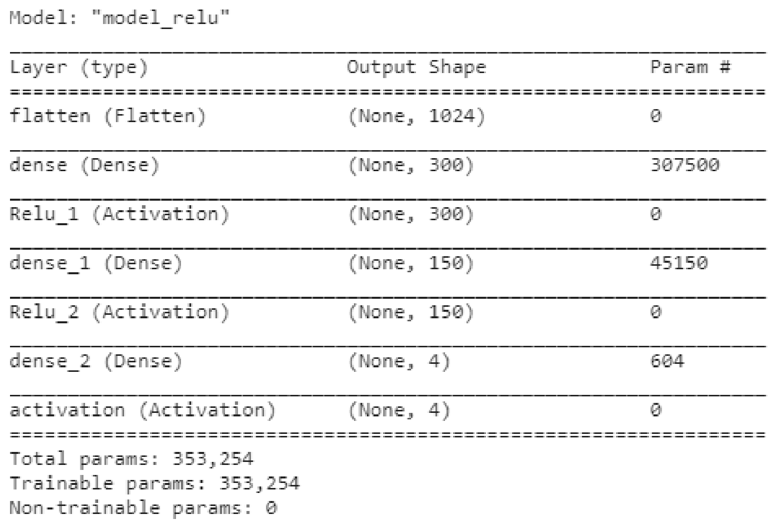
Aseel Nama 319112272
Lilach Barkat 307902270

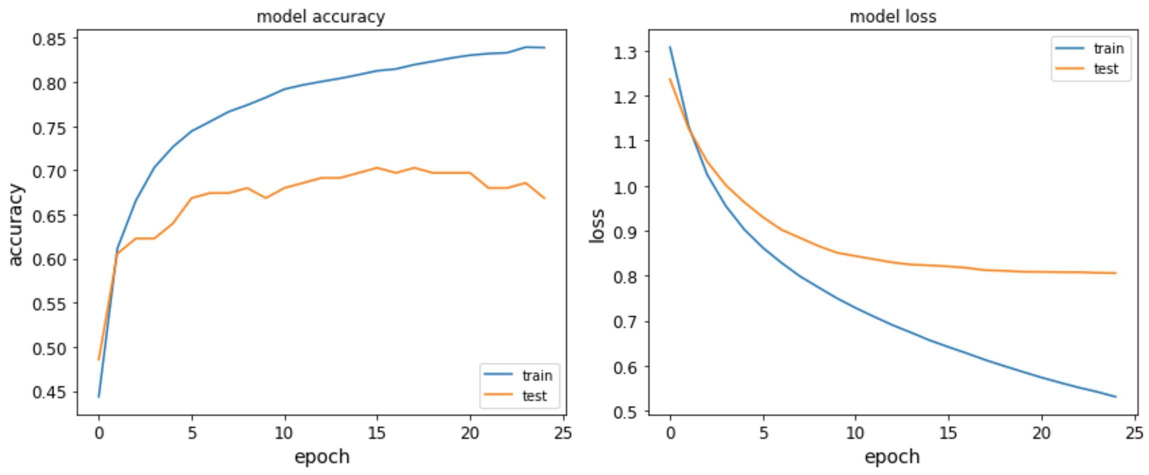
HW4: X-ray images classification

PART 1: Fully connected layers

Task 1: *NN with fully connected layers:



```
Model: "model_relu"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten (Flatten)            (None, 1024)              0         
_________________________________________________________________
dense (Dense)                (None, 300)               307500    
_________________________________________________________________
Relu_1 (Activation)          (None, 300)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 150)               45150     
_________________________________________________________________
Relu_2 (Activation)          (None, 150)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 604       
_________________________________________________________________
activation (Activation)      (None, 4)                 0         
=================================================================
Total params: 353,254
Trainable params: 353,254
Non-trainable params: 0
```

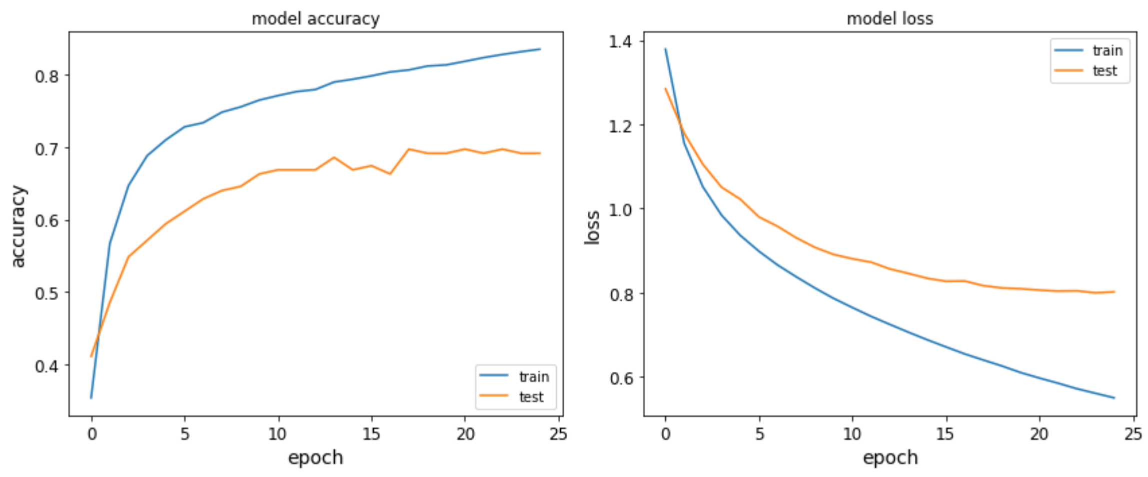


We got train accuracy of 83.9% and test accuracy of 66.8%, the loss over the testing set is 0.81. It seems there is an overfitting of the training data.

Task 2: Activation functions:

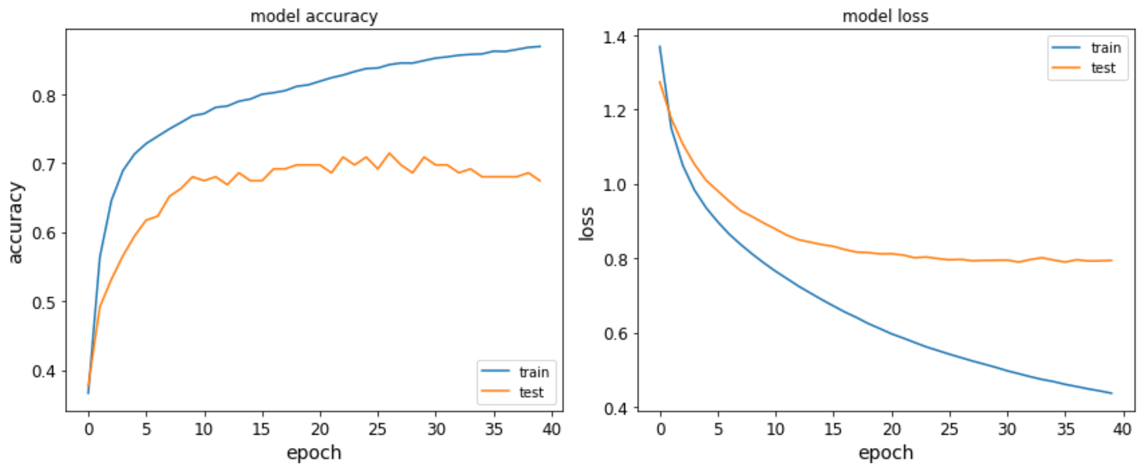
We changed the activation functions to LeakyRelu choosing $\alpha = 0.01$.

The activation function transforms the weighted sum of the inputs into the activation of the node. ReLU, rectified linear unit, is defined as $\varphi(\mathrm{t}) = \max\{\mathrm{t},0\}$. ReLU is Nearly linear and much faster to compute than sigmoid and tanh, but during training can cause vanishing gradient i.e. set lot of the neurons to zero, no gradient flows and the NN performance is affected. This can be corrected by using small slope instead of zero.



We got train accuracy of 83.5% and test accuracy of 69.1%, the loss over the testing set is 0.8. The test accuracy improved as expected.

Task 3: Number of epochs:



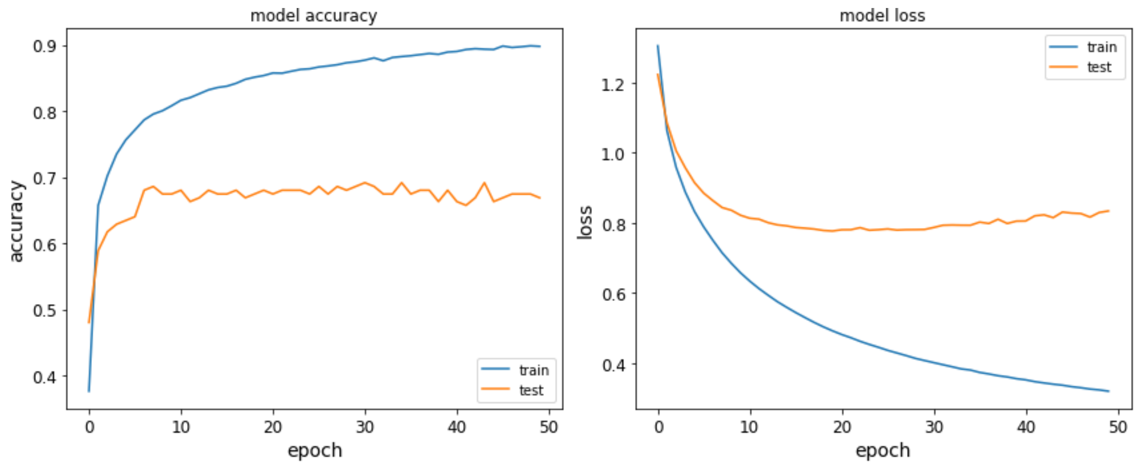
We got train accuracy of 86.9% and test accuracy of 67.4%, the loss over the testing set is 0.79.

Because we already overfit our data, more training just increases the overfitting. In order to improve the test accuracy, we can use more regularization such as dropout layers.

Task 4: Mini-batches:

Stochastic gradient descent compute the gradient of the cost function <u>for each training example</u>, while Mini-batch gradient descent compute the gradient of the cost function <u>for p training examples</u>. In SGD we frequently update the weights with a high variance that cause the objective function to fluctuate heavily. However, performing an update is quick.

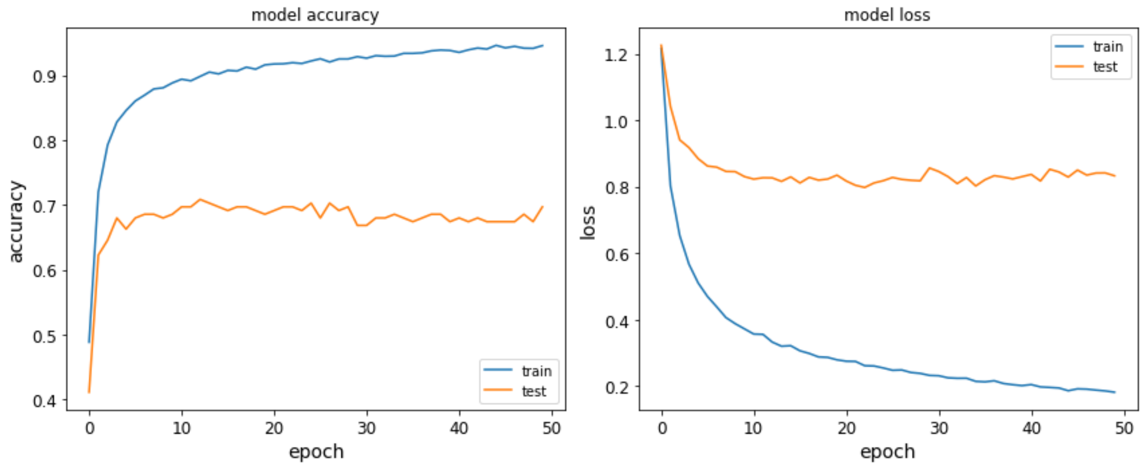
For small Mini-batch we have the effect of quick updates, and as the batch size increases, the objective function does not fluctuate heavily. The vectorization in Mini-batch is computationally efficient because the computer resources are not being used to process a single sample at a time. Compare to a large batch, the Mini-batch easily fits in the memory and because it still has fluctuations it is mostly not stuck in a local minimum. Furthermore, the averaging compare to SGD, produces a more stable and fast gradient descent convergence.



Our results, for train accuracy is 89.7% and for test accuracy is 66.8%, the loss over the testing set is 0.83. We used RelU activation function and decreased the batch size by half. Compare to the first results in Task 1, the test accuracy did not change but we can see more fluctuations in the graph.

Task 5: Batch normalization:

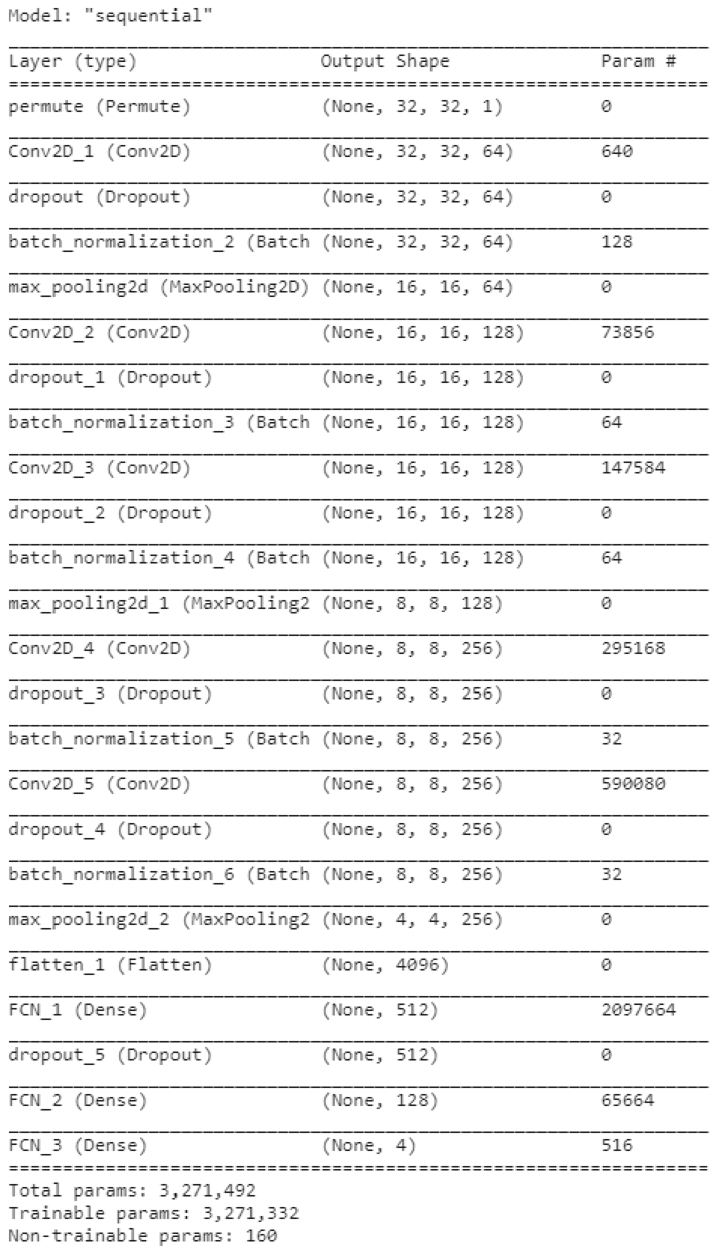
Normalizing the mean and variance of each of the features at every level of representation during training, across the p examples in each mini-batch. Normalization of the activations of each layer, improve the learning speed and outcomes.



Our results, for train accuracy is 94.6% and for test accuracy is 69.7%, the loss over the testing set is 0.83. We used LeakyRelu activation function and added batch normalization layers. Compare to the results in Task 2-3, the test accuracy slightly increased.

PART 2: Convolutional Neural Network (CNN)

Task 1: 2D CNN:



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
permute (Permute)            (None, 32, 32, 1)         0         
_________________________________________________________________
Conv2D_1 (Conv2D)            (None, 32, 32, 64)        640       
_________________________________________________________________
dropout (Dropout)            (None, 32, 32, 64)        0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 64)        128       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 64)        0         
_________________________________________________________________
Conv2D_2 (Conv2D)            (None, 16, 16, 128)       73856     
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 128)       0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 128)       64        
_________________________________________________________________
Conv2D_3 (Conv2D)            (None, 16, 16, 128)       147584    
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 128)       0         
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 128)       64        
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 128)         0         
_________________________________________________________________
Conv2D_4 (Conv2D)            (None, 8, 8, 256)         295168    
_________________________________________________________________
dropout_3 (Dropout)          (None, 8, 8, 256)         0         
_________________________________________________________________
batch_normalization_5 (Batch (None, 8, 8, 256)         32        
_________________________________________________________________
Conv2D_5 (Conv2D)            (None, 8, 8, 256)         590080    
_________________________________________________________________
dropout_4 (Dropout)          (None, 8, 8, 256)         0         
_________________________________________________________________
batch_normalization_6 (Batch (None, 8, 8, 256)         32        
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 256)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0         
_________________________________________________________________
FCN_1 (Dense)                (None, 512)               2097664   
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0         
_________________________________________________________________
FCN_2 (Dense)                (None, 128)               65664     
_________________________________________________________________
FCN_3 (Dense)                (None, 4)                 516       
=================================================================
Total params: 3,271,492
Trainable params: 3,271,332
Non-trainable params: 160
```

- In this NN there are 8 layers – 5 convolution layers and another three fully connected layers.
- The number of filters for the convolution layers are: 64, 128, 128, 256, 256.
- The number of parameters for this NNet is 3,271,492 and still, much less than the number of parameters of a fully connected NN with the same architecture. Fully connected NN has connections for every neuron and therefore the weight matrix is larger and contains more parameters.

- Yes, there is L2 regularization at the convolution layers, dropout layers, batch normalization and pooling.

Task 2: Number of filters:

For the NNet the test accuracy was 34.2%. We trained the model with the number of filters reduced by half, with hidden_dim = [32, 64, 64, 128, 128], this time the test accuracy was 25.7%. Maybe we got these results because we significantly decrease the number of parameters for training.