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.**

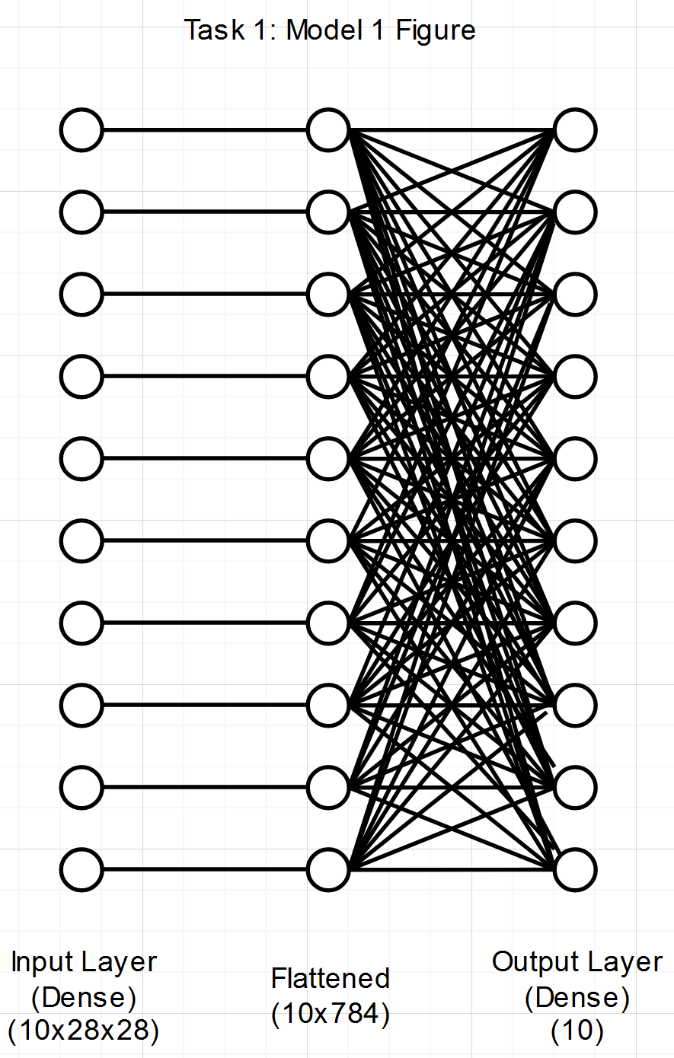
## Task 1: Model 1 - Initial Model

### 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 using an initial dense layer of 10 neurons to process the 28x28x1 pixel grayscale images, which I then flattened into a vector, 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 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 78,430 trainable parameters and achieved an accuracy of 83.66% in correctly identifying images in the dataset.



## 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 initial dense layer from 10 to 512, a change that only increased accuracy by 0.12% from 83.66% to 83.78%. Despite increasing the number of neurons and number of trainable parameters dramatically from 78,430 to 4,014,090, the performance increase was relatively negligible, and it seems that just arbitrarily increasing the number of neurons in a single dense layer is not an effective strategy to optimise model performance.

The second experiment was to add an additional 10 neuron hidden dense layer to the base model, a change that only resulted in an increase of 110 trainable parameters from 78,430 to 78,540 but managed to increase accuracy by 0.46% to 84.12% for relatively little cost. At this point, adding more dense layers to the initial model is much more effective then increasing the number of neurons in a single layer.

Following on from the results of the second experiment, the third experiment uses a total of five dense layers with a decreasing number of neurons from 50 to 30 to 20 to 10 to 2. This caused accurate performance to reduce from the 84.12% in the second experiment to 83.83%, and it looks like we might be overfitting the model or there could be some issues with certain dense layers, such as the layer with two neurons.

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. Performance increased to 84.54%, indicating that using varied activation functions across dense layers might have some benefits.

Experiment 5 tried to build on top of the previous experiments by using an initial sigmoid dense layer with 100 neurons, a sigmoid dense layer with 50 neurons, a relu dense layer with 20 and a relu dense layer with 10. Performance dropped to 84.17%, and it looks like either the model might be overfitted or there’s an issue with using too many powerful sigmoid layers.

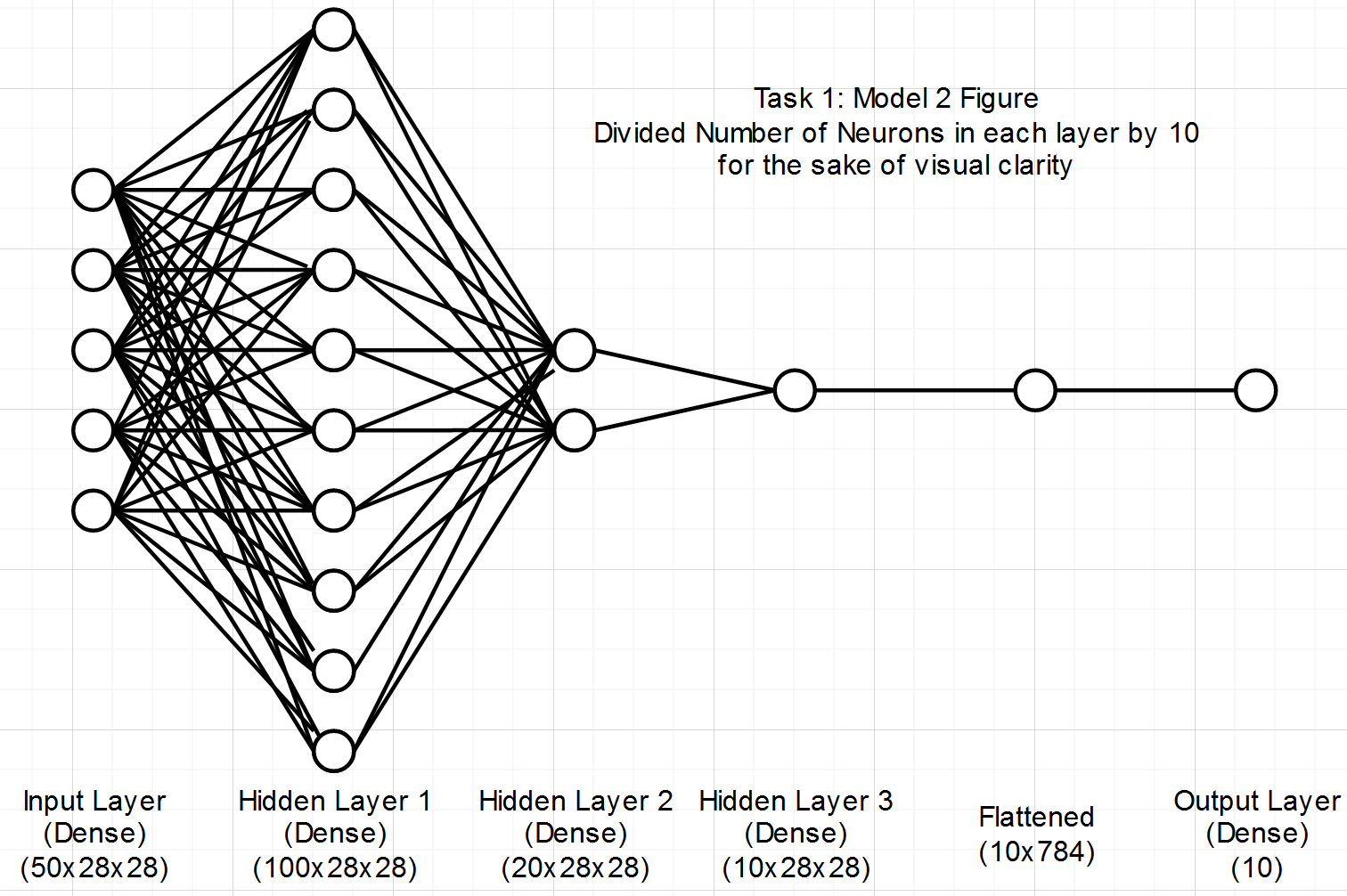
Experiment 6 took the idea that too many sigmoid layers might have been used in experiment 5, and modified it to use an initial sigmoid 50 neuron layer, a relu 100 neuron layer, a relu 20 neuron layer and a 10 neuron layer. As suspected, performance increased to 85.61%, which is best performance that we’ve seen. It appears that using enough dense layers with different numbers of neurons and with different activation functions is very effective at extracting different levels of features from 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. However, this decreased accuracy to 83.63%. Given the fact that we have 60,000 training samples and 84,840 trainable parameters, it seems unlikely that overfitting is an issue and as such using dropout 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 6. This model uses an initial dense layer of 50 neurons with a sigmoid activation function, another dense layer of 100 neurons with a relu activation function, a third layer of 20 neurons with a relu activation function, a fourth layer of 10 neurons, flattening into a vector, and then a fifth and final output layer of 10 neurons with a softmax activation function. The default SGD optimizer continues to be used, which will be changed and experimented with in Task 3.



# Task 2: Build a Neural Network WITH Convolutional Layers

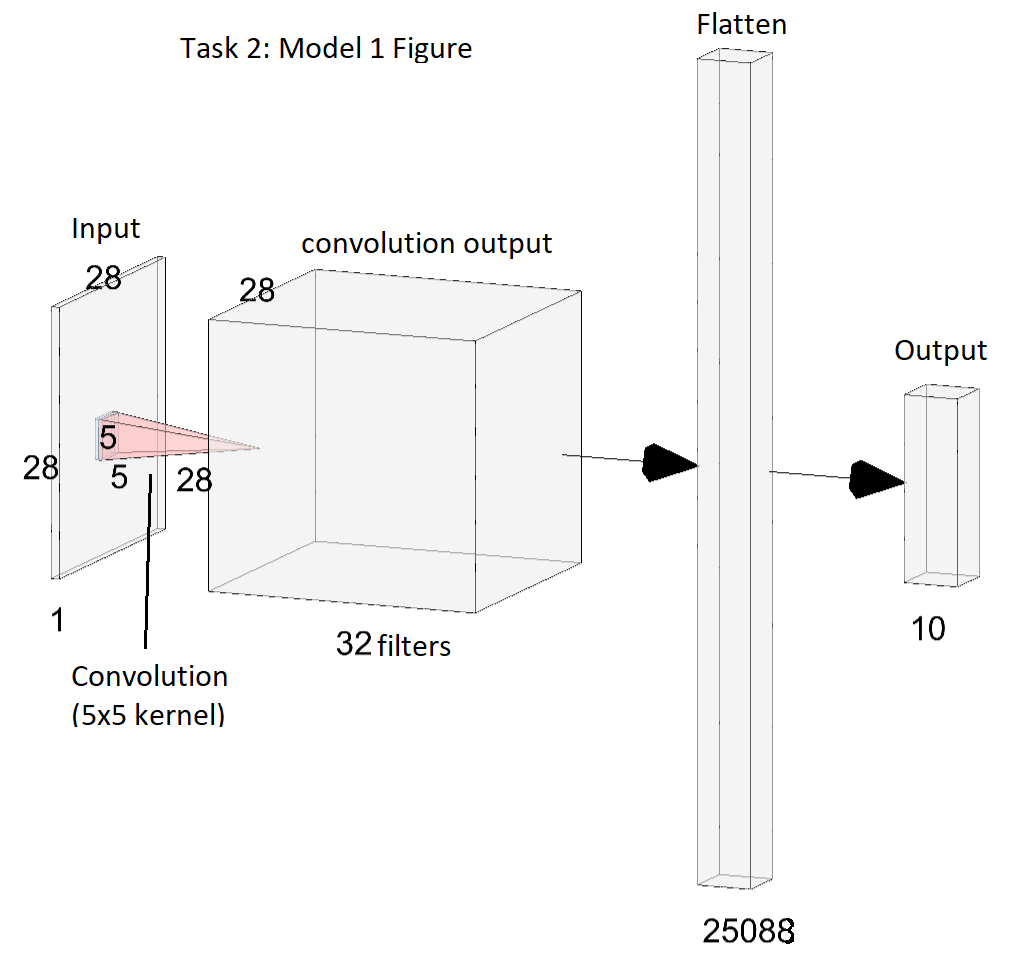
**Build a neural network with the use of convolutional layers (you can decide other layer types you want to include in your network). Then, you can change: the number of convolutional layers, the number of filters, or activation function functions in convolutional layers, to be able to improve network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

To test this idea, inspiration was drawn from the VGG16 **\*\*\*UP TO HERE\*\*\***

## Task 2: Model 2 - Better Accuracy

### Detailed Description of Model Architecture with Illustrated Figure

Graphical user interface, application, Word

Description automatically generated

### Detailed Description of Optimizers/Learning Rates

# 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

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

### Detailed Description of Optimizers/Learning Rates

## Task 3: Experiments Inbetween Models & In-Depth Discussion on Improvements/Deterioration of NN’s Performance

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

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