50.021 Artificial Intelligence

Fingerprint Inpainting

Submitted by:

Chai Bing Zhe	1005959
Ibrahim	1005972
Jin Chou Chua	1005988
Ng Sue Chi	1006112

Date: April 22, 2025

Contents

1 Introduction

Fingerprints are unique biometric markers that are used widely for identity verification. Various societal functions hinge on accurate fingerprint scans, from identity verification at access control points such as borders, hospitals and aid in crime scene investigations.

However, fingerprint degradation can happen due to eczema, manual labour and ageing, among other factors. Typical methods to solve this involve hardware, such as higher resolution imaging technology (multi-spectral scanners) or manually recreating them by hand via fingerprint experts.

A purely image-based, AI-driven method would complement these efforts and streamline various processes that use fingerprints.

2 Problem Description

Restoring missing or corrupted parts of fingerprint images is challenging, even for advanced inpainting models. In this project, we seek to develop an AI-driven image inpainting solution that accurately restores damaged fingerprint scans to ensure reliable identity verification in critical real-world applications.

2.1 Existing Works/State-Of-The-Art

Methods for image fingerprint recovery are broadly categorised into 2 main groups, filtering methods and neural networks. Classical image inpainting methods, such as diffusion and Generative Adversarial Networks (GANs), often rely on strong low-level assumptions, which may be challenged in the face of extensive masking. In the context of fingerprints, there are higher requirements on the accuracy of the restoration.

Mean Accuracy Error (MAE) and Mean Squared Error (MSE) is typically able to measure the exactness of pixel values between the inpainted and original image, with lower values signalling higher accuracy. Some commonly used metrics for inpainting are Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM). PSNR is able to quantify fidelity in the reconstructed image, measuring the "noise" introduced during inpainting, while SSIM measures the image communication efficiency.

3 Data Preparation

3.1 Original Training Dataset

3.2 Data Preprocessing

For this project, the fingerprint patterns were primarily classified into four categories: arches, loops, concentric whorls and imploding whorls. In Roboflow, the fingerprints were uploaded and labelled by drawing bounding boxes around the specific pattern regions. These labels were then associated with the corresponding classes (arches, loops, imploding whorls, and concentric whorls).

Dataset used: https://www.kaggle.com/datasets/baracuda0118/nist-db4-fingerprint-pattern

To support the fingerprint inpainting model, it was essential to first classify and label fingerprints by their core pattern type. The main purpose of this labelling process was to create training data for a dedicated **YOLOv8-based classifier model**, which was tasked with automatically identifying the fingerprint type in unseen images. By drawing bounding boxes around the central pattern area and associating them with one of four classes—**arches**, **loops**, **concentric whorls**, and **imploding whorls**—the classifier was able to learn the distinguishing visual features of each type.

The decision to classify fingerprints into these four distinct types was also driven by the hypothesis that different pattern types might vary in their difficulty of inpainting. For instance, more irregular or asymmetrical structures could pose greater challenges for reconstruction than patterns with strong radial symmetry or repetition. By separating the data into these categories,

we aimed to enable more targeted evaluation of the model's performance across each type and to later explore whether reconstruction quality correlated with fingerprint morphology.

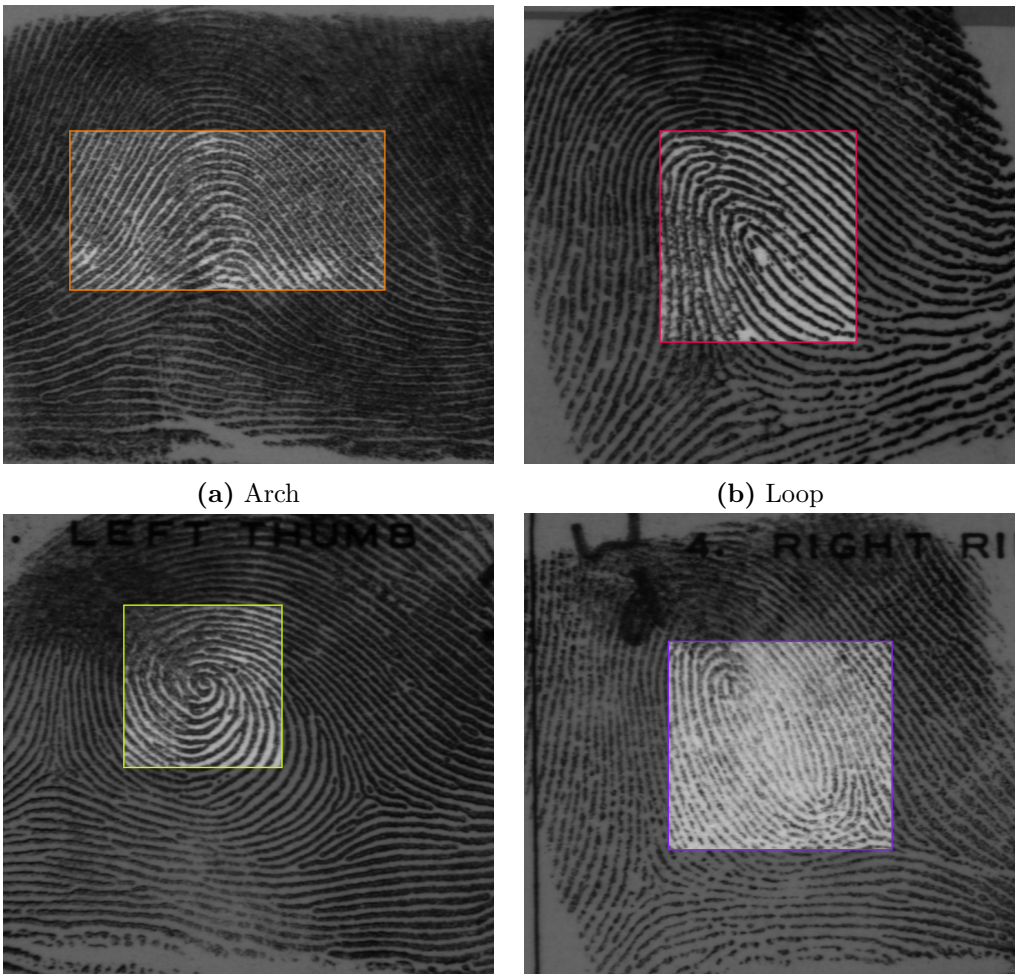
This classification step is critical in the overall pipeline, as the pattern type determines which downstream detection model is used for localizing the fingerprint's core region prior to masking. In other words, effective labelling ensures that the classifier can reliably route each image to the correct pattern-specific detector and ultimately improve the quality of both detection and inpainting.

A total of **1,243 annotated fingerprint images** were labeled and used for training, validation, and testing. The dataset was distributed across four classes as follows:

- **Arches (400 images)** – patterns with ridges that flow in a wave-like motion, generally without a core or delta.
- **Loops (395 images)** – patterns where ridges enter and exit on the same side, curving around a central point.
- **Concentric Whorls (311 images)** – circular or spiral patterns with well-defined central cores.
- **Imploding Whorls (137 images)** – irregular whorl patterns where ridge density increases toward the center, but with less radial symmetry.

During the classification stage, it was observed that **arches and loops were more difficult for the model to identify** compared to whorls. This difficulty is likely due to the structural similarity between the ridges of arches and loops and the background ridges commonly found around the main pattern area in other fingerprint types. This made it harder for the classifier to distinguish the core feature from its surrounding context.

To annotate the dataset, **Roboflow** was used as the primary labeling platform. Roboflow was chosen for its clean interface, collaborative workflow, and direct export compatibility with YOLO model formats. Annotators manually drew bounding boxes around the core pattern region in each image and labeled them according to the four class categories. These annotated images were then used to train the classifier model, enabling automatic fingerprint type identification during the preprocessing phase. Roboflow's tools significantly reduced labeling overhead and helped maintain consistency across a large number of images.



(a) Arch **(b)** Loop

(c) Concentric Whorl **(d)** Imploding Whorl

Figure 1: Example fingerprint images from each class with bounding boxes annotated around the pattern core.

3.3 Processing Pipeline

3.3.1 Design Process Iteration

Image datasets such as ImageNet and The Places used for training inpainting models contains of millions of images Çelik, 2023. As such, our team recognised the importance of preparing a sufficiently robust and large fingerprint dataset to train an effective inpainting model. Manually pre-processing the images would be impractical and unfeasible due to the lack of manpower and project funds, hence an automated image augmentation pipeline would be the best alternative.

While designing the data pipeline, the team considered the following key points:

1. Given an input directory containing unmasked fingerprint images, the data pipeline should be able to apply masks automatically onto the images and return them.

2. The masked area should be within the bounds of the fingerprint in each image and cover

key areas of interest on the fingerprint to ensure that the fingerprint patterns are masked in meaningful areas.

3. The data pipeline should run as a single workflow with several masking options and be able to make multiple augmentations to a single fingerprint image.

Considerations 1 and 3 can be achieved through simple rule-based programming. As not all fingerprints are centered in the original images, the masks could not be simply applied to the centre of images. Instead, consideration 2 could be achieved using an object detection model to first identify bounding boxes containing the fingerprint pattern. Image masks can then be applied within the pre-defined bounding boxes. This ensures that the generated image masks are within the bounds of the fingerprints and that key fingerprint patterns are masked.

3.3.2 Initial Data Pipeline Flow

The team's initial approach was to train a single object detection model on all four fingerprint patterns and use it to identify the bounding box for the fingerprint pattern. Images from the NIST DB4 fingerprint dataset was labeled on Roboflow for all four fingerprint labels. The labeled dataset is then accessed using the Roboflow Python Software Development Kit (SDK) to train a Yolov8s model from ultralytics. A training data set of 1130 hand-labeled images were used for training this model.

While the model was able to detect some fingerprint patterns with decent confidence levels, the team found that the fingerprint pattern detection model will also incorrectly identify multiple fingerprint patterns on the same fingerprint. This was likely due to fingerprint patterns sharing large similarities and the small training dataset used. While the predicted label and bounding box with the highest confidence level could be used as the pattern's bounding box, the team decided against this approach as further refining this model would require more time for manual processing of the data and carried the risk of yielding poor results still. Hence, the team went on to design a new data pipeline structure.

3.3.3 Data Pipeline Flow

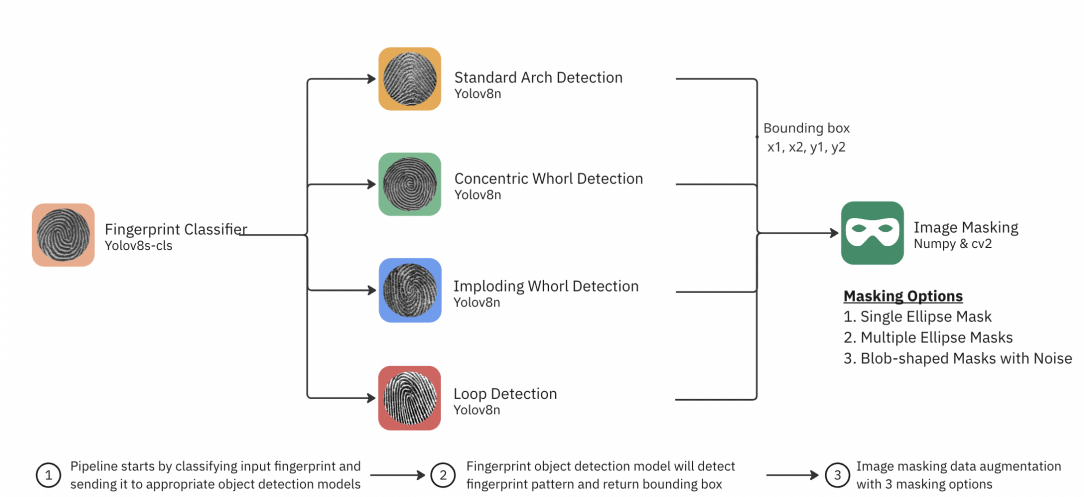


Figure 2: Image Augmentation Data Pipeline

The final data pipeline flow is as follows: Fingerprint Pattern Classification → Fingerprint Pattern Detection → Image Masking.

Fingerprint Classification

A yolov8n-cls model was trained with a training dataset of **1183 hand-labeled fingerprint images** from the NIST DB4 dataset. Some images were mislabeled in the original NIST dataset, hence the images were manually relabeled by the team on Roboflow to ensure the strictest data quality.

The dataset consisted of 392 standard arch images, 315 loop images, 280 concentric whorl images and 196 imploding whorl images. There is a class imbalance in the training dataset, however the model was still able to discern clearly between each fingerprint pattern, hence the team did not address this issue. The images were then augmented with an variation of image brightness between -15% to +15%, which increased the original hand-labeled dataset to **2799 images**. Finally, the images were split into train, validation, test datasets in the ratio of 87:9:4 (rounded up).

The image classification model used was a yolov8n-cls model from ultralytics. Hyperparameter tuning was conducted on the model with several settings (kindly refer to Section C in the appendice for the hyperparameter finetuning results and model's performance on training data), and the team found that the following hyperparameter combination produced the best model weights:

- Dropout: 0.2
- Batch size: 32
- Image size: 256

- Epochs: 20

The optimizer function was automatically set, which was a AdamW optimizer with a learning rate of 0.01.

After the input training image is classified into one of the four classes, the image will be directed to the appropriate fingerprint detection model to identify the bounding box.

Fingerprint Detection

The fingerprint detection stage consists of four fingerprint detection models, each trained to detect bounding boxes on a specific fingerprint pattern. Each detection model was trained from a yolov8n object detection model and the training data was trained on **4135 fingerprint images** from the NIST DB4 dataset.

The team started out by hand-labeling **1730 fingerprint images** for bounding box detection. Samples of our fingerprint labeling could be found in section 3.2 in the report. The data labeling process for this dataset was stricter and images that were could not be clearly defined in any class were marked as null. The images were then augmented with an increase of image brightness between 0% to 25%, which increased the original hand-labeled dataset to **4135 images**. Finally, the images were split into train, validation, test datasets in the ratio of 88:9:4 (rounded up).

The original 1730 image dataset consisted of 400 standard arch images, 395 loop images, 311 concentric whorl images, 137 imploding whorl images and 487 null images. Similar to the fingerprint classification dataset, there is a class imbalance in the training data. This is due to the lack of imploding whorl images in the dataset as well as the more stringent quality check process that filtered out a portion of the imploding whorl images. However, since each fingerprint detection model is only trained on a single class, the issue of class imbalance is not as prevalent for these models as there will be no issues of biased predictions. Nonetheless, the imploding whorl detection model was still able to accurately identify bounding boxes for imploding whorl patterns.

All four fingerprint detection models were trained with the following hyperparameters, with the best set of model weights being used in the pipeline:

- Dropout: 0.2
- Batch size: 32
- Image size: 256
- Epochs: 50

The optimizer function was automatically set, which was a AdamW optimizer with a learning rate of 0.01.
The fingerprint detection models will return a list of coordinates x1, x2, y1, y2, wherein x1 and y1 represent the xy coordinates of the top left corner of the bounding box and x2 and y2

are the xy coordinates of the bottom right corner of the bounding box.

After obtaining the bounding box coordinates for the input fingerprint image, the bounding box and image move on to image masking.

Image Masking

Image masking refers to the process of applying a mask to key areas of interest in a fingerprint image. Masking important details in training images enables inpainting models to determine the areas in an image to fill in Inpainting, n.d. This step automates the image masking process, allowing the team to create large datasets in a short period of time.

Utilizing the coordinates returned from the fingerprint detection models, a green mask will then be applied within the bounding box. This ensures that the mask will be applied to key features of the fingerprint pattern. The mask was coloured green as it contrasts greatly from the black and white fingerprint, allowing the inpainting model to identify the mask easily during training. Unlike the preceding steps in the pipeline, this step only involves rule-based programming.

To use the pipeline, the user will first specify the type of masking to apply and the number of images to generate from each training image supplied. From there, the pipeline will apply the masks automatically and return the processed images in the given directory. The masked area for different masks of the same image are the same to ensure consistency in the data.

There are 3 options for images masking:

1. **Single ellipse mask:** A single ellipse opaque green mask on the image. If more than one image is generated, a rule-based algorithm is utilized to determine the combination of coordinates with the most amount of space between each other. This minimizes the overlap between masks in the training images, as compared to simply randomly generating the coordinates. The mask area is set to be 50% of the bounding box area.
2. **Multiple ellipse masks:** Multiple smaller masks on the image. A random number of masks between 3 to 7 are generated for each instance of the image. The total mask area is set to be 10% of the image size.
3. **Noisy blob mask:** A blob-shaped mask with noise on the image. A seperate Python package was used to generate a random blob shaped mask, with the maximum size of the blob equal the bounding box. However, due to the inpainting model leaving a green outline after filling in masks, this masking option was not used for the final training dataset.

Kindly refer to section D in the appendice for samples of each image masking option.

Final Pipeline Output

The complete pipeline outputs:

- A set of fingerprint images with controlled masking, which is used as training data for the inpainting model.
- A list of failed images (due to low classification or detection confidence), which can be reviewed for dataset refinement or model tuning.

Using the data pipeline, the team generated 2 training datasets.

- **Fingerprint Dataset Ver1:** Consists of 12,505 single ellipse masked images generated from the NIST DB4 dataset. Each input image had five masked images produced from the pipeline.
- **Fingerprint Dataset Ver2:** Consists of 27,355 single ellipse masked images and multiple ellpise masked images from the NIST DB4 dataset. Each input image also had five masked images produced from the pipeline.

4 Model Architecture

4.1 State-of-the-Art Models

Before designing the model architecture, the team conducted some primary research that yielded many promising ideas for a reliable inpainting model. The following section will delve into these identified models against the current state-of-the-art.

Stable Diffusion

Stable Diffusion models are the current state-of-the-art for generating photorealistic images.

Masked Input | Inpainted Output | Ground Truth

Figure 3: Visualisation of Output from Pre-trained Diffusion Model

The inpainted output, while hyper-realistic, does not retain the integrity of the original fingerprint data. Instead, it generates plausible but entirely synthetic patterns that may differ significantly from the true fingerprint features (compare between the Inpainted Output and Ground Truth in Figure 3). As a result, critical identification markers can be distorted or replaced, rendering the inpainted output unsuitable for any biometric identification purposes. This highlights a key limitation: although the model excels at aesthetic reconstruction, it fails to preserve forensic or identity-relevant information, which is essential in applications involving personal verification or legal evidence.

Thus, it was imperative to develop a model for this specific use case where preservation of identity-relevant markers constitutes a greater weightage than hyper-realism.

U-Net

The U-Net architecture is a kind of model that "demonstrated the ability to learn the mapping between a noisy image to a 'clean' version of that image" (Mansar, 2018). U-Net has been proven to work well on fingerprint denoising and inpainting. Unlike diffusion or GAN models, U-Net does not incorporate semantic or probabilistic guidance. As such, it cannot be easily conditioned on prompts like "restore clear ridge structures". Moreover, the quality of the dataset has a significant impact on the training network (Jing et al., 2023).

GLCIC

Another method the team examined was inpainting using Globally and Locally Consistent Image Completion (Hong & Choi, 2024). This method requires training of a generator and a discriminator. However, modal collapse is a significant challenge in Generative Adversarial Networks (GANs). GANs are hard to train in general as it requires training the discriminator and generator at the same pace, otherwise the gradients from the discriminator will not be used which leads to modal collapse. As such, there is a need for more complex or diverse GAN architectures and training strategies.

4.2 Loss Functions

The inpainting-related loss functions (from various computer vision studies) are laid out below:

4.2.1 L1 Reconstruction Loss

This loss takes the average absolute difference between the inpainted pixels and the original pixels, bounded within the mask. The formulation is:

$$\mathcal{L}_{\ell_1} = \frac{1}{\sum_{i,j} M_{i,j}} \sum_{i,j} M_{i,j} \left| \hat{I}_{i,j} - I_{i,j} \right|$$

where:

- $I \in \mathbb{R}^{H \times W \times 3}$ is the ground-truth image.
- $\hat{I} \in \mathbb{R}^{H \times W \times 3}$ is the inpainted output.
- $M \in \{0, 1\}^{H \times W}$ is the binary mask (1 inside the hole).

This simple loss function promotes following the exact pixels from the original image. However, testing showed that this tended to create very "scratchy" patterns as the model would prefer to be "cautious" when putting a pixel down, resulting in strange artifacts in the inpainted image.

4.2.2 Perceptual Loss

This loss extracts feature maps from layers of a pretrained VGG-16 network, and computes their L1 difference:

$$\begin{aligned}\phi_\ell(X) &= \text{features of image } X \text{ at VGG layer } \ell,\\ \mathcal{L}_{\text{perc}} &= \sum_{\ell\in\{1,2,3\}} \big\|\, \phi_\ell(\hat{I}\odot M) \;-\; \phi_\ell(I\odot M)\big\|_1\end{aligned}$$

Instead of comparing raw pixels, this loss compares high-level features (like edges and shapes) from a pretrained network (VGG-16). It makes the model match the look and feel of the surrounding area, not just the raw numbers from the loss calculations. This helps the filled region blend in more naturally and avoids simple blurs.

4.2.3 Texture–Matching Loss

A small encoder F produces mid-level texture features at a lower spatial resolution. We then apply an MSE over the masked area:

$$\begin{aligned}F_\ell(X) &= \text{texture features of } X \text{ at encoder stage } \ell,\\ M'_\ell &= \text{mask } M \text{ downsampled to the feature map size},\\ \mathcal{L}_{\text{tex}} &= \sum_{\ell} \big\|\, \big(F_\ell(\hat{I})\odot M'_\ell\big) \;-\; \big(F_\ell(I)\odot M'_\ell\big)\big\|_2^2.\end{aligned}$$

This loss checks that the fine patterns (such as ridge width and spacing) inside the hole matches those in the rest of the fingerprint image. It looks at mid-level texture maps (from the encoder) rather than the full images, thus enabling it to capture subtle details. By doing so, it stops the model from creating smooth but fake-looking patches.

4.2.4 Total Variation (TV) Loss

To suppress checkerboard artifacts and enforce smoothness, we penalize first-order differences:

$$\mathcal{L}_{\text{TV}} = \sum_{i,j}\Big(\big|\hat{I}_{i,j}-\hat{I}_{i+1,j}\big| \;+\; \big|\hat{I}_{i,j}-\hat{I}_{i,j+1}\big|\Big) \quad \text{over } (i,j) \text{ where } M_{i,j}=1.$$

TV loss smooths out rough spots by penalizing big jumps between neighboring pixels in the masked area. It removes speckles, checkerboard artifacts, and harsh lines that don't belong in a natural fingerprint. This makes the inpainting blend better with the surrounding ridges.

4.2.5 Total Loss

The overall training objective is a weighted sum:

$$\mathcal{L}_{\text{total}} = \lambda_1\,\mathcal{L}_{\ell_1} + \lambda_{\text{perc}}\,\mathcal{L}_{\text{perc}} + \lambda_{\text{tex}}\,\mathcal{L}_{\text{tex}} + \lambda_{\text{TV}}\,\mathcal{L}_{\text{TV}}. \tag{1}$$

Balancing these weights controls whether the model focuses more on exact pixel match (L1 Loss), natural appearance (Perceptual Loss), fine detail (Texture Loss), or smoothness (TV Loss). The weights the team used are $\lambda_1 = 0.5,\ \lambda_{\text{perc}} = 0.3,\ \lambda_{\text{tex}} = 0.5,\ \lambda_{\text{TV}} = 0.02$. during

testing as it yielded satisfactory results. With more time and resources, more exploration regarding these parameters could be done, potentially resulting in better inpainting.

4.3 Model Architecture Development

The team focused most of the development on examining multiple architectures with the same combined loss function laid out in the previous section. The rationale for doing so is to study how well different model architectures captured the fingerprint features.

Mask Handling

In Dataset 1, each original image is matched with 5 variations of a single-masked image. Dataset 2 expands this to multi-masked images as well.

Models first create binary masks using the green masks from the datasets, which is fed into the model as a 4-channel input tensor. The images are 512x512 pixels in dimensions, which proved to be computationally intensive when tested. Thus, the images are processed down to 256x256 size images. This binary mask tells the model which segments of the image are 'unknown', and the 'known' parts serve to provide the model with a global context for which the inpainting starts. Additionally, these areas mark out the area to be inpainted.

Training Hardware

GPU resources used include the school's cluster (24GB RAM), Google Colab Pro+'s high-RAM T4 GPU (15GB GPU RAM) and A100 GPU (40GB GPU RAM).

Each avenue proved to have its own limitations. The school's cluster, while free and considerably fast, was unreliable in its uptime, causing crashes when training a model. The initial Google Colab (free tier) that was used severely limited the amount of training that could be done. The team purchased Google Colab Pro+ ($72; shared with another team) with a total of 500 compute hours and access to the A100 GPU. This enabled training of much larger and complex models on a larger machine, with even a dedicated uptime guarantee where the instance could run for 24 hours in the background.

Dataset sizes proved to be a challenge, with Dataset 1 being 1.8GB and Dataset 2 being 3.4GB, causing tedious upload and unzipping process. It takes between 4-8 hours. This led the team to train each model in about **roughly 12 hours** for fairness in comparisons, by calculating the per-epoch training time and extrapolating that.

Therefore, to optimise GPU usage for faster model training, one of the models incorporates optimisation techniques, which are detailed in Implementation.

The iterations of the team's model are elaborated on below.

4.3.1 Base Model → base_model.txt

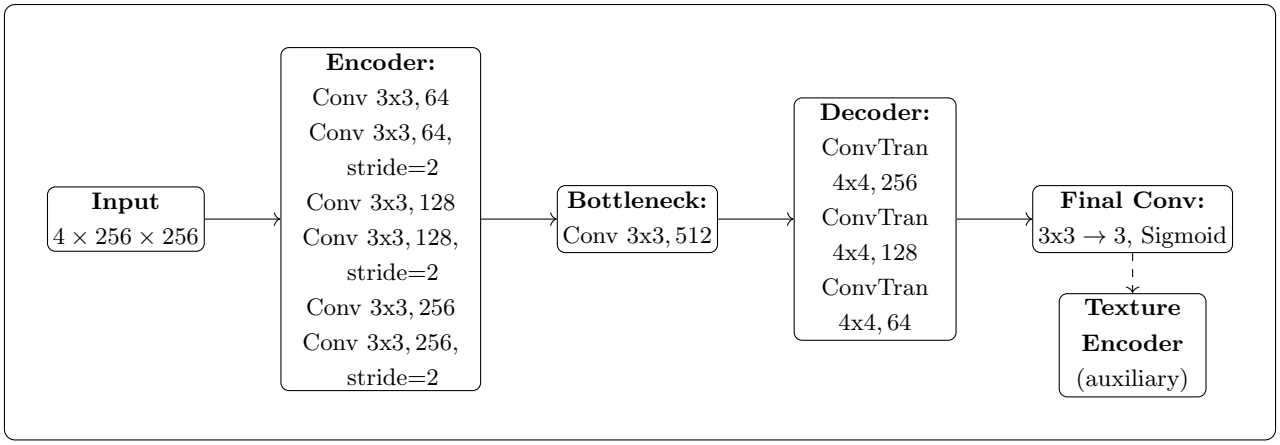


Figure 4: High-level architecture diagram of the Base Model, including the auxiliary texture encoder (used only during training) at the output stage.

The base model (Figure 4) processes a four-channel 256×256 input (three color channels plus a binary mask) and produces a three-channel inpainted patch. After inference, this patch is merged back into the original image so that only the masked region is changed, while all known pixels remain exactly as they were.

In the encoder, six 3×3 convolution layers gradually build up feature representations. The first convolution outputs 64 feature maps that capture simple edges and contrasts. The second convolution also outputs 64 maps but uses stride 2 to reduce the feature size from 256×256 to 128×128, grouping together nearby ridge textures. The next two layers increase the channel count to 128 and include another stride 2 step down to 64×64. Finally, two more convolutions raise the channels to 256 and downsample to 32×32. This sequence moves from fine-scale details toward broader fingerprint patterns.

At the bottleneck, a single 3×3 convolution with 512 output channels works on the 32×32 feature map. This stage mixes information from across the entire hole so that distant parts of the missing region can inform each other.

The decoder mirrors the encoder with three 4×4 transpose convolutions. These layers upsample the features from 32×32 back to 64×64 (256 channels), then to 128×128 (128 channels), and finally to 256×256 (64 channels). At each step the network sharpens ridge edges and refines texture details.

A final 3×3 convolution reduces the 64 channels down to 3 RGB channels, and a sigmoid activation constrains pixel values to $[0, 1]$.

The model output is blended with the original image so that unmasked pixels remain identical to the input.

In addition, a small auxiliary texture encoder (two extra convolutions) extracts mid-level texture features. During training, these features feed into a texture-matching loss that encourages the filled-in region to exhibit the same fine ridge patterns as its surroundings.

This design confirms that the training code and multi-loss setup work correctly. However, because it relies only on local convolutions without skip connections or attention, it cannot fully share detailed information across large holes or adapt its filters to irregular mask shapes.

Implementation

The script base_model.txt was run on the school cluster. The team only tested the model briefly to examine if the model was capable of generating decent inpaints in its learning process. It was trained on the the first dataset.

4.3.2 Model 1 → model1.txt

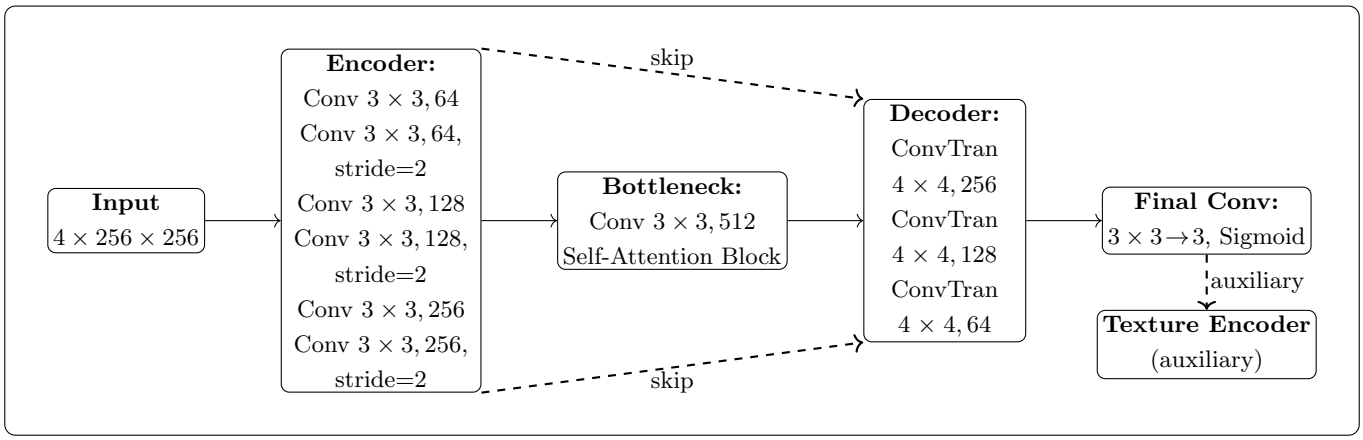


Figure 5: Model 1 adds a self-attention block after the bottleneck, carries encoder features into each upsampling step via skip-connections.

A self-attention block immediately after the bottleneck convolution allows information from distant, unmasked ridge regions to guide the filling of large holes. In addition, skip-connections shuttle the detailed edge features learned in the encoder directly into each upsampling layer of the decoder, so that fine ridge contours and local textures are preserved rather than re-derived from scratch.

These enhancements produce inpainted patches that follow the natural flow of fingerprint ridges more faithfully and maintain sharper, more realistic line detail. The limitation is that all convolutional filters are still fixed in size and orientation, so the network cannot dynamically adjust its processing to the exact shape or scale of each masked region.

Implementation

Similarly to the base model, the script model1.txt was run on the school cluster. The team only tested the model briefly to examine if the model was capable of generating decent inpaints in

its learning process. It was trained on the the first dataset. A screenshot of Model 1's training (epoch 2) is shown below to show how the model learns.

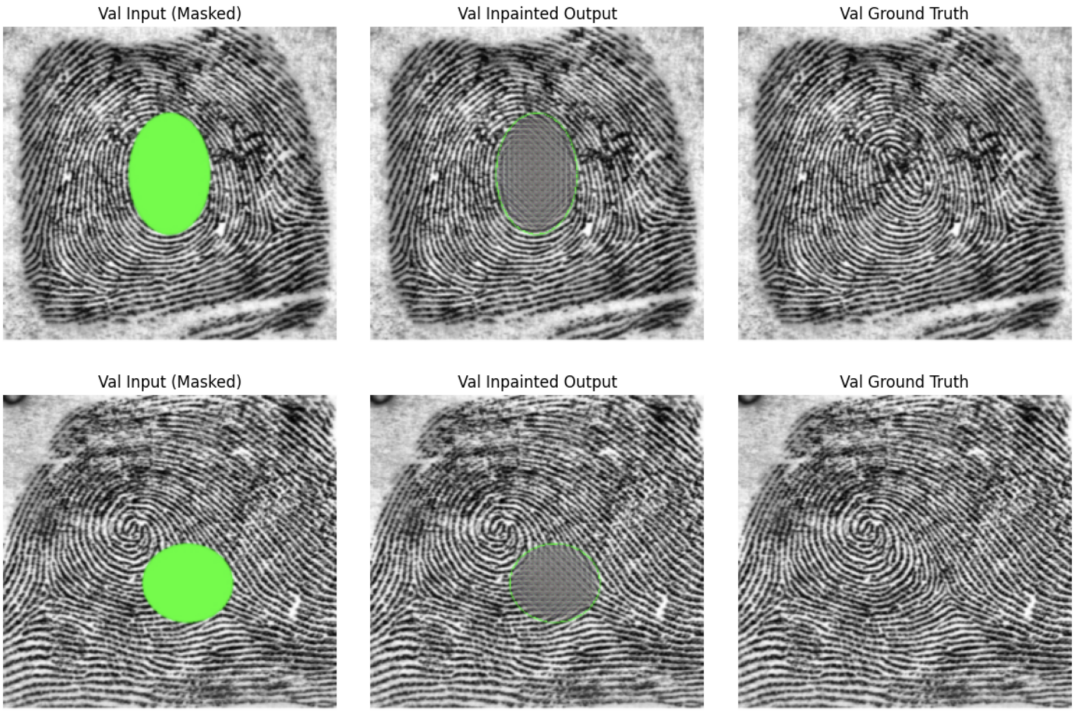


Figure 6: Early epoch Inpaint by Model 1

As seen, Model 1 learns from the "outside-in" - it gradually traces the border and builds up the patterns inside. The little grids closer to the centre follow the general shape of the underlying pattern. This was promising to look at, and the team continued to explore potentially better architectures at this juncture.

4.3.3 Model 2 → model2.txt, model2.pt, model2test.ipynb

Model 2 keeps exactly the same architecture as Model 1 but trains for more epochs with a slower learning-rate schedule and larger batch sizes. The extended training allows the network to refine subtle texture and edge artifacts through extra gradient updates. As a result, ridge connections become smoother and small inconsistencies are reduced. However, since the network itself is unchanged, it still applies the same learned filters to every mask shape and cannot generalize perfectly to mask configurations not seen during training.

Implementation

The script model2.ipynb was run on the school cluster using the first dataset. Some of its in-training screenshots are attached below.

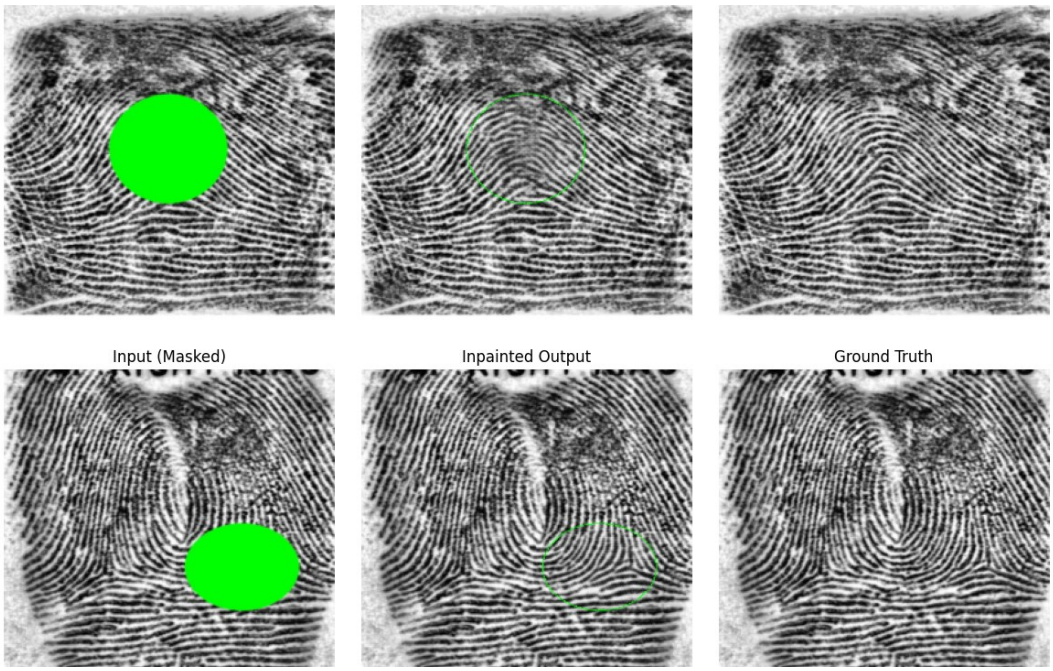


Figure 7: Sample Inpaint by Model 2

Model 2 is capable of reconstruction. However, it does not match the ground truth/original image well enough.

4.3.4 Model 2 (using improved dataset) → model2v2.ipynb, model2v2.pt, model2v2test.ipynb

Model 2v2 preserves Model 2's architecture but trains on a richer set of mask patterns (dataset 2) that combines the single ellipse masks and multiple ellipse masks and employs runtime optimizations (CuDNN autotune, shared feature caches for perceptual loss, efficient data loading). Learning on a more varied dataset teaches the model to handle a wider range of mask geometries, and the speedups keep training stable. Yet without architectural changes that directly condition on mask shape, the network still processes every hole with a mostly uniform set of filters.

Implementation

The script model2v2.ipynb was run on a T4 GPU Google Colab instance using the second dataset. Some of its in-training screenshots are attached below.

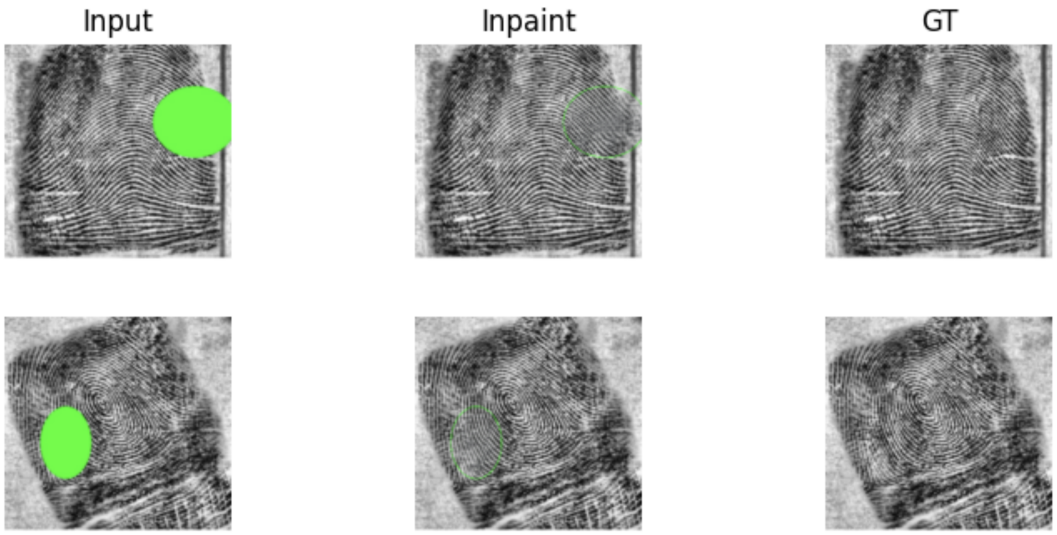


Figure 8: Sample Inpaint from Model 2 v2

Model 2 v2, like Model 2, is capable of reconstruction. While trained on a much larger and more varied dataset, it does seem to show some improvement in how well it is able to learn patterns. The pictures shown might not do it justice, thus the metrics used to measure over a test set would inform the team if the model actually improved. However, it still does not match the ground truth/original image well enough, like Model 2. Interestingly, it was at this juncture that the team found that edge cases existed, such as the ellipse mask being at the very edge of a fingerprint, which could be useful to study how well models can handle these cases.

4.3.5 Model 3 → model3.txt

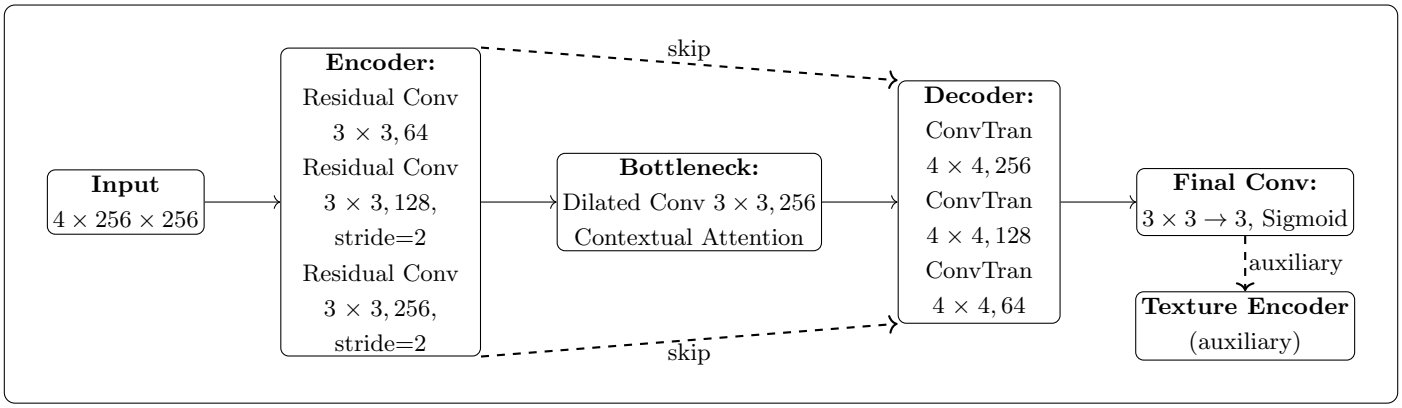


Figure 9: Model 3 uses residual blocks in the encoder, dilated convolutions with contextual attention in the bottleneck, and carries encoder features into each upsampling step via skip-connections.

Model 3 swaps each plain convolution in the encoder for a small "residual block," which adds the block's input back into its output. This shortcut helps the network learn sharper ridge

outlines without losing detail in deep layers. In the bottleneck, "dilated" convolutions spread each filter's view over a larger area without shrinking the feature maps further, so the model can see more of the hole at once. A contextual-attention module then looks around the unmasked regions for similar texture patches and blends them into the missing area, guiding the fill with real fingerprint patterns. Skip-connections still carry the encoder's fine-grained edge information straight into each upsampling stage, making sure small ridge curves survive the reconstruction. Together, these changes give cleaner, more accurate inpainting around complex mask borders—though the filters themselves still don't change based on the mask shape.

Implementation

Regrettably, Model 3 crashed on almost all attempts at training it midway through training. Thus, the team decided to train Model 4 instead as it seemed far more promising. Some of the in-training inpaints from Model 3 are shown below:

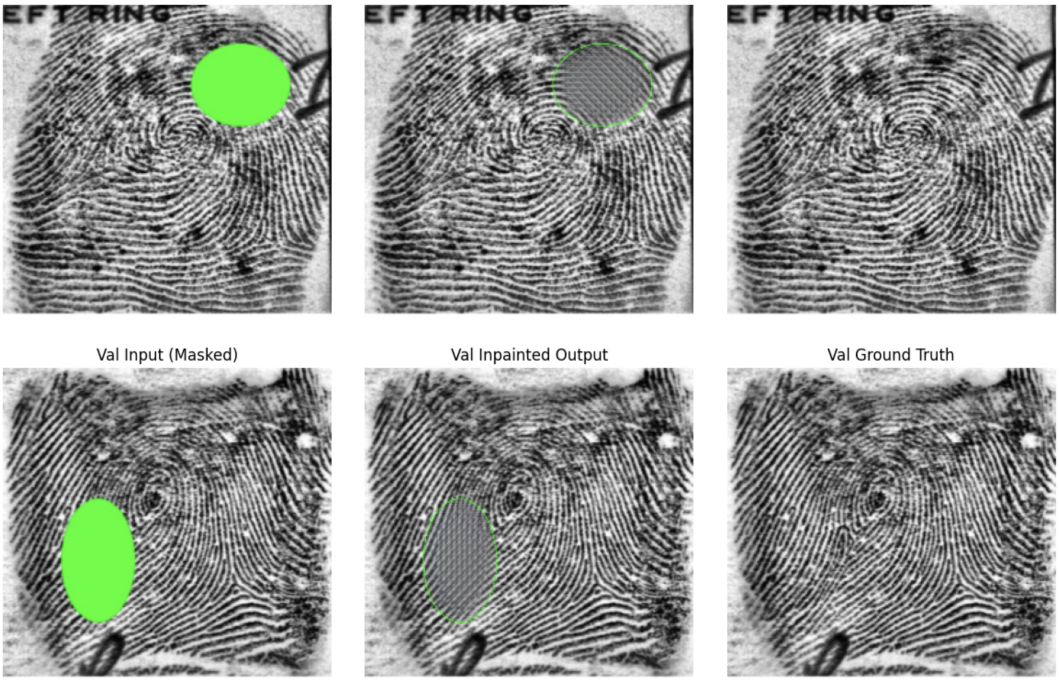


Figure 10: Sample Inpaint by Model 3 in training

Similar to the base model and Model 1, Model 3 learns in a similar way ("outside-in"), however it learns significantly faster, as the outer edges of the mask become coherent fingerprint ridges by just epoch 11, as opposed to epoch 79 in the Model 1, proving that its representational power is far greater. It was trained on the first dataset.

4.3.6 Model 4 → model4.ipynb, model4.pt, model4test.ipynb

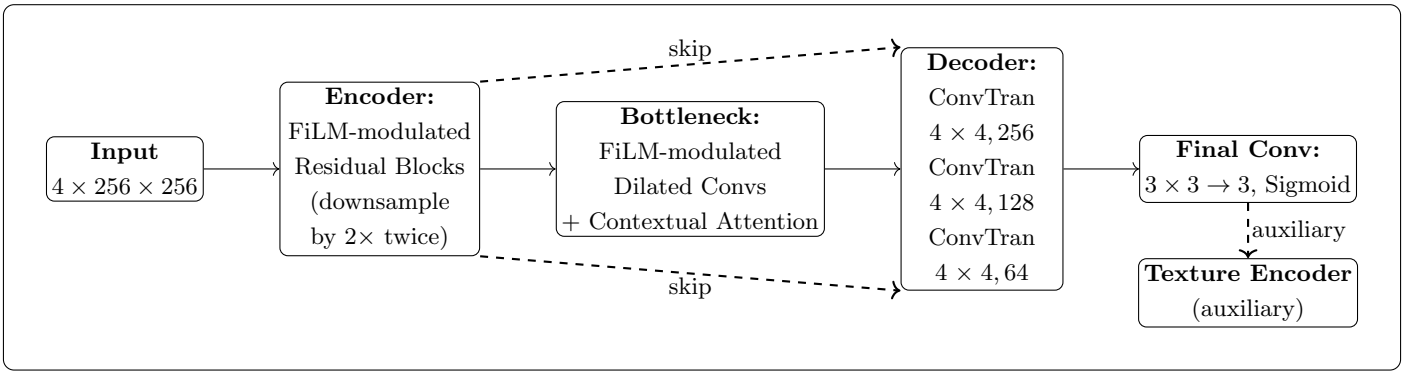


Figure 11: Model 4 adds mask-conditioned FiLM layers in every block while retaining skip-connections and auxiliary texture encoding.

Model 4 introduces "FiLM" conditioning: a lightweight MaskEncoder turns the binary mask into a small vector of scale-and-shift values, which is applied inside every residual and dilated block. By modulating feature channels based on mask position, the network learns to sharpen the transition between known and unknown regions and to adapt its filters to the hole's shape. Skip-connections still feed the encoder's crisp edges into every decoder layer, preserving fine ridges. This mask-aware gating makes each block respond differently to different mask geometries, but the single-head attention and fixed FiLM layout can still miss some long-range texture matches.

Implementation

Model 4's architecture enables it to capture context much faster and better than Model 3, as expected. At a mere 5 epochs, it was able to capture the bounds of a fingerprint (centre bottom image). Additionally, the realism and continuity of the inpaints are better as well.

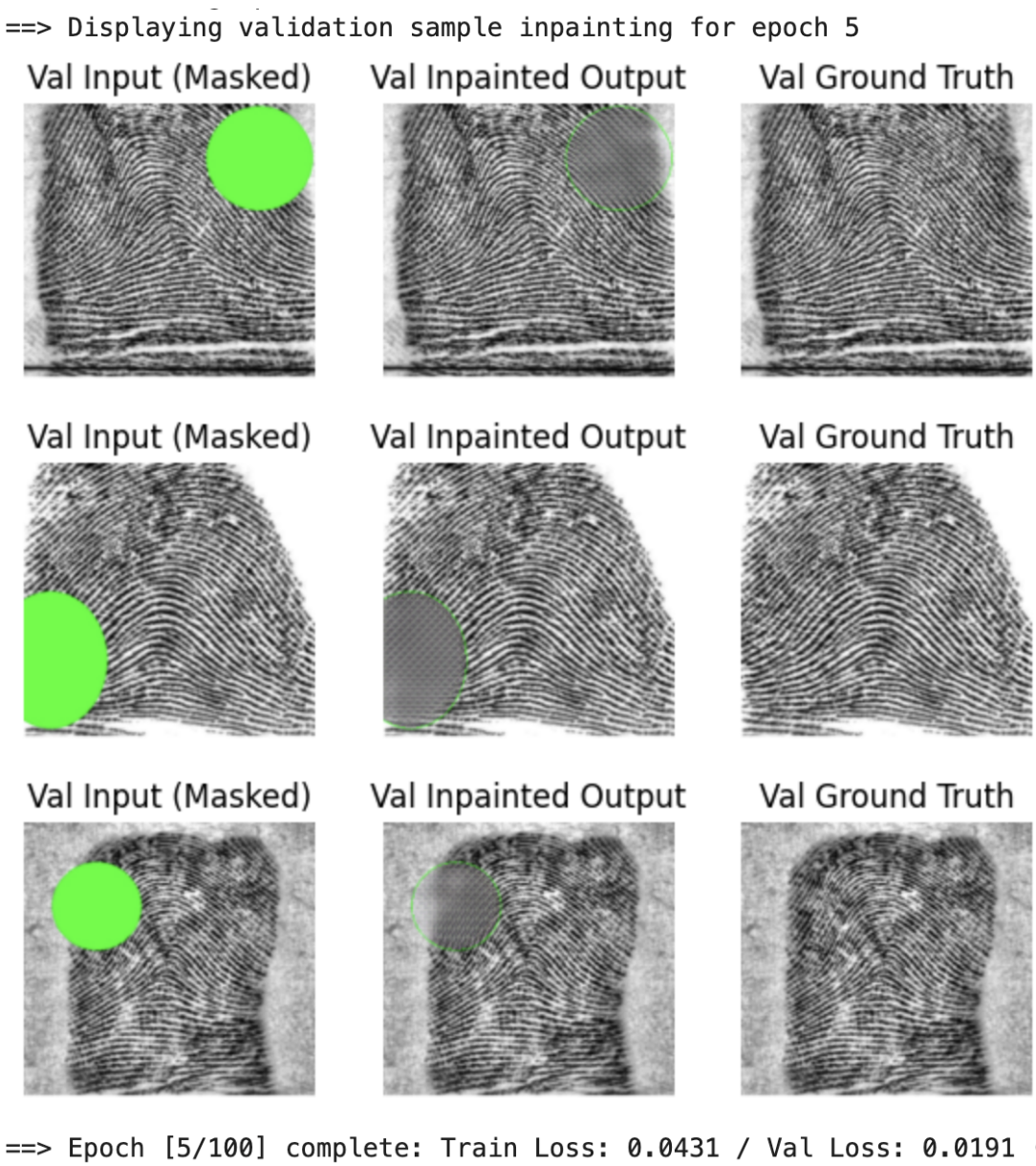


Figure 12: Inpaint during Training of Model 4

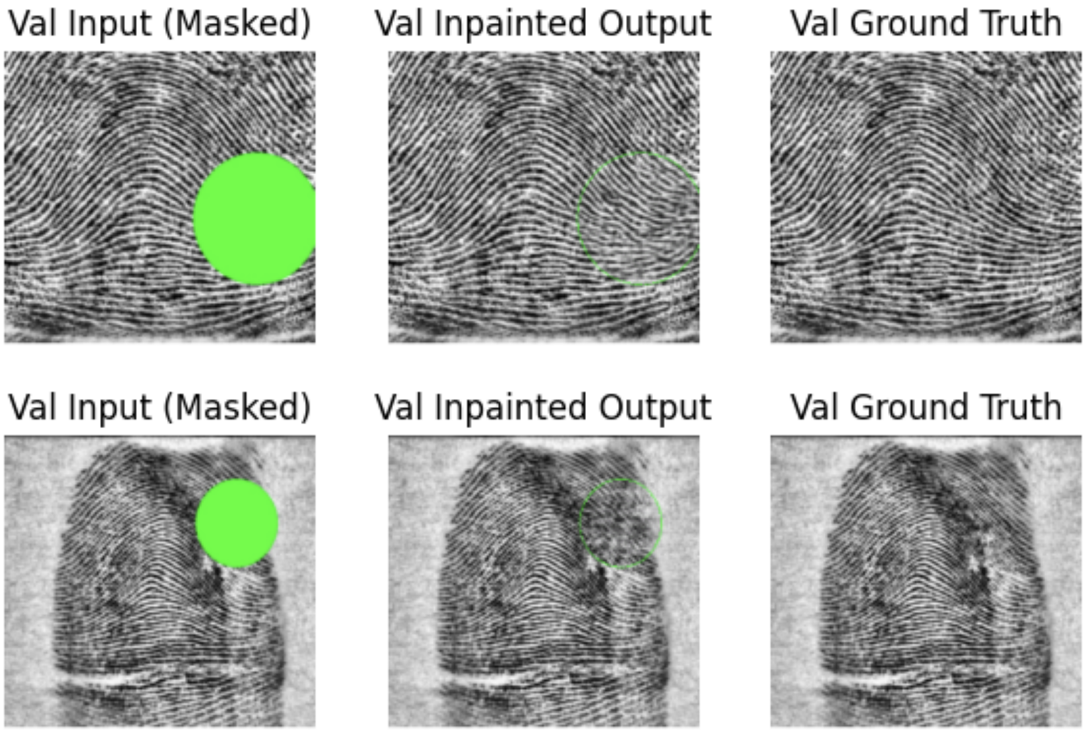


Figure 13: Validation Inpaint by Model 4

Model 4 proves itself to be a proficient inpainter, being able to capture contexts quickly. It is also able to capture subtle contexts like the variation of dark and bright spots in the fingerprints.

4.3.7 Model 5 → model5.ipynb, model5.pt, model5test.ipynb

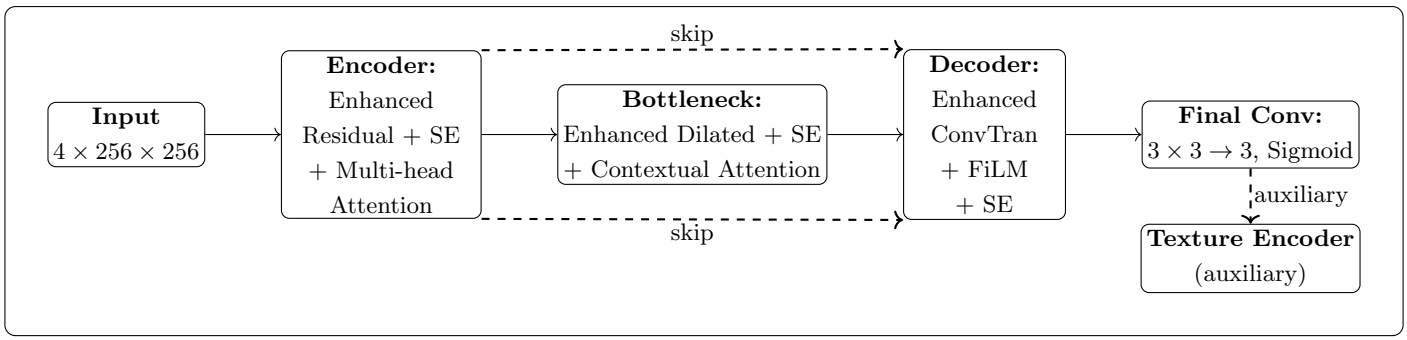


Figure 14: Model 5 integrates adaptive FiLM gating, Squeeze-Excitation, multi-head attention, enhanced contextual attention, skip-connections, and auxiliary texture encoding.

Model 5 adds Squeeze-Excitation (SE) blocks and multi-head self-attention inside the encoder and bottleneck. A SE block learns to weigh each feature channel up or down, so the network can focus on the most important ridge patterns at each layer. Multi-head attention brings several distinct "views" of global context, letting the model spot relationships between distant ridges more reliably. The bottleneck's contextual-attention is also enhanced to match patches at

multiple scales, improving copy-and-paste quality for complex textures. FiLM gating remains in every block to adapt to mask shape, and skip-connections continue to ferry fine details across the network. Together, these mechanisms let Model 5 dynamically reweight, reorient, and reuse its filters based on both mask geometry and global fingerprint structure, yielding the most faithful and detailed inpaintings to date.

Implementation

Model 5's architecture is far superior to all the other models tested thus far. In particular, it was able to capture detail and essentially skip the "outside-in" learning process entirely. Within 3 epochs, it learned a considerable amount of high-level features.

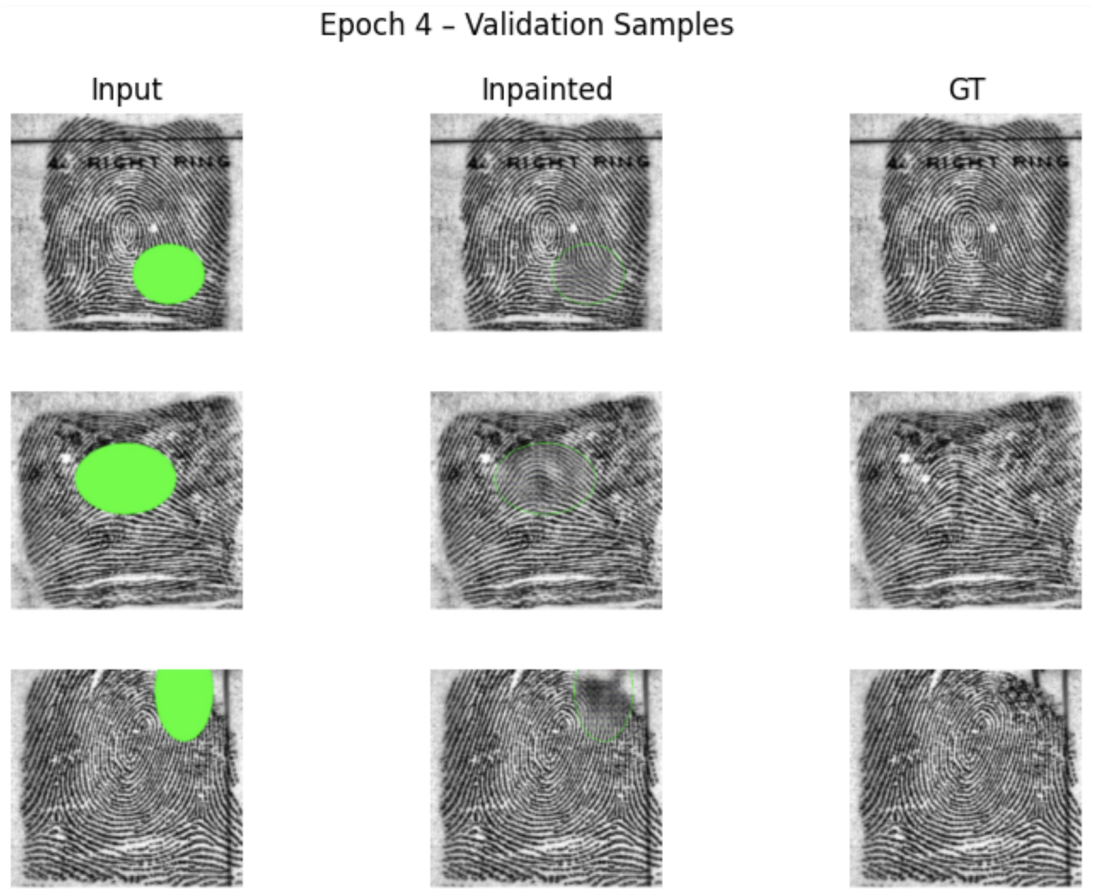


Figure 15: Early epoch Inpaint by Model 5

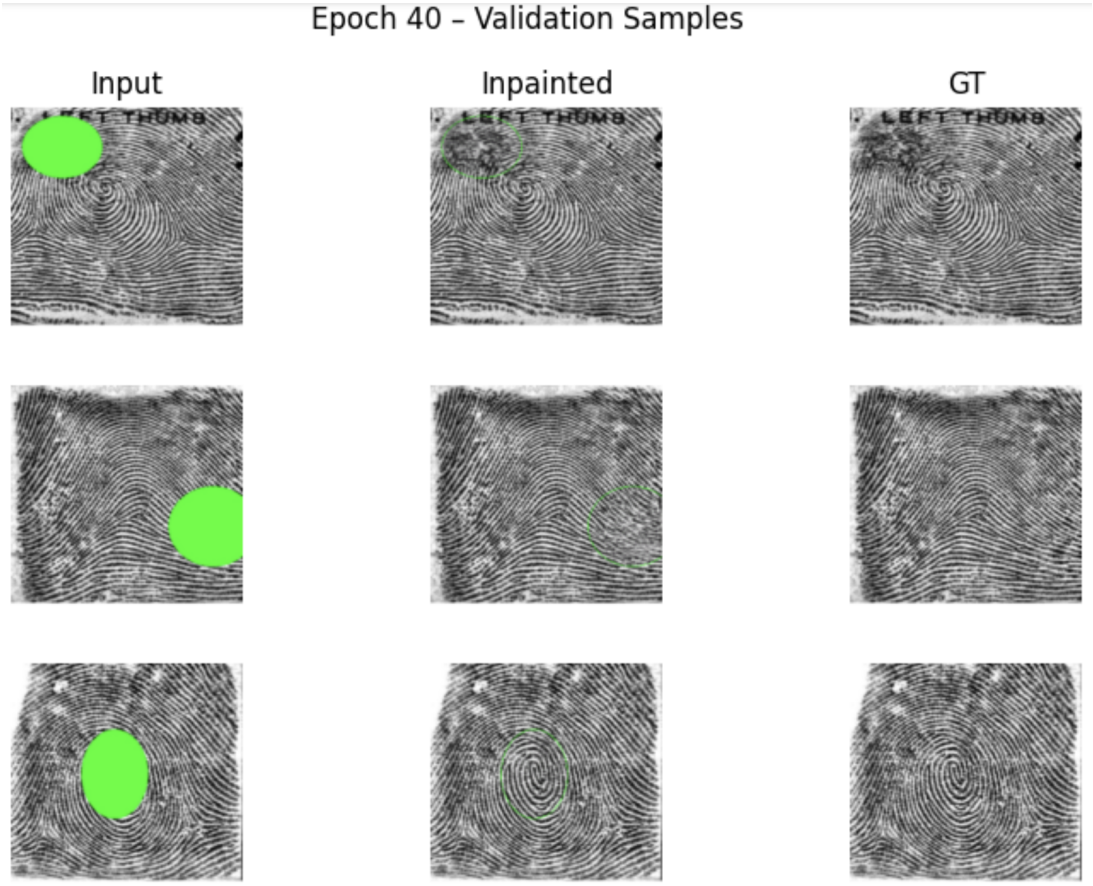


Figure 16: Later epoch Inpaint by Model 5

In Figure 16, (topmost, leftmost) Model 5 was able to infer, through the faint dark spots surrounding the mask, that the mask was covering a darker spot on the fingerprint, which is sublime performance. Regrettably, the team could only train Model 5 for 50 epochs only due to the expensive cost of training it on the A100 GPU.

4.3.8 Hyper-parameter Tuning & Evaluation of Model Training Process

As mentioned in 4.2.5, one of the parameters that could be tuned would be the combined loss function. The batch size, learning rate, weight decay as well as number of epochs could be experimented with further given more time and resources.

As it stands, the only hyperparamter tuning done was the number of epochs between model1 and model2; as well as batch sizes between model4 and model5 (even though the architecure is different). Training such models take a lot of time and troubleshooting, resources that the team lacked when developing the models - each model took roughly 12 hours to train each time, thus training a separate model with slightly different parameters was decidedly less worthwhile than training a model with new architecture with a greater promise of delivering better results. To some degree, this strategy and thought process worked out, as the results show that indeed, better architecture yielded better and better inpainting. The following section delves into the results.

5 Analysis and Discussion of Results

5.1 Evaluation Metrics

To quantify how closely the inpainted output matches the original fingerprint, four measures were recorded: mean absolute error (MAE), mean squared error (MSE), peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM).

- MAE computes the average absolute difference in pixel values between inpaint and ground truth, giving a clear sense of how far, on average, each inpainted pixel deviates from the original. Its value ranges from 0 to ∞, with 0 indicating a perfect match, so a **smaller** MAE value would indicate better model performance.
- MSE squares those differences before averaging, penalizing larger errors and reflecting the overall "amplitude" of the discrepancy. Like MAE, MSE runs from 0 to ∞; an MSE of 0 means no error, and thus a **smaller** MSE value would indicate a better model.
- PSNR converts MSE into a decibel scale, expressing the ratio between the maximum possible signal and the noise introduced by inpainting; a higher PSNR means the inpaint more closely preserves overall brightness and contrast. PSNR typically ranges from about 0 dB (very poor) up to 50 dB or more for high-quality reconstructions, so a **larger** PSNR value would indicate better model performance.
- SSIM measures perceptual similarity by comparing local patches of the two images—taking into account luminance, contrast and structure—and yields a score between 0 and 1. A score of 1 indicates perfect structural match, so a **larger** SSIM value would indicate better model performance.

In the implementation, both ground-truth and inpainted images are first normalised to $[0, 1]$ and resized to identical dimensions. For SSIM, a sliding window was used up to 7×7 (library default if the image is smaller) so that local ridge-texture consistency is properly assessed. The measured values and side-by-side images would be compared.

The comparison with **state-of-the-art** was done in the same way.

5.2 Results of Models & Comparison with State-of-the-Art

A consistent test set that contains an equal mix of each pattern (loop, whorl, arch, imploding whorl) masked in both ways (single mask & multi-mask) was used to calculate the average values for each metric outlined above. The results and analysis are shown below.

Model	MAE	MSE	PSNR (dB)	SSIM
model2	0.0601	0.0091	20.6478	0.8535
model2v2	0.0604	0.0088	20.6322	0.8529
model4	0.0611	0.0099	20.3667	0.8499
model5	0.0556	0.0069	21.7667	0.8755
State of the art				
Diffusion (un-tuned)	0.1009	0.0243	16.2351	0.6735
U-Net (un-tuned)	0.1357	0.0420	24.6811	0.8314

Table 1: Average evaluation metrics with per-column shading (green=best, red=worst).

- **Model 2 vs. Model 2 v2:** These two share the same architecture but differ only in their training data. Their metrics are nearly identical (MAE 0.0601→0.0604; PSNR 20.65dB→20.63dB), indicating that the dataset variation in Model 2 v2 had only a marginal effect on inpainting quality.
- **Progression from Model 2 to Model 4 to Model 5:** Moving from Model 2 to Model 4 causes a slight drop in quality (MAE ↑0.0601→0.0611; PSNR ↓20.65dB→20.37dB), whereas the jump from Model 4 to Model 5 delivers a substantial improvement (MAE ↓0.0611→0.0556; PSNR ↑20.37dB→21.77dB). Thus, the architectural and loss-function enhancements in Model 5 yielded far greater gains than those between Models 2 and 4.
- **Diffusion baseline:** The off-the-shelf diffusion approach (MAE 0.1009; PSNR 16.24dB; SSIM 0.6735) lags well behind all U-Net-based models. Without task-specific fine-tuning, it underperforms even the original Model 2, showing the value of end-to-end training on fingerprint inpainting.
- **Overall:** Model 5 sets the mark on this fingerprint test suite. Model 2 and Model 2 v2 perform almost identically, Model 4 sits in the middle, and the diffusion baseline remains the weakest.

5.2.1 Comparing by Fingerprint Pattern

The team picked out a random test suite of images that displayed the various possible scenarios that the models can encounter, and a comparison on the results of how different models inpainted them.

In single-mask inpainting, Model 2 and Model 2 v2 performed badly, perhaps due to their model architectures being unable to capture long-range dependencies. Sometimes, they even fail to properly "fill up" the inpaints, leaving artifacts in the centre. Model 4 and 5 were capable of generating full inpaints, with Model 5 generating a noticeably higher quality inpaint.

In multi-mask inpainting, Model 2 performed worse than Model 2 v2. Surprisingly, Model 4 performed worse than Model 2 on some of these tasks. Model 5 unsurprisingly was capable of creating realistic and accurate inpaints.

Arch

Single-masked Task

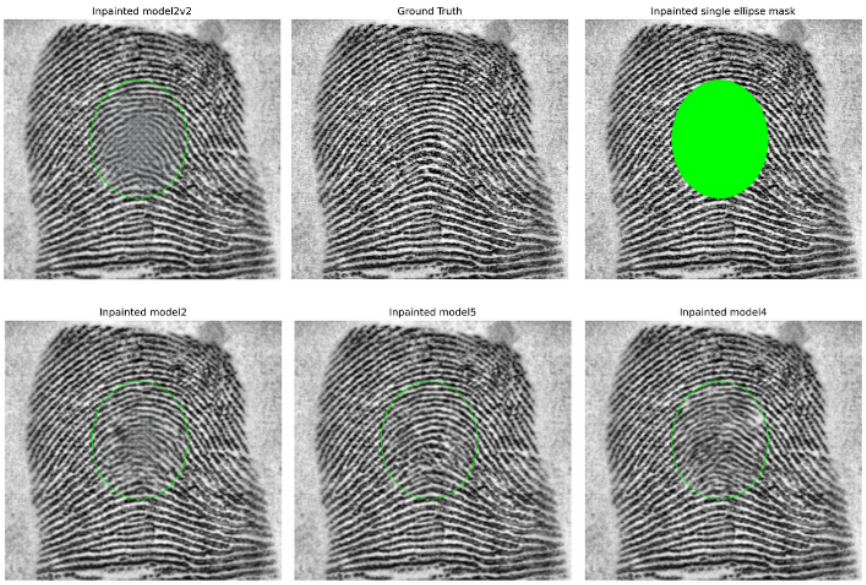


Figure 17: Single-masked arc inpainting

Multi-masked Task

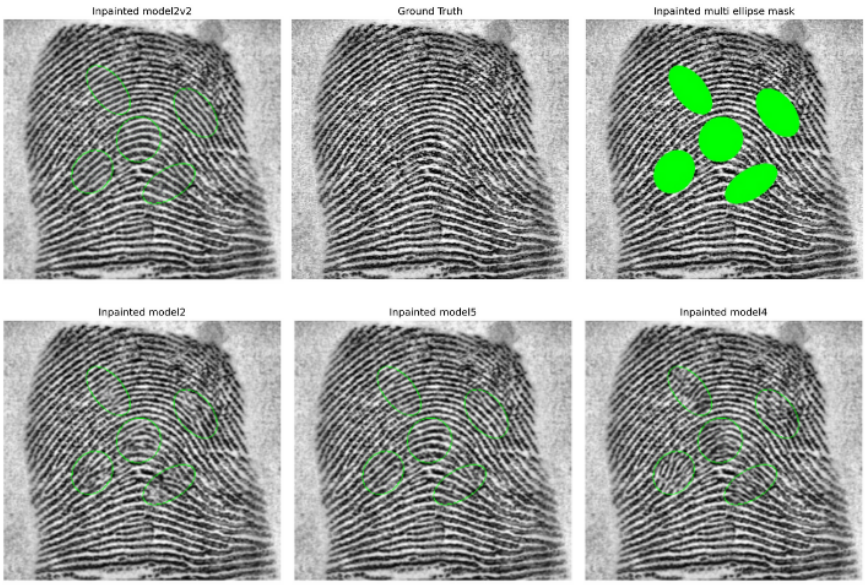


Figure 18: Multi-masked arc inpainting

Loop

Single-masked Task

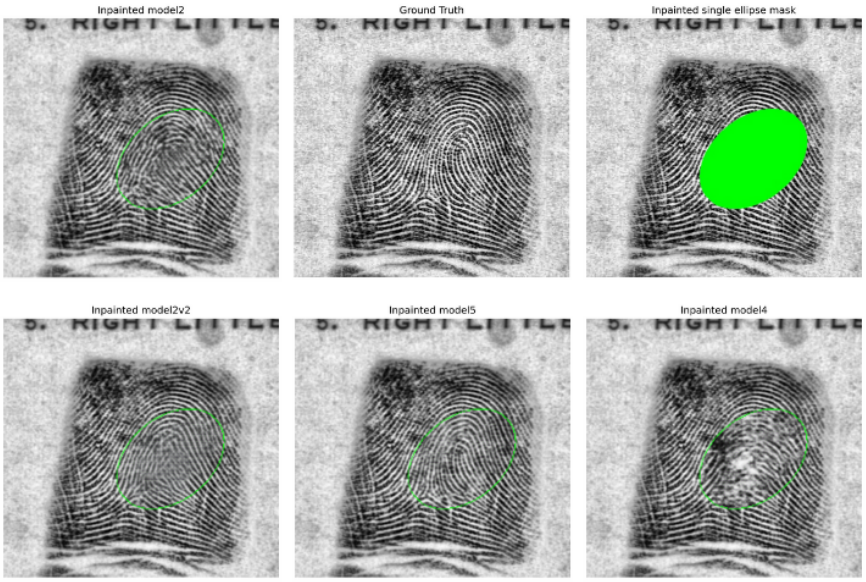


Figure 19: Single-masked loop inpainting

Multi-masked Task

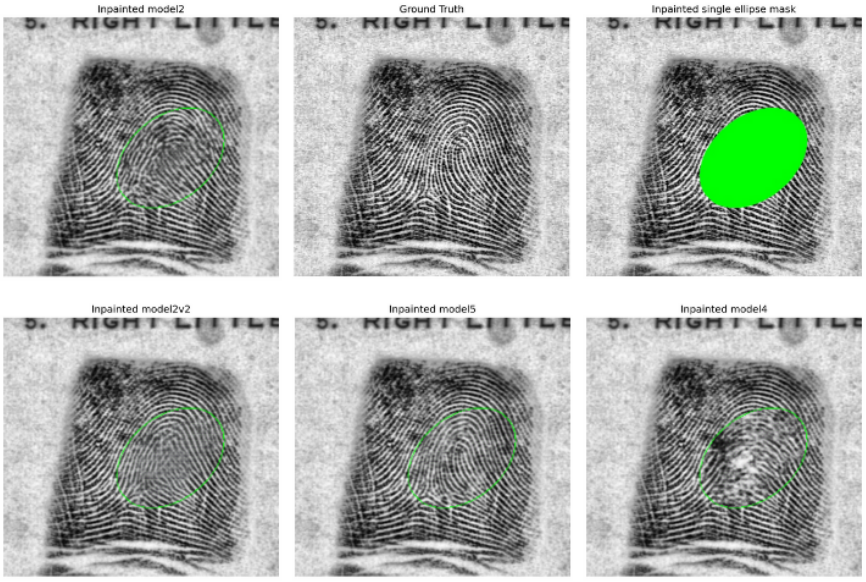


Figure 20: Multi-masked loop inpainting

Whorl

Single-masked Task

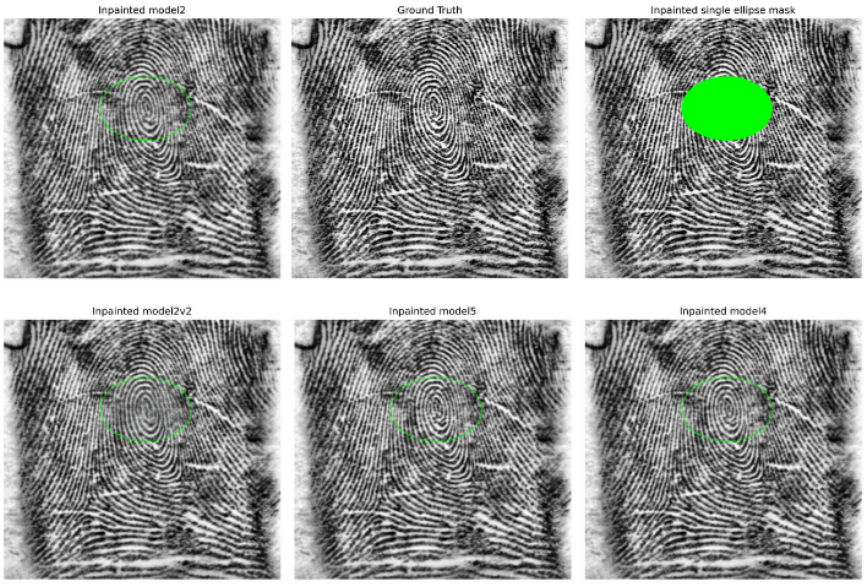


Figure 21: Single-masked concentric whorl inpainting

Multi-masked Task

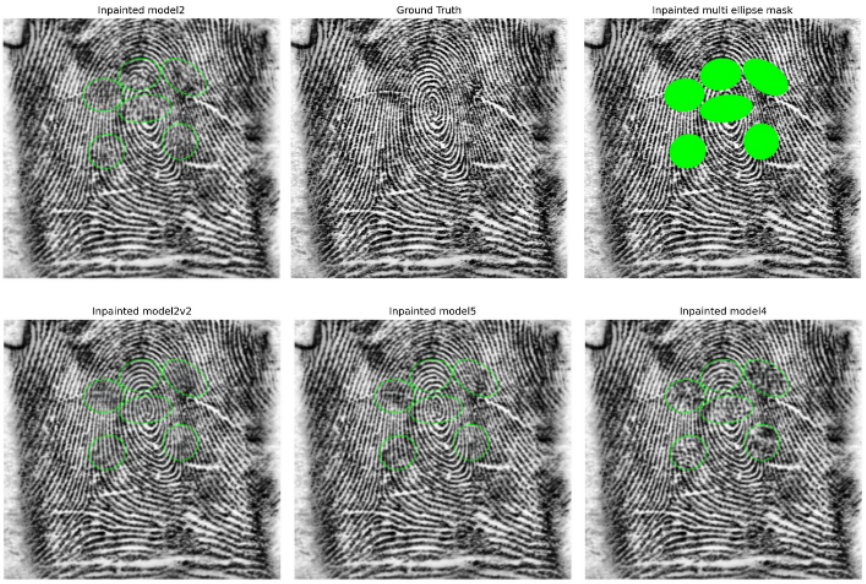


Figure 22: Multi-masked concentric whorl inpainting

Imploding Whorl

Single-masked Task

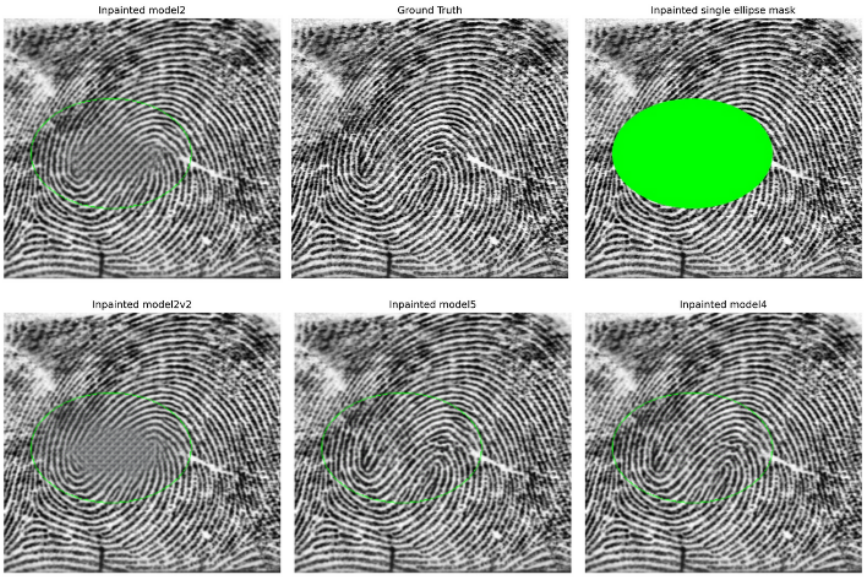


Figure 23: Single-masked imploding whorl inpainting

Multi-masked Task

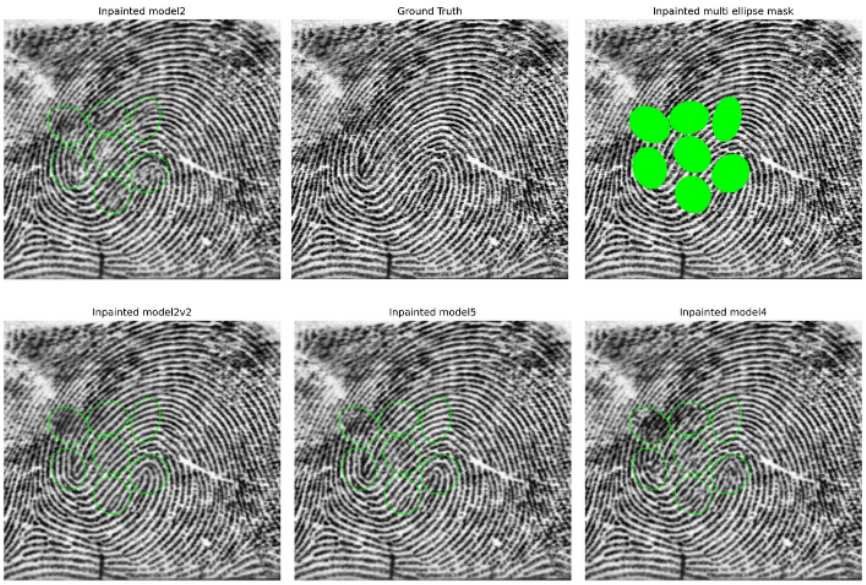


Figure 24: Multi-masked imploding whorl inpainting

5.2.2 Comparing by Edge Cases

Edge cases refer to the anomalies in the dataset and can be useful to understanding how well the model responds to masks in unpredictable areas.

Edges

In edge case 1, the ellipse is placed near the boundary. The models are expected to be able to detect this and inpaint accordingly. The performance of the models in this case is surprising, with Model 2 performing the best at detecting the edge, followed by Model 2 v2, Model 4 and Model 5 in terms of how clean the boundary was reconstructed.

Edge Case 1 Task

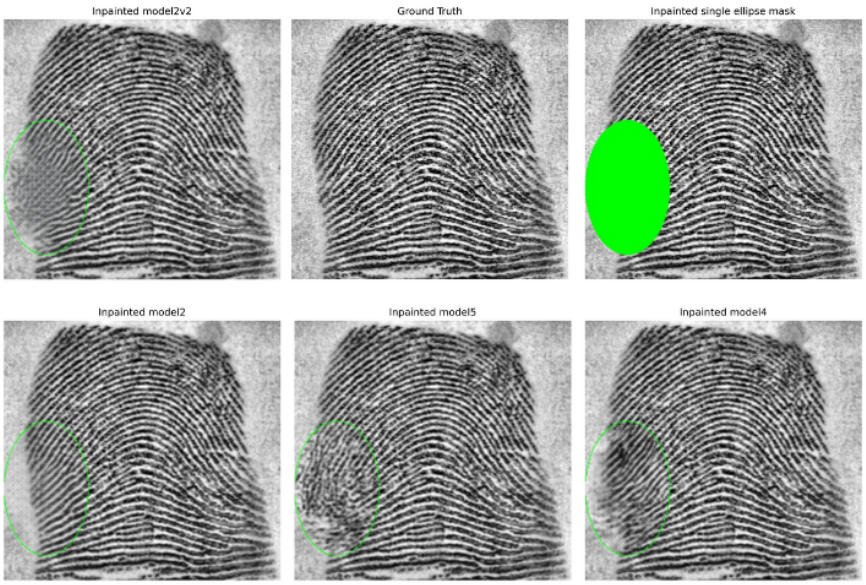


Figure 25: Edge case 1 inpainting

Edge Case 2 Task

In edge case 2, the ellipse is placed near the boundary, but less than in Edge case 1. Similarly, the models are expected to be able to detect this and inpaint accordingly. The performance of the models in this case are relatively similar, as all of them are capable of producing a clean boundary; however the same order of "clean-ness" is observed here as well, with Model 2 having the cleanest boundary and Model 5 the messiest.

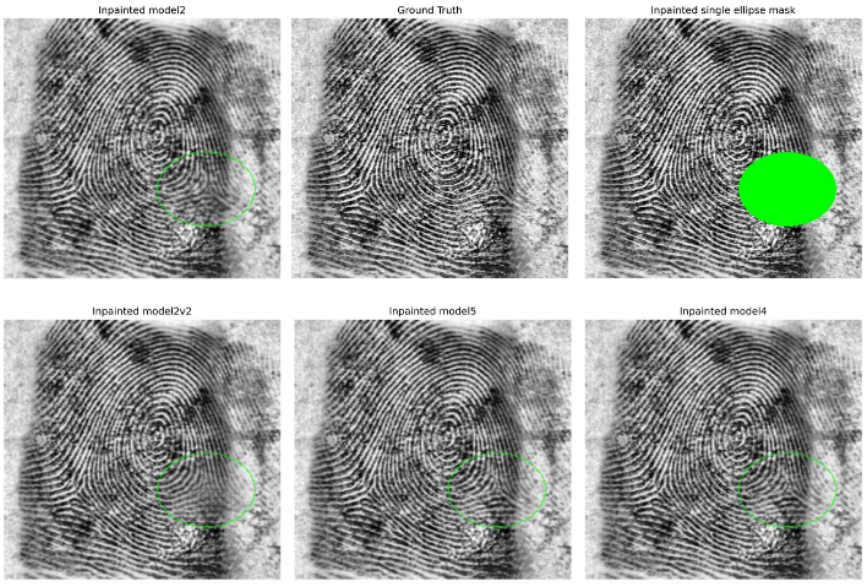


Figure 26: Edge case 2 inpainting

Text Block Task

Another unexpected piece of data in the images are text. These texts label the fingerprint's feature, e.g right ring. These are sparse but not particularly uncommon. This edge case has an ellipse covering part of the text. Surprisingly, Model 2 and 5 were able to produce the clean text followed by Model 2 v2 and Model 4.

Text Block Task

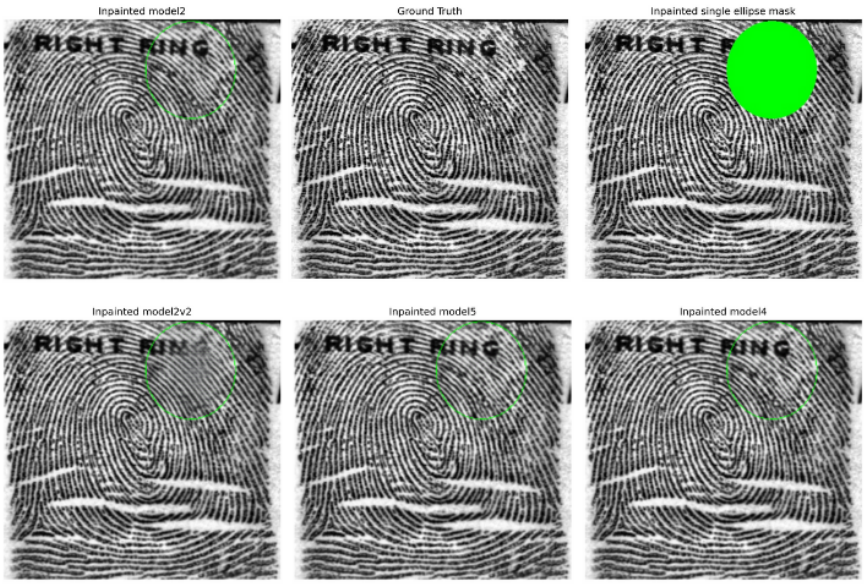


Figure 27: Text block inpatining

6 Conclusion

Model 5 demonstrates state-of-the-art performance across all evaluation metrics, achieving significant improvements over traditional architectures and un-tuned diffusion models. Structurally, it employs an encoder-decoder backbone enhanced with multiple architectural innovations. The inclusion of a self-attention block enables the model to learn spatial context and capture geometric relationships intrinsic to fingerprint patterns. Residual blocks replace standard convolutional layers to facilitate deeper learning and improved gradient flow, while Squeeze-and-Excitation (SE) modules adaptively recalibrate channel-wise feature responses. Additionally, Feature-wise Linear Modulation (FiLM) is integrated to condition the model dynamically, allowing for better generalization across diverse input patterns.

Training-wise, Model 5 benefits from a longer optimization horizon, with scheduled learning-rate reductions, larger batch sizes, and runtime optimizations tailored to stabilize convergence. The model is also trained on an enhanced dataset with greater coverage of fingerprint variations, contributing to its robust generalization.

Empirically, Model 5 achieves the lowest MAE (0.0556) and MSE (0.0069), while attaining the highest PSNR (21.77 dB) and SSIM (0.8755) among all tested models. These results reflect superior pixel-wise accuracy, perceptual fidelity, and structural coherence. Collectively, the design and training choices of Model 5 lead to a well-regularized and high-performing architecture that significantly advances the quality of fingerprint inpainting.

6.1 Impact on SDG Goals

This project is believed to significantly aid in Goal 16 of the Sustainable Development Goals which seeks to "promote peaceful and inclusive societies for sustainable development, provide access to justice for all and build effective, accountable and inclusive institutions at all levels". Better fingerprint inpainting solutions help law enforcement solve crimes using partial or smudged prints, enhancing justice delivery.

Firstly, it can prevent wrongful convictions due to ambiguity of fingerprint matches to low-quality latent fingerprints that are found at crime scenes. Secondly, it can reduce the number of cold cases where latent prints from crime scenes are often partial or smudged due to surface type or environmental degradation. This often leads to the dismissal of evidence in court.
Thirdly, it can circumvent the discharge of guilty parties due to inconclusive evidence. In some cases, suspects might have self-mutilated prints, leading to insufficient forensic certainty without reliable inpainting which can partially reconstruct and potentially link the suspect.

The failure to reconstruct damaged fingerprints impedes both prosecution and exoneration. With better fingerprint inpainting, there is increased trust in biometrics forensics and enhances efficiency in the justice system.

6.2 Future Recommendations

Future improvements should incorporate domain-specific constraints, especially around ridge flow and minutiae structure. Tu et al., 2020's research into mathematical algorithm that restores fragmented ridges in fingerprints using cubic Bezier curve. Incorporating orientation field and minuatae preservation constraints can avoid hallucination, increasing the accuracy of the fingerprint restoration.

In addition, model can be significantly optimised to result in better model training and testing. Increasing the efficiency of training helps in validating different models quicker, developing more robust models.

Appendix

A Setup of Code

Data Pipeline Setup Instructions

1. Create a Python virtual environment:

   ```
   python -m venv venv
   ```

2. Activate the Python virtual environment:

   ```
   source venv/bin/activate
   ```

3. Navigate to the Fingerprint Classifier Directory:

   ```
   cd "Fingerprint Classifier"
   ```

4. Install all required dependencies from requirements.txt:

   ```
   pip install -r requirements.txt
   ```

5. Create a .env file using the .env.example as the template and obtain a Roboflow key from Roboflow website.

6. Run the data pipeline:

   ```
   python pipeline.py
   ```

Data Model Setup Instructions

To load the models and tests, load up either the school cluster or a Google Colab notebook (depending on the code). The dataset (either fingerprint2 or fingerprint3) would need to be uploaded to the same directory as the model codes.

1. Start up AI-GPU cluster or Google Colab Notebook :

2. Upload zipped dataset to the same directory as the model code. Unzip the dataset.

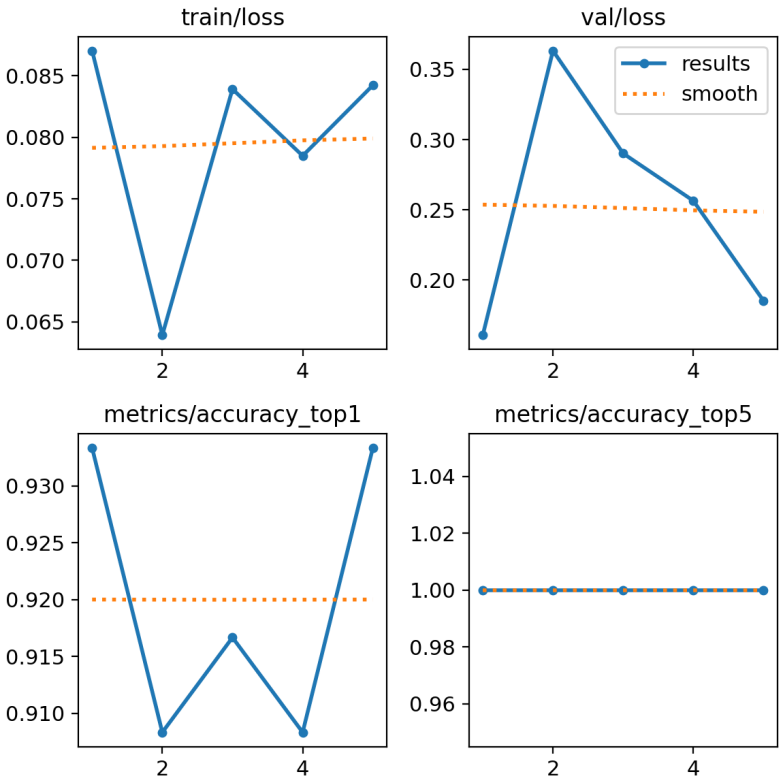
   ```
   python -m venv venv
   ```

3. Set data directories path

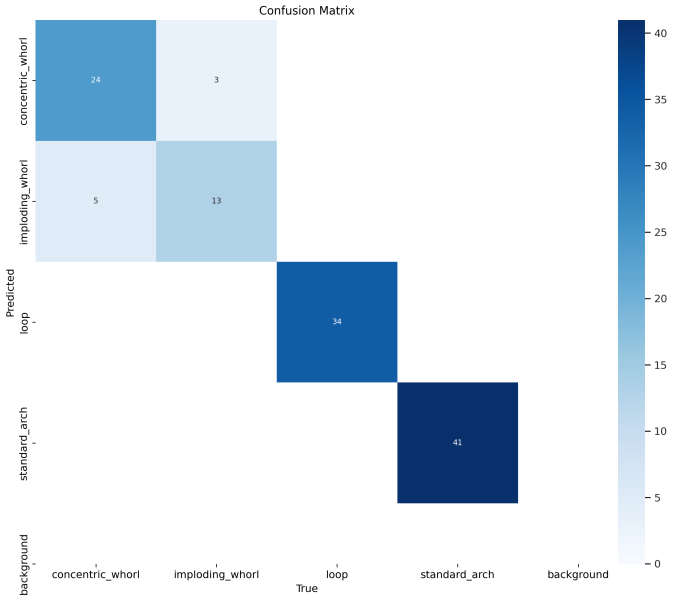
4. Run the cell in the notebook

B Hyperparameter Tuning

B.1 Hyperparameter tuning results for fingerprint classifier model

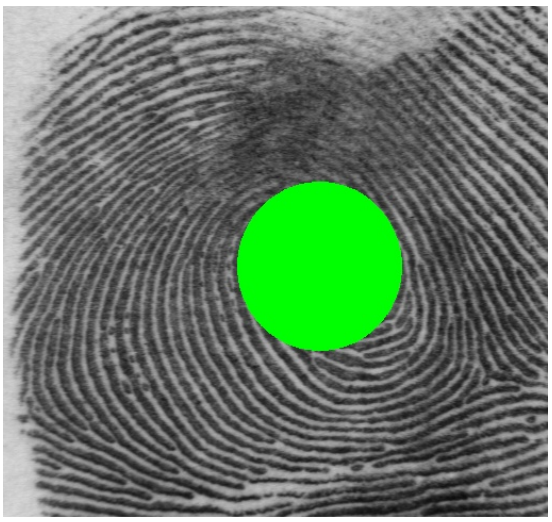


B.2 Confusion matrix for fingerprint classifier model

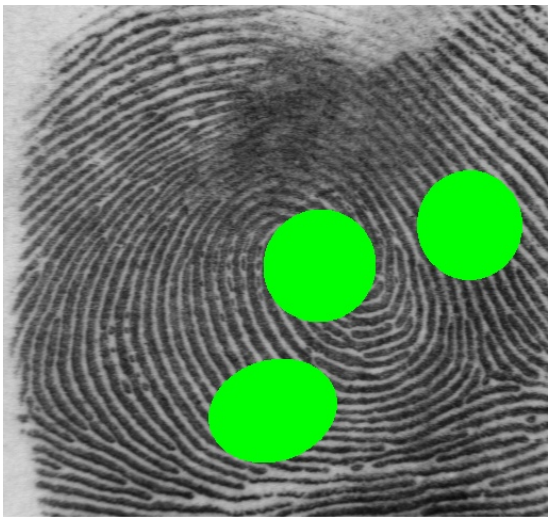


C Image Masking

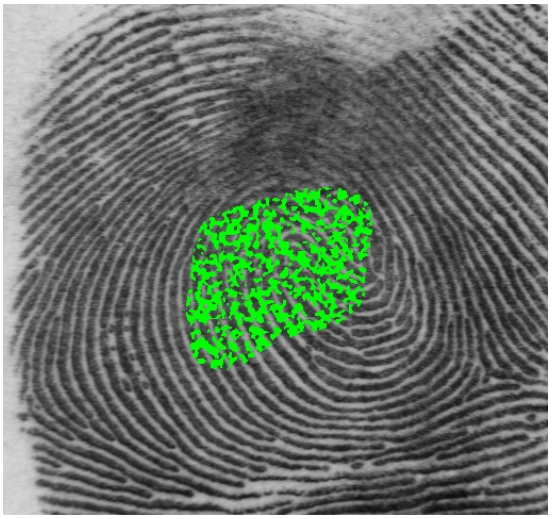
C.1 Image Masking with single ellipse mask



C.2 Image Masking with multiple ellipse masks



C.3 Image Masking with blob-shaped noisy mask



D Inpainting Model Output

D.1 Results from Models

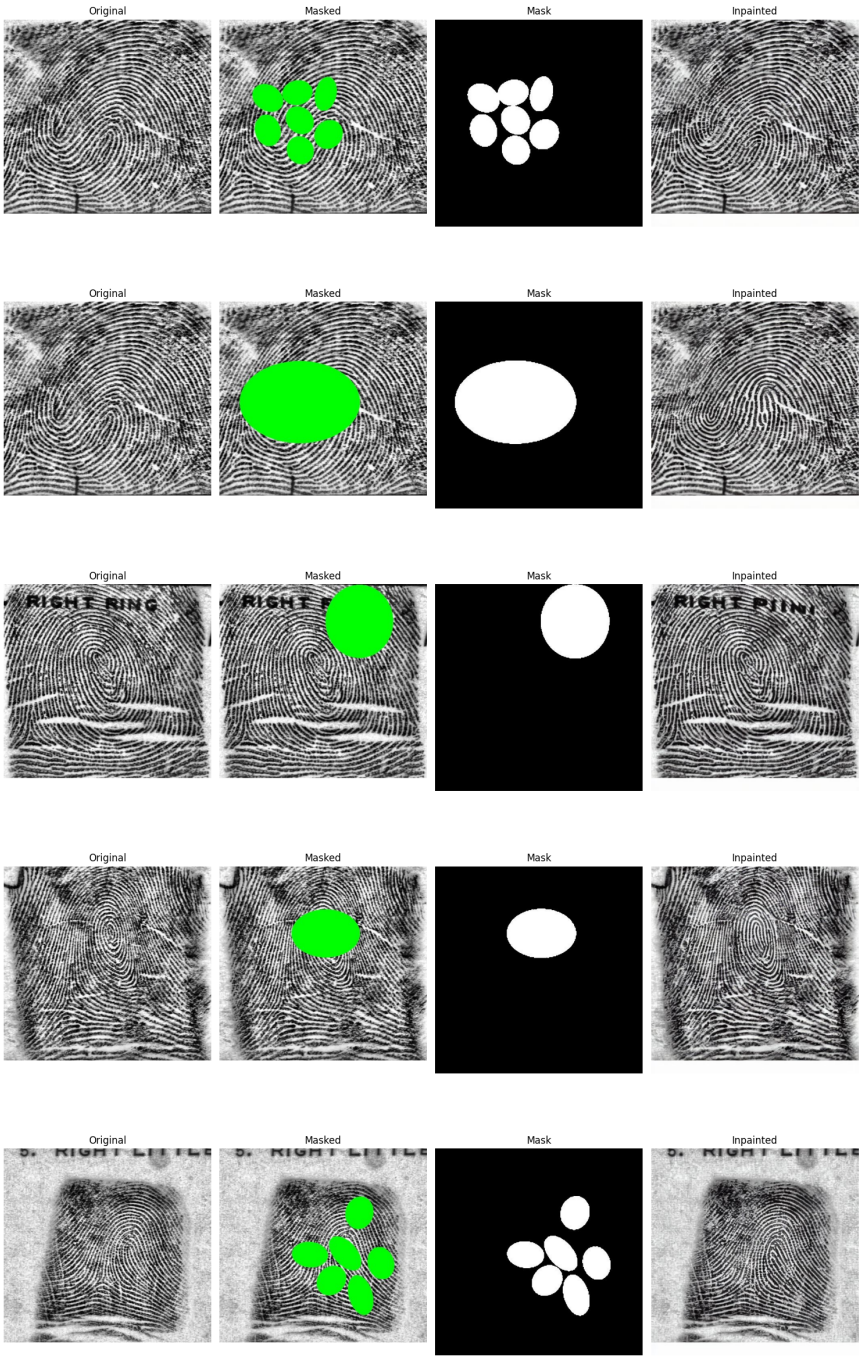


Figure 28: Comparison Grid of 5 Samples Inpainted using Stable Diffusion

E References

References

Çelik, M. (2023). Brief summary of datasets available for image inpainting. *Medium*. https://muratclk.medium.com/brief-summary-of-datasets-available-for-computer-vision-dc47b0c18342

Hong, S., & Choi, K. H. (2024). Correcting faulty road maps by image inpainting. *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2710–2714. https://doi.org/10.1109/ICASSP48485.2024.10446744

Inpainting. (n.d.). Using diffusers for inpainting.

Jing, Y., Li, C., Du, T., Jiang, T., Sun, H., Yang, J., Shi, L., Gao, M., Grzegorzek, M., & Li, X. (2023). A comprehensive survey of intestine histopathological image analysis using machine vision approaches. *Computers in Biology and Medicine*. https://doi.org/10.1016/j.compbiomed.2023.107388

Mansar, Y. (2018). Fingerprint denoising and inpainting using fully convolutional networks. *Medium*. https://medium.com/data-science/fingerprint-denoising-and-inpainting-using-fully-convolutional-networks-e24714c3233

Tu, Y., Yao, Z., Xu, J., Liu, Y., & Zhang, Z. (2020). Fingerprint restoration using cubic bezier curve. *Biomed Central*. https://doi.org/10.1186/s12859-020-03857-z