Code can be found here https://colab.research.google.com/drive/1W9t3YAlNqXccIQJSBUiKrgHEH9B1gxzz?usp=sharing

Question 1 - Optimization (2.5 Marks)

1.1

Let the current epoch be t,
We start off with $t = 0$, $x_0 = 0$.
From x = 0 to 1, the gradient is $\frac{\partial L}{\partial x} = \frac{(1-0)}{(0-1)} = -1$

When $t = 1$, $x_1 = 0 - (0.3)(-1) = 0.3$
When $t = 2$, $x_2 = x_1 - (0.3)(-1) = 0.6$
When $t = 3$, $x_3 = x_2 - (0.3)(-1) = 0.9$
When $t = 4$, $x_4 = x_3 - (0.3)(-1) = 1.2$

From x = 1 to 1.5, the gradient is $\frac{\partial L}{\partial x} = \frac{0.5-0}{1.5-1} = 1$
When $t = 5$, $x_5 = x_4 - (0.3)(1) = 0.9$

The steps in the fourth and fifth epoch will keep on repeating and x will be stuck in the local minima.

1.2

h = 0.41

$\beta_1 = 0.9, \beta_2 = 0.999$
In the first epoch,
$m_0 = 0.9 * 0 + (1 - (0.9)) * (-1) = -0.1$
$v_0 = (0.999) * 0 + (1 - (0.999) * (-1)^2 = 0.001$

$\hat{m} = m_0/(1-(0.9)^1) = -1$
$\hat{v} = v_0/(1-(0.999)^1) = 1$
$\theta_0 = 0 - 0.3 * \hat{m}/\sqrt{\hat{v}} + 0 = 0.3$

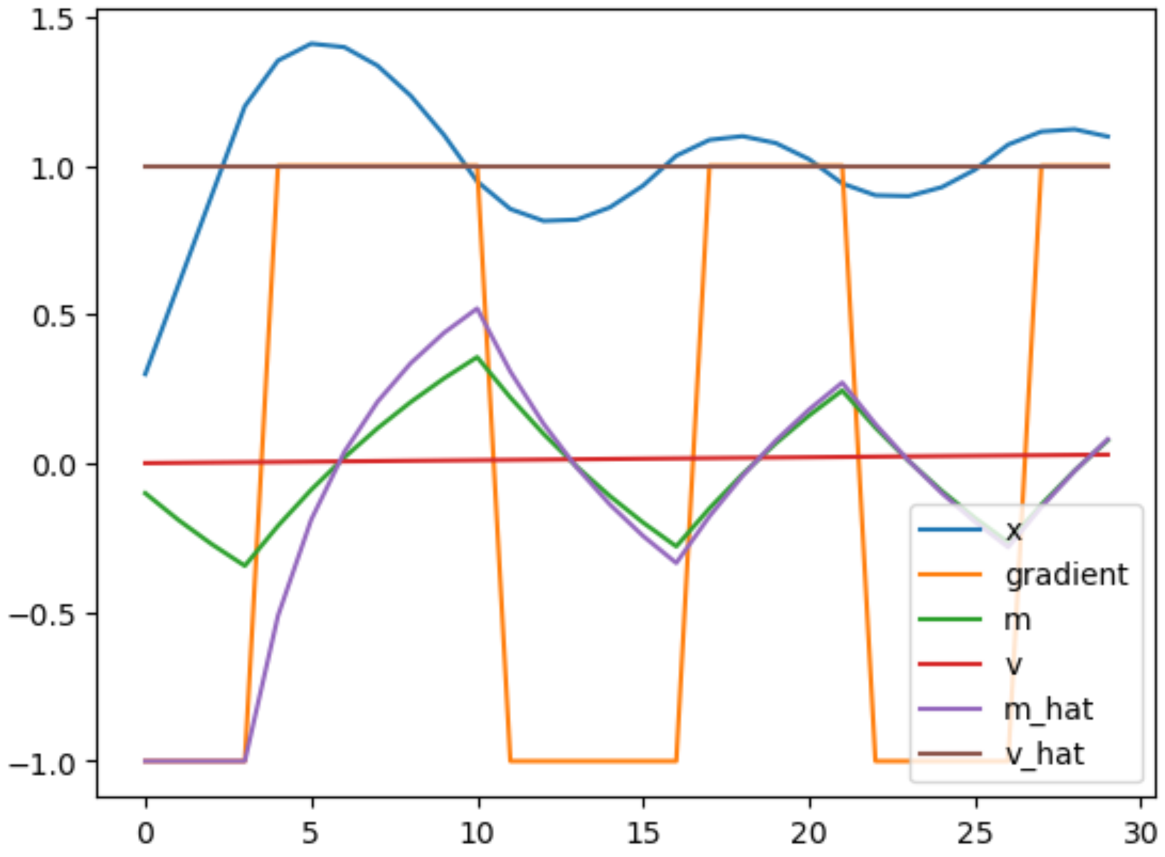
In the second epoch,
$m_1 = 0.9 * m_0 + (1-(0.9)) * (-1) = -0.19$
$v_1 = (0.999) * v_0 + (1-(0.999) * (-1)^2 = 0.00199$
$\hat{m} = m_1/(1-(0.9)^2) = -1$
$\hat{v} = v_1/(1-(0.999)^2) = 1$
$\theta_0 = 0.3 - 0.3 * \hat{m}/\sqrt{\hat{v}} + 0 = 0.6$

...

```
Epoch: 0, theta: 0.30000, gradient: -1.0, m: -0.10000, v: 0.00100, m_hat: -1.00000, v_hat: 1.00000
Epoch: 1, theta: 0.60000, gradient: -1.0, m: -0.19000, v: 0.00200, m_hat: -1.00000, v_hat: 1.00000
Epoch: 2, theta: 0.90000, gradient: -1.0, m: -0.27100, v: 0.00300, m_hat: -1.00000, v_hat: 1.00000
Epoch: 3, theta: 1.20000, gradient: -1.0, m: -0.34390, v: 0.00399, m_hat: -1.00000, v_hat: 1.00000
Epoch: 4, theta: 1.35348, gradient: 1.0, m: -0.20951, v: 0.00499, m_hat: -0.51161, v_hat: 1.00000
Epoch: 5, theta: 1.41018, gradient: 1.0, m: -0.08856, v: 0.00599, m_hat: -0.18900, v_hat: 1.00000
Epoch: 6, theta: 1.39851, gradient: 1.0, m: 0.02030, v: 0.00698, m_hat: 0.03891, v_hat: 1.00000
```

After epoch 5, adam will not have sufficient momentum to escape the local minima. Thus, h = 0.41 (to 2 d.p.)

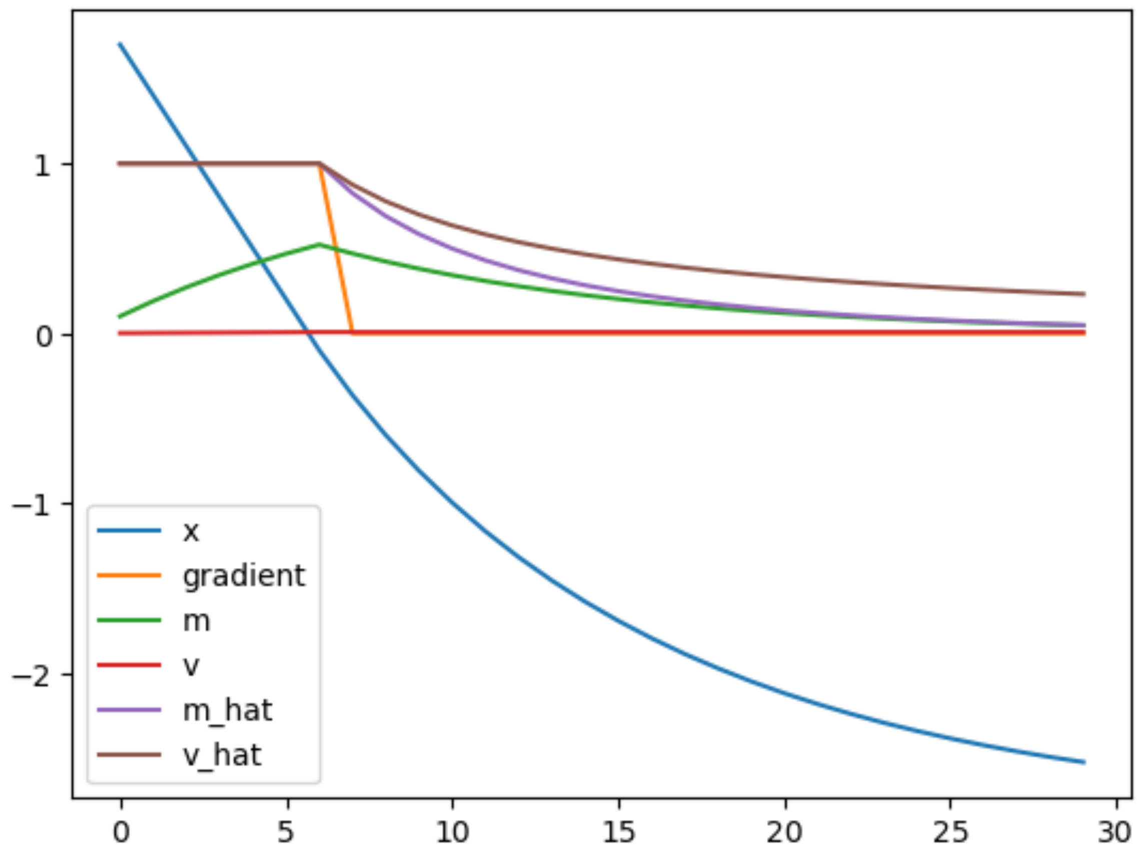
1.3



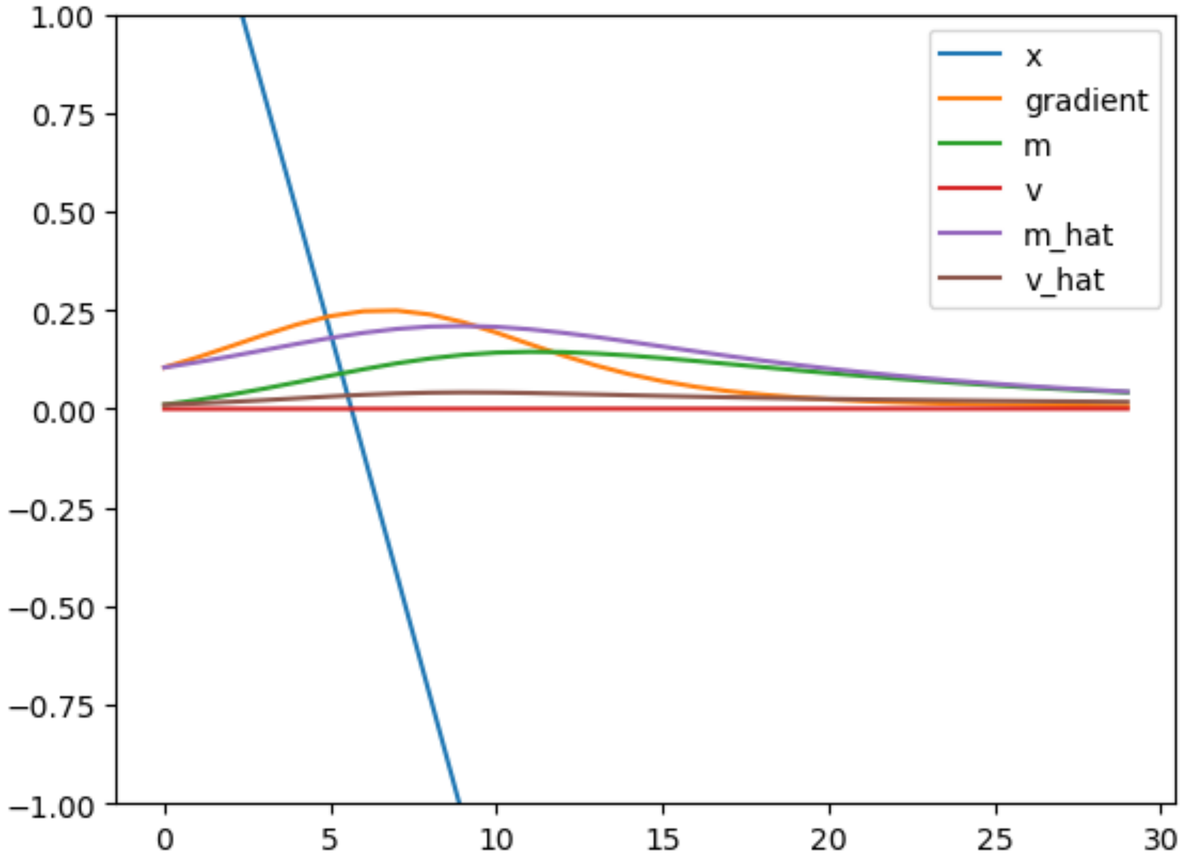
The adam gradient descent gets stuck in the local minima (i.e. x can't exceed ~1.4). As it continues to remain stuck, it gradually loses its momentum as can be seen by the lower and lower peaks in m and $\hat{m}$ values as t increases.

v and $\hat{v}$ are used for z-scoring, but in this case, the variance is too small and doesn't really contribute.

1.4

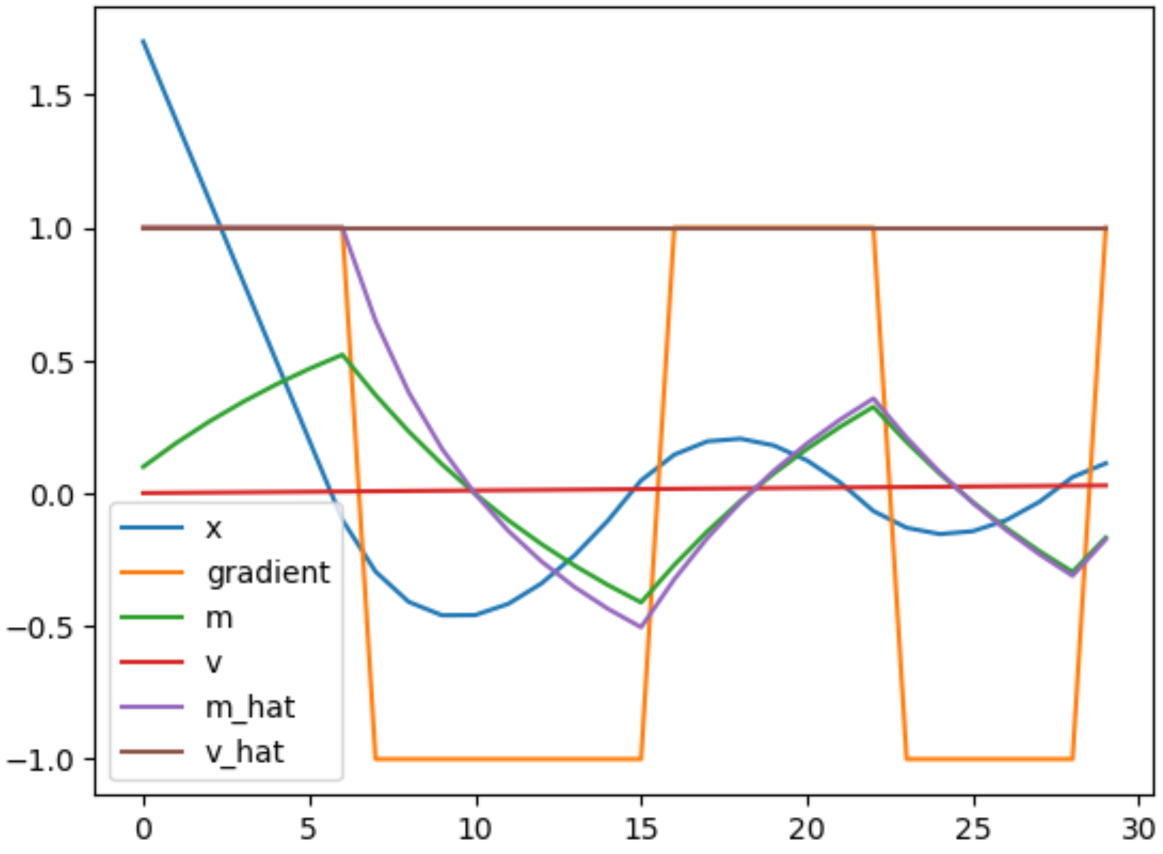


For ReLU(x), at x = 2, the gradient is 1. Thus, for the gradient descent, the adam algorithm starts to "move backwards". Since ReLU has a zero gradient when x < 0, the adam algorithm will trickle backwards until it loses its momentum.



Similar behavior to ReLU except with different, smoother gradients. The adam algorithm will continue gradient descent moving backwards and stagnate as it loses momentum.

Since sigmoid doesn't have a sharp dropoff to 0 gradient unlike ReLU, the gradient descent is able to progress much further (i.e. more negative) than ReLU.

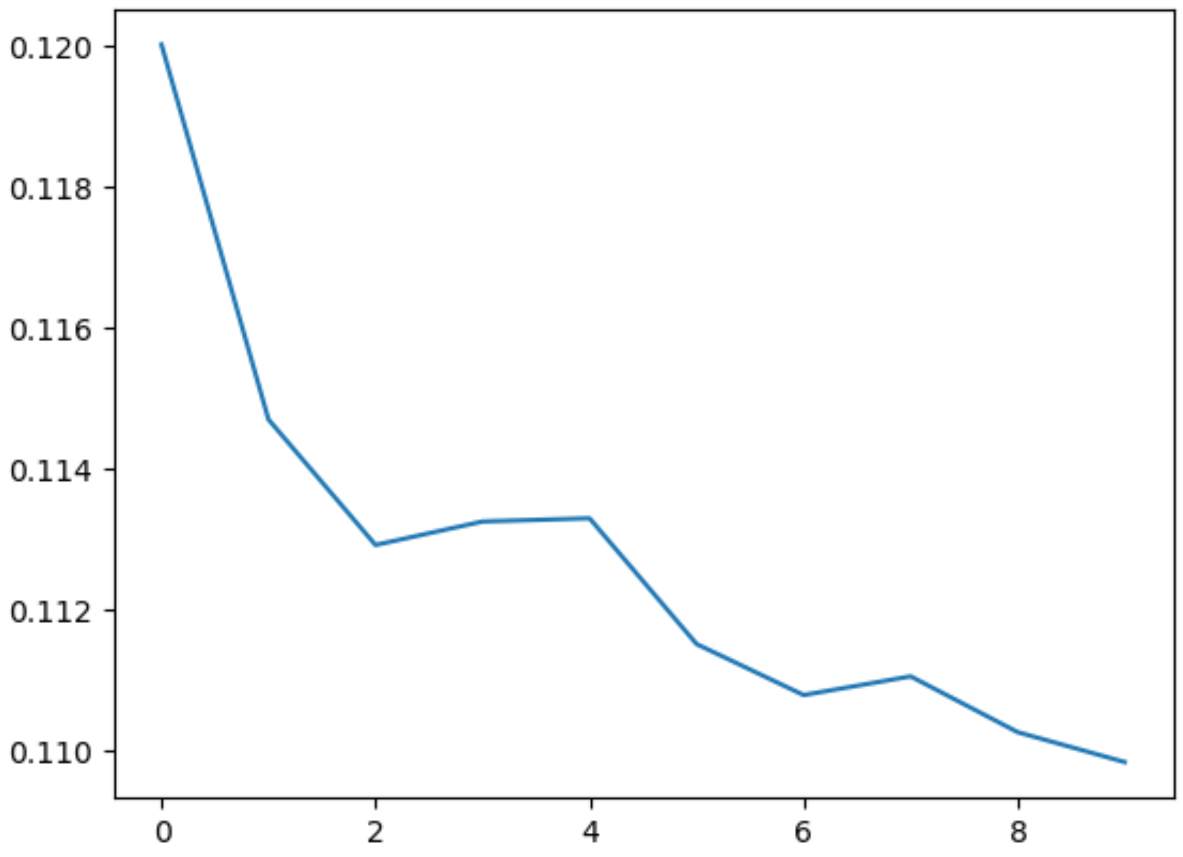


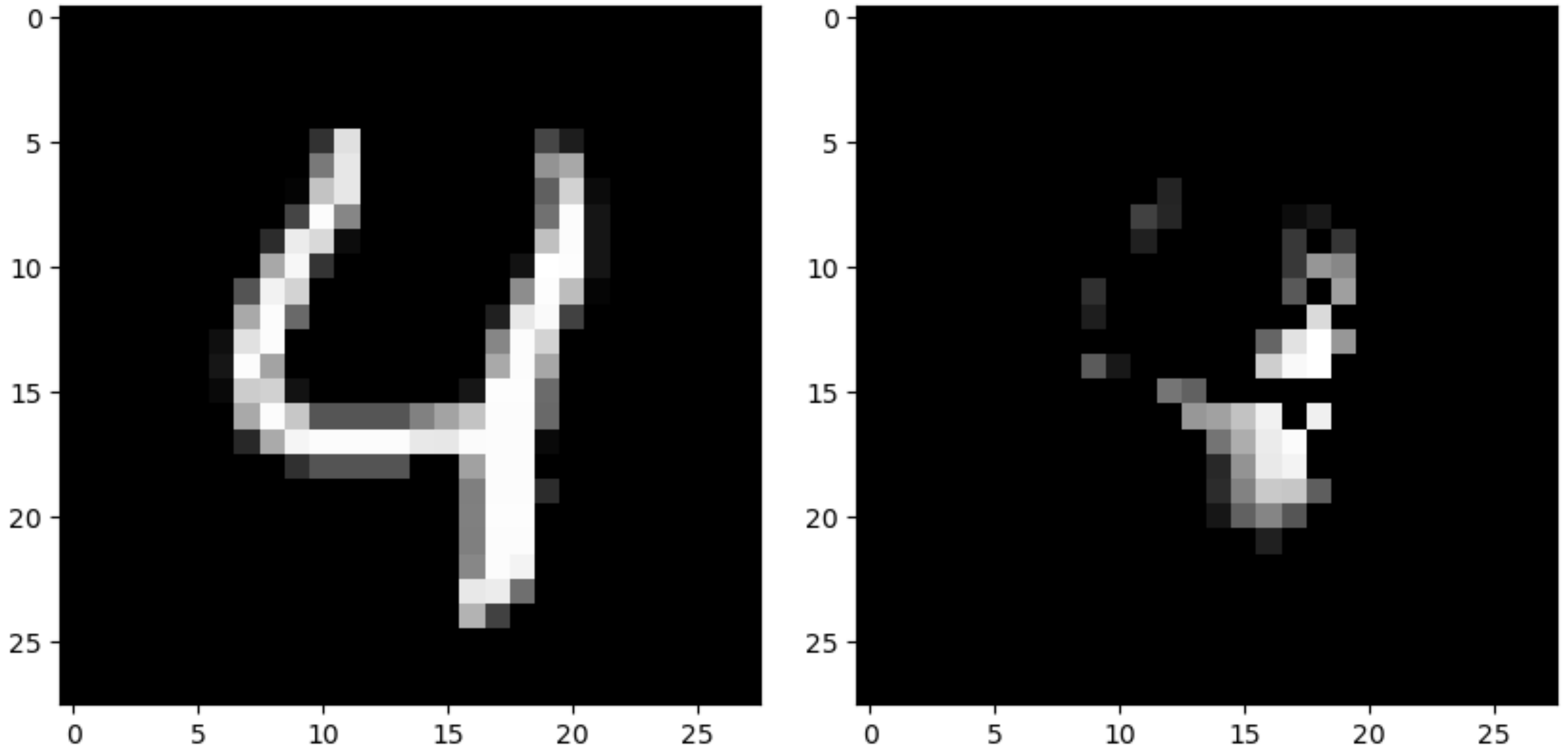
The algorithm alternates between the positive and negative gradients of the |x| function, losing momentum with each cycle and is expected to eventually converge at x = 0.

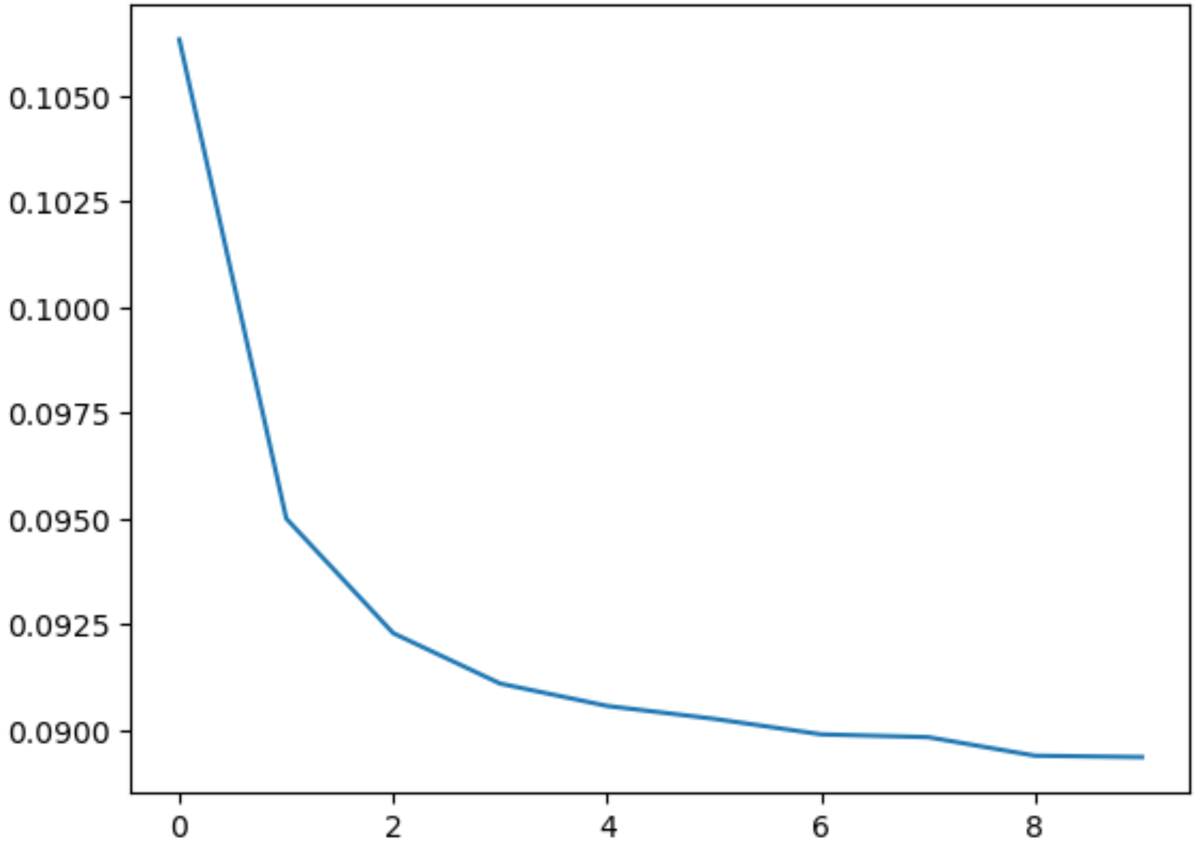
Question 2 - Autoencoder

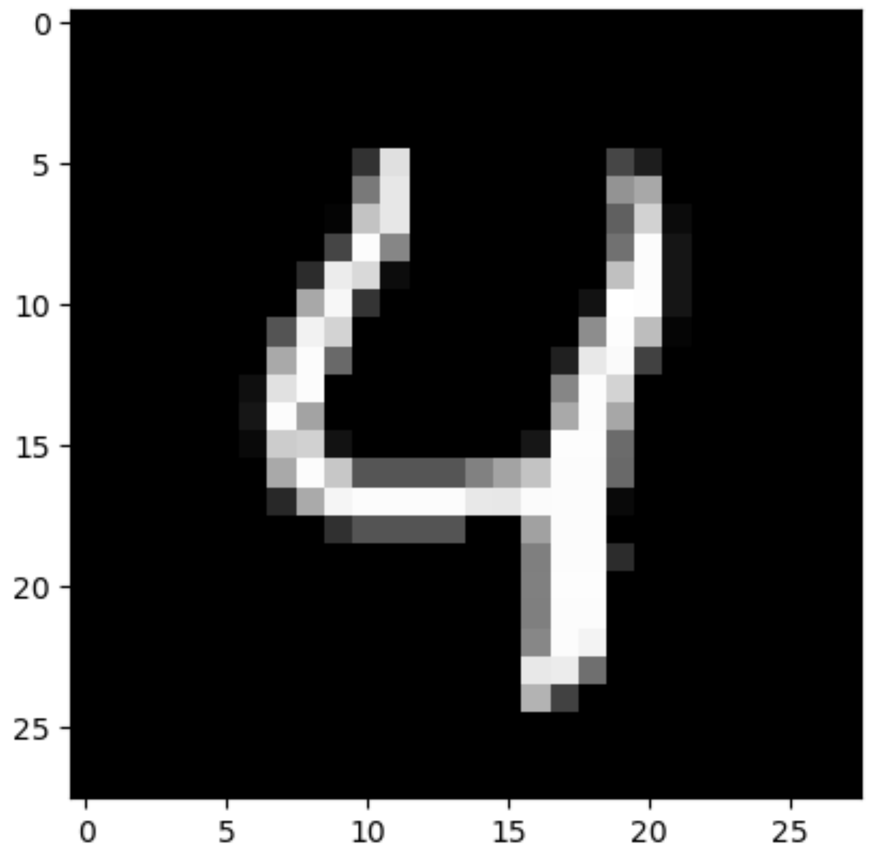
2.1

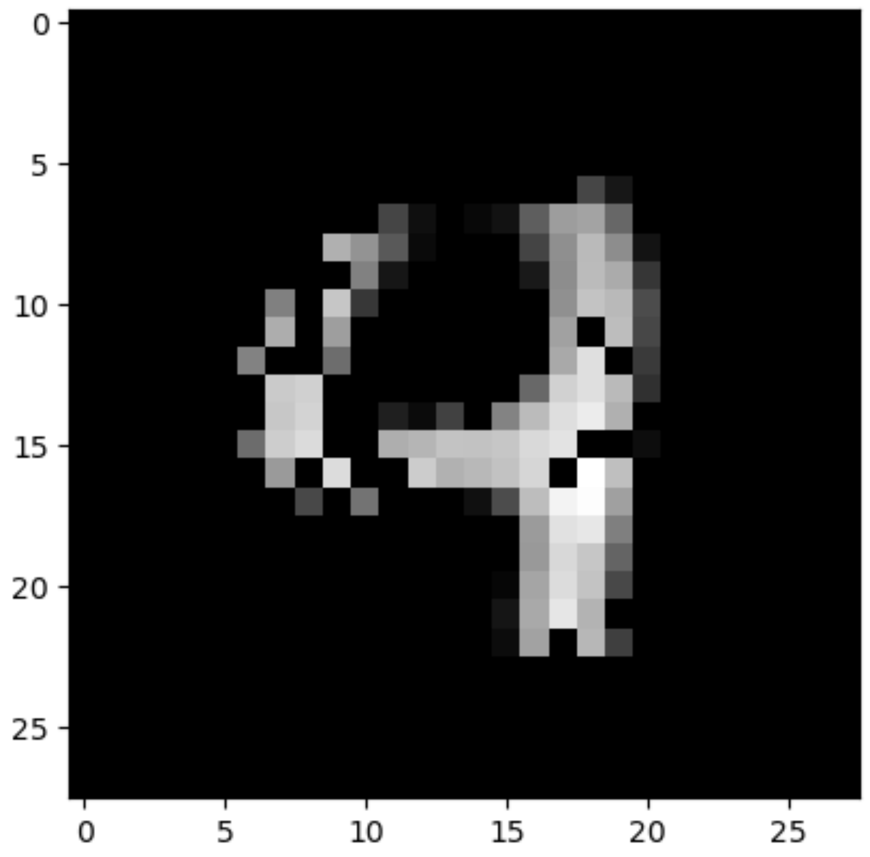
The smaller the latent space, the more "lossy" the autoencoder. This can be seen by the blurer features, less pixels, more noise etc. The loss value is also higher and the model tends to stagnate earlier as compared to $\lambda = 32$ where the model is able to make more improvements with more iterations.




0
5
10
15
20
25
0
5
10
15
20
25


0.1050
0.1025
0.1000
0.0975
0.0950
0.0925
0.0900
0
2
4
6
8



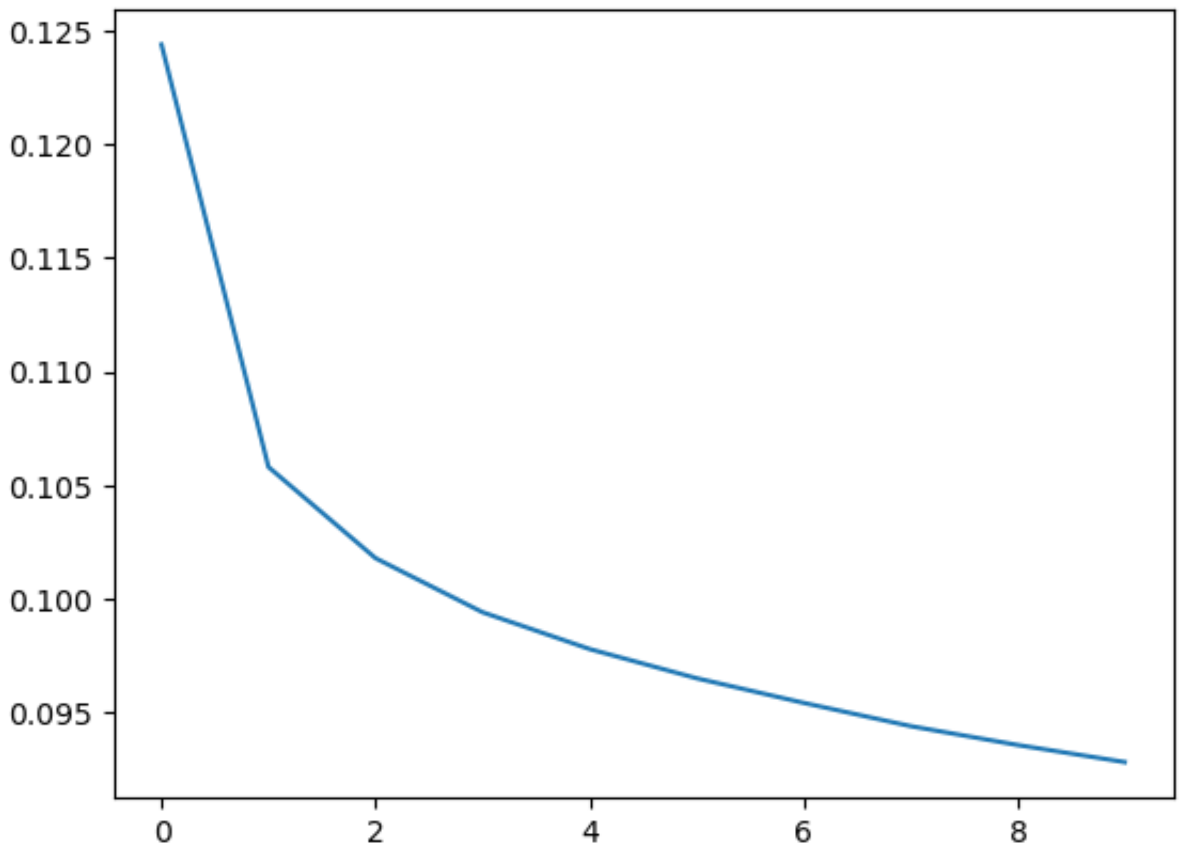


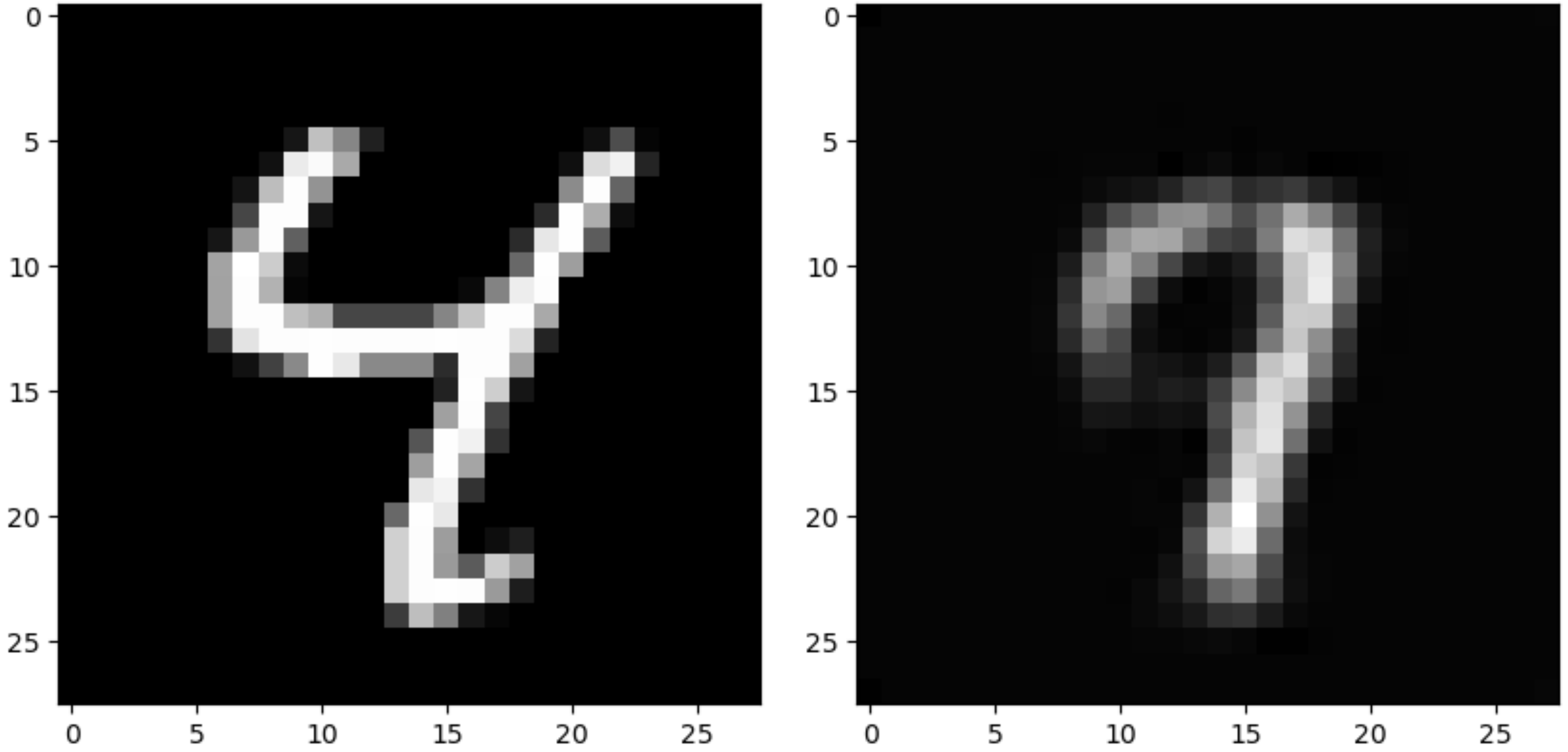
2.2

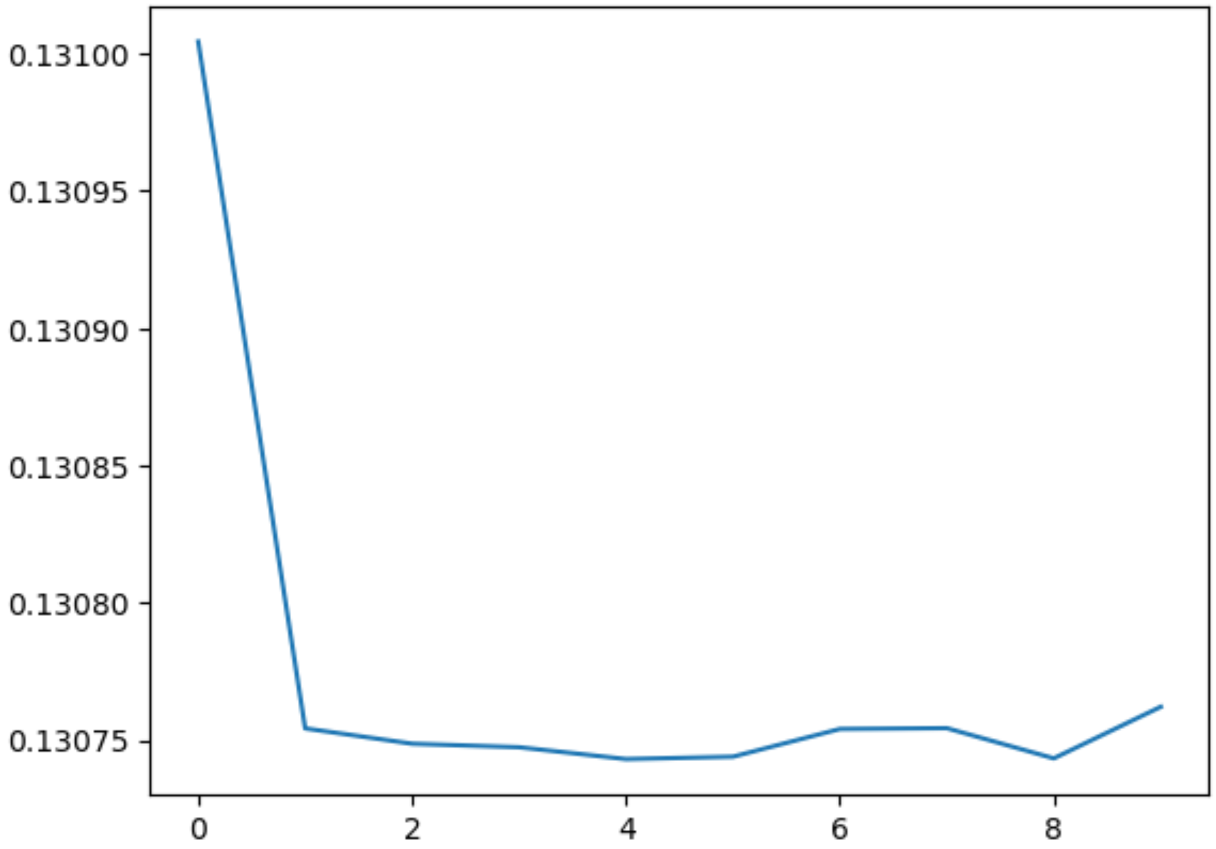
The results tends to be better in the CNN autoencoders than the typical autoencoders.

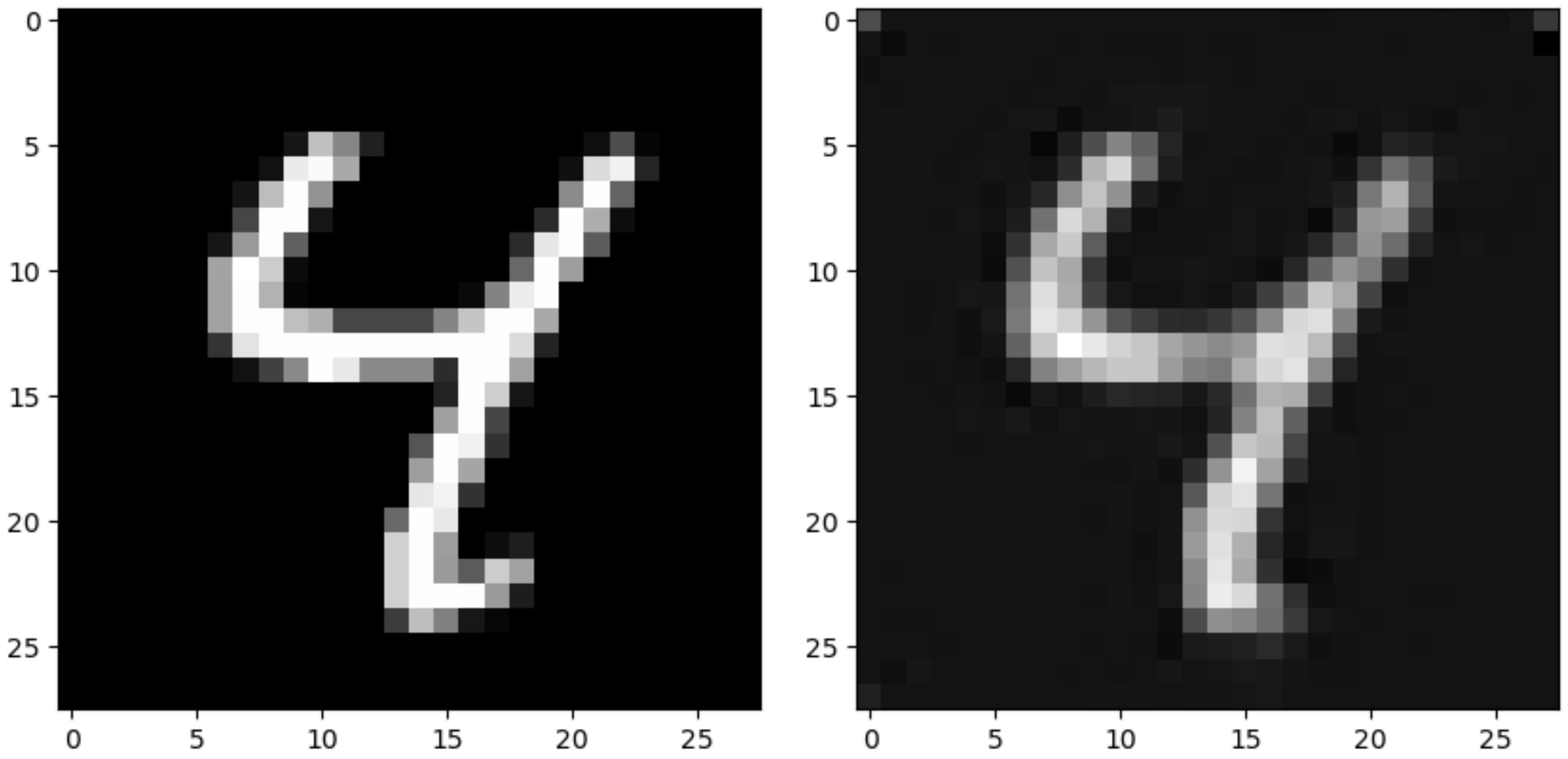
Similar to the non-CNN autoencoder, when $\lambda = 2$, the latent space is a lot smaller and the amount of information passed onto the decoder is also a lot lesser. This results in outputs that feel more "compressed" and retaining less features. We can see it in the outputs whereby $\lambda = 2$ tends to be blurer and have less features as compared to $\lambda = 32$.

One difference between CNN autoencoders and the non-CNN autoencoder is that the images seem to be "smoother" (e.g. non-cnn has very sharp holes compared to cnn where pixels have like an anti-aliasing effect). This makes sense as the CNN uses the kernel to take into account surrounding pixels rather than individual pixels.




0
5
10
15
20
25
0
5
10
15
20
25

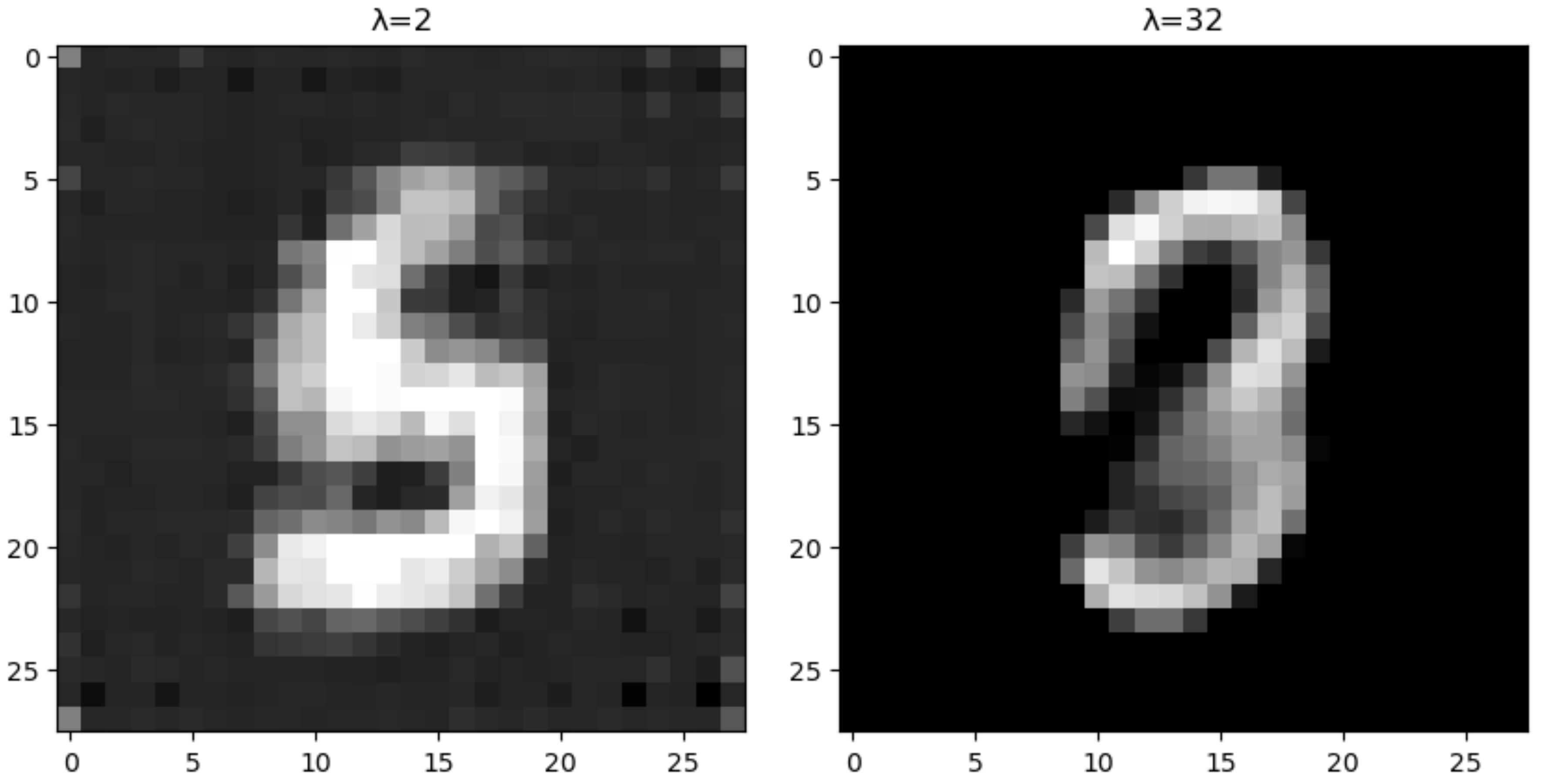

0.13100
0.13095
0.13090
0.13085
0.13080
0.13075
0
2
4
6
8



Question 3: Common features

3

Both have similar shapes, but $\lambda = 32$ has better defined features than $\lambda = 2$. This makes sense since we are taking the mean of the latent spaces and $\lambda = 2$ is a smaller latent space.



Question 5: CIFAR10

Question 5.1

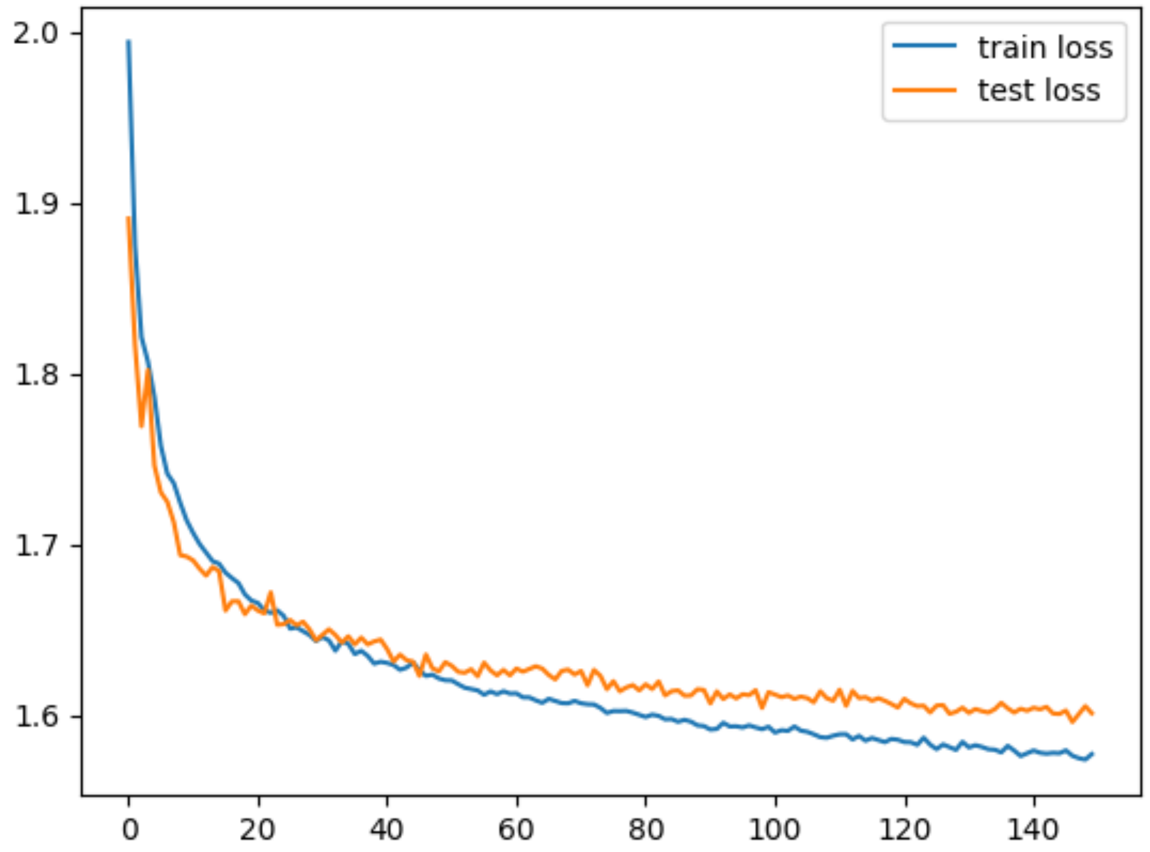
No.

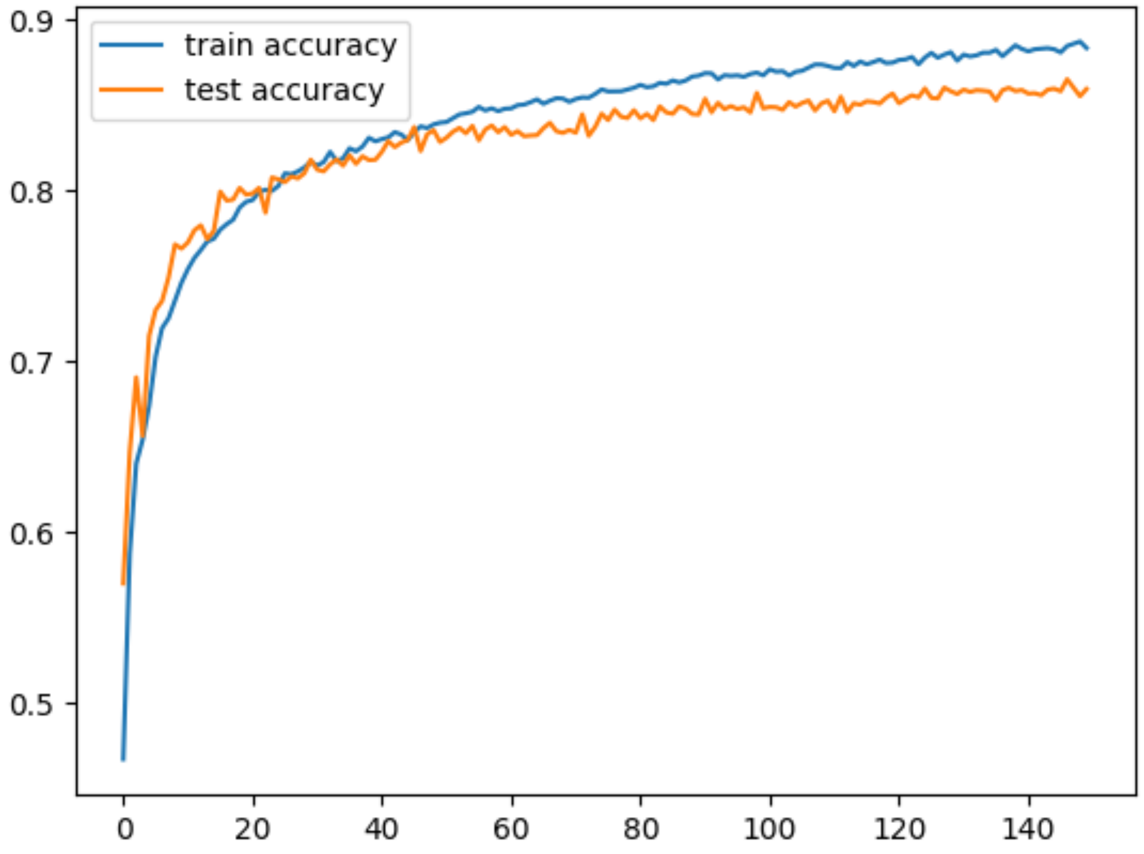
Although it was possible to get really high training accuracy (>90%) with just a simple architecture, the test accuracy often lagged behind. This showed that I needed better regularisation techniques in order to allow my model to generalise better into the test dataset. The following techniques were used:

- Data Augmentation
- Dropout
- Batch Normalisation

With these regularisation techniques, test and training accuracy started to be a lot closer. However, it was still difficult to obtain above 90% as later runs only resulted in marginal gains.

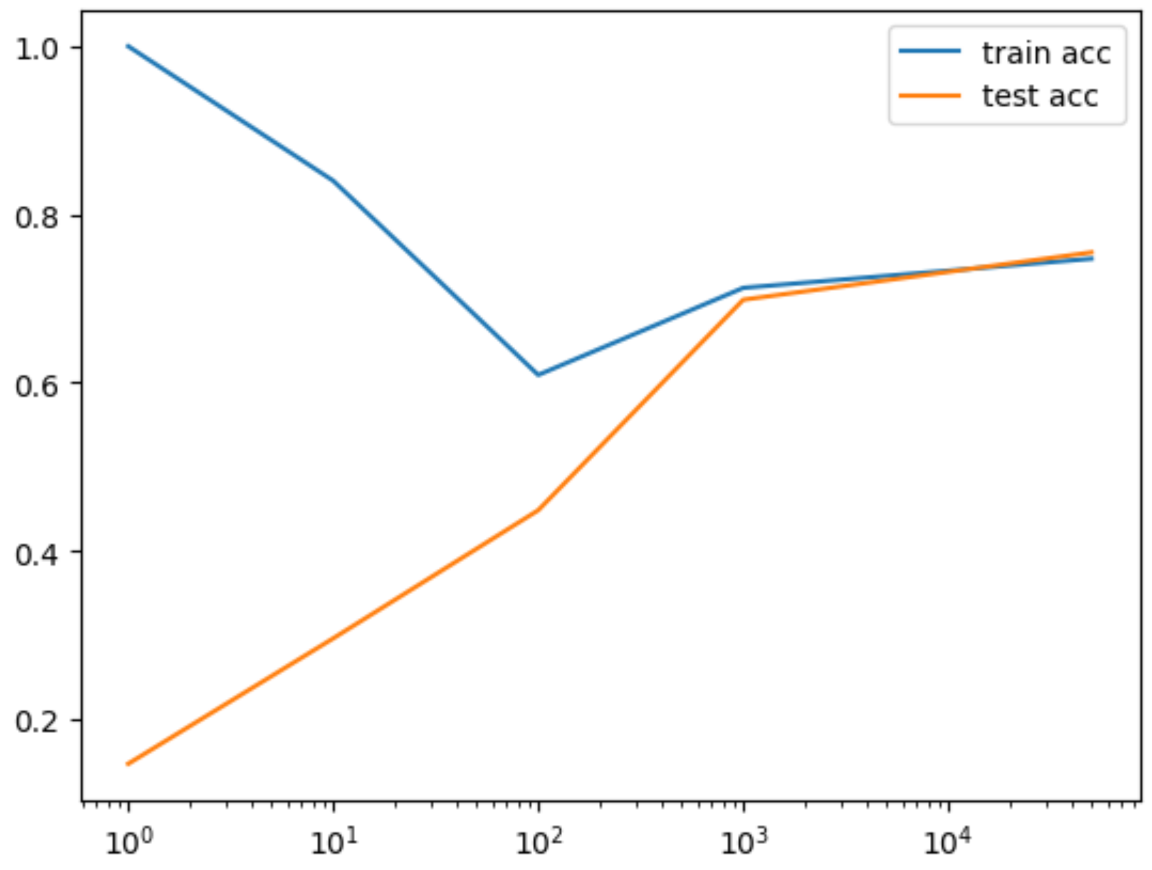
This particular architecture seemed to still to be able to improve further. However, training was starting to get too expensive (>150 epochs).





train accuracy
test accuracy
0.9
0.8
0.7
0.6
0.5
0
20
40
60
80
100
120
140

5.2



Question 6: Receptive field

S.No	layer1	layer2	layer3	layer4	layer 5	receptive field
1	conv2d(3,1)	conv2d(3,1)	conv2d(1,1)	maxpool2d(2,2)	conv2d(2,2)	8x8
2	conv2d(3,1)	maxpool2d(2,2)	conv2d(2,2)	maxpool2d(2,2)	conv2d(3,1)	26x26
3	conv2d(5,1)	maxpool2d(3,1)	conv2d(4,1)	maxpool2d(2,2)	conv2d(3,1)	15x15
4	conv2d(7,2)	conv2d(5,1)	conv2d(3,2)	conv2d(3,1)	conv2d(1,1)	27x27

5	conv2d(4,1)	conv2d(3,2)	maxpool2d(2,2)	conv2d(3,1)	maxpool2d(2,2)	20x20
6	conv2d(7,2)	conv2d(5,2)	maxpool2d(2,2)	conv2d(5,2)	conv2d(7,2)	147x147
7	conv2d(3,3)	maxpool2d(2,2)	conv2d(5,3)	maxpool2d(3,3)	conv2d(5,1)	282x282
8	conv2d(4,2)	maxpool2d(2,2)	conv2d(3,1)	conv2d(1,1)	conv2d(2,1)	18x18
9	conv2d(4,2)	maxpool2d(2,2)	conv2d(3,1)	conv2d(1,1)	conv2d(2,1)	18x18
10	conv2d(11,2)	conv2d(7,2)	maxpool2d(2,2)	conv2d(5,2)	conv2d(3,1)	91x91