Assignment #3

Code Checks for Missing and Null Values (Data Issues)

The provided code does not explicitly check for null or missing values in the dataset of images. However, there are a few points to note:

1. **Loading Images:** The load_images function loads images from the specified folders using OpenCV (cv2). If an image cannot be read (e.g. due to a corrupted file or unsupported format), the cv2.imread function will return 'None' for that image. The code does include a check to see if 'image is not None' before processing an image, so it effectively skips images that cannot be read.

2. **Image Normalization**: After loading and processing the images, they are normalized to the [0, 1] range by dividing by 255.0. This is a common practice in image processing and machine learning to ensure that pixel values are in a suitable range.

3. **Data Conversion:** The code converts the lists of images (foggy_images and clear_images) to NumPy arrays. If any issues related to missing or null values were present in the loaded images, they would likely manifest as errors during this conversion or normalization process.

4. **Training:** The loaded and preprocessed images (foggy_images and clear_images) are then used for training the dehazing model. The model's compile and fit functions are called to train the model using the provided data.

In summary, while the code doesn't explicitly check for null or missing values, it does handle cases where images cannot be read or loaded using the 'image is not None' check. Any images that cannot be read would be effectively excluded from the training process. It's important to ensure that the image folders contain valid image files in a supported format.

Code Variations/Changes Comparison (Model Issues)

Attempt #	Attempt Title	Optimizer	Loss Function	Learning Rate	Epochs	Batch Size	Loss Value	Validation Loss
1	ConvNet	Adam	MSE	-	100	5	0.0464	-
2	MultiConvNet	Adam	MSE	-	1000	16	0.0034	-
3	DehazeNet(VGG16 Integrated)	Adam	Perceptual Loss(VGG16)	1.00E-04	1000	32	0.2232	0.5056

I compared my multiple attempts of improving the 1st model as each epoch takes over 40 minutes to train and due to the duration of the assignment, that is not possible to retrain the model with multiple new changes for comparison, as image regeneration model has a very high time complexity and space complexity.

Looking at the table above, we can see that the loss value didn't have much change. Actually with the final model (attempt #3), the loss is actually higher, that is due to the MSE, alone, is not effective for image regeneration case in calculating the loss correctly. For image regeneration, VGG16 is used which extracts the features of the predicted image and the actual expected image and supplies them to the MSE for calculation which provide a more effective loss value. You can see the code snippet for the VGG16 perceptual loss function below:

```python
# Perceptual loss using VGG16 features
vgg = VGG16(include_top=False, weights='imagenet', input_shape=(480, 640, 3))  # Update input shape
vgg.trainable = False
output_layer = vgg.get_layer('block3_conv3').output
vgg_model = Model(vgg.input, output_layer)

def perceptual_loss(y_true, y_pred):
    y_true_features = vgg_model(y_true)
    y_pred_features = vgg_model(y_pred)
    return mean_squared_error(y_true_features, y_pred_features)

# Compile the model with Adam optimizer and perceptual loss
model.compile(optimizer=Adam(learning_rate=1e-4), loss=perceptual_loss)
```

To outline the differences in each attempt, below I have listed their model structure:

Attempt 1:
Model composed of only 4 convolutional layers with same filter size (128) and no padding

Attempt 2:
Model composed of only 12 convolutional layers with same filter size (256), but with padding, trained over 4 datasets (separate models)

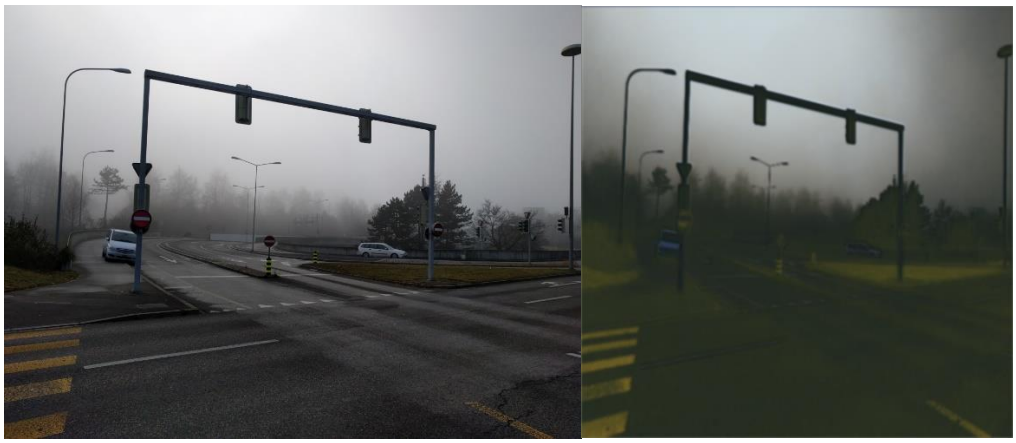
Attempt 3:
Model composed of Encoder decoder structure composed of multiple convolutions, maxpooling, upsampling to improve feature understanding by the model. For encoder the filter size increases and for the decoder the filter size decreases . (shown on the next page)

```
def build_dehaze_net(input_shape, output_channels):
    inputs = Input(shape=input_shape)

    # Encoder
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(inputs)
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(128, 3, activation='relu', padding='same')(pool1)
    conv2 = Conv2D(128, 3, activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(256, 3, activation='relu', padding='same')(pool2)
    conv3 = Conv2D(256, 3, activation='relu', padding='same')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    # Decoder
    conv4 = Conv2D(512, 3, activation='relu', padding='same')(pool3)
    conv4 = Conv2D(512, 3, activation='relu', padding='same')(conv4)
    up1 = UpSampling2D(size=(2, 2))(conv4)

    concat1 = Concatenate()([conv3, up1])
    conv5 = Conv2D(256, 3, activation='relu', padding='same')(concat1)
    conv5 = Conv2D(256, 3, activation='relu', padding='same')(conv5)
    up2 = UpSampling2D(size=(2, 2))(conv5)

    concat2 = Concatenate()([conv2, up2])
    conv6 = Conv2D(128, 3, activation='relu', padding='same')(concat2)
    conv6 = Conv2D(128, 3, activation='relu', padding='same')(conv6)
    up3 = UpSampling2D(size=(2, 2))(conv6)

    concat3 = Concatenate()([conv1, up3])
    conv7 = Conv2D(64, 3, activation='relu', padding='same')(concat3)
    conv7 = Conv2D(64, 3, activation='relu', padding='same')(conv7)

    # Output
    output = Conv2D(output_channels, 1, activation='sigmoid',
padding='same')(conv7)

    model = Model(inputs=inputs, outputs=output)
    return model
```

Output Comparison of the 3 Attempts:

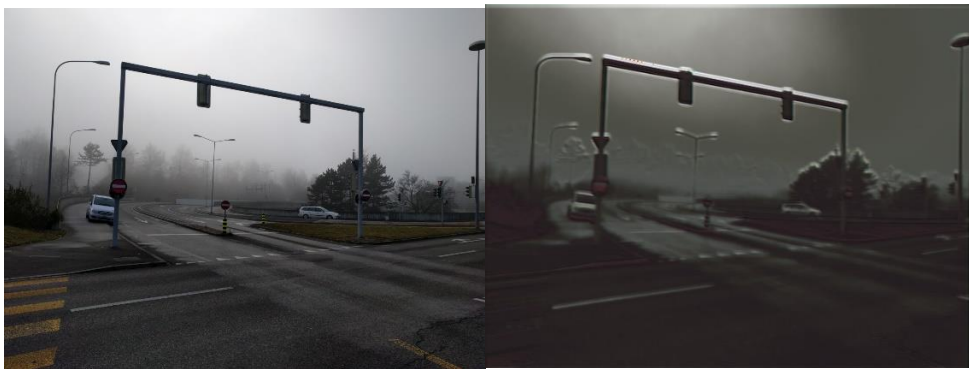
Format = Test Image on left and output image on right

Attempt 1:

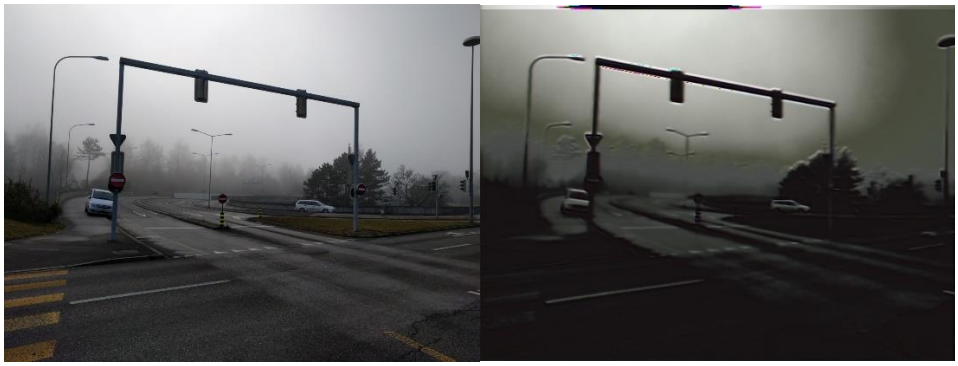


Attempt 2:

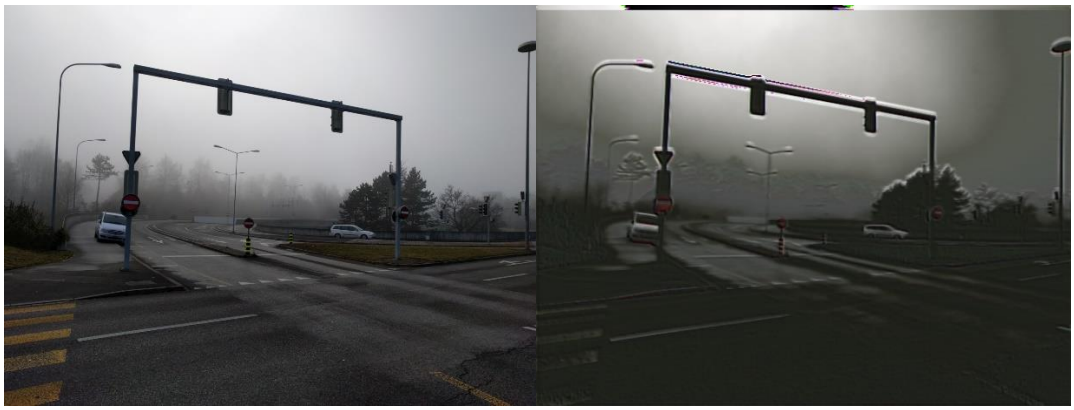
- Model K



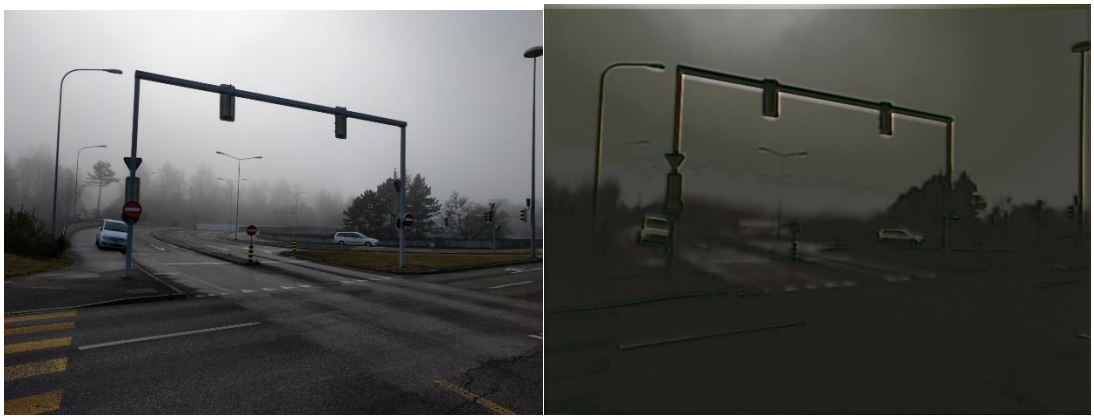
- Model L



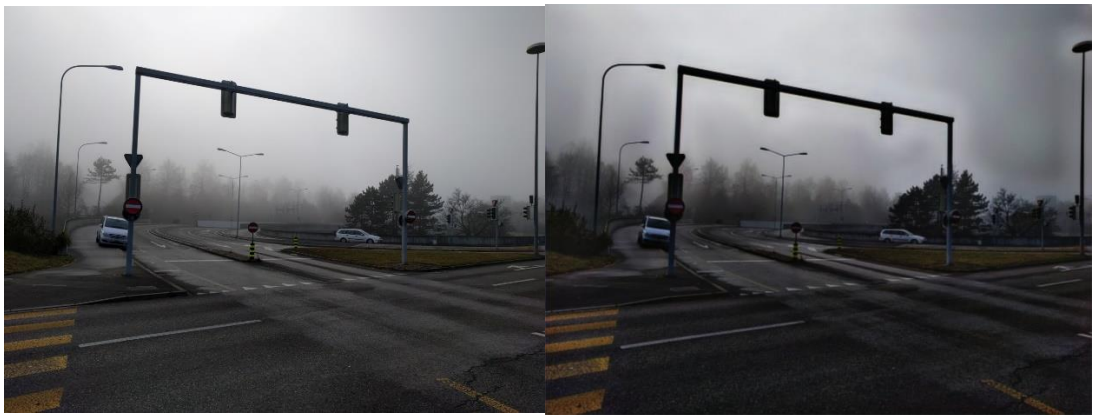
- Model M



- Model U



Attempt 3:



As you can clearly see, the final model had a vast increase in both picture quality, colour maintenance, and fog reduction.