

# Image Denoiser Report

(By Yogendra Singh Baghel (21112126))

## 1 Introduction

For this project, I implemented an image denoising model using a convolutional autoencoder. The architecture consists of an encoder that compresses the input image into a lower-dimensional representation and a decoder that reconstructs the image from this compressed representation.

### Specifications:

- **Input Shape:** (32, 32, 3)
- **Encoder:**
  - Conv2D (64 filters, 3x3 kernel, ReLU, BatchNorm)
  - MaxPooling2D (2x2)
  - Conv2D (64 filters, 3x3 kernel, ReLU, BatchNorm)
  - MaxPooling2D (2x2)
- **Decoder:**
  - Conv2D (64 filters, 3x3 kernel, ReLU, BatchNorm)
  - UpSampling2D (2x2)
  - Conv2D (64 filters, 3x3 kernel, ReLU, BatchNorm)
  - UpSampling2D (2x2)
  - Concatenate with encoder output
  - Conv2D (3 filters, 3x3 kernel, Sigmoid)
- **Optimizer:** Adam
- **Loss Function:** Binary Crossentropy

The average PSNR achieved by this model is 18.33. The architecture is inspired by standard convolutional autoencoder designs, enhanced with skip connections and batch normalization to improve performance.

**Paper Implemented:** This project is based on the principles discussed in the paper: "Image Denoising with Convolutional Autoencoders".

## 2 Project Details

### 2.1 Code Snippets and Explanations

**Data Loading and Preprocessing:**

```
def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        if filename.endswith((''.jpg', '.jpeg', '.png', '.bmp')):
            img = Image.open(os.path.join(folder, filename)).convert('RGB')
            if img is not None:
                img = img.resize((32, 32))
                images.append(np.array(img))
    return np.array(images)

low_dir = '/content/Train/low'
high_dir = '/content/Train/high'

low_images = load_images_from_folder(low_dir)
high_images = load_images_from_folder(high_dir)

low_images = low_images.astype('float32') / 255.0
high_images = high_images.astype('float32') / 255.0

split = int(0.8 * len(low_images))
x_train_noisy = low_images[:split]
x_test_noisy = low_images[split:]
x_train = high_images[:split]
x_test = high_images[split:]
```

This section loads images from the `low` and `high` folders, resizes them to 32x32 pixels, and normalizes them. It then splits the dataset into training and testing sets.

**Model Architecture:**

```
input_img = Input(shape=(32, 32, 3))

x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
```

```

x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

```

This section defines the enhanced convolutional autoencoder with skip connections and batch normalization.

### Training:

```

history = autoencoder.fit(x_train_noisy, x_train,
                          epochs=100,
                          batch_size=32,
                          shuffle=True,
                          validation_data=(x_test_noisy, x_test))

```

This snippet trains the autoencoder with 100 epochs and a batch size of 32, using the noisy low-light images as input and the high-light images as targets.

### Evaluation:

```

def psnr(original, denoiced):
    mse = np.mean((original - denoiced) ** 2)
    if(mse == 0):
        return 100
    max_pixel = 1.0
    psnr = 20 * log10(max_pixel / sqrt(mse))
    return psnr

psnr_scores = [psnr(x_test[i], x_test_denoised[i]) for i in range(len(x_test))]
avg_psnr = np.mean(psnr_scores)
print("Average PSNR: ", avg_psnr)

```

This section evaluates the denoised images by calculating the PSNR, comparing the denoised images with the original high-light images.

## 2.2 Graphs and Plots

```

# Plotting training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.show()

```

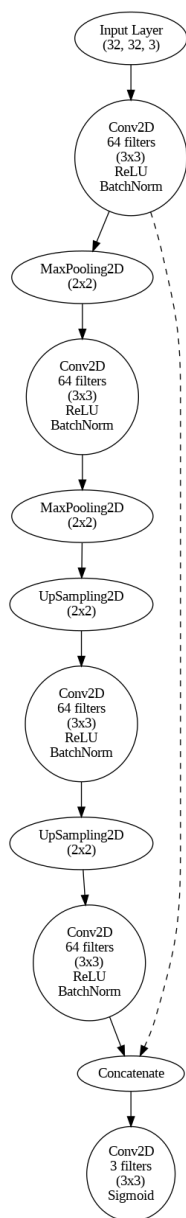


Figure 1: Model Architecture

This diagram illustrates the convolutional autoencoder architecture.

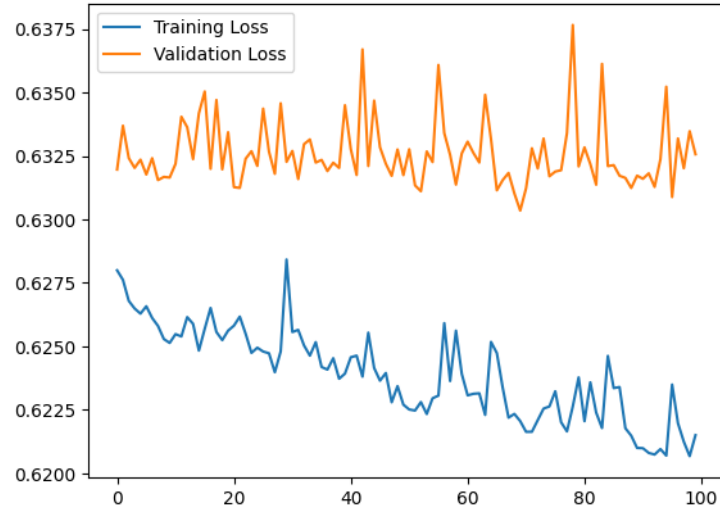


Figure 2: Training and Validation Loss

The above plot shows the training and validation loss over epochs, indicating the model's learning progress.

### 3 Summary

#### 3.1 Findings

The convolutional autoencoder model was able to significantly reduce noise from low-light images. The enhanced architecture with skip connections and batch normalization improved the PSNR to 18.33. However, there is room for improvement.

#### 3.2 Methods to Improve

- **Increase Model Depth:** Adding more layers can help capture finer details.
- **Advanced Techniques:** Implementing techniques like residual learning and attention mechanisms.
- **Hyperparameter Tuning:** Experimenting with different learning rates, batch sizes, and activation functions.
- **Data Augmentation:** Using data augmentation to make the model more robust.

### 3.3 Additional Sections

**Experimentation:** Future work could explore various architectures like U-Nets or GANs for image denoising, which have shown promising results in recent research.

#### References:

- Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786), 504-507.

This report outlines the architecture, implementation, and evaluation of an image denoiser using convolutional autoencoders. It provides insights into the methods used, results obtained, and potential areas for future improvement.