Photo Reconstruction Project Report

Sapir Dahan Noam Levin

January 2025

Abstract

This report presents our work on photo reconstruction using a dataset specifically designed to simulate missing parts of images. The project explores multiple approaches, including a baseline method, linear regression, a basic neural network, and an advanced Attention Model. Each method is evaluated based on its performance, with results demonstrated through both quantitative metrics and visual reconstructions. Key challenges encountered during the project, along with initial attempts and subsequent improvements, are discussed. The complete project code is available on GitHub at github.com/SapirDahan/PhotoReconstruction.

1 Introduction

1.1 Background and Motivation

The goal of this project is to reconstruct missing regions in images using machine learning techniques. The reconstructed images are compared with the original ground-truth images using quantitative metrics, such as Mean Squared Error (MSE) and Mean Absolute Error (MAE), as well as visual examples.

1.2 Dataset and Data Preparation

An image tensor is essentially a collection of numerical values where each number corresponds to a pixel in the image. These pixel values capture specific features of the image, such as intensity and color. These numerical features collectively describe the visual content of the image in a form that can be processed by a computer, enabling models to analyze and learn patterns from the image data.

The dataset for this project was curated using 10 selected classes from the ImageNet dataset. The ImageNet dataset has been retrieved from the Hugging-face repository. The 10 selected classes include the following animals: butterfly, panda, parrot, pomeranian, goldfish, elephant, monkey, persian cat, penguin and red panda. Each image was resized to a resolution of 224×224 pixels, and blacked-out regions were simulated by masking rectangular regions of random sizes ranging from 30 to 60 pixels for each dimension. The positions of the blacked-out rectangles were randomized as well. These masked regions mimic missing data, which the models aim to reconstruct. The dataset was organized into three subsets: 70% of the images were allocated for training, 10% for validation, and 20% for testing.

1.3 Models and Objectives

This project explores a variety of models to address the image reconstruction task, starting with simple methods and advancing to more complex architectures. A baseline model allocates random pixel values for the masked regions, providing the simplest reference point. Any other method should outperform this baseline. Linear regression builds upon this by incorporating surrounding pixel features to improve predictions. A basic neural network is then introduced to learn complex patterns and relationships within the data. Finally, an advanced Attention Model leverages spatial and channel-wise attention mechanisms to dynamically focus on the most critical regions of the image, leading to enhanced reconstruction quality.

The primary objectives of this project are to evaluate the effectiveness of these models in reconstructing masked image regions, compare their performance through both quantitative metrics and visual results, and address the challenges inherent in reconstructing diverse visual patterns. These findings aim to provide information on the strengths and limitations of each approach.

2 Models

This section describes the models used for photo reconstruction, ranging from a simple baseline model to an advanced Attention-based Model. This document does not provide code examples, as they are quite long and can be easily accessed and understood from the provided GitHub repository. The code is written in Jupyter Notebooks and includes comprehensive documentation.

2.1 Baseline Model

The baseline model reconstructs the image by providing each pixel in the masked region with a random RGB value. This simple approach ensures that the masked regions are visually distinct while providing a baseline to evaluate the performance of more advanced models.

2.1.1 Results

The results of the baseline model on the test dataset are presented in Table 1, including the Mean Squared Error (MSE) and Mean Absolute Error (MAE) metrics.

Metric	**Test Value**
Mean Squared Error (MSE)	1139.2030
Mean Absolute Error (MAE)	7.0292

Table 1: Performance metrics for the baseline model.

2.1.2 Visualization

Although the baseline model provides a straightforward implementation, it is incapable of capturing the underlying spatial or semantic patterns in the masked regions. Figure 1 illustrates the visual results produced by the baseline model on example test images.

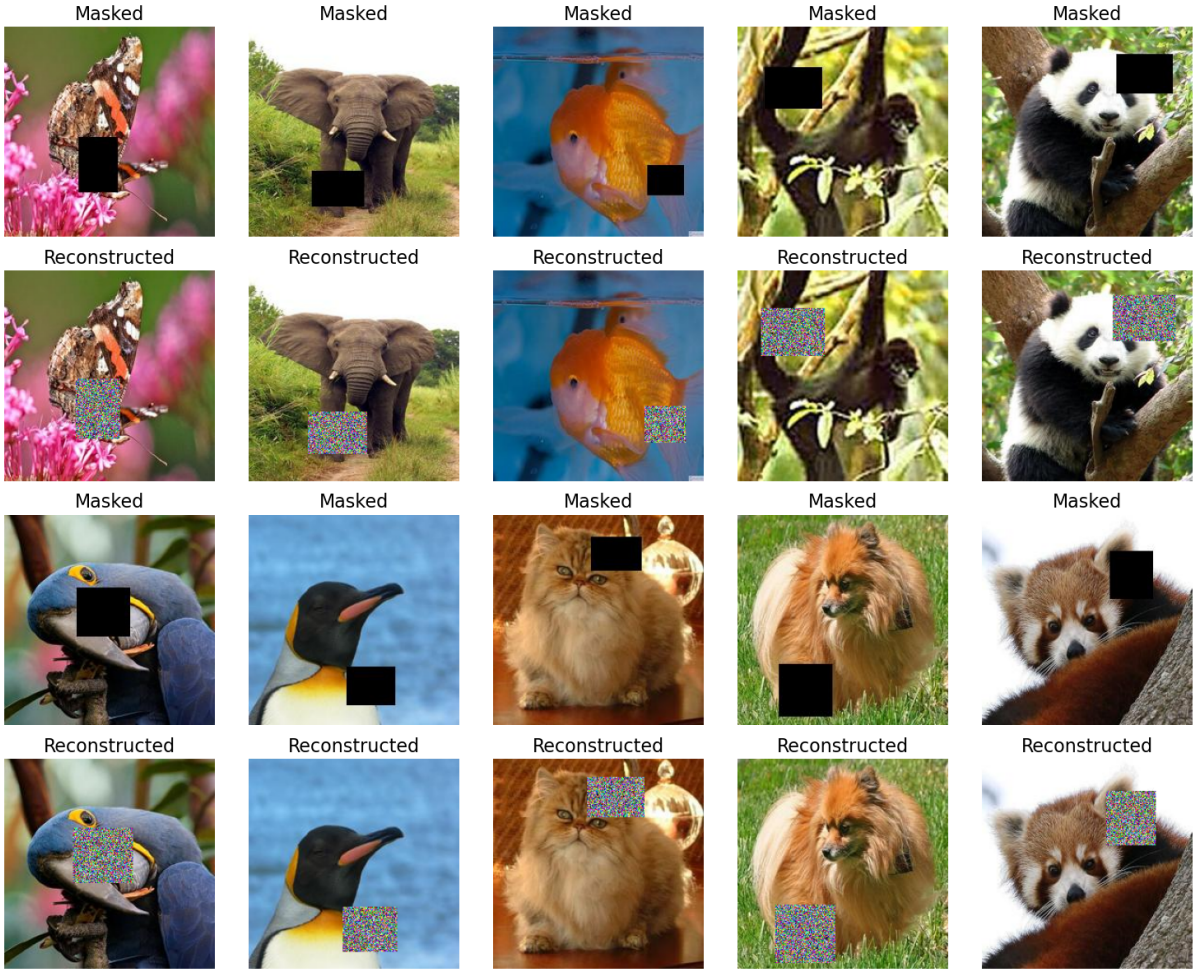

Figure 1: Example outputs from the baseline model. The images were taken directly from the test set for evaluation.

2.2 Linear Regression Model

The Linear regression model reconstructs an image by splitting the masked region into 4 subregions (equal-sized) and then computing linear regression on each of them. The regression works in a way where we train a weight for each of the 3 RGB gates. After we have our 3 trained weights per subregion, we multiply the weights with the surrounding pixels in the masked image and take the mean value of each gate. Then, we assign those mean values as the values of the whole subregion, and eventually we get a colored subregion.

2.2.1 Training

The model is trained by collecting the surrounding pixels for each of the 4 subregions and the mean value of every RGB gate in that same region in the original image. With that data, we perform linear regression and train our weights with gradient descent (we determine the initial weights according to Kaiming Initialization, which is a method in which we calculate a random number with a Gaussian probability distribution with a mean of 0.0 and a standard deviation of $\sqrt{\frac{2}{n}}$). Finally, we obtain our final weight for each RGB gate per subregion.

2.2.2 Challenges

To achieve the final version of the model, we encountered two main challenges. The first was ensuring that the reconstructed parts were in RGB rather than grayscale, which was resolved by adding a conversion step to RGB and normalizing values between 0 and 1. The second was initializing the weights effectively, where we used Kaiming Initialization instead of random initialization for improved results.

2.2.3 Results and Visualization

Linear regression is applied to predict masked pixel values. Table 2 compares performance metrics.

Metric	**Test Value**
Mean Squared Error (MSE)	446.1911
Mean Absolute Error (MAE)	4.4127

Table 2: Linear regression model results.

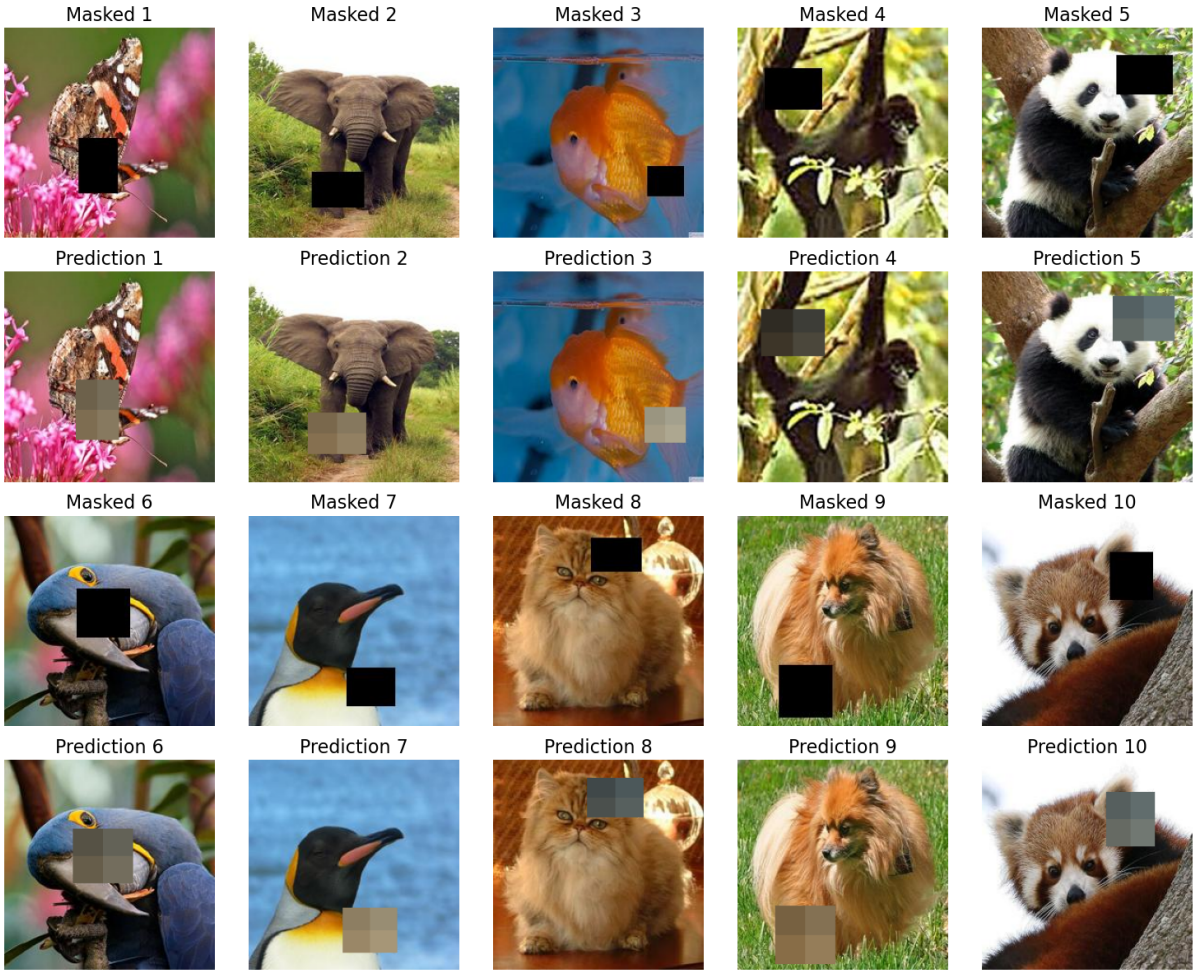

Figure 2: Example images demonstrating the linear regression model's performance.

2.3 Basic Neural Network

The neural network used in this project is a CNN model, where all layers in the model architecture are individually implemented in PyTorch. A detailed explanation of the model's layers can be found in the Jupyter Notebook of the project on GitHub. The model predicts the mean RGB value of each cell in a 4×4 matrix in the masked areas, allowing a low resolution reconstruction of the masked region in each image.

2.3.1 Training

The model is trained using paired masked and original images. The training process involves automatic detection of the masked regions (black rectangles), splitting them into 4×4 subregions, and calculating the mean RGB values for each subregion. The training dataset is loaded using a custom PyTorch `Dataset` class. The training process minimizes the Mean Squared Error (MSE) loss.

The training loop performs the following steps:

- A batch of images and labels is loaded from the training dataset.
- The model predicts the RGB values for 4×4 masked sub-regions.
- The loss is computed using the Mean Squared Error (MSE) function.
- The optimizer updates the model's weights via backpropagation.

2.3.2 Challenges

One of the main challenges was the need for a model capable of predicting a large number of outputs, requiring significant capacity and strength. However, this was complicated by the relatively small size of the dataset, which limited the amount of data available for training. Initially, we started with a simple neural network, but the task also required localization capabilities. To address this, we added convolutional layers to incorporate spatial information, enabling the model to handle the task more effectively.

2.3.3 Training Visualization

Two graphs are plotted to visualize the training process:

1. **Loss Curve:** Shows training and validation loss over epochs (Figure 3).
2. **Reconstruction Quality:** Predictions from the test set example for masked regions compared to ground truth (Figure 4).

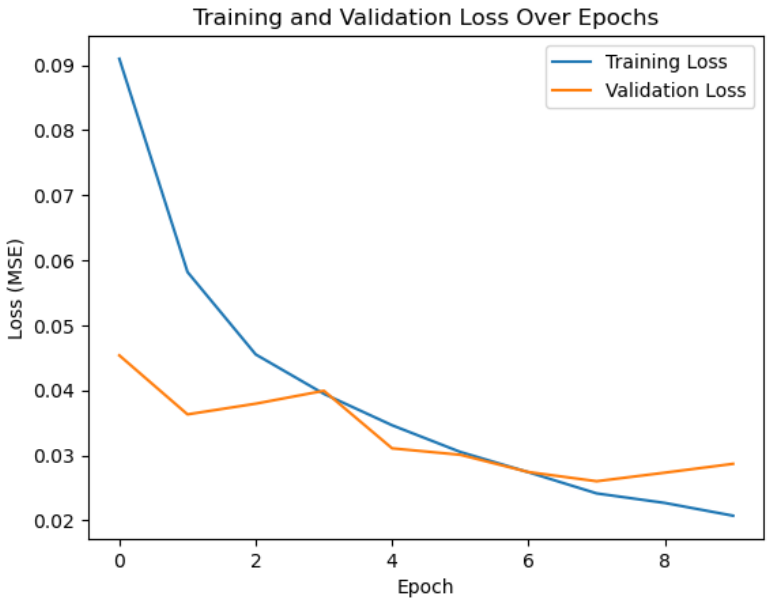

Figure 3: Training and validation loss curves for the neural network.

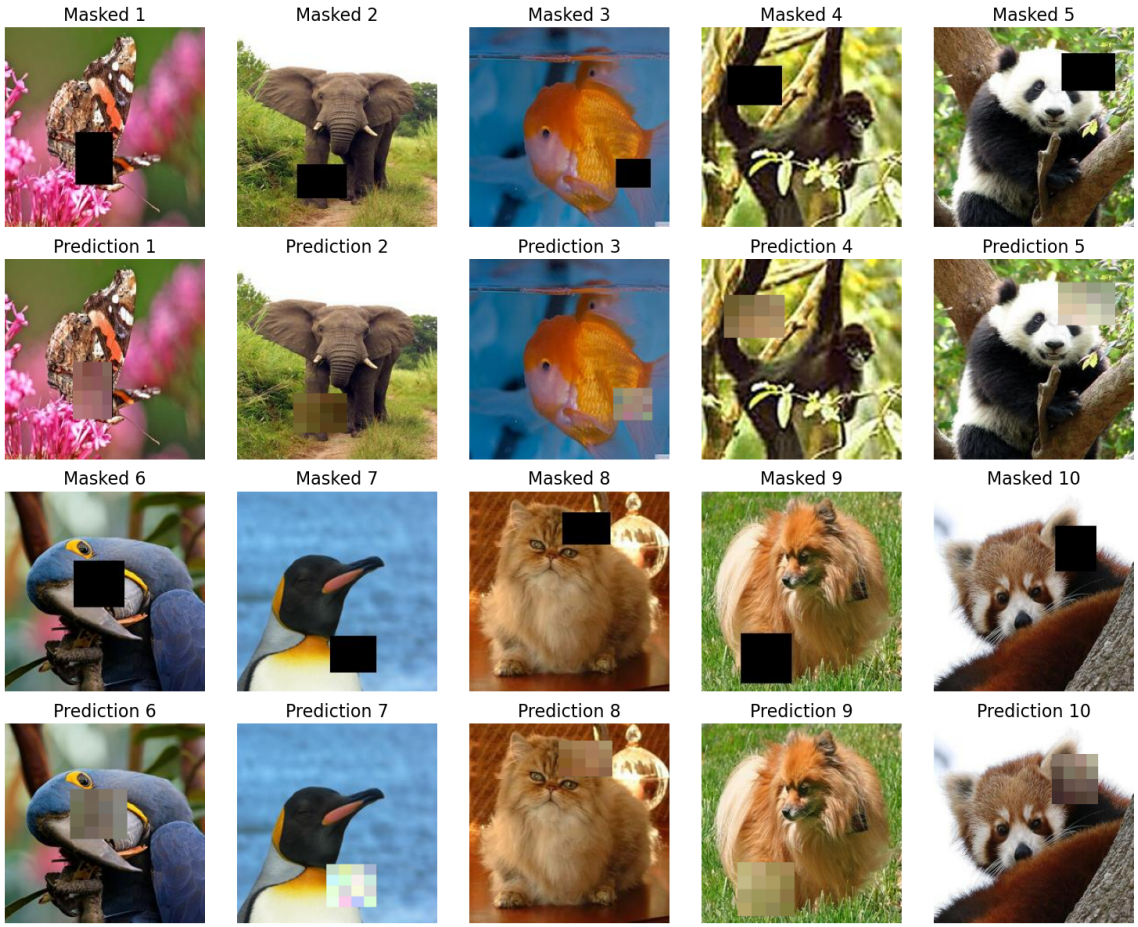

Figure 4: Comparison of ground truth and reconstructed images from the test set. In some images, such as the butterfly, regular panda and red panda, localization is noticeably effective.

2.3.4 Results

Table 3 presents the performance metrics of the neural network model.

Metric	Value
Mean Squared Error (MSE)	336.2765
Mean Absolute Error (MAE)	3.4692

Table 3: Performance metrics of the neural network model.

2.4 Attention Model

The Attention Model is a Convolutional Neural Network (CNN) enhanced with attention mechanisms to improve the reconstruction of masked regions in images. This model combines traditional CNN layers with spatial and channel attention mechanisms, residual connections, and skip connections to achieve high-quality reconstructions. All layers in the model architecture are individually implemented in PyTorch.

2.4.1 Model Explanation

The Attention Model architecture includes the following key components:

- **Convolutional Neural Network Backbone**: The core of the model consists of convolutional layers that extract spatial and channel-wise features from the input image, capturing local patterns essential for reconstruction.
- **Spatial Attention Mechanism**: A lightweight convolutional layer generates an attention map, highlighting the most important spatial regions of the image. This map is applied element-wise to the input features, amplifying critical areas for reconstruction.
- **Channel Attention Mechanism**: This mechanism dynamically adjusts the importance of each feature channel (e.g., R, G, B). A fully connected layer computes channel-wise weights based on globally pooled features, which are then applied to the feature maps.
- **Residual and Skip Connections**: Residual connections help mitigate gradient vanishing and allow the model to learn deeper representations. Skip connections reuse low-level features from earlier layers, aiding in reconstructing fine details.
- **Output Layer**: A final convolutional layer predicts the pixel values of the masked regions, reconstructing the masked region in full resolution.

Data augmentation techniques, including flip and rotation, along with learning rate scheduling, have been implemented to enhance the performance and robustness of the model.

The model architecture can be found in (Figure 5). The complete implementation of the model, training loop, and dataset preparation code can be found on the project's GitHub repository.

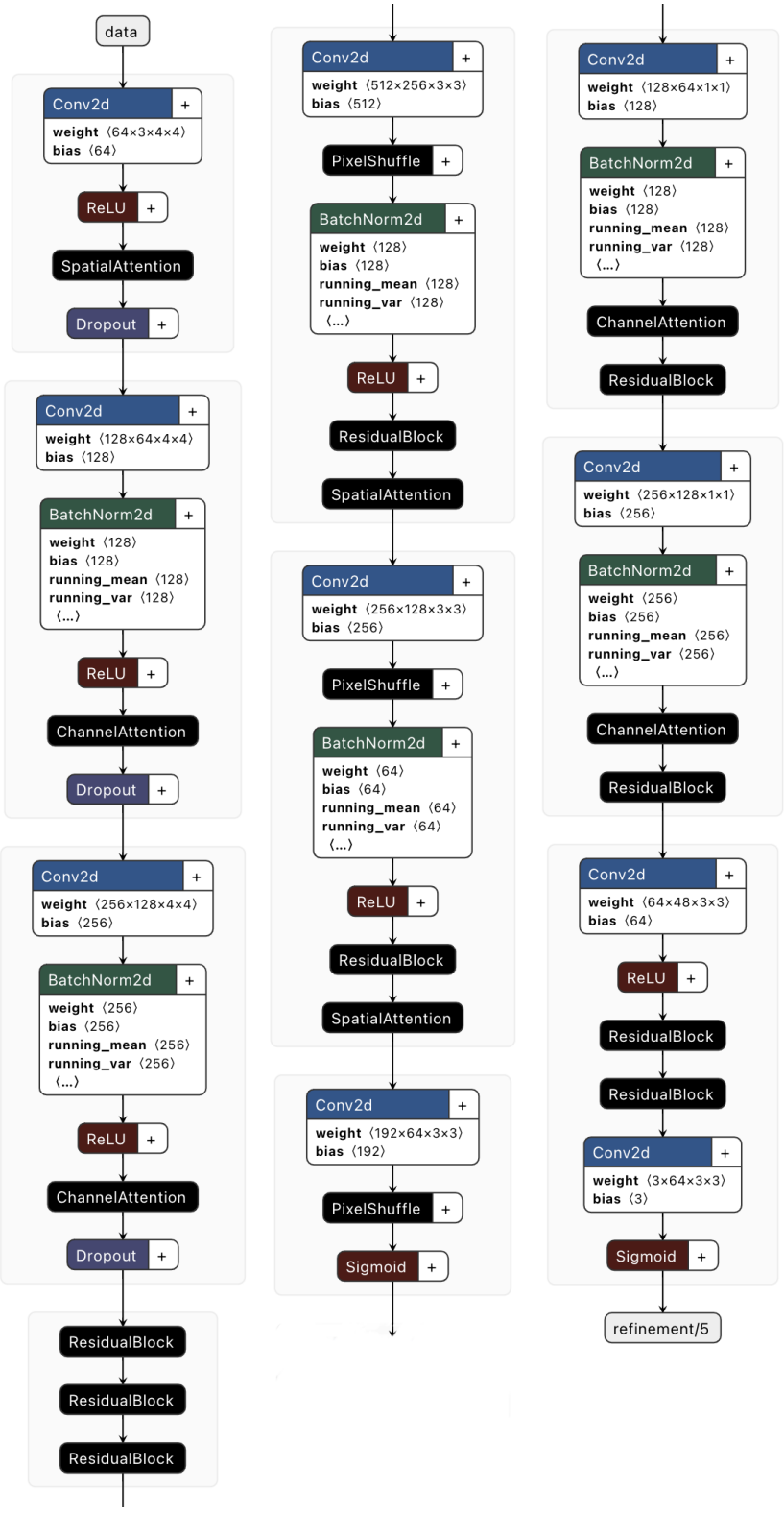

Figure 5: Attention Model Architecture (Using Netron App)

2.4.2 Training Visualization

The training process is visualized in two key graphs:

1. **Loss Curve**: Tracks the training and validation loss over epochs (Figure 6).

2. **Reconstruction Quality**: Compares reconstructed images with the ground truth (Figure 7).

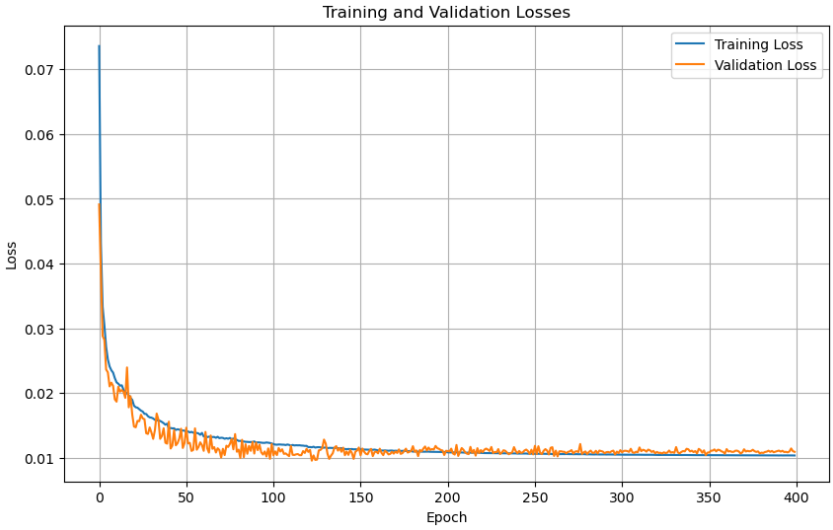

Figure 6: Training and validation loss curves for the Attention Model.

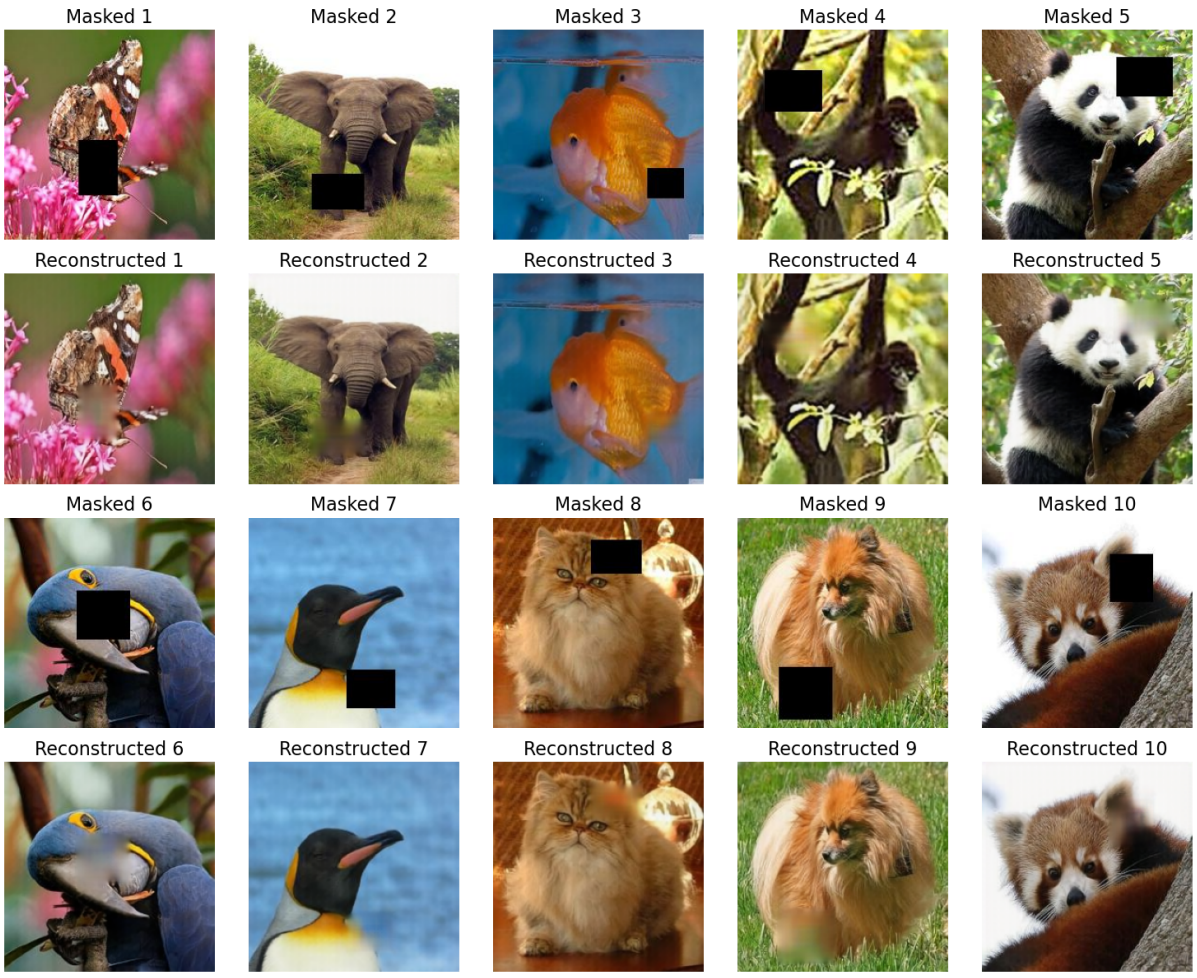

Figure 7: Comparison of ground truth and reconstructed images.

2.4.3 Challenges

Initially, we experimented with a Generative Adversarial Network (GAN) to tackle the reconstruction task. However, achieving a balance between the generator and the discriminator proved to be challenging. Despite attempts to stabilize training, including pretraining the generator, the validation loss increased consistently during training, indicating overfitting or instability in the GAN setup.

Upon closer evaluation, we observed that the generator alone, when enhanced with attention mechanisms, produced promising results. This led us to focus solely on the generator model, incorporating spatial and channel attention layers to refine its performance. This decision led to better performance, as demonstrated in the final results.

2.4.4 Results

The performance of the Attention Model is summarized in Table 4. The model achieves high-quality reconstruction, as evidenced by low MSE and MAE values.

Metric	**Value**
Mean Squared Error (MSE)	57.1340
Mean Absolute Error (MAE)	1.3617

Table 4: Performance metrics for the Attention Model.

2.4.5 How It Works

The model processes a masked image through the following steps:

1. **Input**: The model receives a masked image with blacked-out regions.
2. **Feature Extraction**: Convolutional layers extract multi-scale features from the input image.
3. **Attention Layers**:
    - Spatial attention identifies the most important regions of the image.
    - Channel attention adjusts the importance of individual feature channels.
4. **Residual and Skip Connections:**: Refine the features, which are passed through the output layer to reconstruct the masked regions.

2.4.6 Advantages of Attention Mechanisms

The use of attention mechanisms enhances the model's performance by:

- **Efficiency**: Attention mechanisms prioritize the most relevant regions and channels, reducing redundant computations.
- **Accuracy**: By focusing on critical features, the model achieves better reconstruction quality, especially in challenging cases.
- **Flexibility**: Spatial and channel attention adapt dynamically to different occlusion patterns and image types.

3 Conclusion

This project highlights the evolution from basic to advanced models for image reconstruction. The attention-based model significantly exceeds simpler methods and excels at capturing patterns in masked regions. Its effectiveness is evident both quantitatively, through improved metrics, and qualitatively, as reflected in the visual reconstructions.

The following is a summary table of all the methods and their associated metrics.

Metric	Model	Value
Mean Squared Error (MSE)	Baseline	1139.2030
	Linear Regression	446.1911
	Neural Network	336.2765
	Attention (Best)	**57.1340**
Mean Absolute Error (MAE)	Baseline	7.0292
	Linear Regression	4.4127
	Neural Network	3.4692
	Attention (Best)	**1.3617**

Table 5: Summary of model performance metrics.

A Appendix

This appendix includes instructions for running the project code and accessing its datasets.

A.1 Dataset Preparation

The dataset can be retrieved from the Hugging Face repository using the provided script:

`Dataset_Pipeline/Download_Dataset.ipynb`

To use this script, ensure that you provide your Hugging Face token, which is required to grant access to the repository. In this script you can also select the classes that will be used during model training. This script stores the requested images in:

`Dataset_Pipeline/imagenet_selected_raw_classes`

The following script defines all the necessary user-specified parameters for creating the project dataset. These include the required paths, mask size, dataset split ratios, and image dimensions.

`Dataset_Pipeline/Dataset_Preparation.ipynb`

Since data downloading and preparation can be time-consuming, you may skip this process by directly downloading an example dataset from Google Drive to your local project directory. The example dataset is available at the following link:

`https://drive.google.com/drive/folders/16jMNDl2-trXoFd3qisRZFyc2A2pp858E?usp=sharing`

A.2 Saved Models

Each time a Neural or Attention-based mode is trained, its architecture and weights are saved in a `.pth` file. The trained models corresponding to the example dataset are available on Google Drive via the link provided above.