COP528 Applied Machine Learning Coursework

Task1: Machine Learning Pipeline

I. Introduction

For the Applied Machine Learning module, the assessment is composed of a coursework which is split into two tasks. The first task is represented by designing a pipeline, an evaluation strategy and a set of experiments to determine the best parameters and machine learning algorithm based on the results of empirical evaluation derived from a dataset that we had to choose from the UCI machine learning repository. The dataset I chose is related to an early stage diabetes risk prediction. I will analyze the dataset before performing any action, carefully read it and check for obvious errors that can be handled in a text editor. I strongly consider that understanding the dataset is a crucial step before making any decisions. Later on, I will design a pipeline that will apply three machine learning algorithms and compare the results in the following chapters. The result will be composed of the best F1-score, parameter set and the scores for each of the models and justify which algorithm is the best towards modelling my dataset.

II. Data and Preliminary Analysis

As specified in the Section I, Introduction, the dataset I selected for the first task is related to the early stage diabetes risk prediction. All the data has been collected from the patients of Sylhet Diabetes Hospital in Sylhet, Bangladesh, where the patients completed a questionnaire, which was approved by a doctor. The dataset contains the signs and symptoms of a patient which is or would be diabetic. In Table 1 we can see the characteristics of the dataset, with the number of instances, number of attributes and the associated tasks, which will be "Classification" in this case [1].

Number of instances	520
Number of attributes	17
Missing values	N/A
Associated tasks	Classification

Table 1 – Dataset characteristics

It is important to understand the nature of the attributes. Besides the class diagram, there is one numerical attribute, related to the ages of the patients, and the others are all categorical, binary attributes such as the patient's sex or symptoms that may lead to a result.

For a better understanding of the dataset, from a technical point of view, after storing the dataset into a data frame in Python, I displayed its information, and I found out that the data type of the "age" column is int, and all the other 16 columns are stored as objects. Before performing machine learning algorithms on this dataset, I also checked and found out that it is balanced and it has no missing values.

III. Methods

After understanding the dataset and see if cleaning is needed, I prepared it for applying machine learning algorithms to it. First thing I've done was storing all the data into one variable, X, after dropping the "class" column and split it into training and testing set. Therefore, I used the 80:20 method, to have 80% of the attributes for training and only 20% for testing. Now, I have 4 variables, X_train, X_test, y_train and y_test, where "y" is the label, in our case the class column with all its attributes. After testing the shapes of the train and test variables, out of the 520 instances, we have 416 instances ready for training, and 104 for testing.

Next step was creating and displaying the correlation matrix of the dataset, it can be seen in Fig. 1. The correlation matrix is designed to show the correlation between all features [2]. I decided to visualize the confusion matrix of the dataset to demonstrate the linear relationship between all the attributes.

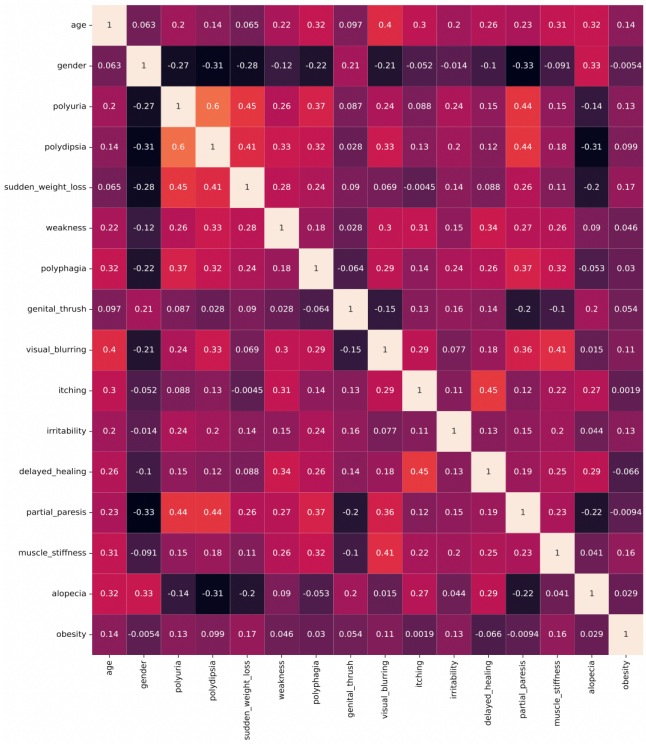

Figure 1 – Correlation Matrix

The result of the correlation matrix analysis shows that we do not have correlated features, as a result of not having any "-1" or "1" in Fig. 2.

After analyzing the dataset, I created a preprocessor in the pipeline, that transforms the numerical values using the "MinMaxScaler" function. The "MinMaxScaler" function transforms the features available into values to a given range. Therefore, all the individual values are now in the zero and one range. We can see the formula of transformation in (Formula1), where the min and max variables are 0 and 1. [3]

$$X_{std} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

$$X_{scaled} = X_{std} * (\max - \min) + min$$

Where: min, max = feature_range

Formula 1 – "MinMaxScaler" Function Formula

Our aim was to normalize the dataset before applying the machine learning algorithms. Data normalization is crucial for decision-making methods such as the machine learning algorithms, because it converts the values into a common scale which will help towards classification and a better accuracy of the models [4].

Moreover, the preprocessor uses the "OneHotEncoder" function for the categorical fields. This function converts the values into encoding scheme. It creates a binary column for each option [5]. In our case, we will encode the categorical columns, therefore the values will change from "yes/no" to "1/0".

After applying all these pre-processing methods to the dataset, it finally is ready to be used in classification algorithms.

IV. Experiments

For the experiments, I decided to use three machine learning algorithms that will analyze the input data to predict the likelihood of the data to fall into one of the classes.

The metrics I used for the experiments are support, precision, recall and F1-score, which is a good balance metric between the precision and recall. We will have the precision and recall values, but there is a trade-off between precision and recall. Given our situation, where the dataset's aim is to predict whether a patient has or does not have diabetes based on some symptoms, I strongly believe that the recall prevails to the precision, because we would like to have more false positives in trade off to have less false negatives. False negatives would have a bigger impact on a patient in our case. From a humanly point of view, it is much worse to have a false negative result related to a disease.

In the pipeline, I ran three experiments using three different machine learning models: Logistic Regression, Random Forest Classifier and Decision Tree Classifier.

Logistic regression is the algorithm used when the class variable is binary. Given the fact that regression analysis is used for predictions, logistic regression aims to describe data and explain the relationship between the dependent variable and the other independent variables. According to the data available, the logistic regression algorithm will outline the probability of having one of the characteristics outlined in the class variable. The major assumptions of binary logistic regression are that the dependent variable is binary, that the data has no outliers, and that there are no high correlations between the predictors [6].

In our case, we discussed that there are only two outputs (positive or negative), the dataset has no outliers, and according to the correlation analysis done in the previous section, there is no high correlations among the features predictors.

In the pipeline, I pursued hyperparameter tunning for the logistic regression algorithm. I set the "solver" parameter, which will be crucial towards optimization, as "liblinear" because I consider the dataset as small, and this is the right choice for small datasets, therefore the penalty options are either "l1" or "l2", and I set the random state as 42 in order to not generate a different set of tests every time I run the code. I also set the "C"-value as one of the following: 0.01, 0.1, 1, 10 and 100, which is the inverse strength and must always be a positive float value [7].

In order to avoid overfitting, we are applying K-Fold validation, by splitting into a 5-fold validation (k = 5), with 2 repeats. The last parameter I set to the models is the "GridSearchCV", which is a method that tries every combination of hyperparameter values and returns the optimal ones and the results achieved [13].

The results of the model were satisfying, having the best value of F1-score 0.924, achieved for this optimized set of parameters: C value of 10, and l1 for penalty.

Scores	Precision	Recall	F1-score	Support
Negative	0.90	0.85	0.88	33
Positive	0.93	0.96	0.94	71
Accuracy			0.92	104
Macro avg	0.92	0.90	0.91	104
Weighted avg	0.92	0.92	0.92	104

Table 2 – Scores of Logistic Regression

Out of table 2 we can see that the model achieved 92% accuracy, and it is more likely for the recall value to be higher than the precision value for positive classification, and a lower value for negative classification.

Random forest classifier is the machine learning algorithm that combines the result of multiple decision trees into a single result. Its popularity in classification problems is given by its ease of use. Given the nature of singular decision trees, being common supervised learning models that can result into overfitting, by combining them into the

random forest algorithm, the results will be more accurate [8].

For my model, I set the number of estimators, which represents the number of trees that are build before taking the averages of predictions, to 50,100 and 200, the max depth which is the maximum depth of the tree to 5, 10 and 20, the maximum leaf nodes to none, 5 and 10, while keeping the random state to 42.

The results of the model were better than the previous one applied, having the best F1-score 0.968, when the optimal parameters were 10 for max depth, no maximum leaf nodes and 100 estimators.

Scores	Precision	Recall	F1-score	Support
Negative	0.97	1.00	0.99	33
Positive	1.00	0.99	0.99	71
Accuracy			0.99	104
Macro avg	0.99	0.99	0.99	104
Weighted avg	0.99	0.99	0.99	104

Table 3 – Scores of Random Forest Classifier

Given the results in Table 3, we can see that this model achieved 99% accuracy, but the recall value is lower than the precision value for positive outcomes.

The last machine learning algorithm I applied is Decision Tree Classifier. This is a supervised machine learning algorithm that makes decision based on a set of rules, being very similar to the humanly decision making. This model has to decide based on the rules given as input which class to assign each instance of the dataset. The decision tree are popular when it comes to classification and regression tasks, and creates a number of "questions" having "yes" and "no" as answers that lead to choices [10].

I strongly believe that this is the most suitable algorithm for our dataset, where, except the age attribute, all the others are binary categorical. As parameters, I set the max depth to 5 and 10, and the maximum leaf nodes to 5, 10, 15 and 20.

The best F1-score of the model was 0.952, when the maximum depth was 5 and the number of maximum leaf nodes was 20.

Scores	Precision	Recall	F1-score	Support
Negative	0.89	1.00	0.94	33
Positive	1.00	0.94	0.97	71
Accuracy			0.96	104
Macro avg	0.95	0.97	0.96	104
Weighted avg	0.97	0.96	0.96	104

Table 4 – Results of the Decision Tree Classifier

The results in Table 4, outline that the Decision Tree Classifier model achieved a 96% accuracy, and there are values of 100% for positive precision and negative recall, which is not necessarily good in our case.

V. Reflection

In the requirements.txt file, there can be found all the information related to the virtual environment, using python 3.9.7, matplotlib 3.5.1, and pandas version 1.4.1.

After applying the three machine learning algorithms to the Diabetes predictor dataset, we can conclude that, in our case, the Random Forest Classifier performed the best in terms of accuracy, achieving 99% F1-score. On the other hand, as we discussed earlier, the precision and recall metrics also need to be taken into consideration when ranking algorithms. Given our situation, when the aim is to predict if a patient has or hasn't diabetes, I strongly believe that an algorithm with a higher recall value on positive would outsmart others, because, when it comes to medical datasets, everyone would choose false positive instead of false negative, the Logistic Regression method in our case.

Task 2: Deep Learning for image classification

A. Introduction

For the Applied Machine Learning module, the second task represents using a CNN network for classifying test images into the predefined class. We have been provided with an image dataset containing meaningful objects. I chose not to use a pre-defined model, but to create a new one and gain more understanding of how the convolutional neural network works, make some experiments with the parameters and improve the decision-making skills of setting the parameters.

B. Data and preliminary analysis

The provided dataset consists in two major folders of data, one for training and one for validation. Each folder is split into 10 sub-classes, containing several images that represent particular objects. I decided to store the dataset locally and not on cloud, because it was much easier for me to visualize the images.

Before starting the technical analysis, I created a "conda" environment where I'm aiming to perform a CNN model for deep learning in the tensorflow library in Python 3.6.13.

C. Methods

After understanding the dataset, I decided to create a function called "load data" that will load the dataset. I adapted the code

from [10]. I considered more effective to read the images from the directory, resize them based on input value and then rescale the images, because the quality was too high, and the time wouldn't have been efficient. Therefore, now there are two NumPy arrays: X_data and y_data that contain the dataset cleaned and shuffled. Moreover, the data type of the arrays is float and integer.

The next step was calling the "load data" function and storing the images into four variables: X_train, y_train, X_test and y_test. To make sure that the steps were good so far, I visualized the images in a plot, as seen in Figure 2.

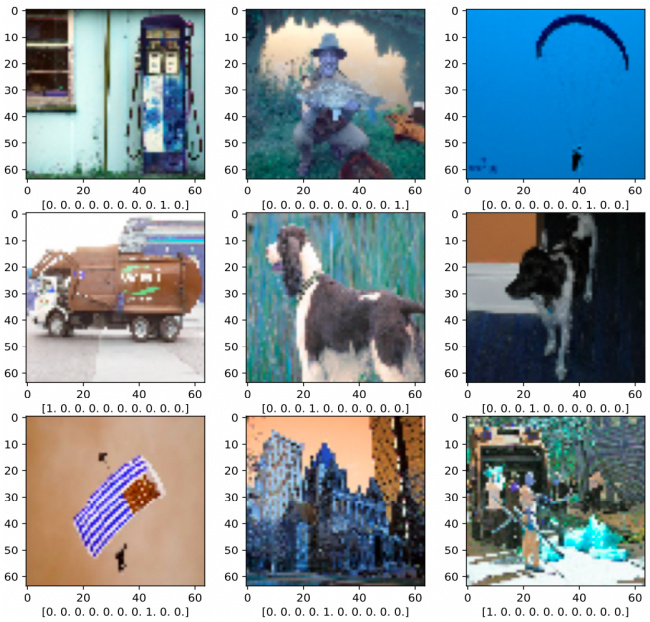

Figure 2 – Images from the training dataset

As we can see, the quality of the images is poorer than the original one because the resolution was changed to 64x64, for time-efficient training purposes.

Given the fact that there are 10 major classes with objects, I used the "one-hot encoding" method using numpy to change the value of y_train and y_test to categorical. The next step in the pre-processing of the data was rescaling the images.Therefore, given the fact that a pixel is in the range 1-255, I divided all the dataset by 255 just to have pixel values between 0 and 1, same like a normalized dataset.

D. Experiments

After having all the data stored in the X and y variables, I created a sequential CNN model and ran a set of experiments to see how the model is performing on this dataset. The differences in the experiments consist in the parameters applied for the training as well as the image quality. I started with the premises that a lower quality of the image will result in a faster training, therefore faster results. The model was created by adapting the code from the lectures and labs, [11], [12] and [13], where the importance of every parameter applied is explained and I was aiming to build a powerful image classification machine learning model, having little data.

Experiment 1

The first experiment consisted in creating a model with two convolutional layers, each one by adding a Conv2D and MaxPool2D. in the first convolutional layer, the Conv2D created a convolution kernel with 75 layers, having the same padding as the pictures, and having the input shape of 32,32,3, the details of the images, and the MaxPool2d having a 2x2 pool size. The next layer is the same, just having 125 layers in the Conv2D parameters. The next step was using flattening, because the cubic shapes can't be used as inputs, so the layers were flattened to create a single, very long vector of faces of the cube. (Model summary in Fig 3)

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 32, 32, 75)	2100
max_pooling2d_3 (MaxPooling2	(None, 16, 16, 75)	0
conv2d_4 (Conv2D)	(None, 16, 16, 125)	84500
max_pooling2d_4 (MaxPooling2	(None, 8, 8, 125)	0
dropout_4 (Dropout)	(None, 8, 8, 125)	0
flatten_2 (Flatten)	(None, 8000)	0
dense_3 (Dense)	(None, 500)	4000500
dropout_5 (Dropout)	(None, 500)	0
dense_4 (Dense)	(None, 250)	125250
dropout_6 (Dropout)	(None, 250)	0
dense_5 (Dense)	(None, 10)	2510

Total params: 4,214,860
Trainable params: 4,214,860
Non-trainable params: 0

Figure 3 – Experiment 1 model summary

The next step was applying the "dense" function of keras, which aimed to process the input of the previous layers. For avoiding overfitting, I chose to add a dropout of 0.4, and finally I compiled the model with the "adam" optimizer and having the accuracy as metrics. The batch size for fitting the model was 128, and I ran it in 10 epochs. The results (Table 5) were better than expected, but I decided to aim for a better accuracy of training.

Loss	Accuracy	Val Loss	Val Accuracy
0.6979	0.7685	1.1032	0.6573

Table 5 – Experiment 1 results

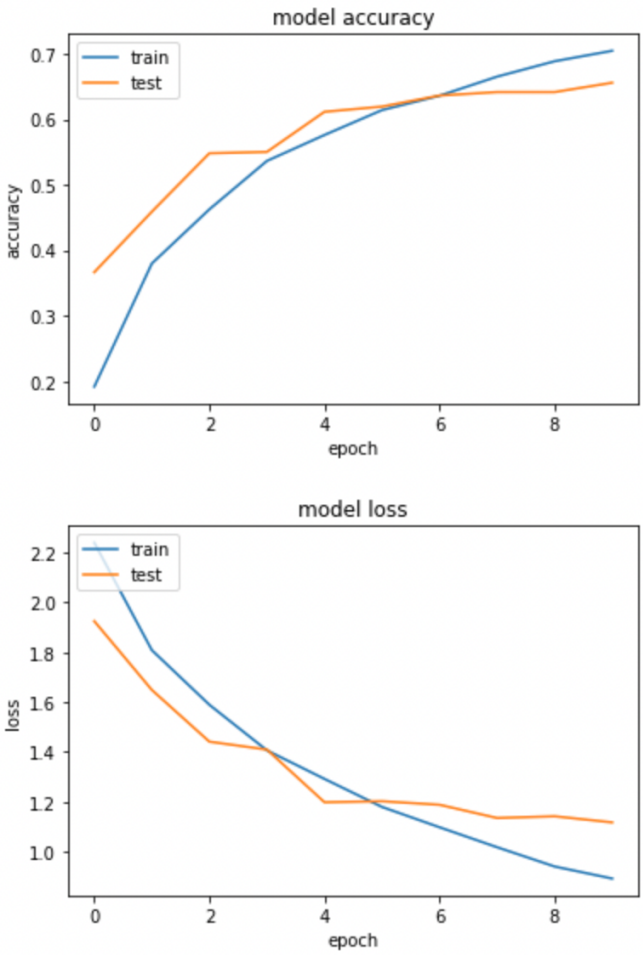

Figure 4 – Experiment 1 model accuracy and loss

Experiment 2

For the second experiment, after visualizing the images with the (32, 32, 3) shape, I decided to increase their resolution, so the new model has 64x64 as the resolution of the images. I will apply Conv2D with 64 layers twice and maxPool2D with 2x2 pool size. Also, when creating the model, I set the batch size to 128 and split the training data into 0.8 for training and 0.2 for validation, within 10 epochs. A summary of the model can be seen in Figure 5, having approximately 2 million trainable parameters.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 64, 64, 64)	1792
max_pooling2d_1 (MaxPooling2	(None, 32, 32, 64)	0
dropout_1 (Dropout)	(None, 32, 32, 64)	0
conv2d_2 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 16, 16, 64)	0
dropout_2 (Dropout)	(None, 16, 16, 64)	0
flatten_1 (Flatten)	(None, 16384)	0
dense_1 (Dense)	(None, 128)	2097280
dropout_3 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290

Total params: 2,137,290
Trainable params: 2,137,290
Non-trainable params: 0

Figure 5 – Experiment 2 model summary

The results of the tests, after fitting a model with 10 epochs can be found in Table 6 alongside with the model accuracy and model loss in Figure 6.

Loss	Accuracy	Val Loss	Val Accuracy
0.8923	0.7046	1.1177	0.6558

Table 6 – Experiment 2 results

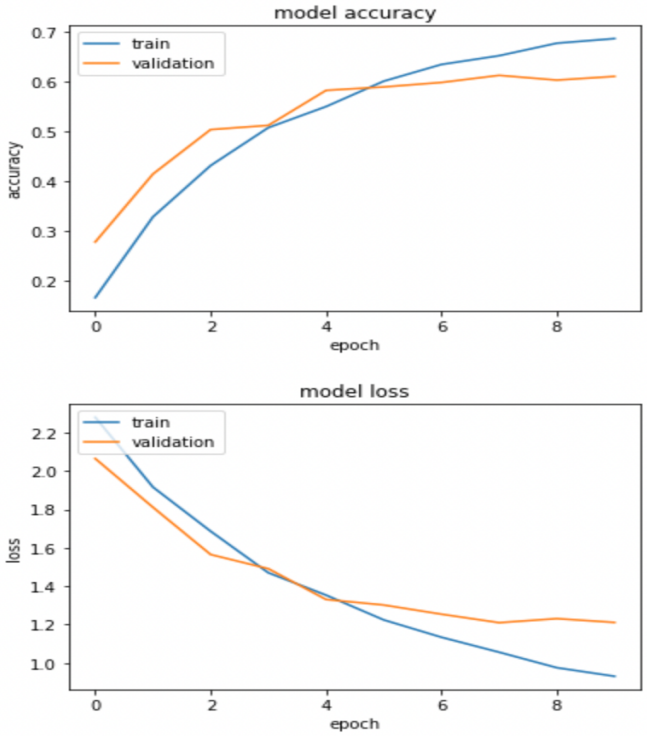

Figure 6 - Experiment 2 model accuracy and model loss

The results are quite satisfactory, without the issue of overfitting.

```
Evaluate model on test data
3925/3925 [==============================] - 36s 9ms/step
test loss, test acc: [1.152766961109866, 0.6321018934249878]
```

Figure 7 – Experiment 2 test data evaluation

After evaluating the model on the test data, the results can be seen in Figure 7, achieving 1.1528 test loss and 0.6321 test accuracy for the second set of data we have, after training and evaluating on the first one.

E. Reflections

It is important to know that this study was conducted in a conda environment, with matpotlib 3.3.4, python 3.6.13, scikit-learn 0.24.2 and tensorflow 1.15.0.

To sum up, out of these two experiments I understood how a CNN network architecture is used for classifying test images into the predefined classes. Given the results extracted from the studies, the first experiment conducted on the dataset is very similar to the second one when it comes to accuracy/loss learning curves for both training and validation data, even though we could not find good fit learning curves (Fig 4 and Fig 6)

In the model loss plot, the model from the second experiment has a lower distance between the training and validation lines than the first model, but unfortunately the training curve did not reach the point of stability. Also, overfitting is not an issue because even though the training loss keeps going down, the validation loss does not start going up again.

In order to improve the results in the future, I would have ran the model with more epochs and add more data to the training, aiming for a higher accuracy of the model, but in terms of compiling the model, I am delighted with the "adam" optimizer.

REFERENCES

[1] https://archive.ics.uci.edu/ml/datasets/Early+stage+diabetes+risk+prediction+dataset.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] https://medium.com/analytics-vidhya/correlation-matrix-5e764bcee34K. Elissa, "Title of paper if known," unpublished.

[3] scikit-learn. 2022. *sklearn.preprocessing.MinMaxScaler*. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html> [Accessed 18 March 2022].

[4] International Journal of Information and Decision Sciences. 2022. Data normalisation techniques in decision making: case study with TOPSIS method | International Journal of Information and Decision Sciences. [online] Available at: <https://www.inderscienceonline.com/doi/10.1504/IJIDS.2018.090667> [Accessed 18 March 2022].

[5] scikit-learn. 2022. *sklearn.preprocessing.OneHotEncoder*. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html> [Accessed 18 March 2022].

[6] Statistics Solutions. 2022. *What is Logistic Regression? - Statistics Solutions*. [online] Available at: <https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/what-is-logistic-regression/> [Accessed 18 March 2022].

[7] scikit-learn. 2022. *sklearn.linear_model.LogisticRegression*. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html> [Accessed 18 March 2022].

[8] IBM, 2022. *What is Random Forest?*. [online] Ibm.com. Available at: <https://www.ibm.com/cloud/learn/random-forest> [Accessed 18 March 2022].

[9] Towards Data Science. 2022. *Decision Tree Classifier explained in real-life: picking a vacation destination*. [online] Available at: <https://towardsdatascience.com/decision-tree-classifier-explained-in-real-life-picking-a-vacation-destination-6226b2b60575?source=----------0----------> [Accessed 18 March 2022].

[10] Stack Overflow. 2022. *How to Load custom image and make them a dataframe ? Directory name invalid error*. [online] Available at: <https://stackoverflow.com/questions/67852083/how-to-load-custom-image-and-make-them-a-dataframe-directory-name-invalid-erro> [Accessed 18 March 2022].

[11] TensorFlow. 2022. *Deep Convolutional Generative Adversarial Network | TensorFlow Core*. [online] Available at: <https://www.tensorflow.org/tutorials/generative/dcgan> [Accessed 18 March 2022].

[12] Analytics India Magazine. 2022. *Hands-On Guide To Implement Batch Normalization in Deep Learning Models*. [online] Available at: <https://analyticsindiamag.com/hands-on-guide-to-implement-batch-normalization-in-deep-learning-models/> [Accessed 18 March 2022].

[13] Blog.keras.io. 2022. *Building powerful image classification models using very little data*. [online] Available at: <https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html> [Accessed 18 March 2022].

[14] LaptrinhX. 2022. *Everything You Should Know About Dropouts And BatchNormalization In CNN*. [online] Available at: <https://laptrinhx.com/everything-you-should-know-about-dropouts-and-batchnormalization-in-cnn-3578533565/> [Accessed 18 March 2022].

[15] Medium. 2022. *A Comparison of Grid Search and Randomized Search Using Scikit Learn*. [online] Available at: <https://medium.com/@peterworcester_29377/a-comparison-of-grid-search-and-randomized-search-using-scikit-learn-29823179bc85> [Accessed 18 March 2022].