CSE3CI: Machine Learning

Assignment 2: Report

Student Name: James Worcester

Student Number: 2076 7086

# Task 1: Build a Neural Network WITHOUT Convolutional Layers

**Build a neural network without convolutional layers to do the classification task (hint: you will need the use of dense layers). Then, you can change the model structure (i.e., number of dense layers, number of neurons in dense layers, or activation functions), to be able to improve network performance.**

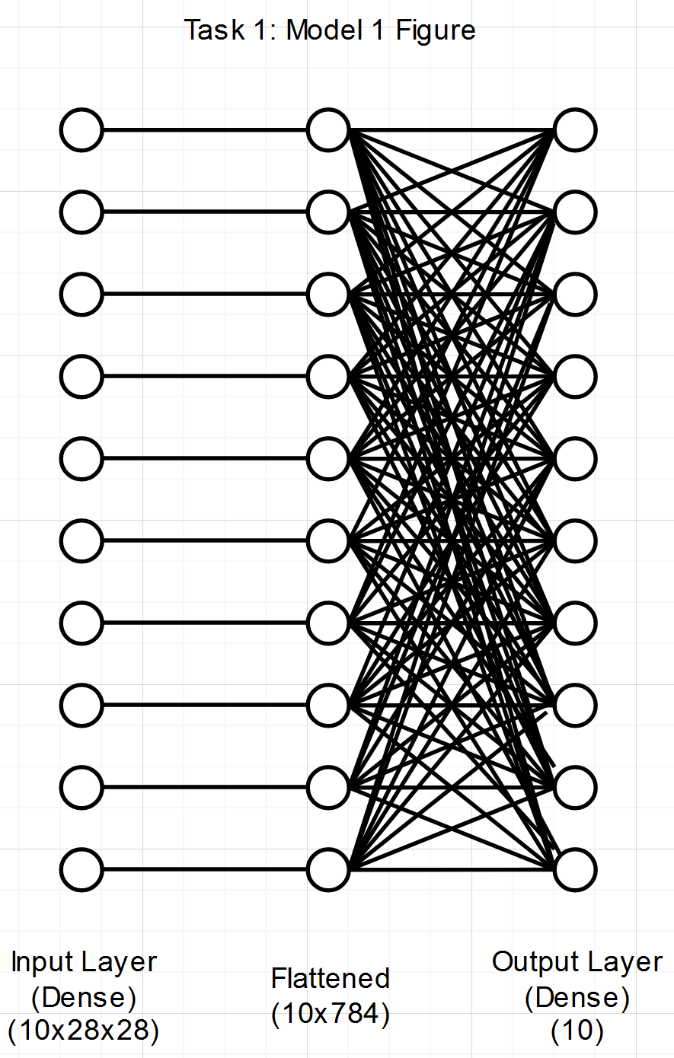
## Task 1: Model 1 - Initial Model

### Detailed Description of Model Architecture with Illustrated Figure

To design the base model of a non-convolutional neural network that processes the Fashion MNSIT dataset, I decided to keep it simple by firstly flattening the 28x28x1 grayscale images into a vector so that it could be used by dense layers. I then added a hidden dense layer of 10 neurons, before using an output layer of 10 neurons (equal to the number of unique classes in the dataset). It’s important to note that a softmax activation function was used for the output layer because we’re dealing with a multi-class classification problem. The model was compiled with a ‘categorical\_crossentropy’ loss function and the default stochastic gradient descent optimizer.

Keeping the initial non-convolutional model simple allows us to experiment and more easily see the relationship between input, model, output and accuracy. It also helps us to identify how model performance can be improved or reduced, and allows us to identify trends and correlations in how model architecture and components effect model accuracy.

This base model had 7960 trainable parameters and achieved an accuracy of 82.06% in correctly identifying images in the dataset



## **REDO THIS MODEL**

## Task 1: Experiments In-between Models & In-Depth Discussion on Improvements/Deterioration of NN’s Performance

The first experiment was to increase the number of neurons in the hidden dense layer from 10 to 512, a change that increased accuracy by 1% from 82.06% to 83.06%. Despite increasing the number of neurons and number of trainable parameters dramatically from 7960 to 407,050, the performance increase was relatively small, and it seems that just arbitrarily increasing the number of neurons in a model with only one hidden dense layer is not too much of an effective strategy to optimise model performance.

The second experiment was to take the base model and add an additional 10 neuron hidden dense layer to it, a change that only resulted in an increase of 110 trainable parameters from 7960 to 8070 and increased accuracy by 0.12% from 82.06% to 82.18%. By increasing accuracy at a relatively low cost, adding more dense layers to the initial model might be much more effective then simply increasing the number of neurons in a single hidden dense layer

Following on from the results of the second experiment, the third experiment uses a total of four hidden dense layers with a decreasing number of neurons from 50 to 30 to 20 to 10. This caused accuracy performance to increase from 82.18% in the second experiment to 82.90%, indicating that there might be a sweet spot for the number of dense layers and the number of neurons in each dense layer.

To see if performance could be improved from the third experiment, the fourth experiment gave the 50 neuron layer a relu activation function, the 30 neuron layer a sigmoid activation function, and the 20 neuron layer a tanh activation function, but left all other parameters the same. Performance actually decreased from 82.90% to 80.09%, indicating that there may be an issue with using particular activation functions and/or using many different activation functions together in the one model.

Experiment 5 tried to build on top of the previous experiments by using 100 neurons with sigmoid activation in the first hidden dense layer, followed by a sigmoid hidden dense layer with 50 neurons, a relu dense layer with 20 neurons and a relu dense layer with 10 neurons. Performance dropped dramatically, with the model only achieving an accuracy of 65.82%, indicating that there might be a big issue using a model that is predominantly made up of sigmoid hidden dense layers.

Experiment 6 took the idea that too many sigmoid layers with a large amount of neurons in them might have been sued in experiment 5, and modified the model to use an initial 50 neuron hidden dense layer with sigmoid activation, followed by a relu 100 neuron layer, a relu 20 neuron layer and a 10 neuron layer. As suspected, performance increased from 65.82% to 79.93%, an accuracy rate that still performs below that of the base model, but that also tells us a lot about using too many sigmoid layers and how they struggle to extract different levels of features from the images in the dataset.

Experiment 7 tried to build upon experiment 6 by keeping all layers the same except adding two dropout layers of 20% each between the dense layers. This, however, only decreased accuracy from 79.93% to 76.42%, and given the fact that we have 60,000 training samples and 46,690 trainable parameters, it seems unlikely that overfitting is an issue and as such using dropout layers with this small number of parameters works against the model.

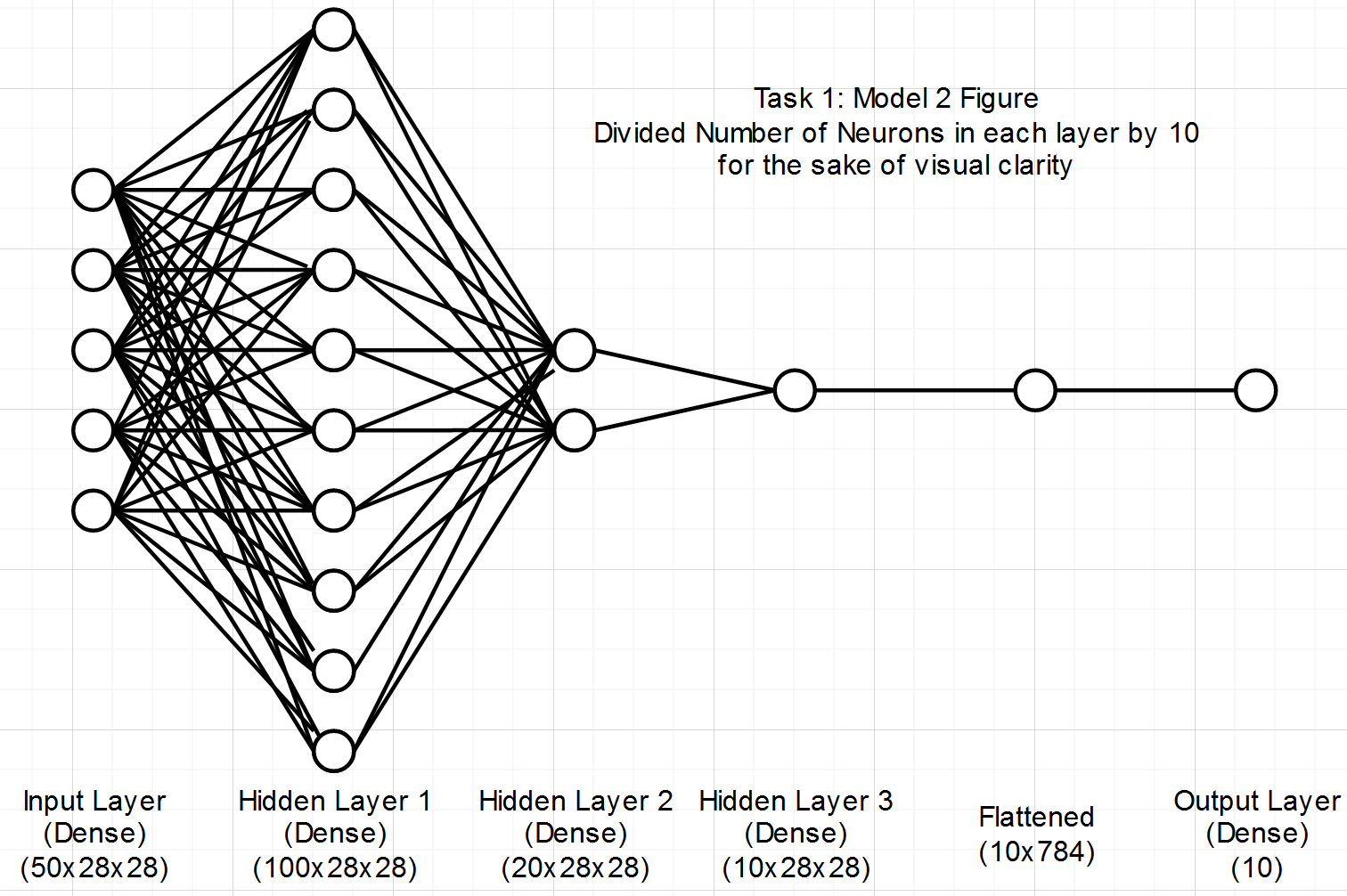
## Task 1: Model 2 - Better Accuracy

### Detailed Description of Model Architecture with Illustrated Figure

The second model for task 1 is based on the best performing experimental model above: Experiment 1, however it’s been modified using the lessons learned through the experimentation that’s been done. From the first experiment, we learned that increasing the number of neurons in a model with only one hidden dense layer would likely increase accuracy but would do so rather inefficiently. From experiment 2, we learned that adding another hidden dense layer could increase accuracy from the base model. The fourth, fifth and sixth experiments indicated there might be a problem using heavily weighted sigmoid activation functions, and from the seventh model we can infer that using dropout layers is inappropriate given the number of training samples and trainable parameters.

From all of this, I developed a model that flattened the image into a vector, then used two hidden dense layers of 256 neurons each with relu activation functions, before and output dense layer of 10 neurons with softmax activation and the default stochastic gradient descent optimizer. We accounted for using multiple layers with a high number of neurons, avoided using any sigmoid activations and avoided using dropout. The results speak for themselves, with the model achieving an accuracy of 84.32%, which significantly improved on the base model and all experimental models.

This model initially flattens the image into a vector, before using a hidden dense layer of 512 neurons connected to an output layer of 10 neurons with a softmax activation function.



## **REDO THIS MODEL**

# Task 2: Build a Neural Network WITH Convolutional Layers

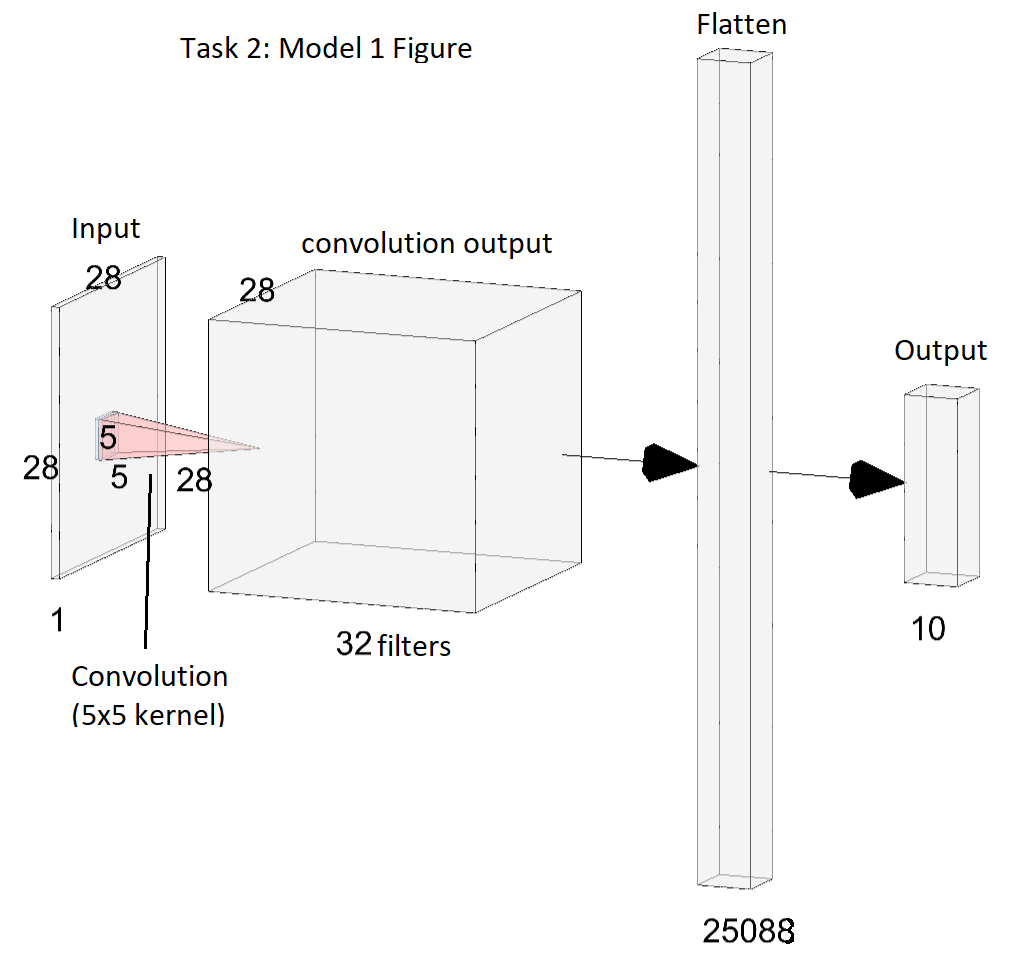
**Build a neural network with the use of convolutional layers (you can decide other layer types you want to include in your network). Then, you can change: the number of convolutional layers, the number of filters, or activation function functions in convolutional layers, to be able to improve network performance.**

## Task 2: Model 1 - Initial Model

### Detailed Description of Model Architecture with Illustrated Figure

The initial base model I’ve designed of a convolutional neural network to process the Fashion MNSIT dataset keeps it simple utilising one 5x5 kernel to convolute each 28x28x1 image into a convolutional output with 32 filters using a relu activation function and maxnorm kernel constraint of 3 for regularisation. From there, the output is flattened into a vector, before using an output layer of 10 neurons with a softmax function, one for each class of clothing item.

Additionally, I customised the stochastic gradient descent optimiser with an initial learning rate of 0.002 across 5 epochs with a momentum of 0.7. This initial convolutional model has 251,722 trainable parameters and achieved an accuracy of 82.66%.



## Task 2: Experiments In between Models & In-Depth Discussion on Improvements/Deterioration of NN’s Performance

The first experiment was to change the original model’s convolutional layer to use a kernel of size 3x3 instead of 5x5. Doing so increased the accuracy by 0.23% up to 82.89%, showing that there may be a correlation between a smaller kernel size and increased performance across the Fashion MNSIT dataset.

The second experiment was to keep the 3x3 kernel and change the number of filters for the sole convolutional layer from 32 to 64. This resulted in an almost doubled number of trainable parameters, however accuracy reduced to 82.80%. At this stage, it might be hard to draw any conclusions when it comes to changing model parameters because only one convolutional layer has been used and no pooing layers have been used

To test this idea for experiment 3, inspiration was drawn from the VGG16 neural network model we discussed in lab 9 to try to produce a model with many convolutional and pooling layers. I ended up creating a model that started with two convolutional layers with 3x3 kernels, 64 filters and relu activation functions, followed by a max pooling layer, then two more 3x3 convolutional layers with 128 filters, another max pooling layer and two more 3x3 layers with 256 filters before flattening and having a softmax output layer. This resulted in an improved accuracy of 83.89% with 1,269,706 trainable parameters. Drawing inspiration from a well-developed model paid dividends, and I continued to modify this model.

For the fourth experimental model, I simply added two dense layers of 50 neurons each to the VGG16-inspired experiment 3 model to bring it more inline with the VGG16 model from the labs. Doing so resulted in increasing the accuracy by 0.54% to 84.43%.

From there I started to experiment again, removing one of each of the 64 filter, 128 filter and 256 filter convolutional layers from experiment 4, and modifying the remaining 64 filter layer by changing its activation function to sigmoid. This experimentation ultimately resulted in reducing the accuracy to 78.38%, showing that deviation from a well-established image-classification model, combined with trying another sigmoid activation layer resulted in worse model performance.

Experiment 6 saw me change the sigmoid layer back to relu, as well as trialing the addition of two 50% dropout layers between convolutional layers. Doing so only furthered the accuracy loss from experiment 5 to 76.88%, which when taken in the context of building upon the experiments performed with dropout layers in task 1, seems to indicate that using dropout layers in this way with a large training dataset may only be detrimental.

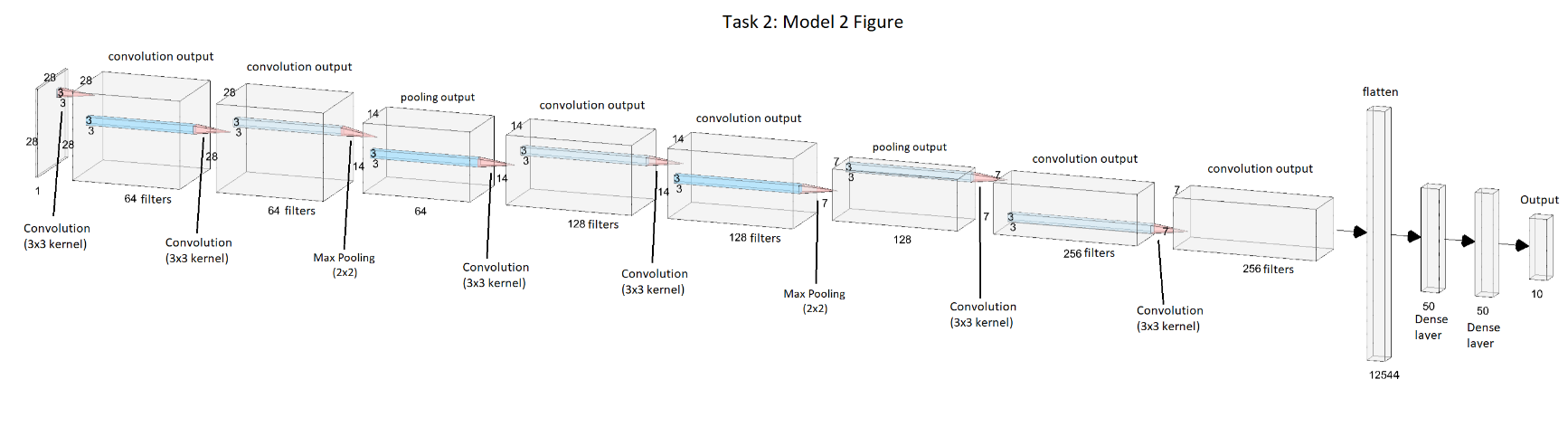
For the seventh experiment I took inspiration from the ‘Deeper CNN network and optimization’ model presented in Lab 8, using two 3x3 kernel 32 filter convolutional relu layers followed by a max pooling layer, another two 3x3 kernel 64 filter convolutional relu layers followed by a max pooling layer, another two 3x3 kernel 128 filter convolutional relu layers followed by a max pooling layer, then a 1024 neuron dense layer with relu activation, a 512 neuron dense layer with relu activation, a dropout layer of 20% and the final 10 neuron softmax output layer. This experimentation produced a model that had an accuracy of 83.18%, which is an average result at this point.

For the eighth and final experiment, I took the best performing experiment (experiment 4) and changed the stochastic gradient descent (SGD) optimizer settings by changing the learning rate from 0.002 to 0.001 and the momentum from 0.7 to 0.9 after taking further inspiration from ‘Lab 8: Deeper CNN network and optimization’. Doing so provided excellent results, with the model achieving an accuracy of 86.27%, a remarkable result when compared to all other experiments. From this we can infer that optimizers can have a very large effect on the performance of a convolutional model.

## Task 2: Model 2 - Better Accuracy

### Detailed Description of Model Architecture with Illustrated Figure

The second model for task 2 is based on the best performing experimental model above: Experiment 8. This model uses six convolutional layers with relu activation functions, two max pooling layers and two hidden dense layers. Specifically, the model starts with two 3x3 kernel 64 filter convolutional relu layers followed by a max pooling layer, two 3x3 kernel 128 filter convolutional relu layers followed by a max pooling layer, two 3x3 kernel 256 filter convolutional relu layers, followed by being vectorized (flattened) and having two 50 neuron hidden dense layers followed by a 10 neuron softmax output layer. The SGD learning rate was set to 0.001 with 5 epochs and a momentum of 0.9, and achieved an accuracy of 86.28%.

For this final model, I took advantage of reducing the kernel size from 5x5 to 3x3 as established in the first experiment, utilising knowledge gained from the VGG16 neural network from the third and fourth experiments, making sure to avoid halving the number of convolutional layers and avoid using any sigmoid activation functions as established in the fifth experiment, not using any dropout layers like experiment 6, and utilising the benefits of changing the SGD optimizer learning rate and momentum from experiment 8.

# Task 3: Change the type of Optimizer or Learning Rate you applied in the Previous Tasks

**Change the type of optimizer or learning rate that you applied in the previous tasks, and see how these changes can influence model performance (You can keep the final network structure you applied in task 2, and try at least one different optimizer setting).**

## Task 3: Changing Optimizer or Learning Rate for Task 1

### Detailed Description of Optimizers/Learning Rates

Taking the final model for Task 1 that achieved an accuracy of 84.32% using the standard stochastic gradient descent (SGD) optimizer, I tried using the adam optimizer that we experimented with during labs instead, which led to a huge gain in accuracy of 3.97% to 88.29%. This gain appears to be in line with the gains from adam that we saw through its use in Lab 7, which makes sense because adam builds upon the standard stochastic gradient descent method.

## Task 3: Changing Optimizer or Learning Rate for Task 2

### Detailed Description of Optimizers/Learning Rates

Taking the final model for Task 2 that achieved an accuracy of 86.28% using an already modifed SGD optimizer with a learning rate of 0.001 and momentum of 0.9, I tried to again utilise the adam optimizer instead, resulting in a massive boost in accuracy by 6.48% to 92.76%, far and away the most impressive model of the whole assignment. Once again, this gain from adam appears to be in line with the gains from adam that we saw through its use in Lab 7.

# Task 4: In-Depth Discussion on Improvements/Deterioration of NN’s Performance Across All Three Tasks

**UP TO HERE**

# Task 5: Ranking Neural Network Performance from All Three Tasks