Project in digital mapping 1 - 014869

Semester – spring

By: Ilya Vorontsov

Supervisor – dr. Lior Rubanenko

Denoising Aerial Imagery Using Autoencoder

Abstract

This project explores the application of autoencoders for denoising aerial imagery, a critical enhancement for improving the use of such images in urban planning, environmental monitoring, and disaster response. Through the development and training of a convolutional autoencoder, this study aims to effectively reduce noise from high-resolution aerial images. The noises which the project aims to denoise are mainly in the form of smearing, an effect caused by the expose of the camera.

Introduction

Aerial imagery is used in many modern applications but often suffers from significant unclarity due to sensor issues, atmospheric conditions, and other noise-inducing factors. Traditional methods for image denoising are often insufficient for handling the complex nature of noise in aerial images. This project proposes the use of an autoencoder, a deep learning model known for its efficacy in image reconstruction tasks, to address this challenge. The project is based on a dataset comprising 10,000 images from different places around the word, of different objects, like forests, beaches, airports, schools, ports etc.

Objectives

Model Development: Design and train a convolutional autoencoder to remove noise from aerial images, and eventually reconstruct the noised images.

Performance Evaluation: using loss function to train an autoencoder , which measures the difference between the original image and the reconstructed image:

$$L(x, x_i) = \frac{1}{n}\sum_{i=1}^{n}(x_i - x_i)^2$$

Where n is the number of pixels in the image.

Comparison with other Techniques: Compare the results with those obtained using other image processing methods to highlight the improvements offered by deep learning approaches.

Methodology

1. Data Collection

As mentioned before, the data set consists of 10,000 pairs of aerial images from different subjects, with each pair containing a noised and a corresponding clean image. These images were firstly taken from publicly available datasets and been applied with smearing noises to mimic typical real-world noising that causing in arial imagery.

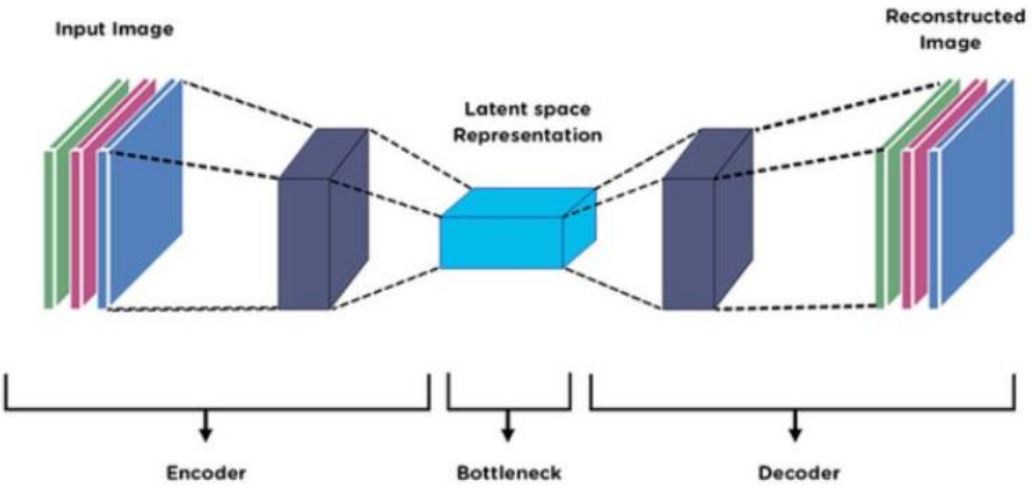
The smearing noise was created in a specific function that mainly uses the random application of blurring over small patches of the image, which mimics the effect of smearing or motion blur. This effect seems to be similar to the one naturally occurring when trying to take an image of a moving car, in which the car seems to appear smeared on more pixels than its actually is.

The image patch defined by (x, y) and smear size is extracted and blurred using OpenCV's cv2.blur function. The same area on the original image is then replaced with this blurred patch.

For example, showing the output of the data preparing for a set of 5 random images:

As can be seen from the results, the smearing noises have been applied to the original images, which now the images seem to be taken when the exposure of the camera wasn't calibrated, causing the smear effect.

2. Model Architecture



The Autoencoder:

An autoencoder is a type of neural network which uses dimensionality reduction or feature learning. It can be particularly effective because of its ability to learn a compressed representation of the input data and then reconstruct it, ideally with less noise.

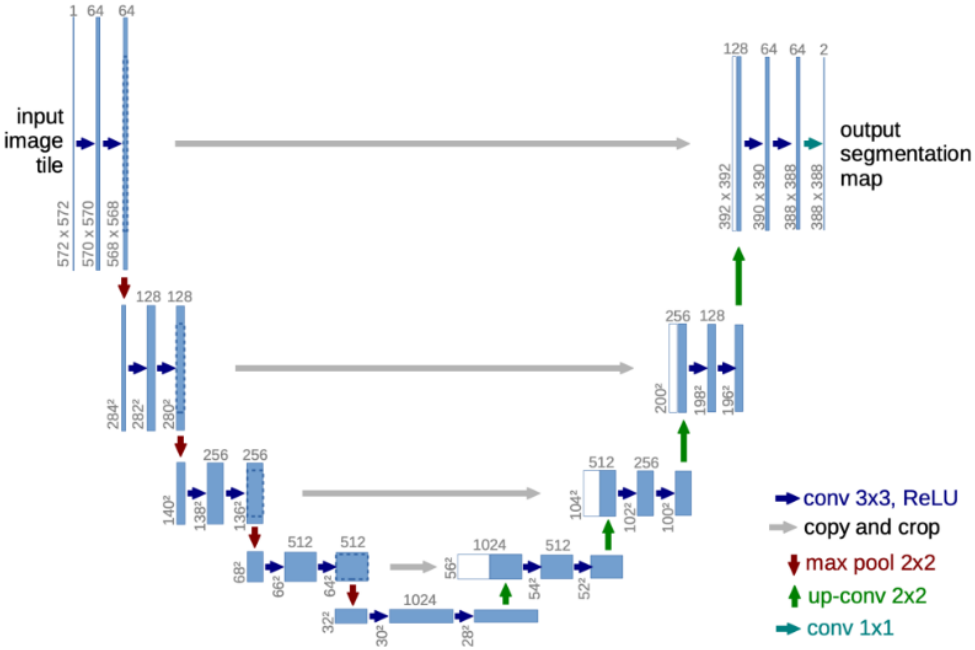
General explanation of the architecture:

Encoder: The encoder employs convolutional layers to down sample the image, reducing dimensionality while capturing key features. These layers apply various filters (also known as kernels) to the input image to create feature maps. Each filter is designed to detect specific features such as edges, textures, or other patterns. The output of the encoder is a compressed representation of the input image. This representation is smaller in dimension than the original image, mainly focusing on essential information.

Decoder: The decoder uses convolutional layers to reconstruct the image from the compressed representation created by the encoder. It aims to produce an output that closely resembles the original input image but ideally without the noise. The decoder also uses convolutional layers, but instead of reducing the dimensions, these layers are used to progressively increase the spatial dimensions of the representations.

U-Net Autoencoder: this type of architecture which is used in this implementation is particularly adapted to tasks like image denoising.

- **Skip Connections**: this architecture uses skip connections, where features from the encoder are directly connected to the corresponding layers in the decoder. This allows the network to retain high-resolution information that might otherwise be lost during the down-sampling process in the encoder. These skip connections help preserve fine details in the image during reconstruction, which is crucial for tasks like denoising, where small features can be easily lost.
- **High-Resolution Feature Maps:** This preservation of spatial information helps the network to better differentiate between noise and actual image content, leading to more effective denoising.
- **Effective Use of Data**: The skip connections and convolutional layers in UNet allow the model to make more efficient use of the training data. This can help in preventing overfitting, where the model learns noise patterns in the training data rather than the true underlying structure.



As can be seen from the image, the structure and the flow of data through the network is represented as a shape of a 'U'. the left side gradually downsamples the input image, increasing feature complexity while reducing spatial resolution. the central contains the most compressed and abstract representation of the image. The right contains the most compressed and abstract representation of the image.

Project's UNet Architecture:

- **Encoder**: Composed of two blocks, each with two convolutional layers followed by a max-pooling operation. The encoder captures increasingly complex features while reducing the spatial resolution.
- **Bottleneck**: The compressed representation of the input.
- **Decoder**: Consists of two blocks, each with up-convolutions followed by concatenation with the corresponding encoder features (skip connections). These blocks upsample the feature maps back to the original image resolution.
- **Output**: The final output is a reconstructed image with the same resolution as the input, with pixel values scaled between 0 and 1.

Encoder:

1. **First Convolutional Block (c1)**:

c1 = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(inputs):

- A 3x3 convolution is applied with 32 filters

c1 = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(c1):

- Another convolution layer is applied to the output of the previous convolution.

p1 = layers.MaxPooling2D((2, 2))(c1):

- Max pooling with a 2x2 window is applied to downsample the feature map by a factor of 2.

2. **Second Convolutional Block (c2)**:

c2 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(p1):

c2 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(c2):

p2 = layers.MaxPooling2D((2, 2))(c2):

Bottleneck:

3. **Third Convolutional Block (c3):**

c3 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(p2):

c3 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(c3):

Decoder:

1. **First Up-Convolution Block (u4):**

u4 = layers.Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c3):

u4 = layers.concatenate([u4, c2]):

c4 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(u4):

c4 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(c4):

2. **Second Up-Convolution Block (u5):**

u5 = layers.Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same')(c4):

u5 = layers.concatenate([u5, c1]):

c5 = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(u5):

c5 = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(c5):

Output Layer:

outputs = layers.Conv2D(3, (1, 1), activation='sigmoid')(c5):

- A final 1x1 convolution is applied to reduce the feature maps to the desired number of output channels (3 in this case, corresponding to RGB channels). The sigmoid activation scales the output pixel values to the range [0, 1].

Training

Training an autoencoder for denoising aerial imagery is a critical process that involves several systematic steps. Each step in the training pipeline is designed to ensure the model effectively learns to filter out noise from the images while retaining important features. During training, the model's performance is tracked through metrics like loss and accuracy, which are calculated at the end of each epoch.

The data, which is eventually the 10,000 original and noised images is divided into training (80%), validation (10%), and testing (10%).

Preparation for Training

- Loss Function: The Mean Squared Error (MSE) is used to quantify the difference between the denoised output and the ground truth (clean images), which is typical for regression tasks like image reconstruction.
- Optimizer: The Adam optimizer is used with a learning rate of 0.0001, which is a popular choice due to its efficiency in converging in training deep learning models.

Training Loop

- Epochs: The model is trained for 25 epochs, where each epoch represents a full pass through the entire training dataset.
- Batch Processing: batch number is set to 256. In each epoch, the model processes 256 of images, performing forward passes to generate outputs and backward passes to update weights based on the computed loss.
- Validation: After each training epoch, the model's performance is evaluated on a separate validation dataset which helps monitor for overfitting and provides insight into how well the model generalizes to new data.

Evaluation on Test Data

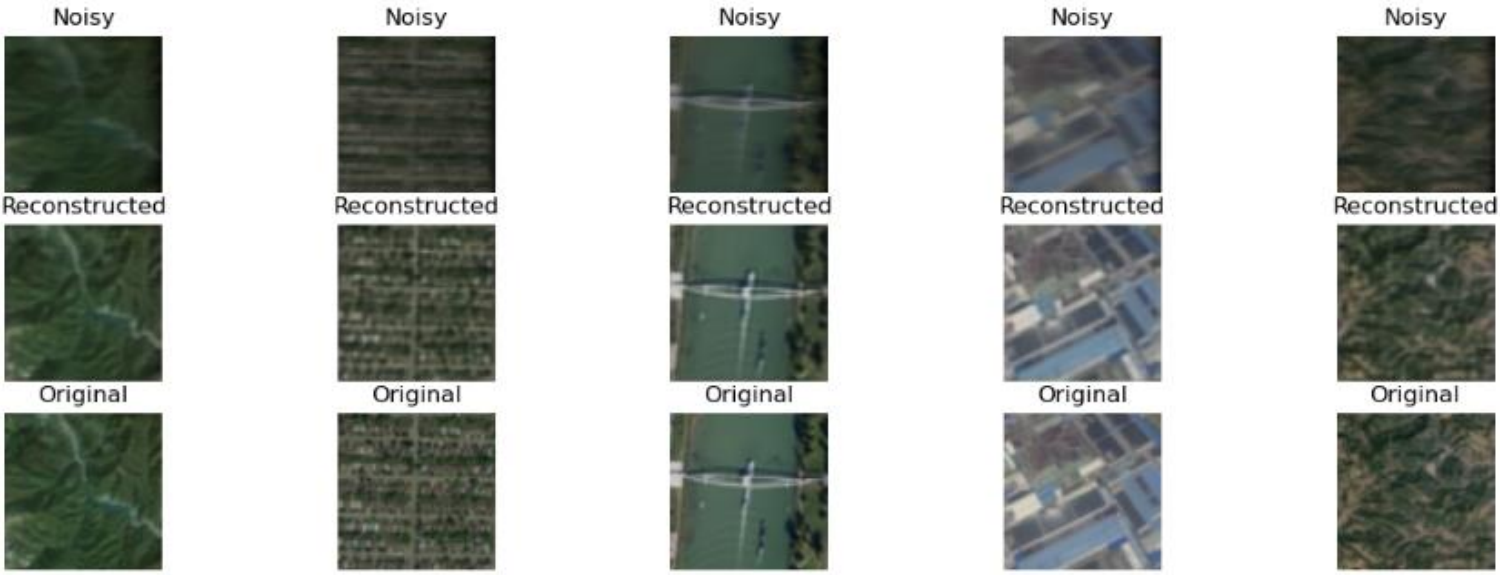
After training and validation, the model's final performance is assessed on a test dataset that the model has never seen during training. This provides a measure of the model's ability to generalize and perform denoising on completely new data.

Visualization of Results

Visualization Function: A custom function imshow is defined to display images. It adjusts the normalization of images and handles the formatting necessary for displaying them using matplotlib.

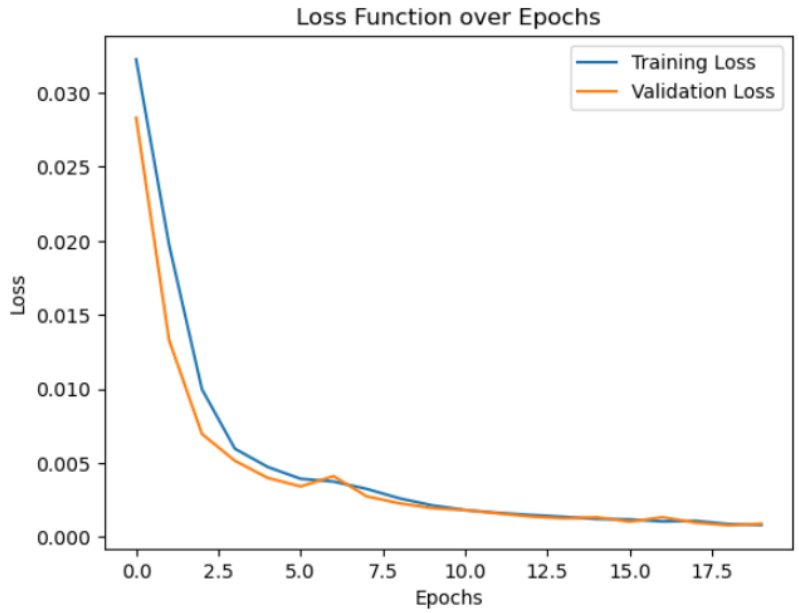
Displaying Images: The code visualizes original, noised, and denoised images from the test dataset. This visual comparison is crucial for qualitatively assessing the performance of the autoencoder.

For example, showing the output of the training, validation and testing with the current settings of the autoencoder:

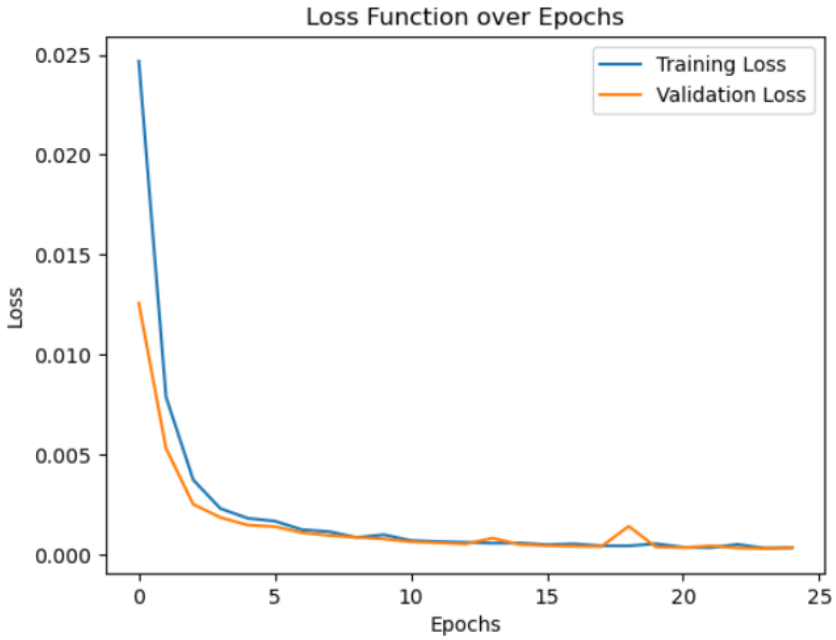


As can be seen from the results, the denoised images are reconstructed very well in the current settings of the model architecture. The reconstructed images seem to match the original ones, except for minor changes.

For the setting of 512 batches and 20 epochs, the overall accuracy of the model was about 80.6%.



For the setting of 256 batches and 25 epochs, the overall accuracy of the model was better with almost 85%



Examining the Model:

In order to further analyze the results of the trained model, there's a need for another metric. PSNR is used to evaluate how well the UNet autoencoder can remove motion blur from images. By calculating PSNR between the reconstructed sharp images and the original sharp images, we can determine how effectively the model restores the sharpness.

Using PSNR allows us to compare the model's performance against other deblurring models and If the PSNR is significantly lower, it indicates that the model may need further tuning or that it struggles with the specific type of blur present in the dataset.

PSNR (Peak Signal-to-Noise Ratio) is a common image quality metric used to measure the accuracy of image reconstruction, compression, or transformation. It is used in image processing to evaluate how close a reconstructed or transformed image is to the original reference image. It is expressed in decibels (dB), where higher values indicate better image quality.

The PSNR is calculated by:

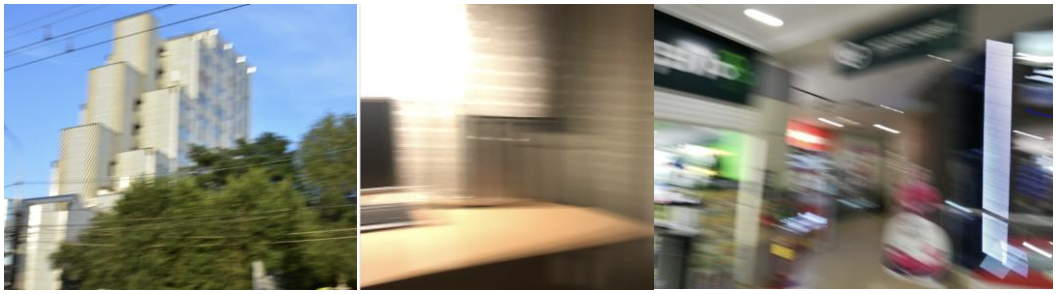
$$PSNR = 10 * \log_{10}\left(\frac{MAX_I^2}{MSE}\right)$$

Where:

- MAX_I is the maximum possible pixel value of the image. For 8-bit images, this value is 255.
- MSE is the Mean Squared Error between the original image and the reconstructed image.

PSNR values above 30 dB are considered good for most image reconstruction tasks, whereas values lower than 20 dB indicate significant differences between the original and reconstructed images.

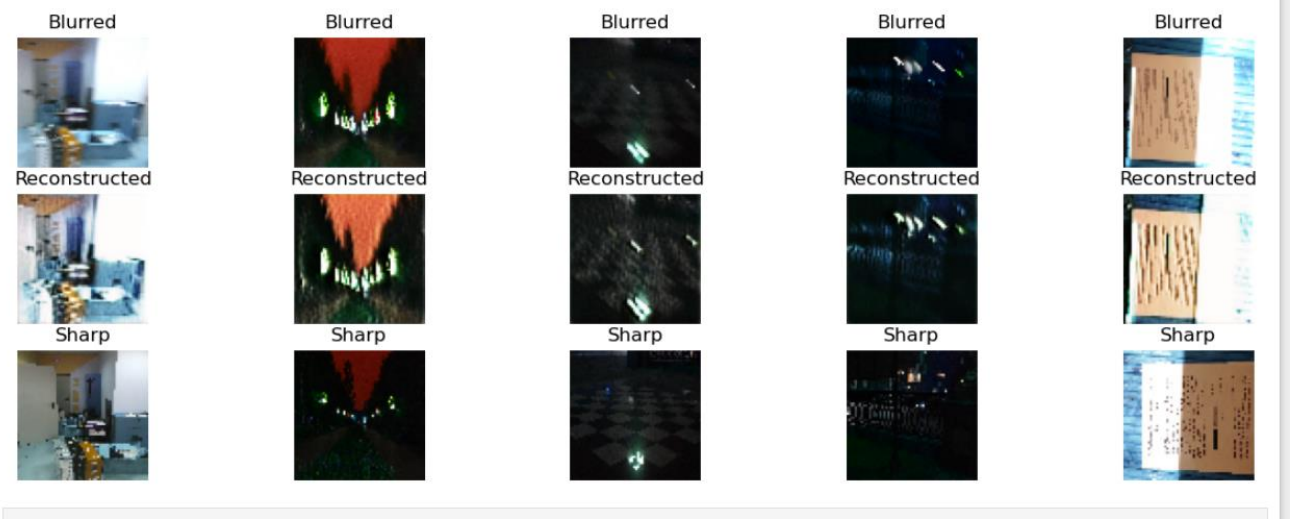
In our project, another dataset was used in order to asses the quality of the model. The new datset contains 350 motion-blurred images and 350 sharp images.



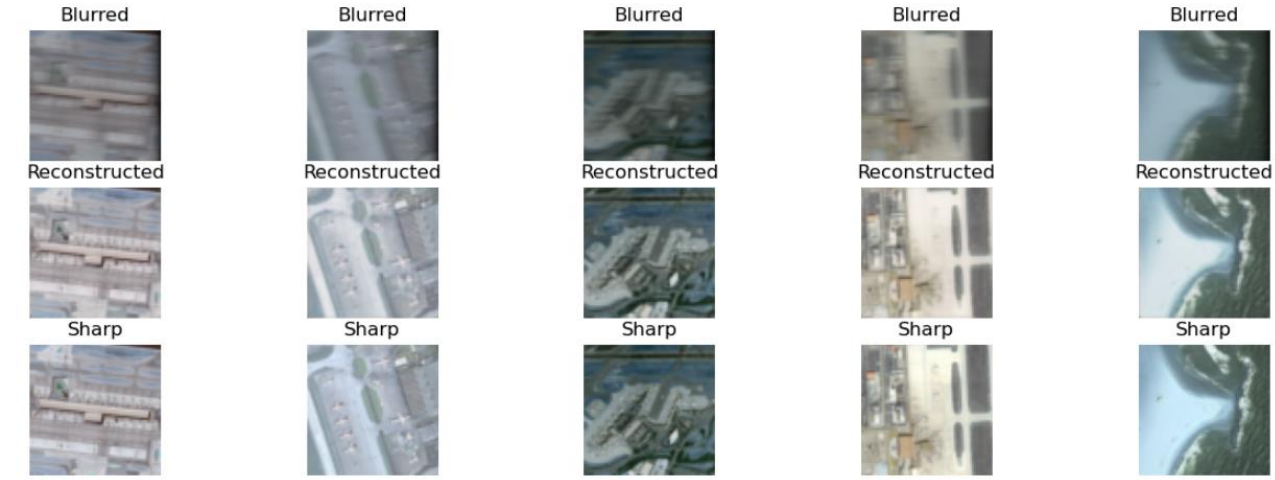
As can be seen from the images, the motion blurring effecet in them is very noticeable, and much more segnificant in the images, that the motion blurred image used to train our model. Also, another main difference is the images themselves.

Firstly, when testing PSNR on the orignal dataset, the average PSNR was around 33 dB which indicates pretty good reconstruction of the motion blurred images.

Secondly, when testing on the other data, the Average PSNR was around 10-11 dB, which is relatively low compared to the expected results (above 30). When comapring results to the values which evaluated from the trained model, it could be seen that there are noticeable difference in performance. The model didn't sucseeded in recunstructing the images as from the pre-trained model and not much of a difference was between the motion blurred and the reconstructed images.



While on the original dataset the results where: (with PSNR of 33.2db)



The difference in results may be due to several factors:

1. **Dataset Differences**: The new dataset contains motion blur, but the characteristics of the blurring may differ from those in the original dataset on which the model was trained on. Different types of blurring, such as the extent and direction of motion, can affect the model's performance significantly.

2. **Training on a Larger Dataset**: The pre-trained models were likely trained on a much larger and more diverse dataset, giving them a better ability to generalize across different types of blurring. In contrast, our model was trained on a smaller dataset, which may not have exposed it to the full range of motion blur effects present in the new dataset.

3. **Model Complexity**: The pre-trained models might also have more complex architectures or additional techniques (data augmentation or specific motion deblurring strategies) that make them better suited for handling motion blur, whereas our model was trained on more generalized image reconstruction tasks.

Conclusions and thougts for future:

In this project, the goal was to enhance and reconstruct aerial images by applying a UNet autoencoder to sharpen images that had been artificially noised with motion blur. This simulated real-world conditions, such as photographing from moving aircraft or drones. The goal was to restore these artificially blurred images to closely resemble their original, sharp state. The model was trained to recognize patterns of blurring and to learn how to reverse them, improving the clarity and detail of the aerial images.

The performance of the model was tested using the several metrics such as loss function and PSNR (Peak Signal-to-Noise Ratio). Results showed that the model was able to successfully sharpen the aerial images and improve their overall quality. However, the results on other test datasets were lower than expected, indicating that the model doesn't suit well for all types of blurred images. It was found that the type and intensity of motion blur in the images, along with the variety of conditions seen in aerial imagery, took a significant role in determining the model's effectiveness.

To further improve the model's performance on aerial image deblurring, as well as other types of imagers, there are more several steps that could be explored:

- Expand and Diversify the Dataset: Increasing the variety of images used in training—covering different types of motion blur, weather conditions, and terrain—will help the model generalize better to real-world scenarios.

- Advanced Data Augmentation: Introducing more complex blur patterns and distortions during training can make the model more adaptable to various types of motion blur commonly seen in aerial photography.

- Experiment with Specialized Architectures: Testing more advanced deblurring architectures, could enhance the model's ability to handle fine details in high-resolution aerial images.

These improvements could significantly enhance the model's ability to handle a wide range of conditions in aerial imagery, making it more versatile and robust for real-world tasks. Being able to train eventually such a model can be very useful fur many fields, like remote-sensing applications, photogrammetry, image processing and etc.