Project 3 Report

IMAGE INPAINTING OF BEANS

Authors:

Martyna Stasiak 156071

Maria Musiał 156062

Link to the GitHub repository containing code for the project:

https://github.com/mmartyna123/CV-ImageInpainting

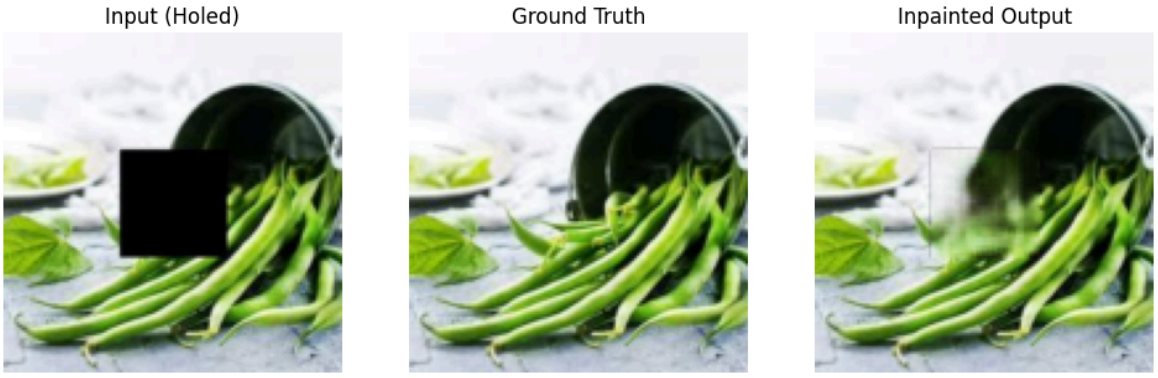
Table with points

Introduction

Image inpainting is the process of restoring missing or damaged parts of an image by generating visually coherent content. The missing regions, or holes, can be artificially created or result from damage in old photographs.

The goal is to reconstruct a complete image by seamlessly filling in these gaps. Given an image with a hole and a mask indicating the missing area, the expected output is a realistic, fully restored image.

This project explores different techniques to achieve high-quality inpainting results, ensuring smooth blending and structural consistency.



Dataset

We chose the vegetable dataset from Kaggle. It has over 20,000 images of various vegetables, but we have decided to work only with Bean data (also we have discovered that the dataset named the vegetables incorrectly since those are not beans but rather peas in the pods, but we will refer to them as beans). This Bean dataset consists of 1400 images of size 224x224. Those are examples of images:



Our own part of the dataset

Additionally, we have also collected 567 images of various beans from the internet using a web scraper. These images are used solely for evaluation, as they contain noise and variability. The dataset includes a mix of recipe photos, plant images, and other visuals, with diverse and inconsistent backgrounds, making it more challenging for inpainting tasks.



Fun fact: scraper also returned this image with the query "raw green beans":

The **dataset is split** 70%-10%-20%, resulting in 1008 training, 112 validation and 280 test images. All images are normalized to <0,1> for better learning.

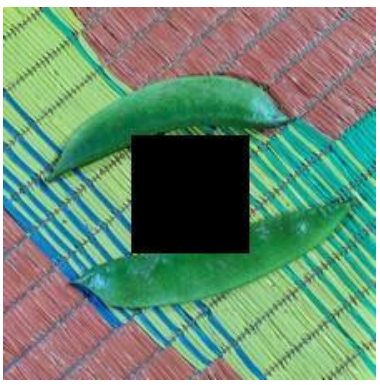
Data Augmentation

To increase dataset diversity, we have applied **data augmentation** with

- Random rotation (-15 to 15 degrees)
- Horizontal flip (50% probability)
- Vertical flip (50% probability)
- Random brightness/contrast adjustments
- Random contrast adjustment (Gamma)
- Random crop to the same size as the original images
- Random blur (Gaussian blur)

This is to expand the variety of the dataset, so that the model learns more about beans, in different rotations, orientations and contrasts.

We generate holes in images using a custom function, which places a black rectangle in the centre. A corresponding binary mask is also created, serving as an additional input to the model.


Loss functions and metrics

To assess the performance of the models, we have used a combination of loss functions and evaluation metrics to measure both pixel-level accuracy and perceptual quality.

Loss Functions

- Masked Mean Squared Error (MSE) Loss

  It is the primary loss function that we have used. It ensures that only the missing (holed) region contributes to the loss calculation. This prevents the model from being penalized for correctly preserving unmasked areas.

- Mean Absolute Error (MAE) Loss

  We have used it in loss function testing; It computes the absolute pixel difference instead of the squared error.

- Perceptual Loss

  We have used it in loss function testing; Instead of comparing raw pixel values, this loss compares high-level feature maps from different layers.

- Adversarial Loss (for GAN Model)

  In the GAN-based model, we have used Binary Cross-Entropy (BCE) loss to train the discriminator. The generator tries to fool the discriminator by generating realistic inpainted regions, while the discriminator learns to distinguish real from fake images.

Evaluation Metrics

- Masked Mean Squared Error (MSE) Loss
- Peak Signal-to-Noise Ratio

  PSNR measures the difference between the generated and ground truth images, providing a numerical score of image quality, where the higher PSNR indicates better reconstruction.

Architectures

Autoencoder

The autoencoder-based architecture for image inpainting consists of an encoder, a bottleneck, and a decoder. The encoder processes the input—an image with a missing region and a mask—using a series of convolutional layers, reducing its spatial dimensions while capturing essential features. The bottleneck further compresses the representation before passing it to the decoder, which reconstructs the missing parts by upsampling through transposed convolutions. The final output is a three-channel image with values normalized between 0 and 1 using a sigmoid activation function. This structure allows the model to learn how to restore missing regions based on the surrounding context.

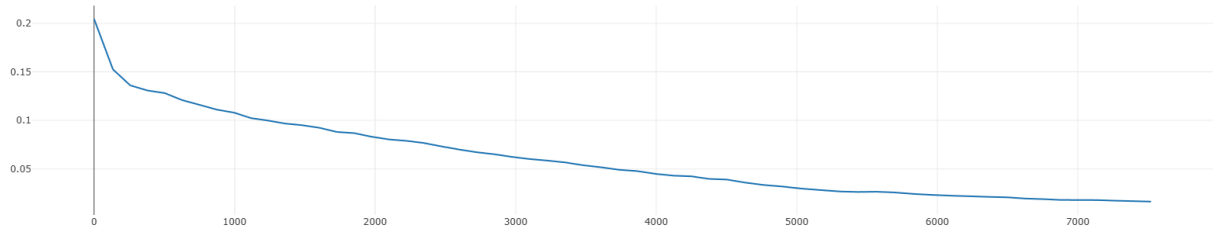
Diagram of the model: Autoencoder Diagram

Model Analysis:

- Size in memory (total): 107.02 MB
- Number of parameters: 13,904,003
- Training time: 2.1h

Training and evaluation of Autoencoder

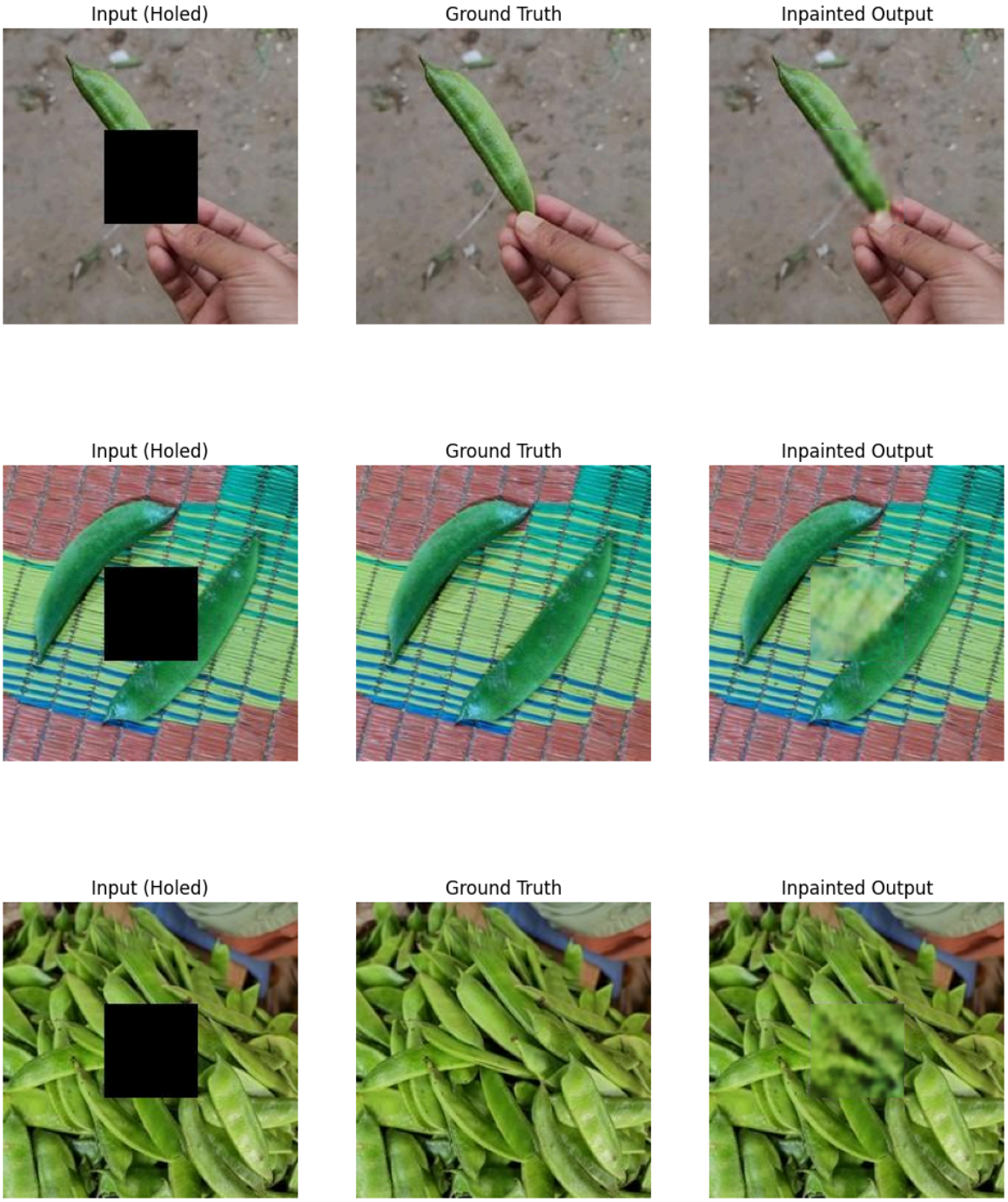
Metric	Latest	Min	Max
Train Loss	0.016333588573615998 (step=59)	0.016333588573615998 (step=59)	0.20450065843760967 (step=0)



Masked MSE in each epoch during training

Metric	Value
Test Loss	0.08312534168362617
Test PSNR	21.95092165023974

Results of Autoencoder:



Autoencoder with skip connections

The Autoencoder with Skip Connections improves image reconstruction by keeping more spatial details. It has an encoder, which downsamples the input with convolutional layers, a bottleneck that compresses the data, and a decoder that upsamples it back. Unlike a standard autoencoder, this model has skip connections, which pass features from the encoder to the decoder, helping to retain important details. The final layer uses a sigmoid activation to keep pixel values between <0,1>. This setup allows the model to generate more accurate inpainted regions while being efficient.

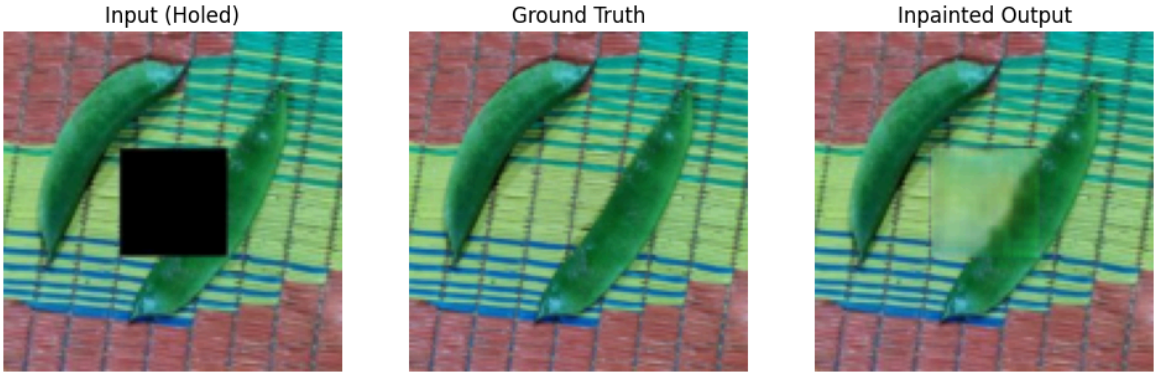
A full diagram of the model: Autoencoder with skip connections Full Diagram

Simplified diagram: Autoencoder with skip connections Simplified Diagram

Model Analysis:

- Size in memory (total): 38.82MB
- Number of parameters: 1,546,755
- Training time: 53min

Results of Autoencoder with skip connections



Training

Testing was done on this model since it was the fastest that we have made and it was the best one to do so.

Hyperparameters & Hyperparameter tuning

Done using optuna. We tested 20 arrangements, each on 20 epochs. Additionally, we put 3 **different optimizers** to test. Adam optimizer was the best, but RMSprop also seemed good. To determine which is really better we would need to run it on more epochs, what was computationally expensive.

We performed hyperparameter optimization on 20 epochs alongside optimizer testing. From this, we chose a Learning rate of 0.01, Batch size of 32, and Adam optimizer. Tested were:

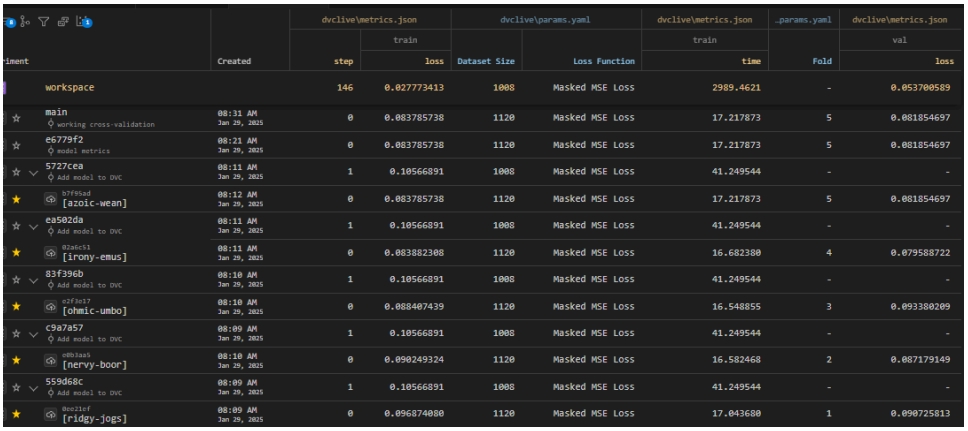
- 🫛 Learning rate: 1e-4 to 1e-2
- 🫛 Batch size: 32, 64, 128
- 🫛 Optimiters: Adam, SGD, RMPprop

```
{'lr': 0.01, 'batch_size': 32, 'optimizer': 'Adam', 'loss': 0.07367236132267863}
```

All Hyperparameter Tuning Results

Cross-validation

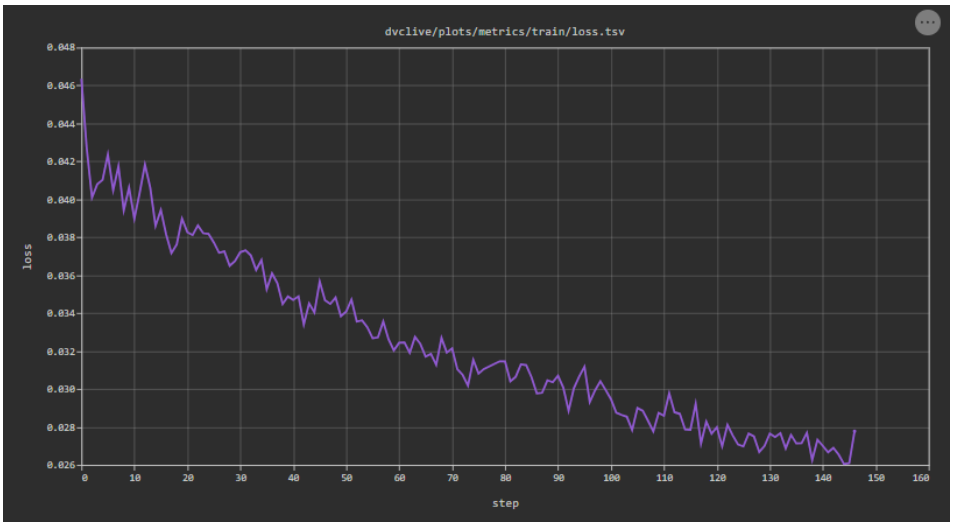
We performed 5 k fold as cross-validation. It was run on 20 epochs each fold. We can see that their losses are similar, which leads us to believe that the model is generalizing data enough. Here are the parameters logged for their runs:



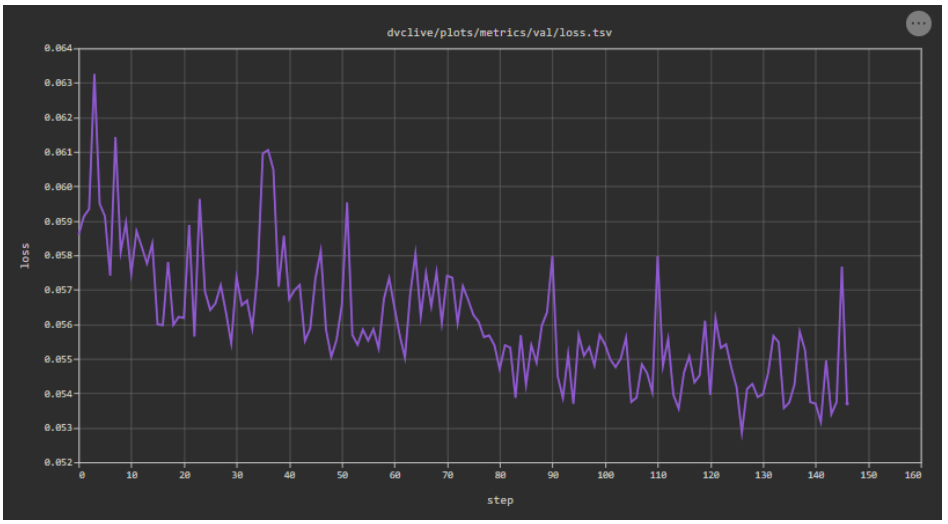
iment	Created	dvclive\metrics.json train step	dvclive\metrics.json train loss	dvclive\params.yaml Dataset Size	dvclive\params.yaml Loss Function	dvclive\metrics.json train time	_params.yaml Fold	dvclive\metrics.json val loss
workspace		146	0.027773413	1008	Masked MSE Loss	2989.4621	-	0.053700589
main working cross-validation	08:31 AM Jan 29, 2025	0	0.083785738	1120	Masked MSE Loss	17.217873	5	0.081854697
e6779f2 model metrics	08:21 AM Jan 29, 2025	0	0.083785738	1120	Masked MSE Loss	17.217873	5	0.081854697
5727cea Add model to DVC	08:11 AM Jan 29, 2025	1	0.10566891	1008	Masked MSE Loss	41.249544	-	-
b7f65ad [azoic-wean]	08:12 AM Jan 29, 2025	0	0.083785738	1120	Masked MSE Loss	17.217873	5	0.081854697
ea502da Add model to DVC	08:11 AM Jan 29, 2025	1	0.10566891	1008	Masked MSE Loss	41.249544	-	-
02a6cb1 [irony-emus]	08:11 AM Jan 29, 2025	0	0.083882308	1120	Masked MSE Loss	16.682380	4	0.079588722
83f396b Add model to DVC	08:10 AM Jan 29, 2025	1	0.10566891	1008	Masked MSE Loss	41.249544	-	-
e2f3e17 [ohmic-umbo]	08:10 AM Jan 29, 2025	0	0.088407439	1120	Masked MSE Loss	16.548855	3	0.093380209
c9a7a57 Add model to DVC	08:09 AM Jan 29, 2025	1	0.10566891	1008	Masked MSE Loss	41.249544	-	-
e0b3aa5 [nervy-boor]	08:10 AM Jan 29, 2025	0	0.090249324	1120	Masked MSE Loss	16.582468	2	0.087179149
559d68c Add model to DVC	08:09 AM Jan 29, 2025	1	0.10566891	1008	Masked MSE Loss	41.249544	-	-
0ee21ef [ridgy-jogs]	08:09 AM Jan 29, 2025	0	0.096874080	1120	Masked MSE Loss	17.043680	1	0.090725813

Each run is starred on the left; Don't mind the working model at the top, there is no option to hide it.

Train Loss:



Validation Loss:



Evaluation on test:

```
Test Loss: 0.0591
Test PSNR: 26.9849
Total Test Images: 320
```

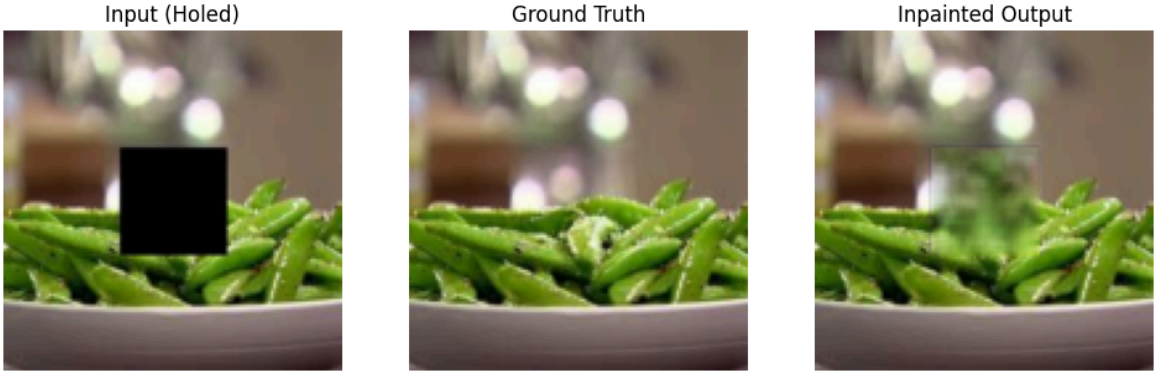
Performance on Images from the Internet:

We used the scraped images only for testing how our model performs, as it was too noisy to use for training. Here are the results:

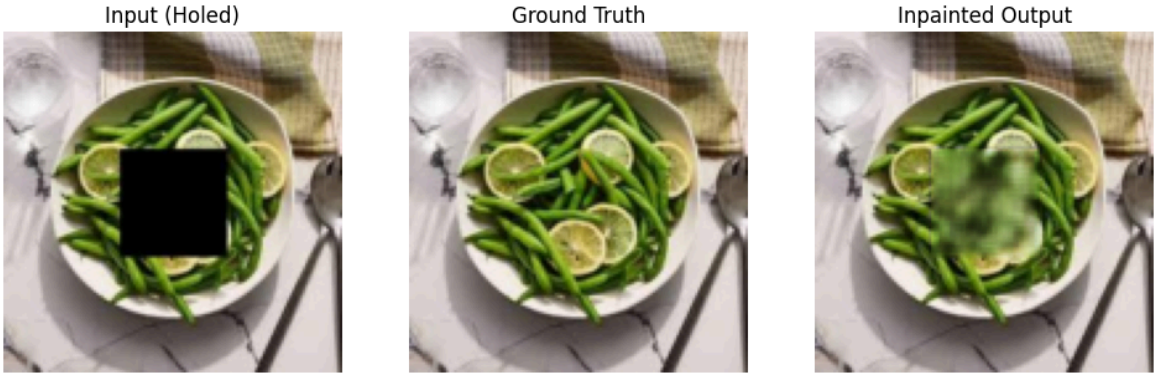
Evaluation:

```
Test L1: 0.0954
Test PSNR: 29.8129
Total Test Images: 576
```

It performs similarly to our initial dataset.



We can see model has problems with additional things in the middle, like citrus here:



Input (Holed)

Ground Truth

Inpainted Output

Input (Holed)

Ground Truth

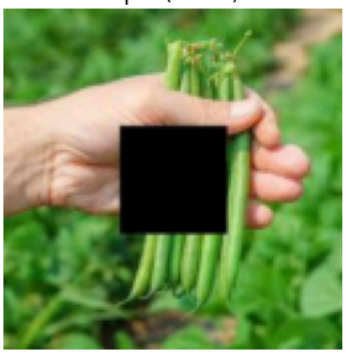
Inpainted Output

Input (Holed)

Ground Truth

Inpainted Output

Input (Holed)



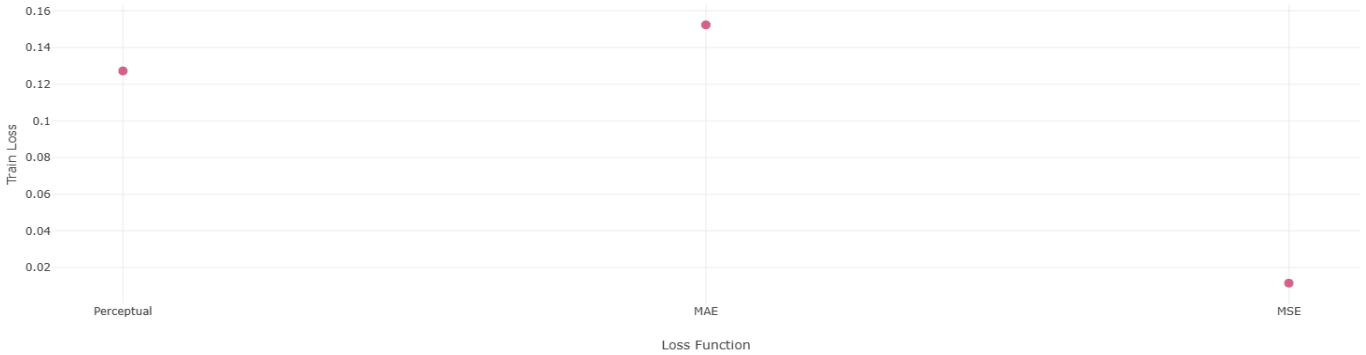
Ground Truth

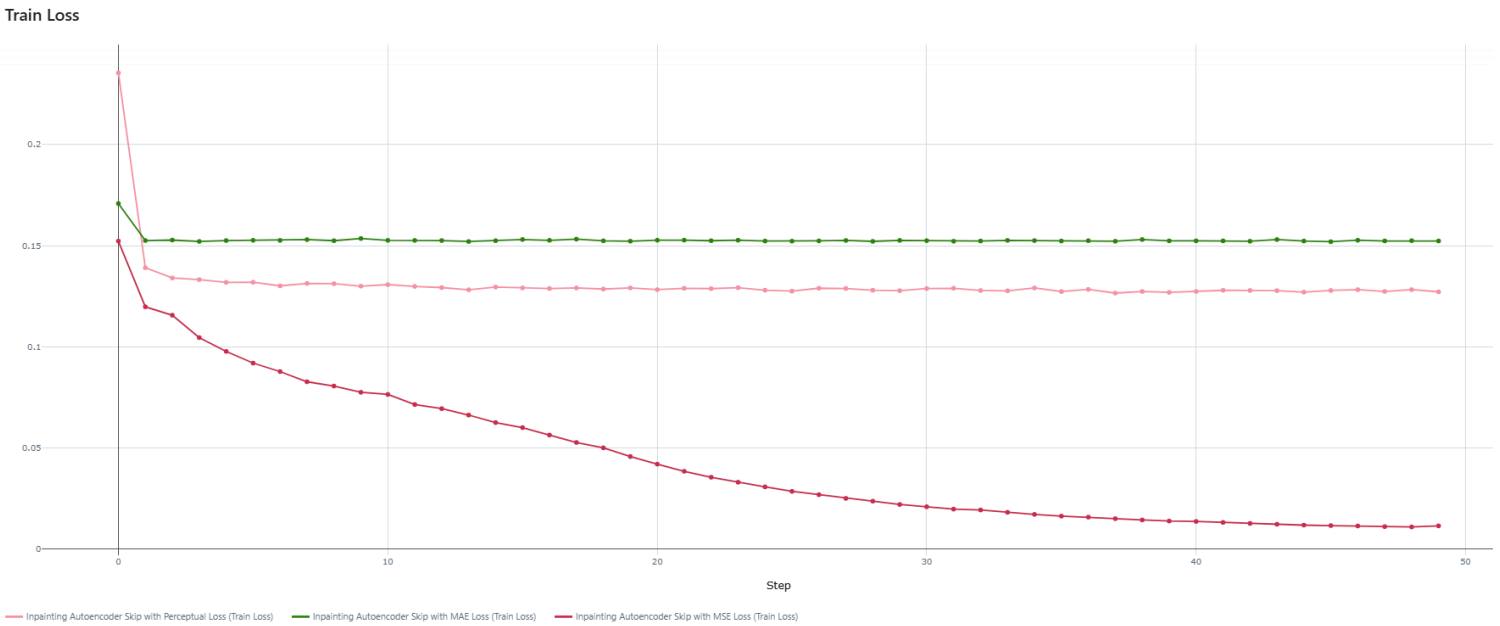
Inpainted Output

Testing Loss Functions

For testing different loss functions we have chosen (masked) Mean Squared Error, Mean Average Error and Perceptual Loss; they are explained at the beginning in the Loss functions and Metrics section.

From this, we got:





From the plots and images, we got that the masked MSE (red on plots) is the better than Perceptual Loss(pink) and Mean Average Error(green).

GAN with generator Autoencoder with skip connections

To improve the quality of inpainted images, we have also implemented a Generative Adversarial Network (GAN) for image inpainting. This approach consists of two neural networks: a Generator and a Discriminator, trained together in an adversarial setup.

The Generator follows an encoder-decoder structure with skip connections. It takes a holed image along with a mask as input and reconstructs the missing region, ensuring continuity in texture and colour. The Discriminator is a convolutional network that distinguishes between real and generated images, pushing the generator to produce more realistic results.

During training, we have used used a combination of losses:

- Masked MSE Loss to ensure pixel-wise similarity in the missing region.
- Adversarial Loss to encourage realistic textures.
- Total Variation Loss to reduce artefacts and enforce smoothness.

The model was trained for 60 epochs with label smoothing for the Discriminator to improve stability. The Generator loss was weighted to prioritize reconstruction over adversarial feedback.

Full diagram of the model's generator: Generator Full Diagram

Simplified diagram of the model's generator: Generator Simplified Diagram

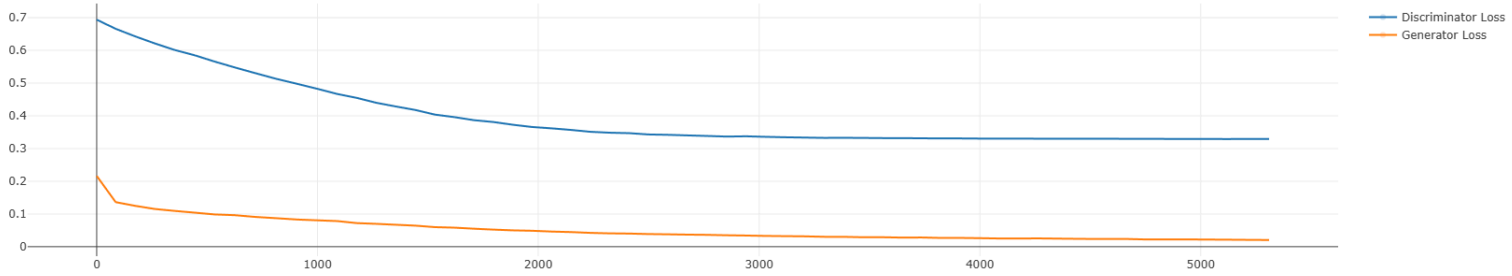
Diagram of the model's discriminator: Discriminator Diagram

Model Analysis:

- Size in memory (total): 86.33 (Generator) and 11.22 (Discriminator)
- Number of parameters: 6,173,699 (Generator) and 663,745 (Discriminator)
- Training time: 1.5h

Training and evaluation of GAN

Metric	Latest	Min	Max
Generator Loss	0.020343970856629312 (step=59)	0.020343970856629312 (step=59)	0.2163631310686469 (step=0)
Discriminator Loss	0.32928864285349846 (step=59)	0.3290625363588333 (step=57)	0.69399294257164 (step=0)

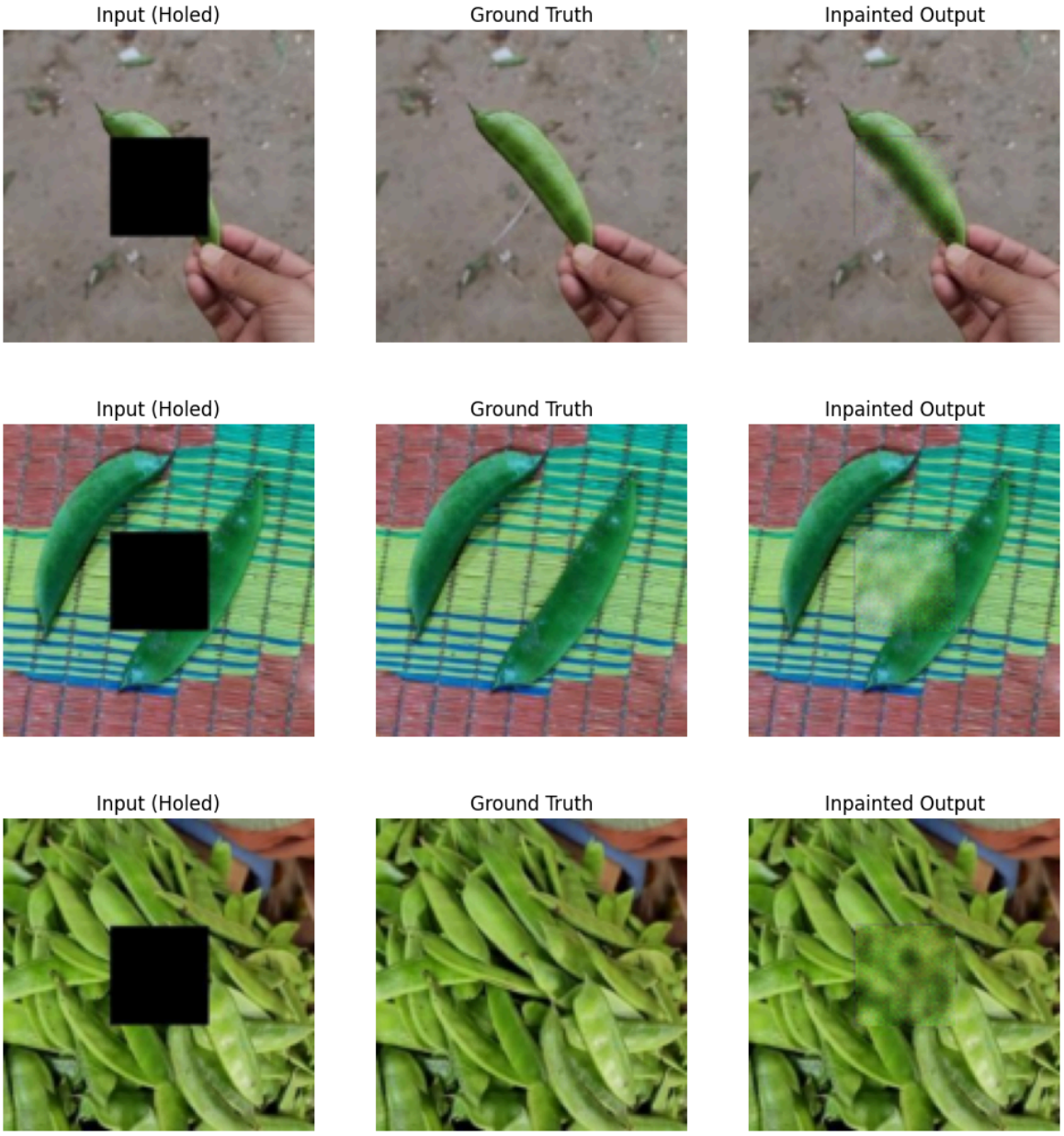


Generator loss (Orange) and Discriminator loss (Blue) each epoch during training

Metric	Value
Test Loss	0.06938419491052628
Test PSNR	21.852040724700288

Evaluation on the test set

Results of GAN



Architecture tuning

We have 3 architectures: Autoencoder, Autoencoder with skip connections and GAN based on previous architecture. Looking at the results of metrics and inpainted images we can deduce that the best architecture is Autoencoder with skip connections.

Tools

MLFlow

We integrated MLflow into our workflow to systematically track and log key aspects of the training process. This included storing model checkpoints, recording loss values, capturing visualizations of inpainting results after each epoch and on the test dataset, measuring training time, and logging essential hyperparameters such as the number of epochs and learning rate.

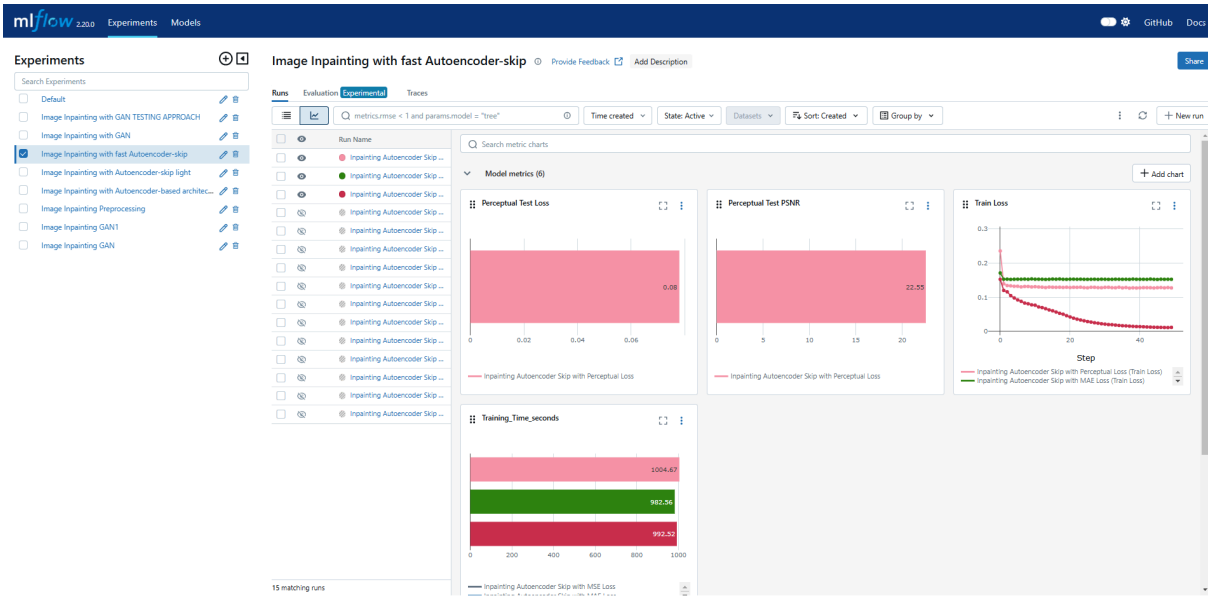
The plots from training and testing Autoencoder, GAN and testing different loss functions come from the MLFlow.

DVC

We used DVC to track the dataset with AWS S3 and manage experiment tracking. The DVC Live extension in VS Code provided real-time updates during training, allowing us to monitor progress dynamically. Through DVC, we logged models, hyperparameters, and loss values to ensure efficient version control and experiment management.

The plots from training and testing Autoencoder with skip connections and hyperparameter tuning come from the DVC.

MLFlow:



DVC:

Experiment	Created	dvclive\metrics.json train step	dvclive\metrics.json train loss	dvclive\params.yaml Batch Size	dvclive\params.yaml Dataset Size	dvclive\params.yaml ..arning Rate	dvclive\params.yaml Loss Function	dvclive\metrics.json train time	_params.yaml Fold	dvclive\metrics.json val loss
Workspace		146	0.027773413	32	1008	0.001	Masked MSE Loss	2989.4621	-	0.053700589
(HEAD detached from c0b278c)										
2db0578 final ae	03:30 PM Jan 29, 2025	0	0.083785738	64	1120	0.001	Masked MSE Loss	17.217873	5	0.081854697
60a6388 Add model to DVC	12:43 PM Jan 29, 2025	0	0.083785738	64	1120	0.001	Masked MSE Loss	17.217873	5	0.081854697
c116f71 [Training-EarlyStop]	12:43 PM Jan 29, 2025	146	0.027773413	32	1008	0.001	Masked MSE Loss	2989.4621	-	0.053700589
0173cf4 Add model to DVC	11:47 AM Jan 29, 2025	0	0.083785738	64	1120	0.001	Masked MSE Loss	17.217873	5	0.081854697
f50139c [gaunt-clay]	11:47 AM Jan 29, 2025	25	0.042540159	32	1008	0.001	Masked MSE Loss	533.07207	-	0.054683845

Completed items with points:

(each item is a reference to a corresponding section in rapport to this 'point')

Problem	3 -> Image inpainting
Models	5 -> Autoencoder(1), Autoencoder with skip connections(1), GAN with over 50% of our own layers(3)
Our own part of the dataset	1
Hyperparameter tuning	1
Architecture tuning	1
Data augmentation	1
Cross-validation	1
Testing optimizers	1
Testing loss functions	1
MLflow	1
DVC	2
SUMA	18

Requirements.txt is available on GitHub

Sources:

- https://www.kaggle.com/datasets/misrakahmed/vegetable-image-dataset
- https://medium.com/analytics-vidhya/review-high-resolution-image-inpainting-using-multi-scale-neural-patch-synthesis-4bbda21aa5bc
- https://www.geeksforgeeks.org/generative-adversarial-network-gan/
- https://aws.amazon.com/what-is/gan/
- https://aws.amazon.com/?nc2=h_lg
- https://wandb.ai/wandb_fc/articles/reports/Introduction-to-image-inpainting-with-deep-learning--Vmlldz01NDI3MjA5
- OpenCV documentation

Runtime environment

- GPU: Intel(R) Iris(R) Xe Graphics
- CPU: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
- RAM: 8GB
- OS: Microsoft Windows 11 10.0.22631
- Python 3.10.7
- Libraries are in the requirements.txt file