WESTERN SYDNEY
UNIVERSITY

Project 2024

KRISHNAKANTH KURUVACHIRA SABU
22078053

A Project report submitted for
MATH7017: Probabilistic Graphical Models

June 2024

INTRODUCTION

In this project, we implemented and compared Conditional Variational Autoencoders (C-VAEs) and Conditional Generative Adversarial Networks (C-GANs) for generating Japanese characters from the Kuzushiji-49 dataset. C-VAEs and C-GANs are deep learning models that can generate data conditioned on labels, which in this case are character classes and style_labels which we added thick or thin characters.

DATASET PREPERATION

We begin by loading the Kuzushiji-49 dataset, which contains images of 49 different Japanese Hiragana characters. The dataset is provided in .npz files, which we load into numpy arrays. This dataset is valuable for training models to recognize and generate Japanese characters. Each image in these arrays is a 28x28 array of pixel values ranging from 0 to 255 in grayscale. The dataset consist of training and testing data is stored into variables as train_images, train_labels, test_images , test_labels

Concatenate and normalize data.

For easier processing, we concatenate the training and test datasets into a single dataset, which is stored in imagedata and labeldata. This combined dataset allows us to apply the same preprocessing steps to all images and labels without needing to differentiate between training and test sets. Normalizing the image data to a range of 0 to 1, which is crucial for better performance of the neural networks. For the performance we made the imagedata type into float and divide it by 255. Normalization helps in stabilizing and speeding up the training process by ensuring that the input data has a consistent scale.

Creating Style labels and counting foreground pixels.

For creating the style_labels (thick or thin) to each image, we need to count the number of foreground pixels which is non-zero pixels and compare it to the median number of foreground pixels for each character class. We define a function called count_foreground_pixels to count the number of non-zero pixels in each image. The function converts the image into a binary image where pixels greater than the threshold are set to True foreground, and then it sums these True values to get the pixel count.

Calculate the median foreground pixels.

For each character class, we compute the median number of foreground pixels across all its images. The steps we used to compute:

1. Get all unique labels from 0 to 48.
2. For each label we extract the corresponding images.
3. Now count the foreground pixels for each image.
4. Compute the median of these counts for the class.

Assign the style label.

We assign a style label 0 for thin and 1 for thick to each image based on its foreground pixel count relative to the median for its class. We create an array named style_labels to store these images. We initialized an array of zeros for style labels, for each image need to count the foreground pixels. Now Compare this count to the median for the class and assign the style label based on whether the count is above or below the median.

Implementation of One-Hot Encoding

we use one-hot encoding to transform the class labels into a binary matrix representation from the python library torch and the function eye to implement the encoding.

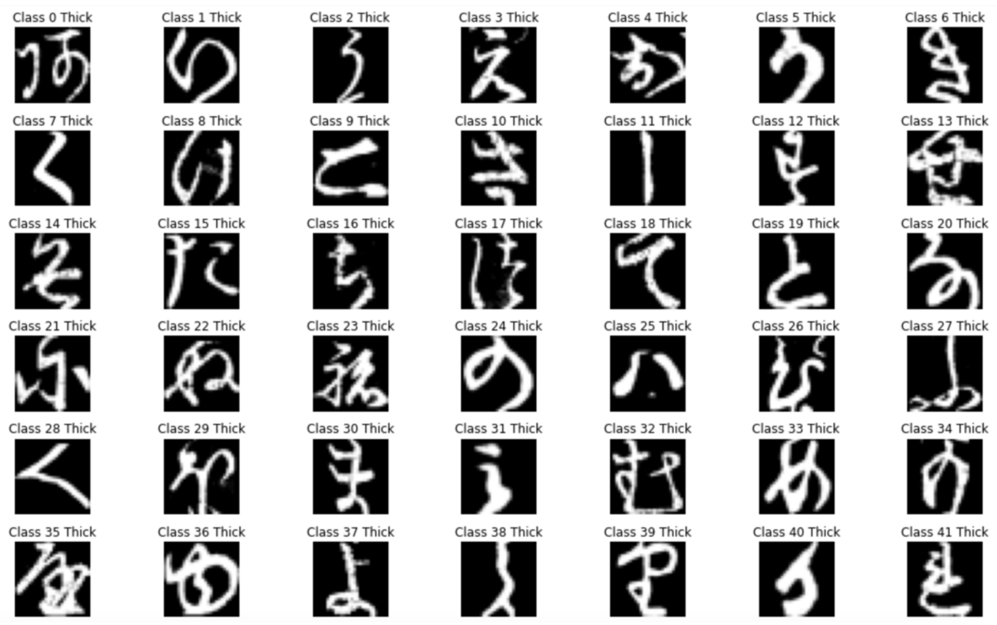
Reshaping

we reshape the image data from its original 3D shape to a 2D shape to make it compatible with the input requirements of the neural network layers. This reshaping flattens each 28x28 image into a 784-dimensional vector. We assigned -1 because, -1 automatically infers the size of the second dimension based on the original shape, resulting in a vector of size 784 (28 * 28) where original size is shape.[0] which represent the number of samples.

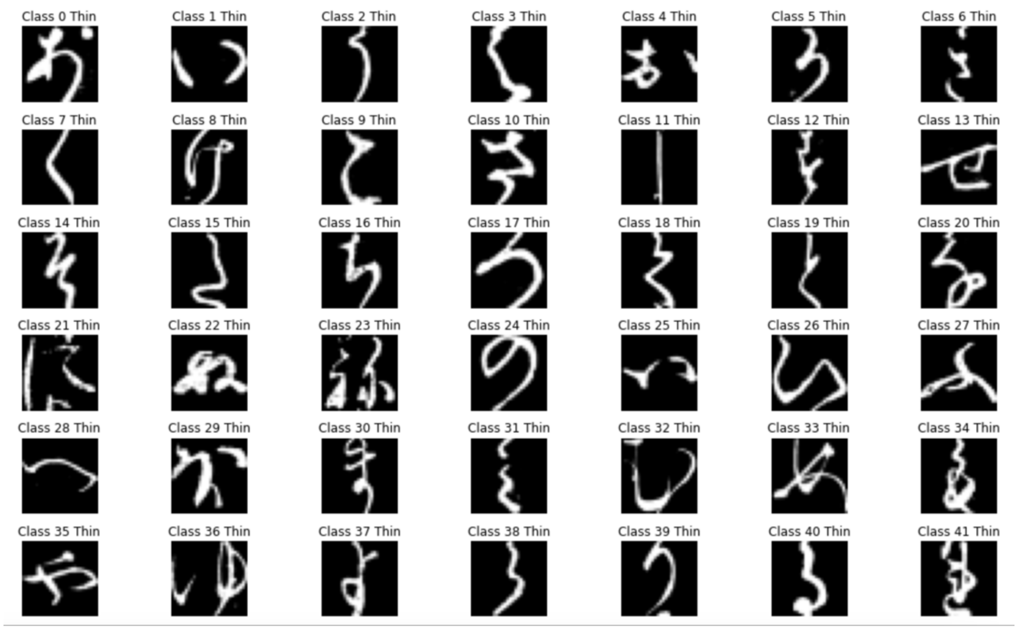
Plotting Processed images:

To visualize the images and their corresponding labels, we created a function called plot_image that plots them. This function helps in verifying the data preparation steps and understanding how well the model is generating images during training. Visualizing the images and labels helps ensure that the data has been correctly loaded, normalized, and labelled. We use three parameters to create this function, images, labels, and style_labels. The images below shows the processed images after pre-processing.

This are images having the styles = thick.



This are images having the styles = thin.



CONDITIONED VARIATIONAL AUTOENCODER (C-VAE)

This model is particularly useful for tasks where additional context or labels can guide the generation, such as in recommendation systems, text-to-speech, and image generation [1]. Here, the application of C-VAE to the Kuzushiji-49 dataset, showing its effectiveness in generating high-quality images conditioned on class and style labels.

Model Architecture:

- Extends the VAE by conditioning the generation process on additional information,
- The encoder takes the input data and conditioning variables, mapping them to a latent space.
- The decoder reconstructs the input data from the latent space, conditioned on the same variables.

Encoder Network:

The encoder network maps the input data and conditioning variables to a latent space.

- fc1: Takes the concatenated input and label vectors and maps them to a hidden representation of dimension hidden_dim.
- fc2_mu: Outputs the mean of the latent space distribution.
- fc2_logvar: Outputs the log-variance of the latent space distribution.

Decoder Network:

The decoder network reconstructs the input data from the latent representation, conditioned on the same variables.

- fc3: layer that takes the concatenated latent vector and label vector and maps them to a hidden representation of dimension
- fc4: layer that reconstructs the input data from the hidden representation.

Model Definition:

The CVAE class defines the architecture of the Conditional Variational Autoencoder. It includes an encoder, a reparameterization, and a decoder.

Loss, Training and generation Function:

1. The loss function combines Binary Cross Entropy and KLD to train the VAE [1].
2. The training function trains the CVAE model for one epoch.
3. The generation function generates new samples from the trained model given a label.

Now we setup a directory for saving all the images we generate, then training loop runs for a specified number of epochs, trains the model, generates, and saves samples at the end of each epoch. After the model run, we can check the average loss of each epoch and ensure getting a better model. Here I have run 100 epochs, The 100th epoch image is fig 3.

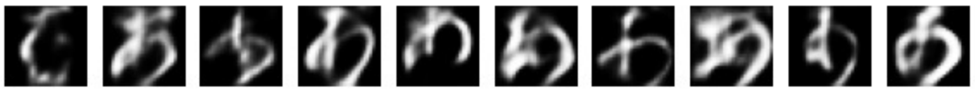


Fig 4, 100th epoch.

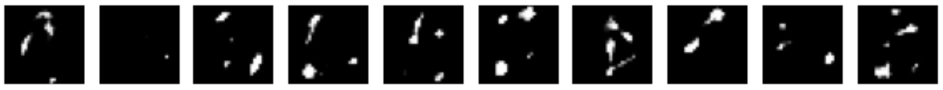


Fig 5, 6th epoch

While training the model's performance is satisfied, now we generate and try to plot the sample by initializing the label and number of samples in the generate function.
Lets generate new sample with label = 6, num samples as 10.

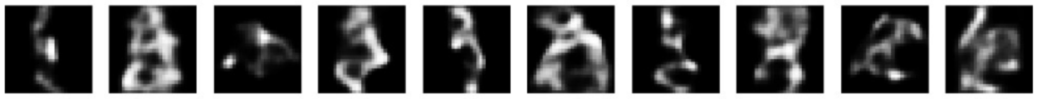
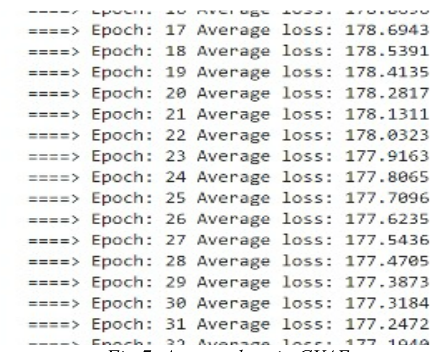


Fig 6, label = 6, num_samples = 10

Within the generate function we call the function to save the figure image after generating, as a result whenever new images are generated it is stored in the directory we are working.

Average loss in training loop:



```
====> Epoch: 17 Average loss: 178.6943
====> Epoch: 18 Average loss: 178.5391
====> Epoch: 19 Average loss: 178.4135
====> Epoch: 20 Average loss: 178.2817
====> Epoch: 21 Average loss: 178.1311
====> Epoch: 22 Average loss: 178.0323
====> Epoch: 23 Average loss: 177.9163
====> Epoch: 24 Average loss: 177.8065
====> Epoch: 25 Average loss: 177.7096
====> Epoch: 26 Average loss: 177.6235
====> Epoch: 27 Average loss: 177.5436
====> Epoch: 28 Average loss: 177.4705
====> Epoch: 29 Average loss: 177.3873
====> Epoch: 30 Average loss: 177.3184
====> Epoch: 31 Average loss: 177.2472
```

Fig 7. Average loss in CVAE

Evaluate the learning loss Curve:

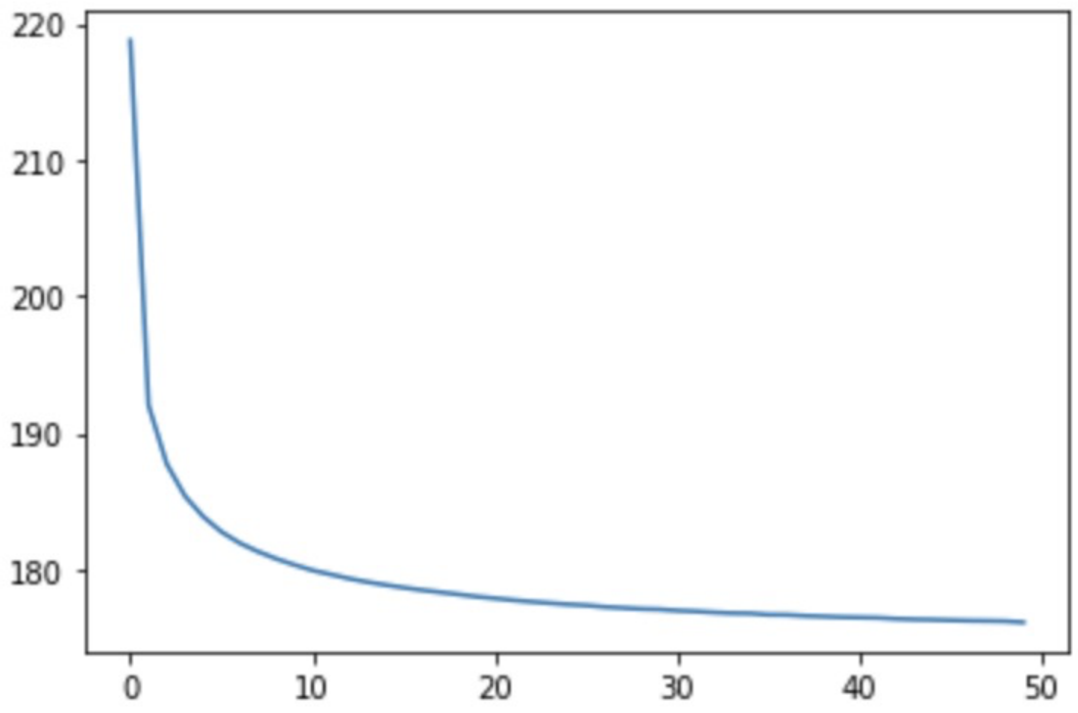


Fig 8, Loss curve

Train Loss curve shows its coming down and moving towards the end of epoch which is a good sign.

CONDITIONAL GENERATIVE ADVERSARIAL NETWORK (C-GAN)

Conditional GANs are an extension of GANs where the generation process is conditioned on additional information such as class labels or other types of auxiliary information. This approach has been successfully applied in various tasks, including image generation, where it helps to generate more meaningful and controllable outputs.

Model Architecture

The C-GAN consists of two neural networks:

- Generator: Takes random noise vector and class and style labels as inputs to produce a specified Japanese character.
- Discriminator: Takes image and class and style labels as inputs and determines whether the image is real or fake.

Generator and Discriminator network:

The Generator network takes a noise vector and class and style labels as input and produces a fake image that attempts to mimic real data from the dataset. Where, the Discriminator network takes an image and class and style labels as input and determines whether the image is real or fake.

Training C-GAN.

The training loop for the C-GAN involves alternating between training the Discriminator and the Generator. The Discriminator learns to distinguish real images from fake ones generated by the Generator, while the Generator learns to produce images that the Discriminator classifies as real.

Initialization:

We initialize the Generator and Discriminator models and move them to the device (CPU or GPU). We set up the optimizers for both models with a learning rate of 0.0002 and betas (0.5, 0.999) for the Adam optimizer. We define the binary cross-entropy loss function (nn.BCELoss()) for adversarial loss. We create a TensorDataset and DataLoader to load the training data in batches. We run epoch (a complete pass over the entire training dataset),
For each batch of real images and their corresponding labels we need it to prepare real and fake labels where real labels are set to 0.9 instead of 1.0 (smoothing) to help stabilize training.
Fake labels are set to 0 after we Train Discriminator by real Images then compute the Discriminator's output on real images and calculate the loss (d_real_loss). Fake Images:
Generate random noise and corresponding labels to produce fake images using the Generator which then compute the discriminator's output on fake images and calculate the loss (d_fake_loss).

Total Discriminator Loss:

The total loss for the Discriminator is the average of d_real_loss and d_fake_loss. Then backpropagate the loss and update the Discriminator's parameters. Train generator which Generates fake images again. Compute the Discriminator's output on these fake images and calculate the loss (g_loss). The goal is to maximize the Discriminator's output for these fake images which is to trick the Discriminator into thinking they are real.Backpropagate the loss and update the Generator's parameters. Print the losses for the Discriminator and Generator at regular intervals to monitor training progress. Save generated images at the end of each epoch to visually inspect the quality of generated images over time.

We train the C-GAN for a 50 times of epochs and monitor the Discriminator and Generator losses to ensure they are improving. The generated images at different epochs are saved in the cgan_images directory.

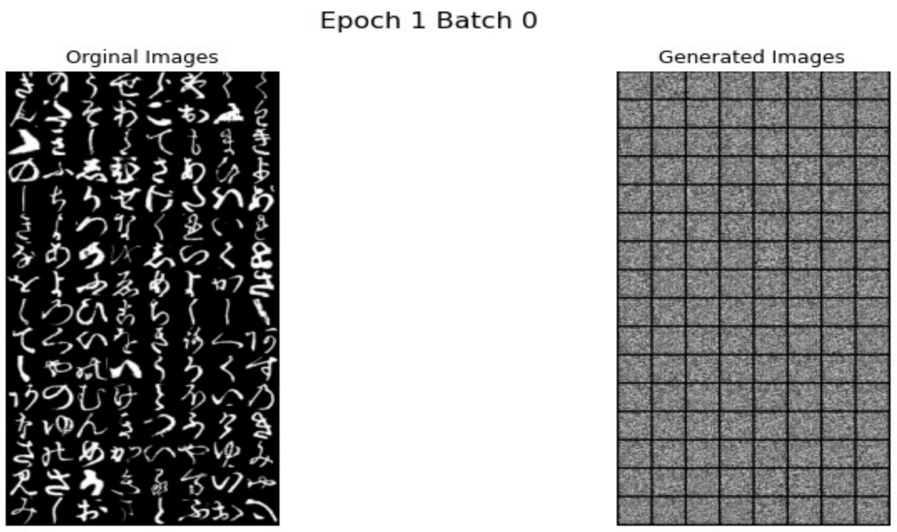


Fig 9. C-GAN training loop, epoch 1 .

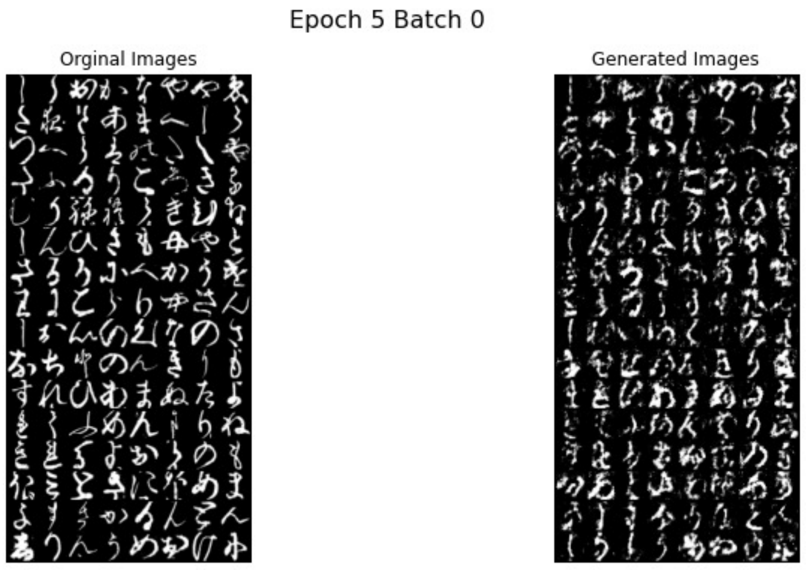


Fig 10. C-GAN training loop, epoch 5 .

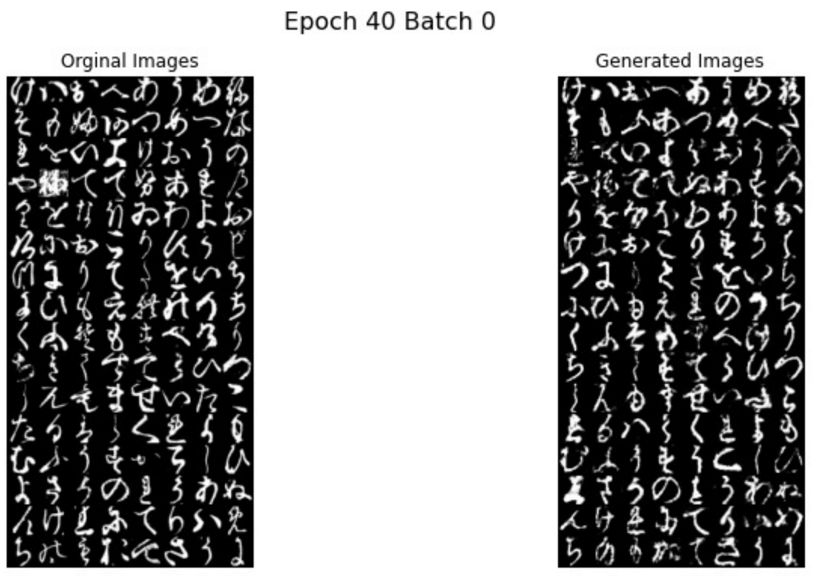


Fig 11.. C-GAN training loop, epoch 40

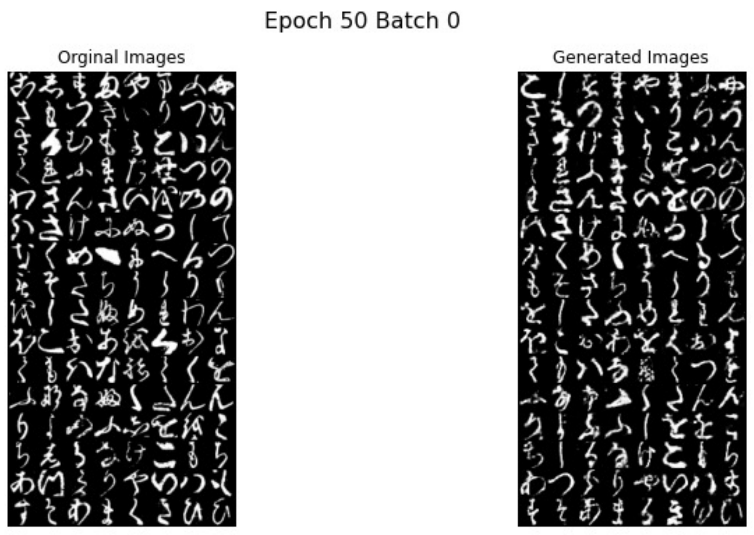


Fig 12.. C-GAN training loop, epoch 50

Learning Curve:

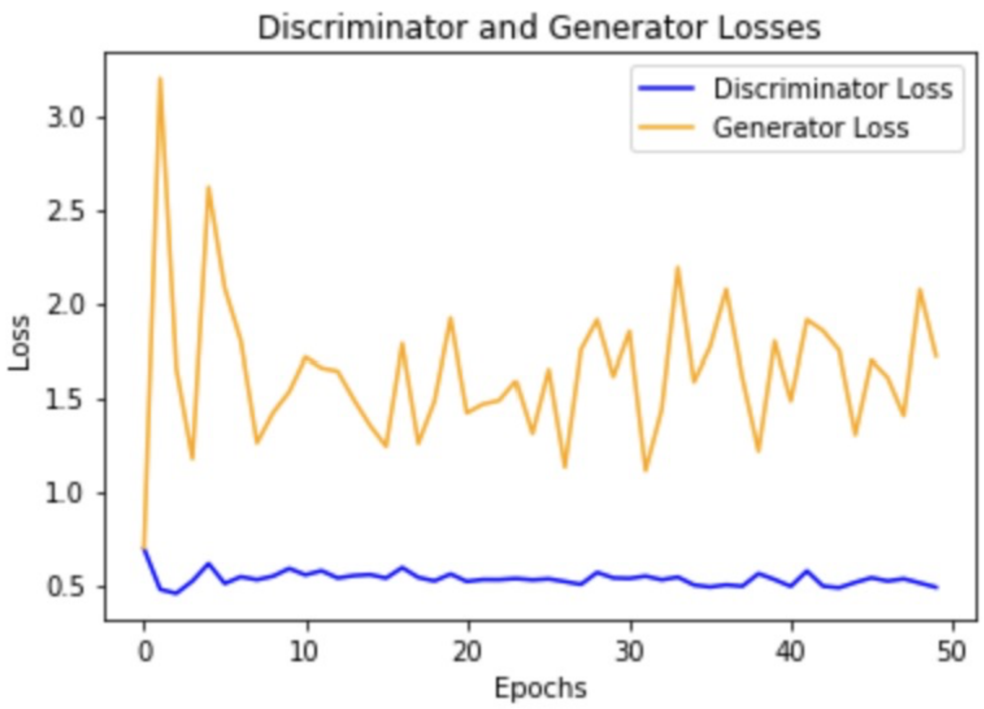


Fig 13.. C-GAN learning curve

Generator shows bit noise all over running the epoch and initially it went high and eventually it went down. Discriminator shows stable which means the testing data isn't affected.

Result and Conclusion :

The Conditional GAN successfully generates Japanese characters conditioned on class and style labels. The generated images improve over epochs, as seen in the saved samples and the displayed samples. This approach can be further refined by tuning hyperparameters and improving the device architecture from CPU to GPU and also run more counts of epochs which will result in better performance.

MODEL COMPARISON

C-VAE:

Architecture:
The C-VAE consists of an encoder, a reparameterization, and a decoder. The encoder maps the input data and conditioning variables to a latent space, the reparameterization step samples from this latent space, and the decoder reconstructs the input data from the latent representation, conditioned on the same variables.

C-GAN:

Architecture:
The C-GAN consists of a generator and a discriminator. The generator takes a random noise vector and class and style labels as inputs to produce a specified Japanese character. The discriminator takes an image and class and style labels as inputs and determines whether the image is real or fake.

Performance Comparison:

C-VAE:

- The C-VAE generates samples by sampling from a latent space that is conditioned on the input labels ,class and style.
- The C-VAE model trained faster compared to the C-GAN. Each epoch took less time because it involves a single forward and backward pass through the encoder and decoder.
- The generated samples tend to be smooth and even resemble the training data well.
- Generally more stable and less sensitive to hyperparameters, which can lead to more consistent training times. Mostly the images generated by C-VAE are generally coherent and clear.
- There might be some blurriness due to the faces of the variational approach, which focuses on minimizing the reconstruction loss.

Learning Curve:

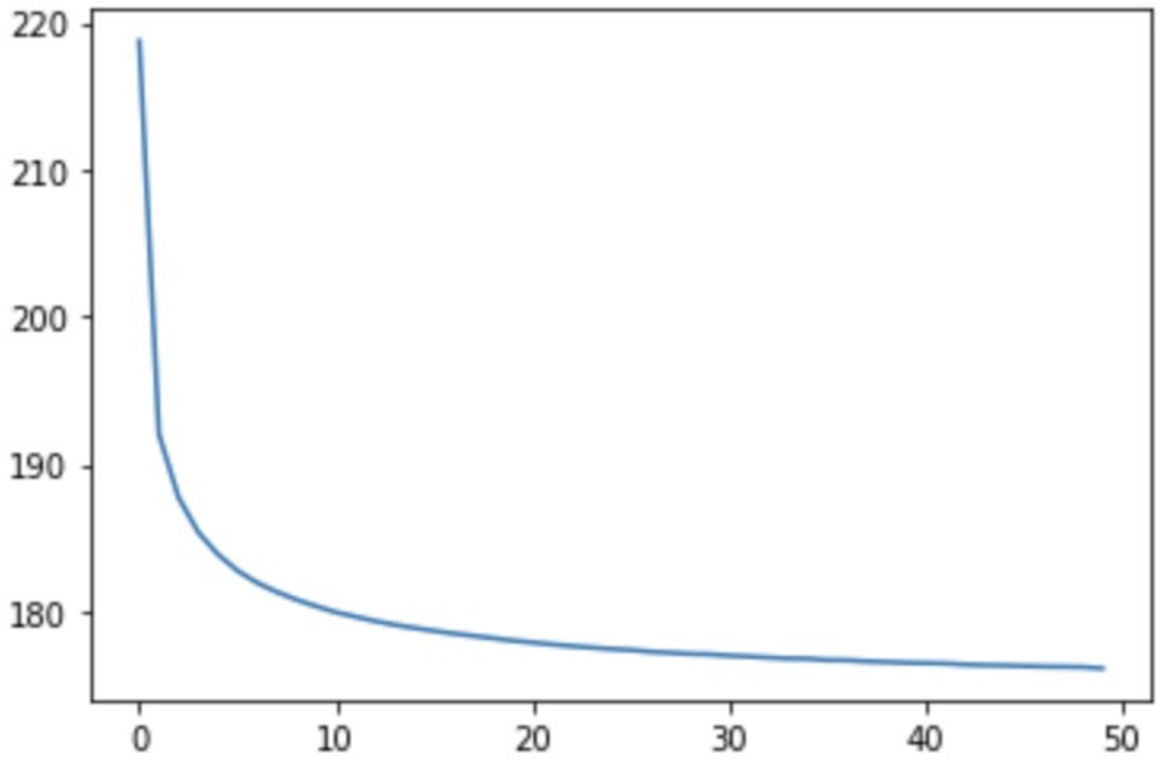


Fig 14.. C-VAN training LOSS

Image Generated:

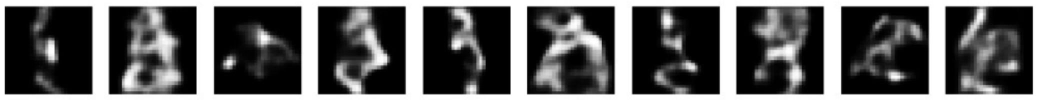


Fig 15, label = 6, num_samples = 10

C-GAN

- The C-GAN generates samples by learning to produce images that the discriminator cannot distinguish from real images.
- The generated samples are conditioned on both class and style labels.
- Involves training two separate models generator and discriminator alternately. Each epoch requires multiple forward and backward passes through both networks.
- Can suffer from instability issues such as mode collapse, requiring careful tuning of hyperparameters and additional techniques to stabilize training.

Learning Curve:

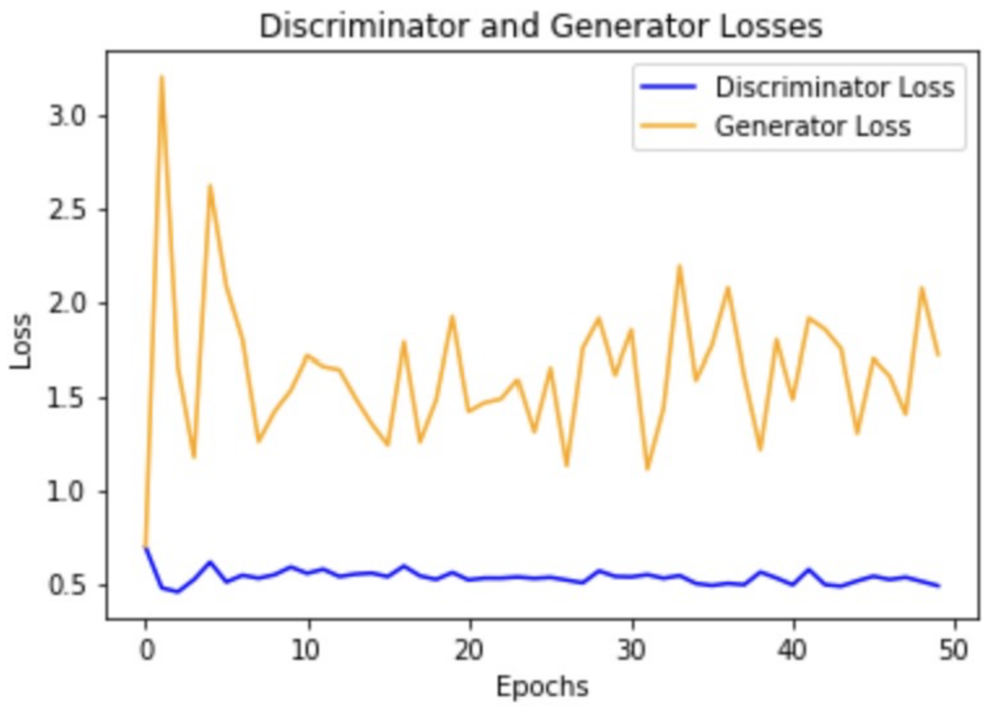


Fig 16.. C-GAN LEARNING CURVE

Image Generated:

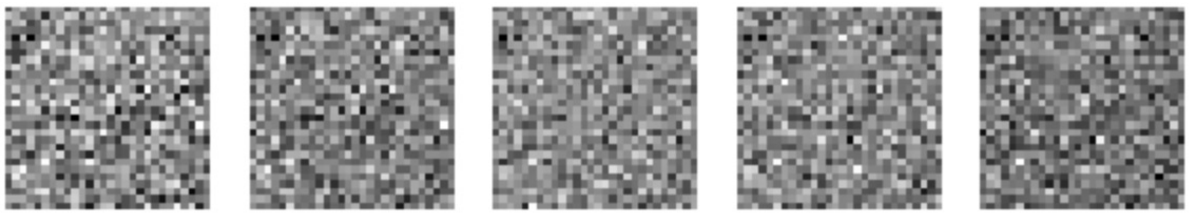


Fig 17.. C-GAN GENERATED IMAGE

Image generated if in GPU WITH more epochs will be better

Result and Conclusion:

C-VAE:

- Strengths: Faster training, stable convergence, and the ability to produce relatively clear images quickly. It is suitable for tasks where training speed and stability are critical.
- Weaknesses: The generated images may appear slightly blurry and lack the fine clearer image details seen in GAN-generated images.

C-GAN:

- Strengths: It generates sharp and highly detailed images. It excels in tasks where high realism and image quality are crucial.
- Weaknesses: Slower training requires more epochs to achieve high-quality images, and is sensitive to hyperparameter tuning and stability issues.

Conclusion:

The choice between C-VAE and C-GAN depends on the specific requirements of the task at hand. If faster training and stability are more important, C-VAE is the preferred model. However, if the goal is to generate the highest quality and most realistic images, C-GAN is the better option, despite its longer training time.

References:

1. Carraro, T., Polato, M., Bergamin, L. and Aiolli, F., 2022, September. Conditioned variational autoencoder for top-n item recommendation. In International Conference on Artificial Neural Networks (pp. 785-796). Cham: Springer Nature Switzerland.

2. Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K. and Ha, D., 2018. Deep learning for classical japanese literature. arXiv preprint arXiv:1812.01718.