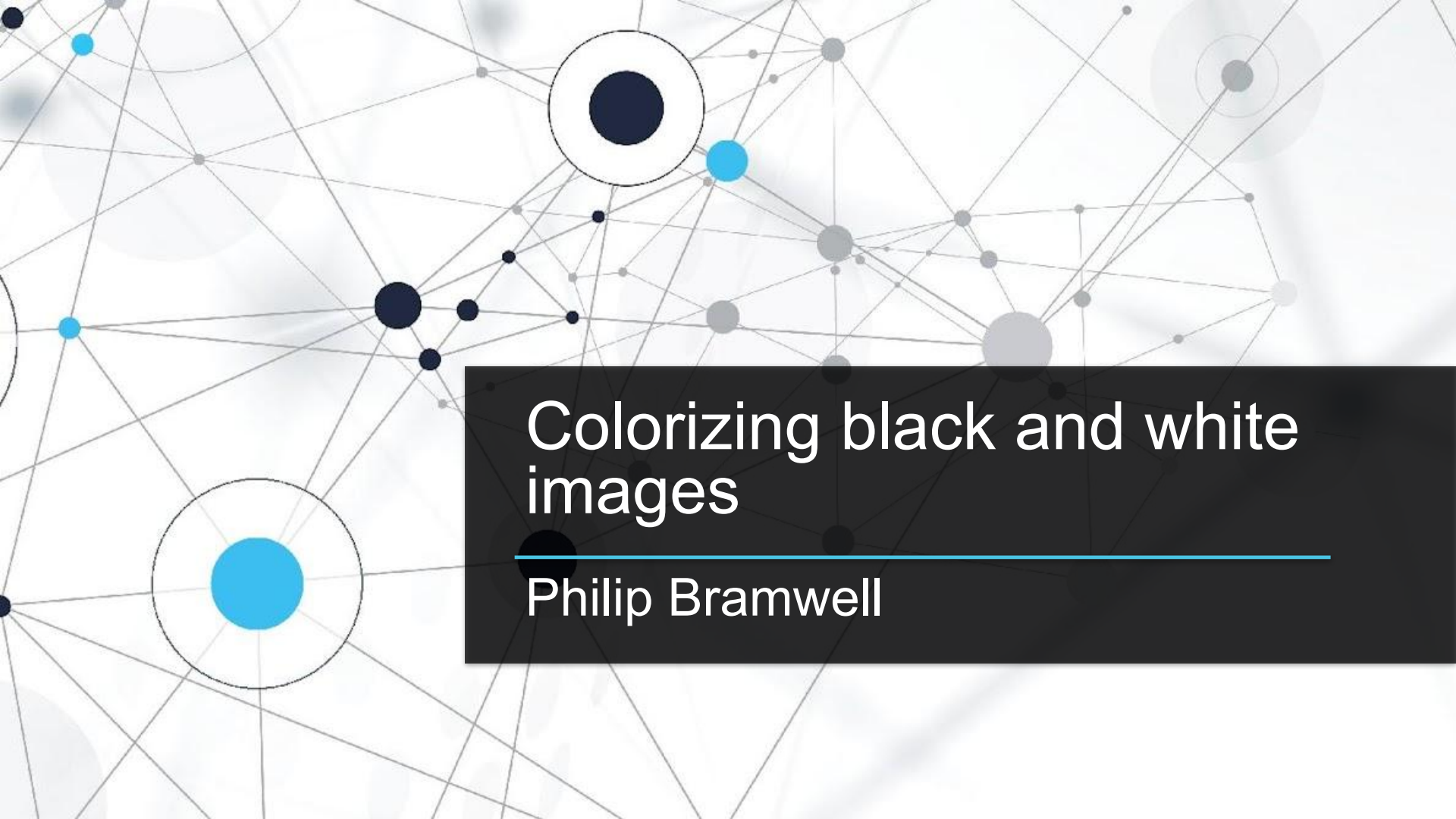


Colorizing black and white images

Philip Bramwell

Agenda

- Introduction
- Project Goals
- Data
- Methods and Models
- Results

Introduction

- Using a neural network to predict RGB values
- Grayscale images as input and RGB images as labels

Project Goals

- Predict acceptable RGB images
- Use the network to predict RGB images for old black and white images that have a similarity to the training/validation data used

Data

- Mainly images of landscapes
- Relatively small dataset, so image augmentation was used
- Although the images mainly show landscapes, there are a few outliers

Data (landscape examples)




Data (outlier example)

Used Methods/Models

- ImageDataGenerator was used to provide the network with training/validation input and labels
- For the training input and label images image augmentation was used

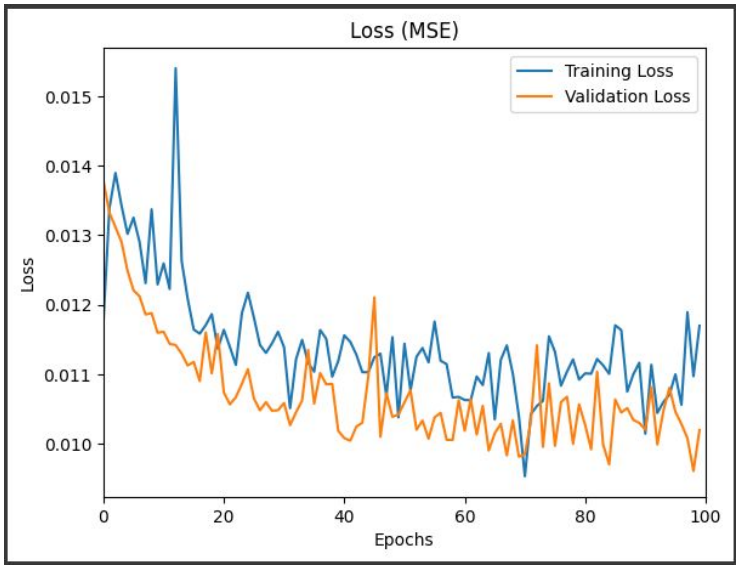
Used Methods/Models

- Baseline model

```
input_layer = Input(shape=input_shape)

# Encoding

conv_1 = Conv2D(64, (3, 3), padding='same', activation='relu')(input_layer)
max_pooling_1 = MaxPooling2D((2, 2))(conv_1)

# Decoding

conv_out = Conv2DTranspose(3, (3, 3), strides=(2, 2), padding='same')(max_pooling_1)

model = tf.keras.Model(inputs=input_layer, outputs=conv_out)
```

Used Methods/Models

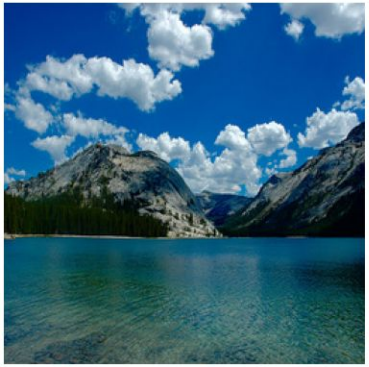
- The baseline model is already able to predict RGB images but the course of the loss is very unstable



Used Methods/Models

- The predicted RGB images of the baseline model are very blurry (no skip connections)






Used Methods/Models

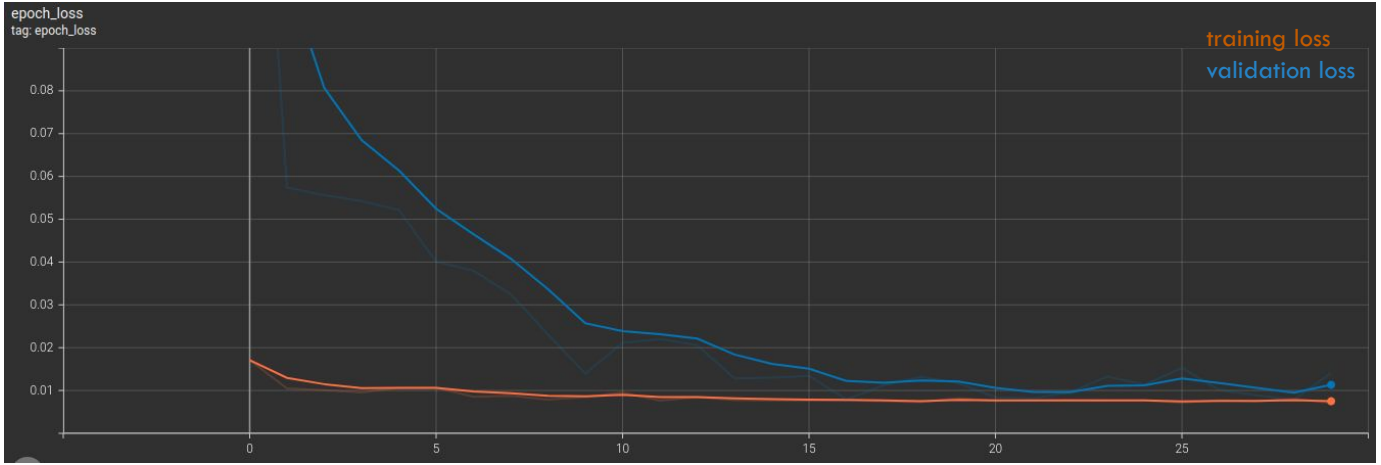
- U-Net model

```
# Encoding

conv_1 = Conv2D(64, (3, 3), padding='same')(input_layer)
batch_norm_1 = BatchNormalization()(conv_1)
relu_1 = ReLU()(batch_norm_1)
max_pooling_1 = MaxPooling2D((2, 2))(relu_1)

conv_2 = Conv2D(128, (3, 3), padding='same')(max_pooling_1)
batch_norm_2 = BatchNormalization()(conv_2)
relu_2 = ReLU()(batch_norm_2)
max_pooling_2 = MaxPooling2D((2, 2))(relu_2)

conv_3 = Conv2D(256, (3, 3), padding='same')(max_pooling_2)
batch_norm_3 = BatchNormalization()(conv_3)
relu_3 = ReLU()(batch_norm_3)
max_pooling_3 = MaxPooling2D((2, 2))(relu_3)

conv_4 = Conv2D(512, (3, 3), padding='same')(max_pooling_3)
batch_norm_4 = BatchNormalization()(conv_4)
relu_4 = ReLU()(batch_norm_4)
max_pooling_4 = MaxPooling2D((2, 2))(relu_4)

conv_5 = Conv2D(1024, (3, 3), padding='same', activation='relu')(max_pooling_4)
batch_norm_5 = BatchNormalization()(conv_5)
relu_5 = ReLU()(batch_norm_5)
```

```
# Decoding

conv_trans_5 = Conv2DTranspose(512, (3, 3), strides=(2, 2), padding='same')(relu_5)
batch_norm_5 = BatchNormalization()(conv_trans_5)
relu_up_5 = ReLU()(batch_norm_5)

concat_4 = Concatenate()([relu_up_5, relu_4])
conv_trans_4 = Conv2DTranspose(256, (3, 3), strides=(2, 2), padding='same')(concat_4)
batch_norm_up_4 = BatchNormalization()(conv_trans_4)
relu_up_4 = ReLU()(batch_norm_up_4)

concat_3 = Concatenate()([relu_up_4, relu_3])
conv_trans_3 = Conv2DTranspose(128, (3, 3), strides=(2, 2), padding='same')(concat_3)
batch_norm_up_3 = BatchNormalization()(conv_trans_3)
relu_up_3 = ReLU()(batch_norm_up_3)

concat_2 = Concatenate()([relu_up_3, relu_2])
conv_trans_2 = Conv2DTranspose(64, (3, 3), strides=(2, 2), padding='same')(concat_2)
batch_norm_up_2 = BatchNormalization()(conv_trans_2)
relu_up_2 = ReLU()(batch_norm_up_2)

concat_1 = Concatenate()([relu_up_2, relu_1])
conv_out = Conv2D(3, (1, 1), padding='same', activation='sigmoid')(concat_1)

model = tf.keras.Model(inputs=input_layer, outputs=conv_out)
```

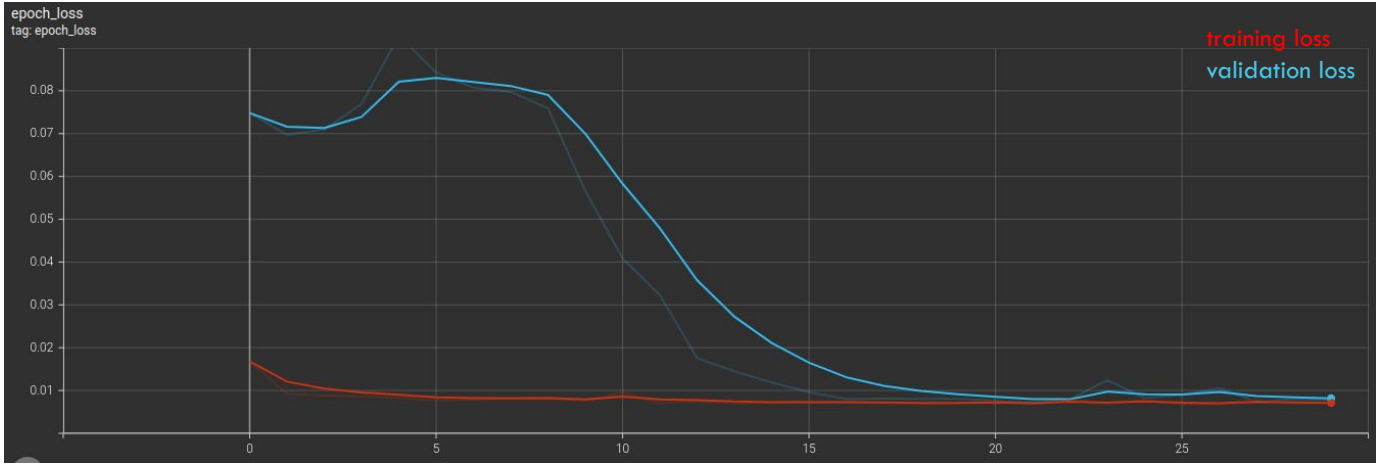
Used Methods/Models

- Learning rate 0.01



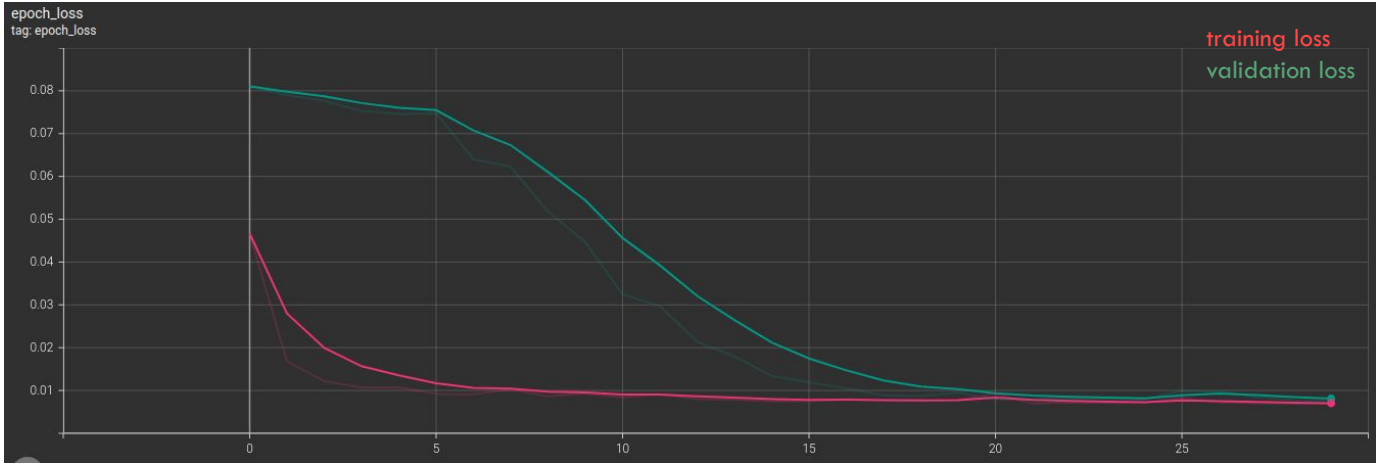
Used Methods/Models

- Learning rate 0.001



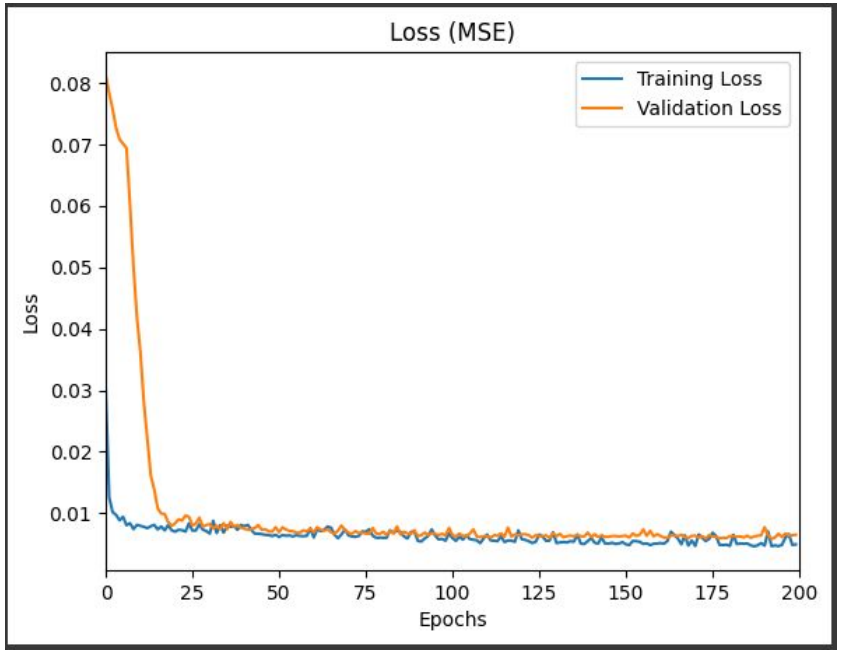
Used Methods/Models

- Learning rate 0.0001



Used Methods/Models

- The U-Net model has a much more stable course of the loss and because skip connection are used the images are sharper

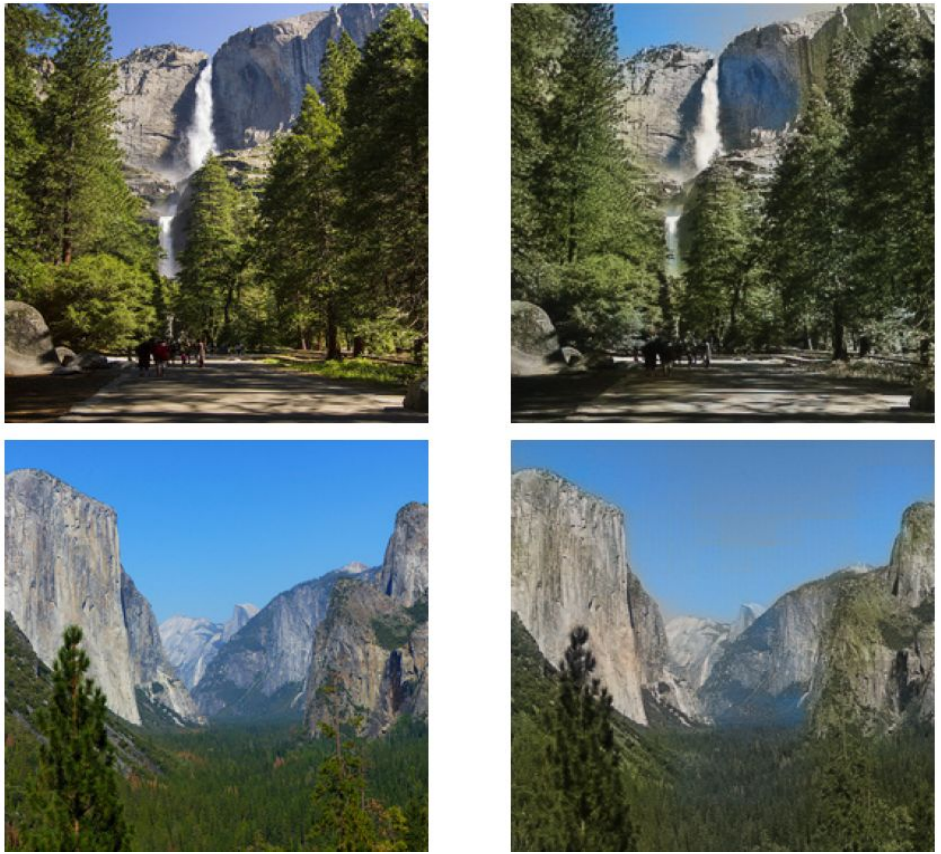


Results

- Baseline model
  - MSE: 0.01020
- U-Net model
  - MSE: 0.00652

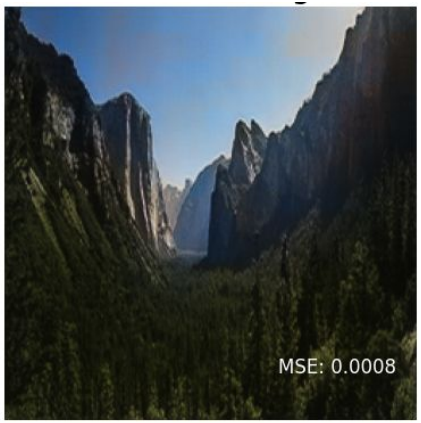
Results

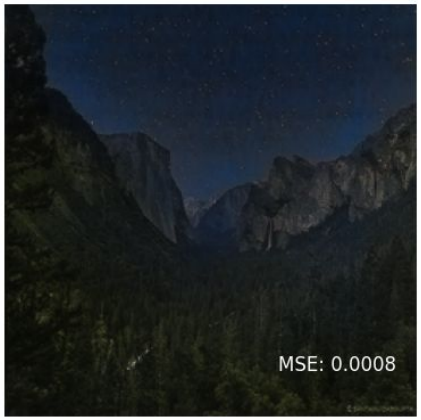
- Some predicted images of the U-Net model



Results

- Predicted images of the U-Net model with the lowest MSE score





Results

- Predicted images of the U-Net model with the highest MSE score

Results

- Predicted images on random web images




Results

- Predicted images on random web images with the best MSE

Biggest Problems

- Getting the ImageDataGenerator to work with grayscale images as a input generator and the original label images as a label generator
- Deciding which specific U-Net architecture so use

Outlook

- Possible Improvements
  - Training the model on LAB color representation rather than RGB images to increase performance
  - Using Generative Adversarial Network (GAN)

Thank you