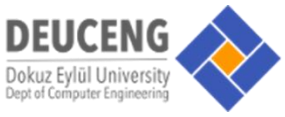


DOKUZ EYLUL UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CME4403 – Introduction to Machine Learning

Project Report

By

İrfan Erdem Şenyener

Derviş Çömlekci

Lecturer

Assoc. Prof. Dr. Zerrin Işık

December, 2022

İZMİR

TABLE OF CONTENTS

INTRODUCTION

For our term projects, we decided to use an image dataset. In this paper we will explain how we have accomplished our tasks, and how we have failed in a few instances.

Said dataset contains 90.000+ images of fruits and vegetables, split into %80 training and %20 testing data. These fruits and vegetables are labeled by their file name.

We have used three image classification models; CNN (convolutional neural network), kNN (k-th nearest neighbour) and naive-bayes.

DATASET PRE-PROCESSING AND FEATURE EXPLORATION

Dataset Properties:

- Total number of images : 90483
- Training set size : 67692 (%80)
- Test set size : 22688 (%20)
- Number of Classes : 131
- Image size : 100x100

Because our data was standardized, we didn't have to do cleaning and normalization on our data. We read the images from the directory, converted them to matrices that hold their color values, ranging from 0 to 255, (later re-scaled to be between 0 and 1), and labeled them according to their file names.

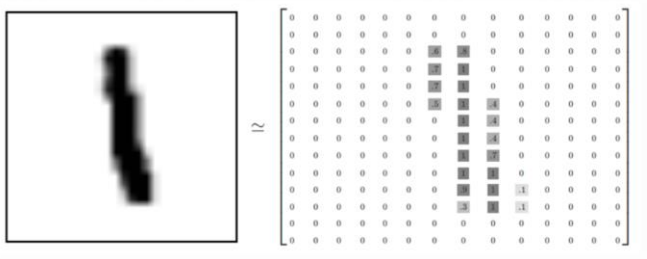


Figure 1 **An image as a matrice**

For distribution, we used all of the dataset for CNN(%80 training which its %10 is for validation) . But we had to cut the dataset for kNN and naive-bayes. Because the workload these methods put the memory and the processor makes it hard to to use these models. We used training data set and split it to train and test datasets.

As represented in Figure 1, our important features are colors of the image that represent the fruit or the vegetable.

IMPLEMENTATION DETAILS OF MACHINE LEARNING MODELS

In our projects, we have used three image classification models. These are in order: CNN, kNN and Naive-Bayes models.

1- CNN

"CNNs are a class of Deep Neural Networks that can recognize and classify particular features from images and are widely used for analyzing visual images. Their applications range from image and video recognition, image classification, medical image analysis, computer vision and natural language processing.

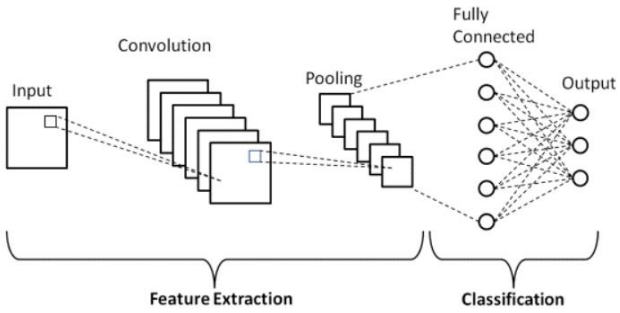


Figure 2 **Layers of CNN**

The term 'Convolution" in CNN denotes the mathematical function of convolution which is a special kind of linear operation wherein two functions are multiplied to produce a third function which expresses how the shape of one function is modified by the other. In simple terms, two images which can be represented as matrices are multiplied to give an output that is used to extract features from the image."

In our implementation we used the Python library TensorFlow.

```
data_train = tf.keras.utils.image_dataset_from_directory(
    'train',
    image_size=(100, 100),
    validation_split=0.2,
    subset="training",
    seed=13)
```

Figure 3 **TensorFlow database creation**

As can be seen in Figure 3, TensorFlow has built-in image database creation tools. We pick a validation split and select the directory of train and test data to create a database. Then we normalize our data to be in between 0 and 1.

```
# Rescaling our data from 0-255 to 0-1
normalization_layer = layers.Rescaling(1./255)
```

Figure 4 **Rescaling**

Then we load in our cnn model, Sequantial, and add layers to our network.

```
model = Sequential([
  layers.Rescaling(1./255, input_shape=(100, 100, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(256, activation='relu'),
  layers.Dense(num_classes)
])
```

Figure 5 **Network layers**

Then we compile and train our model using built-in methods, using optimizer adam.

```
# Compiling our model, using adam optimizer
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Number of epochs to retrain our model
epochs = 3

# Training our model
history = model.fit(
  data_train,
  validation_data=data_val,
  epochs=epochs
)
```

Figure 6 **Trainin of the model**

Now we have a trained model. Only thing left is to cross-validate it using the test data.

```
# Cross Validating our data
print("Testing")
testing = model.evaluate(data_test)
print("test loss, test acc: ", testing)
```

Figure 7 **Testing the model**

2- KNN

"K-Nearest Neighbours (k-NN) is a supervised machine learning algorithm i.e. it learns from a labelled training set by taking in the training data X along with it's labels y and learns to map the input X to it's desired output y.

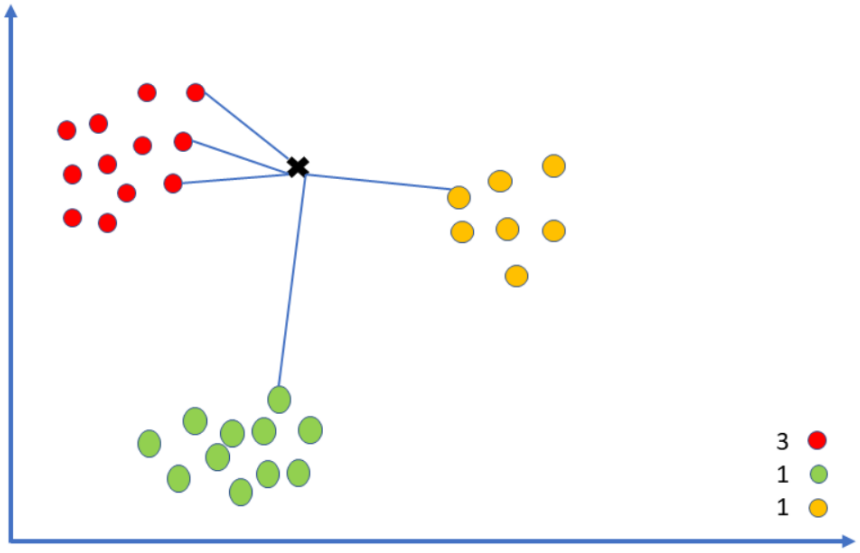


Figure 8 **Knn model**

The k-NN algorithm is arguably the simplest of the machine learning algorithms. The model only consists of the training data, that is, the model simply learns the entire training set and for prediction gives the output as the class with the majority in the 'k' nearest neighbours calculated according to some distance metric."

We used sklearn for both naive-bayes and knn. But first we needed to import our dataset. We used OpenCV to read images and label them.

```
def createData(imagePaths):
    # Variables to hold our data and our labels
    data = []
    labels = []
    interval = 100
    # Main loop to loop over every image in our input file
    for (i, imagePath) in enumerate(imagePaths):
        image = cv2.imread(imagePath)
        # file/image -> file is the label of the image
        label = imagePath.split(os.path.sep)[-2]
        data.append(image)
        labels.append(label)

        # A print function to show the progress of creating data
        if interval > 0 and i > 0 and (i + 1) % interval == 0:
            print("{}/{} processed".format(i + 1, len(imagePaths)))

    return np.array(data), np.array(labels)
```

Figure 9 **Reading images and labeling them, from a directory.**

Then we reshape our data into a 2 dimensional tuple. From (x, 100, 100, 3) to (x, 30000)

```
# Reshaping our data from (x, 100, 100, 3) to (x,30000) [100x100 pixels and 3 is for rgb]
data_train = data_train.reshape((data_train.shape[0], 100 * 100 * 3))
```

Figure 10 **Reshaping of data.**

And we are ready to load in our model and train it.

```
# Loading our model
model = KNeighborsClassifier(n_neighbors=neighbour, n_jobs=4)

# Splitting our data into subsets, %80 for training and %20 for testing
trainX, testX, trainY, testY = train_test_split(data_train, labels, test_size=0.4, random_state=42)

# Training our model
model.fit(trainX, trainY)
```

Figure 11 **Loading and training of the model**

Only thing left is to cross-validate our data with the test data.

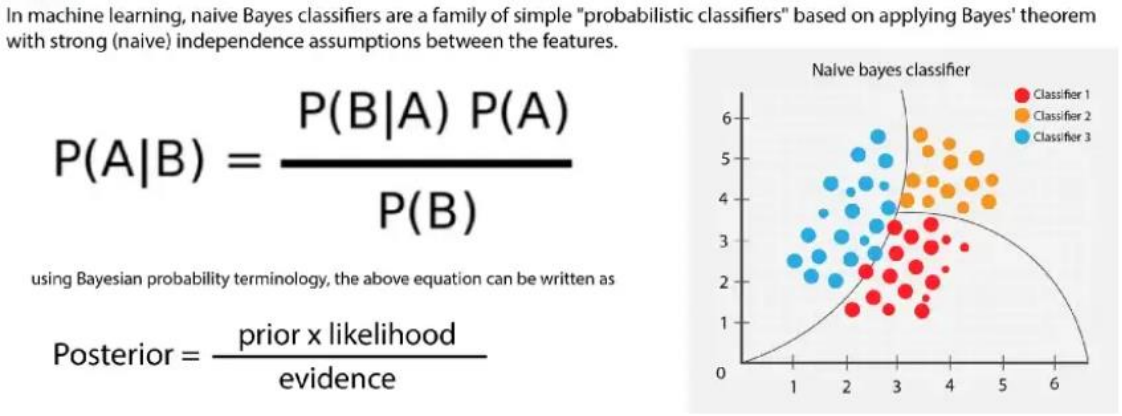
```
# Testing our model
pred = model.predict(testX)

# Printing a classification report to get accuracy, precision, recall and f1 scores.
print(classification_report(testY, pred))
```

Figure 12 **Cross-Validating the model**

3- Naive Bayes

"Naïve Bayes is the simplest classifier, which used the language of graphical models. This method assumes that each category has its own distribution over the codebook, and the distribution of each category is observably different from those of others. For instance, a building category might emphasize codewords that represent window, door, or floor, while a face category might show a significant representation of codewords for eye, nose, and mouth."



Figurec 13 **Naive-Bayes classifier**

The only differance ,in implementation, from knn is the model itself. Data set gathering and normalization is the same as knn.

```
# Loading our model -> Naive bayes using GaussianNB
tester = GaussianNB()

# Training our model
print(tester.fit(trainX, trainY))

# Testing our model
print("Please wait")
pred = tester.predict(testX)

# Printing a classification report to get the information.
report = classification_report(testY, pred)
print(report)
```

Figure 14 **Loading, Testing and Training of the model**

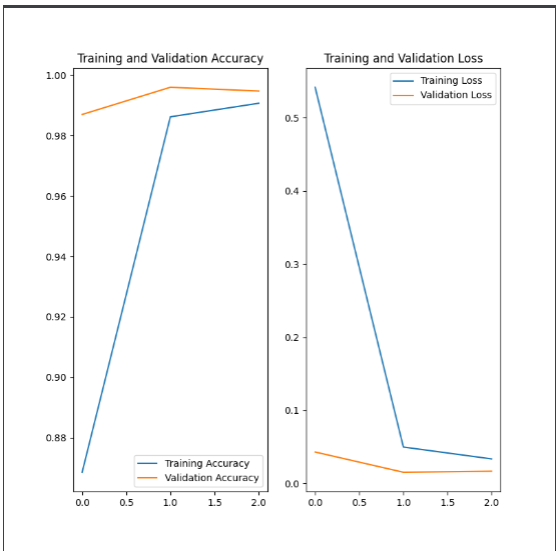
EXPERIMENTAL RESULTS

1- CNN

For CNN, we have 3 different variables to change the outcome of the cross-validation. Epoch count, %Validation and %Dropout. If we take 1.1 as our referance point, we can conclude that ;

- Increasing Epoch increases Accuracy and reduces Test Loss,
- Increasing %Validation only reduces Test Loss
- Increasing Dropout reduces Accuracy and increases Test Loss

1.1- Epoch: 3 | Validation: .2 | Dropout: .1



Test Loss: 0.3624

Accuracy: 0.9378

Figure 15 **CNN plot 1**

1.2- Epoch: 5 | Validation: .2 | Dropout: .1

Test Loss: 0.1818

Accuracy: 0.9653

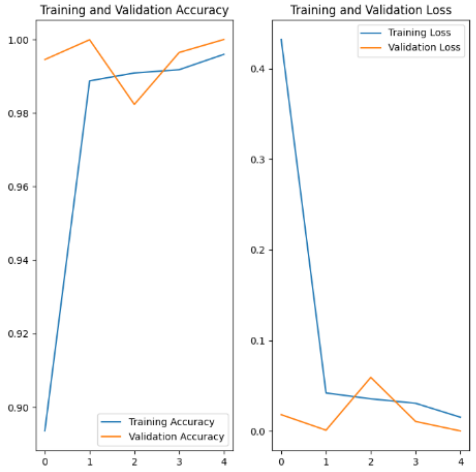


Figure 16 **CNN plot 2**

1.3- Epoch: 3 | Validation: .4 | Dropout: .1

Test Loss: 0.3069

Accuracy: 0.9318

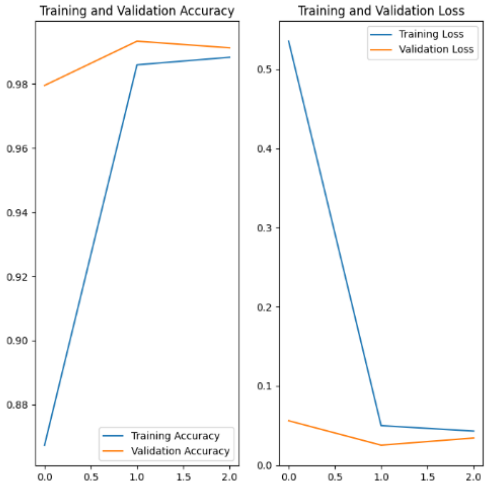
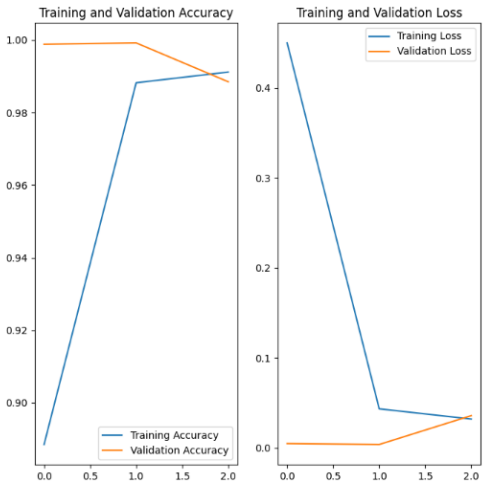


Figure 17 **CNN plot 3**

1.4- Epoch: 3 | Validation: .2 | Dropout: .2



Test Loss: 0.4230

Accuracy: 0.9098

Figure 18 **CNN plot 4**

2- KNN

In kNN, no mather what variable we changed, result was always

- Recall : 1.00
- Precision : 1.00
- Accuracy : 1.00
- F1 Score : 1.00

2.1 K : 3 Train .8

accuracy			1.00
macro avg	1.00	1.00	1.00
weighted avg	1.00	1.00	1.00

Figure 19 **kNN 1**

2.2 K : 6 Train .8

accuracy			1.00
macro avg	1.00	1.00	1.00
weighted avg	1.00	1.00	1.00

Figure 20 **kNN 2**

2.3 K : 3 Train .6

accuracy			1.00
macro avg	1.00	1.00	1.00
weighted avg	1.00	1.00	1.00

Figure 21 **kNN 3**

3- Naive-Bayes

In Naive-Bayes, the only changable variable is %Test data.

3.1 %20 Test Data

Accuracy:

Precision | Recall | F1-Score

accuracy			0.87	13539
macro avg	0.88	0.86	0.86	13539
weighted avg	0.88	0.87	0.87	13539

Figure 22 **NB 1**

3.2 %40 Test Data

Accuracy:

Precision | Recall | F1-Score

accuracy			0.84	27077
macro avg	0.87	0.84	0.84	27077
weighted avg	0.87	0.84	0.85	27077

Figure 23 **NB 2**

CONCLUSION

In this projects, we have learned how to represent images in a way that algorithms can interact and learn from it, how to make sense of an image in computer language and what features are important for it, how deep learning algorithms can use these features to learn and understand and classify them, how a simple algorithm like kNN can be better than an advanced method like CNN when the data set is perfect for it.

Our understanding of machine learning has increased and we have learned how to alternate data to get different results for our goals.

We learned how to use Python libraries efficiently and how to understand what they do and produce.

In research to pick models for our Project, we came into contact advanced algorithms that we previously considered too advanced for us, and found out that they are complex but understandable subjects.

REFERENCES

https://www.kaggle.com/datasets/ishandandekar/fruitimagedataset

https://github.com/shilparai/image_classification_knn/blob/master/knn_model.py

https://betterprogramming.pub/na%C3%AFve-bayes-vs-svm-for-image-classification-75c16b29a96d

https://www.tensorflow.org/tutorials/images/classification

https://www.upgrad.com/blog/basic-cnn-architecture/

https://www.guru99.com/convnet-tensorflow-image-classification.html

https://medium.com/swlh/image-classification-with-k-nearest-neighbours-51b3a289280

https://betterprogramming.pub/na%C3%AFve-bayes-vs-svm-for-image-classification-75c16b29a96d