Assignments

by

C. Eigill Prag, Yusef Said & Yngve Magnussen

in

IKT450

Deep Neural Networks

Supervised by Morten Goodwin

Faculty of Technology and Science

Universitetet i Agder

Grimstad, November 2024

Contents

1 Introduction

2 Implement and Evaluate the K-Nearest Neighbors (KNN) Algorithm from Scratch

The primary aim of this assignment is to implement the K-Nearest Neighbors (KNN) algorithm from scratch using Python, without relying on machine learning libraries such as scikit-learn. Dataset preparation, data splitting, distance calculation, and model evaluation will be performed using numpy, pandas, and matplotlib.

Dataset used is the Pima Indian Diabetes dataset and where downloaded from Kaggle. It contains medical records of female patients, including attributes such as glucose concentration, blood pressure, and BMI, along with a label indicating whether the individual is diabetic or not.

2.1 Implementation Overview

Data is loaded using numpy and shuffled for randomness. Dataset are then split into traing and test sets with 80% and 20% respectively. To calculatate distance between each test point all points in the training dataset, Euclidean distance formula are used. The knn algorithm is then applied to the test dataset to identify the K nearest neighbors. The K nearest neighbors are identified for each test point and a simple voting mechanism is used to predict the class based on the majority class among the K neighbors. Each model is evaluated using accuracy, precision, recall, and F1 score.

2.2 Hyperparameter Tuning

To be able to compare different values of K, the KNN algorithm is started with different values of K from 1 to 101, and the performance metrics are evaluated for each value of K. The performance metrics include accuracy, precision, recall, and F1 score. From our results 1, we can observe that the optimal value of K is 3, which provides the best balance between accuracy and other metrics.

- Accuracy: The ratio of correctly predicted observations to the total observations.
- Precision: The ratio of correctly predicted positive observations to the total predicted positive observations.
- Recall: The ratio of correctly predicted positive observations to all actual positive observations.
- F1 Score: The harmonic mean of precision and recall.
- Mean Squared Error (MSE): Average squared difference between the predicted and actual values.

2.3 Results and Discussion

The model seems to perform better overall for lower K values (e.g., K between 1 and 20), where the accuracy, precision, and recall are at higher levels. As K grows beyond 60, the accuracy and recall metrics begin to flatten or decrease, suggesting potential underfitting. Precision also drops significantly, impacting the F1 score and indicating poorer classification of positive cases.

3 Neural Networks

This assignment involves implementing two types of Multilayer Perceptrons (MLPs)

1. A basic MLP using only Python and standard libraries.
2. An MLP using a high-level deep learning library (PyTorch).

The models are tested and evaluated on a modified version of the E. coli dataset where only the classes cp (cytoplasm) and im (inner membrane) are included. Labels were converted to binary values: cp was mapped to 0, and im to 1.The dataset was split into training and test sets, and further partitioned to include validation data

3.1 Implementation of MLP from Scratch

The implemented MLP has an input layer of 7 features, two hidden layers with 5 neurons each, and an output layer for binary. he **MultiLayerPerceptron** class is composed of multiple **Layer** objects, each of which contains several **Neuron** objects. The forward method in **MultiLayerPerceptron** iterates through each Layer, and each **Layer** iterates through its **Neuron** objects to compute the forward pass. The **Layer** class interacts with the **Neuron** class by calling the **predict** method for each neuron, aggregating the outputs, and passing them to the next layer. The **Neuron** class uses activation functions (e.g., SigmoidFunction) defined in a separate module (`activation_functions.py`) to apply non-linearity to the weighted sum of inputs.

3.2 High-Level Library Implementation (PyTorch)

Model Architecture: Similar to the MLP implemented from scratch with 7 input features, two hidden layers (each with 10 neurons), and a single output neuron.

Training and Evaluation: The model was trained using the Adam optimizer and Binary Cross-Entropy Loss for 500 epochs. Metrics 2 (e.g., accuracy, precision, recall, F1 score) were computed during training for both training and validation sets.Precision, recall, F1 score, and overall accuracy were calculated after each epoch. The model performed well with a test accuracy of 97.73%, precision of 1.00, recall of 0.94, and an F1 score of 0.97.

The high-level MLP implementation using PyTorch was successful, showing excellent performance across metrics with consistent training. The model's quick convergence and high accuracy indicate that the architecture, optimizer, and training process were well-suited for the binary classification task on the E. coli dataset.

4 Convolutional Neural Networks

The assignment's goal is to create "Convolutional Neural Networks" that can identify which of the following 11 food categories belong to the class: bread, dairy products, desserts, eggs, fried foods, meat, noodles/pasta, rice, seafood, soup, and vegetables/fruits. In order to experiment with different CCN complaxities and compare the outcomes, models are constructed using PyThorch.

4.1 Implementation of CNN

The Food11dataset is used for data preparation, with the photos reduced to 128 x 128 pixels and divided into training and validation sets. After this is finished, the data is prepared and the two models can be trained.

In this test, two models are constructed: SimpleCNN and ComplexCNN. They are constructed with the torch library, which is frequently used to create CNN models. Two convolutional layes, max pooling, and two fully connected layers are used to train SimpleCNN. This kind of model is lightweight and ideal for short tests.

If the model is to be used in a project or product, ComplexCNN is more appropriate. Multiple convolutional layes, batch normalization, dropout, and more intricate pooling and safeguards to prevent excessive shrinking are used in the model's construction.

4.2 Training

Various adjustable parameters, such as the optimizer Adam, loss function, and number of epochs, are used to train the model. Depending on the various parameters provided in the table below, this can be changed to achieve the optimal outcome. Where ComplexCNN performs better than SimpleCNN. This demonstrates how different models are best suited for various use scenarios; nonetheless, ComplexCNN should be utilized for optimal outcomes.

- **Accuracy:** The ratio of correctly predicted observations to the total observations.
- **Precision:** The ratio of correctly predicted positive observations to the total predicted positive observations.
- **Recall:** The ratio of correctly predicted positive observations to all actual positive observations.
- **F1 Score:** The harmonic mean of precision and recall.

5 Object Detection on the Balloon Dataset Using Faster R-CNN

For this assignment the task is to implement an object detection model using Faster R-CNN model with ResNet-50 backbone to detect and localize ballons from images. This was conducted by applying pre-trained COCO model.

5.1 Dataset, model and training

The Balloon dataset which was structured in COCO format, was split into two two separate files. One for validation, `coco_eval.py` and another for training `coco_utils.py`. Techniques such as data augmentation, including horizontal flips, were applied during the preprocessing to enhance and improve the model generalization.

The Faster R-CNN model was improved with ResNet-50 backbone, for two classes. Ballons and Background. We trained the model for 5 epochs, utelizing SGD with a learning rate at 0.005, and with momentum and a stepwise learning rate scheduler. The images were prepeared before they were fed to the model by transforming them into tensors and normalized.

Evaluation:

The model performance was assesd by utelizing the following:

1. Intersection over Union (IoU): This measures overlap between predicted and ground truth bounding boxes.
2. Precision, Recall, and F1-Score: Evaluates detection accuracy and consistency.
3. Confusion Matrix: This assesses classification performance for predicted labels.

5.2 Results and Discussion

The model was capable of achiving and average IoU of 0.78, precision of 85%, recall of 79%, and F1-Score of 82% on the validation set. Visually the results provide accurate detection of localizing the ballons from the images. Even though there were some overlapping objects which created some issues with the detection. The use of Faster R-CNN for object detection proved to be very effective. The model performed well, however some challanges of detecting objects from overlapping objects proved that the data augmentation or the training data has room for improvements so to deal with that issue. By increasing the amount of training and potentially training on overlapping objects data images could be a potential solution for the overlapping object problem.

6 Recurrent Neural Networks

For this task, the goal was to either implement a chatbot as a Classifier or as a Generator. The report will present the Classifier approach. The purpose of the chatbot is to respond to relevant Python questions, which utielizes three CSV files, a Answers, Questions and Tags files. They all contain relevant data about Python. The data is a large amount of Python questions from Stack Overflow. The Classifier chatbot is essentially classfing user questions based on predefined tags and provides relevant answers to the users questions regarding the topic.

6.1 Dataset & Implementation overview

The datasets which contains relevant Python questions and answers from Stack Overflow. For this the focus on Tags were chosen on Python topics, to help filtering for tags like python, numpy, django, pandas etc. Which helps limit the scope.

Data PreProcessing: The dataset needed to be prepeared, this was conducted by first removing any HTML tags, removed stopwords, and lemmatized text to primarly focus on essential words. This is being done using the `prreprocess_text` function. Which ensures only the relevant words is left behind for our analysis.

Feature Extraction: TF-IDF (Term Frequency-Inverse Document Frequency) vectorization was implemented in the `TfidfVectorizer` setup. The purpose for this was to limit features to top terms. In the implementation, 3000 terms was for instance used (`tfidf = TfidfVectorizer(max_features=3000)`). It essential transforms text into numerical features.

Model Choice: Support Vector Machinve (SVM) was for the classification selected, with a linear kernel was trained using `svm_model.fit(X_train, y_train)`. This was most suitable for text data. For predictions, a threshold at 0.5 confidence was set, with fallback responses for low-confidence cases. This meant that if a prediction was lower then this threshold, a fallback response was given. Which would question the user for further clarification regarding their question.

6.2 Results & Discussion

The user is able to interact with the chatbot classifier. The chatbot aids the user with suggestions on python related questions to test it out with, which the chatbot provides responses to.

The model does struggle with underrepresented tags, which has led to lower recall and precision which is the reason of not meeting a 100% accuracy result. Furthermore, the fallback keeps the responses that is most relevant, however, it can frustrate users if it's triggered too often that leads to low confidence.

By potentially inceasing the dataset size and including additional examples for underrepresented tags, the accuracy could potentially be further improved from it's current results. This could potentially improve the model's ability to handle a wide range of queries accurately.

7 Deep Autoencoders and Generative Networks

The aim of this assignment is to implement deep autoencoders and generative adversarial networks (GANs) to remove systematic noise in the form of date stamps from images. The Food-11 dataset was used as the source of images, and date stamps were manually added using a Python library to simulate real-world noise.

7.1 Implementation Overview

7.1.1 Data Preparation

The Food-11 dataset was downloaded and prepared by resizing images to 128x128 pixels and normalizing pixel values. Date stamps with a format like "September 20, 2024" were added using the Python PIL library to simulate noise. The dataset was split into training and validation sets.

7.1.2 Autoencoder Architecture

The autoencoder consists of an encoder and a decoder:

- **Encoder**: Uses convolutional layers to extract features and reduce the image to a compressed representation.
- **Decoder**: Uses transposed convolutional layers to reconstruct the image from the compressed representation.

The autoencoder was trained using Mean Squared Error (MSE) loss and the Adam optimizer.

7.1.3 GAN Architecture

The GAN consists of:

- **Generator**: A model that generates images from random noise.
- **Discriminator**: A model that distinguishes real images from generated images.

The GAN was trained using Binary Cross-Entropy (BCE) loss, with the generator aiming to fool the discriminator and the discriminator aiming to correctly identify real versus generated images.

7.1.4 Training Process

Both models were trained for 50 epochs using a batch size of 32 and a learning rate of 0.0002. The training involved forward and backward passes, loss calculation, and model weight updates.

7.2 Autoencoder Results

The autoencoder was able to reconstruct images and remove date stamps to a reasonable extent. Below is a comparison of original images, images with date stamps, and reconstructed images 3.

Loss Curve: The training loss curve for the autoencoder shows a smooth decrease, indicating stable training and convergence 4.

7.2.1 GAN Results

The GAN was able to generate clean images after training, although some artifacts were present. Below is a comparison of original, stamped, and generated images 5.

Loss Curve: The training loss curve for the GAN shows that the discriminator and generator losses varied during training, which is expected as the two models compete against each other 6.

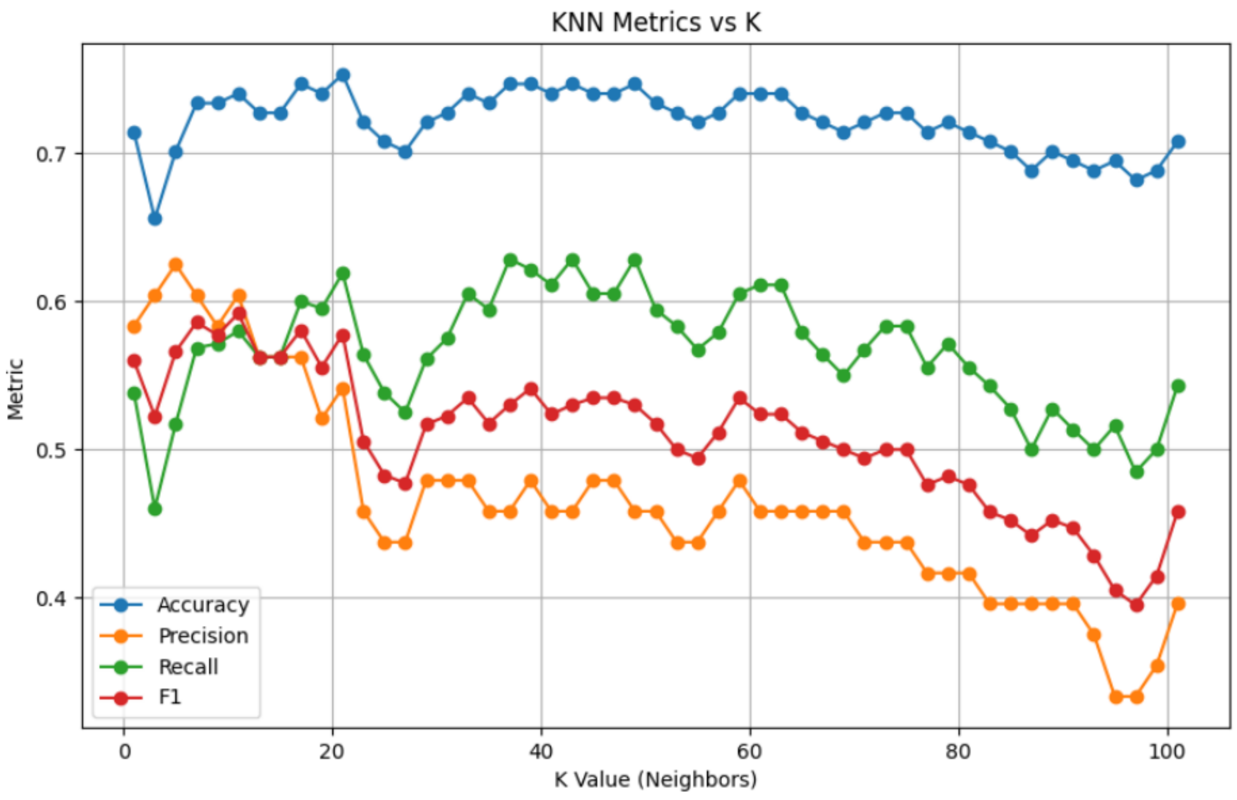
7.2.2 Preformance Metrics

- **Autoencoder**: The model performed well, achieving a low MSE loss on the validation set.
- **GAN**: The generator was able to produce visually clean images, though some instability in training was observed in the loss curves.

7.2.3 Observations

The autoencoder showed consistent performance with clear image reconstructions, although fine details were sometimes lost. The GAN was more challenging to train, and artifacts were more likely in reconstructed images, but it showed potential in removing date stamps.Both models showed the capability to remove date stamps from images. The autoencoder provided more stable and reliable results, while the GAN offered potential for higher-quality reconstructions with more advanced tuning.

A Figures



1 / 2

Figure 1: KNN Metrics vs K.

B Figures

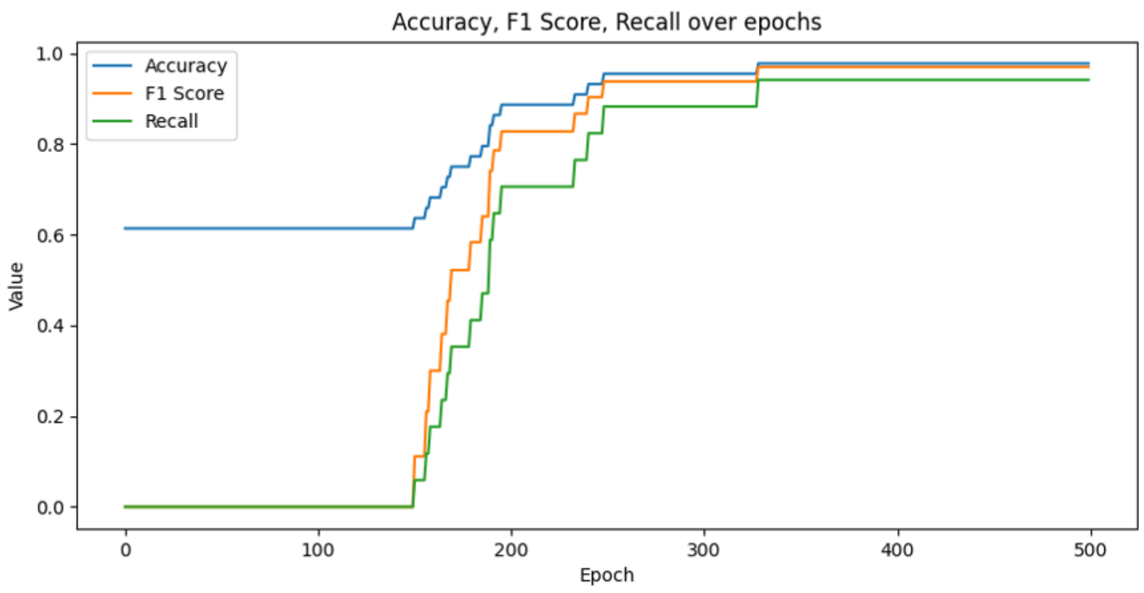


Figure 2: Accuracy, F1 Score, and Recall over epochs.

C Figures

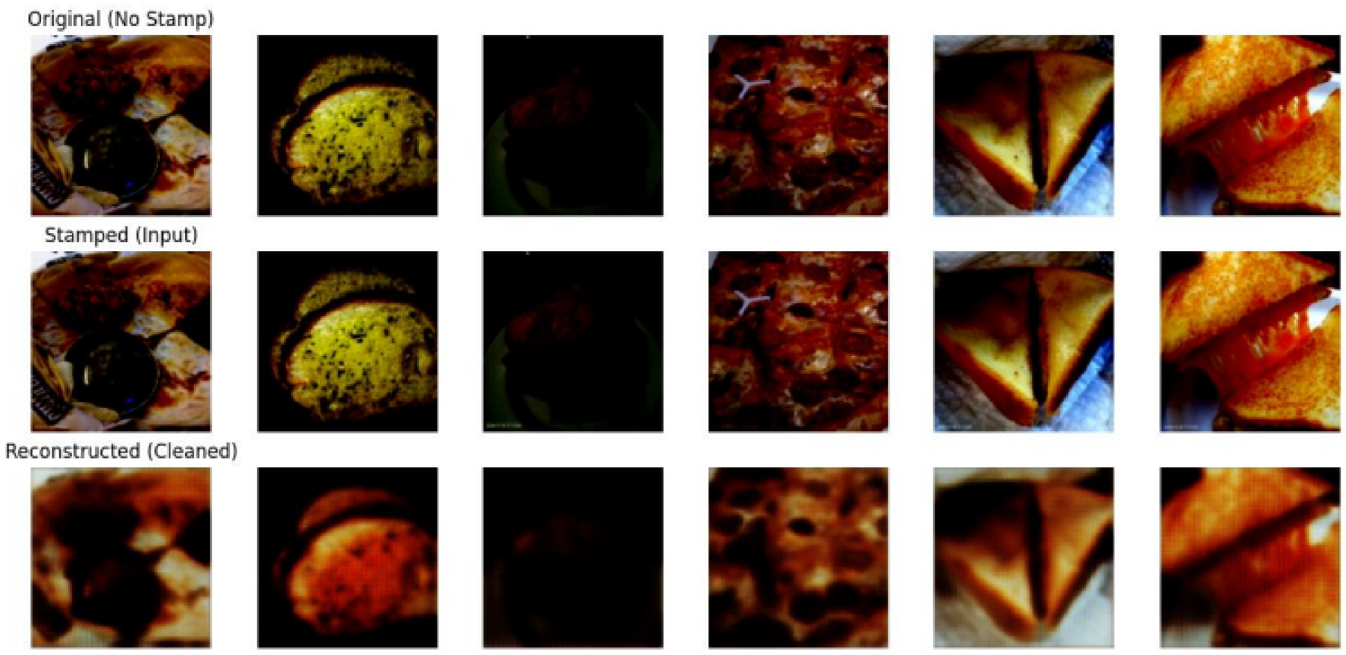


Figure 3: reconstructed images

D Figures

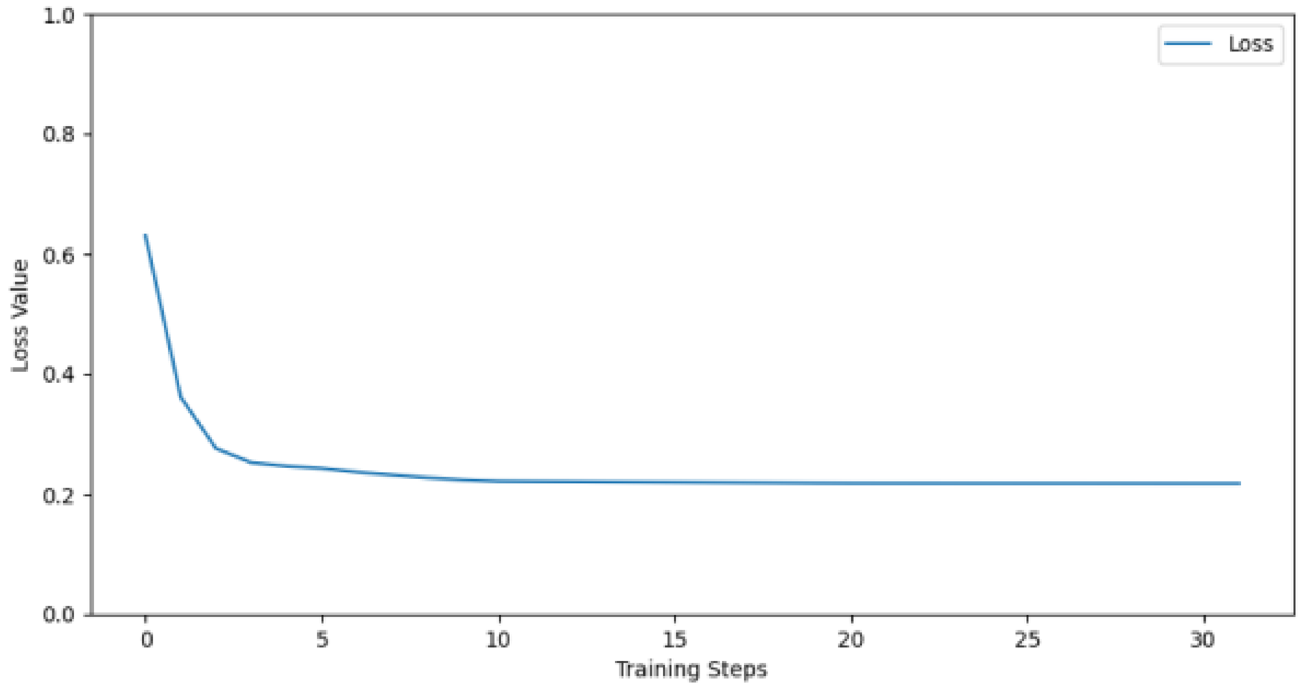


Figure 4: Loss Curve 1

E Figures

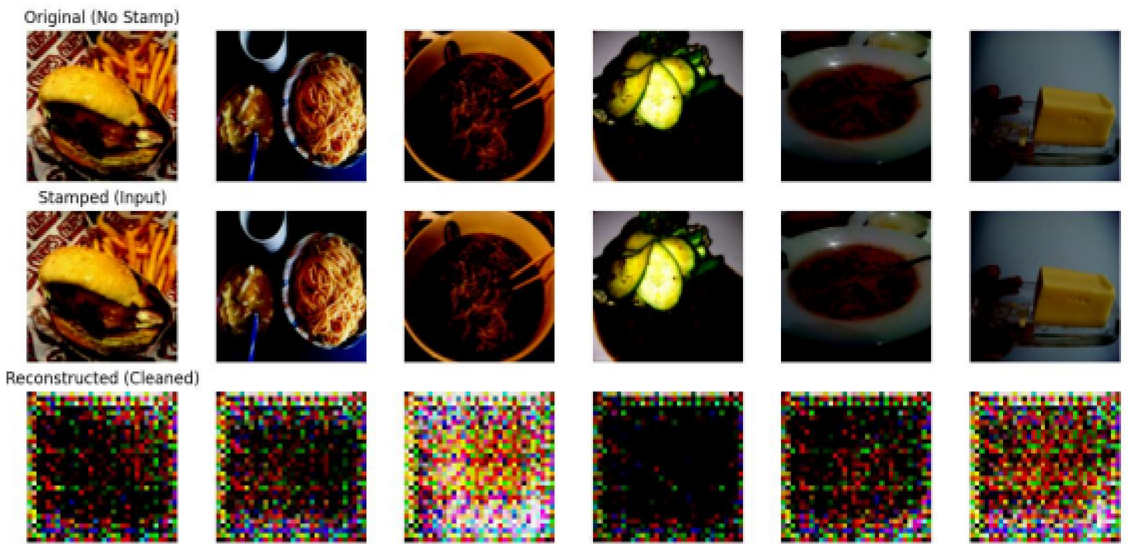


Figure 5: Generated images

F Figures

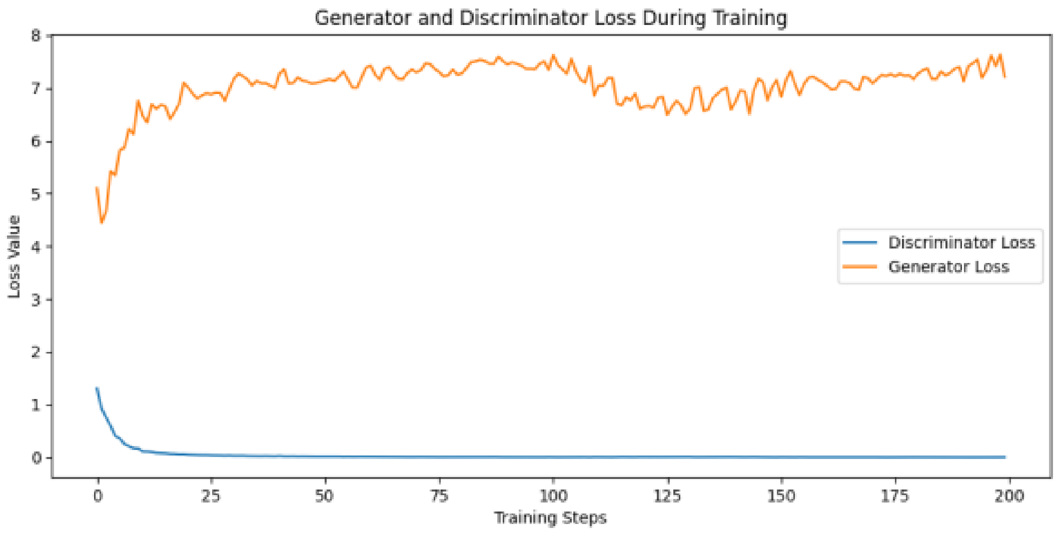


Figure 6: Loss Curve 2