch size and reach the same accuracy or even better. Moreover, batch normalization will allow higher learning rates without risking too high fluctuations, benefiting the speed of the process as well.

In our results, we got a higher accuracy (0.68) after applying batch normalization, compared to the previous, un-normalized models (0.657). The result conforms with our expectations as explained above.

Part 2 :

Task 1: *2D CNN.*

Have a look at the model below and answer the following:

- *How many layers does it have?*

  The model has 8 (hidden) layers:
  - Convolutional 2D layers (5)
  - Fully-connected Dense layers (3)

  Note that other processing operations might be included in each hidden layer, such as *BatchNormalization*, *MaxPooling*, *Dropout* etc.

- *How many filters in each layer?*

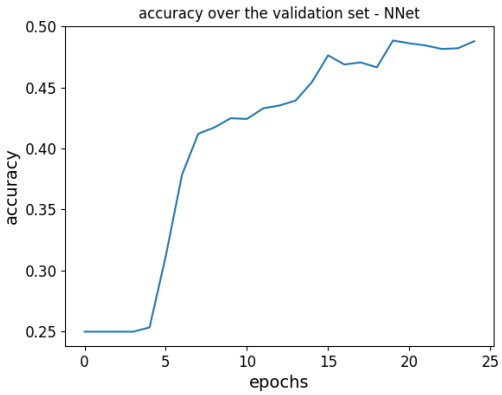
  In each of the five convolutional 2D layer, the number of filters is (respectively): [64, 128, 128, 256, 256]

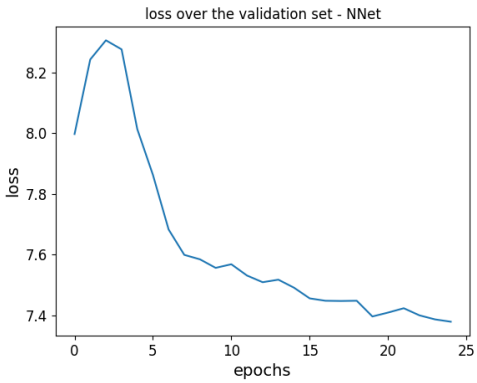
- *Would the number of parameters be similar to a fully connected NN?*

  No, the number of parameters will not be similar to a fully connected NN. In our 2D CNN, we use filters and *MaxPooling* processing, which decrease the number of parameters our model uses.

- *Is this specific NN performing regularization?*

  Yes, this specific NN do perform regularization. Each convolutional 2D layer include a regularization kernel (L2, specifically).

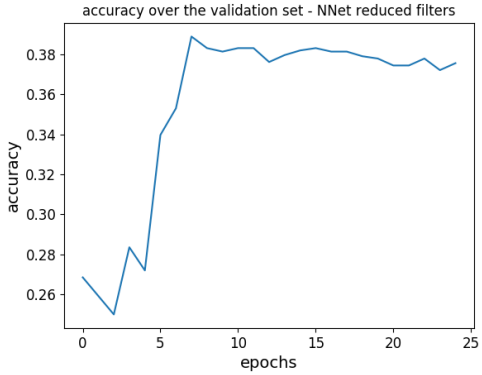


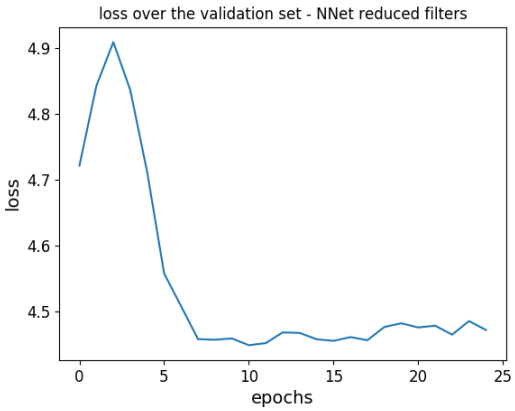


Task 2: *Number of filters*

Rebuild the function `get_net` to have as an input argument a list of number of filters in each layers, i.e. for the CNN defined above the input should have been `[64, 128, 128, 256,`

256]. Now train the model with the number of filters reduced by half. What were the results.





Tests results :

model	loss	acc
ReLU model, batch = 64	0.845	0.668
New model, 25 epochs	0.809	0.6629
New model, 40 epochs	0.809	0.657
ReLU model, batch = 32	0.866	0.63
New model, batch norm	0.908	0.68
NNet model	7.9	0.342
NNet reduced filters	4.869	0.257

References:

[1] https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

[2] https://www.geeksforgeeks.org/choose-optimal-number-of-epochs-to-train-a-neural-network-in-keras/

[3] Jaron Collis, Glossary of Deep Learning: Batch Normalization; https://medium.com/deeper-learning/glossary-of-deep-learning-batch-normalisation-8266dcd2fa82