Autoencoder as Classifier on Cifar-10 Dataset

Islam Raihan ull

Contents:

01.Introduction

This report is written to explain the assignment that has been given from Ridge-i company. The topic of this assignment is to use autoencoders to classify images from Cifar-10 dataset using supervised and unsupervised learning. The basic target is to take the features an Autoencoder learns to encode an image and use those extracted features as classifiers. In theory, since the task of an Autoencoder is to generate input data as accurately as possible, it should learn really good features. And from that, classification should be good.

In this report two basic types of autoencoders will be shown. The data will be fed to the autoencoders in another two forms, normal input and noisy input. In essence, I will try:

1. Standard autoencoder with standard input data.
2. Standard autoencoder with noisy input data.
3. Autoencoder with skip connection on standard input data.
4. Autoencoder with skip connection on noisy input data.

I will try to compare results from each case and try to come up with a conclusion of which approach is more suitable in terms of performance.

02.Task Description

Design a network that combines supervised and unsupervised architectures in one model to achieve a classification task. The model must start with autoencoder(s) (stacking autoencoder is ok) that is connected through its hidden layer to another network of your choice, as shown in the figure below. This autoencoder takes an input (image) at node 1 and reconstructs it at its output at node 3. It creates valuable features at its hidden layers (node 2) during this process. it is hypothesized that if node 2 is used as input for the CNN (node 4) then the classification can be improved.

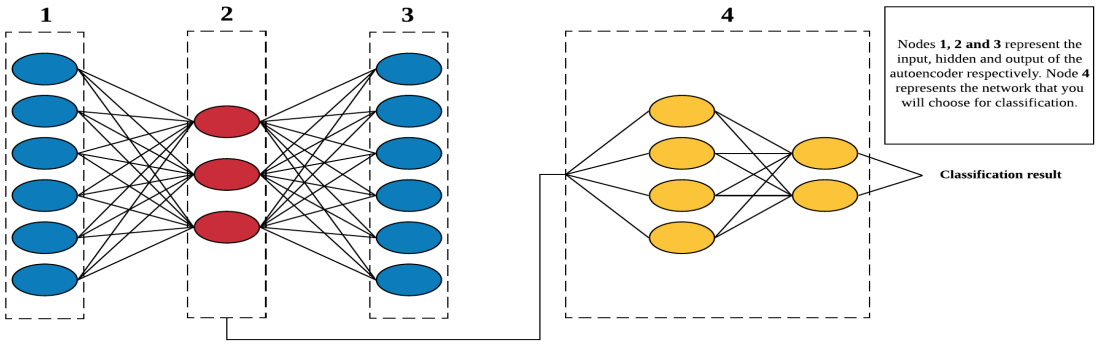


Fig1. Neural network architecture explained in the assignment file

03.Dataset Description

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. Here are the classes in the dataset, as well as 10 random images from each:

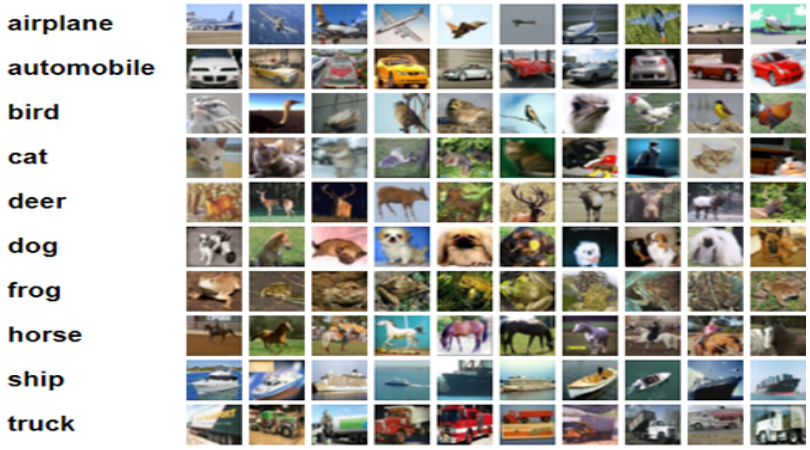


Fig2. Cifar-10 dataset classes

According to the instruction from the challenge, the model will be used to classify Cifar-10 dataset by using at max 2500 training images for each of the bird, deer, and truck classes while using 5000 for the other classes. Delivering a working model with less than 2500 training images for the three referenced classes shows your skills. The model should be evaluated by the test set of 10000 images (1000 for each class).

04.Introduction to Autoencoders

Autoencoder is a nonlinear feature extraction technique that uses dimensionality reduction to reduce the computation time. It is trained in a way that sets the target values to be equal to the inputs. We can take the advantage of convolutional neural networks to exploit local connectivity without training the network on full size images. In this context, convolutional autoencoders are proposed as unsupervised feature extractors to learn features and discover good CNN initializations from higher resolution images. In convolutional networks, I use pooling layers that combine the outputs of neuron clusters. This operation can reduce the resolution of the feature map to achieve

spatial invariance. The weights learned in this first step can then be used as a starting point to initialize a neural network's convolution layers.

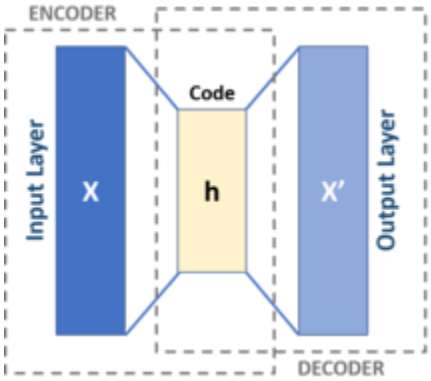


Fig3. General Autoencoder architecture

The target, as mentioned previously, is to use the autoencoder as a classifier. So, in principle: the general architecture of the full model will look like this:

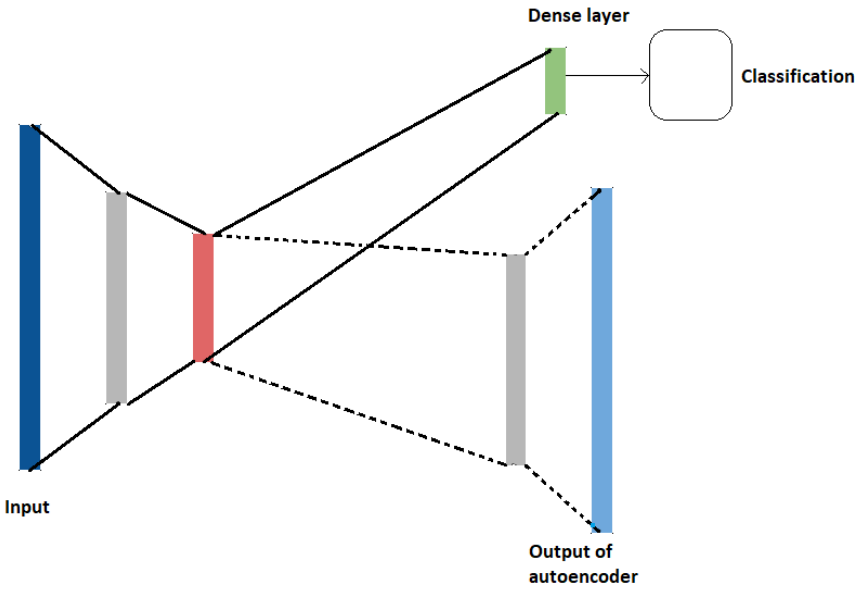
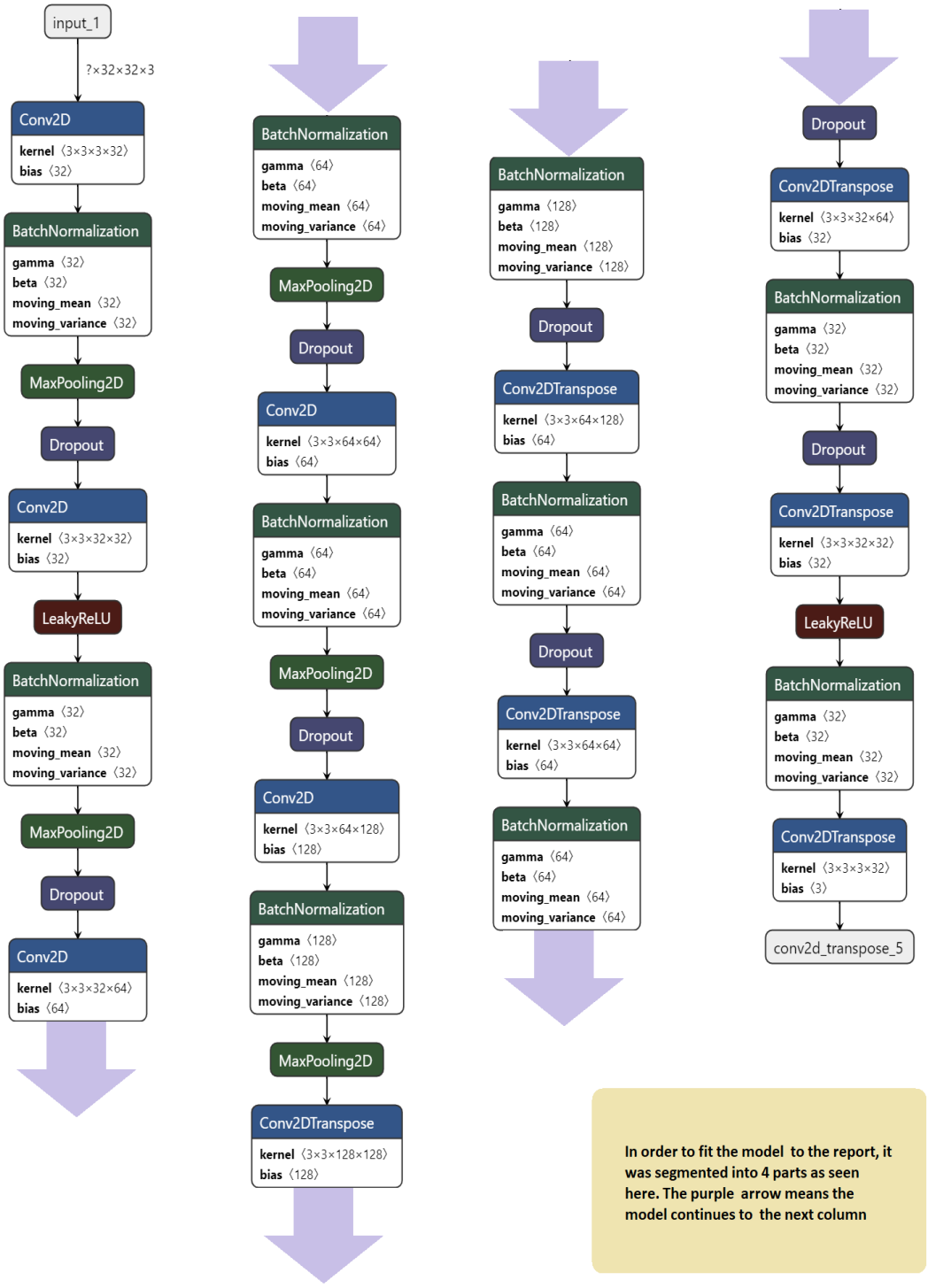


Fig4. General architecture of my approach

05.Task approaches

5.1 Normal approach

At first I designed an autoencoder that I will train. Here you can see the full architecture of my autoencoder model:



In order to fit the model to the report, it was segmented into 4 parts as seen here. The purple arrow means the model continues to the next column

Fig5. Full autoencoder architecture

After training the autoencoder, I built a classifier. I added a few dense layers before flattening the output. I copy the weight from the encoder for this. I made the encoder part trainable as False since if we train again, it will essentially work as a convolutional neural network and nullify our task. The classifier looks like this:

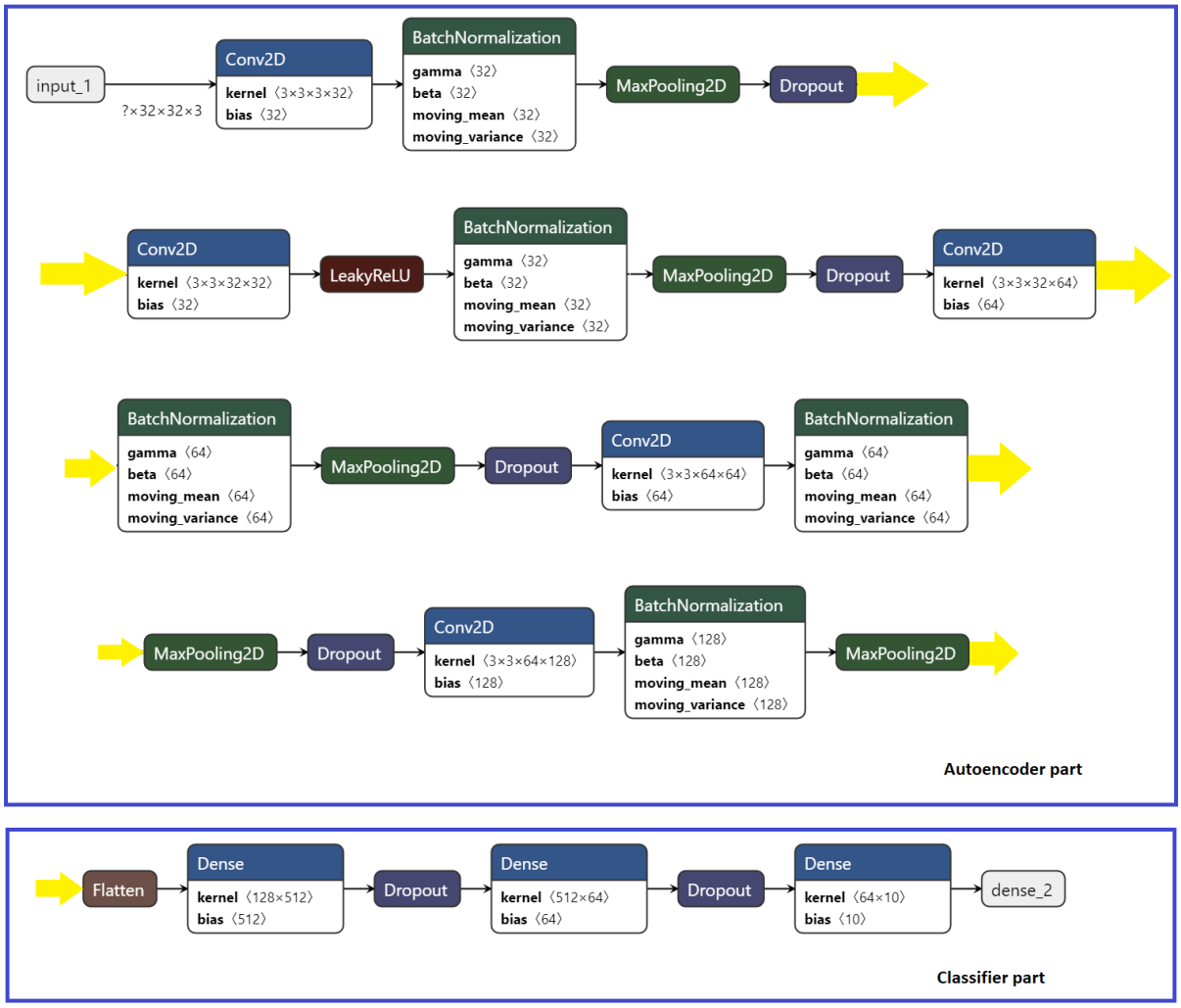


Fig6. Autoencoder with classifier architecture

5.2 Autoencoder with Skip connection approach

For this approach, I added a skip connection. I read in a blog post that adding skip connection may make the model perform better. There is an argument however, if you add a skip connection from encoder to decoder, that is essentially considered

cheating as you are directly feeding from one part to another. In order to avoid that, I add the skip connection way before the bottleneck, as you can see from the picture:

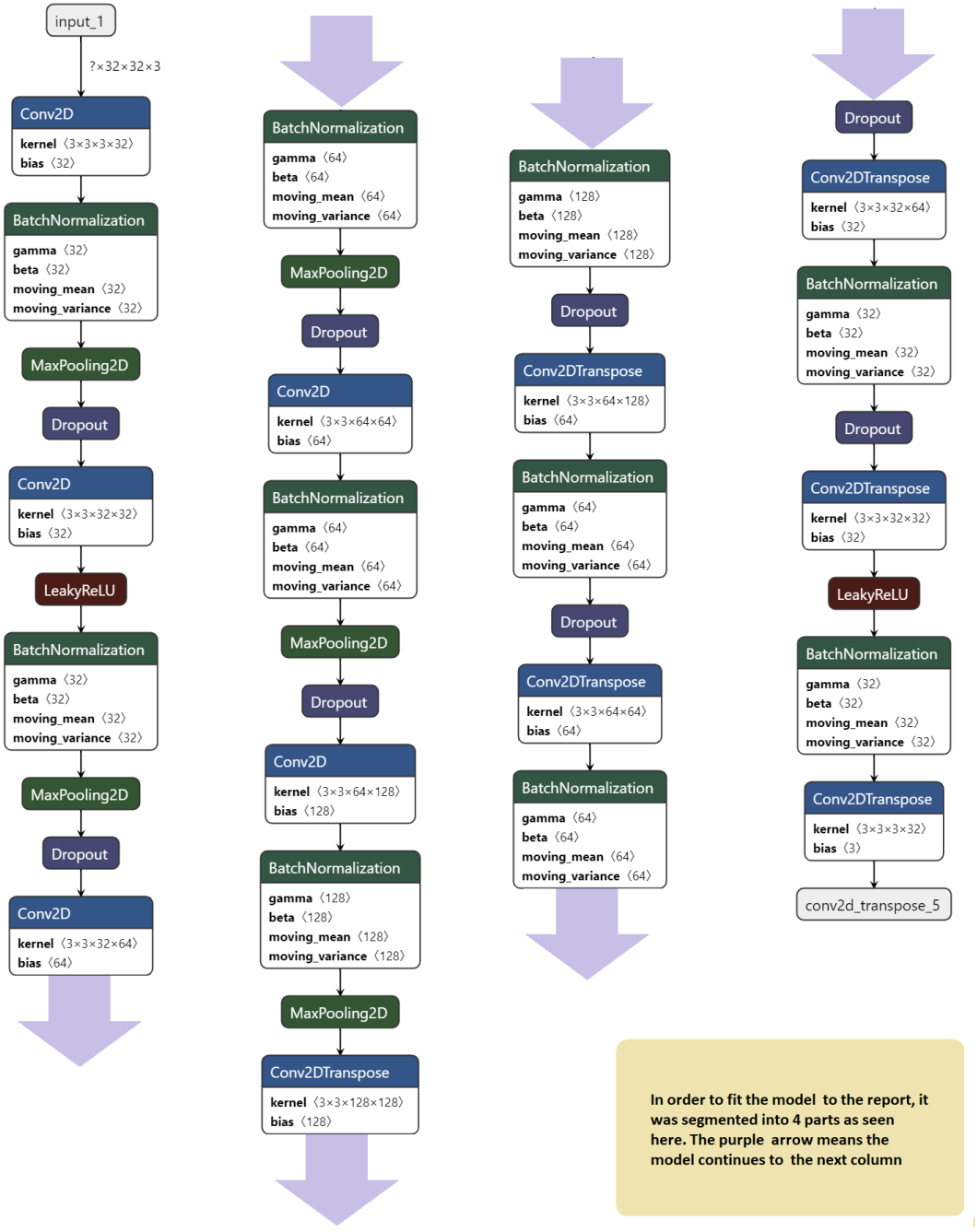


Fig7. Autoencoder with skip connection

5.3 Augmentation of input images

This was fairly basic. I added some basic augmentations of images using Keras's image data generator to increase training samples.

5.4 Adding noise to input data

During my first interview with Ridge-i, we had a really fun conversation of how to make an autoencoder perform better. At that time, my understanding of autoencoders were very basic. During that time, I understood that, if we add a little bit of noise to input images, and the autoencoder can still generate good images from that, the feature it learns, will essentially be better. Because the autoencoder is learning features from less than ideal input data. So I added a bit of noise to the input image and tried both autoencoder approaches as well.

06.Result

I have used the following hyperparameters for each of the attempts:

Learning rate for autoencoder = 0.0001
Learning rate for classifier = 0.001

Epochs for autoencoder = 150
Epochs for classifier = 50

Loss for autoencoder = mse
Loss for classifier = categorical_crossentropy

Optimizer for autoencoder = RMSproop
Optimizer for classifier = Adam.

I found these parameters to give me best results after some trial and error and running Kerasttuner hyperparameter tuner.

Let's look at a comparative table of precision,recall and F1-score of the different approaches that I have tried:

1st approach:

- Skip Connection: False
- Augmentation: False
- Noisy data: False

2nd approach:

- Skip Connection: False
- Augmentation: True
- Noisy data: False

3rd approach:

- Skip Connection: False
- Augmentation: True
- Noisy data: True

4th approach:

- Skip Connection: True
- Augmentation: False
- Noisy data: True

Class	Precision				Recall				F1-Score				Data set
	1st	2nd	3rd	4th	1st	2nd	3rd	4th	1st	2nd	3rd	4th	
class0	40%	45%	54%	55%	77%	85%	91%	88%	52%	59%	68%	68%	1200
class1	90%	89%	89%	89%	82%	88%	90%	88%	86%	89%	89%	89%	1200
class2	83%	87%	79%	75%	38%	45%	58%	63%	52%	60%	67%	68%	3000
class3	69%	71%	66%	64%	28%	43%	58%	54%	40%	54%	62%	59%	1200
class4	61%	75%	82%	79%	73%	68%	66%	65%	67%	71%	74%	71%	3000
class5	83%	71%	74%	72%	39%	63%	67%	62%	53%	67%	70%	67%	1200
class6	83%	81%	75%	72%	79%	85%	91%	92%	81%	83%	82%	80%	1200
class7	72%	77%	85%	88%	82%	85%	80%	79%	77%	81%	83%	83%	1200
class8	67%	83%	87%	85%	91%	90%	88%	92%	77%	86%	87%	88%	1200
class9	75%	87%	93%	94%	85%	84%	77%	73%	80%	85%	84%	83%	3000
Accuracy	68%	74%	76%	76%									17400

From the table above: we can say that there is no clear cut winner approach. In general, the standard approach performs the worst. However, if we add augmentation/noise/skip connection, we can see performance improving.

Let's look at the confusion matrices for each approach:

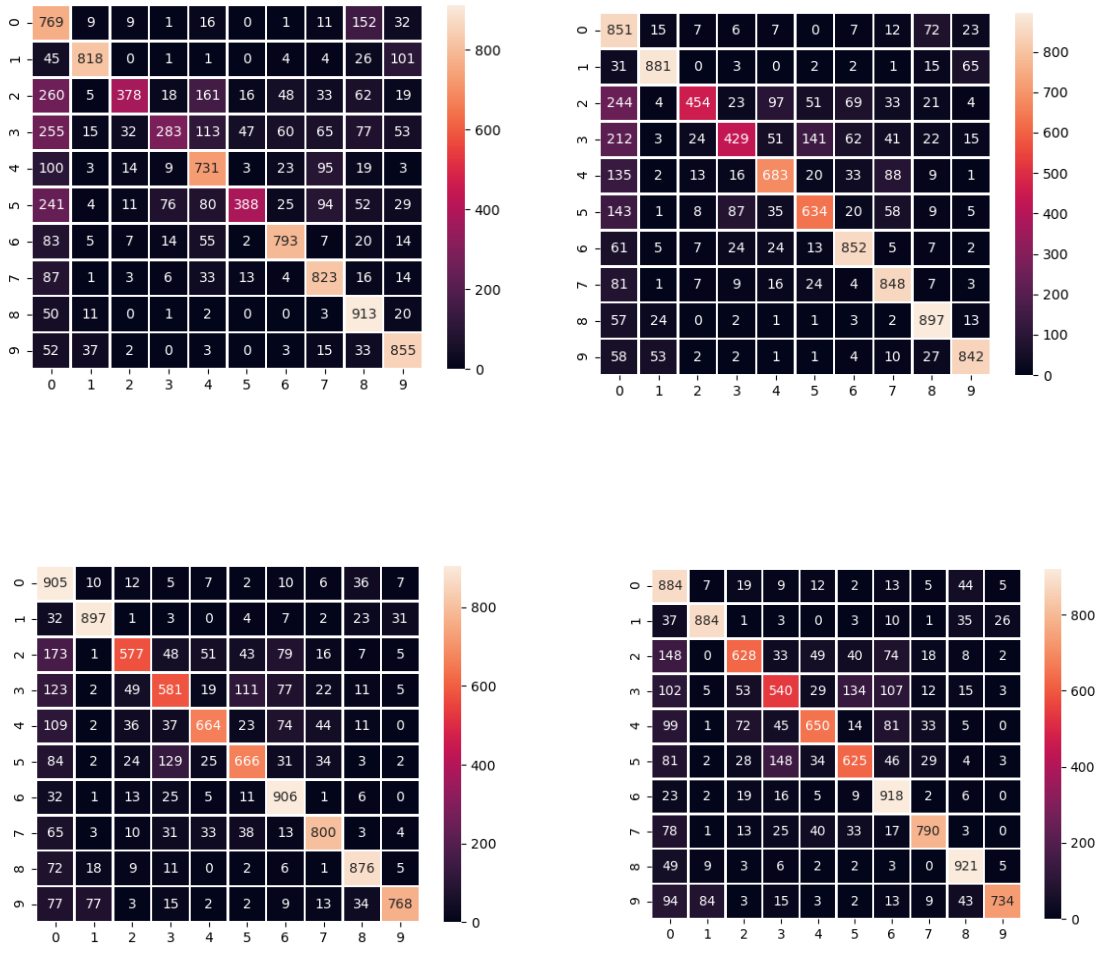


Fig7. From top-left to right

a. Augmentation False, Noise False, Skip connection False
b. Augmentation True, Noise False, Skip connection False
c. Augmentation True, Noise True, Skip connection False
d. Augmentation False, Noise True, Skip connection True

07.Conclusion

This task has been a great learning experience for me. This is the first time I have done in-depth coding for an autoencoder. Not only that, I have learned to use autoencoders as classifiers. The result of the experiment is not satisfactory however. There are plenty of rooms to explore and improve accuracy. Such as:

1. Try different architectures of autoencoder (i.e stacking autoencoder)
2. Try different types of noise, such as salt and pepper noise using OpenCV
3. Try pre-trained autoencoders(If there is any)
4. Try residual network autoencoders.

I also learned that autoencoders can be very effective and efficient means of classification architecture. Compared to traditional convolutional neural networks, autoencoder classifiers can be superior.