Image Recognition using Neural Networks

Parth Bansal

July 21, 2023

1 Overview

The purpose of this assignment was to get familiar with the concept of neural networks and libraries like Pytorch to implement them.
I tried to train different CNN models to classify set of hand-written digits from the MNIST dataset and images from the CIFAR dataset.
I also tried to implement GANs to generate new anime faces from a random input.

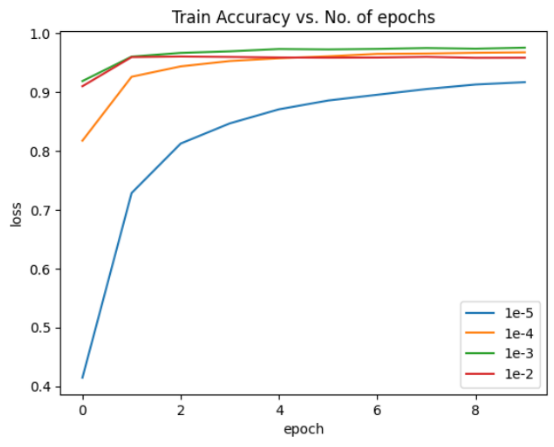
2 Method

- I used PyTorch and standard techniques to create and implement the CNN models with several convolutional layers,pooling layers and batch normalisation.
- I also tried to vary parameters like batchsize, learning-rate and activation functions in the MNIST model to determine the effect of these parameters on accuracy,losses and training time.
- For the CIFAR dataset I used some techniques like normalisation,variable learning rate,weight decay, Randomized data augmentations and used Resnet architecture so that the model can perform better prevent overfitting.
- For the GAN, I followed the standard approach of training the discriminator and the generator alternatively and set targets in such a way that the generator always moves its weights in a way to fool the discriminator.

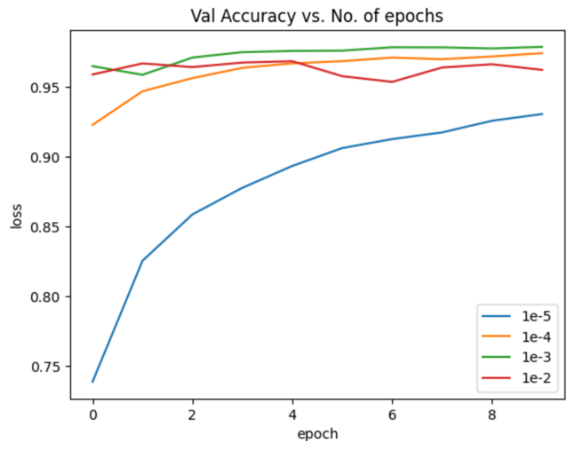
3 Results

3.1 MNIST

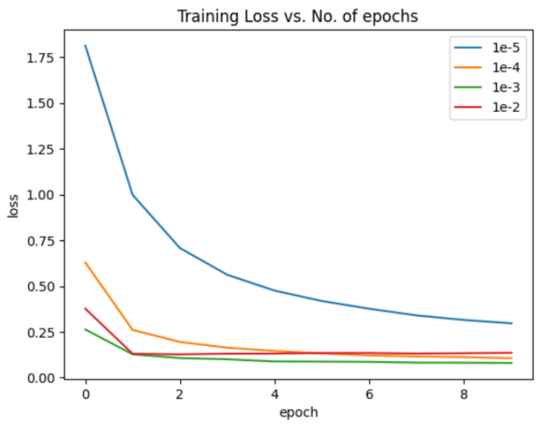
- I was able to achieve an accuracy of 98.86% on the test dataset after training for only 10 epochs.
- Next I experimented with different hyperparameters- batch-size, learning-rate and activation functions in the MNIST model to determine the effect of these parameters on accuracy,losses and training time and I got the following results-



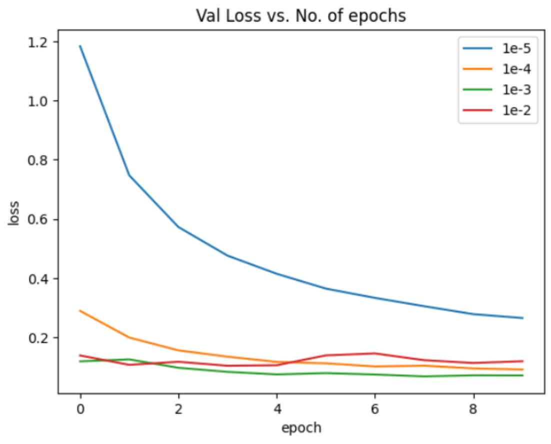
(a)



(b)



(c)

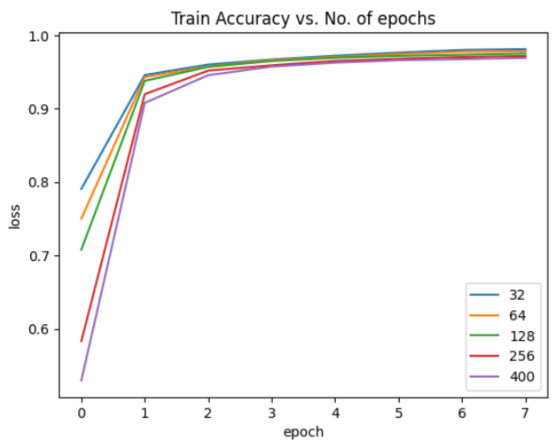


(d)

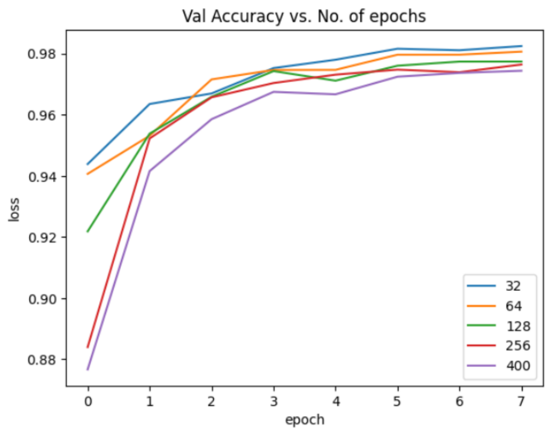
3.1.1 Learning-Rate

- Learning Rate—Training Times-
  - 1e-5=6:48, val accuracy=93%
  - 1e-4=6:45, val accuracy=97%
  - 1e-3=6:22, val accuracy=97.88%
  - 1e-4=6:38, val accuracy=96.24%
- Conclusion-
  - We need to find precise learning rate for fastest training.
  - Very high lr leads to fast training initially but then jumps around the saturation point as seen in accuracy for largest learning rate.
  - Very low lr takes very long to reach the same accuracy
  - One solution is to use scheduler/ warm up which start the learning rate from low value, takes it to the max specified lr and again decreases as we come near end of the training epochs, which allows the benefit of both high and low learning rates at appropriate times

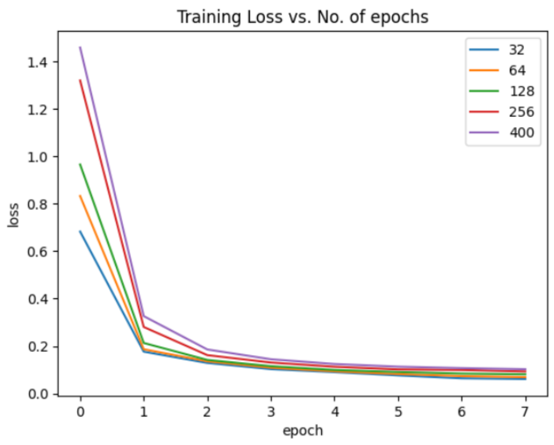
3.1.2 Batch-Size



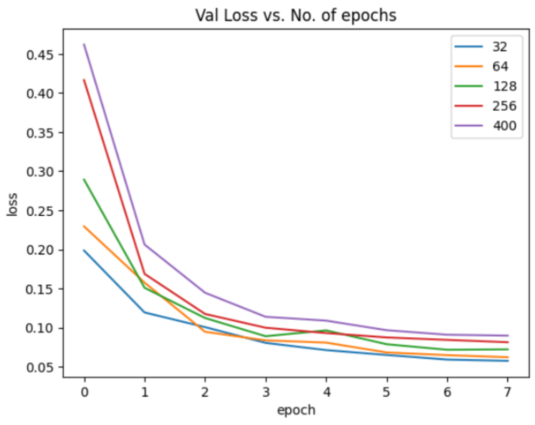
(a)



(b)

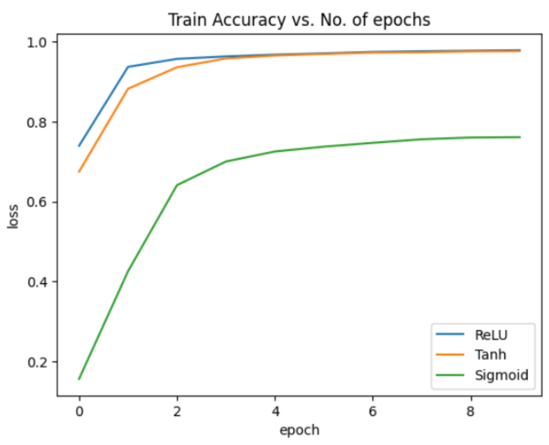


(c)

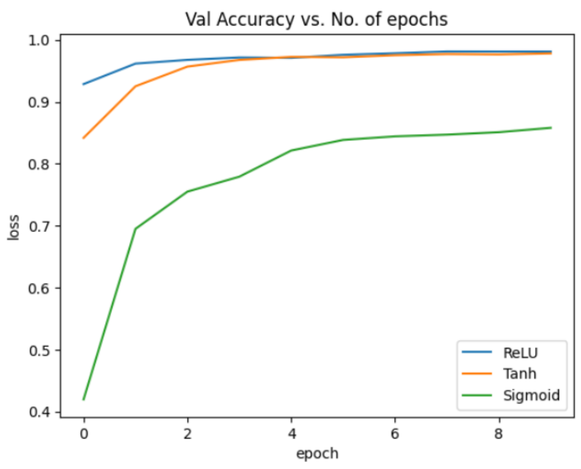


(d)

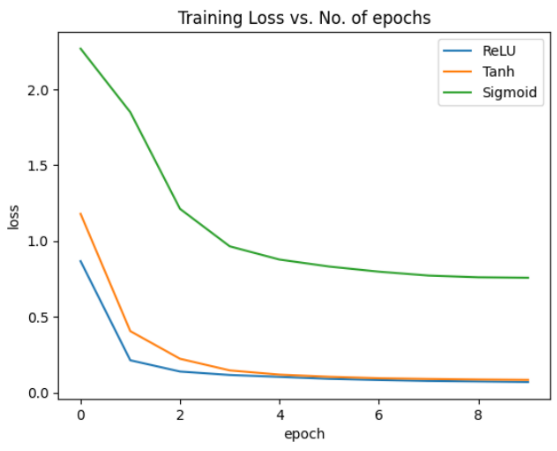
- Batch-Size—Training Times-
  - 32=6:34, val accuracy=98.25%
  - 64=5:58, val accuracy=98%
  - 128=5:53, val accuracy=97.7%
  - 256=6:08, val accuracy=97.64%
  - 400=5:54, val accuracy=97.44%
- Conclusion-
  - Smaller batchsize are more favourable for faster and accurate training, they also require less gpu memory.
  - Smaller batches take longer time to train



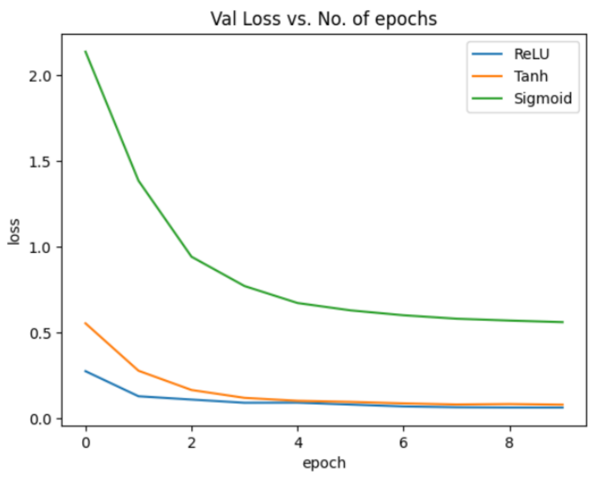
(a)



(b)



(c)



(d)

3.1.3 Activation Function

- Training Times-
  - ReLU=6:46, val accuracy=98.12%
  - Tanh=6:41, val accuracy=97.82%
  - Sigmoid=6:47, val accuracy=85.8%
- Conclusion-
  - sigmoid- not suitable for multi-class classification, actually it is only suitable for binary classification
  - tanh and relu give good results with relu being better due to its simple non-linearity inducing behaviour which leads to lower computations of gradients and non-saturation.

3.2 CIFAR10

- I was able to get above 90% accuracy on test dataset in 30 epochs and 11 minutes using resnet architecture.
- Using data augmentation and normalisation techniques increased the performance of the model, as on using basic cnn model and no normalisation the model only reached about 80% accuracy in 30 epochs.

3.3 GAN

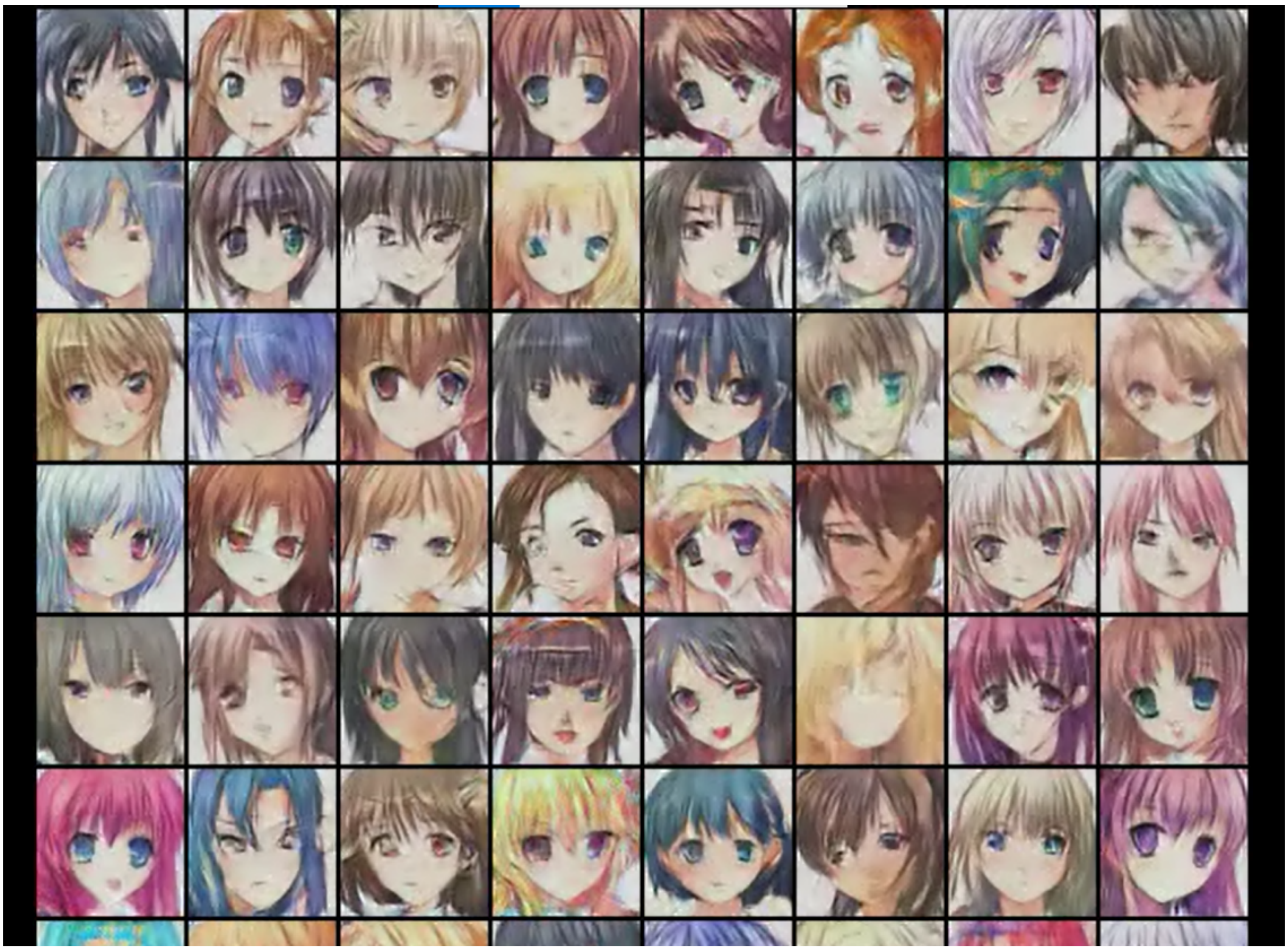


Figure 4

- The training dataset initially had 128*128 pixel images, but I resized them to 64 *64 for faster training and lesser memory for GPU.
- I created anime faces of the size 64*64*3 from random inputs of 128*1 size by training the generator.
- It trained for 25 epochs and gave good enough results.

- It took a lot of time(40 min) to train because it had to train 2 networks, i.e. discriminator and the generator alternatively.
- We can observe some errors like disfigured nose and mouth in some images but overall the generator had become quite good at generating faces, but the discriminator had become 98% accurate on real images and generator could only get a score of about 10% when put against the discriminator, which means it was not able to fool very effectively, so only generator needed to be trained separately to improve its score on the discriminator.
- I was unable to train the generator separately as I couldn't fix the GPU memory allocation error.
- We have a 1-d 128 number vector as input, which can be randomly generated and passed through the generator gives out a random face. These 128 values control different features of the image to be generated. If we are able to identify which index value controls which feature and the range of values, we can generate customized images according to our needs.

4 References

Jovian-https://jovian.com/learn/deep-learning-with-pytorch-zero-to-gans