ML Case Study Report

a-) In order to load the image data from the provided images, I first used the "imread" function from the "tifffile" library of Python. Then, I preprocessed the image data I read. While preprocessing, I reshaped the images for matching the pixel pairs. Moreover, I changed the format of the images from 2D to 1D. After that, I applied mapping from int16 to int8 values. For applying it, I defined a function called "map_int16_to_int8". In this case study, the raw RGB image is given as an image file which consists of int16 values ranging from 0 to 7000. Furthermore, the true color RGB image is given as an image file which consists of int8 values ranging from 0 to 255. Therefore, I rescaled the int16 values of the raw RGB image by dividing them with 7000 and multiplying them with 255. After that, I converted the rescaled values to the int8 format by using "astype(np.uint8)" function. Here, "np.uint8" represents the 8-bit unsigned integer coming from the NumPy library. Then, I split the given image data into training and test parts, namely X_train, X_test, Y_train, Y_test. During this splitting process; for each image, I assigned 25 percent of the pixels to the test data and the rest (75 percent) to the training data. For achieving consistent results for each splitting, I specified the random seed as 43 within the "train_test_split" function. After that, I applied the resampling technique called "bootstrap resampling". I used the number of bootstrap samples as 3. Then, I created 2 lists to store predictions from each bootstrap sample. One list was to store the predictions of the Linear Regression model and another list was to store the predictions of the Neural Network model. Then, by using a for loop, I iterated through the bootstrap samples. Inside the for loop, by utilizing the "random.choice" function from the numpy library, I generated random indices via replacement from the range of indices which correspond to the X training set. Then, by passing the created random indices to the X train and Y train data, I created a bootstrap sample of the input features and a bootstrap sample of the target variable. After that, I trained a LinearRegression model. While doing so, I have used a built-in LinearRegression model constructor from the "sklearn.linear_model" module. Then, by using fit function with the arguments of bootstrap samples fort he input features and output variable (X_bootstrap and Y_bootstrap), I fitted the linear model that I created. After that, I predicted the true values with the X_test data. For this purpose, I used the "predict" function on top of the generated linear regression model. After that, I appended the generated linear prediction to the list in which I keep all predictions for the linear regression model. Then, I created a neural network model by utilizing the Sequential class from the Keras API of Python. This enabled me to create a linear stack of layers. Then, I added a dense layer to the sequential neural network model that I created. By doing so, I used the Dense function from the Keras API which takes place under the "tensorflow" library of Python. This enabled me to create a fully connected layer where each neuron has a connection with every neuron in the previous layer. In the dense layer; I specified the number of neurons as 32, which is the first argument of the Dense function. Then, as the second parameter of the Dense function, I specified the activation function. I utilized ReLU (Rectified Linear Unit) as the activation function. Then, I specified the input shape parameter as (3,) to show that each input sample has three features. Subsequently, I added another dense layer to the neural network model where the number of neurons is 32 and the activation function is the Rectified Linear Unit (ReLU). After that, I added a dense layer with 3 neurons to the generated neural network model and specified this layer as linear. Then, I compiled the generated neural network model by using "compile" function on top of the NN model. While performing this compilation, I specified the loss function and optimizer. As the optimizer, I utilized the "ADAM" optimizer. In addition, as the loss function, I used "mean squared error (MSE)" in compilation. After that, I trained the generated neural network model by using the "fit" function with the NN model. During this process, I utilized the training data named "X_bootstrap" as the input and "y_bootstrap" as the target. Moreover, I specified the epoch parameter as **5** in the training process for NN model. Epoch is a measure which specifies the number of times the NN model will go through the entire training

dataset. Furthermore, in the training process for the NN model, I specified the batch size parameter as 32. In other words, I assigned the **"number of processed samples before the NN model's internal parameters such as weights and biases are changed"** to 32. After that, I made predictions of true values by utilizing the trained neural network model on the test data for input variables. Then, I appended the obtained prediction for NN to the list in which I keep all predictions for the neural network model.

After the for loop finishes, I converted the predictions coming from the linear regression and neural networks models to the NumPy arrays. For doing that, I used the "np.array" function from the NumPy library of Python. Then, both for the linear regression and NN model, I computed the mean prediction across all bootstrap samples. While doing so, I utilized the "np.mean" function from the NumPy library of Python. Next, as expected, I rounded the mean predictions to the nearest integers by using the "np.round" function from the NumPy library. While rounding, I specified the unsigned integer type as "uint8" with the usage of the "astype(np.uint8)" function on top of "np.round".

Finally, I evaluated the performances of the trained linear regression and NN models. For this purpose, I choose the "mean squared error" (MSE) "mean absolute error" (MAE), and "r-squared" (coefficient of determination) as the evaluation metrics for the model performance. I used the functions of each of these metrics from the "sklearn" library. For each of these metrics, I passed the y_test to the parameter which represents the ground truth target values (1st parameter), and mean predictions to the parameter which represents the predicted values for each model (2nd parameter). At the end of my code, I printed the values of these evaluation metrics. Here, MSE represents the mean squared error between the ground truth target values (y_test) and predicted mean values. MAE is the mean absolute error between the ground truth (y_true) and predicted values (y_pred). MAE measures the model performance by considering the absolute differences between its predictions and ground truth values. Unlike MAE, the MSE (Mean Squared Error) measures the model performance by considering the squared differences between its predictions and ground truth values. Moreover, I also used the r2 score (coefficient of determination) as the evaluation metric. I used this metric to measure how well the linear regression and NN models fit the given image data and it gives the proportion of the variance in the target variable which is predictable from the input train data via the trained ML models.

b-) In the end, for the linear regression model, I found the value of the mean squared error (MSE) as 0.852. This means that the squared difference between the predicted and ground truth values is slightly small for the linear regression model. It also indicates that the predictions of the linear regression model are generally close to the ground truth values, which likely brings low bias. Furthermore, I found the value of mean absolute error (MAE) as 82.815. In a given fixed range of pixel intensities from 0 to 255 for int8 values, this indicates that the predicted pixel intensities have a slightly high deviation/error from the ground truth and thus the performance of linear regression can be considered as subject to better optimality. Moreover, I found the value of the R^2 Score (R-Squared/Coefficient of Determination) as 0.9987. It means that the variance in the target is successfully explained and the trained linear regression model has a highly suitable/successful fit to the provided image data. Overall, while the high R-squared value and low MSE value I obtained increase the model performance, high MAE which I found decreases the performance of the linear regression model.

In this study, I did not use any regularization. Regularization can be used to prevent overfitting issues in the ML models. When regularization is used, then it enables us to constrain the complexity of the coefficients in the ML models and likely avoid overfitting when the ML model becomes more complex compared to the given image data. Regularization techniques do so by adding penalty terms to the loss function. For example; it can add penalty terms proportional to the absolute values of the

model coefficients (L1 Regularization (Lasso)) and/or it can add penalty terms proportional to the square of the model coefficients (L2 Regularization (Ridge)). Moreover, regularization can help better handle small or noisy input data, which might even decrease the overfitting more and bring better accuracy & lower error values in the model evaluation part.

c-) Among the linear regression model and neural network model, I faced a tradeoff based on complexity vs flexibility. Linear regression is an ML model which is simpler than the neural network. Linear regression assumes a linear relationship between the input features and the target variable. The linear regression model is appropriate when the relationship between inputs and output is linear or nearly linear. However, it might struggle to cover complex patterns and relationships that are non-linear or close to non-linear for the provided image data.

On the other hand, neural network models are more complex than linear regression and capable of capturing non-linear relationships. They have more flexibility to learn complicated patterns and relationships from the image data. However; in this case study, the time complexity of the neural network (NN) model was higher than the time complexity of the linear regression model. Moreover, the neural network model required more computational resources to be trained. Therefore, the neural networks might be more tend to overfit the given image data. As a result, in comparison to the linear regression models, they may need to be better regularized to effectively handle overfitting.

In terms of performance, the neural network model brought relatively higher MSE and MAE values in comparison to the linear regression model. This shows that the prediction error and deviation for the NN model were slightly higher than LR. This also indicates that neural networks are capable of understanding non-linear relationships and complex patterns in image data. However, this flexibility comes at the cost of an increased probability of overfitting and complexity. However, both the linear regression and neural network models have high R^2 (R-squared) values, which shows their strong fit to the given image data and strong ability to capture variance in the given image data.

NOTE-1: You can see the output of the testing process (i.e., resulting values of the evaluation metrics) for both the linear regression and neural network models within the **“pynb”** file called **“doktar_ml”**. In this file, you can also see how I trained the linear regression and neural network models.

NOTE-2: For observing the training process of the linear regression model, you can also see the **“lr_train.py”** file. In order to observe how I tested the linear regression model, you can see the **“lr_test.py”** file. For observing how I trained the neural network model, you can see the **“nn_train.py”** file. In order to observe how I tested the neural network model, you can see the **“nn_test.py”** file. You can also see how I loaded the trained linear regression and neural network models within **“lr_test.py”** and **“nn_test.py”** files respectively.