Image Colorization

Team 6: Crystal Leatvanich, Jeonghee (Christina) Son, Kuang-Ching (Amanda) Ting, Tharfeed Ahmed Unus

Project Overview

Objective

The goal of our project is to perform automatic colorization of black and white images. Specifically, we aim to train a neural network that learns to map grayscale input images to their corresponding colored versions. In this setup, the predictors are the black and white images, and the labels are the fully colored versions of those same images.

Motivation

Colorizing black and white images is an important prediction task with both historical and practical significance. We believe this project could benefit a range of industries, particularly education assistance, historical archives, museums, and the media & film industry. For educational use in the medical field, colorizing images can enhance visualization and offer more accurate diagnostic information for doctors during clinical procedures compared to grayscale images.

In the context of museums and archives, AI powered colorization offers a compelling way to restore and reimagine old photographs, making them more engaging and accessible for modern audiences. Adding color not only enhances visual storytelling but also helps preserve cultural memory in a more immersive and relatable format.

In the media and film industry, this technology is often used to remaster classic films and footage for modern distribution. Whether it's archivists preparing exhibitions or editors bringing old scenes back to life, being able to generate realistic color from grayscale images opens up space for both creative exploration and historical impact.

While image colorization is not a new concept, we believe that there is still a lot to uncover in this domain. Our approach brings a fresh angle in terms of both data and model design. Instead of relying on public datasets, we're planning to use our own set of images and build a full deep learning pipeline from scratch. This allows us to take on a classic challenge in a customized approach.

Data

We began our project by compiling a dataset of 265 images from full scene portraits from our own camera rolls. Later in the process, we introduced another category, the MSBA student headshots (186 images), to explore whether the model could generalize to clean studio-like portraits. The dataset was randomly split into 90/10 for training and validation sets during training. A small number of additional images for each dataset were reserved as a test set to evaluate generalization on unseen photos.

Dataset: [Google Drive]

The results of our work on this project is also available to read as a Medium article, please [click here] to view. The slide deck used to present our work can be found [here].

Initializations and Importations

```
!pip install pillow-heif
```

```
Collecting pillow-heif
  Downloading pillow_heif-0.22.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.6 kB)
Requirement already satisfied: pillow>=10.1.0 in /usr/local/lib/python3.11/dist-packages (from pillow-heif) (11.2.1)
Downloading pillow_heif-0.22.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 42.8 MB/s eta 0:00:00
Installing collected packages: pillow-heif
Successfully installed pillow-heif-0.22.0
```

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from PIL import Image
from pillow_heif import register_heif_opener

from google.colab import drive

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Input, Conv2D, UpSampling2D, BatchNormalization, Concatenate, Lambda, LayerNormalizatior
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

from skimage.color import rgb2lab, lab2rgb
```

```
# Allows for heic/heif (iPhone image formats) to be handled, and declaring acceptable image extensions
register_heif_opener()
image_extensions = ('.png', '.jpg', '.jpeg', '.heic', '.heif')
```

```
drive.mount('/content/drive', force_remount= True)
```

```
Mounted at /content/drive
```

Image Preprocessing and Visualization Functions

Preprocessing Methodology

We first standardized color images, converting them to RGB format to ensure compatibility with JPEG, rotating portrait images to landscape orientation when necessary, and then resized and center-cropped photos to fix the resolution of 176x120 pixels preserving the ratio. This ensured that all images fed into the model had consistent sizing and formatting. After saving the resized images as JPG files, we converted them to grayscale.

The standardized color images were normalized as NumPy arrays, scaling pixel values in the [0,1] range for model training stability. To enhance the diversity of the dataset and reduce the risk of overfitting, data augmentation was applied using TensorFlow's ImageDataGenerator. Augmentation techniques included random rotations, width and height shifts, shearing, zooming, and horizontal flipping. The augmented images were then incorporated into the main dataset getting both datasets to the 400-500 range.

To prepare for model training, we created two types of input/output pairs and a visualization function that displays the grayscale input, the model's predicted color output, and the ground truth color image side by side to easily evaluate the model's performance.

- In the RGB setup, each color image was converted to a grayscale version as the model input and the RGB image as the output target. The model directly predicts an RGB image, so the image visualization was able to display the prediction directly.
- For LAB colorization, images were transformed into the LAB color space, with the lightness channel being the input and the normalized A and B color channels used as output. Grayscale inputs were reshaped appropriately to match the input dimensions. To visualize the predicted colorization, we needed to reconstruct the images and convert from LAB to RGB to show the prediction.

The preprocessing pipeline standardizes, augments, and structures the dataset to provide the best possible foundation for training and evaluating high-quality image colorization models. This section contains the functions used in the preprocessing flow.

```
# This function will a) convert an incoming image to RGB to allow conversion to the JPEG format, b) rotate to landscape if ne
def image_resizer(input_path, output_path, target_size=(720, 480)):
    try:
        img = Image.open(input_path)

        # Converting images to RGB in case they have transparency in them, also makes them compatible with the JPEG format
        img = img.convert('RGB')

        if img.height > img.width:
            img = img.rotate(90, expand=True)
```

```
        # Resize image without distortion or black bars (ideally)
        target_width, target_height = target_size
        img_width, img_height = img.size

        img_ratio = img_width / img_height
        target_ratio = target_width / target_height

        if img_ratio > target_ratio:
            # If the image is wider than our target resolution, scale height to match target
            new_height = target_height
            new_width = int(new_height * img_ratio)
        else:
            # If the image is taller than our target resolution, scale width to match target
            new_width = target_width
            new_height = int(new_width / img_ratio)

        img = img.resize((new_width, new_height), Image.Resampling.LANCZOS)

        left = (new_width - target_width) // 2
        top = (new_height - target_height) // 2
        right = left + target_width
        bottom = top + target_height

        img = img.crop((left, top, right, bottom))

        # Changing final image file name to have a .jpg extension
        base, _ = os.path.splitext(output_path)
        output_path = base + '.jpg'

        img.save(output_path, 'JPEG', quality=100)
        # print(f"Processed: {os.path.basename(input_path)}")
        return True

    except Exception as e:
        print(f"Failed to process {os.path.basename(input_path)}: {e}")
        return False


# This function will help us show the image predictions for the models that process images in the RGB colour space
def show_predictions(model, X, Y, num_samples=5):
    preds = model.predict(X[:num_samples])

    plt.figure(figsize=(15, 5 * num_samples))

    for i in range(num_samples):
        # Grayscale Input
        ax = plt.subplot(num_samples, 3, i * 3 + 1)
        plt.imshow(tf.squeeze(X[i]), cmap='gray')
        ax.set_title("Input (Grayscale)")
        ax.axis("off")

        # Model Prediction
        ax = plt.subplot(num_samples, 3, i * 3 + 2)
        pred_img = preds[i]
        plt.imshow(np.clip(pred_img, 0, 1))
        ax.set_title("Predicted (RGB)")
        ax.axis("off")

        # Ground Truth
        ax = plt.subplot(num_samples, 3, i * 3 + 3)
        plt.imshow(np.clip(Y[i], 0, 1))
        ax.set_title("Ground Truth (RGB)")
        ax.axis("off")

    plt.tight_layout()
    plt.show()


# This function will help us show the image predictions for the models that process images in the LAB colour space
def show_predictions_lab(model, X, Y, num_samples=5):
    preds = model.predict(X[:num_samples])

    plt.figure(figsize=(15, 5 * num_samples))
```

```
    for i in range(num_samples):
        # Grayscale Input
        ax = plt.subplot(num_samples, 3, i * 3 + 1)
        plt.imshow(tf.squeeze(X[i]), cmap='gray')
        ax.set_title("Input (Grayscale)")
        ax.axis("off")

        # Model Prediction
        ax = plt.subplot(num_samples, 3, i * 3 + 2)
        pred_img = preds[i].reshape(120, 176, 2)
        L_channel = X[i].reshape(120, 176, 1)[:,:,0]
        lab_image = np.dstack((L_channel, pred_img * 128))
        rgb_image = lab2rgb(lab_image)
        plt.imshow(np.clip(rgb_image, 0, 1))
        ax.set_title("Predicted (RGB)")
        ax.axis("off")

        # Ground Truth
        ax = plt.subplot(num_samples, 3, i * 3 + 3)
        ground_truth_rgb = lab2rgb(np.dstack((X[i].reshape(120, 176, 1)[:,:,0], Y[i] * 128)))
        plt.imshow(np.clip(ground_truth_rgb, 0, 1))
        ax.set_title("Ground Truth (RGB)")
        ax.axis("off")

    plt.tight_layout()
    plt.show()
```

Methodology

We experimented with a range of different model architectures to tackle the image colorization task, aiming to find the most accurate approach. Detailed later in the notebook are each of the 6 modeling attempts, highlighting how each approach contributes to our understanding of the colorization problem.

Model Evaluation

We used Mean Absolute Error (MAE) and Mean Squared Error (MSE) as both loss functions and evaluation metrics during model training. These metrics are commonly used in image regression tasks because they give us a way to quantify the difference between the predicted and true pixel values. However, we found these values not applicable to our project, as they didn't always reflect how good the results actually looked. We also experimented with custom loss functions such as perceptual loss and hybrid loss, but they performed worse than the standard losses in our case. Ultimately, we relied on the human eye evaluation to judge which model produced the most realistic and natural-looking colorization. While quantitative metrics provided a baseline, it was how natural the colors looked to a human observer that guided our judgement on the prediction quality.

Neural Network Overview

Model 1: Sequential RGB Model

This model has a simple structure that predicts the RGB channels. The process begins with two convolutional layers with 64 filters each, followed by another two convolutional layers with 128 filters. These layers use ReLU activation functions and apply downsampling, which reduces the spatial resolution while increasing the depth to capture more abstract features.

In the decoding stage, the model reverses this process through upsampling, followed by additional convolutional layers. A final Lambda layer is used to resize the output image to match the original input dimensions. Since the model is simple, it is efficient and easy to train and it serves as a strong baseline for comparison with more complex architectures.

Model 2: Simple CNN

This model follows a simple encoder-decoder architecture for image colorization. It's a also simplified version, designed to capture the core concept using only a few convolutional layers. The image first passes through two convolutional layers, which are Conv2D(64) and Conv2D(128), both using ReLU activation and 'same' padding. A MaxPooling2D layer then downsamples the feature map. A single

Conv2D(128) layer serves as a shallow bottleneck. The decoder then upsamples the feature map using Conv2DTranspose and reconstructs the image. The final layer, Conv2D(3) with sigmoid activation, generates the 3-channel RGB output scaled between 0 and 1.

Model 3: Improved CNN

The model first passes the input through an encoder for downsampling, which reduces the spatial dimensions while capturing important features. In the first convolution block, a Conv2D (64) layer applies 64 filters to extract local patterns. This is followed by BatchNormalization to stabilize and speed up training, and a LeakyReLU activation to introduce non-linearity while allowing a small gradient when the output is negative. Finally, MaxPooling2D is applied to reduce the image size by half.

In the second convolution block, the same structure is used, but with 128 filters. After another pooling layer, the image is downsampled to a size of 30x44. At the bottleneck, a deeper Conv2D (256) layer is used to capture more abstract and high-level features. This layer serves as the compressed representation of the image. After the encoder and bottleneck, the model enters the decoder, where it up-samples the image twice using UpSampling2D and convolution layers. This restores the image to its original resolution and predicts pixel-level color in RGB format.

Model 4: U-Net Implementation

We then introduced a U-Net architecture with skip connections between the encoder and decoder to improve both training stability and output quality. First, we added deeper encoder layers to enable the model to extract more complex features. Batch normalization was applied after each convolutional layer to stabilize training and serve as a form of regularization, helping to reduce overfitting. We also incorporated skip connections to pass high-resolution features directly from the encoder to the decoder, allowing the model to better recover spatial details lost during downsampling. Additionally, we deepened the bottleneck and enhanced the decoder blocks to support more accurate and detailed color reconstruction.

Model 5: Sequential LAB Model

This model was designed to predict colors in the LAB color space using a deep sequential autoencoder architecture. The network encodes the input through multiple convolutional layers, compressing it into a lower dimensionality representation, and then decoding it back to image resolution. The final output has a tanh activation to ensure that the predicted values are normalized into the range of [-1,1] to align with the LAB color space's structure.

The training was conducted using the Adam optimizer and the Mean Absolute Error (MAE) loss function, early stopping, and model checkpointing based on validation loss. We had experimented with different learning rates including extremely small values such as 1e-14 and 1e-16. We found that extremely low rates significantly slowed convergence and potentially prevented effective learning. We also tested both MAE and MSE loss function, MAE produced more realistic results.

Model 6: Transformers with VGG

The idea of this model is to use VGG to extract strong spatial features, Transformers to model global context, and the decoder to reconstruct a full-resolution color image. It begins by converting the grayscale input to 3 channels, then passes it through a pretrained VGG16 encoder, which extracts spatial features using filters trained on millions of natural images. Unlike basic U-Nets or autoencoders that rely purely on local convolutional context, this model integrates a Transformer block after VGG to capture global attention and long-range dependencies across the entire image. This allows it to better understand the overall scene and apply colorization based on contexts. Finally, a CNN decoder upsamples the features back to full resolution, producing a RGB output. Compared to U-Net and autoencoder models, this architecture aims to preserve fine image structure via VGG features and enhances global color consistency through attention.

While we are exploring the inclusion of VGG in this project, this was done more in a direction of exploration and learning. We understand that VGG is better suited to object identification, and that this use case requires significant retention of detail which VGG may be unable to help with. However, we would like to explore if the model can roughly identify objects and apply the right/similar colours to the same, in order to open future implementations where such a model is used in parallel with other networks that allow better/more complete retention of detail.

We will now explore the first of our implementations, utilizing the full-scene portrait images to train our neural networks.

Full-Scene Portrait Implementations

Image Preprocessing

```
# Defining the required Google Drive paths for the Full Scene Portrait Implementation
color_image_path = '/content/drive/MyDrive/BA865/BA865 Group Project/Dataset/Portrait/Color'
color_resized_image_path = '/content/drive/MyDrive/BA865/BA865 Group Project/Dataset/Portrait/Color Resized'
test_color_image_path = '/content/drive/MyDrive/BA865/BA865 Group Project/Dataset/Portrait/Test'
test_color_resized_image_path = '/content/drive/MyDrive/BA865/BA865 Group Project/Dataset/Portrait/Test Resized'
best_models_path = '/content/drive/MyDrive/BA865/BA865 Group Project/Best Models Portrait'
```

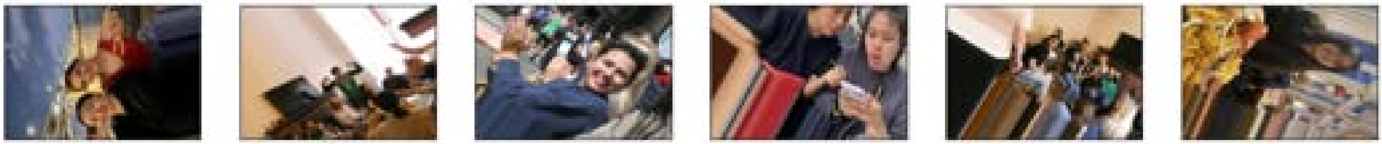
Train Color Image Preprocessing

```
# Creating color resized image folder if it doesn't exist
os.makedirs(color_resized_image_path, exist_ok=True)

# Ensuring the color resized image folder is empty
for file_name in os.listdir(color_resized_image_path):
    file_path = os.path.join(color_resized_image_path, file_name)
    if os.path.isfile(file_path):
        os.remove(file_path)

resize_success_count = 0
resize_fail_count = 0

# Looping through all color images and applying the resizing, cropping, rotation, and compression
for image in os.listdir(color_image_path):
    if image.lower().endswith(image_extensions):
        input_img_path = os.path.join(color_image_path, image)
        output_img_path = os.path.join(color_resized_image_path, image)

        # The below if statement resizes the image while also maintaining a count of successful and unsuccessful conversions
        if image_resizer(input_img_path, output_img_path, (176, 120)):
            resize_success_count += 1
        else:
            resize_fail_count += 1

print(f"Successfully resized {resize_success_count} images.")
print(f"Failed to resize {resize_fail_count} images.")
```

```
Successfully resized 265 images.
Failed to resize 0 images.
```

```
color_images = []

for image in os.listdir(color_resized_image_path):
    color_image_path = os.path.join(color_resized_image_path, image)

    # Converting the color image to numpy array
    if os.path.isfile(color_image_path):
        col_img = Image.open(color_image_path).convert('RGB')
        color_image_array = np.array(col_img).astype('float32') / 255.0
        color_images.append(color_image_array)


# This code will use ImageDataGenerated to create additional images as a form of data augmentation, helping against overfitt
datagen = ImageDataGenerator(rotation_range=40, width_shift_range=0.2, height_shift_range=0.2, rescale=1./255,
                             shear_range=0.2, zoom_range=0.2, horizontal_flip=True, fill_mode='nearest')

datagen.fit(color_images)

image_batches = datagen.flow(np.stack(color_images, axis=0), batch_size=40)

n = 10
plt.figure(figsize= (20,10))

q = 0
for i,batch in enumerate(image_batches):
    for j, image in enumerate(batch):
        if j == 1:
            if i <= 5:
                ax = plt.subplot(1, 6, i+1)
                plt.imshow(tf.keras.utils.array_to_img(image))
                ax.get_xaxis().set_visible(False)
                ax.get_yaxis().set_visible(False)

        # The iterator outputs image data with pixel values between 0-1; we are using values between 0-255 elsewhere. We nee
```

```
        color_images.append(np.multiply(image,255))

    if q > 5:
      break

    q += 1

plt.show()

print(f'We now have a total of {len(color_images)} images.')
```



```
We now have a total of 530 images.
```

```
# Splitting the predictor and labels in the RGB color space, for the training set
X = []
Y = []

for img in color_images:
    try:
        gray = tf.image.rgb_to_grayscale(img)
        X.append(gray.numpy())
        Y.append(img)
    except Exception as e:
        print("error:", e)

X = np.array(X)
Y = np.array(Y)

print("X_rgb shape:", X.shape)
print("Y_rgb shape:", Y.shape)
```

```
X_rgb shape: (530, 120, 176, 1)
Y_rgb shape: (530, 120, 176, 3)
```

```
# Splitting the predictor and labels in the LAB color space, for the training set
X_lab = []
Y_lab = []

for img in color_images:
  try:
      lab = rgb2lab(img)
      X_lab.append(lab[:,:,0])
      #restrict values to be between -1 and 1
      Y_lab.append(lab[:,:,1:] / 128)
  except Exception as e:
     print("error:", e)

X_lab = np.array(X_lab)
Y_lab = np.array(Y_lab)

#dimensions to be the same for X and Y
X_lab = X_lab.reshape(X_lab.shape+(1,))

print("X_lab shape:", X_lab.shape)
print("Y_lab shape:", Y_lab.shape)
```

```
X_lab shape: (530, 120, 176, 1)
Y_lab shape: (530, 120, 176, 2)
```

Test Color Images Preprocessing

```
# Creating color resized image folder for test images if it doesn't exist
os.makedirs(test_color_resized_image_path, exist_ok=True)
```

```
# Ensuring the color resized image folder is empty
for file_name in os.listdir(test_color_resized_image_path):
    file_path = os.path.join(test_color_resized_image_path, file_name)
    if os.path.isfile(file_path):
        os.remove(file_path)

test_resize_success_count = 0
test_resize_fail_count = 0

# Looping through all color images and applying the resizing, cropping, rotation, and compression
for test_image in os.listdir(test_color_image_path):
    if test_image.lower().endswith(image_extensions):
        input_img_path = os.path.join(test_color_image_path, test_image)
        output_img_path = os.path.join(test_color_resized_image_path, test_image)

        # The below if statement resizes the image while also maintaining a count of successful and unsuccessful conversions
        if image_resizer(input_img_path, output_img_path, (176, 120)):
            test_resize_success_count += 1
        else:
            test_resize_fail_count += 1

print(f"Successfully resized {test_resize_success_count} images.")
print(f"Failed to resize {test_resize_fail_count} images.")
```

```
Successfully resized 6 images.
Failed to resize 0 images.
```

```
test_color_images = []

for image in os.listdir(test_color_resized_image_path):
    test_color_image_path = os.path.join(test_color_resized_image_path, image)

    # Converting the color image to numpy array
    if os.path.isfile(test_color_image_path):
        col_img = Image.open(test_color_image_path).convert('RGB')
        test_color_image_array = np.array(col_img).astype('float32') / 255.0
        test_color_images.append(test_color_image_array)
```

```
# Splitting the predictor and labels in the LAB color space, for the test set
X_lab_test = []
Y_lab_test = []

for img in test_color_images:
  try:
      lab_test = rgb2lab(img)
      X_lab_test.append(lab_test[:,:,0])
      #restrict values to be between -1 and 1
      Y_lab_test.append(lab_test[:,:,1:] / 128)
  except Exception as e:
     print("error:", e)

X_lab_test = np.array(X_lab_test)
Y_lab_test = np.array(Y_lab_test)

#dimensions to be the same for X and Y
X_lab_test = X_lab_test.reshape(X_lab_test.shape+(1,))

print("X_lab_test shape:", X_lab_test.shape)
print("Y_lab_test shape:", Y_lab_test.shape)
```

```
X_lab_test shape: (6, 120, 176, 1)
Y_lab_test shape: (6, 120, 176, 2)
```

```
# Splitting the predictor and labels in the RGB color space, for the test set
X_test = []
Y_test = []

for img in test_color_images:
    try:
        gray = tf.image.rgb_to_grayscale(img)
        X_test.append(gray.numpy())
        Y_test.append(img)
```

```
    except Exception as e:
        print("error:", e)

X_test = np.array(X_test)
Y_test = np.array(Y_test)

print("X_rgb_test shape:", X_test.shape)  # (num_images, 120, 176, 1)
print("Y_rgb_test shape:", Y_test.shape)  # (num_images, 120, 176, 3)
```

```
X_rgb_test shape: (6, 120, 176, 1)
Y_rgb_test shape: (6, 120, 176, 3)
```

```
# Defining the early stopping callback, standard across all models
early_stop = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True)
```

Model 1: Sequential RGB Model

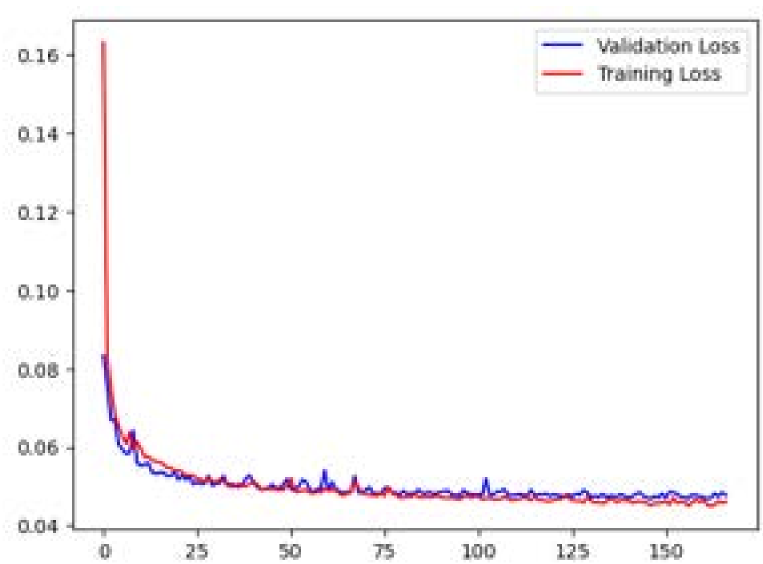
```
# Defining the architecture for the sequential RGB model
input_l = Input(shape=(120, 176, 1))

x = Conv2D(64, (3, 3), activation='relu', padding='same')(input_l)
x = Conv2D(64, (3, 3), activation='relu', strides=2, padding='same')(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = Conv2D(128, (3, 3), activation='relu', strides=2, padding='same')(x)

x = UpSampling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)

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

output_rgb = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)
output_rgb = Lambda(lambda x: tf.image.resize_with_crop_or_pad(x, 120, 176))(output_rgb)

seq_rgb_model = Model(inputs=input_l, outputs=output_rgb)
seq_rgb_model.compile(optimizer='adam', loss='mae', metrics=['mse'])
```

```
# Fitting the model to the training data
history_seq_rgb = seq_rgb_model.fit(X, Y, batch_size=16, epochs=200, validation_split=0.1, callbacks=[early_stop])
```

```
Epoch 1/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 25s 466ms/step - loss: 0.2063 - mse: 0.0617 - val_loss: 0.0833 - val_mse: 0.0132
Epoch 2/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 21s 61ms/step - loss: 0.0849 - mse: 0.0138 - val_loss: 0.0755 - val_mse: 0.0111
Epoch 3/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - loss: 0.0768 - mse: 0.0114 - val_loss: 0.0669 - val_mse: 0.0090
Epoch 4/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - loss: 0.0697 - mse: 0.0098 - val_loss: 0.0669 - val_mse: 0.0089
Epoch 5/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - loss: 0.0664 - mse: 0.0091 - val_loss: 0.0608 - val_mse: 0.0078
Epoch 6/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 62ms/step - loss: 0.0630 - mse: 0.0083 - val_loss: 0.0594 - val_mse: 0.0074
Epoch 7/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - loss: 0.0612 - mse: 0.0079 - val_loss: 0.0583 - val_mse: 0.0073
Epoch 8/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - loss: 0.0636 - mse: 0.0082 - val_loss: 0.0585 - val_mse: 0.0071
Epoch 9/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - loss: 0.0597 - mse: 0.0075 - val_loss: 0.0642 - val_mse: 0.0082
Epoch 10/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - loss: 0.0619 - mse: 0.0080 - val_loss: 0.0558 - val_mse: 0.0068
Epoch 11/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - loss: 0.0597 - mse: 0.0076 - val_loss: 0.0553 - val_mse: 0.0068
Epoch 12/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - loss: 0.0567 - mse: 0.0070 - val_loss: 0.0556 - val_mse: 0.0068
Epoch 13/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - loss: 0.0582 - mse: 0.0074 - val_loss: 0.0560 - val_mse: 0.0069
Epoch 14/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - loss: 0.0567 - mse: 0.0070 - val_loss: 0.0536 - val_mse: 0.0065
Epoch 15/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - loss: 0.0566 - mse: 0.0070 - val_loss: 0.0532 - val_mse: 0.0064
Epoch 16/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - loss: 0.0556 - mse: 0.0069 - val_loss: 0.0535 - val_mse: 0.0064
Epoch 17/200
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 59ms/step - loss: 0.0541 - mse: 0.0065 - val_loss: 0.0536 - val_mse: 0.0065
Epoch 18/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 60ms/step - loss: 0.0545 - mse: 0.0066 - val_loss: 0.0526 - val_mse: 0.0064
Epoch 19/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - loss: 0.0566 - mse: 0.0071 - val_loss: 0.0527 - val_mse: 0.0063
Epoch 20/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - loss: 0.0539 - mse: 0.0066 - val_loss: 0.0537 - val_mse: 0.0065
Epoch 21/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 63ms/step - loss: 0.0556 - mse: 0.0069 - val_loss: 0.0517 - val_mse: 0.0061
Epoch 22/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - loss: 0.0541 - mse: 0.0067 - val_loss: 0.0528 - val_mse: 0.0064
Epoch 23/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - loss: 0.0512 - mse: 0.0061 - val_loss: 0.0517 - val_mse: 0.0062
Epoch 24/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 63ms/step - loss: 0.0528 - mse: 0.0065 - val_loss: 0.0525 - val_mse: 0.0062
Epoch 25/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 64ms/step - loss: 0.0514 - mse: 0.0060 - val_loss: 0.0505 - val_mse: 0.0060
Epoch 26/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - loss: 0.0515 - mse: 0.0062 - val_loss: 0.0512 - val_mse: 0.0061
Epoch 27/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - loss: 0.0514 - mse: 0.0062 - val_loss: 0.0506 - val_mse: 0.0060
Epoch 28/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 63ms/step - loss: 0.0501 - mse: 0.0059 - val_loss: 0.0507 - val_mse: 0.0060
Epoch 29/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - loss: 0.0526 - mse: 0.0064 - val_loss: 0.0527 - val_mse: 0.0062
```

```
# Plotting the training and validation loss
plt.plot(history_seq_rgb.history['val_loss'],c="b")
plt.plot(history_seq_rgb.history['loss'],c="r")
plt.legend(['Validation Loss','Training Loss'])
plt.show()
```



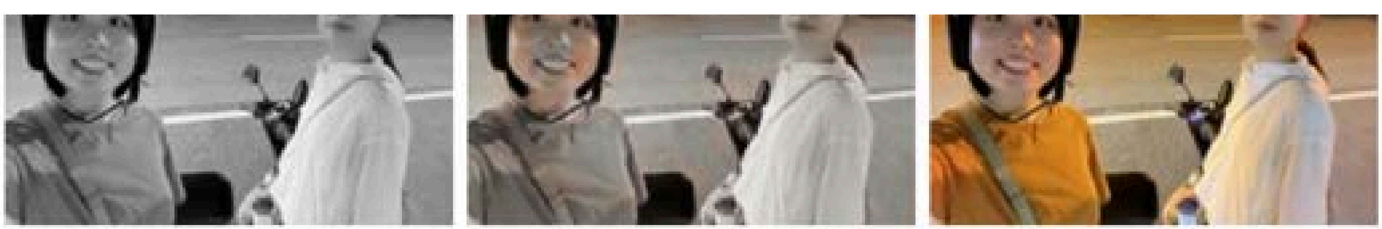
Depiction of Model Performance on Images from the Training Set

```
show_predictions(seq_rgb_model, X, Y, num_samples=5)
```

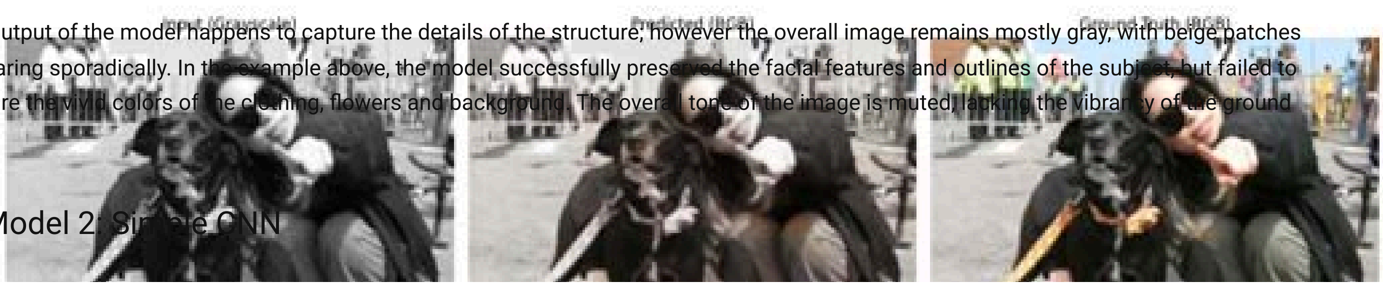
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 664ms/step
```

Depiction of Model Performance on Images from the Test Set

```
show_predictions(seq_rgb_model, X_test, Y_test, num_samples=5)
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
```



The output of the model happens to capture the details of the structure, however the overall image remains mostly gray, with beige patches appearing sporadically. In the example above, the model successfully preserved the facial features and outlines of the subject, but failed to capture the vivid colors of the clothing, flowers and background. The overall tone of the image is muted, lacking the vibrancy of the ground truth.

Model 2: Simple CNN

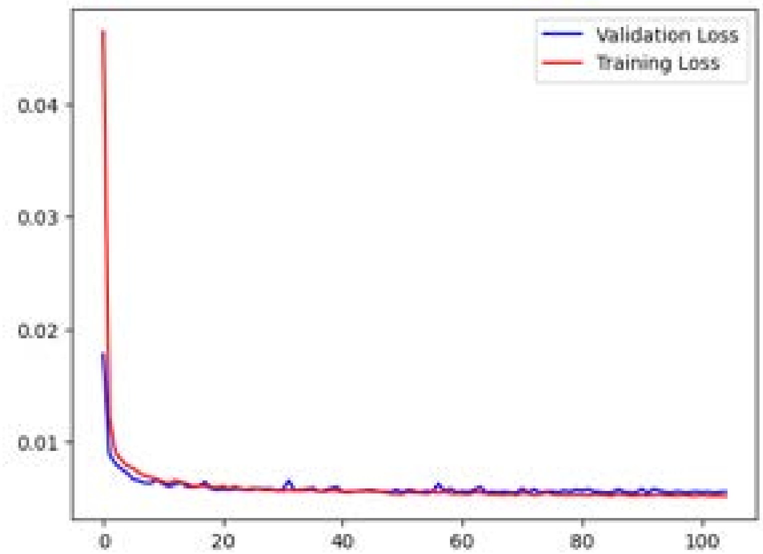
```
# Defining the architecture for the simple CNN model
def build_simple_cnn(input_shape=(120, 176, 1)):
    inputs = layers.Input(shape=input_shape)

    x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = layers.MaxPooling2D((2, 2))(x)

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

    x = layers.Conv2DTranspose(64, (2, 2), strides=2, padding='same')(x)
```

```
    x = layers.ReLU()(x)
    outputs = layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)

    model = models.Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model

model_simple_cnn = build_simple_cnn()


# Defining the path to save the best model
model_checkpoint_path = os.path.join(best_models_path, 'simple_cnn.keras')
model_checkpoint = ModelCheckpoint(model_checkpoint_path, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

# Fitting the model to the training data
history_simple_cnn = model_simple_cnn.fit(X, Y, batch_size=16, epochs=200, validation_split=0.1, callbacks=[early_stop, mode

best_simple_cnn = keras.models.load_model(os.path.join(best_models_path, 'simple_cnn.keras'))
```

```
Epoch 1/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 310ms/step - loss: 0.0632 - mae: 0.2099
Epoch 1: val_loss improved from inf to 0.01775, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/Best
30/30 ━━━━━━━━━━━━━━━━━━━━ 25s 498ms/step - loss: 0.0627 - mae: 0.2086 - val_loss: 0.0177 - val_mae: 0.1050
Epoch 2/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - loss: 0.0138 - mae: 0.0879
Epoch 2: val_loss improved from 0.01775 to 0.00890, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 21s 79ms/step - loss: 0.0138 - mae: 0.0877 - val_loss: 0.0089 - val_mae: 0.0677
Epoch 3/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step - loss: 0.0094 - mae: 0.0684
Epoch 3: val_loss improved from 0.00890 to 0.00803, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 78ms/step - loss: 0.0094 - mae: 0.0684 - val_loss: 0.0080 - val_mae: 0.0640
Epoch 4/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step - loss: 0.0084 - mae: 0.0650
Epoch 4: val_loss improved from 0.00803 to 0.00758, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 78ms/step - loss: 0.0084 - mae: 0.0649 - val_loss: 0.0076 - val_mae: 0.0607
Epoch 5/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 0.0078 - mae: 0.0622
Epoch 5: val_loss improved from 0.00758 to 0.00720, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 80ms/step - loss: 0.0078 - mae: 0.0622 - val_loss: 0.0072 - val_mae: 0.0594
Epoch 6/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - loss: 0.0077 - mae: 0.0610
Epoch 6: val_loss improved from 0.00720 to 0.00667, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 77ms/step - loss: 0.0077 - mae: 0.0610 - val_loss: 0.0067 - val_mae: 0.0558
Epoch 7/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 0.0076 - mae: 0.0602
Epoch 7: val_loss improved from 0.00667 to 0.00653, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 78ms/step - loss: 0.0076 - mae: 0.0601 - val_loss: 0.0065 - val_mae: 0.0548
Epoch 8/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 0.0071 - mae: 0.0579
Epoch 8: val_loss improved from 0.00653 to 0.00631, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 79ms/step - loss: 0.0071 - mae: 0.0578 - val_loss: 0.0063 - val_mae: 0.0538
Epoch 9/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 0.0067 - mae: 0.0562
Epoch 9: val_loss improved from 0.00631 to 0.00630, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 76ms/step - loss: 0.0067 - mae: 0.0562 - val_loss: 0.0063 - val_mae: 0.0530
Epoch 10/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 0.0067 - mae: 0.0562
Epoch 10: val_loss did not improve from 0.00630
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 77ms/step - loss: 0.0067 - mae: 0.0562 - val_loss: 0.0067 - val_mae: 0.0568
Epoch 11/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - loss: 0.0069 - mae: 0.0565
Epoch 11: val_loss improved from 0.00630 to 0.00622, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 81ms/step - loss: 0.0069 - mae: 0.0565 - val_loss: 0.0062 - val_mae: 0.0522
Epoch 12/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - loss: 0.0063 - mae: 0.0533
Epoch 12: val_loss improved from 0.00622 to 0.00600, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 77ms/step - loss: 0.0063 - mae: 0.0533 - val_loss: 0.0060 - val_mae: 0.0512
Epoch 13/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - loss: 0.0065 - mae: 0.0549
Epoch 13: val_loss did not improve from 0.00600
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 75ms/step - loss: 0.0065 - mae: 0.0549 - val_loss: 0.0062 - val_mae: 0.0523
Epoch 14/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - loss: 0.0061 - mae: 0.0529
Epoch 14: val_loss did not improve from 0.00600
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 75ms/step - loss: 0.0061 - mae: 0.0530 - val_loss: 0.0064 - val_mae: 0.0554
Epoch 15/200
```

```
# Plotting the training and validation loss
plt.plot(history_simple_cnn.history['val_loss'],c="b")
```

```
plt.plot(history_simple_cnn.history['loss'],c="r")
plt.legend(['Validation Loss','Training Loss'])
plt.show()
```



Depiction of Model Performance on Images from the Training Set

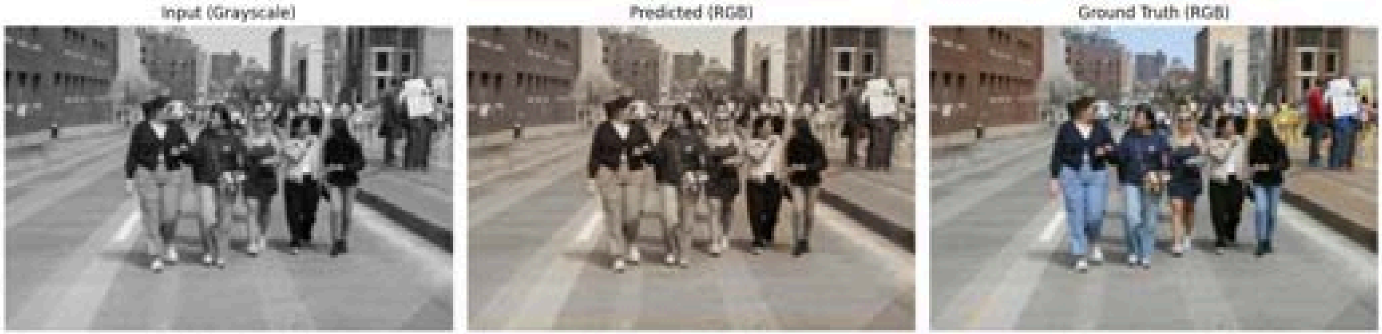
```
show_predictions(best_simple_cnn, X, Y, num_samples=5)
```

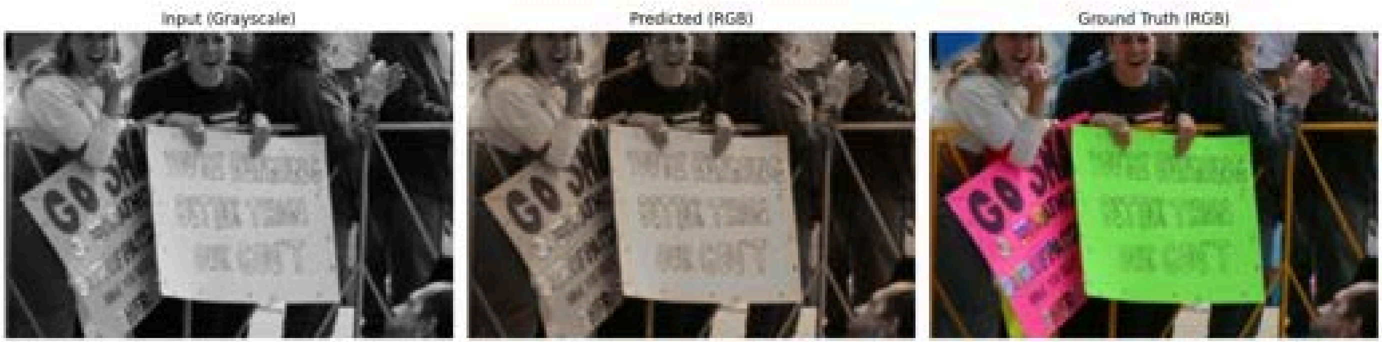
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 269ms/step
```

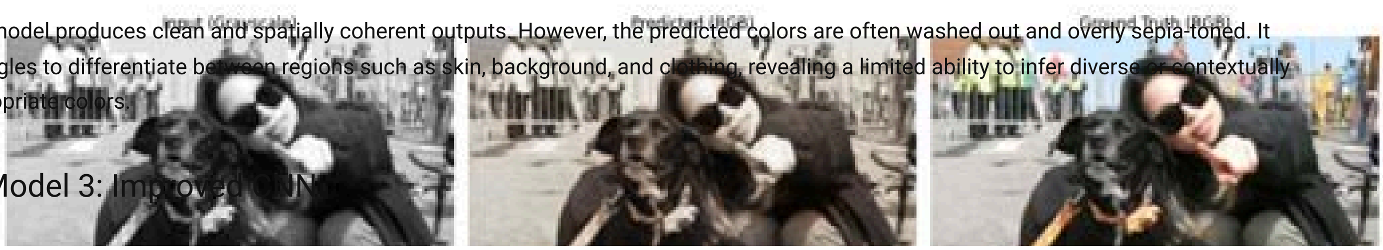
Depiction of Model Performance on Images from the Test Set

```
show_predictions(best_simple_cnn, X_test, Y_test, num_samples=5)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
```







The model produces clean and spatially coherent outputs. However, the predicted colors are often washed out and overly sepia-toned. It struggles to differentiate between regions such as skin, background, and clothing, revealing a limited ability to infer diverse or contextually appropriate colors.

Model 3: Improved CNN

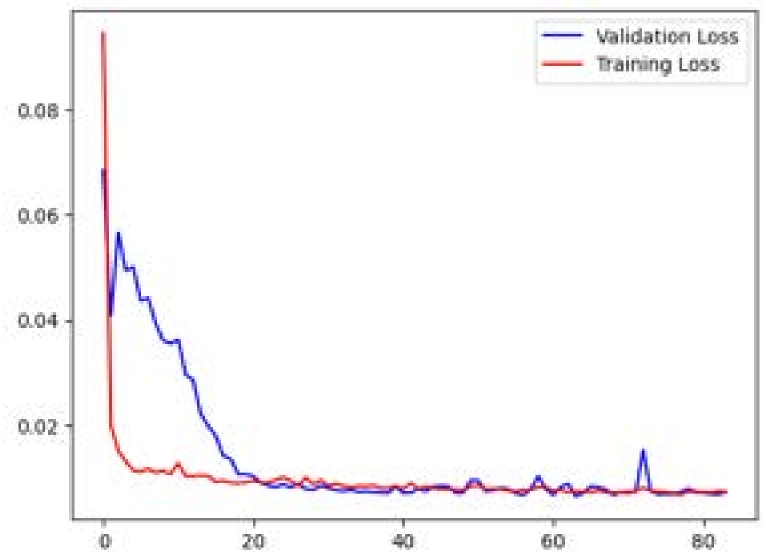
```
# Defining the architecture for the improved CNN model
def build_improved_cnn():
    inputs = layers.Input(shape=(120, 176, 1), name="grayscale_input")
    x1 = layers.Conv2D(64, (3, 3), padding='same')(inputs)
    x1 = layers.BatchNormalization()(x1)
    x1 = layers.LeakyReLU(negative_slope=0.1)(x1)
    p1 = layers.MaxPooling2D((2, 2))(x1)
    x2 = layers.Conv2D(128, (3, 3), padding='same')(p1)
    x2 = layers.BatchNormalization()(x2)
    x2 = layers.LeakyReLU(negative_slope=0.1)(x2)
    p2 = layers.MaxPooling2D((2, 2))(x2)
    x = layers.Conv2D(256, (3, 3), padding='same')(p2)
```

```
    x = layers.BatchNormalization()(x)
    x = layers.LeakyReLU(negative_slope=0.1)(x)
    x = layers.UpSampling2D((2, 2))(x)
    x = layers.Conv2D(128, (3, 3), padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.LeakyReLU(negative_slope=0.1)(x)
    x = layers.UpSampling2D((2, 2))(x)
    x = layers.Conv2D(64, (3, 3), padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.LeakyReLU(negative_slope=0.1)(x)
    outputs = layers.Conv2D(3, (3, 3), activation='tanh', padding='same', name="ab_output")(x)
    return models.Model(inputs=inputs, outputs=outputs, name="ColorizationModel")


improved_cnn = build_improved_cnn()
improved_cnn.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Defining the path to save the best model
model_checkpoint_path = os.path.join(best_models_path, 'improved_cnn.keras')
model_checkpoint = ModelCheckpoint(model_checkpoint_path, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

# Fitting the model to the training data
history_improved_cnn = improved_cnn.fit(X, Y, batch_size=16, epochs=1000, validation_split=0.1, callbacks=[early_stop, mode

best_improved_cnn = keras.models.load_model(os.path.join(best_models_path, 'improved_cnn.keras'))
```

```
Epoch 1/1000
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 275ms/step - loss: 0.2068 - mae: 0.3204
Epoch 1: val_loss improved from inf to 0.06849, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/Best
30/30 ━━━━━━━━━━━━━━━━━━━━ 19s 388ms/step - loss: 0.2032 - mae: 0.3168 - val_loss: 0.0685 - val_mae: 0.2137
Epoch 2/1000
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step - loss: 0.0220 - mae: 0.1142
Epoch 2: val_loss improved from 0.06849 to 0.04086, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 9s 105ms/step - loss: 0.0220 - mae: 0.1140 - val_loss: 0.0409 - val_mae: 0.1714
Epoch 3/1000
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - loss: 0.0154 - mae: 0.0944
Epoch 3: val_loss did not improve from 0.04086
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 99ms/step - loss: 0.0154 - mae: 0.0943 - val_loss: 0.0567 - val_mae: 0.1982
Epoch 4/1000
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step - loss: 0.0145 - mae: 0.0909
Epoch 4: val_loss did not improve from 0.04086
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 97ms/step - loss: 0.0145 - mae: 0.0907 - val_loss: 0.0494 - val_mae: 0.1882
Epoch 5/1000
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - loss: 0.0112 - mae: 0.0780
Epoch 5: val_loss did not improve from 0.04086
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 99ms/step - loss: 0.0112 - mae: 0.0781 - val_loss: 0.0502 - val_mae: 0.1890
Epoch 6/1000
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step - loss: 0.0113 - mae: 0.0781
Epoch 6: val_loss did not improve from 0.04086
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 98ms/step - loss: 0.0113 - mae: 0.0780 - val_loss: 0.0436 - val_mae: 0.1776
Epoch 7/1000
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step - loss: 0.0117 - mae: 0.0803
Epoch 7: val_loss did not improve from 0.04086
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 96ms/step - loss: 0.0117 - mae: 0.0803 - val_loss: 0.0442 - val_mae: 0.1783
Epoch 8/1000
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - loss: 0.0109 - mae: 0.0777
Epoch 8: val_loss improved from 0.04086 to 0.03930, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 103ms/step - loss: 0.0109 - mae: 0.0777 - val_loss: 0.0393 - val_mae: 0.1681
Epoch 9/1000
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step - loss: 0.0110 - mae: 0.0785
Epoch 9: val_loss improved from 0.03930 to 0.03606, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 102ms/step - loss: 0.0110 - mae: 0.0786 - val_loss: 0.0361 - val_mae: 0.1609
Epoch 10/1000
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - loss: 0.0102 - mae: 0.0754
Epoch 10: val_loss improved from 0.03606 to 0.03551, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 106ms/step - loss: 0.0103 - mae: 0.0755 - val_loss: 0.0355 - val_mae: 0.1600
Epoch 11/1000
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step - loss: 0.0133 - mae: 0.0881
Epoch 11: val_loss did not improve from 0.03551
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 97ms/step - loss: 0.0132 - mae: 0.0881 - val_loss: 0.0363 - val_mae: 0.1613
Epoch 12/1000
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step - loss: 0.0113 - mae: 0.0796
Epoch 12: val_loss improved from 0.03551 to 0.02960, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 102ms/step - loss: 0.0112 - mae: 0.0795 - val_loss: 0.0296 - val_mae: 0.1453
Epoch 13/1000
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step - loss: 0.0097 - mae: 0.0728
Epoch 13: val_loss improved from 0.02960 to 0.02856, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 102ms/step - loss: 0.0097 - mae: 0.0729 - val_loss: 0.0286 - val_mae: 0.1424
Epoch 14/1000
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - loss: 0.0097 - mae: 0.0727
Epoch 14: val_loss improved from 0.02856 to 0.02211, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 105ms/step - loss: 0.0097 - mae: 0.0728 - val_loss: 0.0221 - val_mae: 0.1239
Epoch 15/1000
```

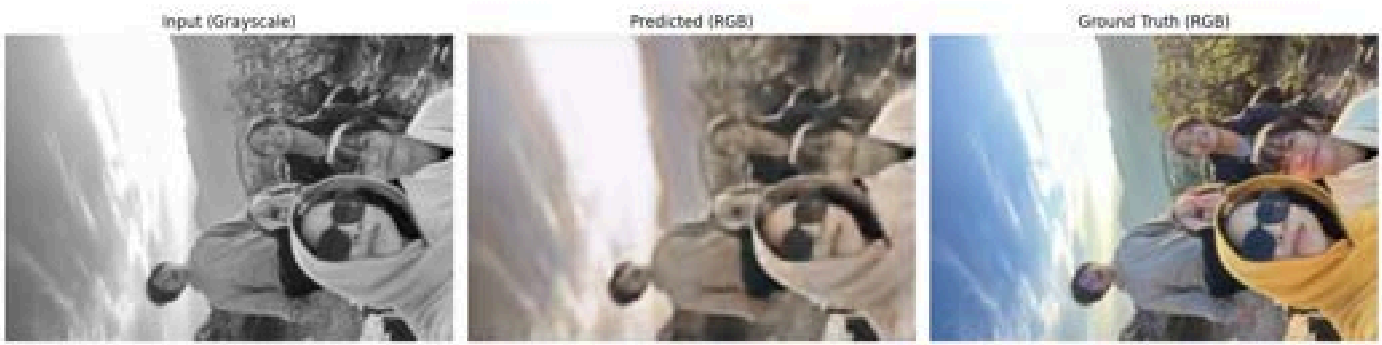
```
# Plotting the training and validation loss
plt.plot(history_improved_cnn.history['val_loss'],c="b")
plt.plot(history_improved_cnn.history['loss'],c="r")
plt.legend(['Validation Loss','Training Loss'])
plt.show()
```

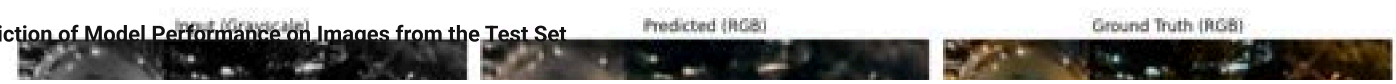


Depiction of Model Performance on Images from the Training Set

```
show_predictions(best_improved_cnn, X, Y, num_samples=5)
```

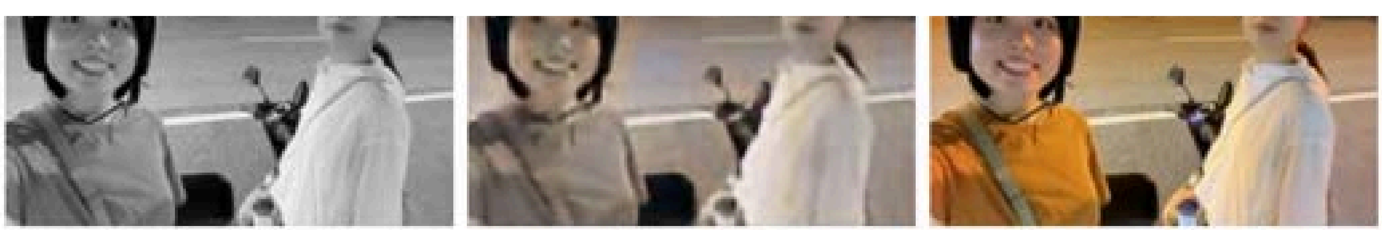
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 964ms/step
```



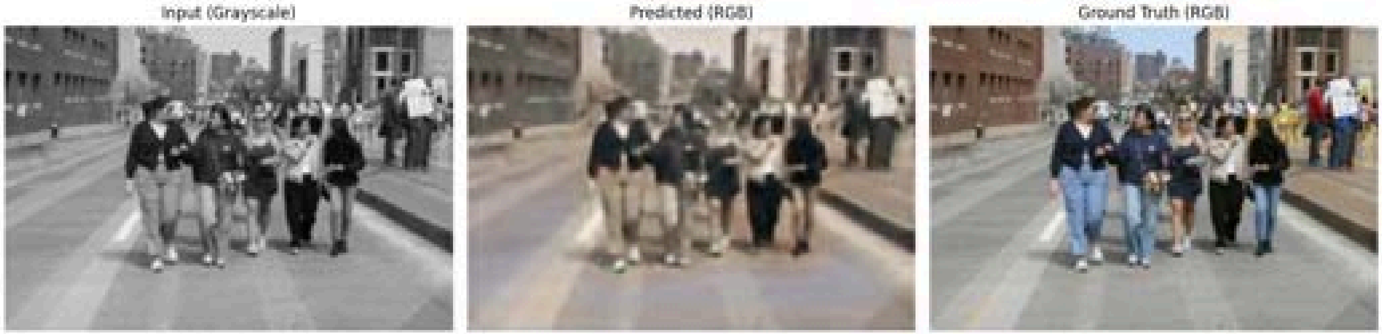


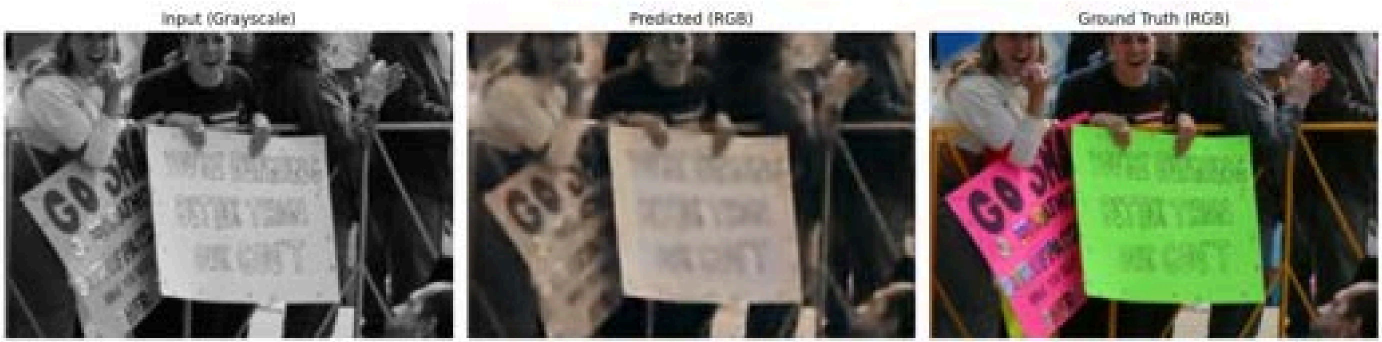
Depiction of Model Performance on Images from the Test Set

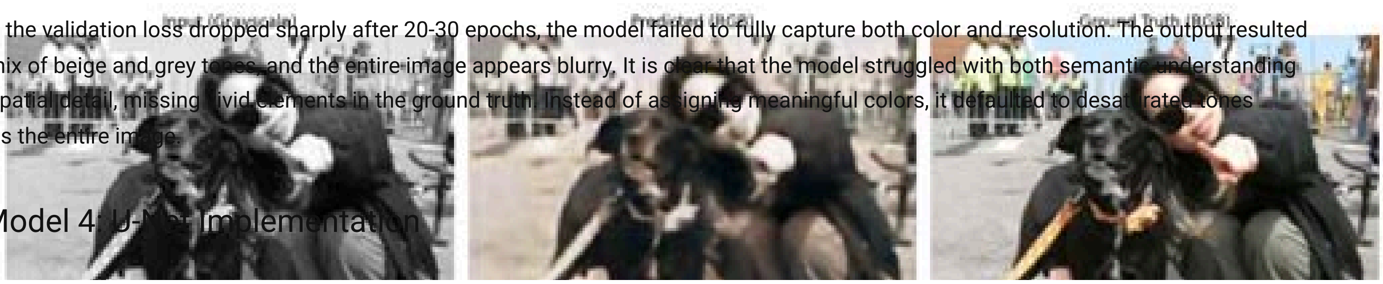
```
show_predictions(best_improved_cnn, X_test, Y_test, num_samples=5)
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
```







While the validation loss dropped sharply after 20-30 epochs, the model failed to fully capture both color and resolution. The output resulted in a mix of beige and grey tones, and the entire image appears blurry. It is clear that the model struggled with both semantic understanding and spatial detail, missing vivid elements in the ground truth. Instead of assigning meaningful colors, it defaulted to desaturated tones across the entire image.

Model 4: U-Net Implementation

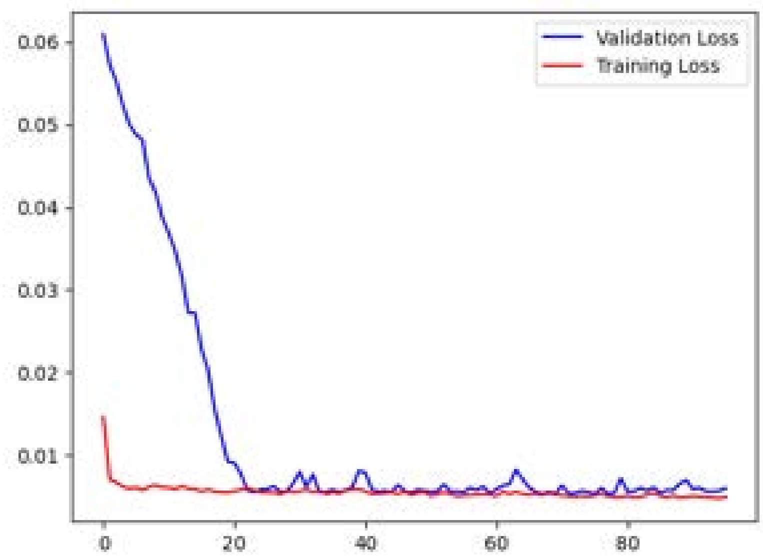
```
# Defining the architecture for the U-Net model
def build_unet_model(input_shape=(120, 176, 1)):
    inputs = layers.Input(shape=input_shape)

    # Encoder
    conv1 = layers.Conv2D(64, (3, 3), padding='same')(inputs)
    conv1 = layers.BatchNormalization()(conv1)
    conv1 = layers.ReLU()(conv1)
    conv1 = layers.Conv2D(64, (3, 3), padding='same')(conv1)
    conv1 = layers.BatchNormalization()(conv1)
    conv1 = layers.ReLU()(conv1)
```

```
    pool1 = layers.MaxPooling2D((2, 2))(conv1)

    conv2 = layers.Conv2D(128, (3, 3), padding='same')(pool1)
    conv2 = layers.BatchNormalization()(conv2)
    conv2 = layers.ReLU()(conv2)
    conv2 = layers.Conv2D(128, (3, 3), padding='same')(conv2)
    conv2 = layers.BatchNormalization()(conv2)
    conv2 = layers.ReLU()(conv2)
    pool2 = layers.MaxPooling2D((2, 2))(conv2)

    # Bottleneck
    bottleneck = layers.Conv2D(256, (3, 3), padding='same')(pool2)
    bottleneck = layers.BatchNormalization()(bottleneck)
    bottleneck = layers.ReLU()(bottleneck)

    # Decoder
    up2 = layers.Conv2DTranspose(128, (2, 2), strides=2, padding='same')(bottleneck)
    up2 = layers.Concatenate()([up2, conv2])
    up2 = layers.Conv2D(128, (3, 3), padding='same')(up2)
    up2 = layers.BatchNormalization()(up2)
    up2 = layers.ReLU()(up2)

    up1 = layers.Conv2DTranspose(64, (2, 2), strides=2, padding='same')(up2)
    up1 = layers.Concatenate()([up1, conv1])
    up1 = layers.Conv2D(64, (3, 3), padding='same')(up1)
    up1 = layers.BatchNormalization()(up1)
    up1 = layers.ReLU()(up1)

    outputs = layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')(up1)

    model = models.Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model

unet_model = build_unet_model()


# Defining the path to save the best model
model_checkpoint_path = os.path.join(best_models_path, 'unet_model.keras')
model_checkpoint = ModelCheckpoint(model_checkpoint_path, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

# fitting the model to the training data
history_unet_model = unet_model.fit(X, Y, batch_size=16, epochs=200, validation_split=0.1, callbacks=[early_stop, model_chec

best_unet_model = keras.models.load_model(os.path.join(best_models_path, 'unet_model.keras'))
```

```
Epoch 1/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 569ms/step - loss: 0.0297 - mae: 0.1139
Epoch 1: val_loss improved from inf to 0.06086, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/Best
30/30 ━━━━━━━━━━━━━━━━━━━━ 42s 715ms/step - loss: 0.0292 - mae: 0.1128 - val_loss: 0.0609 - val_mae: 0.2085
Epoch 2/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 136ms/step - loss: 0.0069 - mae: 0.0585
Epoch 2: val_loss improved from 0.06086 to 0.05717, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 150ms/step - loss: 0.0069 - mae: 0.0585 - val_loss: 0.0572 - val_mae: 0.2022
Epoch 3/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 136ms/step - loss: 0.0068 - mae: 0.0573
Epoch 3: val_loss improved from 0.05717 to 0.05512, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 150ms/step - loss: 0.0068 - mae: 0.0573 - val_loss: 0.0551 - val_mae: 0.1991
Epoch 4/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 138ms/step - loss: 0.0057 - mae: 0.0524
Epoch 4: val_loss improved from 0.05512 to 0.05224, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 156ms/step - loss: 0.0057 - mae: 0.0524 - val_loss: 0.0522 - val_mae: 0.1942
Epoch 5/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 138ms/step - loss: 0.0057 - mae: 0.0508
Epoch 5: val_loss improved from 0.05224 to 0.04996, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 152ms/step - loss: 0.0057 - mae: 0.0509 - val_loss: 0.0500 - val_mae: 0.1901
Epoch 6/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 138ms/step - loss: 0.0058 - mae: 0.0526
Epoch 6: val_loss improved from 0.04996 to 0.04879, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 158ms/step - loss: 0.0058 - mae: 0.0527 - val_loss: 0.0488 - val_mae: 0.1879
Epoch 7/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step - loss: 0.0059 - mae: 0.0519
Epoch 7: val_loss improved from 0.04879 to 0.04815, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 154ms/step - loss: 0.0059 - mae: 0.0518 - val_loss: 0.0481 - val_mae: 0.1871
Epoch 8/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step - loss: 0.0059 - mae: 0.0540
Epoch 8: val_loss improved from 0.04815 to 0.04351, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 154ms/step - loss: 0.0060 - mae: 0.0541 - val_loss: 0.0435 - val_mae: 0.1778
```

```
Epoch 9/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step - loss: 0.0071 - mae: 0.0586
Epoch 9: val_loss improved from 0.04351 to 0.04177, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 157ms/step - loss: 0.0070 - mae: 0.0586 - val_loss: 0.0418 - val_mae: 0.1747
Epoch 10/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step - loss: 0.0057 - mae: 0.0523
Epoch 10: val_loss improved from 0.04177 to 0.03875, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 155ms/step - loss: 0.0058 - mae: 0.0524 - val_loss: 0.0388 - val_mae: 0.1681
Epoch 11/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step - loss: 0.0063 - mae: 0.0549
Epoch 11: val_loss improved from 0.03875 to 0.03687, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 159ms/step - loss: 0.0063 - mae: 0.0549 - val_loss: 0.0369 - val_mae: 0.1644
Epoch 12/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step - loss: 0.0063 - mae: 0.0551
Epoch 12: val_loss improved from 0.03687 to 0.03466, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 157ms/step - loss: 0.0063 - mae: 0.0550 - val_loss: 0.0347 - val_mae: 0.1593
Epoch 13/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step - loss: 0.0061 - mae: 0.0541
Epoch 13: val_loss improved from 0.03466 to 0.03167, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 156ms/step - loss: 0.0061 - mae: 0.0541 - val_loss: 0.0317 - val_mae: 0.1522
Epoch 14/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 143ms/step - loss: 0.0059 - mae: 0.0524
Epoch 14: val_loss improved from 0.03167 to 0.02722, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 162ms/step - loss: 0.0059 - mae: 0.0524 - val_loss: 0.0272 - val_mae: 0.1398
Epoch 15/200
```

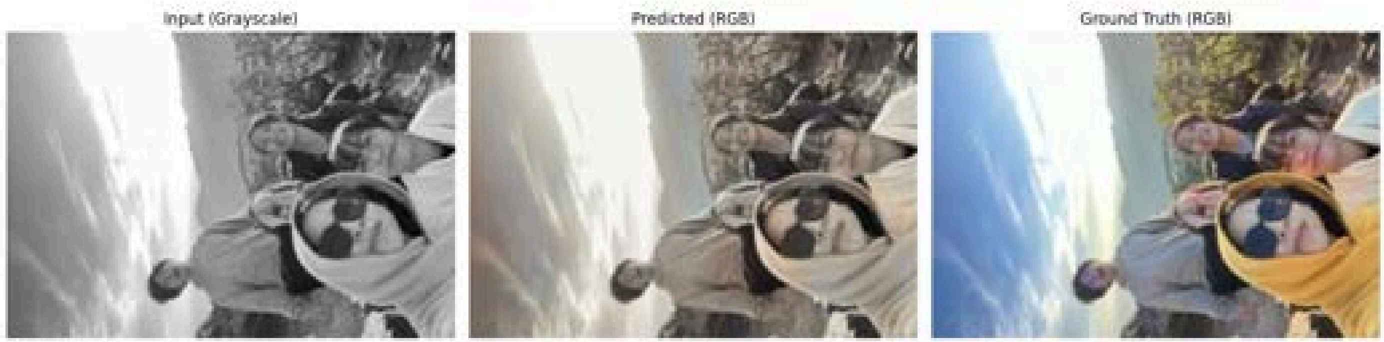
```
# Plotting the training and validation loss
plt.plot(history_unet_model.history['val_loss'],c="b")
plt.plot(history_unet_model.history['loss'],c="r")
plt.legend(['Validation Loss','Training Loss'])
plt.show()
```



Depiction of Model Performance on Images from the Training Set

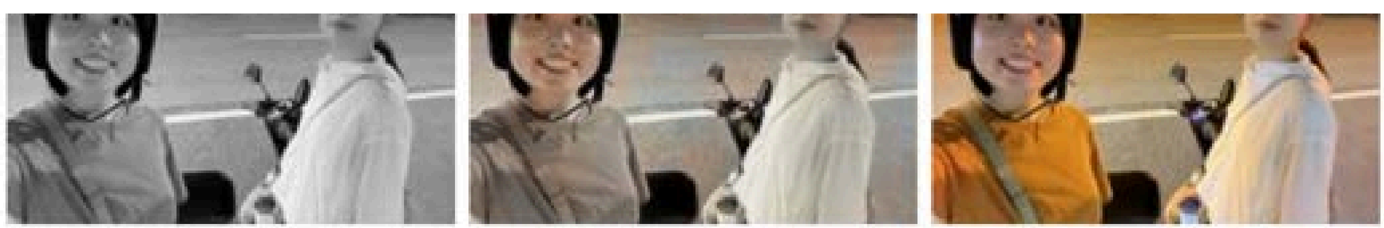
```
show_predictions(best_unet_model, X, Y, num_samples=5)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 997ms/step
```

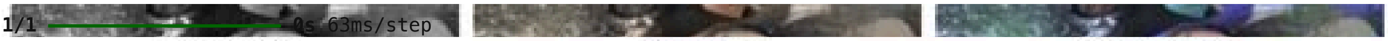


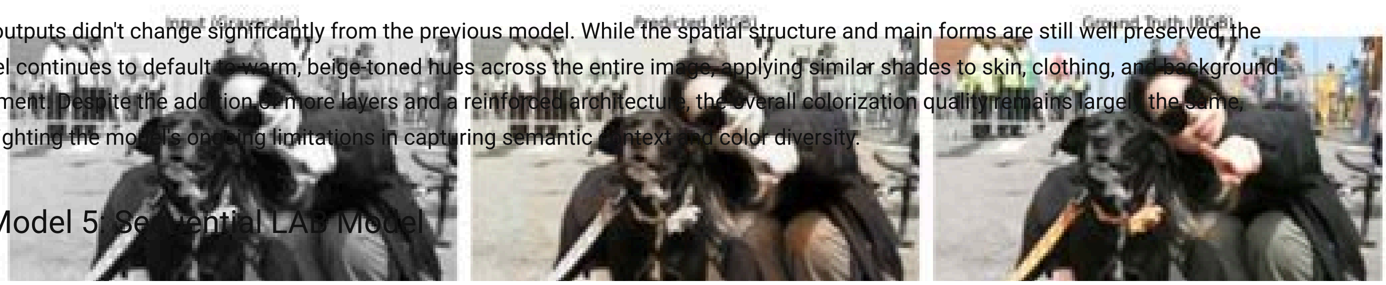
Depiction of Model Performance on Images from the Test Set

```
show_predictions(best_unet_model, X_test, Y_test, num_samples=5)
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
```





The outputs didn't change significantly from the previous model. While the spatial structure and main forms are still well preserved, the model continues to default to warm, beige-toned hues across the entire image, applying similar shades to skin, clothing, and background pavement. Despite the addition of more layers and a reinforced architecture, the overall colorization quality remains largely the same, highlighting the model's ongoing limitations in capturing semantic context and color diversity.

Model 5: Sequential LAB Model

```
# Defining the architecture for the sequential LAB model
# Encoder
seq_lab_model = Sequential()
seq_lab_model.add(Conv2D(64, (3, 3), activation='relu', padding='same', strides=2, input_shape=(120, 176, 1)))
seq_lab_model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
seq_lab_model.add(Conv2D(128, (3,3), activation='relu', padding='same', strides=2))
seq_lab_model.add(Conv2D(256, (3,3), activation='relu', padding='same'))
seq_lab_model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
seq_lab_model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
seq_lab_model.add(Conv2D(256, (3,3), activation='relu', padding='same', strides=2))
seq_lab_model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
```

```
seq_lab_model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
seq_lab_model.add(Conv2D(256, (3,3), activation='relu', padding='same'))
# Decoder
seq_lab_model.add(Conv2D(128, (3,3), activation='relu', padding='same'))
seq_lab_model.add(UpSampling2D((2, 2)))
seq_lab_model.add(Conv2D(64, (3,3), activation='relu', padding='same'))
seq_lab_model.add(UpSampling2D((2, 2)))
seq_lab_model.add(Conv2D(32, (3,3), activation='relu', padding='same'))
seq_lab_model.add(Conv2D(16, (3,3), activation='relu', padding='same'))
seq_lab_model.add(Conv2D(2, (3, 3), activation='tanh', padding='same'))
seq_lab_model.add(UpSampling2D((2, 2)))
seq_lab_model.compile(optimizer=Adam(learning_rate=1e-4), loss='mae', metrics=['mse'])
seq_lab_model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `ir
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d_24 (Conv2D)	(None, 60, 88, 64)	640
conv2d_25 (Conv2D)	(None, 60, 88, 128)	73,856
conv2d_26 (Conv2D)	(None, 30, 44, 128)	147,584
conv2d_27 (Conv2D)	(None, 30, 44, 256)	295,168
conv2d_28 (Conv2D)	(None, 30, 44, 512)	1,180,160
conv2d_29 (Conv2D)	(None, 30, 44, 512)	2,359,808
conv2d_30 (Conv2D)	(None, 15, 22, 256)	1,179,904
conv2d_31 (Conv2D)	(None, 15, 22, 512)	1,180,160
conv2d_32 (Conv2D)	(None, 15, 22, 512)	2,359,808
conv2d_33 (Conv2D)	(None, 15, 22, 256)	1,179,904
conv2d_34 (Conv2D)	(None, 15, 22, 128)	295,040
up_sampling2d_4 (UpSampling2D)	(None, 30, 44, 128)	0
conv2d_35 (Conv2D)	(None, 30, 44, 64)	73,792
up_sampling2d_5 (UpSampling2D)	(None, 60, 88, 64)	0
conv2d_36 (Conv2D)	(None, 60, 88, 32)	18,464
conv2d_37 (Conv2D)	(None, 60, 88, 16)	4,624
conv2d_38 (Conv2D)	(None, 60, 88, 2)	290
up_sampling2d_6 (UpSampling2D)	(None, 120, 176, 2)	0

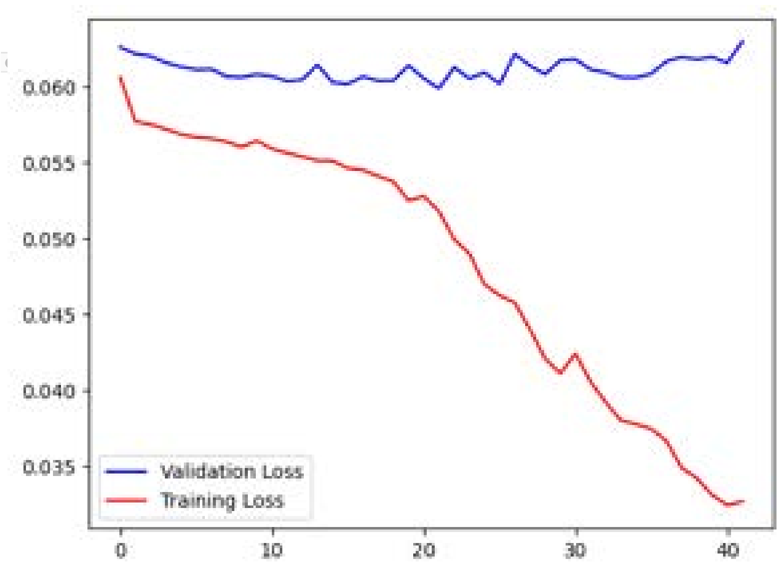
Total params: 10,349,202 (39.48 MB)
Trainable params: 10,349,202 (39.48 MB)
Non-trainable params: 0 (0.00 B)

```
# Defining the path to save the best model
model_checkpoint_path = os.path.join(best_models_path, 'seq_lab_model.keras')
model_checkpoint = ModelCheckpoint(model_checkpoint_path, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

# Fitting the model to the training data
history_seq_lab_model = seq_lab_model.fit(X_lab,Y_lab,batch_size=16, epochs=200, validation_split=0.1, callbacks=[early_sto

best_seq_lab_model = keras.models.load_model(os.path.join(best_models_path, 'seq_lab_model.keras'))
```

```
Epoch 1/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 676ms/step - loss: 0.0637 - mse: 0.0091
Epoch 1: val_loss improved from inf to 0.06261, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/Best
30/30 ━━━━━━━━━━━━━━━━━━━━ 53s 994ms/step - loss: 0.0636 - mse: 0.0091 - val_loss: 0.0626 - val_mse: 0.0095
Epoch 2/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step - loss: 0.0583 - mse: 0.0085
Epoch 2: val_loss improved from 0.06261 to 0.06218, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 40s 180ms/step - loss: 0.0583 - mse: 0.0085 - val_loss: 0.0622 - val_mse: 0.0095
Epoch 3/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step - loss: 0.0580 - mse: 0.0085
```

```
Epoch 3: val_loss improved from 0.06218 to 0.06203, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 10s 174ms/step - loss: 0.0580 - mse: 0.0085 - val_loss: 0.0620 - val_mse: 0.0095
Epoch 4/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 143ms/step - loss: 0.0577 - mse: 0.0086
Epoch 4: val_loss improved from 0.06203 to 0.06159, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 174ms/step - loss: 0.0577 - mse: 0.0086 - val_loss: 0.0616 - val_mse: 0.0093
Epoch 5/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 144ms/step - loss: 0.0584 - mse: 0.0086
Epoch 5: val_loss improved from 0.06159 to 0.06131, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 174ms/step - loss: 0.0584 - mse: 0.0086 - val_loss: 0.0613 - val_mse: 0.0092
Epoch 6/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 144ms/step - loss: 0.0547 - mse: 0.0078
Epoch 6: val_loss improved from 0.06131 to 0.06114, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 10s 176ms/step - loss: 0.0547 - mse: 0.0078 - val_loss: 0.0611 - val_mse: 0.0091
Epoch 7/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 144ms/step - loss: 0.0566 - mse: 0.0083
Epoch 7: val_loss did not improve from 0.06114
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 156ms/step - loss: 0.0566 - mse: 0.0082 - val_loss: 0.0612 - val_mse: 0.0092
Epoch 8/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step - loss: 0.0576 - mse: 0.0085
Epoch 8: val_loss improved from 0.06114 to 0.06068, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 194ms/step - loss: 0.0576 - mse: 0.0085 - val_loss: 0.0607 - val_mse: 0.0090
Epoch 9/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step - loss: 0.0568 - mse: 0.0083
Epoch 9: val_loss improved from 0.06068 to 0.06063, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 181ms/step - loss: 0.0568 - mse: 0.0083 - val_loss: 0.0606 - val_mse: 0.0090
Epoch 10/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step - loss: 0.0542 - mse: 0.0078
Epoch 10: val_loss did not improve from 0.06063
30/30 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - loss: 0.0543 - mse: 0.0078 - val_loss: 0.0608 - val_mse: 0.0091
Epoch 11/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step - loss: 0.0556 - mse: 0.0084
Epoch 11: val_loss did not improve from 0.06063
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 164ms/step - loss: 0.0556 - mse: 0.0083 - val_loss: 0.0607 - val_mse: 0.0091
Epoch 12/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step - loss: 0.0570 - mse: 0.0084
Epoch 12: val_loss improved from 0.06063 to 0.06037, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 181ms/step - loss: 0.0569 - mse: 0.0084 - val_loss: 0.0604 - val_mse: 0.0089
Epoch 13/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step - loss: 0.0545 - mse: 0.0076
Epoch 13: val_loss did not improve from 0.06037
30/30 ━━━━━━━━━━━━━━━━━━━━ 10s 158ms/step - loss: 0.0545 - mse: 0.0076 - val_loss: 0.0605 - val_mse: 0.0088
Epoch 14/200
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step - loss: 0.0563 - mse: 0.0079
Epoch 14: val_loss did not improve from 0.06037
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 164ms/step - loss: 0.0563 - mse: 0.0079 - val_loss: 0.0614 - val_mse: 0.0093
Epoch 15/200
```

```python
# Plotting the training and validation loss
plt.plot(history_seq_lab_model.history['val_loss'],c="b")
plt.plot(history_seq_lab_model.history['loss'],c="r")
plt.legend(['Validation Loss','Training Loss'])
plt.show()
```



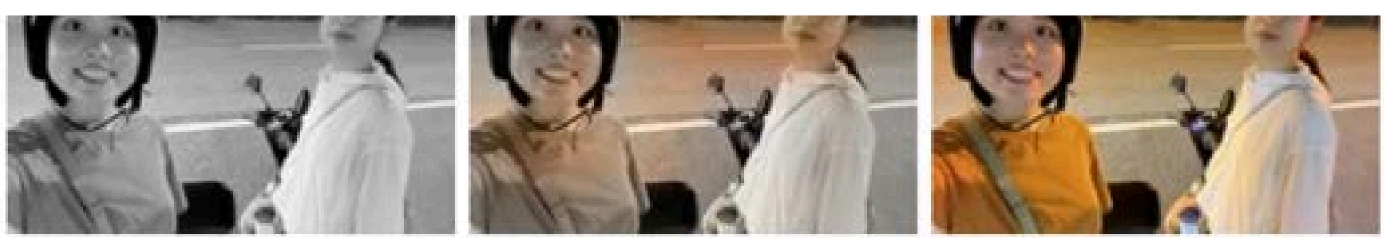
Depiction of Model Performance on images from the Training Set

```
show_predictions_lab(best_seq_lab_model, X_lab, Y_lab, num_samples=5)
```

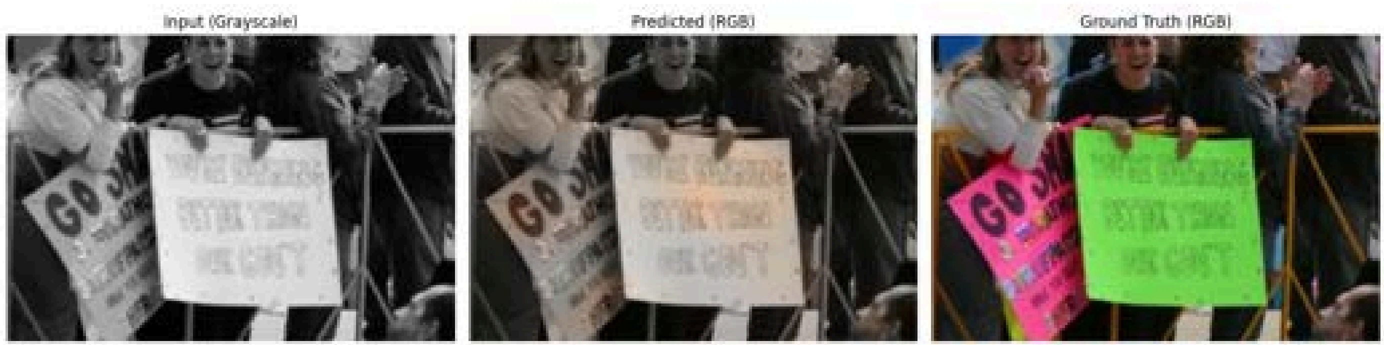
```
WARNING:tensorflow:5 out of the last 9 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_c
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 752ms/step
```

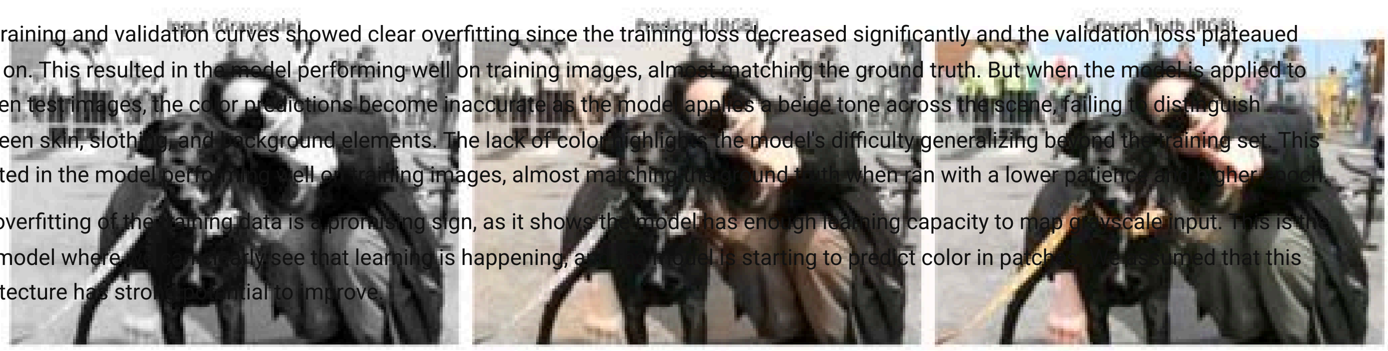
Depiction of Model Performance on images from the Test Set

```
show_predictions_lab(best_seq_lab_model, X_lab_test, Y_lab_test,num_samples=5)
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
```





The training and validation curves showed clear overfitting since the training loss decreased significantly and the validation loss plateaued early on. This resulted in the model performing well on training images, almost matching the ground truth. But when the model is applied to unseen test images, the color predictions become inaccurate as the model applies a beige tone across the scene, failing to distinguish between skin, clothing, and background elements. The lack of color highlights the model's difficulty generalizing beyond the training set. This resulted in the model performing well on training images, almost matching the ground truth when ran with a lower patience and higher epoch.

The overfitting of the training data is a promising sign, as it shows the model has enough learning capacity to map grayscale input. This is the first model where we can clearly see that learning is happening, and the model is starting to predict color in patches. We assumed that this architecture has strong potential to improve.

Model 6: Transformer with VGG

```
# Defining the custom loss functions that we experimented with
def perceptual_loss(y_true, y_pred):
    # Scale from [0, 1] to [0, 255] before using VGG
    y_true_proc = preprocess_input(y_true * 255.0)
    y_pred_proc = preprocess_input(y_pred * 255.0)
```

```
    y_true_features = feature_extractor(y_true_proc)
    y_pred_features = feature_extractor(y_pred_proc)

    return tf.reduce_mean(tf.square(y_true_features - y_pred_features))

def hybrid_loss(y_true, y_pred):
    mae_loss = tf.reduce_mean(tf.abs(y_true - y_pred))
    p_loss = perceptual_loss(y_true, y_pred)
    return 0.3 * p_loss + 0.7 * mae_loss


# Defining the transformer encoder and decoder blocks
def transformer_encoder(inputs, num_heads=4, ff_dim=256):
    # Flatten spatial dims into sequence
    _, H, W, C = inputs.shape
    x = Reshape((H * W, C))(inputs)

    # LayerNorm + MHA
    x_norm = LayerNormalization(epsilon=1e-6)(x)
    attn_output = MultiHeadAttention(num_heads=num_heads, key_dim=C)(x_norm, x_norm)
    x = Add()([x, attn_output])

    # Feed-forward network
    x_norm = LayerNormalization(epsilon=1e-6)(x)
    ff_output = Dense(ff_dim, activation='relu')(x_norm)
    ff_output = Dense(C)(ff_output)
    x = Add()([x, ff_output])

    x = Reshape((H, W, C))(x)
    return x

def build_decoder(vgg_output):
    # 15x22x256 → 30x44x256
    x = UpSampling2D((2, 2))(vgg_output)
    x = Conv2D(256, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    # 30x44x256 → 60x88x128
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(128, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    # 60x88x128 → 120x176x64
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(64, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(32, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    output_rgb = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)
    return output_rgb


# Defining the transformer + vgg model architecture
# Input grayscale image
input_l = Input(shape=(120, 176, 1), name='grayscale_input')
x_rgb = Lambda(lambda x: tf.image.grayscale_to_rgb(x))(input_l)

# VGG encoder
vgg_base = VGG16(include_top=False, weights='imagenet', input_tensor=x_rgb)
for layer in vgg_base.layers:
    layer.trainable = False

vgg_features = vgg_base.get_layer('block3_conv3').output  # (15, 22, 256)

# Transformer block
x = transformer_encoder(vgg_features, num_heads=4, ff_dim=512)

# Decoder block
```

```
x = build_decoder(x)

output_rgb = Lambda(lambda x: tf.image.resize_with_crop_or_pad(x, 120, 176))(x)

trans_vgg_model = Model(inputs=input_l, outputs=output_rgb)

# Feature extractor for perceptual loss
vgg_feat_model = VGG16(include_top=False, weights='imagenet', input_shape=(120, 176, 3))
vgg_feat_model.trainable = False
feature_extractor = Model(inputs=vgg_feat_model.input, outputs=vgg_feat_model.get_layer('block3_conv3').output)

trans_vgg_model.compile(optimizer='adam', loss='mse', metrics=['mae'])
trans_vgg_model.summary()
```

Model: "functional_82"

Layer (type)	Output Shape	Param #	Connected to
grayscale_input (InputLayer)	(None, 120, 176, 1)	0	-
lambda_6 (Lambda)	(None, 120, 176, 3)	0	grayscale_input[…
block1_conv1 (Conv2D)	(None, 120, 176, 64)	1,792	lambda_6[0][0]
block1_conv2 (Conv2D)	(None, 120, 176, 64)	36,928	block1_conv1[0][…
block1_pool (MaxPooling2D)	(None, 60, 88, 64)	0	block1_conv2[0][…
block2_conv1 (Conv2D)	(None, 60, 88, 128)	73,856	block1_pool[0][0]
block2_conv2 (Conv2D)	(None, 60, 88, 128)	147,584	block2_conv1[0][…
block2_pool (MaxPooling2D)	(None, 30, 44, 128)	0	block2_conv2[0][…
block3_conv1 (Conv2D)	(None, 30, 44, 256)	295,168	block2_pool[0][0]
block3_conv2 (Conv2D)	(None, 30, 44, 256)	590,080	block3_conv1[0][…
block3_conv3 (Conv2D)	(None, 30, 44, 256)	590,080	block3_conv2[0][…
reshape_4 (Reshape)	(None, 1320, 256)	0	block3_conv3[0][…
layer_normalizatio… (LayerNormalizatio…	(None, 1320, 256)	512	reshape_4[0][0]
multi_head_attenti… (MultiHeadAttentio…	(None, 1320, 256)	1,051,904	layer_normalizat… layer_normalizat…
add_4 (Add)	(None, 1320, 256)	0	reshape_4[0][0], multi_head_atten…
layer_normalizatio… (LayerNormalizatio…	(None, 1320, 256)	512	add_4[0][0]
dense_4 (Dense)	(None, 1320, 512)	131,584	layer_normalizat…
dense_5 (Dense)	(None, 1320, 256)	131,328	dense_4[0][0]
add_5 (Add)	(None, 1320, 256)	0	add_4[0][0], dense_5[0][0]
reshape_5 (Reshape)	(None, 30, 44, 256)	0	add_5[0][0]
up_sampling2d_20	(None, 60, 88,	0	reshape_5[0][0]

```
# Fitting the model to the training data
history_trans_vgg = trans_vgg_model.fit(X, Y, batch_size=16, epochs=200, validation_split=0.20, callbacks=[early_stop])
```

```
Epoch 1/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 39s 1s/step - loss: 0.0810 - mae: 0.2375 - val_loss: 0.2669 - val_mae: 0.4396
Epoch 2/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 24s 567ms/step - loss: 0.0639 - mae: 0.2121 - val_loss: 0.1952 - val_mae: 0.3660
Epoch 3/200
```

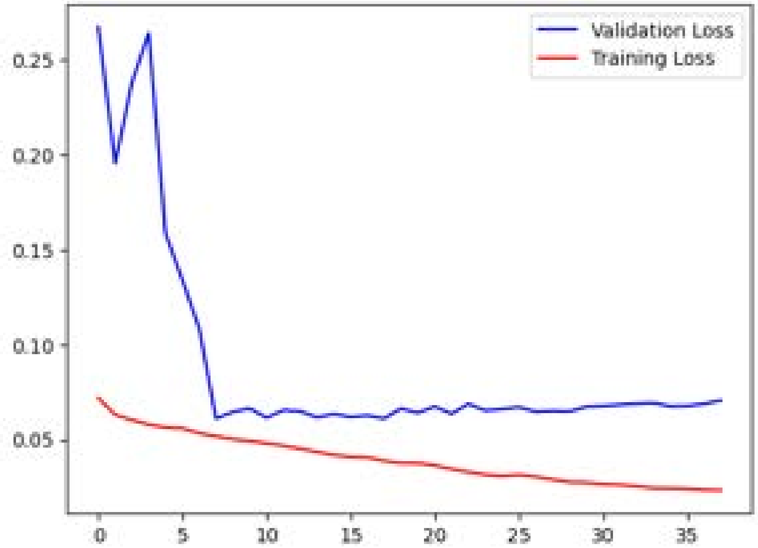
batch_normalizatio… (BatchNormalizatio…	(None, 60, 88, 256)	1,024	conv2d_88[0][0]
activation_8 (Activation)	(None, 60, 88, 256)	0	batch_normalizat…

up_sampling2d_21 (UpSampling2D)	(None, 120, 176, 256)	0	activation_8[0][…
conv2d_89 (Conv2D)	(None, 120, 176, 128)	295,040	up_sampling2d_21…
batch_normalizatio… (BatchNormalizatio…	(None, 120, 176, 128)	512	conv2d_89[0][0]
activation_9 (Activation)	(None, 120, 176, 128)	0	batch_normalizat…
up_sampling2d_22 (UpSampling2D)	(None, 240, 352, 128)	0	activation_9[0][…
conv2d_90 (Conv2D)	(None, 240, 352, 64)	73,792	up_sampling2d_22…
batch_normalizatio… (BatchNormalizatio…	(None, 240, 352, 64)	256	conv2d_90[0][0]
activation_10 (Activation)	(None, 240, 352, 64)	0	batch_normalizat…
conv2d_91 (Conv2D)	(None, 240, 352, 32)	18,464	activation_10[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 240, 352, 32)	128	conv2d_91[0][0]
activation_11 (Activation)	(None, 240, 352, 32)	0	batch_normalizat…
conv2d_92 (Conv2D)	(None, 240, 352, 3)	867	activation_11[0]…
lambda_7 (Lambda)	(None, 120, 176, 3)	0	conv2d_92[0][0]

Total params: 4,031,491 (15.38 MB)
Trainable params: 2,295,043 (8.75 MB)
Non-trainable params: 1,736,448 (6.62 MB)

```
27/27 ━━━━━━━━━━━━━━━━━━━━ 22s 619ms/step - loss: 0.0616 - mae: 0.2074 - val_loss: 0.2376 - val_mae: 0.4241
Epoch 4/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 21s 631ms/step - loss: 0.0567 - mae: 0.1960 - val_loss: 0.2637 - val_mae: 0.4497
Epoch 5/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 20s 594ms/step - loss: 0.0563 - mae: 0.1953 - val_loss: 0.1584 - val_mae: 0.3338
Epoch 6/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 16s 607ms/step - loss: 0.0553 - mae: 0.1941 - val_loss: 0.1340 - val_mae: 0.3006
Epoch 7/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 18s 661ms/step - loss: 0.0528 - mae: 0.1885 - val_loss: 0.1085 - val_mae: 0.2691
Epoch 8/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 21s 670ms/step - loss: 0.0521 - mae: 0.1865 - val_loss: 0.0611 - val_mae: 0.1964
Epoch 9/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 21s 675ms/step - loss: 0.0499 - mae: 0.1817 - val_loss: 0.0645 - val_mae: 0.2026
Epoch 10/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 21s 681ms/step - loss: 0.0496 - mae: 0.1806 - val_loss: 0.0665 - val_mae: 0.2097
Epoch 11/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 20s 682ms/step - loss: 0.0482 - mae: 0.1785 - val_loss: 0.0614 - val_mae: 0.2018
Epoch 12/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 17s 636ms/step - loss: 0.0462 - mae: 0.1747 - val_loss: 0.0655 - val_mae: 0.1989
Epoch 13/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 21s 674ms/step - loss: 0.0450 - mae: 0.1703 - val_loss: 0.0649 - val_mae: 0.1996
Epoch 14/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 21s 681ms/step - loss: 0.0439 - mae: 0.1674 - val_loss: 0.0616 - val_mae: 0.2000
Epoch 15/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 20s 645ms/step - loss: 0.0414 - mae: 0.1625 - val_loss: 0.0634 - val_mae: 0.1966
Epoch 16/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 17s 643ms/step - loss: 0.0411 - mae: 0.1618 - val_loss: 0.0619 - val_mae: 0.2010
Epoch 17/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 18s 677ms/step - loss: 0.0405 - mae: 0.1608 - val_loss: 0.0626 - val_mae: 0.1977
Epoch 18/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 20s 675ms/step - loss: 0.0391 - mae: 0.1573 - val_loss: 0.0611 - val_mae: 0.1965
Epoch 19/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 19s 637ms/step - loss: 0.0375 - mae: 0.1538 - val_loss: 0.0663 - val_mae: 0.2027
Epoch 20/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 21s 649ms/step - loss: 0.0373 - mae: 0.1529 - val_loss: 0.0640 - val_mae: 0.1994
Epoch 21/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 21s 683ms/step - loss: 0.0361 - mae: 0.1498 - val_loss: 0.0674 - val_mae: 0.2067
Epoch 22/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 20s 676ms/step - loss: 0.0351 - mae: 0.1470 - val_loss: 0.0635 - val_mae: 0.1995
Epoch 23/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 19s 639ms/step - loss: 0.0329 - mae: 0.1412 - val_loss: 0.0688 - val_mae: 0.2056
Epoch 24/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 17s 649ms/step - loss: 0.0313 - mae: 0.1381 - val_loss: 0.0653 - val_mae: 0.2010
Epoch 25/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 21s 681ms/step - loss: 0.0296 - mae: 0.1329 - val_loss: 0.0660 - val_mae: 0.2020
Epoch 26/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 20s 677ms/step - loss: 0.0307 - mae: 0.1359 - val_loss: 0.0671 - val_mae: 0.2028
Epoch 27/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 19s 639ms/step - loss: 0.0306 - mae: 0.1358 - val_loss: 0.0647 - val_mae: 0.2017
Epoch 28/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 17s 646ms/step - loss: 0.0293 - mae: 0.1323 - val_loss: 0.0651 - val_mae: 0.2010
Epoch 29/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 18s 678ms/step - loss: 0.0276 - mae: 0.1281 - val_loss: 0.0648 - val_mae: 0.2024
```

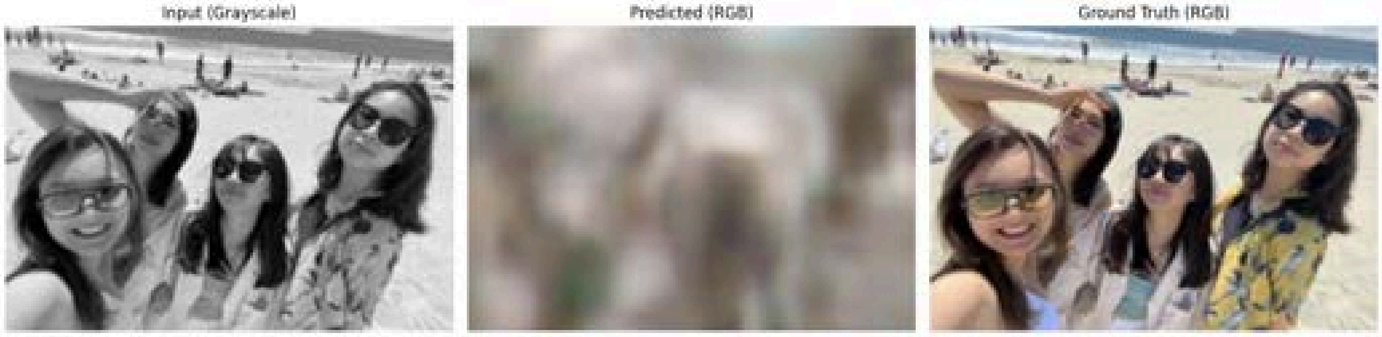
```
# Plotting the training and validation loss
plt.plot(history_trans_vgg.history['val_loss'],c="b")
plt.plot(history_trans_vgg.history['loss'],c="r")
plt.legend(['Validation Loss','Training Loss'])
plt.show()
```

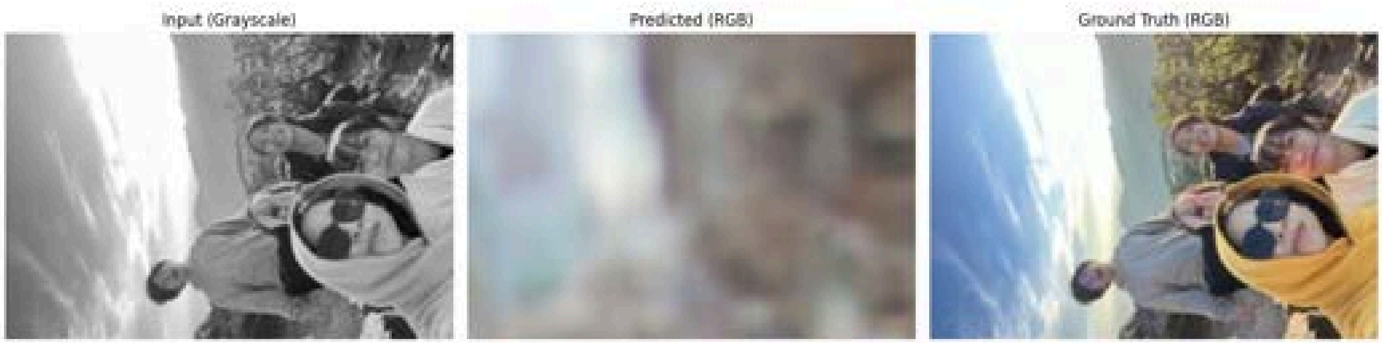


Depiction of Model Performance on images from the Training Set

```
show_predictions(trans_vgg_model, X, Y, num_samples=5)
```

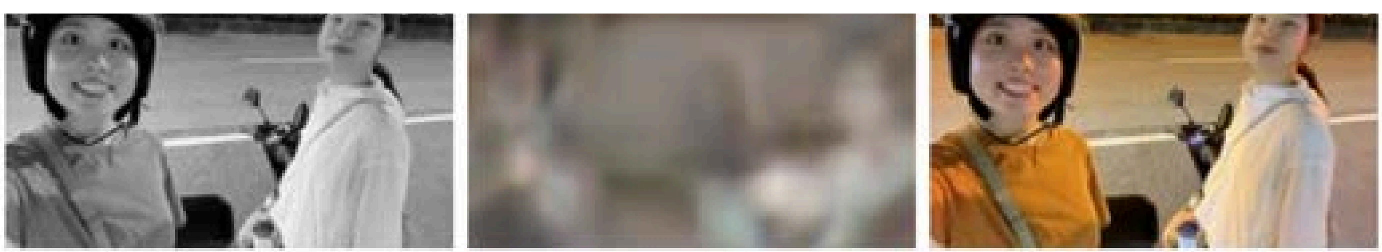
```
WARNING:tensorflow:6 out of the last 11 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_
1/1 ━━━━━━━━━━━━━━━━━━━━ 9s 9s/step
```

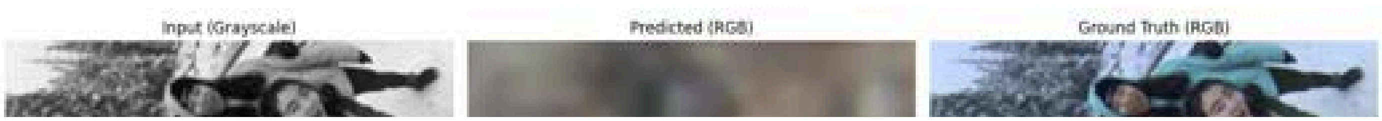




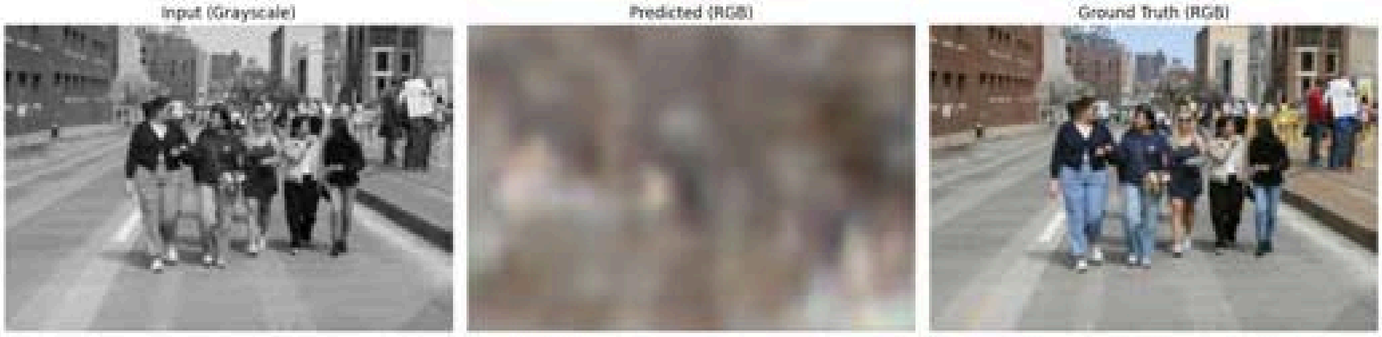
Depiction of Model Performance on images from the Test Set

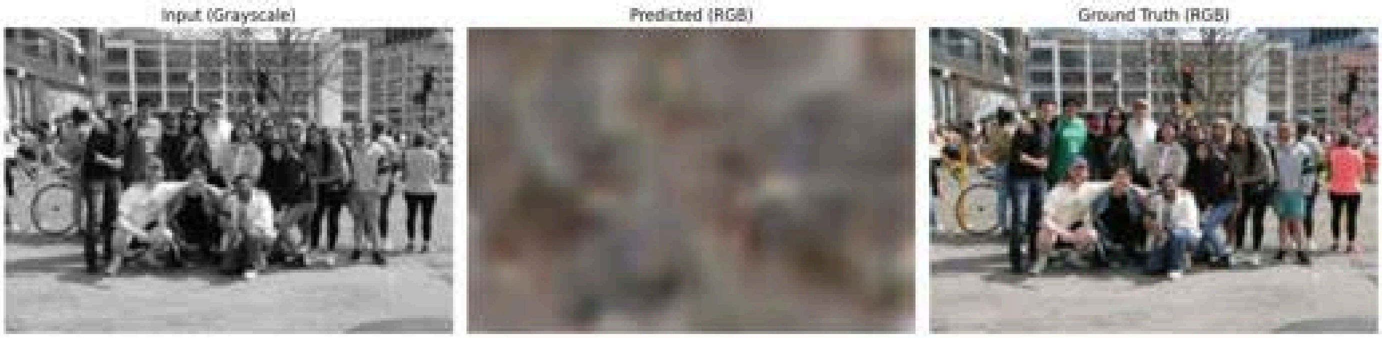

```
show_predictions(trans_vgg_model, X_test, Y_test, num_samples=6)
```

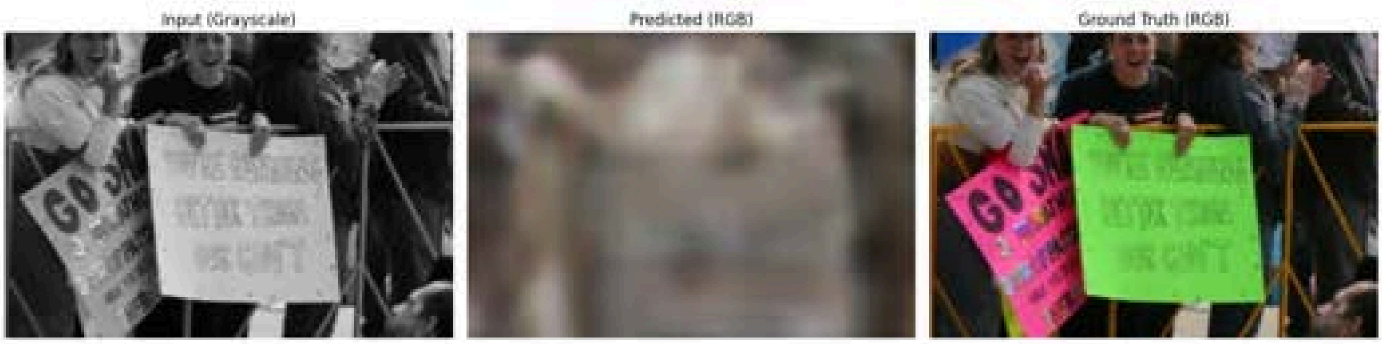


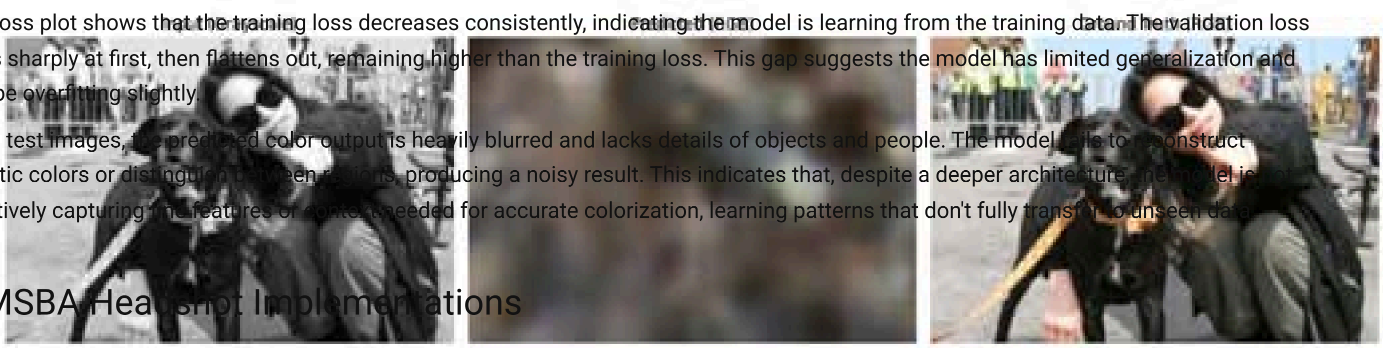


```
1/1 ━━━━━━━━━━━━━━━━━━━━ 10s 10s/step
```







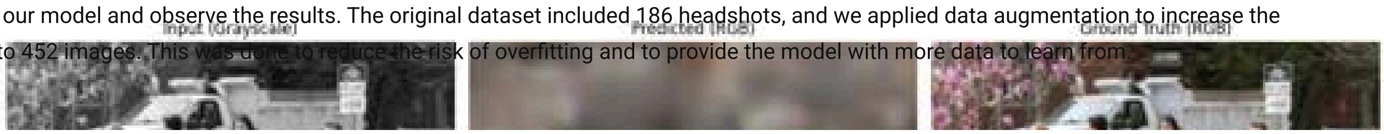


This loss plot shows that the training loss decreases consistently, indicating the model is learning from the training data. The validation loss drops sharply at first, then flattens out, remaining higher than the training loss. This gap suggests the model has limited generalization and may be overfitting slightly.

In the test images, the predicted color output is heavily blurred and lacks details of objects and people. The model fails to reconstruct realistic colors or distinguish between regions, producing a noisy result. This indicates that, despite a deeper architecture, the model is not effectively capturing the features or context needed for accurate colorization, learning patterns that don't fully transfer to unseen data.

MSBA Headshot Implementations

Having explored colorization using full scene portraits, as shown above, we decided to pivot to a new dataset with a consistent background and a more standardized color range. We used headshots taken during the MSBA (Master of Science in Business Analytics) orientation to rerun our model and observe the results. The original dataset included 186 headshots, and we applied data augmentation to increase the total to 452 images. This was done to reduce the risk of overfitting and to provide the model with more data to learn from.



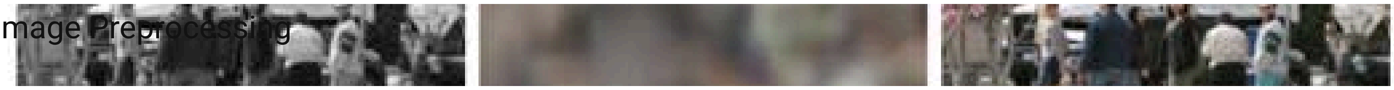


Image Preprocessing

```
# Defining the required Google Drive paths for the MSBA Headshot Implementation
color_image_path = '/content/drive/MyDrive/BA865/BA865 Group Project/Dataset/MSBA/Color'
color_resized_image_path = '/content/drive/MyDrive/BA865/BA865 Group Project/Dataset/MSBA/Color Resized'
test_color_image_path = '/content/drive/MyDrive/BA865/BA865 Group Project/Dataset/MSBA/Test'
test_color_resized_image_path = '/content/drive/MyDrive/BA865/BA865 Group Project/Dataset/MSBA/Test Resized'
best_models_path = '/content/drive/MyDrive/BA865/BA865 Group Project/Best Models MSBA'
```

Train Color Images Preprocessing

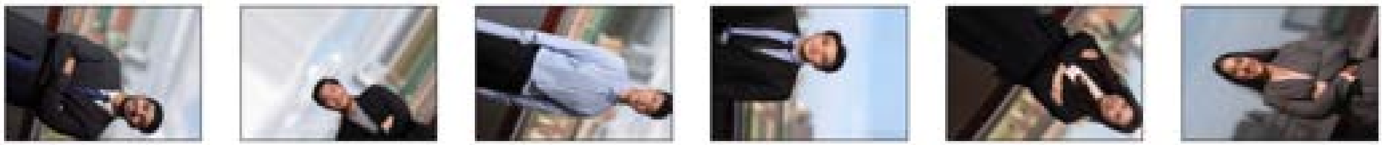

```
# Creating color resized image folder if it doesn't exist
os.makedirs(color_resized_image_path, exist_ok=True)

# Ensuring the color resized image folder is empty
for file_name in os.listdir(color_resized_image_path):
    file_path = os.path.join(color_resized_image_path, file_name)
    if os.path.isfile(file_path):
        os.remove(file_path)

resize_success_count = 0
resize_fail_count = 0

# Looping through all color images and applying the resizing, cropping, rotation, and compression
for image in os.listdir(color_image_path):
    if image.lower().endswith(image_extensions):
        input_img_path = os.path.join(color_image_path, image)
        output_img_path = os.path.join(color_resized_image_path, image)

        # The below if statement resizes the image while also maintaining a count of successful and unsuccessful conversions
        if image_resizer(input_img_path, output_img_path, (176, 120)):
            resize_success_count += 1
        else:
            resize_fail_count += 1

print(f"Successfully resized {resize_success_count} images.")
print(f"Failed to resize {resize_fail_count} images.")
```

```
Successfully resized 186 images.
Failed to resize 0 images.
```

```
color_images = []

for image in os.listdir(color_resized_image_path):
    color_image_path = os.path.join(color_resized_image_path, image)

    # Converting the color image to numpy array
    if os.path.isfile(color_image_path):
        col_img = Image.open(color_image_path).convert('RGB')
        color_image_array = np.array(col_img).astype('float32') / 255.0
        color_images.append(color_image_array)
```

```
# This code will use ImageDataGenerated to create additional images as a form of data augmentation, helping against overfitt
datagen = ImageDataGenerator(rotation_range=40, width_shift_range=0.2, height_shift_range=0.2, rescale=1./255,
                             shear_range=0.2, zoom_range=0.2, horizontal_flip=True, fill_mode='nearest')

datagen.fit(color_images)

image_batches = datagen.flow(np.stack(color_images, axis=0), batch_size=40)

n = 10
plt.figure(figsize= (20,10))

q = 0
for i,batch in enumerate(image_batches):
    for j, image in enumerate(batch):
        if j == 1:
            if i <= 5:
                ax = plt.subplot(1, 6, i+1)
                plt.imshow(tf.keras.utils.array_to_img(image))
```

```
                ax.get_xaxis().set_visible(False)
                ax.get_yaxis().set_visible(False)

        # The iterator outputs image data with pixel values between 0-1; we are using values between 0-255 elsewhere. We nee
        color_images.append(np.multiply(image,255))

    if q > 5:
      break

    q += 1

plt.show()

print(f'We now have a total of {len(color_images)} images.')
```



```
We now have a total of 452 images.
```

```
# Splitting the predictor and labels in the RGB color space, for the training set
X = []
Y = []

for img in color_images:
    try:
        gray = tf.image.rgb_to_grayscale(img)
        X.append(gray.numpy())
        Y.append(img)
    except Exception as e:
        print("error:", e)

X = np.array(X)
Y = np.array(Y)

print("X_rgb shape:", X.shape)
print("Y_rgb shape:", Y.shape)
```

```
X_rgb shape: (452, 120, 176, 1)
Y_rgb shape: (452, 120, 176, 3)
```

```
# Splitting the predictor and labels in the LAB color space, for the training set
X_lab = []
Y_lab = []

for img in color_images:
  try:
      lab = rgb2lab(img)
      X_lab.append(lab[:,:,0])
      #restrict values to be between -1 and 1
      Y_lab.append(lab[:,:,1:] / 128)
  except Exception as e:
     print("error:", e)

X_lab = np.array(X_lab)
Y_lab = np.array(Y_lab)

#dimensions to be the same for X and Y
X_lab = X_lab.reshape(X_lab.shape+(1,))

print("X_lab shape:", X_lab.shape)
print("Y_lab shape:", Y_lab.shape)
```

```
X_lab shape: (452, 120, 176, 1)
Y_lab shape: (452, 120, 176, 2)
```

Test Color Images Preprocessing

```
# Creating color resized image folder for the test set if it doesn't exist
os.makedirs(test_color_resized_image_path, exist_ok=True)

# Ensuring the color resized image folder is empty
for file_name in os.listdir(test_color_resized_image_path):
    file_path = os.path.join(test_color_resized_image_path, file_name)
    if os.path.isfile(file_path):
        os.remove(file_path)

test_resize_success_count = 0
test_resize_fail_count = 0

# Looping through all color images and applying the resizing, cropping, rotation, and compression
for test_image in os.listdir(test_color_image_path):
    if test_image.lower().endswith(image_extensions):
        input_img_path = os.path.join(test_color_image_path, test_image)
        output_img_path = os.path.join(test_color_resized_image_path, test_image)

        # The below if statement resizes the image while also maintaining a count of successful and unsuccessful conversions
        if image_resizer(input_img_path, output_img_path, (176, 120)):
            test_resize_success_count += 1
        else:
            test_resize_fail_count += 1

print(f"Successfully resized {test_resize_success_count} images.")
print(f"Failed to resize {test_resize_fail_count} images.")
```

```
Successfully resized 8 images.
Failed to resize 0 images.
```

```
test_color_images = []

for image in os.listdir(test_color_resized_image_path):
    test_color_image_path = os.path.join(test_color_resized_image_path, image)

    # Converting the color image to numpy array
    if os.path.isfile(test_color_image_path):
        col_img = Image.open(test_color_image_path).convert('RGB')
        test_color_image_array = np.array(col_img).astype('float32') / 255.0
        test_color_images.append(test_color_image_array)
```

```
# Splitting the predictor and labels in the LAB color space, for the test set
X_lab_test = []
Y_lab_test = []

for img in test_color_images:
  try:
      lab_test = rgb2lab(img)
      X_lab_test.append(lab_test[:,:,0])
      #restrict values to be between -1 and 1
      Y_lab_test.append(lab_test[:,:,1:] / 128)
  except Exception as e:
     print("error:", e)

X_lab_test = np.array(X_lab_test)
Y_lab_test = np.array(Y_lab_test)

#dimensions to be the same for X and Y
X_lab_test = X_lab_test.reshape(X_lab_test.shape+(1,))

print("X_lab_test shape:", X_lab_test.shape)
print("Y_lab_test shape:", Y_lab_test.shape)
```

```
X_lab_test shape: (8, 120, 176, 1)
Y_lab_test shape: (8, 120, 176, 2)
```

```
# Splitting the predictor and labels in the LAB color space, for the test set
X_test = []
Y_test = []

for img in test_color_images:
    try:
```

```
        gray = tf.image.rgb_to_grayscale(img)
        X_test.append(gray.numpy())
        Y_test.append(img)
    except Exception as e:
        print("error:", e)

X_test = np.array(X_test)
Y_test = np.array(Y_test)

print("X_rgb_test shape:", X_test.shape)
print("Y_rgb_test shape:", Y_test.shape)
```

```
X_rgb_test shape: (8, 120, 176, 1)
Y_rgb_test shape: (8, 120, 176, 3)
```

Model 1: Sequential RGB Model

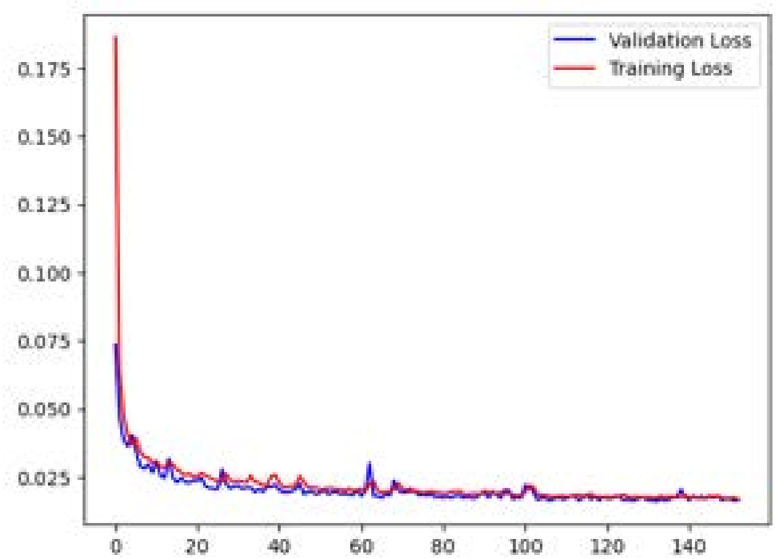
```
# Defining the architecture for the sequential RGB model
input_l = Input(shape=(120, 176, 1))

x = Conv2D(64, (3, 3), activation='relu', padding='same')(input_l)
x = Conv2D(64, (3, 3), activation='relu', strides=2, padding='same')(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = Conv2D(128, (3, 3), activation='relu', strides=2, padding='same')(x)

x = UpSampling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)

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

output_rgb = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)
output_rgb = Lambda(lambda x: tf.image.resize_with_crop_or_pad(x, 120, 176))(output_rgb)

seq_rgb_model = Model(inputs=input_l, outputs=output_rgb)
seq_rgb_model.compile(optimizer='adam', loss='mae', metrics=['mse'])
```

```
# Fitting the model to the training data
history_seq_rgb = seq_rgb_model.fit(X, Y, batch_size=16, epochs=200, validation_split=0.1, callbacks=[early_stop])
```

```
Epoch 1/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 15s 475ms/step - loss: 0.2283 - mse: 0.0750 - val_loss: 0.0735 - val_mse: 0.0106
Epoch 2/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 9s 64ms/step - loss: 0.0718 - mse: 0.0098 - val_loss: 0.0466 - val_mse: 0.0048
Epoch 3/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 80ms/step - loss: 0.0506 - mse: 0.0054 - val_loss: 0.0388 - val_mse: 0.0035
Epoch 4/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - loss: 0.0408 - mse: 0.0038 - val_loss: 0.0361 - val_mse: 0.0029
Epoch 5/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 63ms/step - loss: 0.0372 - mse: 0.0031 - val_loss: 0.0404 - val_mse: 0.0032
Epoch 6/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - loss: 0.0407 - mse: 0.0034 - val_loss: 0.0346 - val_mse: 0.0025
Epoch 7/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - loss: 0.0353 - mse: 0.0027 - val_loss: 0.0290 - val_mse: 0.0019
Epoch 8/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - loss: 0.0327 - mse: 0.0024 - val_loss: 0.0279 - val_mse: 0.0018
Epoch 9/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 63ms/step - loss: 0.0327 - mse: 0.0023 - val_loss: 0.0296 - val_mse: 0.0019
Epoch 10/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - loss: 0.0309 - mse: 0.0021 - val_loss: 0.0266 - val_mse: 0.0016
Epoch 11/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - loss: 0.0306 - mse: 0.0020 - val_loss: 0.0306 - val_mse: 0.0018
Epoch 12/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - loss: 0.0294 - mse: 0.0019 - val_loss: 0.0256 - val_mse: 0.0015
Epoch 13/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 59ms/step - loss: 0.0283 - mse: 0.0018 - val_loss: 0.0247 - val_mse: 0.0013
Epoch 14/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 59ms/step - loss: 0.0301 - mse: 0.0020 - val_loss: 0.0317 - val_mse: 0.0020
Epoch 15/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - loss: 0.0312 - mse: 0.0020 - val_loss: 0.0247 - val_mse: 0.0013
Epoch 16/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - loss: 0.0279 - mse: 0.0017 - val_loss: 0.0230 - val_mse: 0.0012
Epoch 17/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - loss: 0.0273 - mse: 0.0017 - val_loss: 0.0247 - val_mse: 0.0014
```

```
Epoch 18/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 59ms/step - loss: 0.0260 - mse: 0.0016 - val_loss: 0.0231 - val_mse: 0.0012
Epoch 19/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - loss: 0.0255 - mse: 0.0015 - val_loss: 0.0229 - val_mse: 0.0012
Epoch 20/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 59ms/step - loss: 0.0250 - mse: 0.0015 - val_loss: 0.0237 - val_mse: 0.0013
Epoch 21/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 63ms/step - loss: 0.0254 - mse: 0.0015 - val_loss: 0.0235 - val_mse: 0.0012
Epoch 22/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 64ms/step - loss: 0.0271 - mse: 0.0016 - val_loss: 0.0246 - val_mse: 0.0014
Epoch 23/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - loss: 0.0251 - mse: 0.0015 - val_loss: 0.0214 - val_mse: 0.0011
Epoch 24/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 60ms/step - loss: 0.0245 - mse: 0.0014 - val_loss: 0.0210 - val_mse: 0.0011
Epoch 25/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - loss: 0.0243 - mse: 0.0014 - val_loss: 0.0206 - val_mse: 0.0010
Epoch 26/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - loss: 0.0232 - mse: 0.0013 - val_loss: 0.0206 - val_mse: 0.0010
Epoch 27/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 68ms/step - loss: 0.0226 - mse: 0.0012 - val_loss: 0.0277 - val_mse: 0.0015
Epoch 28/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 67ms/step - loss: 0.0272 - mse: 0.0016 - val_loss: 0.0227 - val_mse: 0.0012
Epoch 29/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - loss: 0.0241 - mse: 0.0013 - val_loss: 0.0207 - val_mse: 0.0010
```

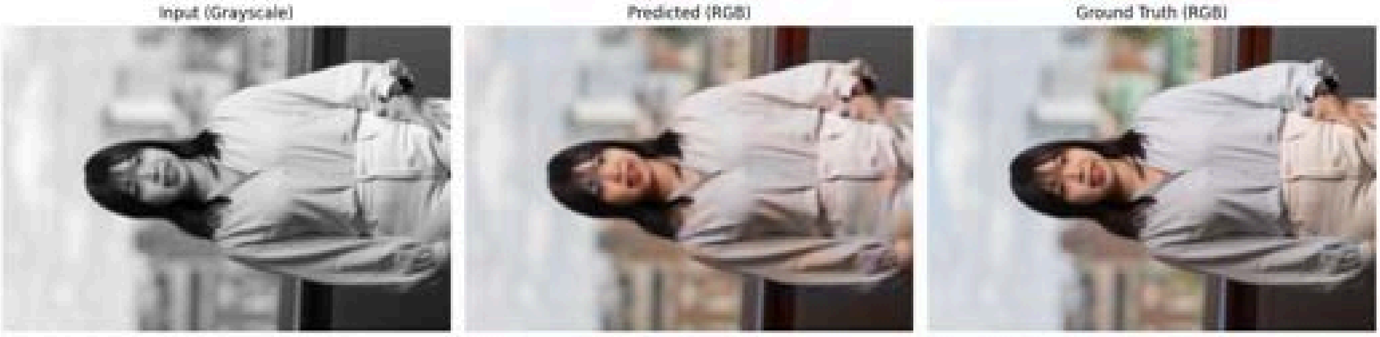
```
# Plotting the training and validation loss
plt.plot(history_seq_rgb.history['val_loss'],c="b")
plt.plot(history_seq_rgb.history['loss'],c="r")
plt.legend(['Validation Loss','Training Loss'])
plt.show()
```

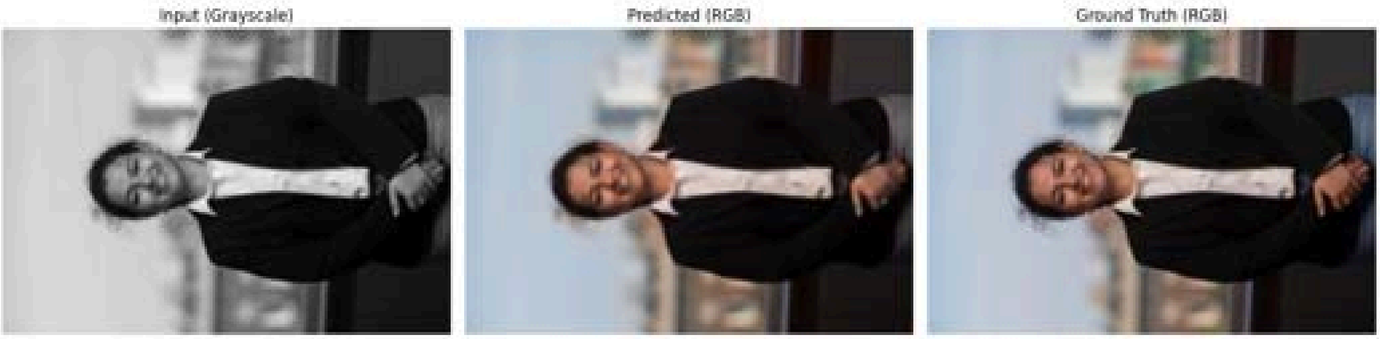


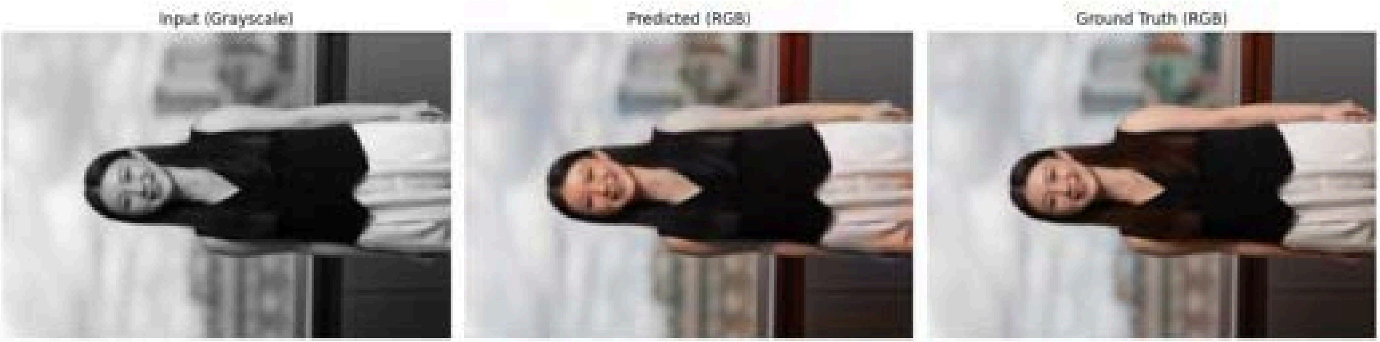
Depiction of Model Performance on images from the Training Set

```
show_predictions(seq_rgb_model, X, Y, num_samples=5)
```

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 716ms/step

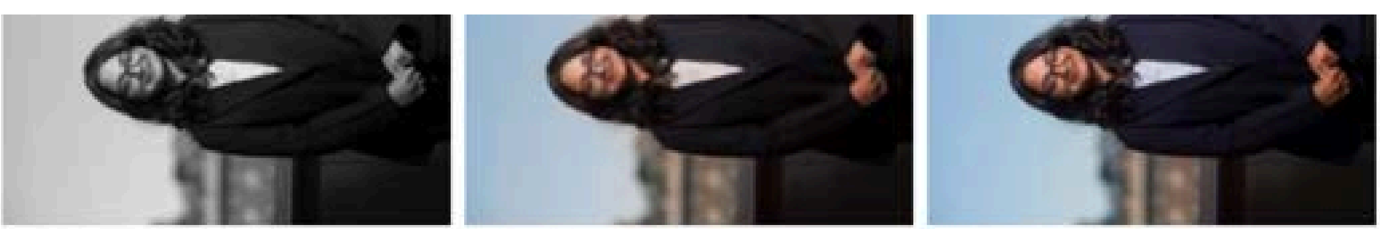


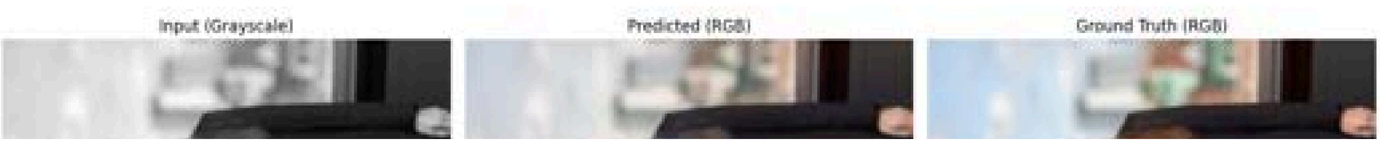




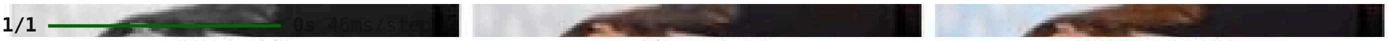
Depiction of Model Performance on images from the Test Set

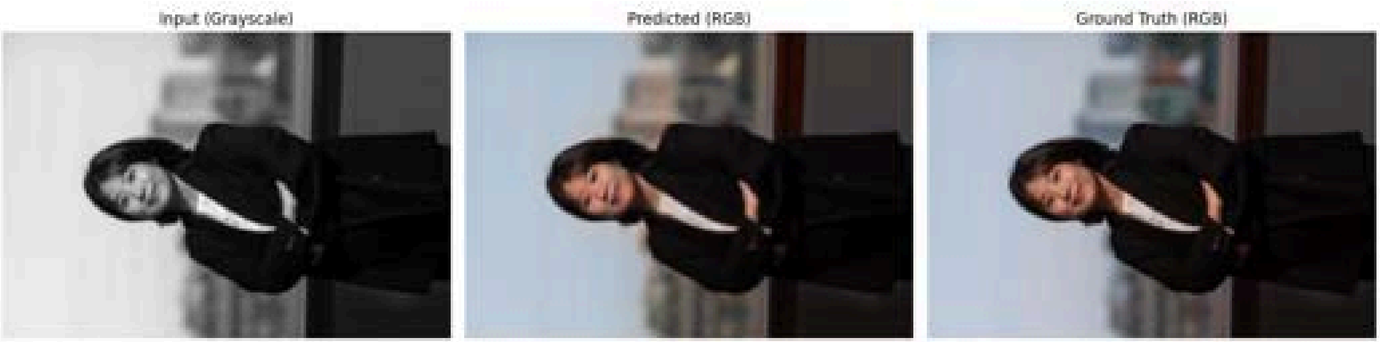

```
show_predictions(seq_rgb_model, X_test, Y_test, num_samples=5)
```

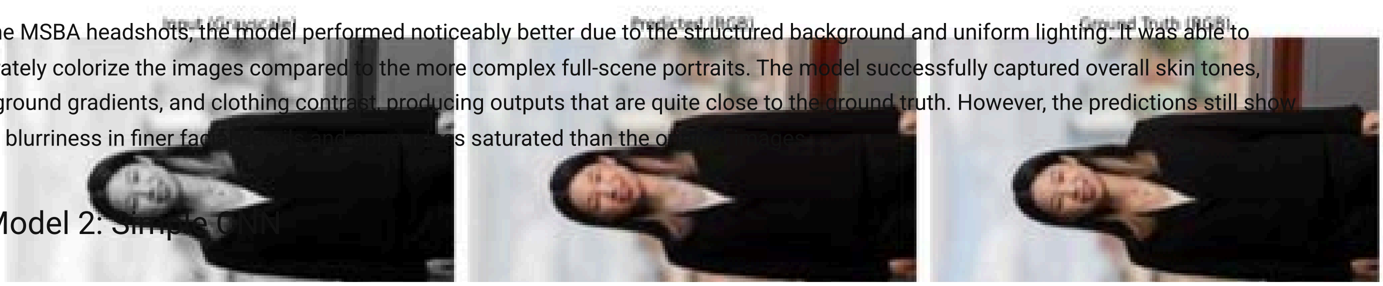




```
1/1
```







For the MSBA headshots, the model performed noticeably better due to the structured background and uniform lighting. It was able to accurately colorize the images compared to the more complex full-scene portraits. The model successfully captured overall skin tones, background gradients, and clothing contrast, producing outputs that are quite close to the ground truth. However, the predictions still show slight blurriness in finer facial details and are less saturated than the original images.

Model 2: Simple CNN

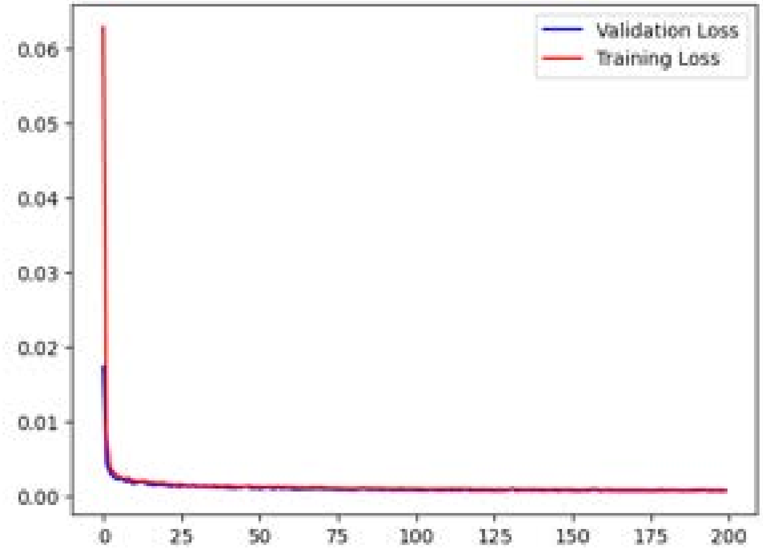
```
# Defining the architecture for the simple CNN model
def build_simple_cnn(input_shape=(120, 176, 1)):
    inputs = layers.Input(shape=input_shape)

    x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = layers.MaxPooling2D((2, 2))(x)

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

    x = layers.Conv2DTranspose(64, (2, 2), strides=2, padding='same')(x)
```

```
    x = layers.ReLU()(x)
    outputs = layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)

    model = models.Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model

model_simple_cnn = build_simple_cnn()


# Defining the path to save the best model
model_checkpoint_path = os.path.join(best_models_path, 'simple_cnn.keras')
model_checkpoint = ModelCheckpoint(model_checkpoint_path, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

# Fitting the model to the training data
history_simple_cnn = model_simple_cnn.fit(X, Y, batch_size=16, epochs=200, validation_split=0.1, callbacks=[early_stop, mode

best_simple_cnn = keras.models.load_model(os.path.join(best_models_path, 'simple_cnn.keras'))
```

```
Epoch 1/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 253ms/step - loss: 0.0780 - mae: 0.2413
Epoch 1: val_loss improved from inf to 0.01737, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/Best
26/26 ━━━━━━━━━━━━━━━━━━━━ 15s 475ms/step - loss: 0.0774 - mae: 0.2401 - val_loss: 0.0174 - val_mae: 0.1139
Epoch 2/200
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 0.0119 - mae: 0.0842
Epoch 2: val_loss improved from 0.01737 to 0.00441, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 9s 78ms/step - loss: 0.0116 - mae: 0.0830 - val_loss: 0.0044 - val_mae: 0.0490
Epoch 3/200
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step - loss: 0.0045 - mae: 0.0473
Epoch 3: val_loss improved from 0.00441 to 0.00328, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 75ms/step - loss: 0.0044 - mae: 0.0471 - val_loss: 0.0033 - val_mae: 0.0395
Epoch 4/200
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step - loss: 0.0034 - mae: 0.0402
Epoch 4: val_loss improved from 0.00328 to 0.00270, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 75ms/step - loss: 0.0034 - mae: 0.0402 - val_loss: 0.0027 - val_mae: 0.0358
Epoch 5/200
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step - loss: 0.0030 - mae: 0.0379
Epoch 5: val_loss improved from 0.00270 to 0.00235, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 86ms/step - loss: 0.0030 - mae: 0.0379 - val_loss: 0.0023 - val_mae: 0.0331
Epoch 6/200
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - loss: 0.0028 - mae: 0.0361
Epoch 6: val_loss improved from 0.00235 to 0.00226, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 86ms/step - loss: 0.0028 - mae: 0.0360 - val_loss: 0.0023 - val_mae: 0.0328
Epoch 7/200
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - loss: 0.0024 - mae: 0.0339
Epoch 7: val_loss did not improve from 0.00226
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 77ms/step - loss: 0.0024 - mae: 0.0340 - val_loss: 0.0024 - val_mae: 0.0342
Epoch 8/200
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - loss: 0.0024 - mae: 0.0337
Epoch 8: val_loss improved from 0.00226 to 0.00197, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 86ms/step - loss: 0.0024 - mae: 0.0337 - val_loss: 0.0020 - val_mae: 0.0303
Epoch 9/200
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 0.0024 - mae: 0.0338
Epoch 9: val_loss did not improve from 0.00197
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 79ms/step - loss: 0.0024 - mae: 0.0339 - val_loss: 0.0022 - val_mae: 0.0331
Epoch 10/200
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 0.0022 - mae: 0.0321
Epoch 10: val_loss improved from 0.00197 to 0.00180, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 75ms/step - loss: 0.0022 - mae: 0.0321 - val_loss: 0.0018 - val_mae: 0.0290
Epoch 11/200
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - loss: 0.0021 - mae: 0.0311
Epoch 11: val_loss improved from 0.00180 to 0.00173, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 80ms/step - loss: 0.0021 - mae: 0.0311 - val_loss: 0.0017 - val_mae: 0.0283
Epoch 12/200
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 0.0019 - mae: 0.0297
Epoch 12: val_loss did not improve from 0.00173
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 73ms/step - loss: 0.0020 - mae: 0.0297 - val_loss: 0.0019 - val_mae: 0.0307
Epoch 13/200
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 0.0021 - mae: 0.0312
Epoch 13: val_loss did not improve from 0.00173
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 73ms/step - loss: 0.0021 - mae: 0.0312 - val_loss: 0.0020 - val_mae: 0.0311
Epoch 14/200
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 0.0020 - mae: 0.0306
Epoch 14: val_loss did not improve from 0.00173
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 73ms/step - loss: 0.0020 - mae: 0.0307 - val_loss: 0.0020 - val_mae: 0.0318
Epoch 15/200
```

```
# Plotting the training and validation loss
plt.plot(history_simple_cnn.history['val_loss'],c="b")
```

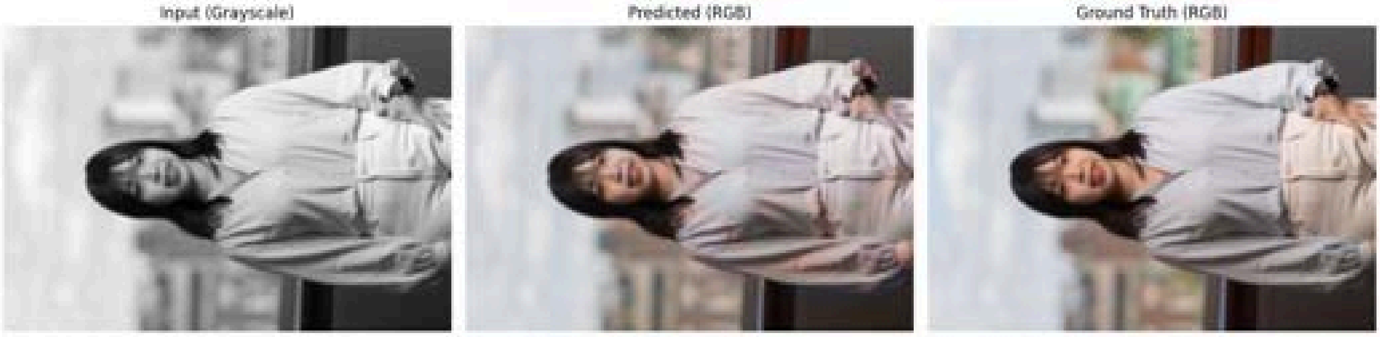
```
plt.plot(history_simple_cnn.history['loss'],c="r")
plt.legend(['Validation Loss','Training Loss'])
plt.show()
```

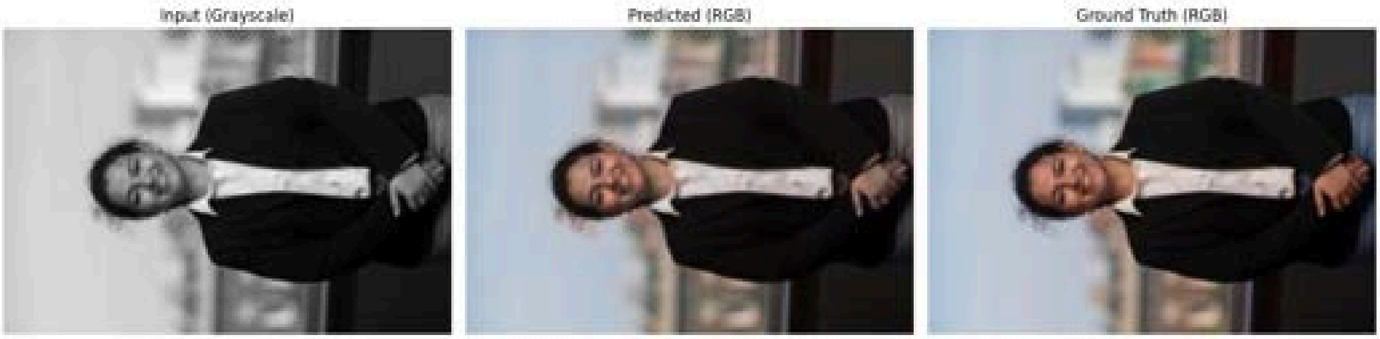


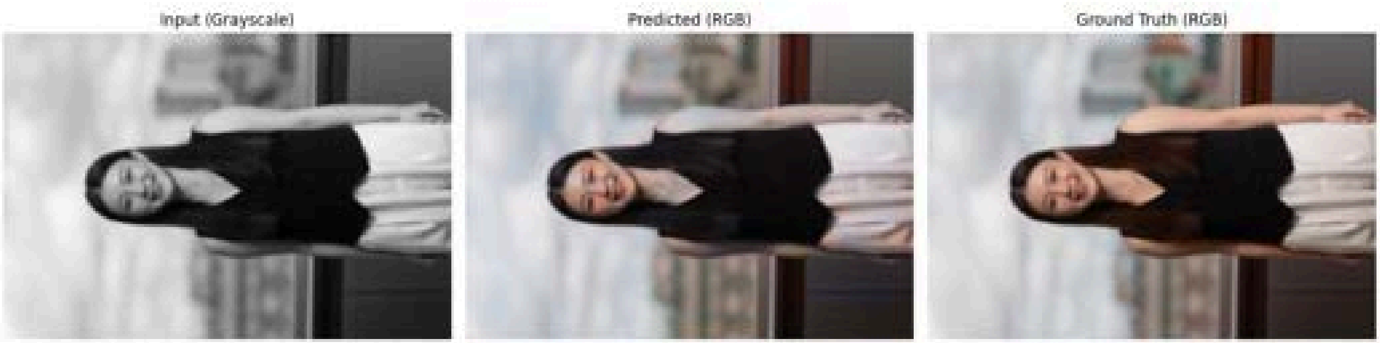
Depiction of Model Performance on images from the Training Set

```
show_predictions(best_simple_cnn, X, Y, num_samples=5)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 243ms/step
```

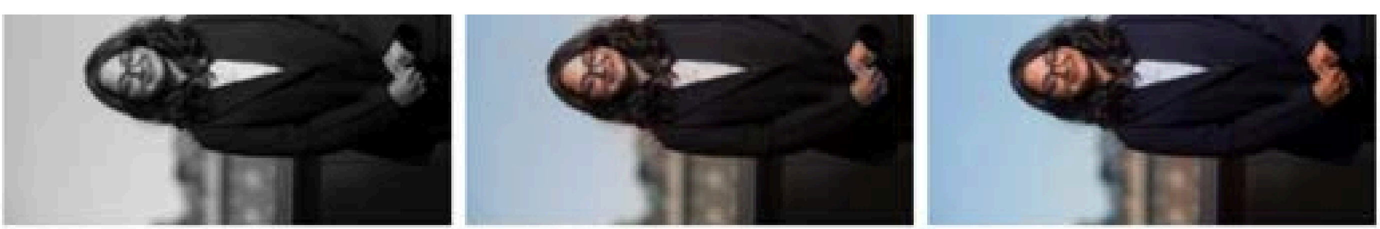




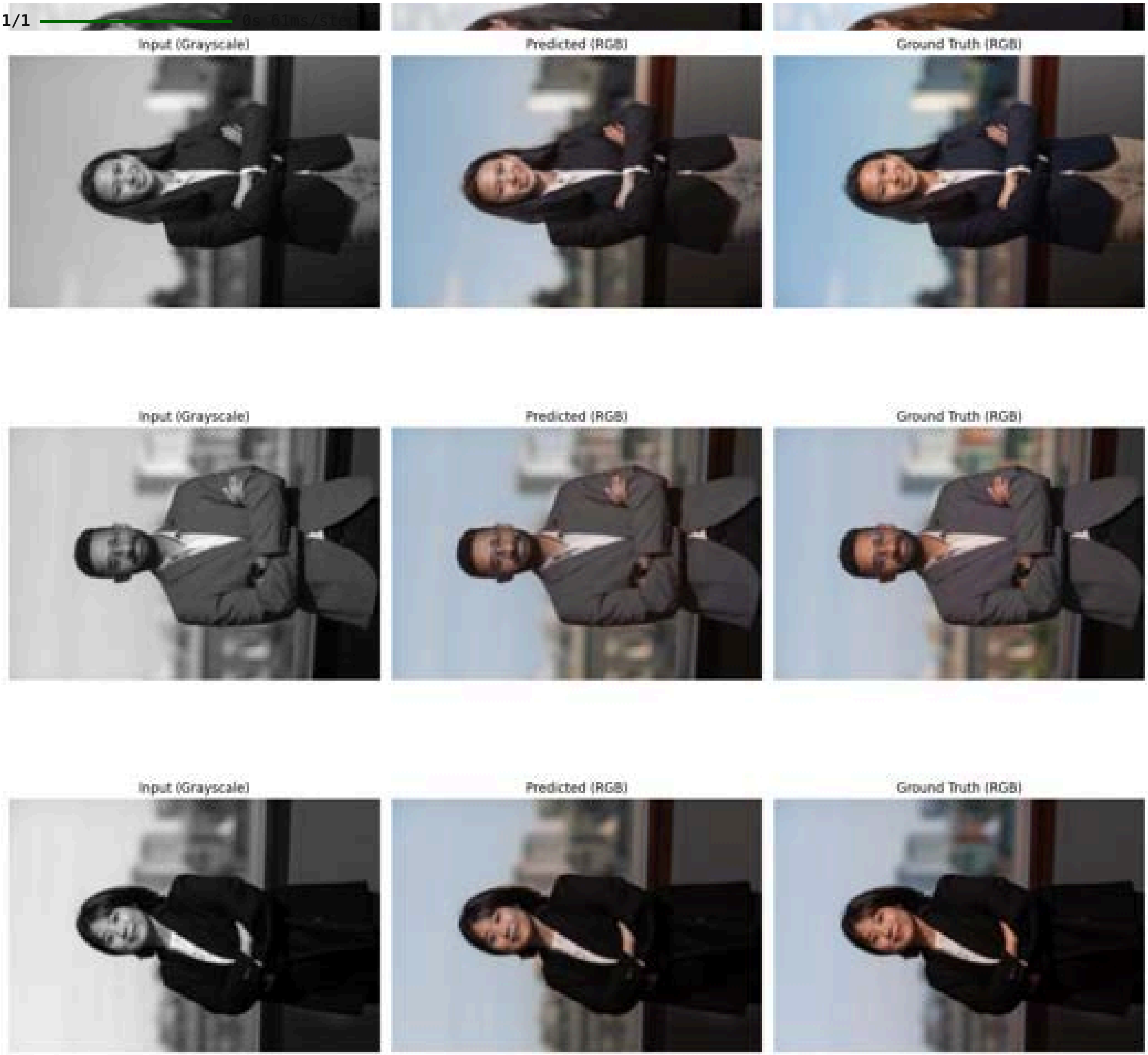


Depiction of Model Performance on images from the Test Set

```
show_predictions(best_simple_cnn, X_test, Y_test, num_samples=5)
```



1/1



The predicted image captures the overall structure and lighting quite well for headshots compared to full scene portraits. However, the colors are slightly muted and less vibrant than the ground truth. This suggests that the model learned the general color distribution but struggles with fine color accuracy, especially for clothing or skin tones.

Model 3: Improved CNN

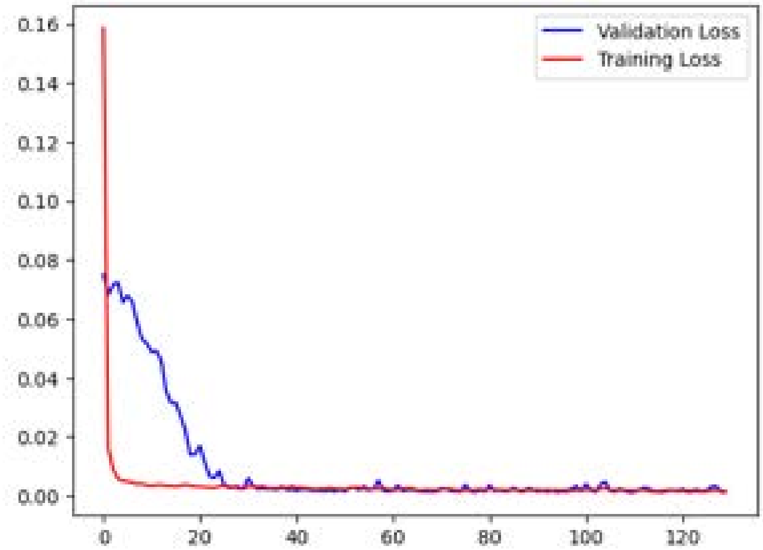
```
# Defining the architecture for the improved CNN model
def build_improved_cnn():
    inputs = layers.Input(shape=(120, 176, 1), name="grayscale_input")
    x1 = layers.Conv2D(64, (3, 3), padding='same')(inputs)
    x1 = layers.BatchNormalization()(x1)
    x1 = layers.LeakyReLU(negative_slope=0.1)(x1)
    p1 = layers.MaxPooling2D((2, 2))(x1)
    x2 = layers.Conv2D(128, (3, 3), padding='same')(p1)
    x2 = layers.BatchNormalization()(x2)
    x2 = layers.LeakyReLU(negative_slope=0.1)(x2)
    p2 = layers.MaxPooling2D((2, 2))(x2)
    x = layers.Conv2D(256, (3, 3), padding='same')(p2)
```

```
    x = layers.BatchNormalization()(x)
    x = layers.LeakyReLU(negative_slope=0.1)(x)
    x = layers.UpSampling2D((2, 2))(x)
    x = layers.Conv2D(128, (3, 3), padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.LeakyReLU(negative_slope=0.1)(x)
    x = layers.UpSampling2D((2, 2))(x)
    x = layers.Conv2D(64, (3, 3), padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.LeakyReLU(negative_slope=0.1)(x)
    outputs = layers.Conv2D(3, (3, 3), activation='tanh', padding='same', name="ab_output")(x)
    return models.Model(inputs=inputs, outputs=outputs, name="ColorizationModel")


improved_cnn = build_improved_cnn()
improved_cnn.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Defining the path to save the best model
model_checkpoint_path = os.path.join(best_models_path, 'improved_cnn.keras')
model_checkpoint = ModelCheckpoint(model_checkpoint_path, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

# Fitting the model to the training data
history_improved_cnn = improved_cnn.fit(X, Y, batch_size=16, epochs=1000, validation_split=0.1, callbacks=[early_stop, mode

best_improved_cnn = keras.models.load_model(os.path.join(best_models_path, 'improved_cnn.keras'))
```

```
Epoch 1/1000
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 372ms/step - loss: 0.3606 - mae: 0.4161
Epoch 1: val_loss improved from inf to 0.07506, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/Best
26/26 ━━━━━━━━━━━━━━━━━━━━ 20s 497ms/step - loss: 0.3531 - mae: 0.4102 - val_loss: 0.0751 - val_mae: 0.2411
Epoch 2/1000
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - loss: 0.0192 - mae: 0.1100
Epoch 2: val_loss improved from 0.07506 to 0.06820, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 107ms/step - loss: 0.0190 - mae: 0.1092 - val_loss: 0.0682 - val_mae: 0.2290
Epoch 3/1000
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step - loss: 0.0100 - mae: 0.0764
Epoch 3: val_loss did not improve from 0.06820
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 103ms/step - loss: 0.0099 - mae: 0.0760 - val_loss: 0.0717 - val_mae: 0.2362
Epoch 4/1000
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step - loss: 0.0060 - mae: 0.0560
Epoch 4: val_loss did not improve from 0.06820
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 101ms/step - loss: 0.0060 - mae: 0.0559 - val_loss: 0.0724 - val_mae: 0.2418
Epoch 5/1000
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - loss: 0.0050 - mae: 0.0504
Epoch 5: val_loss improved from 0.06820 to 0.06559, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 106ms/step - loss: 0.0050 - mae: 0.0505 - val_loss: 0.0656 - val_mae: 0.2316
Epoch 6/1000
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step - loss: 0.0047 - mae: 0.0495
Epoch 6: val_loss did not improve from 0.06559
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 105ms/step - loss: 0.0047 - mae: 0.0496 - val_loss: 0.0679 - val_mae: 0.2353
Epoch 7/1000
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - loss: 0.0048 - mae: 0.0504
Epoch 7: val_loss did not improve from 0.06559
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 101ms/step - loss: 0.0047 - mae: 0.0502 - val_loss: 0.0657 - val_mae: 0.2313
Epoch 8/1000
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - loss: 0.0038 - mae: 0.0443
Epoch 8: val_loss improved from 0.06559 to 0.05880, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 107ms/step - loss: 0.0038 - mae: 0.0443 - val_loss: 0.0588 - val_mae: 0.2202
Epoch 9/1000
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - loss: 0.0040 - mae: 0.0459
Epoch 9: val_loss improved from 0.05880 to 0.05317, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 103ms/step - loss: 0.0040 - mae: 0.0459 - val_loss: 0.0532 - val_mae: 0.2096
Epoch 10/1000
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - loss: 0.0035 - mae: 0.0425
Epoch 10: val_loss improved from 0.05317 to 0.05167, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 105ms/step - loss: 0.0035 - mae: 0.0425 - val_loss: 0.0517 - val_mae: 0.2070
Epoch 11/1000
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - loss: 0.0034 - mae: 0.0422
Epoch 11: val_loss improved from 0.05167 to 0.04880, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 105ms/step - loss: 0.0034 - mae: 0.0422 - val_loss: 0.0488 - val_mae: 0.2010
Epoch 12/1000
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - loss: 0.0036 - mae: 0.0439
Epoch 12: val_loss did not improve from 0.04880
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 99ms/step - loss: 0.0036 - mae: 0.0440 - val_loss: 0.0492 - val_mae: 0.2020
Epoch 13/1000
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step - loss: 0.0037 - mae: 0.0455
Epoch 13: val_loss improved from 0.04880 to 0.04581, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 110ms/step - loss: 0.0037 - mae: 0.0455 - val_loss: 0.0458 - val_mae: 0.1949
Epoch 14/1000
```

```
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step - loss: 0.0032 - mae: 0.0410
Epoch 14: val_loss improved from 0.04581 to 0.03554, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 110ms/step - loss: 0.0032 - mae: 0.0411 - val_loss: 0.0355 - val_mae: 0.1684
```

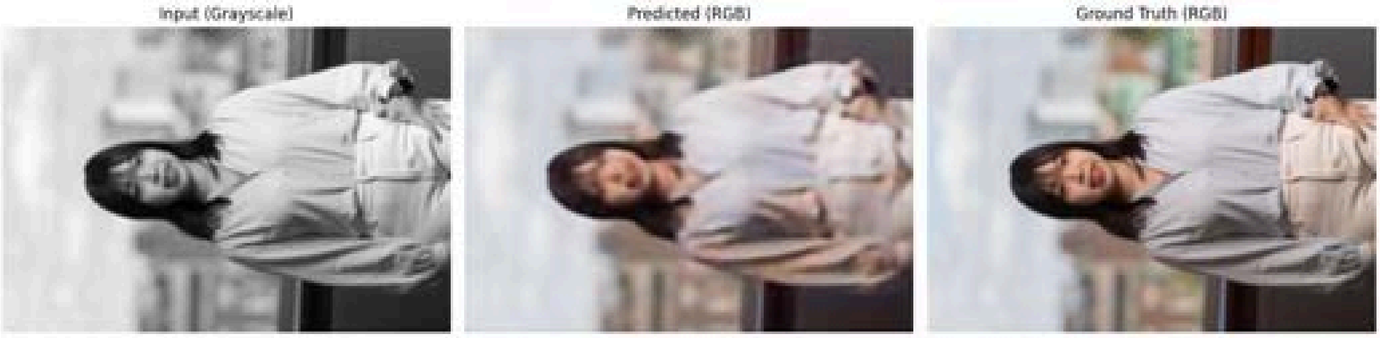
```
# Plotting the training and validation loss
plt.plot(history_improved_cnn.history['val_loss'],c="b")
plt.plot(history_improved_cnn.history['loss'],c="r")
plt.legend(['Validation Loss','Training Loss'])
plt.show()
```

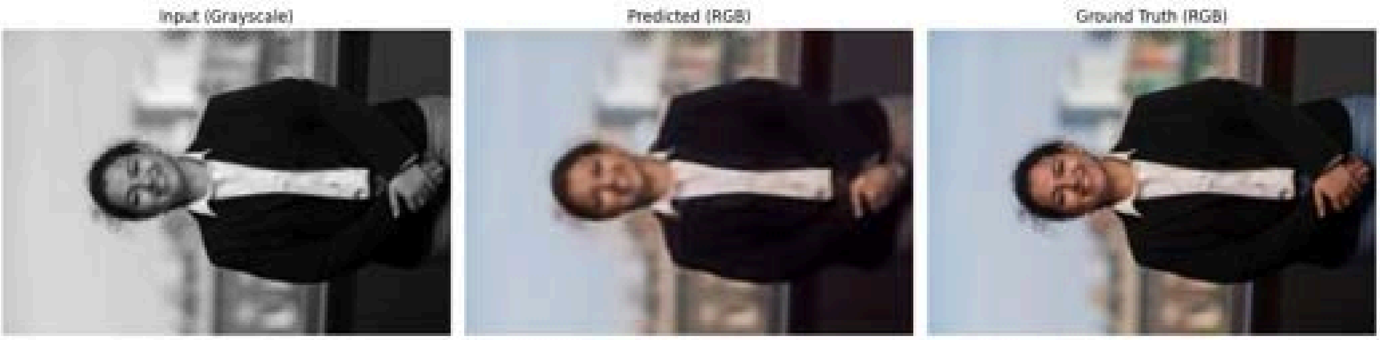


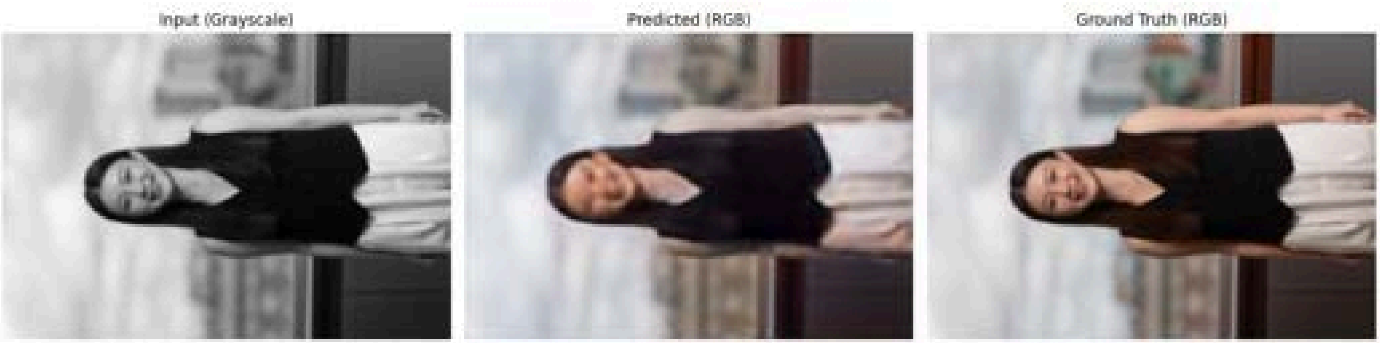
Depiction of Model Performance on images from the Training Set

```
show_predictions(best_improved_cnn, X, Y, num_samples=5)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 608ms/step
```

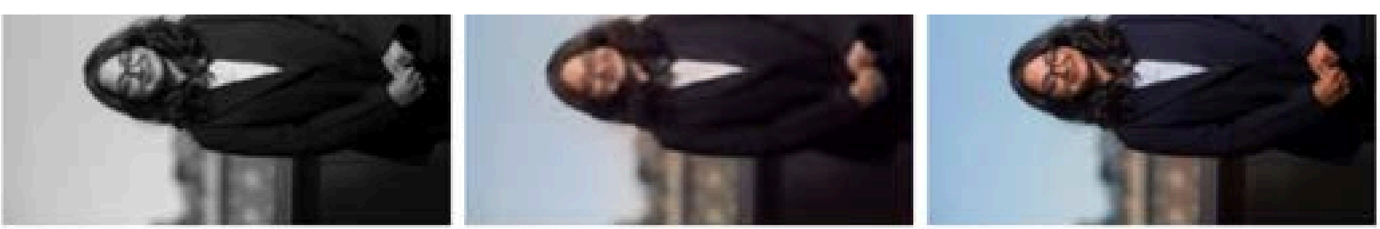




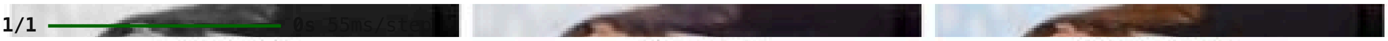


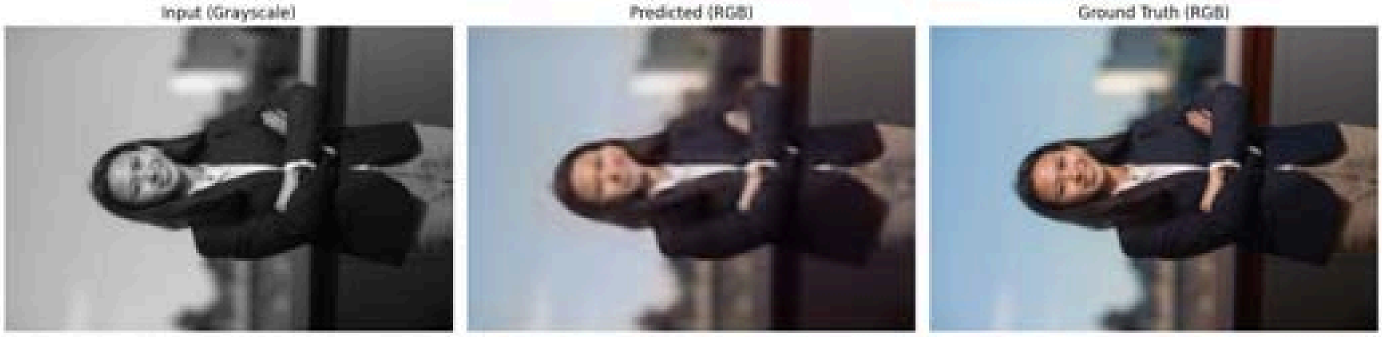
Depiction of Model Performance on images from the Test Set

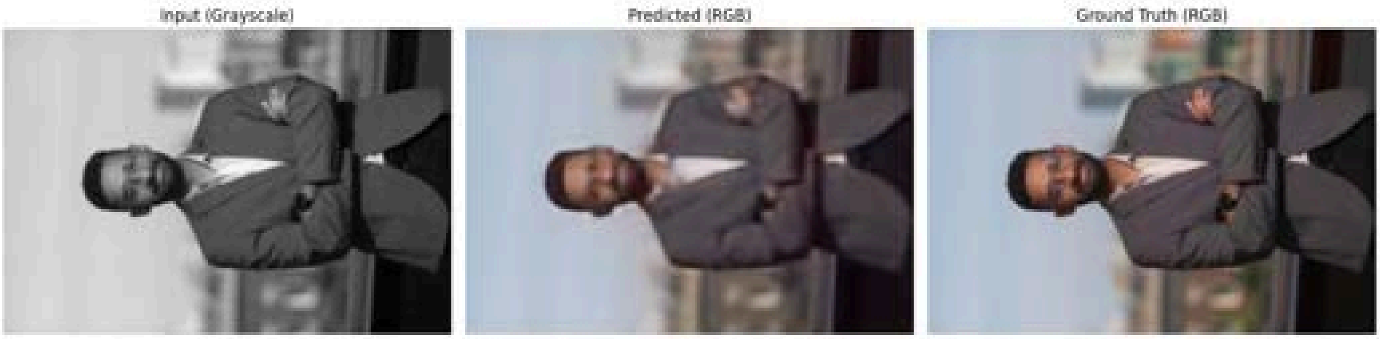

```
show_predictions(best_improved_cnn, X_test, Y_test, num_samples=5)
```

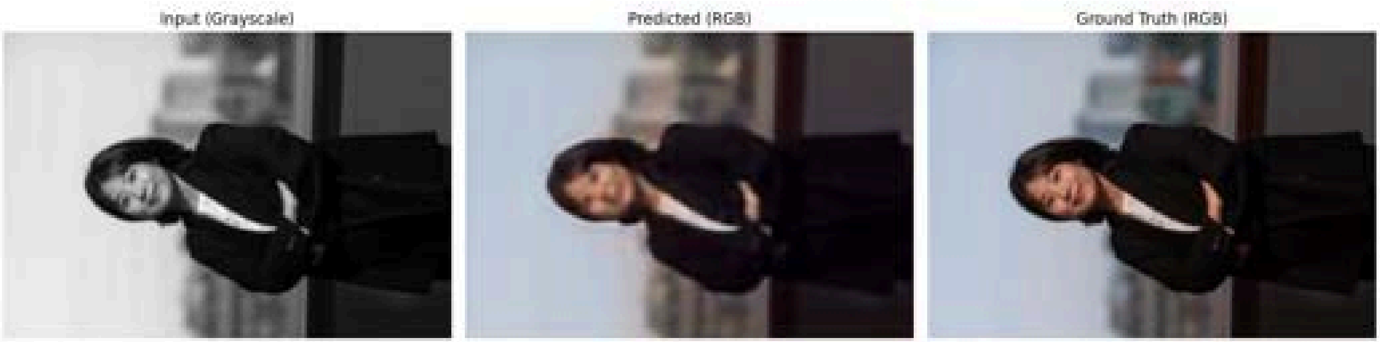


1/1

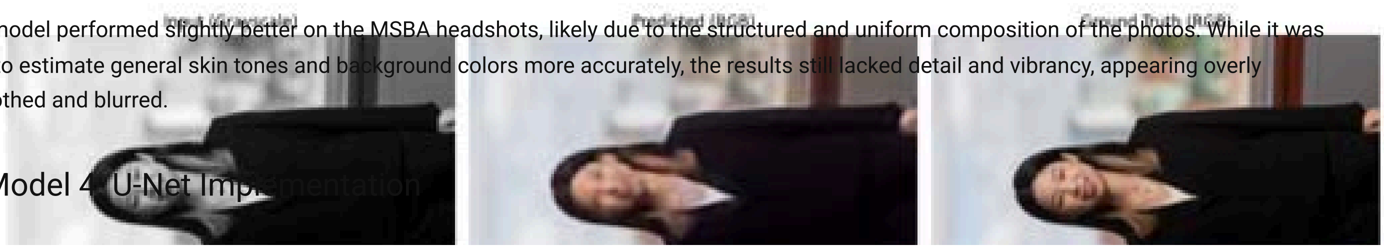








The model performed slightly better on the MSBA headshots, likely due to the structured and uniform composition of the photos. While it was able to estimate general skin tones and background colors more accurately, the results still lacked detail and vibrancy, appearing overly smoothed and blurred.



Model 4: U-Net Implementation

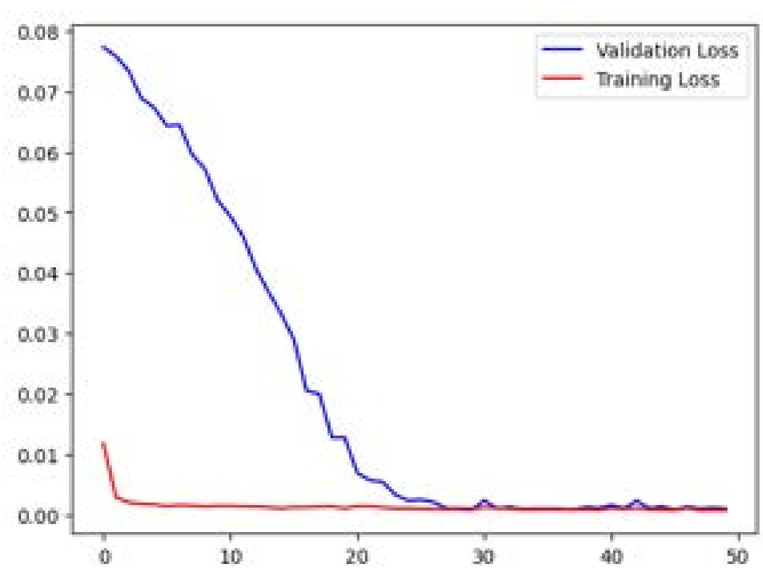
```
# Defining the architecture for the U-Net model
def build_unet_model(input_shape=(120, 176, 1)):
    inputs = layers.Input(shape=input_shape)

    # Encoder
    conv1 = layers.Conv2D(64, (3, 3), padding='same')(inputs)
    conv1 = layers.BatchNormalization()(conv1)
    conv1 = layers.ReLU()(conv1)
    conv1 = layers.Conv2D(64, (3, 3), padding='same')(conv1)
    conv1 = layers.BatchNormalization()(conv1)
    conv1 = layers.ReLU()(conv1)
    pool1 = layers.MaxPooling2D((2, 2))(conv1)
```

```
    conv2 = layers.Conv2D(128, (3, 3), padding='same')(pool1)
    conv2 = layers.BatchNormalization()(conv2)
    conv2 = layers.ReLU()(conv2)
    conv2 = layers.Conv2D(128, (3, 3), padding='same')(conv2)
    conv2 = layers.BatchNormalization()(conv2)
    conv2 = layers.ReLU()(conv2)
    pool2 = layers.MaxPooling2D((2, 2))(conv2)

    # Bottleneck
    bottleneck = layers.Conv2D(256, (3, 3), padding='same')(pool2)
    bottleneck = layers.BatchNormalization()(bottleneck)
    bottleneck = layers.ReLU()(bottleneck)

    # Decoder
    up2 = layers.Conv2DTranspose(128, (2, 2), strides=2, padding='same')(bottleneck)
    up2 = layers.Concatenate()([up2, conv2])
    up2 = layers.Conv2D(128, (3, 3), padding='same')(up2)
    up2 = layers.BatchNormalization()(up2)
    up2 = layers.ReLU()(up2)

    up1 = layers.Conv2DTranspose(64, (2, 2), strides=2, padding='same')(up2)
    up1 = layers.Concatenate()([up1, conv1])
    up1 = layers.Conv2D(64, (3, 3), padding='same')(up1)
    up1 = layers.BatchNormalization()(up1)
    up1 = layers.ReLU()(up1)

    outputs = layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')(up1)

    model = models.Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model

unet_model = build_unet_model()


# Defining the path to save the best model
model_checkpoint_path = os.path.join(best_models_path, 'unet_model.keras')
model_checkpoint = ModelCheckpoint(model_checkpoint_path, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

# fitting the model to the training data
history_unet_model = unet_model.fit(X, Y, batch_size=16, epochs=200, validation_split=0.1, callbacks=[early_stop, model_chec

best_unet_model = keras.models.load_model(os.path.join(best_models_path, 'unet_model.keras'))
```

```
Epoch 1/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 656ms/step - loss: 0.0307 - mae: 0.1103
Epoch 1: val_loss improved from inf to 0.07734, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/Best
26/26 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - loss: 0.0300 - mae: 0.1086 - val_loss: 0.0773 - val_mae: 0.2498
Epoch 2/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 144ms/step - loss: 0.0031 - mae: 0.0407
Epoch 2: val_loss improved from 0.07734 to 0.07580, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 10s 159ms/step - loss: 0.0031 - mae: 0.0407 - val_loss: 0.0758 - val_mae: 0.2488
Epoch 3/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 144ms/step - loss: 0.0023 - mae: 0.0354
Epoch 3: val_loss improved from 0.07580 to 0.07336, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 4s 168ms/step - loss: 0.0023 - mae: 0.0353 - val_loss: 0.0734 - val_mae: 0.2453
Epoch 4/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step - loss: 0.0017 - mae: 0.0297
Epoch 4: val_loss improved from 0.07336 to 0.06890, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 4s 163ms/step - loss: 0.0017 - mae: 0.0297 - val_loss: 0.0689 - val_mae: 0.2385
Epoch 5/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step - loss: 0.0018 - mae: 0.0310
Epoch 5: val_loss improved from 0.06890 to 0.06733, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 4s 165ms/step - loss: 0.0018 - mae: 0.0309 - val_loss: 0.0673 - val_mae: 0.2367
Epoch 6/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step - loss: 0.0014 - mae: 0.0267
Epoch 6: val_loss improved from 0.06733 to 0.06430, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 176ms/step - loss: 0.0014 - mae: 0.0267 - val_loss: 0.0643 - val_mae: 0.2314
Epoch 7/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step - loss: 0.0015 - mae: 0.0280
Epoch 7: val_loss did not improve from 0.06430
26/26 ━━━━━━━━━━━━━━━━━━━━ 4s 159ms/step - loss: 0.0015 - mae: 0.0280 - val_loss: 0.0644 - val_mae: 0.2318
Epoch 8/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step - loss: 0.0016 - mae: 0.0297
Epoch 8: val_loss improved from 0.06430 to 0.05954, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 178ms/step - loss: 0.0016 - mae: 0.0296 - val_loss: 0.0595 - val_mae: 0.2224
Epoch 9/200
```

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step - loss: 0.0014 - mae: 0.0270
Epoch 9: val_loss improved from 0.05954 to 0.05717, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 176ms/step - loss: 0.0014 - mae: 0.0270 - val_loss: 0.0572 - val_mae: 0.2179
Epoch 10/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step - loss: 0.0015 - mae: 0.0283
Epoch 10: val_loss improved from 0.05717 to 0.05200, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
26/26 ━━━━━━━━━━━━━━━━━━━━ 4s 163ms/step - loss: 0.0015 - mae: 0.0282 - val_loss: 0.0520 - val_mae: 0.2076
Epoch 11/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 146ms/step - loss: 0.0014 - mae: 0.0277
Epoch 11: val_loss improved from 0.05200 to 0.04937, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
26/26 ━━━━━━━━━━━━━━━━━━━━ 4s 167ms/step - loss: 0.0014 - mae: 0.0277 - val_loss: 0.0494 - val_mae: 0.2019
Epoch 12/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 143ms/step - loss: 0.0013 - mae: 0.0266
Epoch 12: val_loss improved from 0.04937 to 0.04606, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 166ms/step - loss: 0.0013 - mae: 0.0266 - val_loss: 0.0461 - val_mae: 0.1951
Epoch 13/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step - loss: 0.0012 - mae: 0.0254
Epoch 13: val_loss improved from 0.04606 to 0.04078, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
26/26 ━━━━━━━━━━━━━━━━━━━━ 4s 158ms/step - loss: 0.0012 - mae: 0.0254 - val_loss: 0.0408 - val_mae: 0.1840
Epoch 14/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step - loss: 0.0015 - mae: 0.0283
Epoch 14: val_loss improved from 0.04078 to 0.03688, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
26/26 ━━━━━━━━━━━━━━━━━━━━ 4s 158ms/step - loss: 0.0015 - mae: 0.0282 - val_loss: 0.0369 - val_mae: 0.1756
Epoch 15/200
```

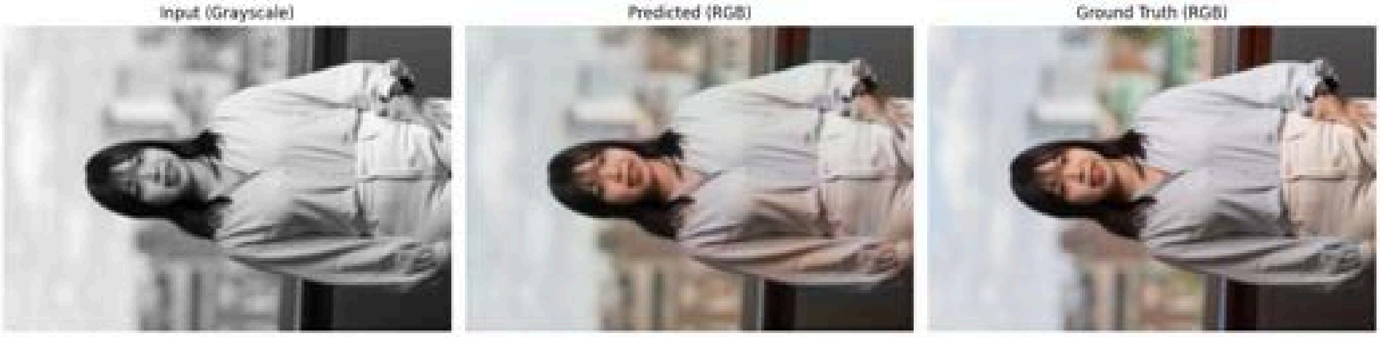
```
# Plotting the training and validation loss
plt.plot(history_unet_model.history['val_loss'],c="b")
plt.plot(history_unet_model.history['loss'],c="r")
plt.legend(['Validation Loss','Training Loss'])
plt.show()
```

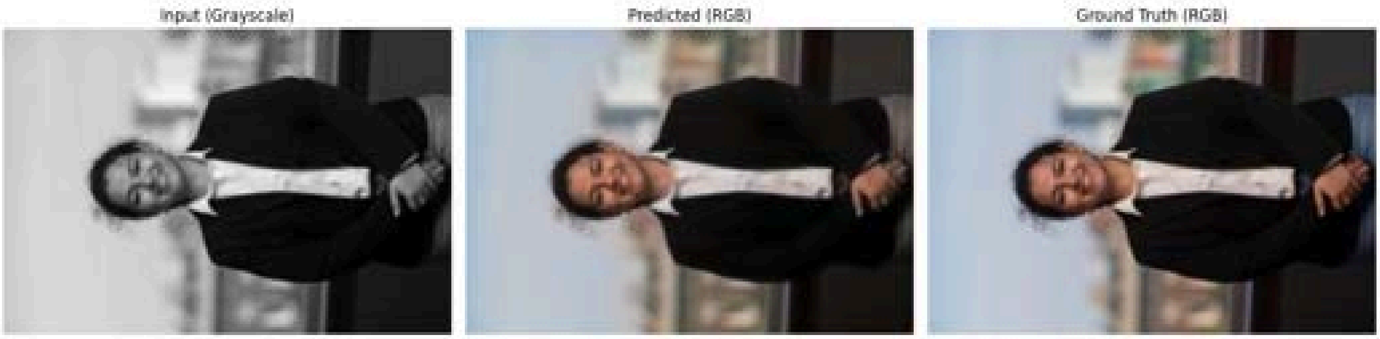


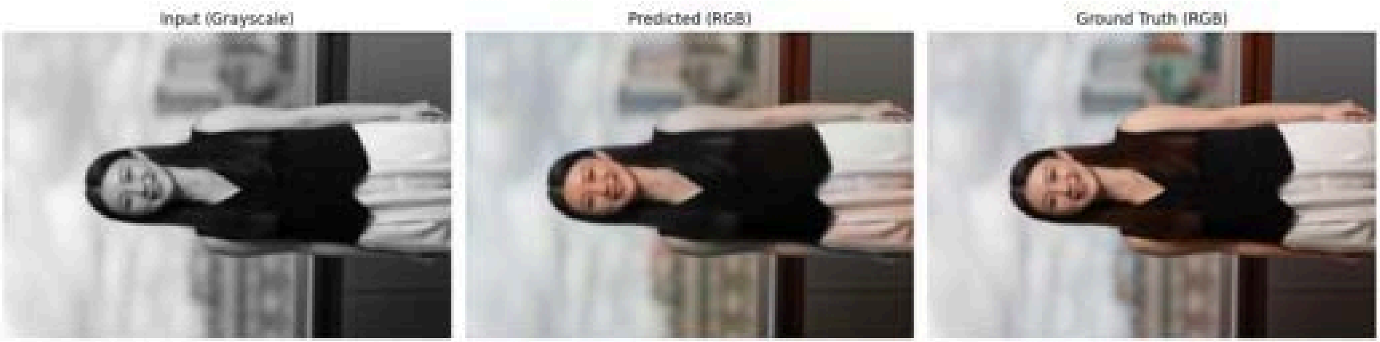
Depiction of Model Performance on images from the Training Set

```
show_predictions(best_unet_model, X, Y, num_samples=5)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 606ms/step
```

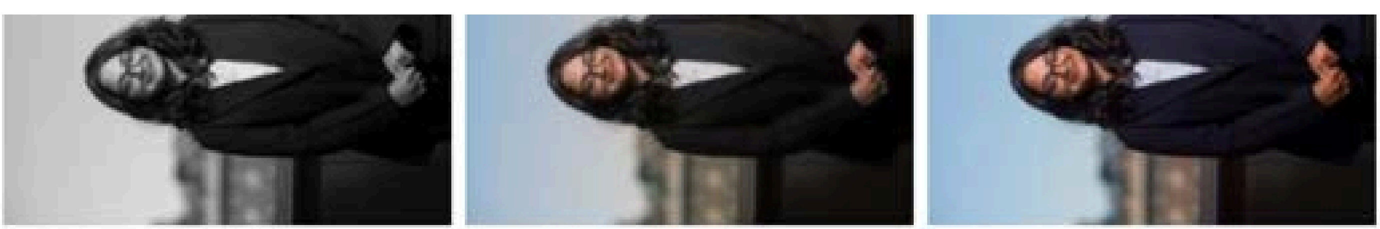


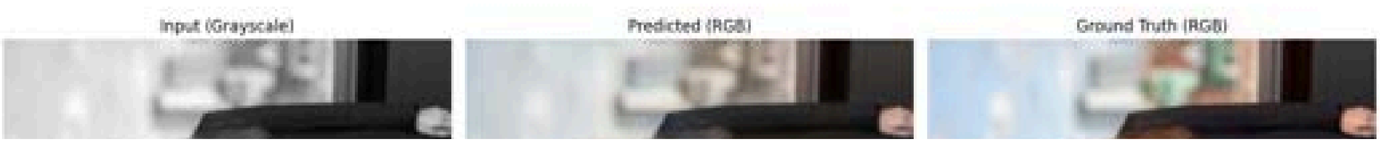




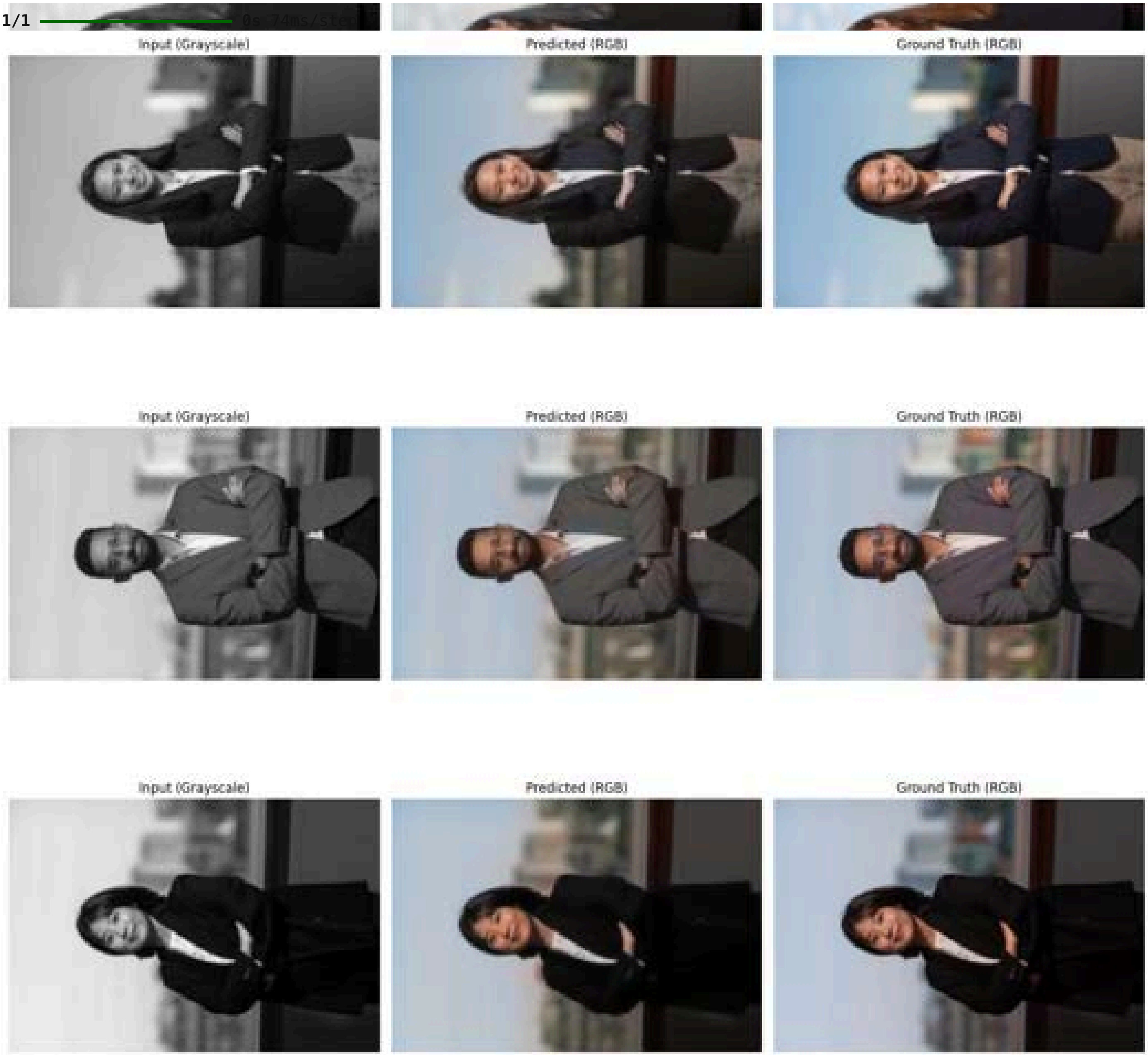
Depiction of Model Performance on images from the Test Set

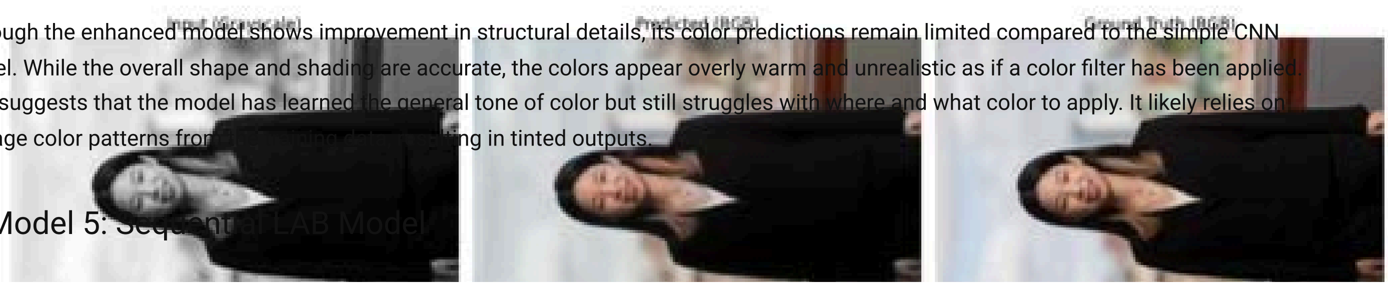
```
show_predictions(best_unet_model, X_test, Y_test, num_samples=5)
```





1/1





Although the enhanced model shows improvement in structural details, its color predictions remain limited compared to the simple CNN model. While the overall shape and shading are accurate, the colors appear overly warm and unrealistic as if a color filter has been applied. This suggests that the model has learned the general tone of color but still struggles with where and what color to apply. It likely relies on average color patterns from the training data, resulting in tinted outputs.

Model 5: Sequential LAB Model

```
# Defining the architecture for the sequential LAB model
# Encoder
seq_lab_model = Sequential()
seq_lab_model.add(Conv2D(64, (3, 3), activation='relu', padding='same', strides=2, input_shape=(120, 176, 1)))
seq_lab_model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
seq_lab_model.add(Conv2D(128, (3,3), activation='relu', padding='same', strides=2))
seq_lab_model.add(Conv2D(256, (3,3), activation='relu', padding='same'))
seq_lab_model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
seq_lab_model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
seq_lab_model.add(Conv2D(256, (3,3), activation='relu', padding='same', strides=2))
seq_lab_model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
```

```
seq_lab_model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
seq_lab_model.add(Conv2D(256, (3,3), activation='relu', padding='same'))
# Decoder
seq_lab_model.add(Conv2D(128, (3,3), activation='relu', padding='same'))
seq_lab_model.add(UpSampling2D((2, 2)))
seq_lab_model.add(Conv2D(64, (3,3), activation='relu', padding='same'))
seq_lab_model.add(UpSampling2D((2, 2)))
seq_lab_model.add(Conv2D(32, (3,3), activation='relu', padding='same'))
seq_lab_model.add(Conv2D(16, (3,3), activation='relu', padding='same'))
seq_lab_model.add(Conv2D(2, (3, 3), activation='tanh', padding='same'))
seq_lab_model.add(UpSampling2D((2, 2)))
seq_lab_model.compile(optimizer=Adam(learning_rate=1e-4), loss='mae', metrics=['mse'])
seq_lab_model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `ir
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_68 (Conv2D)	(None, 60, 88, 64)	640
conv2d_69 (Conv2D)	(None, 60, 88, 128)	73,856
conv2d_70 (Conv2D)	(None, 30, 44, 128)	147,584
conv2d_71 (Conv2D)	(None, 30, 44, 256)	295,168
conv2d_72 (Conv2D)	(None, 30, 44, 512)	1,180,160
conv2d_73 (Conv2D)	(None, 30, 44, 512)	2,359,808
conv2d_74 (Conv2D)	(None, 15, 22, 256)	1,179,904
conv2d_75 (Conv2D)	(None, 15, 22, 512)	1,180,160
conv2d_76 (Conv2D)	(None, 15, 22, 512)	2,359,808
conv2d_77 (Conv2D)	(None, 15, 22, 256)	1,179,904
conv2d_78 (Conv2D)	(None, 15, 22, 128)	295,040
up_sampling2d_14 (UpSampling2D)	(None, 30, 44, 128)	0
conv2d_79 (Conv2D)	(None, 30, 44, 64)	73,792
up_sampling2d_15 (UpSampling2D)	(None, 60, 88, 64)	0
conv2d_80 (Conv2D)	(None, 60, 88, 32)	18,464
conv2d_81 (Conv2D)	(None, 60, 88, 16)	4,624
conv2d_82 (Conv2D)	(None, 60, 88, 2)	290
up_sampling2d_16 (UpSampling2D)	(None, 120, 176, 2)	0

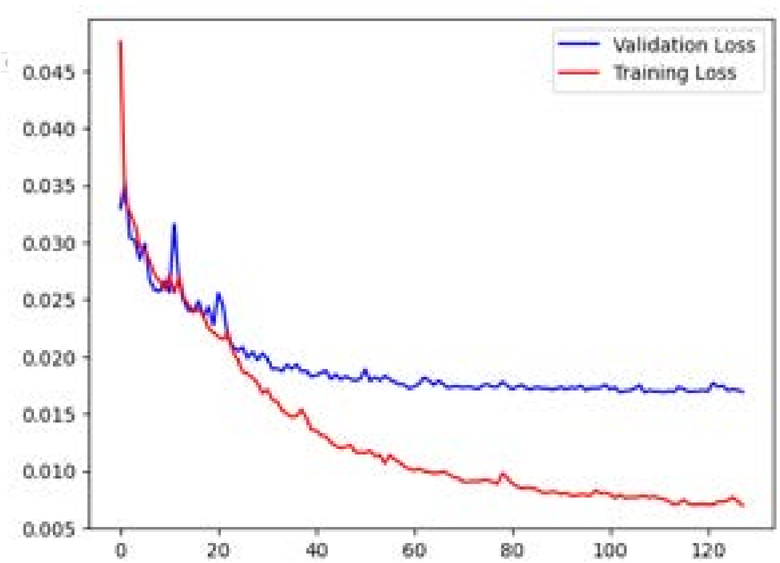
Total params: 10,349,202 (39.48 MB)
Trainable params: 10,349,202 (39.48 MB)
Non-trainable params: 0 (0.00 B)

```
# Defining the path to save the best model
model_checkpoint_path = os.path.join(best_models_path, 'seq_lab_model.keras')
model_checkpoint = ModelCheckpoint(model_checkpoint_path, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

# Fitting the model to the training data
history_seq_lab_model = seq_lab_model.fit(X_lab,Y_lab,batch_size=16, epochs=200, validation_split=0.1, callbacks=[early_sto

best_seq_lab_model = keras.models.load_model(os.path.join(best_models_path, 'seq_lab_model.keras'))
```

```
Epoch 1/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 696ms/step - loss: 0.0659 - mse: 0.0097
Epoch 1: val_loss improved from inf to 0.03299, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/Best
26/26 ━━━━━━━━━━━━━━━━━━━━ 39s 1s/step - loss: 0.0652 - mse: 0.0095 - val_loss: 0.0330 - val_mse: 0.0025
Epoch 2/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step - loss: 0.0337 - mse: 0.0025
Epoch 2: val_loss did not improve from 0.03299
26/26 ━━━━━━━━━━━━━━━━━━━━ 4s 163ms/step - loss: 0.0337 - mse: 0.0025 - val_loss: 0.0353 - val_mse: 0.0026
Epoch 3/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step - loss: 0.0331 - mse: 0.0024
```

```
Epoch 3: val_loss improved from 0.03299 to 0.03038, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 199ms/step - loss: 0.0331 - mse: 0.0024 - val_loss: 0.0304 - val_mse: 0.0021
Epoch 4/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 148ms/step - loss: 0.0317 - mse: 0.0022
Epoch 4: val_loss improved from 0.03038 to 0.03018, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 10s 189ms/step - loss: 0.0317 - mse: 0.0022 - val_loss: 0.0302 - val_mse: 0.0020
Epoch 5/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step - loss: 0.0300 - mse: 0.0020
Epoch 5: val_loss improved from 0.03018 to 0.02847, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 190ms/step - loss: 0.0300 - mse: 0.0020 - val_loss: 0.0285 - val_mse: 0.0018
Epoch 6/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step - loss: 0.0296 - mse: 0.0019
Epoch 6: val_loss did not improve from 0.02847
26/26 ━━━━━━━━━━━━━━━━━━━━ 4s 163ms/step - loss: 0.0296 - mse: 0.0019 - val_loss: 0.0299 - val_mse: 0.0020
Epoch 7/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step - loss: 0.0293 - mse: 0.0019
Epoch 7: val_loss improved from 0.02847 to 0.02672, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 7s 260ms/step - loss: 0.0293 - mse: 0.0019 - val_loss: 0.0267 - val_mse: 0.0015
Epoch 8/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step - loss: 0.0276 - mse: 0.0017
Epoch 8: val_loss improved from 0.02672 to 0.02586, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 9s 190ms/step - loss: 0.0276 - mse: 0.0017 - val_loss: 0.0259 - val_mse: 0.0015
Epoch 9/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step - loss: 0.0263 - mse: 0.0016
Epoch 9: val_loss improved from 0.02586 to 0.02570, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/B
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 195ms/step - loss: 0.0263 - mse: 0.0016 - val_loss: 0.0257 - val_mse: 0.0015
Epoch 10/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step - loss: 0.0250 - mse: 0.0014
Epoch 10: val_loss did not improve from 0.02570
26/26 ━━━━━━━━━━━━━━━━━━━━ 4s 161ms/step - loss: 0.0250 - mse: 0.0014 - val_loss: 0.0266 - val_mse: 0.0015
Epoch 11/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step - loss: 0.0268 - mse: 0.0016
Epoch 11: val_loss improved from 0.02570 to 0.02561, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
26/26 ━━━━━━━━━━━━━━━━━━━━ 6s 241ms/step - loss: 0.0268 - mse: 0.0016 - val_loss: 0.0256 - val_mse: 0.0014
Epoch 12/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step - loss: 0.0259 - mse: 0.0015
Epoch 12: val_loss did not improve from 0.02561
26/26 ━━━━━━━━━━━━━━━━━━━━ 4s 164ms/step - loss: 0.0259 - mse: 0.0015 - val_loss: 0.0316 - val_mse: 0.0021
Epoch 13/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 156ms/step - loss: 0.0273 - mse: 0.0016
Epoch 13: val_loss did not improve from 0.02561
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 171ms/step - loss: 0.0273 - mse: 0.0016 - val_loss: 0.0265 - val_mse: 0.0015
Epoch 14/200
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step - loss: 0.0244 - mse: 0.0013
Epoch 14: val_loss improved from 0.02561 to 0.02480, saving model to /content/drive/MyDrive/BA865/BA865 Group Project/
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 204ms/step - loss: 0.0244 - mse: 0.0013 - val_loss: 0.0248 - val_mse: 0.0013
Epoch 15/200
```

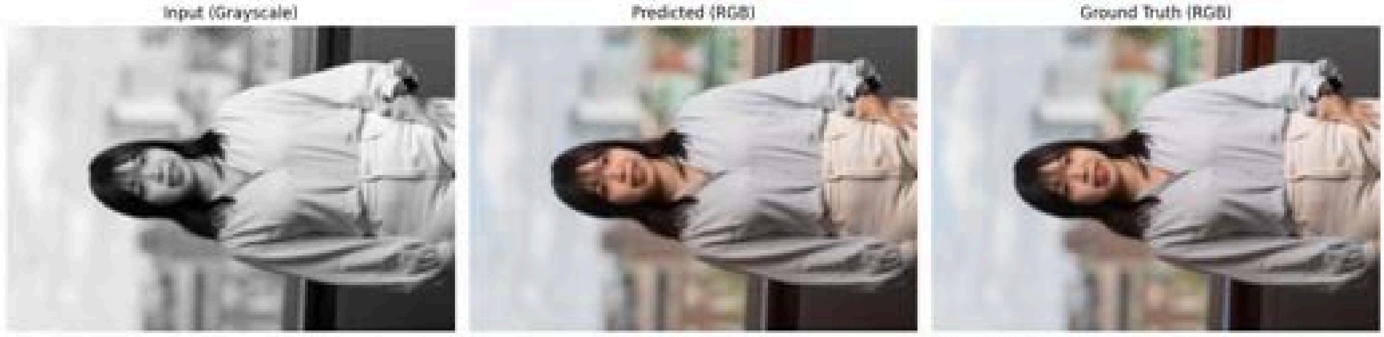
```python
# Plotting the training and validation loss
plt.plot(history_seq_lab_model.history['val_loss'],c="b")
plt.plot(history_seq_lab_model.history['loss'],c="r")
plt.legend(['Validation Loss','Training Loss'])
plt.show()
```

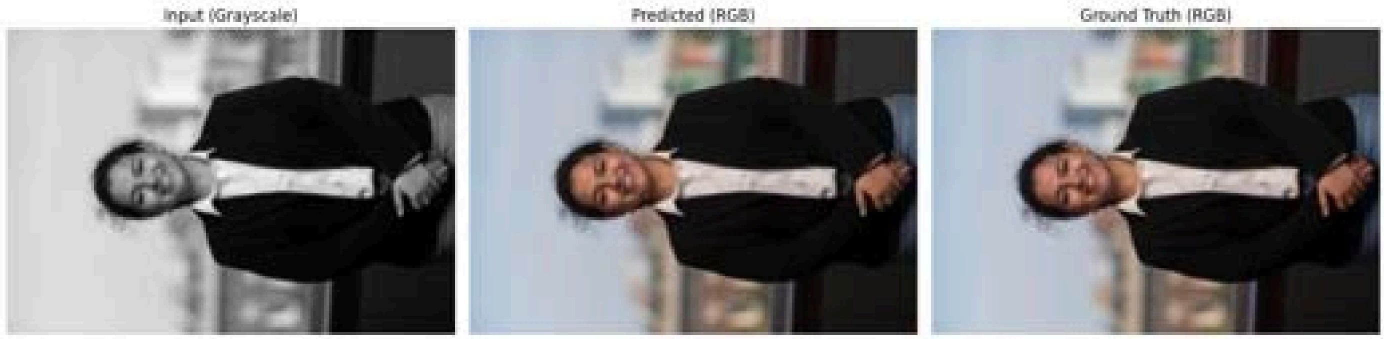


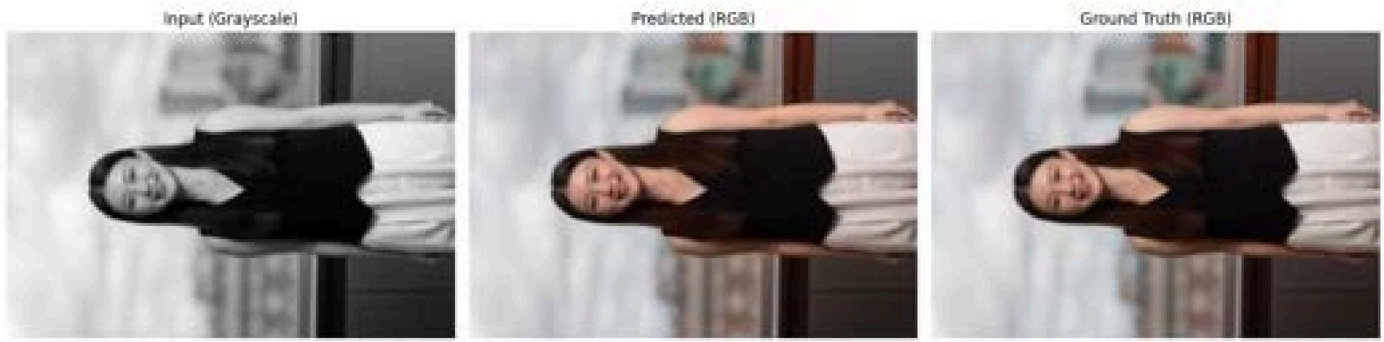
Depiction of Model Performance on images from the Training Set

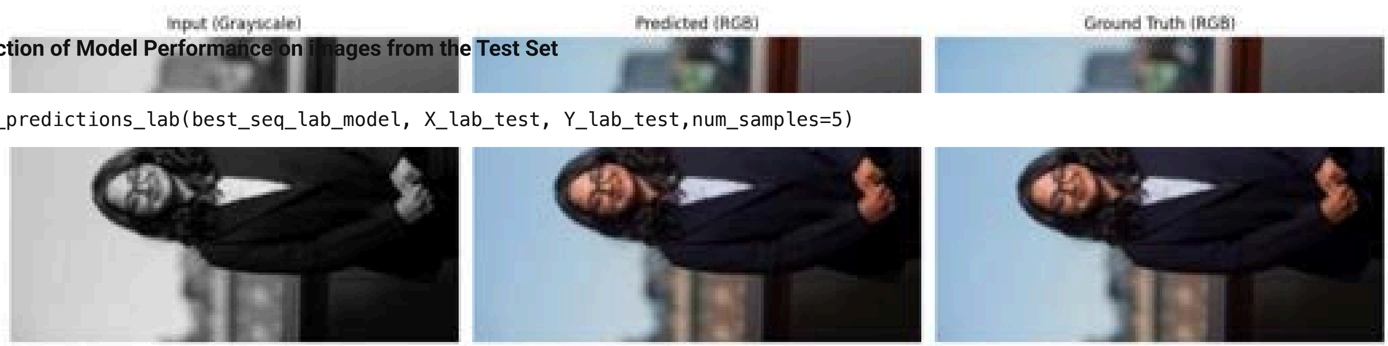
```
show_predictions_lab(best_seq_lab_model, X_lab, Y_lab, num_samples=5)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 748ms/step
```









Depiction of Model Performance on Images from the Test Set

```
show_predictions_lab(best_seq_lab_model, X_lab_test, Y_lab_test,num_samples=5)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step
```