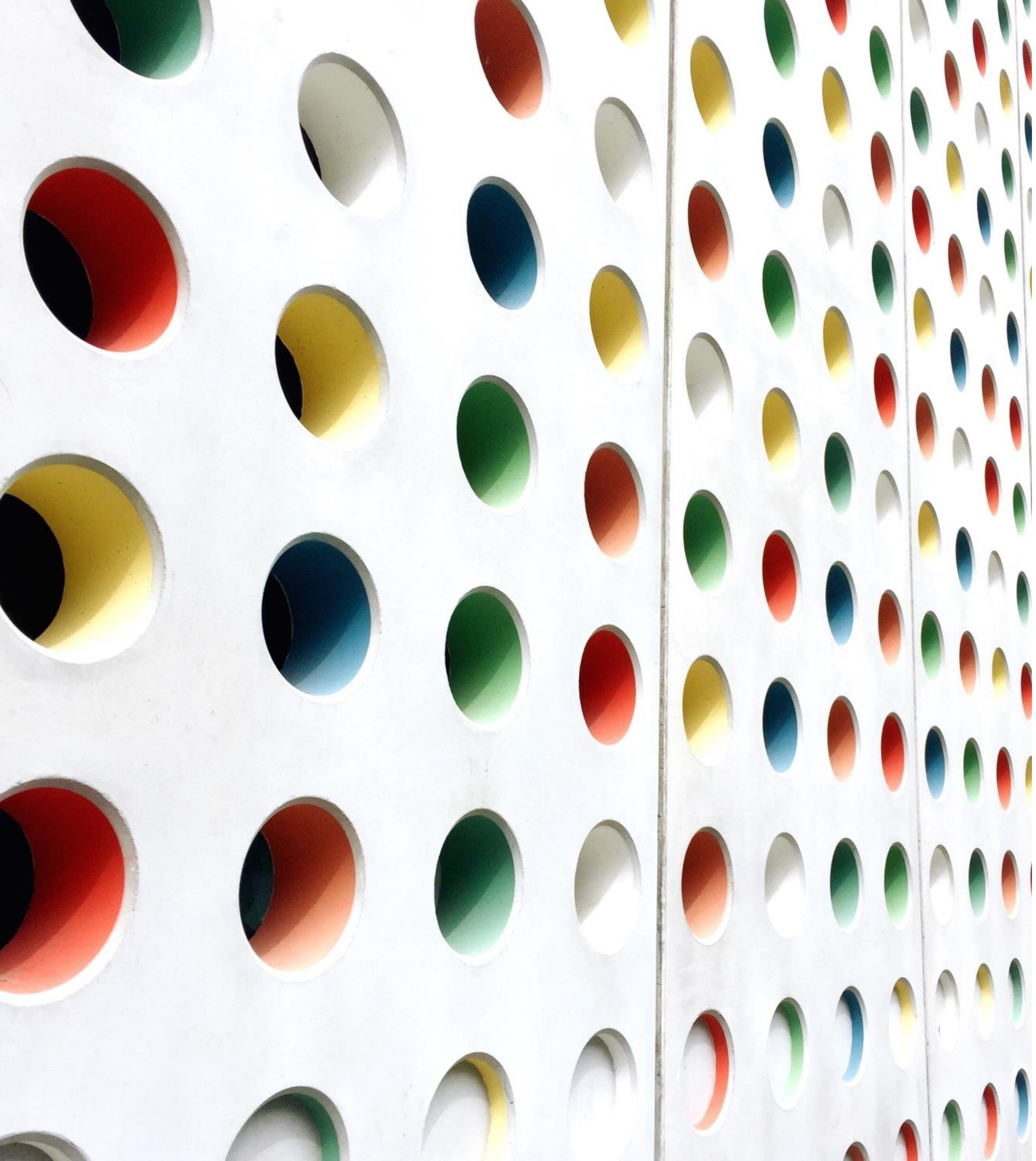


Machine Learning Project Presentation

IMAGE RESTORATION

Introduction

- **Aim** of the Project: **recover** an image from a degraded version
- What is **Image Restoration** all about?
- How does a picture get **degraded**?

Configuration

- Import all the required **modules**
- Definition of **input** images
- Definition of **Hyperparameters**:
  - batch_size
  - number_of_epochs
- Declaration of how many pictures to be shown

```
img_width, img_height, img_channels = 28, 28, 1
input_shape = (img_width, img_height, img_channels)

batch_size = 128
number_of_epochs = 100
verbosity = 1
n = 10
```

Data Import

- Load **MNIST** dataset *(60000 x 28 x 28)*
- Definition of valuable values
  - i.e.: number_of_rows and new_shape
- Data **Normalization**
  - float32 type
  - 0/1 normalization by 255 division
- **Reshape** Data *(60000 x 28 x 28 x n_chan)*

```
(x_train, _), (x_test, _) = mnist.load_data()

# Definition of valuable values.
x_train_number_of_rows = len(x_train)
x_test_number_of_rows = len(x_test)
x_train_new_shape = (x_train_number_of_rows, img_width, img_height, img_channels)
x_test_new_shape = (x_test_number_of_rows, img_width, img_height, img_channels)

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Reshape data
x_train = x_train.reshape(x_train_new_shape)  # (number_of_rows,28,28,1)
x_test = x_test.reshape(x_test_new_shape)  # (number_of_rows,28,28,1)
```

Model definition pt.1

- Definition of the layers of the autoencoder
  - Encoder:
    - Conv2D
    - MaxPooling2D
  - Decoder:
    - Conv2D
    - UpSampling2D

```
input_img = Input(shape=input_shape)  # 28,28,1

# Encoder Layers
# Conv1 #
x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D(pool_size=(2, 2), padding='same')(x)
# Conv2 #
x = Conv2D(filters=8, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D(pool_size=(2, 2), padding='same')(x)
# Conv3 #
x = Conv2D(filters=8, kernel_size=(3, 3), activation='relu', padding='same')(x)
encoder_output = MaxPooling2D(pool_size=(2, 2), padding='same')(x)

# Decoder Layers
# DeConv1
x = Conv2D(filters=8, kernel_size=(3, 3), activation='relu', padding='same')(encoder_output)
x = UpSampling2D(size=(2, 2))(x)
# DeConv2
x = Conv2D(filters=8, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = UpSampling2D(size=(2, 2))(x)
# DeConv3
x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu')(x)
x = UpSampling2D(size=(2, 2))(x)
decoder_output = Conv2D(filters=1, kernel_size=(3, 3), activation='sigmoid', padding='same')(x)
```

Model definition pt.2

- Instantiate & compile the **Autoencoder**:
  - optimizer = 'Adam'
  - loss = 'binary_crossentropy'

- Instantiate & compile the **Encoder**:
  - optimizer = 'Adam'
  - loss = 'binary_crossentropy'

```
autoencoder = Model(input_img, decoder_output, name='Convolutional_Autoencoder')
autoencoder.compile(optimizer='Adam', loss='binary_crossentropy')

encoder = Model(input_img, encoder_output, name='Encoder')
encoder.compile(optimizer='Adam', loss='binary_crossentropy')

autoencoder.summary()
```

Autoencoder Training

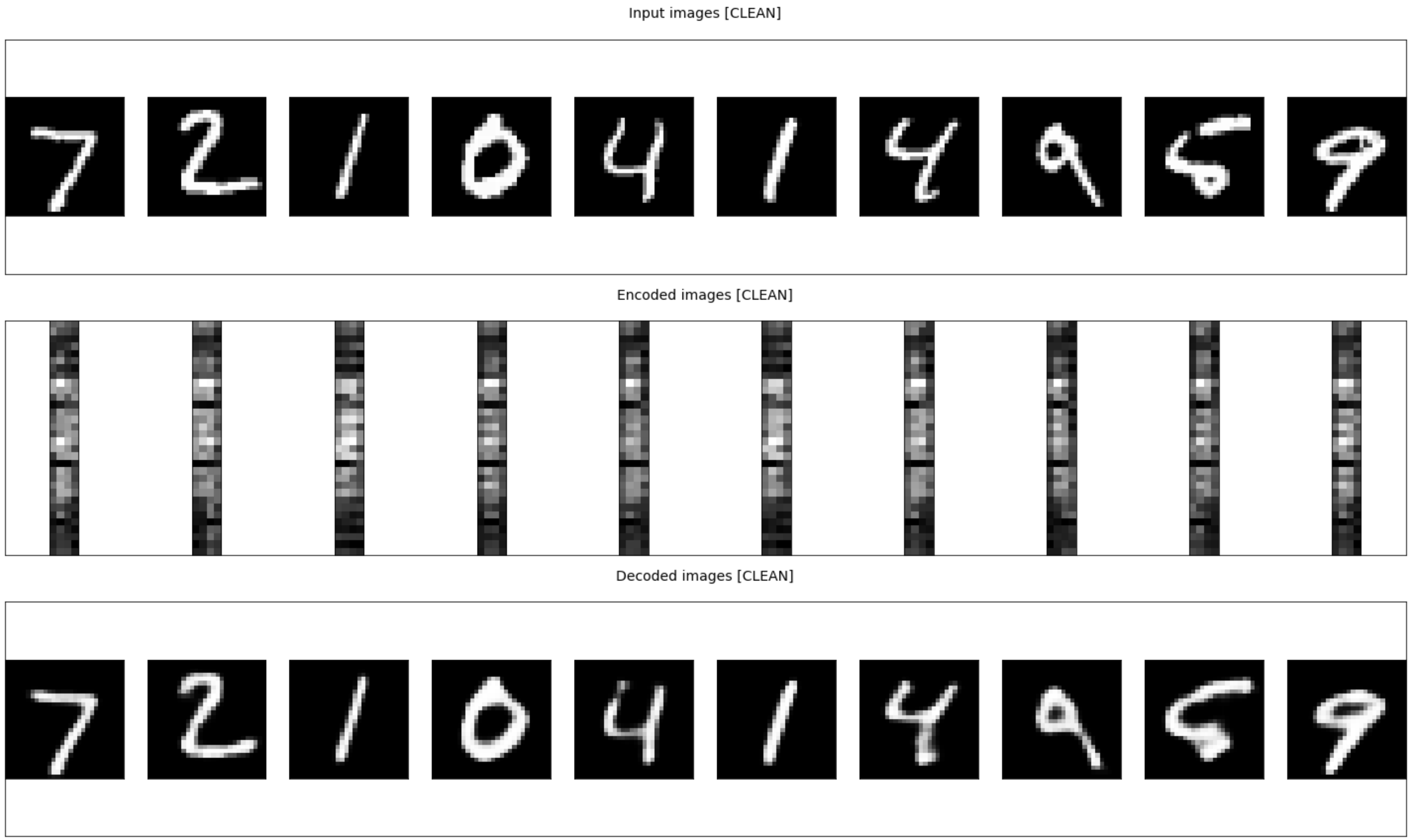
- Creation of a log folder for *Tensorboard*

- **Training** (fitting) the Autoencoder over *x_train*
  - x_train as input
  - x_train as target

- Evaluate **Validation** Error over *x_test*

```
log_dir_clean = "logs/fit_clean/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback_clean = TensorBoard(log_dir=log_dir_clean, histogram_freq=1)
autoencoder_train = autoencoder.fit(x_train, x_train,
                                    epochs=number_of_epochs,
                                    batch_size=batch_size,
                                    shuffle=True,
                                    verbose=verbosity,
                                    validation_data=(x_test, x_test),
                                    callbacks=[tensorboard_callback_clean])
```

Visualization of the MNIST Dataset



De-Noising Application

- Autoencoder can be used to **clear noisy pictures**. Here's a toy application.

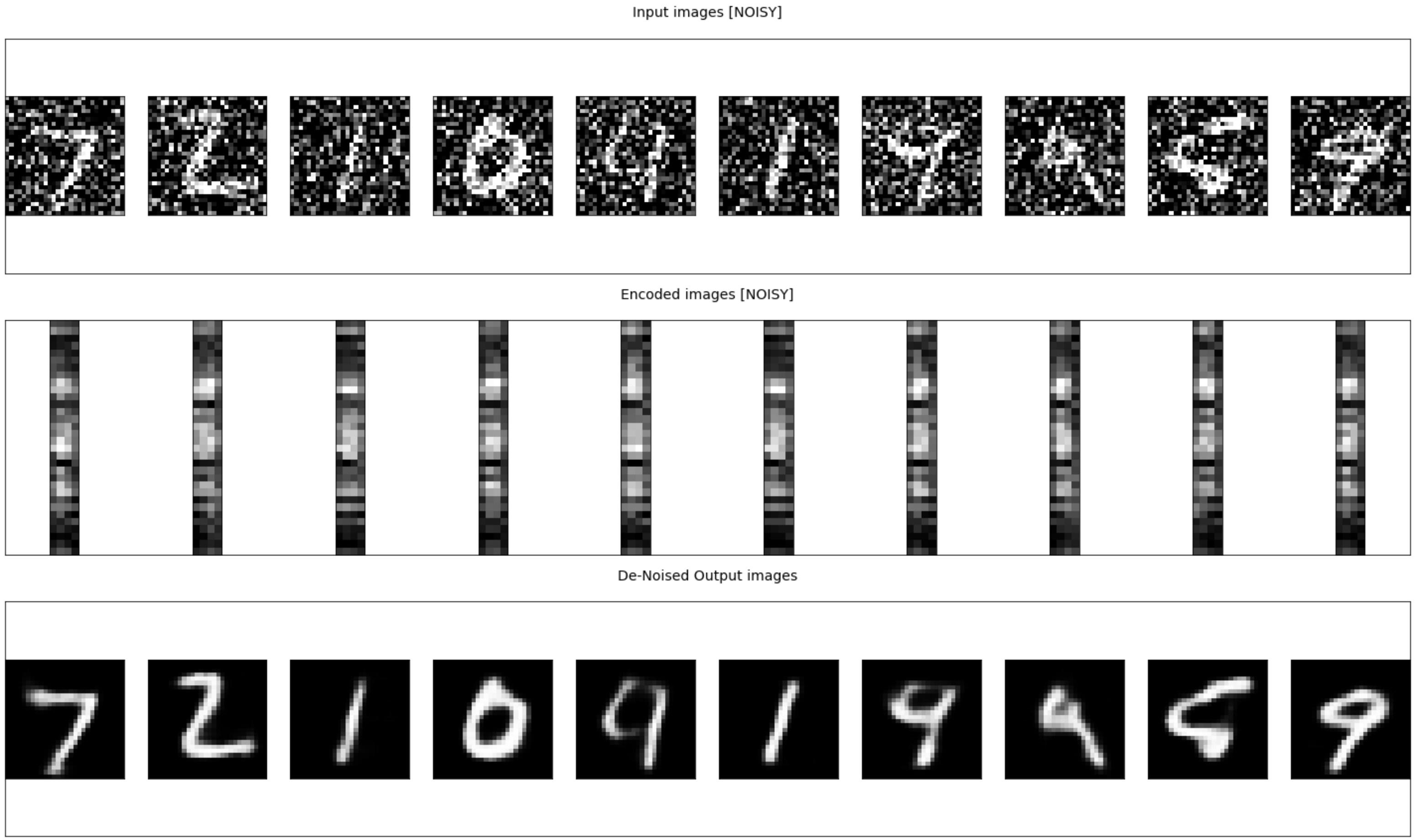
- Generation of **noisy data**
  - Gaussian noise has been added
  - Normalization of noise-affected data

- **Training** of the Autoencoder over *x_train_noisy*
  - x_train_noisy as input
  - x_train as target

- Evaluate **Validation** Error over *x_test_noisy*

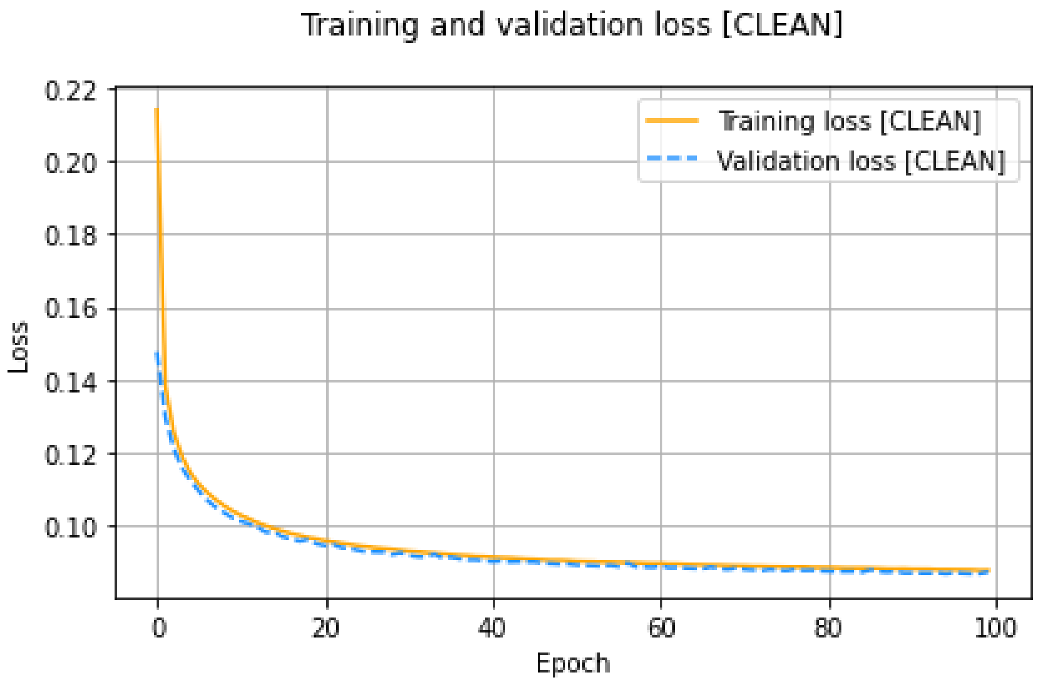
```
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)

x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)

# ========================================================================================== #

log_dir_noisy = "logs/fit_noisy/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback_noisy = TensorBoard(log_dir=log_dir_noisy, histogram_freq=1)
autoencoder_noisy = autoencoder.fit(x_train_noisy, x_train,
                                    epochs=number_of_epochs,
                                    batch_size=batch_size,
                                    shuffle=True,
                                    verbose=verbosity,
                                    validation_data=(x_test_noisy, x_test),
                                    callbacks=[tensorboard_callback_noisy])
```

Visualization of Cleaned Pictures



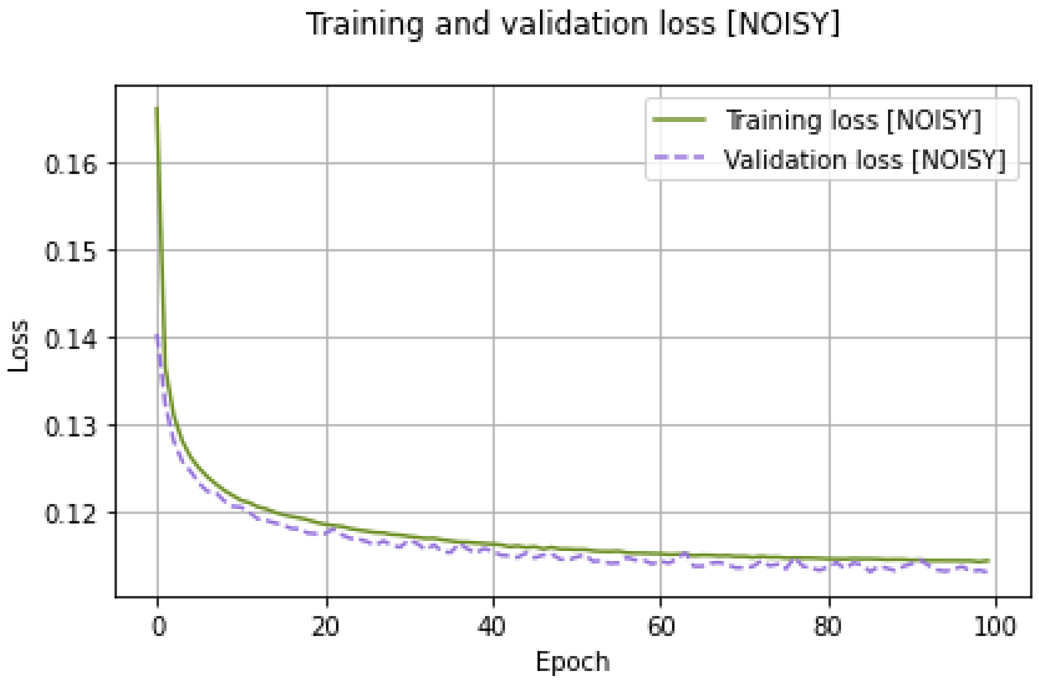
Performance Evaluation 1

- Model **converges** to a loss around 0.087
- Validation Loss and Training Loss are in **sync**
  - Good generalization capabilities as seen in the small gap between both curves
- No overfitting
  - Curves always decreasing



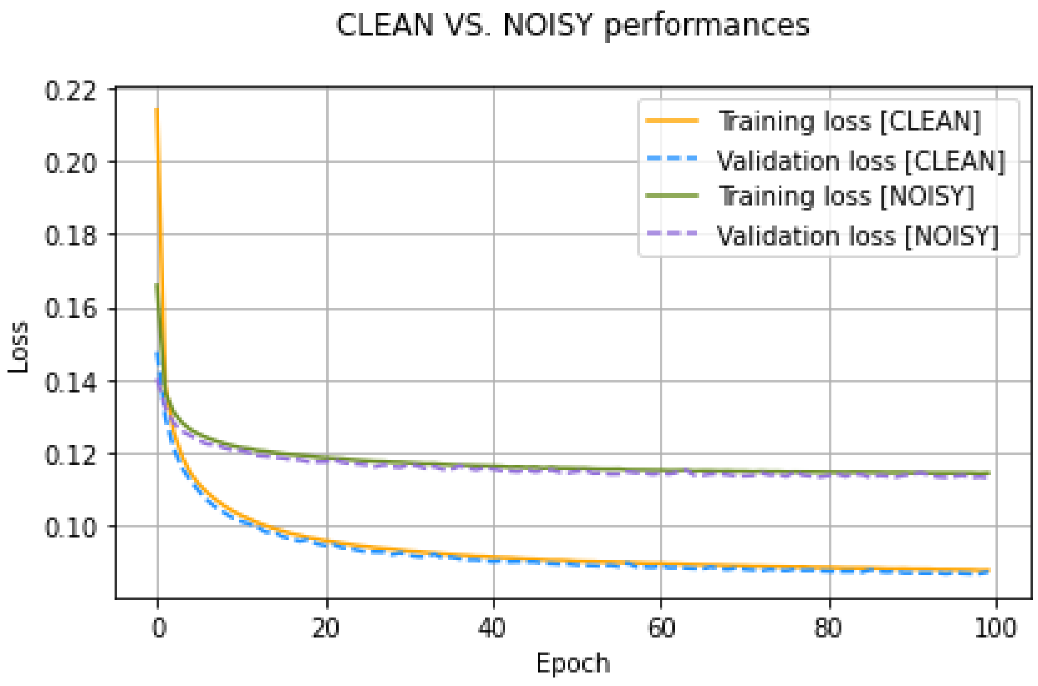
Performance Evaluation 2

- Validation Loss and Training Loss almost in **sync**
- Several **spikes** appear on the Validation Loss
- **Overfitting** at some epochs
  - Improvements can be achieved by increasing complexity



Performance Evaluation 3

- Clean VS Noisy curves are **not so far away**
- Overall, **good generalization** capabilities are shown



Conclusions

- Perks of higher dimensionality
  - Trade between higher entropic capacity and overfitting
- Picture Restoration is a feasable application for the Autoencoder model developed
- Keras VS PyTorch: ease of use VS granularity
- Epochs increasing needs to be evaluated on a case by case basis

Thanks for Your attention