LAB REPORT 1 [LAB1]

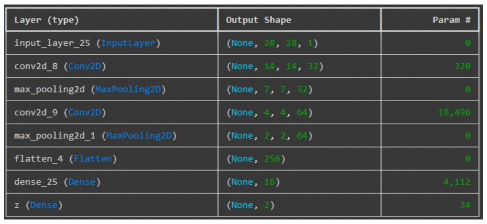
SANTIAGO VILLARREAL VILLARRAGA
8043768
VILLARRS@MYUMANITOBA.CA
07-02-2025

Contents

1. Problems

P1-1 Select a different architecture.

For the new architecture, I wanted less number of params, so I decided only to change the encoder, the params went from 65k to 23k on the new encoder, which can be seen in the figure - 1, I used max-pooling and a flatten layer.



Layer (type)	Output Shape	Param #
input_layer_25 (InputLayer)	(None, 28, 28, 1)	0
conv2d_8 (Conv2D)	(None, 14, 14, 32)	320
max_pooling2d (MaxPooling2D)	(None, 7, 7, 32)	0
conv2d_9 (Conv2D)	(None, 4, 4, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 2, 2, 64)	0
flatten_4 (Flatten)	(None, 256)	0
dense_25 (Dense)	(None, 16)	4,112
z (Dense)	(None, 2)	34

Figure 1. new architecture.

In this new architecture, I could see faster training, but I was using Colab with GPU therefore the change was for 1-2 seconds for each epoch. This improvement in training time didn't help in the output since the loss went to 136 only improving from 140 in the original model. the loss can be seen in the figure - 2

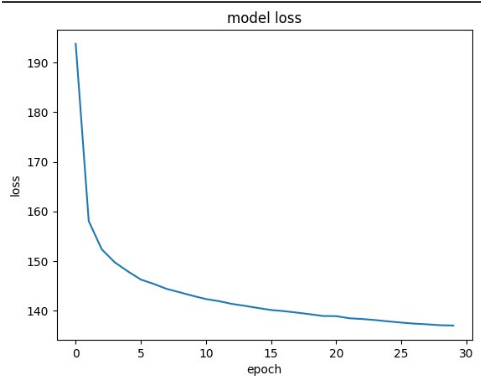


FIGURE 2. new architecture.

P1-2 Uses a different latent space.

I decided to try using a latent space of 1 as first try in order to know how different it was from the original, which can be seen in figure - 3 the loss didn't go down as much as the original, the latent space went to a loss of 157.8 while the original was in 140.

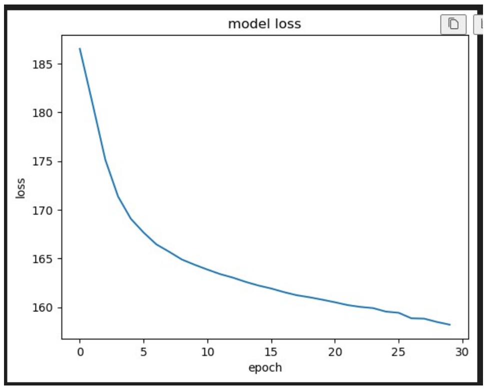


FIGURE 3. loss of the latent space 1.

Thanks to the previous I decided to use a latent space of 10, I wanted to use this thanks to the label of the data set, wich go from 0 to 9, so 1 space per number, with this the loss went to around 75, and how can be seen in the figure - 4 the loss decreased faster than the latent space of 1.

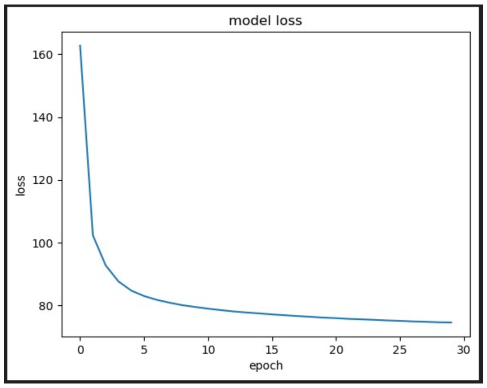


FIGURE 4. loss of the latent space 10.

In both cases, the training time took almost the same time. all the results of the better performance may be explained by the amount of data that the decoder is receiving, with a bigger latent space the decoder has more info to recreate the images. I used google colab with GPU since I only have a laptop without GPU to compile, thanks to that I didn't notice a big change in the compile time, but there shouldn't be a big difference since only 10 more parameters are being added.

P1-3 Use a different loss function.

I decided to use a mean square error to see how the regression error affects the loss, with the same parameters the error could be at 30.2 improving the overall loss, I suppose that this happens because of the nature of the loss functions, since the data set is not 0 or 1 but it has more values, the binary cross entropy penalize it an can produce larger values. also since this problem is a regression problem and not probabilistic (we are not dealing with VAE) the mean square error can be a better solution for it.

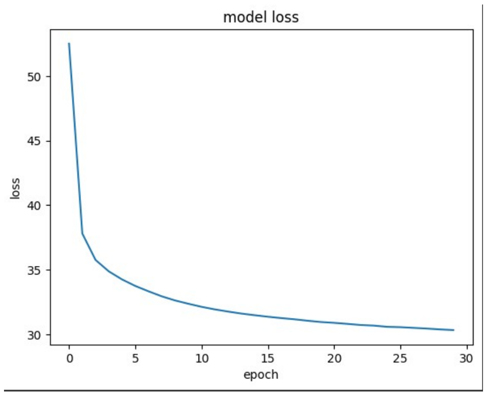


FIGURE 5. Loss with mean square error.

P1-4 Use a different data set

For the new dataset, I chose SVHN Cropped, which contains images of street house numbers. I selected this dataset because numbers are easy to evaluate, and all images share a fixed size of 32x32x3 with different colors. This consistency in dimensions, combined with the complexity of real world number images, makes it a challenging dataset for training. However, its manageable size helps keep compilation times low.

For the architecture I tried to use the original as the base just changing the input and output, the new input of the encoder is 32x32x3 and the output is 128, i selected a big latent space for the complexity of the data set but a bigger one could be better for a final solution. For loss function, I chose the mean square error since the new data has more values than 0 or 1. The learning rate was in learning rate 0.0001, a bigger one took the loss to infinity.

For the loss it went to 5.6 after 100 epochs with a batch of 512, I decided it thanks to having a large data set (1.47 Gb of photos), but even it wasn't enough how can we see in the figure - 6, where the AE took the general idea of the image having similar colors but it wasn't able to reconstruct the number.

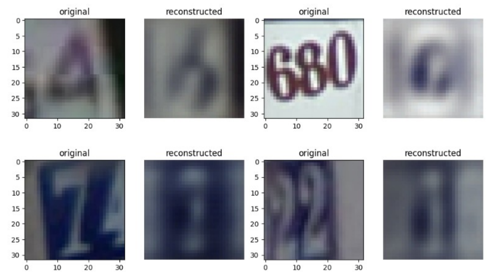


Figure 6. input and output AE RGB.

I tried using a batch of 1020 and 400 epochs but the loss stays above 5

P2-1 Inpaiting.

For all the problems in the second part I will be using the original data set and code but with a latent space of 10 and a mean square error loss function, for the reason that all that changes improved the model.

For inpainting, I used code from a Kaggle tutorial by the user endofnight.17j03 to delete parts of an image. I also adapted its logic to add noise to a batch of images by placing zeros in a random 5x5 pixel box. As shown in Figure - 7, the autoencoder performed well within the trained epochs.

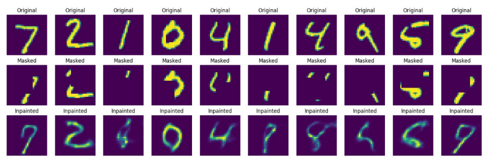
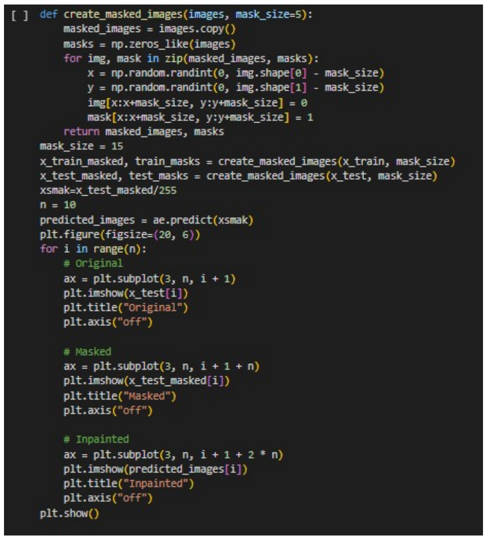


Figure 7. Minst inpainting.

The code takes the first 10 images of the set, then a function creates the mask and is added to the image, and then this image with a mask is used as input in the autoencoder, for doing this and the next 2, it was needed to add a function call() to use de AE as prediction. What is found in this generation is that the inpainting works better when the part that is deleted does contain key details in the number, for example, in the 7th image that was a 4, almost the entire digit was deleted, and the AE predicted an 8, but in other cases like the 2nd prediction, is easily intuitive that is a 2.



```
[ ] def create_masked_images(images, mask_size=5):
        masked_images = images.copy()
        masks = np.zeros_like(images)
        for img, mask in zip(masked_images, masks):
            x = np.random.randint(0, img.shape[0] - mask_size)
            y = np.random.randint(0, img.shape[1] - mask_size)
            img[x:x+mask_size, y:y+mask_size] = 0
            mask[x:x+mask_size, y:y+mask_size] = 1
        return masked_images, masks
    mask_size = 15
    x_train_masked, train_masks = create_masked_images(x_train, mask_size)
    x_test_masked, test_masks = create_masked_images(x_test, mask_size)
    xsmak=x_test_masked/255
    n = 10
    predicted_images = ae.predict(xsmak)
    plt.figure(figsize=(20, 6))
    for i in range(n):
        # Original
        ax = plt.subplot(3, n, i + 1)
        plt.imshow(x_test[i])
        plt.title("Original")
        plt.axis("off")

        # Masked
        ax = plt.subplot(3, n, i + 1 + n)
        plt.imshow(x_test_masked[i])
        plt.title("Masked")
        plt.axis("off")

        # Inpainted
        ax = plt.subplot(3, n, i + 1 + 2 * n)
        plt.imshow(predicted_images[i])
        plt.title("Inpainted")
        plt.axis("off")
    plt.show()
```

FIGURE 8. Used code, from: https://www.kaggle.com/code/endofnight17j03/image-inpainting-autoencoder-gen-ai

P2-2 Denoising.

The code of the previous problem was used, and some changes were added to have a noisy image instead of a mask, as same as the previous one the first 10 images of the data set are used. the result can be seen in the figure - 9

as we can see, the model didn´t work well enough for the noise, making some digits seem more like a 3 or 8, I suppose the reason for it is that maybe the data set has more of these digits since in the previous problem and this one 8 was a strong prediction for the AE

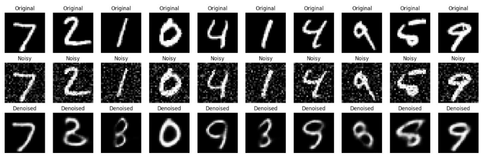


FIGURE 9. Minst noise.

P2-3 Anomaly detection.

I used fashion minst as input for this point, I wanted to maintain the greyscale of the data set trying to trick the AE also I used the figure - 10 and the output was the figure - 11

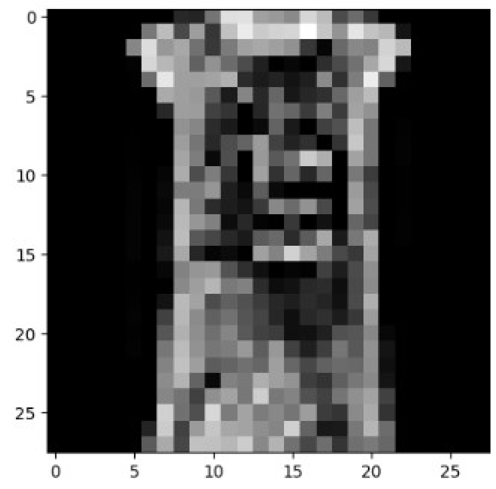


FIGURE 10. Minst fashion used.

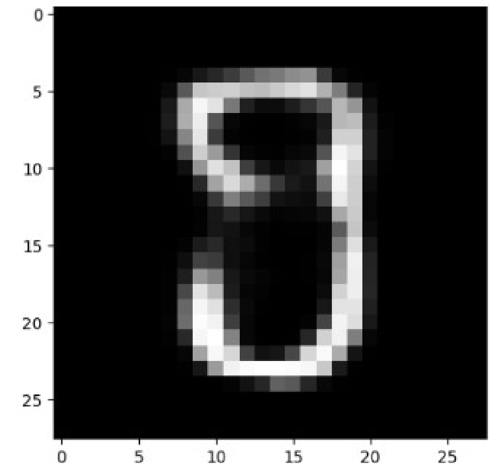


FIGURE 11. AE output from minst fashion.

The auto encoder predicted a 9-8 from a strange input meaning that the input isn't part of the original data.

P2-4 Generation

For the generation, I used an array of 10 (size of the latent space), i had to make sure to try to use the cluster (figure - 12) since the space outside this space seems to be noise, the array selected is [6, 1, -2, 4, -2, 0.3, 0, 2, 0.2, 3].

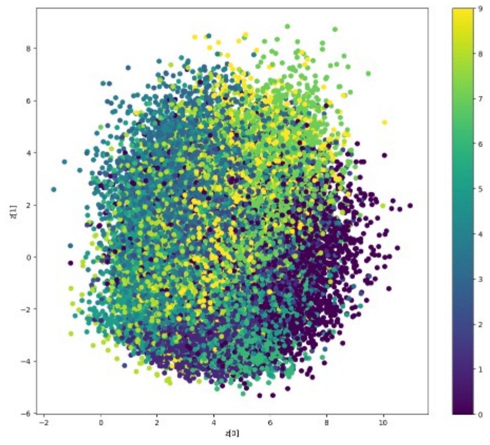
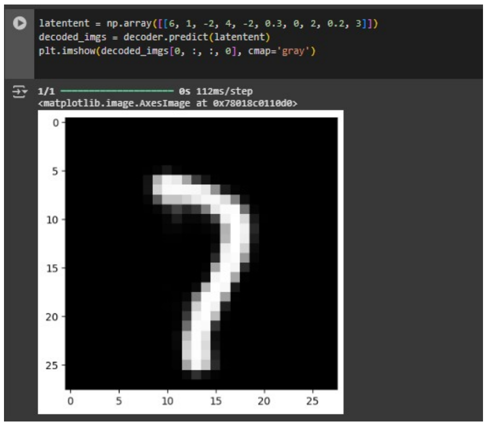


Figure 12. data cluster.

I used the command decoder. predict to run the generation, this process gave me an image that can be seen in the figure - 13



```
latentent = np.array([[6, 1, -2, 4, -2, 0.3, 0, 2, 0.2, 3]])
decoded_imgs = decoder.predict(latentent)
plt.imshow(decoded_imgs[0, :, :, 0], cmap='gray')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step
<matplotlib.image.AxesImage at 0x78018c0110d0>
```

Figure 13. Decoder generation.

The image can be perfected knowing the exact limits of each latent space.

2. Additional Discussion Questions

- For any given application, how might you go about determining an appropriate latent space size for a given AE model?

  The latent space of each solution depends on the complexity of the data, but for me going for the number of labels was a good start, going from there and trying and error to have a good model, trying the number of labels times channels, and other combinations.
- Your tests for Problem 2 focused on samples from the test set. Would performance change significantly if you took samples from the training set? Why or why not?

No, it would be around the same since I used the minst data set, which is already split in test and train , and there is no big variety of outputs, only 10, but if a different data set is used that is less generalized, it is not randomly shuffled, or isn't big enogh, it can affect the performance.

- Can residual blocks be included in an Autoencoder network? Why or why not?

  Yes they can, it can help the model to mitigate the problem of the vanishing gradient, and this can be useful for big neural networks, also reading the class book it states that using it can help to use larger rates and improve performance[1].

References

[1] Understanding Deep Learning (2024) *The TEX Book*, Simon J.D. Prince.