Badrinath Singhal (23302), Gaurav Talekar (21030)

December 2023

We implemented DDPM in pytorch with animals datset. Image training dataset can be viewed in fig 1

We randomly sampled learning scheduler from either linear or cosine for beta values. The graphs of scheduler can be seen in fig 2

We designed the environemnt with parameters as follows, we took colored images for training DDPM with beta min value to be $1e-4$ and beta max value to be $2e-2$. We used time embeddings of dimension 256 with learning rate as $1e-4$ and no weight decay. We fixed total time steps to be 500 and image dimensions to be 64×64 taking computational constraint into account. For noise addition we sampled from Gaussian distribution $N(0, I)$.

To see the addition of noise at different time step using both linear and cosine beta scheduler please refer fig 3.

We trained the network for 30 epochs and loss curve can be seen in fig 4

After training we generated images with different number of time steps and compared the differences which can be seen in fig 5

We've trained 2 more DDPMs on different time steps with everything else the same and seen that using more time steps leads to better generation quality whereas fewer time steps leads to missing finer details.

Applying DDPM on VAE latent space gives one advantage that it reduces inference time significantly which is one of the major issues surrounding DDPM. The results of generated image using decoder of VAE were of similar quality to that of DDPM but since we can reduce the inference time as well as training time, we can now afford to have better generation quality of images.

We can see the results of DDPM on VAE latent space in fig in **??**. Results on VAE is more blurry whereas the loss function is also shown

To train MOCO we used the same animal dataset with image dimension 128×128 using the same dataloader we used all this while. We used VGG model as a feature extractor with pretrained weights on ImageNet dataset. We created two encoders using the pretrained VGG model, one is query encoder and other is key encoder.

We further declared two classifier, a FC classifier and other a CNN classifier. FC network consist of 2 hidden layer with ReLU non linearity and Dropout where output feature size is 32 whereas CNN is 2 conv layer with ReLU non linearity and final layer as a FC layer. To train MOCO we used following augmentations on dataset

- Random Horizontal Flip
- Random Vertical Flip
- Random Crop
- Random Resized Crop

Figure 1: Samples from Training Data

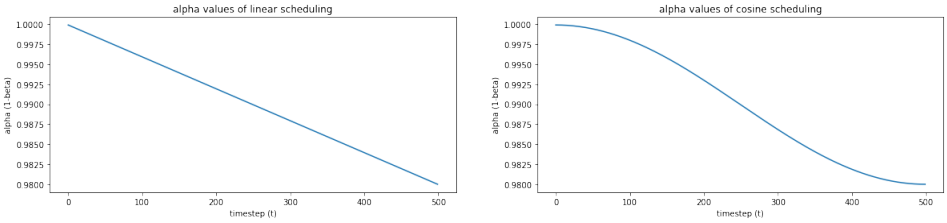
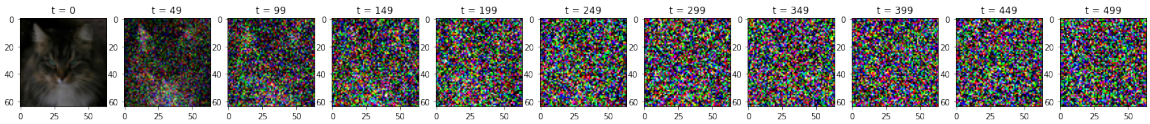
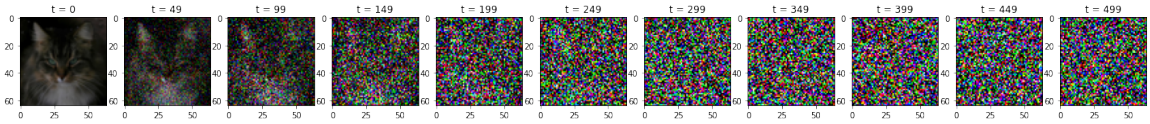


Figure 2: Scheduler



(a) Linear scheduler noise addition



(b) Cosine scheduler noise addition

Figure 3: Linear and cosine noise addition

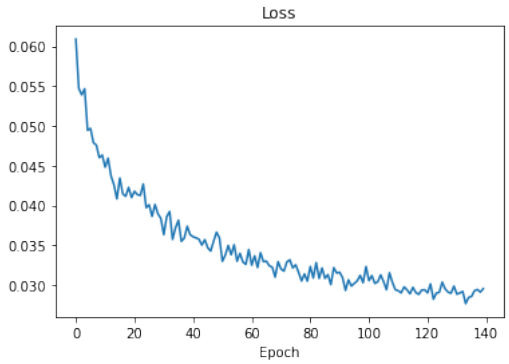


Figure 4: Loss of DDPM

- Gaussian Blur
- Random Auto Contrast

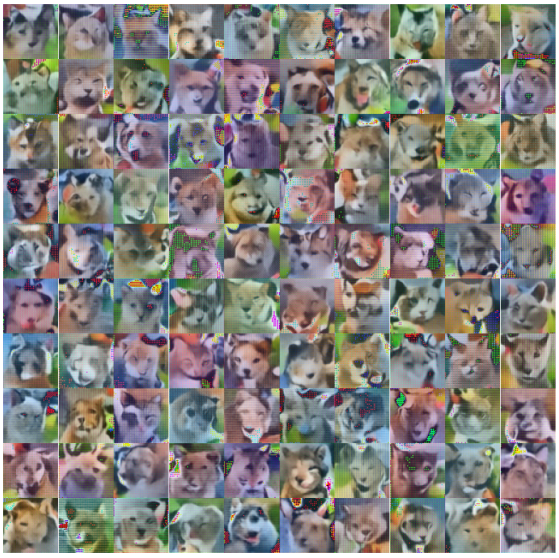
We used Adam optimizer as a choice for updating parameters. For each data we apply randomly one of the augmentation and get the correponding features and apply batch matrix multiplication on them. This act as a positive sample and we apply cross entropy with negative sample and update our model.

After training we tested our model on test dataset and report our results as well as the visualised features in fig 7

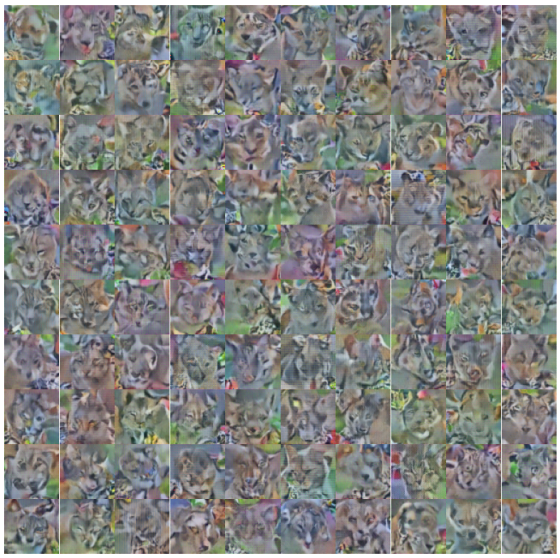
We've performed training multiple times with different data split . We trained on 80/20 split,

- For 80/20 split We've seen that FC network and CNN performs somewhat better where test accuracy for FC network is 51.2% and accuracy for CNN is 60.6%. We've achived that with 25 epochs of training both the networks.
- For 50/50 split we've seen performance to fall for both. FC network accuracy was 37.06% and CNN classifier accuracy was 54.6%
- For 90/10 split we've achieved FC network perfromance to be 41.8% and CNN to be of 33.3%.

(a) Generation with 500/500 time steps



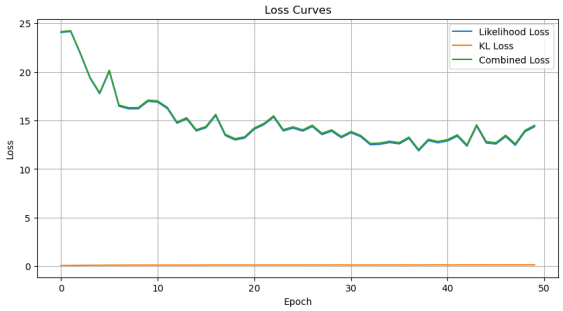
(b) Generation with 300/500 time steps



(c) Generation with 100/500 time steps

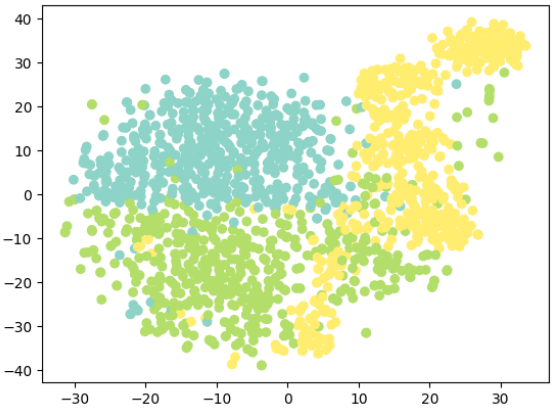
Figure 5: Generation in different time step

(a) DDPM on latent VAE

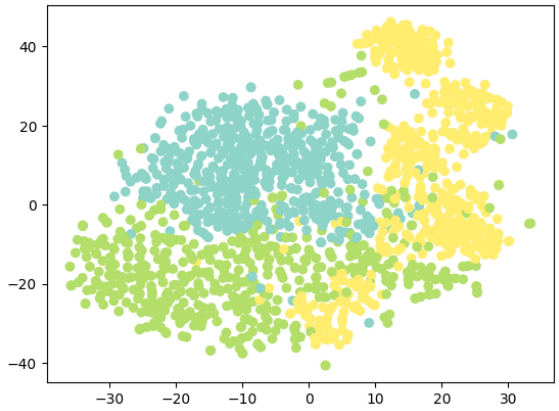


(b) Loss VAE

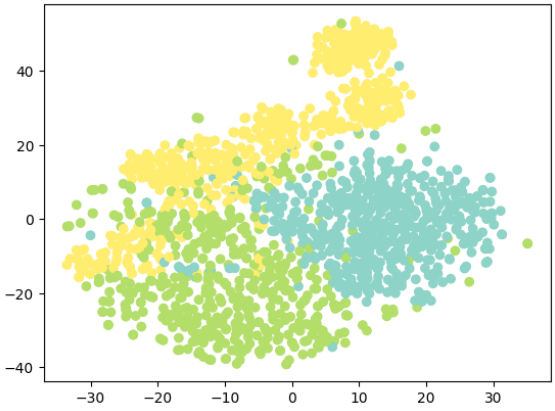
Figure 6: DDPM on VAE



(a) Visualisation for 80/20 split



(b) Visualisation for 50/50 split



(c) Visualisation for 90/10 split

Figure 7: MOCO visualisation on different data split