Evan Tan 27401995
etan0008@student.monash.edu

Model Weights

Model weights and specific checkpoints used are included in their relevant folders. However, I think I might have accidentally overwritten some of them… With the discrepancy in accuracy being about 0.1% sorry. Furthermore **all training cells have been commented out.**

Optimizers

In my SGD optimizer, I have used momentum of 0.9 and weight decay of 1e-4 as specified in the config dict in training my models. I've chosen to use weight decay as a regularization technique that adds a small penalty, which is the product of the weight decay and the L2 norm of all the weights of the model, whenever the loss function is updated in order to reduce overfitting of data as there is a lot of overfitting when the model is trained for large number of epochs.

I've chosen to use momentum of 0.9 as it seems to be a robust choice, but in particular it may help convergence by calculating a moving average of the gradients, instead of taking random directions on the loss landscape. As the gradients are computer over batches of the training data, it is possible that the direction taken may not be optimal as the derivatives are noisy. Thus, if the direction taken is correct, the weighted average helps the model converge faster.

Model Naming

For Q1, the model names for the two networks, where the number of neurons in **fc1, n = 16** corresponds to the model **AE_n16** and the model where **n = 64** corresponds to the model **AE_n64.**
For Q2, the model is referred to as **ShallowMLP**.
For Q3, the model in Q3.1 is referred to as **DeepMLP** and Q3.2 is referred to as **DeepWideMLP**.
For Q4, the model is referred to as **CustomMLP**.

Whenever a fixed or variable (multi-step) learning rate is used, the **FixedLR** or **VarLR** is added to the model checkpoint's name respectively.

Evan Tan 27401995
etan0008@student.monash.edu

1.1

1.1.1 Identifying Learning Rate

Initially, I used Ray Tune to identify several learning rates and batch sizes, however this approach was ineffective as the model outputs were mostly the same even though the input faces were different, indicating model overfit as seen below in Fig. 1.

Fig. 1: Model outputs for AE_n16 and AE_n64 demonstrating overfitting.

In order to identify the appropriate learning rate, I observed the model outputs given multiple batches of input images. Initially all of the faces output by the model were the same, even though input faces were all different. This meant that there was overfitting of the model, so I introduced weight decay as well as decreased learning rate to achieve better results. Additionally, high learning rate was used initially to observe unstable loss, and the loss curve was always inspected to ensure the training loss converged.

For this particular problem, the purpose of the model is to be an autoencoder and automatically learn the features of the dataset. Thus, I observed the model outputs and tried tuning weight decay and learning rate to match the input faces as closely as possible across multiple runs.

Evan Tan 27401995
etan0008@student.monash.edu

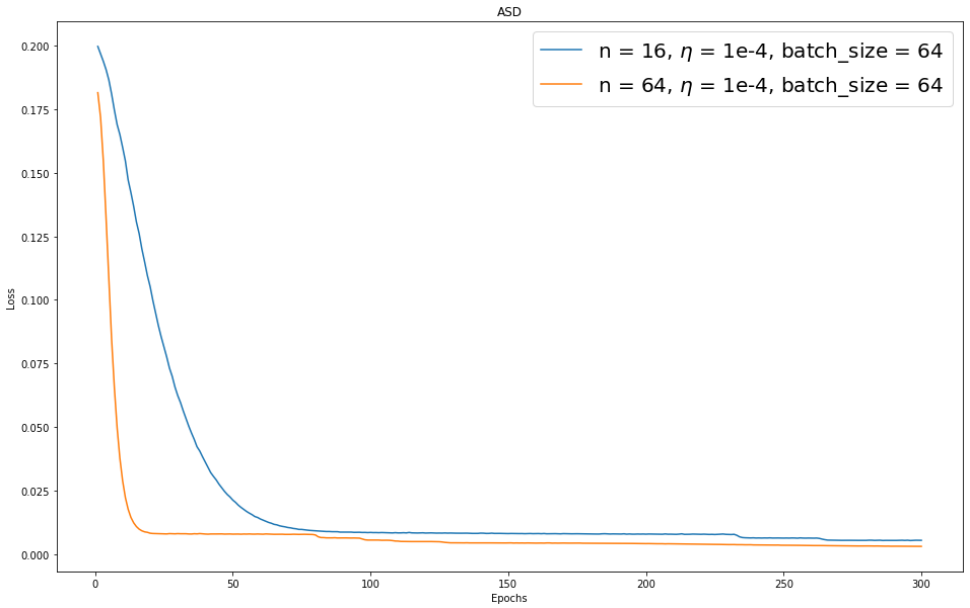


Fig. 2: Training Loss curves for AE_n16 and AE_n64, with learning rate = 1e-4, weight decay = [2.5e-5, 5e-5] for AE_n16 and AE_n64 respectively, and batch size = 64

1.1.2 Final Output of Samples

Evan Tan 27401995
etan0008@student.monash.edu

Fig. 3: Two sets of samples (top rows) for ae_n16 (middle rows) and ae_n64 (bottom rows)

Comparing the models AE_n64 and AE_n16, AE_n64 is much more consistent in being able to learn a representation of the data, and is able to learn a larger set of features perhaps due to the larger number of parameters (4 times larger) in its hidden layer fc1. AE_n16 is unable to reproduce certain features such as different eyebrows, teeth from smiling, whereas AE_n64 has been able to do so as seen in Fig. 3. Perhaps by increasing the number of parameters by increasing **n**, the new architecture will be able to learn a more diverse set of features as **n** is directly correlated to the number of learnable features in the model.

1.1.3 Visualizing the Stored Weights

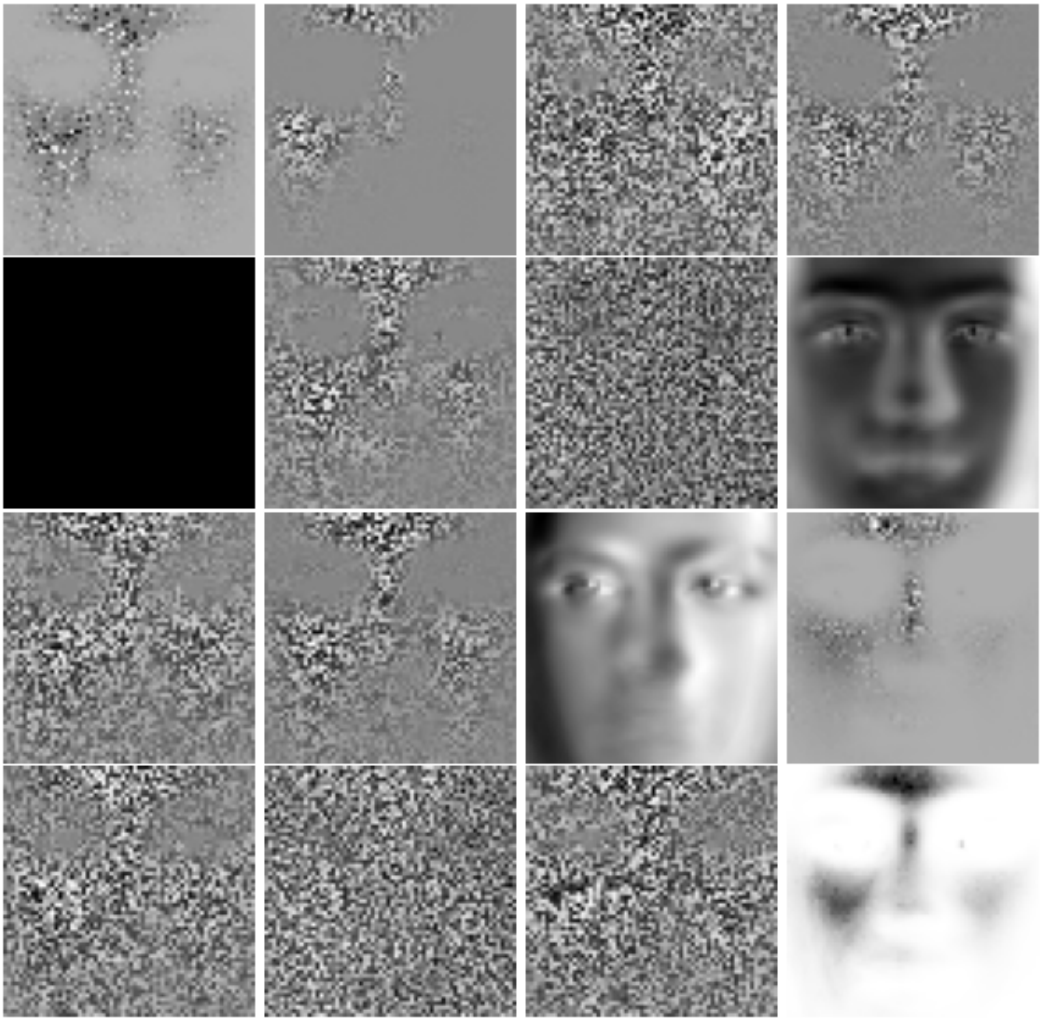


Fig. 4: fc1 weights visualized for 300 epochs for AE_n16

Evan Tan 27401995
etan0008@student.monash.edu

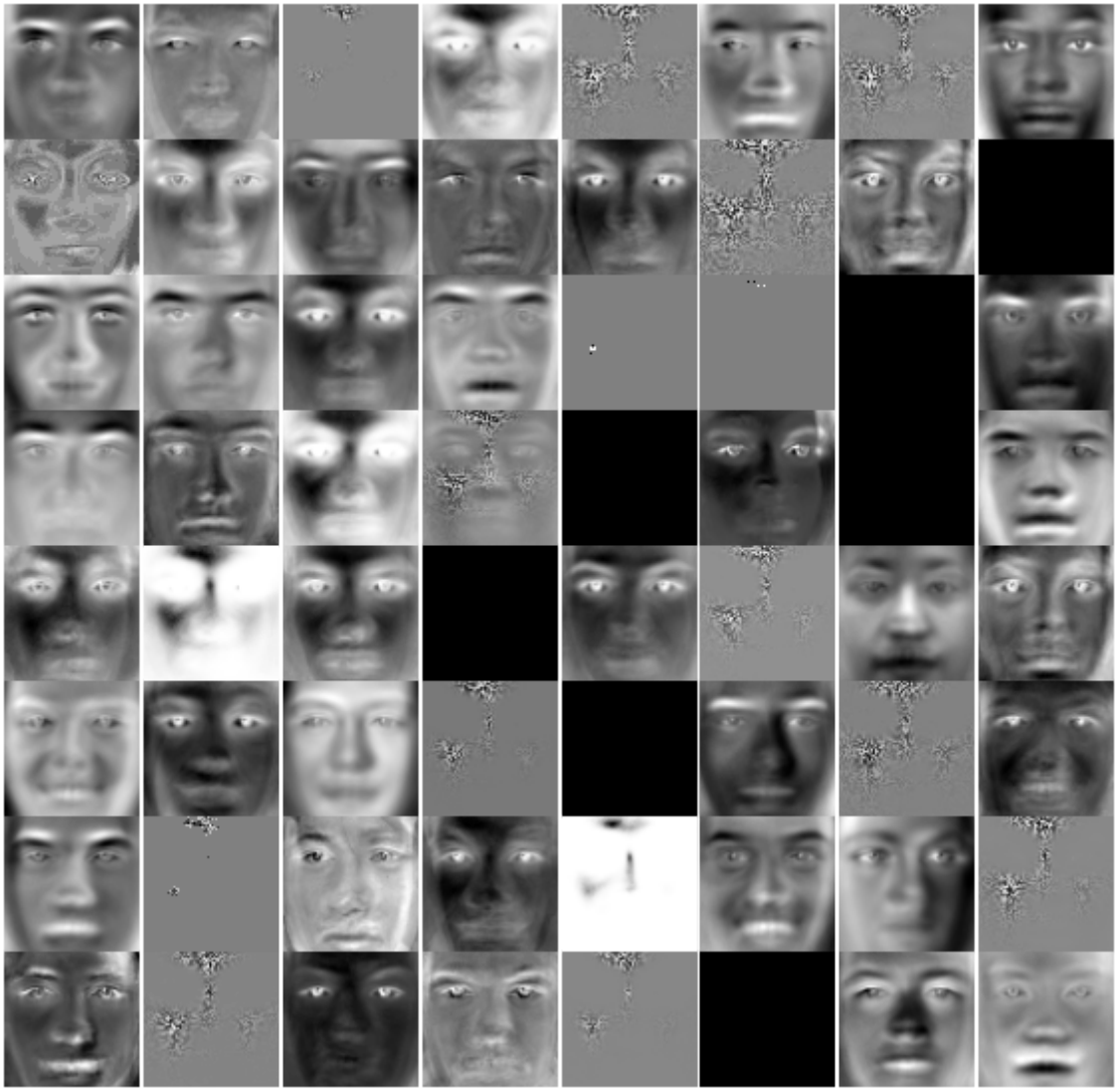


Fig. 4: fc1 weights visualized for 300 epochs for AE_n64

I expected the weights of the fc1 layer to be completely random, as in just random black and white pixels perhaps with noise. I didn't expect the facial features to be visualized at the fc1 layer. Furthermore, I did not expect fully black patches to be formed at the fc1 layer as this represents an empty image, but perhaps with some image normalization, some features would be revealed in those black images.

Comparing Fig. 3 and 4, it can be seen that the smaller network AE_n16 has much fewer "ghost faces" compared to the larger network AE_n64.

1.2

1.3

Yes, I believe that the network would be able to denoise/recover contaminated faces images to a certain extent. Depending on the number of pixels, the detected features may be completely removed and when that happens the network will predict the wrong face due to the change of intensity of pixels. If there was data augmentation i.e. randomly

Evan Tan 27401995
etan0008@student.monash.edu

adding noise, or randomly removing pixels from images, the network could learn in a more robust manner to stitch together the features in order to accurately reproduce the input image given some noise/contamination.

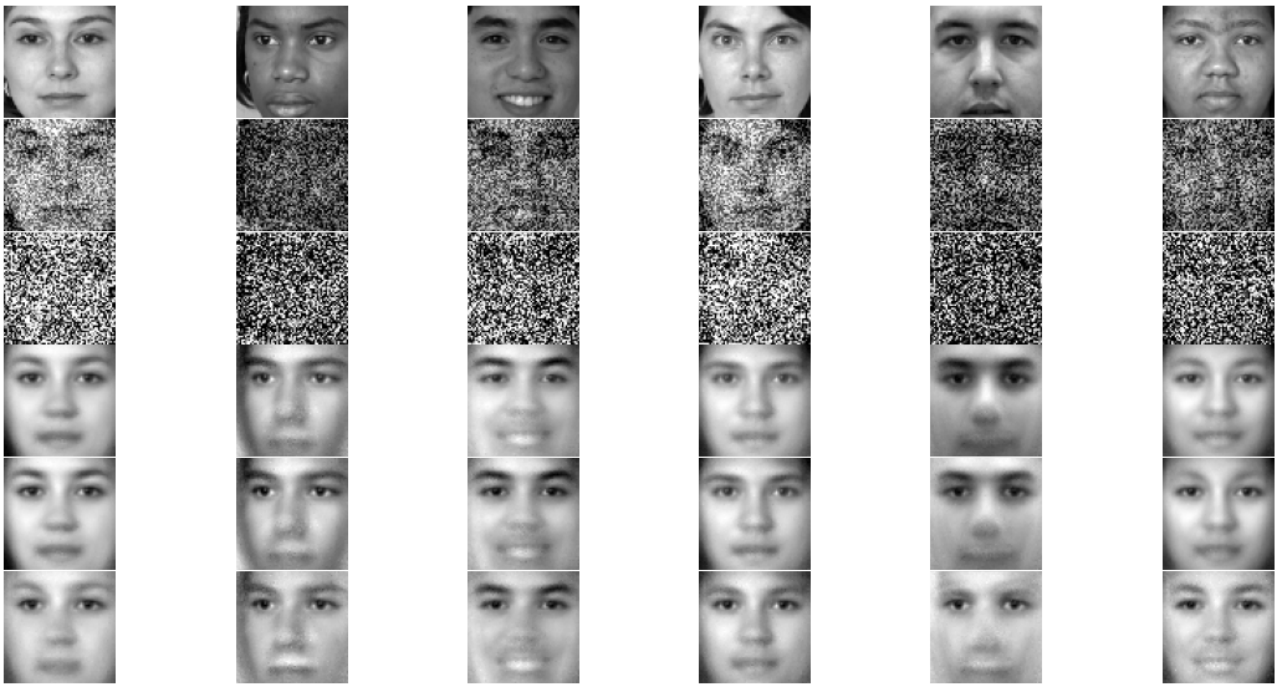


Fig. 5: Top row - actual sample, Second row - noise added with uniform distribution from -0.4 to 0.4), Third row - noise added with uniform distribution from -0.9 to 0.9), Fourth row - model output from clean sample, Fifth row - model output from low noise sample, Sixth row - model output from high noise sample

From The following example, it can be seen that the model is able to produce the same output given a low amount of noise and where features are still humanly visible despite noise. However, when features are almost gone when adding large amounts of noise, the reproduction of the clean sample images is very poor. However, if data augmentation was done before model training, the model might be able to handle extreme cases such as Row 3 in Fig. 5.

Evan Tan 27401995
etan0008@student.monash.edu

2.1

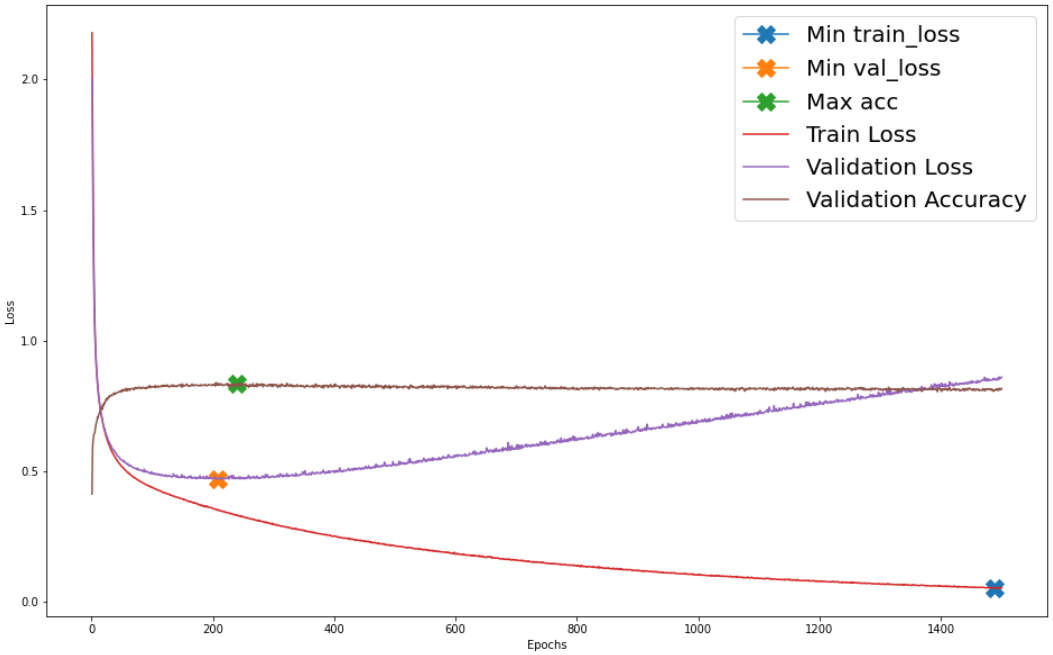


Fig. 6: Training loss, validation loss and validation accuracy for model using learning rate of 1e-3, momentum = 0.9, batch size = 128, and 1500 epochs.

Using the same training parameters, models were selected based on lowest training loss, lowest validation loss and highest validation accuracy and the performance of these models are seen in Table 1. Note that model checkpoints were only saved every 10 epochs therefore some data points such as Epoch 1499 which has the lowest training loss were not saved.

Model Criteria	Epoch Number	Training Loss	Validation Loss	Validation Accuracy (%)	Test Accuracy (%)
Lowest Training Loss	1490	0.053676	0.852106	81.3%	82.94
Lowest Validation Loss	210	0.351337	0.471520	83.45	85.08
Highest Validation Accuracy	240	0.331309	0.473839	83.65	85.13

Table 1: Metrics for model shown in Fig. X above

From Table 1, the best model is the checkpoint at 240 epochs based on highest validation accuracy scoring 85.13% test accuracy and 83.65% validation accuracy, with the model checkpoint at 210 epochs based on lowest validation loss with only a 0.2% lower validation accuracy and 0.05% lower test accuracy. Based on the results in Table 1, I would pick a model based on **highest validation accuracy** as this experiment has shown that it could represent the model's ability to generalize on data that it has *never seen* before by

Evan Tan 27401995
etan0008@student.monash.edu

correctly predict the labels of the data, and that training for a larger number of epochs is quite misleading.

This experiment shows that it is extremely important to separate a dataset into train, validation and test sets, and evaluate accuracies and losses across different datasets to obtain a better overall estimator of model performance.

In this case, there is significant overfitting of the model from approximately 80 epochs onwards where the model validation loss and training loss curves diverge, and where training loss continues to decrease but validation loss increases. This means that the model is perhaps learning and memorizing the training data instead of learning the problem, thus the learned model has a reduced ability to generalize on new data, shown by the test accuracy based on lowest training loss being 82.94%, 2.19% lower than the selected model using highest validation accuracy.

2.2

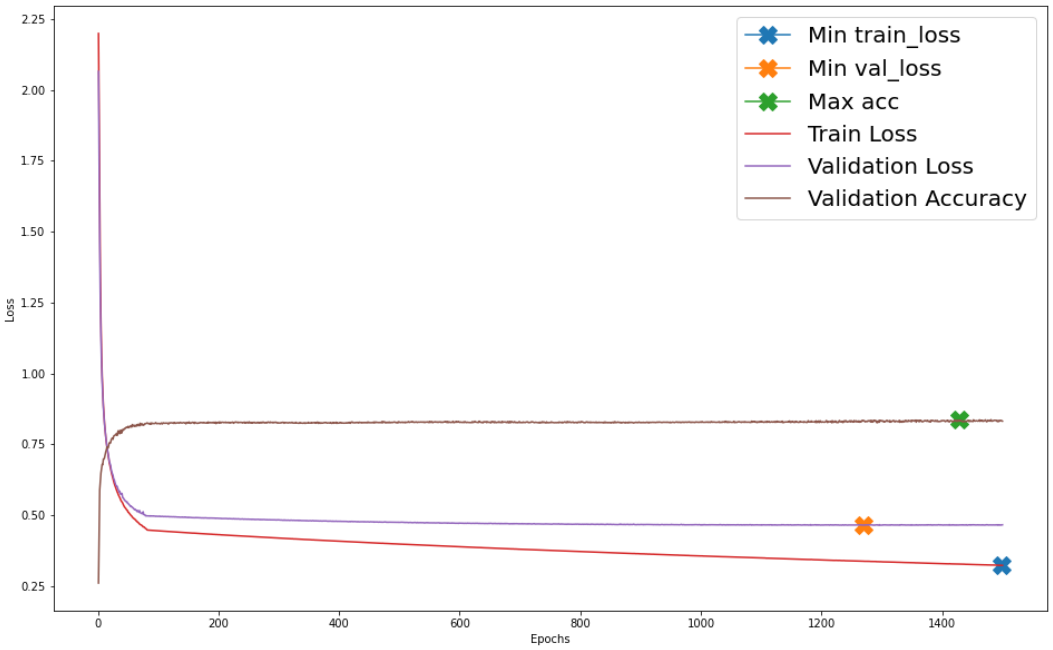


Fig. 7: Training loss, validation loss and validation accuracy for model using multistep learning rate of 1e-3 for epoch 1 to 79, and 1e-4 for epoch 80 to 1500, momentum = 0.9, batch size = 128, and 1500 epochs.

Model Criteria	Epoch Number	Training Loss	Validation Loss	Validation Accuracy (%)	Test Accuracy (%)
Lowest Training Loss	1500	0.323313	0.4665458	83.20	85.31
Lowest Validation Loss	1270	0.337842	0.464899	83.45	85.25
Highest Validation Accuracy	1430	0.327928	0.465672	83.60	85.27

Evan Tan 27401995
etan0008@student.monash.edu

Table 2: Metrics for model with multistep learning rate in Fig. X above

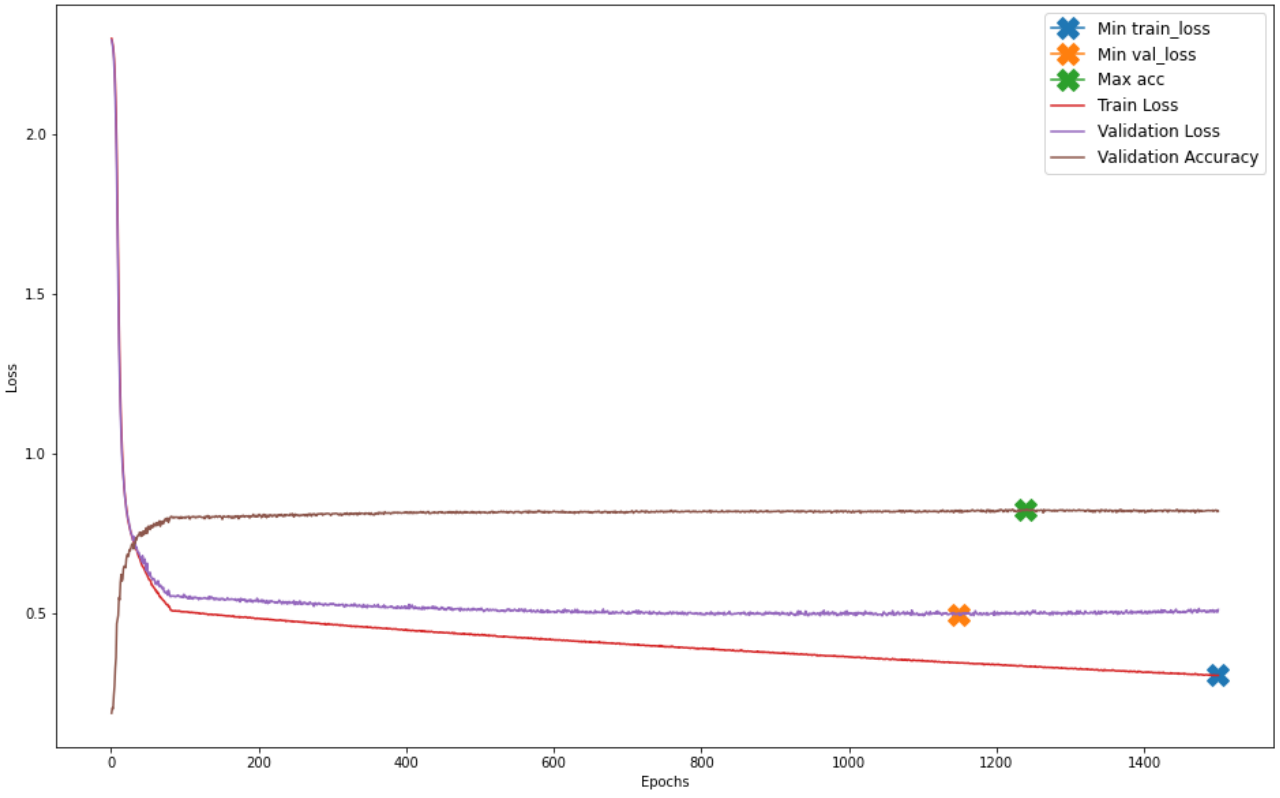
Using the same model in 2.1, and changing only the learning rate at epoch 80 onwards to reduce by a factor of 10 from 1e-3 to 1e-4, the following results were obtained in Fig. 7 and Table 2. With the same training parameters other than multistep learning rate, there is much more consistent test accuracy across the different models as seen in Table 2.

By reducing the learning rate at epoch 80, the updates made to the model are much smaller, thus the updates to the learned model are reduced by a factor of 10. The smaller updates benefit the model when the training loss has stopped changing significantly as there is less overfitting of the training data seen in Fig. 7 compared to Fig. 6 where there is less difference in the training loss and validation loss curves. Furthermore, as the loss landscape is quite complex, it would mean that the loss approaches a minima quickly but moves slowly as it gets closer and closer to the minima.

In Table 2, the maximum difference in test accuracy across different models is 0.06% in Table 2, compared to 2.19% in Table 1. This shows that using some learning rate decay (multistep in this case) is beneficial as it prevents the model from overfitting, ensuring more consistent performance from better generalization and less memorization.

3.1

I have chosen to select the models based on the **highest validation accuracy**.
To compare model baselines fairly between the Deep MLP and Shallow MLP fairly, *everything was kept the same except for model architecture* and the model performance between Q2.2 and Q3.1 is compared in Table 3.



Evan Tan 27401995
etan0008@student.monash.edu

Fig. 8: Training progress for DeepMLP, learning rate = 1e-3 for epoch 1 to 79, 1e-4 for epoch 80 to 1500, momentum = 0.9, batch size = 128 and 1500 epochs.

Model	Epoch Number	Training Loss	Validation Loss	Validation Accuracy (%)	Test Accuracy (%)
DeepMLP (Q3.1)	1240	0.335566	0.501744	82.60	84.39
ShallowMLP (Q2.2)	1430	0.327928	0.465672	83.60	85.27

Table 3: Metrics for model with multistep learning rate as seen in Fig. 8

Selecting models based on the highest validation accuracy, a model checkpoint at epoch 1240 for DeepMLP was selected. The training loss and validation loss were higher for DeepMLP by 0.007638 or 2.33% and 0.036072 or 7.74% respectively. Additionally, validation accuracy was lower by 1% and test accuracy was 0.88% lower for DeepMLP compared to ShallowMLP.

Increasing the depth of the neural network does not offer any improvement in performance as seen from Table 3, this could be due to the size of the thinnest hidden layer having the same dimensionality i.e. 32 as before. Furthermore, the difference in performance could be due to the difference in random initialization across runs.

3.2

The model with the new architecture is referred to as **DeepWideMLP.**
To compare DeepMLP and the DeepWideMLP, all training parameters from Q2.2, and Q3.1 were used in order to test the effects of changing the model architecture.

Evan Tan 27401995
etan0008@student.monash.edu

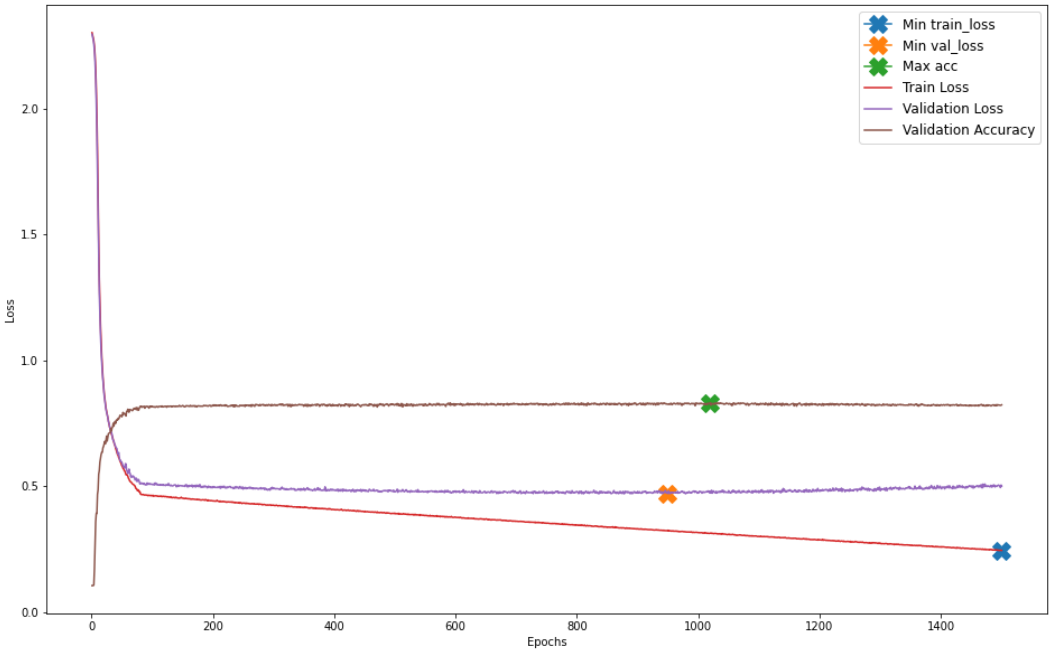


Fig. 9: Training progress for DeepMLP, learning rate = 1e-3 for epoch 1 to 79, 1e-4 for epoch 80 to 1500, momentum = 0.9, batch size = 128 and 1500 epochs.

Evan Tan 27401995
etan0008@student.monash.edu

Model	Epoch Number	Training Loss	Validation Loss	Validation Accuracy (%)	Test Accuracy (%)
DeepWideMLP (Q3.2)	1010	0.314494	0.475864	82.95	85.21
DeepMLP (Q3.1)	1240	0.335566	0.501744	82.60	84.39

Table 4: Metrics for model with multistep learning rate as seen in Fig. 9

By increasing the width of the neural network in DeepMLP to DeepWideMLP, the model has improved very slightly. Training loss has reduced from 0.335566 to 0.314494, a 6.28% improvement, validation loss has decreased from 0.501744 to 0.475864, a 5.16% improvement, validation accuracy has increased by 0.30% from 82.60% to 82.95% and test accuracy has improved by 0.82% from 84.39% to 85.21%.

However, by increasing the width of the model, the number of parameters that are being learned during training are quadrupled as seen in Q3.3, thus the DeepWideMLP architecture's miniscule performance increase compared to four times the number of parameters is not a better network and is not an efficient design choice.

3.3

For the DeepMLP in 3.1, there are 29,642 trainable parameters, including weights and biases.
For the DeepWideMLP in 3.2, there are 118,346 trainable parameters, including weights and biases.
These values were obtained by manual calculation (by summing all the connections and biases in the MLPs) and confirmed using the **torchsummary package**, but can also be manually calculated.

3.4

```
CarbonTracker: The following components were found: GPU with device(s) NVIDIA GeForce RTX 3070.
CarbonTracker:
Actual consumption for 1 epoch(s):
        Time:   0:00:00
        Energy: 0.000010 kWh
        CO2eq:  0.003026 g
        This is equivalent to:
        0.000025 km travelled by car
CarbonTracker:
Predicted consumption for 1500 epoch(s):
        Time:   0:11:39
        Energy: 0.015426 kWh
        CO2eq:  4.538467 g
        This is equivalent to:
        0.037695 km travelled by car
```

Clearly I have done a disservice to the environment, oops.

The carbon footprint of training the model in 3.2 for 1500 epochs, with an NVIDIA 3070 and Ryzen 5900X is **4.538467g**.

Evan Tan 27401995
etan0008@student.monash.edu

4.1

When changing the optimizer from SGD to Adam, I have reduced the learning rate by a factor of 5, from 1e-3 to 2e-4 as the Adam optimizer typically uses a smaller learning rate. Furthermore, I have removed the multistep learning rate scheduler which has been used in Q2.2, Q3.1 and Q3.2.

The model I have designed uses 3 hidden layers as compared to 2 hidden layers in Q3, and is both deeper and wider than the models in Q3. I have done this so that perhaps the model could learn perhaps lower level representations of the problem and maybe combine stitch it with higher level representations to correctly predict labels.

From the training progress in Fig. 10, the test accuracy peaks at epoch 40, and steadily decreases after. A possibility for this is that the model is overfitting, as Adam seems to converge faster than SGD with momentum. However, the test accuracy cannot be compared to the models in Q3 as the models in that question use a different dataset.

As seen in Q2, training loss itself is not a good indicator of the performance of the model as training loss will always be decreasing, but the validation loss and accuracy will help indicate whether the model is overfitting. To determine underfit, the training loss would not even converge so that can still be inferred from the training loss curve alone. From training this model, it could be misleading the viewer as one might think a model with lower loss is better, but the model checkpoint at epoch 500 with a very low loss of 0.041344 scores 66.92% on the test dataset, whereas the model checkpoint at epoch 40 with a much higher loss of 1.263527 scores 84.19% on the test dataset.

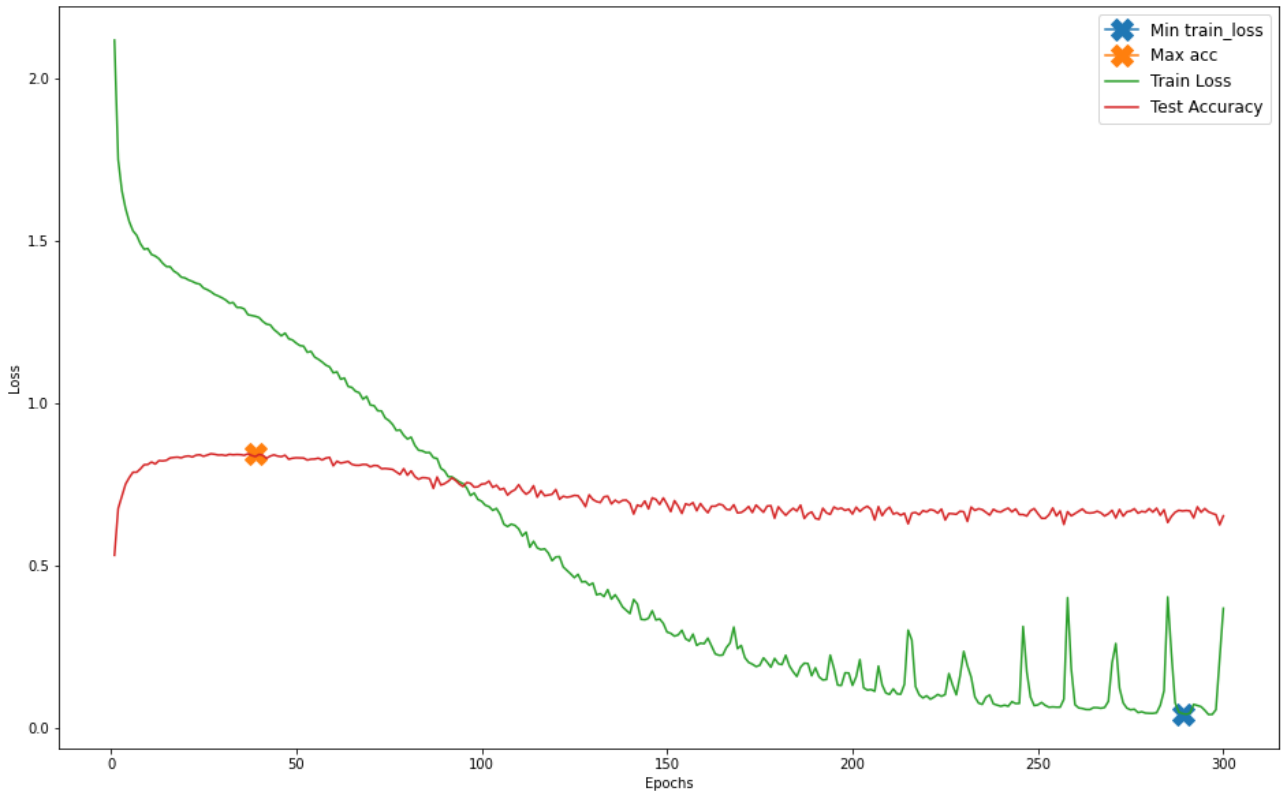


Fig. 10: Training progress for CustomMLP with learning rate = 2e-4, batch size = 128