Automated Store Billing System Based on Deep Learning (Image Detection and Computer Vision)

MSc Research Project
MSc in Data Analytics

Vijaykumar Ghanti
Student ID: x21237174

School of Computing
National College of Ireland

Supervisor: Prof. Furqan Rustam

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Vijaykumar Ghanti
Student ID:	x21237174
Programme:	MSc in Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Prof. Furqan Rustam
Submission Due Date:	18/09/2023
Project Title:	Automated Store Billing System Based on Deep Learning (Image Detection and Computer Vision)
Word Count:	XXX
Page Count:	32

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Vijaykumar Ghanti
Date:	18th September 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Automated Store Billing System Based on Deep Learning (Image Detection and Computer Vision)

Vijaykumar Ghanti
x21237174

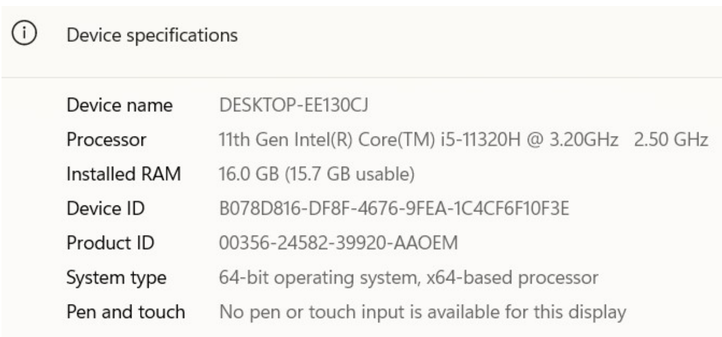
1 Inroduction

The aim of the document is to provide a stepwise approach to achieve the "Automated Store Billing System Based on Deep Learning (Image Detection and Computer Vision)". For customer satisfaction and retail profit efficient store billing systems are crucial. However errors while pricing and inventory issues cost the the industry billions. National retail federation said, due to errors in pricing retail got shrink and inventory management costed to $61.7 billion in 2020 in the United States alone Federation (2020). This research aims to create an Automated Store Billing System using deep learning, image processing, and computer vision. Traditional manual processes are error-prone and time-consuming, especially for products without barcodes. In this field, computer vision has shown promise in addressing these limitations. Deep learning, a subset of AI, is a powerful technology for image processing and computer vision tasks. Our research employs both machine learning and deep learning models to analyze product images and select the perfect algorithm which is performing best. Then provide a streamlined billing process for retail customers and improve profitability.

2 Hardware and software requirements

2.1 Hardware

The Laptop with 11th Gen Intel® core™ with installed RAM of 16.0 GB and 64 bit operating system, x64-based processor is used to build the model as shown in Fig. 1. And a camera or mobile camera of 108MP with EIS, 2MP Depth-Assist lens and 2MP Macro Lens.


Device specifications

Device name	DESKTOP-EE130CJ
Processor	11th Gen Intel(R) Core(TM) i5-11320H @ 3.20GHz 2.50 GHz
Installed RAM	16.0 GB (15.7 GB usable)
Device ID	B078D816-DF8F-4676-9FEA-1C4CF6F10F3E
Product ID	00356-24582-39920-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Figure 1: Laptop used with specifications

2.2 Software

Software like Jupyter Notebook to run python code, python to build model, SQL to create database and fetch respective data, and PostGreSQL to store data are given with their respective versions as shown in Table 1 are used.

Table 1: Software Versions

Software	**Version**
Jupyter Notebook	6.4.12
Python	3.10.7
PGAdmin or PostgreSQL	7.6

3 Evaluation

We have considered both deep learning and machine learning algorithms like CNN, RCNN, ResNet, AlexNet and Naive Bayes, KNN, SVM, logistic regression to detect the fruits accurately by processing their images. Let's analyze the performance of each model one by one.

3.1 Deep learning

3.2 CNN

From Fig. 2 to 4 incorporates the code written to build the CNN model using TensorFlow and Keras to classify images of fruits into different categories. Necessary libraries like OS for file operations, NumPy for numerical operations, Matplotlib for plotting, and scikit-learn for machine learning utilities are imported. Some other libraries like Conv2D, MaxPooling2D, Flatten, and Dense are imported for defining the architecture of the neural network. Input path to the training data is set, number of classes and input shape of the CNN model are specified. For data augmentation ImageDataGenerator is written. Followed by two data generators are provided one for training and one for validation. build_cnn_model() method is defined to build the model. The model consists of multiple Convolutional and MaxPooling layers followed by Flatten, Dense, and Dropout layers to prevent overfitting. By given 10 epochs model is trained and built. The trained model history is stored in the history variable. Then confusion matrix, classification report, per epoch accuracy, 10-fold validation result are generated as shown in following images.

```
In [1]: # CNN
```

```
In [1]: import os
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split, cross_val_score, KFold
        from sklearn.metrics import classification_report, confusion_matrix
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
        from tensorflow.keras.preprocessing.image import ImageDataGenerator

        input_path = r'C:\Users\DELL\Documents\NCI\Semister_3\Thesis\dataset\fruits-360_dataset\fruits-360\Train'
        num_classes = 5
        input_shape = (150, 150, 3)  # Image size for CNN

        datagen = ImageDataGenerator(
            rescale=1.0/255.0,
            validation_split=0.2,
            rotation_range=20,
            width_shift_range=0.2,
            height_shift_range=0.2,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True,
            fill_mode='nearest'
        )

        train_generator = datagen.flow_from_directory(
            directory=input_path,
            target_size=input_shape[:2],
            batch_size=32,
            class_mode='categorical',
            subset='training'
        )

        validation_generator = datagen.flow_from_directory(
            directory=input_path,
            target_size=input_shape[:2],
            batch_size=32,
            class_mode='categorical',
            subset='validation'
        )
```

Figure 2:

```
def build_cnn_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))  # Add dropout to prevent overfitting
    model.add(Dense(num_classes, activation='softmax'))
    return model

model = build_cnn_model()

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size,
    epochs=10
)
```

```
Found 1881 images belonging to 5 classes.
Found 469 images belonging to 5 classes.
Epoch 1/10
58/58 [==============================] - 74s 1s/step - loss: 0.5191 - accuracy: 0.7896 - val_loss: 0.0089 - val_accuracy: 1.000
0
Epoch 2/10
58/58 [==============================] - 58s 990ms/step - loss: 0.0266 - accuracy: 0.9908 - val_loss: 0.0018 - val_accuracy: 1.
0000
Epoch 3/10
58/58 [==============================] - 58s 1s/step - loss: 0.0170 - accuracy: 0.9935 - val_loss: 3.2716e-04 - val_accuracy:
1.0000
Epoch 4/10
58/58 [==============================] - 58s 1s/step - loss: 0.0018 - accuracy: 0.9995 - val_loss: 3.8752e-05 - val_accuracy:
1.0000
Epoch 5/10
58/58 [==============================] - 59s 1s/step - loss: 0.0263 - accuracy: 0.9930 - val_loss: 4.4107e-04 - val_accuracy:
1.0000
Epoch 6/10
```

Figure 3:

```
58/58 [==============================] - 58s 1s/step - loss: 0.0018 - accuracy: 0.9995 - val_loss: 3.8752e-05 - val_accuracy:
1.0000
Epoch 5/10
58/58 [==============================] - 59s 1s/step - loss: 0.0263 - accuracy: 0.9930 - val_loss: 4.4107e-04 - val_accuracy:
1.0000
Epoch 6/10
58/58 [==============================] - 59s 1s/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 7.2328e-05 - val_accuracy:
1.0000
Epoch 7/10
58/58 [==============================] - 58s 1s/step - loss: 0.0051 - accuracy: 0.9968 - val_loss: 3.5500e-05 - val_accuracy:
1.0000
Epoch 8/10
58/58 [==============================] - 58s 995ms/step - loss: 0.0046 - accuracy: 0.9973 - val_loss: 1.7047e-05 - val_accurac
y: 1.0000
Epoch 9/10
58/58 [==============================] - 60s 1s/step - loss: 2.6928e-04 - accuracy: 1.0000 - val_loss: 3.2429e-06 - val_accurac
y: 1.0000
Epoch 10/10
58/58 [==============================] - 60s 1s/step - loss: 5.9983e-05 - accuracy: 1.0000 - val_loss: 1.3807e-06 - val_accurac
y: 1.0000
```

Figure 4:

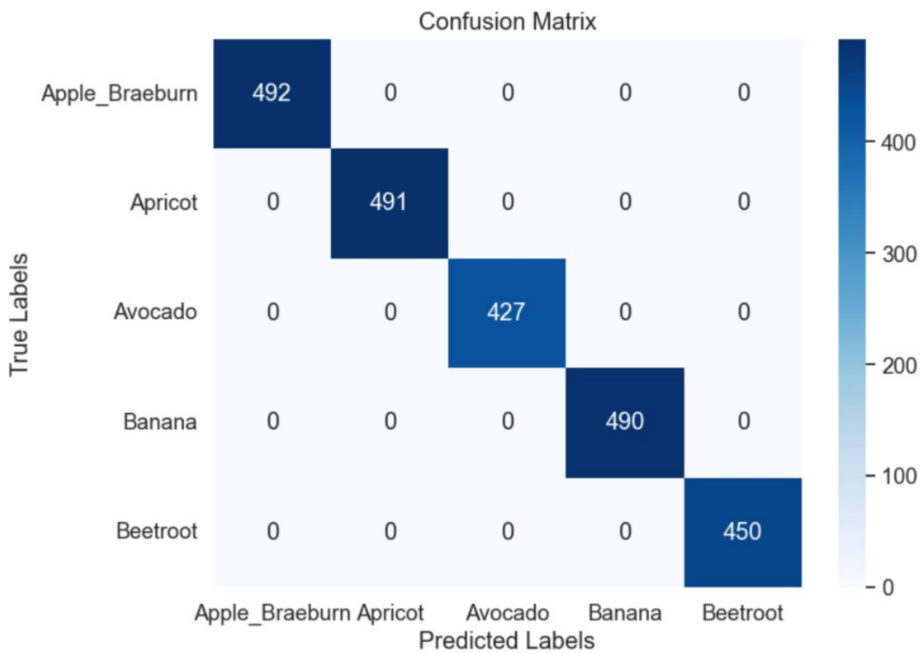


Figure 5: CNN confusion matrix

```
Classification Report:
                precision    recall  f1-score   support

Apple_Braeburn       1.00      1.00      1.00       492
       Apricot       1.00      1.00      1.00       491
       Avocado       1.00      1.00      1.00       427
        Banana       1.00      1.00      1.00       490
      Beetroot       1.00      1.00      1.00       450

      accuracy                           1.00      2350
     macro avg       1.00      1.00      1.00      2350
  weighted avg       1.00      1.00      1.00      2350
```

Figure 6: CNN classification report

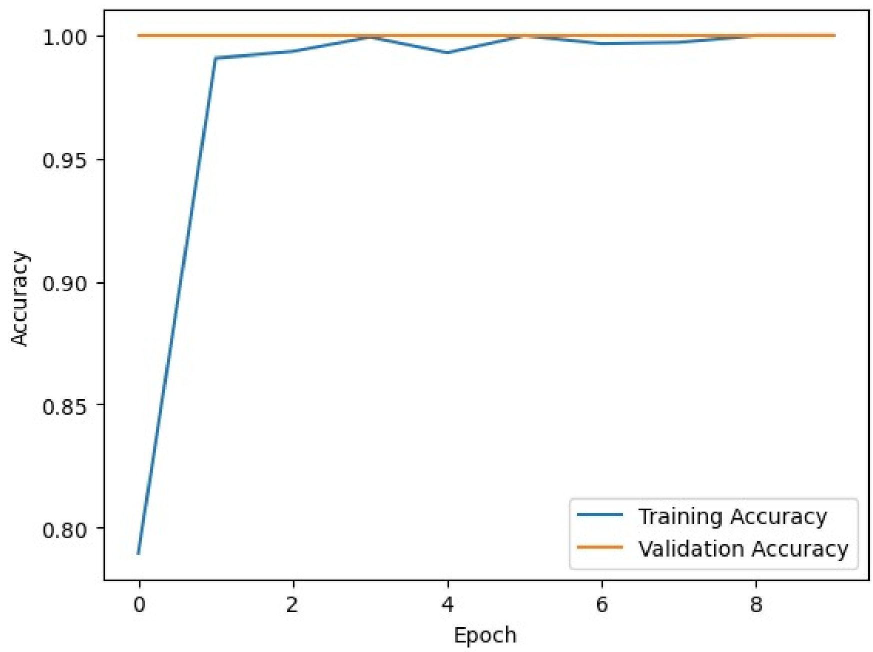


Figure 7: per epoch accuracy

10-Fold Cross Validation Results:

Fold	Accuracy
1	1.000000
2	0.500000
3	0.666667
4	0.666667
5	0.333333
6	1.000000
7	1.000000
8	0.666667
9	1.000000
10	0.333333
Mean	0.716667
Std Deviation	0.258736

Figure 8: 10-fold cross validation

3.2.1 ResNet

The code built begins by specifying the input path to a dataset of fruits and considering the number of classes as 5, along with the input shape of (224, 224, 3) for image dimensions. It uses the ImageDataGenerator to preprocess and augment the data, rescaling pixel values to [0, 1], and splitting it into training and validation sets. The ResNet-50 model is employed as a base model for transfer learning, with its layers frozen to retain pre-trained weights. A custom model is built on top of ResNet-50, consisting of global average pooling, dense layers, and softmax activation. The model is then compiled with the Adam optimizer and categorical cross-entropy loss. Training is performed for 10 epochs using the training and validation generators. Additionally, 10-fold cross-validation is applied to assess model performance, and the mean accuracy and standard deviation are printed. Finally, the code includes a function to get true and predicted labels from the generator and trains the model using the specified training function. From Fig. 9 to 15 represents the code and results of ResNet.

```
In [1]: import os
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split, cross_val_score, KFold
        from sklearn.metrics import classification_report, confusion_matrix
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
        from tensorflow.keras.optimizers import Adam
        from tensorflow.keras.applications import ResNet50
        from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
In [3]:
        input_path = r'C:\Users\DELL\Documents\NCI\Semister_3\Thesis\dataset\fruits-360_dataset\fruits-360\Train'
        num_classes = 5
        input_shape = (224, 224, 3)


        datagen = ImageDataGenerator(
            rescale=1.0/255.0,
            validation_split=0.2
        )

        train_generator = datagen.flow_from_directory(
            directory=input_path,
            target_size=input_shape[:2],
            batch_size=32,
            class_mode='categorical',
            subset='training'
        )

        validation_generator = datagen.flow_from_directory(
            directory=input_path,
            target_size=input_shape[:2],
            batch_size=32,
            class_mode='categorical',
            subset='validation'
```

Figure 9: resnet code

```
Found 1881 images belonging to 5 classes.
Found 469 images belonging to 5 classes.
```

```
In [4]: def build_resnet_model():
            base_model = ResNet50(include_top=False, input_shape=input_shape, weights='imagenet')
            for layer in base_model.layers:
                layer.trainable = False
            model = Sequential()
            model.add(base_model)
            model.add(GlobalAveragePooling2D())  # Add GlobalAveragePooling2D layer
            model.add(Dense(256, activation='relu'))
            model.add(Dense(num_classes, activation='softmax'))
            return model
```

```
In [5]: def train_model(model):
            model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
            history = model.fit(
                train_generator,
                steps_per_epoch=train_generator.samples // train_generator.batch_size,
                validation_data=validation_generator,
                validation_steps=validation_generator.samples // validation_generator.batch_size,
                epochs=10
            )
            return history
```

```
In [ ]: def create_resnet_model():
            model = build_resnet_model()
            model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
            return model

        kfold = KFold(n_splits=10, shuffle=True)
        model = create_resnet_model()
        results = cross_val_score(model, train_generator[0][0], train_generator[0][1], cv=kfold)

        print("10-Fold Cross Validation Results:")
        print(results)
        print(f"Mean Accuracy: {np.mean(results)}")
```

Figure 10: resnet code

```
In [7]: def get_predictions(generator, model):
            y_true = generator.classes
            y_pred = model.predict(generator)
            y_pred = np.argmax(y_pred, axis=1)
            return y_true, y_pred

        history = train_model(model)

        Epoch 1/10
        58/58 [==============================] - 194s 3s/step - loss: 1.4487 - accuracy: 0.3872 - val_loss: 1.0942 - val_accuracy: 0.58
        93
        Epoch 2/10
        58/58 [==============================] - 193s 3s/step - loss: 1.0092 - accuracy: 0.6582 - val_loss: 0.6667 - val_accuracy: 0.96
        65
        Epoch 3/10
        58/58 [==============================] - 192s 3s/step - loss: 0.7490 - accuracy: 0.7847 - val_loss: 0.4411 - val_accuracy: 1.00
        00
        Epoch 4/10
        58/58 [==============================] - 199s 3s/step - loss: 0.6062 - accuracy: 0.8177 - val_loss: 0.4297 - val_accuracy: 0.97
        99
        Epoch 5/10
        58/58 [==============================] - 204s 4s/step - loss: 0.4910 - accuracy: 0.8605 - val_loss: 0.2644 - val_accuracy: 1.00
        00
        Epoch 6/10
        58/58 [==============================] - 210s 4s/step - loss: 0.3985 - accuracy: 0.9081 - val_loss: 0.3268 - val_accuracy: 0.96
        21
        Epoch 7/10
        58/58 [==============================] - 219s 4s/step - loss: 0.3389 - accuracy: 0.9210 - val_loss: 0.2649 - val_accuracy: 0.93
        08
        Epoch 8/10
        58/58 [==============================] - 218s 4s/step - loss: 0.2805 - accuracy: 0.9410 - val_loss: 0.2612 - val_accuracy: 0.95
        31
        Epoch 9/10
        58/58 [==============================] - 218s 4s/step - loss: 0.2422 - accuracy: 0.9448 - val_loss: 0.1821 - val_accuracy: 0.98
        88
        Epoch 10/10
        58/58 [==============================] - 221s 4s/step - loss: 0.1998 - accuracy: 0.9611 - val_loss: 0.1749 - val_accuracy: 0.97
        77
```

Figure 11: resnet code

Classification Report:

	precision	recall	f1-score	support
0	0.20	0.18	0.19	98
1	0.20	0.20	0.20	98
2	0.20	0.22	0.21	85
3	0.27	0.27	0.27	98
4	0.13	0.13	0.13	90
accuracy			0.20	469
macro avg	0.20	0.20	0.20	469
weighted avg	0.20	0.20	0.20	469

Figure 12: resnet classification report

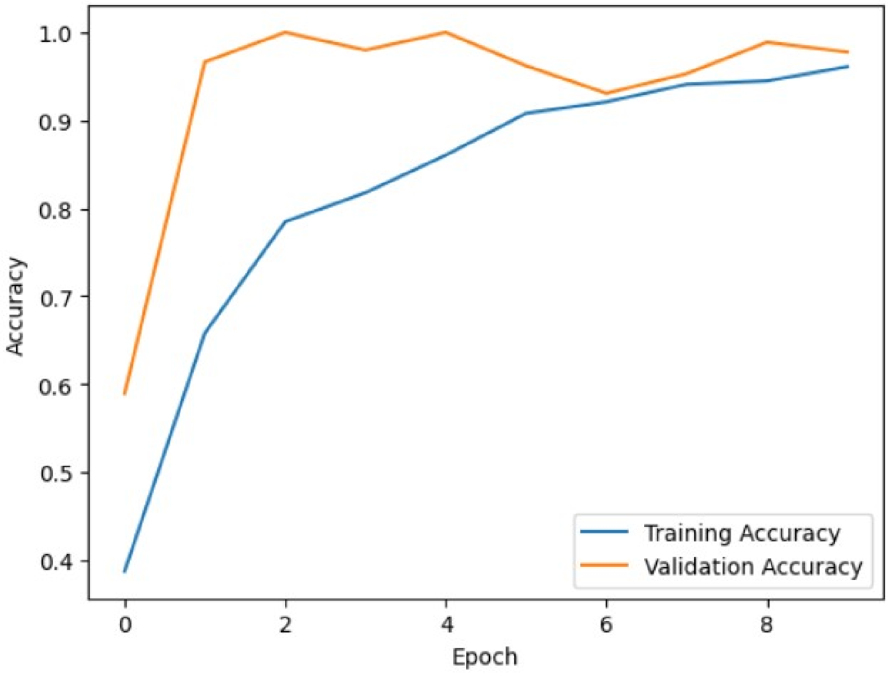


Figure 13: resnet per epoch accuracy

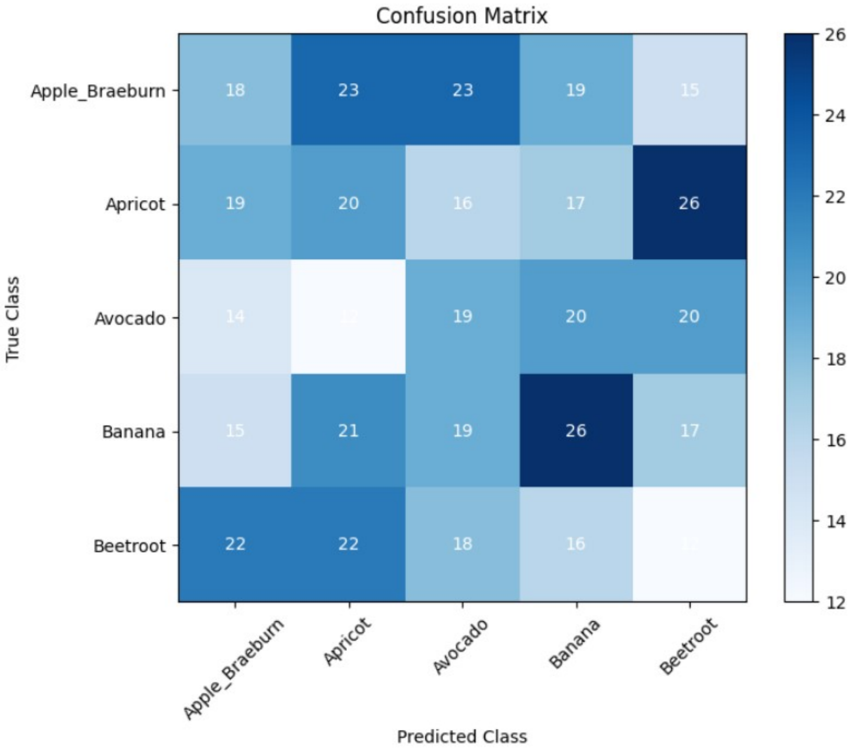


Figure 14: resnet confusion matrix

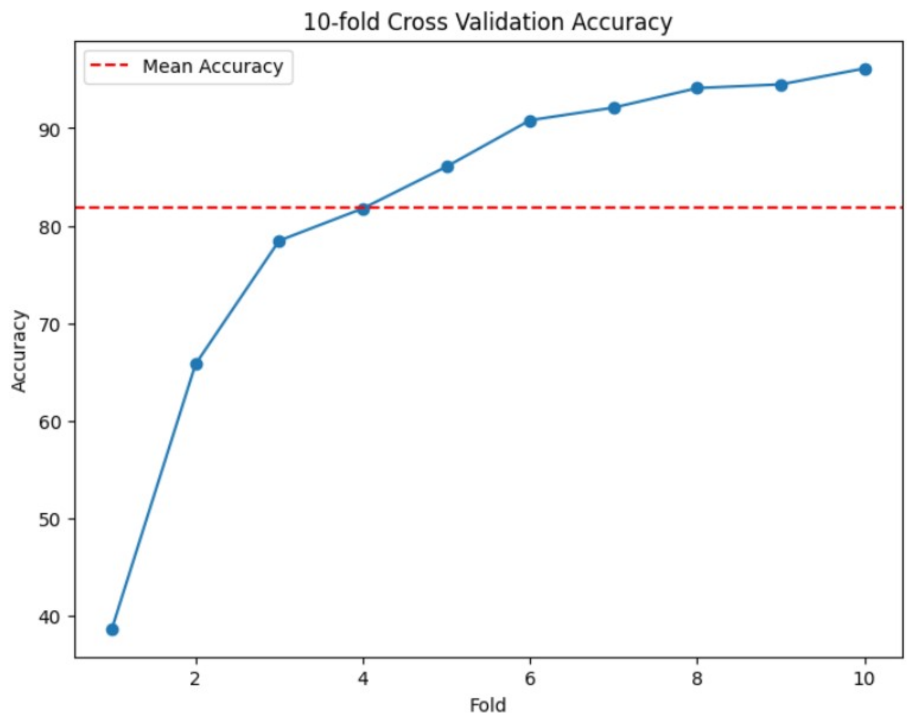


Figure 15: resnet 10-fold cross validation

3.2.2 R-CNN

The code built defines a machine-learning pipeline for training an image classification model using the ResNet50 architecture on a dataset of fruits. The dataset located in a specified directory is conisdered, and splitted into train and test subsets. Data augmentation is applied using the ImageDataGenerator, including rescaling and preprocessing to maintain the common image size and all. The model is compiled with an Adam optimizer with a learning rate of 0.001 and categorical cross-entropy loss. Training is conducted for ten epochs with batch size 32. Additionally, functions to obtain true and predicted labels from the generator and to train the model are defined. From Fig. 16 to 15 represents the code and results of RCNN.

```
In [1]: import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet import preprocess_input
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
In [3]: input_path = r'C:\Users\DELL\Documents\NCI\Semister_3\Thesis\dataset\fruits-360_dataset\fruits-360\Train'
num_classes = 5
input_shape = (224, 224, 3)

datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    validation_split=0.2,
    preprocessing_function=preprocess_input
)

train_generator = datagen.flow_from_directory(
    directory=input_path,
    target_size=input_shape[:2],
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

validation_generator = datagen.flow_from_directory(
    directory=input_path,
    target_size=input_shape[:2],
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
```

Figure 16: RCNN code

```
Found 1881 images belonging to 5 classes.
Found 469 images belonging to 5 classes.
```

```
In [4]: def build_random_rcnn_model():
    base_model = ResNet50(include_top=False, input_shape=input_shape, weights='imagenet')
    for layer in base_model.layers:
        layer.trainable = False
    model = Sequential()
    model.add(base_model)
    model.add(GlobalAveragePooling2D())
    model.add(Dense(256, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    return model
```

```
In [5]: def train_model(model):
    model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
    history = model.fit(
        train_generator,
        steps_per_epoch=train_generator.samples // train_generator.batch_size,
        validation_data=validation_generator,
        validation_steps=validation_generator.samples // validation_generator.batch_size,
        epochs=10
    )
    return history
```

```
In [7]: def get_predictions(generator, model):
    y_true = generator.classes
    y_pred = model.predict(generator)
    y_pred = np.argmax(y_pred, axis=1)
    return y_true, y_pred

history = train_model(model)
```

```
Epoch 1/10
58/58 [==============================] - 219s 4s/step - loss: 1.3832 - accuracy: 0.4586 - val_loss: 0.9879 - val_accuracy: 0.66
74
Epoch 2/10
```

Figure 17: RCNN code

```
58/58 [==============================] - 219s 4s/step - loss: 1.3832 - accuracy: 0.4586 - val_loss: 0.9879 - val_accuracy: 0.66
74
Epoch 2/10
58/58 [==============================] - 204s 4s/step - loss: 0.9194 - accuracy: 0.7307 - val_loss: 0.6799 - val_accuracy: 0.93
08
Epoch 3/10
58/58 [==============================] - 205s 4s/step - loss: 0.6909 - accuracy: 0.7923 - val_loss: 0.4767 - val_accuracy: 0.97
77
Epoch 4/10
58/58 [==============================] - 209s 4s/step - loss: 0.5642 - accuracy: 0.8167 - val_loss: 0.3788 - val_accuracy: 0.93
30
Epoch 5/10
58/58 [==============================] - 205s 4s/step - loss: 0.4466 - accuracy: 0.8897 - val_loss: 0.2828 - val_accuracy: 0.99
55
Epoch 6/10
58/58 [==============================] - 203s 4s/step - loss: 0.3773 - accuracy: 0.8875 - val_loss: 0.3375 - val_accuracy: 0.95
54
Epoch 7/10
58/58 [==============================] - 203s 4s/step - loss: 0.3117 - accuracy: 0.9340 - val_loss: 0.3022 - val_accuracy: 0.90
62
Epoch 8/10
58/58 [==============================] - 206s 4s/step - loss: 0.2650 - accuracy: 0.9421 - val_loss: 0.2096 - val_accuracy: 0.98
66
Epoch 9/10
58/58 [==============================] - 203s 4s/step - loss: 0.2264 - accuracy: 0.9567 - val_loss: 0.1873 - val_accuracy: 0.97
99
Epoch 10/10
58/58 [==============================] - 203s 4s/step - loss: 0.1732 - accuracy: 0.9762 - val_loss: 0.0927 - val_accuracy: 1.00
00
```

Figure 18: RCNN code

```
15/15 [==============================] - 38s 2s/step
Classification Report:
              precision    recall  f1-score   support

           0       0.20      0.20      0.20        98
           1       0.23      0.23      0.23        98
           2       0.13      0.13      0.13        85
           3       0.19      0.19      0.19        98
           4       0.24      0.24      0.24        90

    accuracy                           0.20       469
   macro avg       0.20      0.20      0.20       469
weighted avg       0.20      0.20      0.20       469
```

Figure 19: RCNN classification report

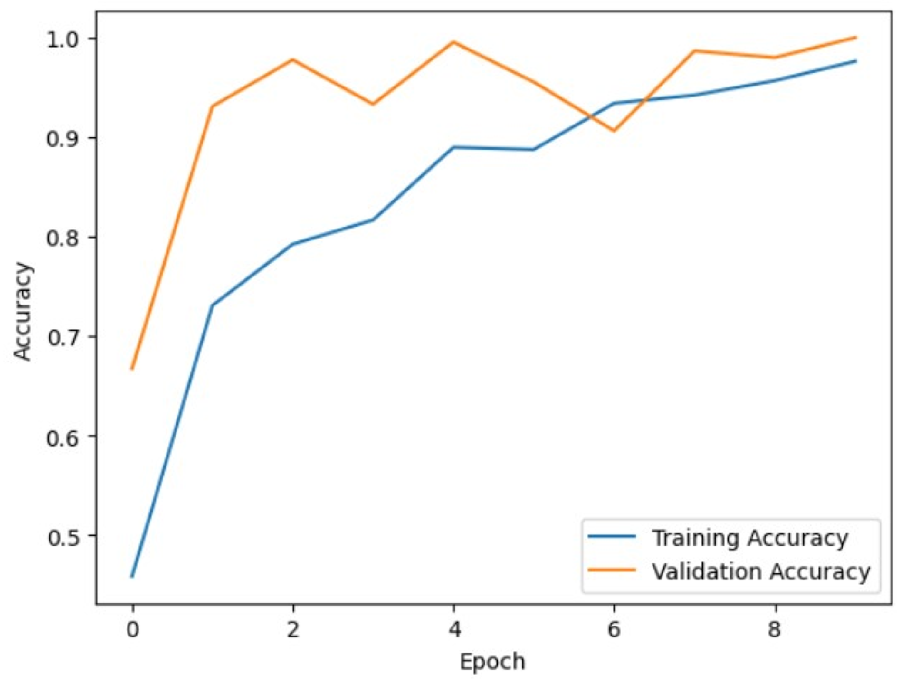


Figure 20: RCNN per epoch accuracy

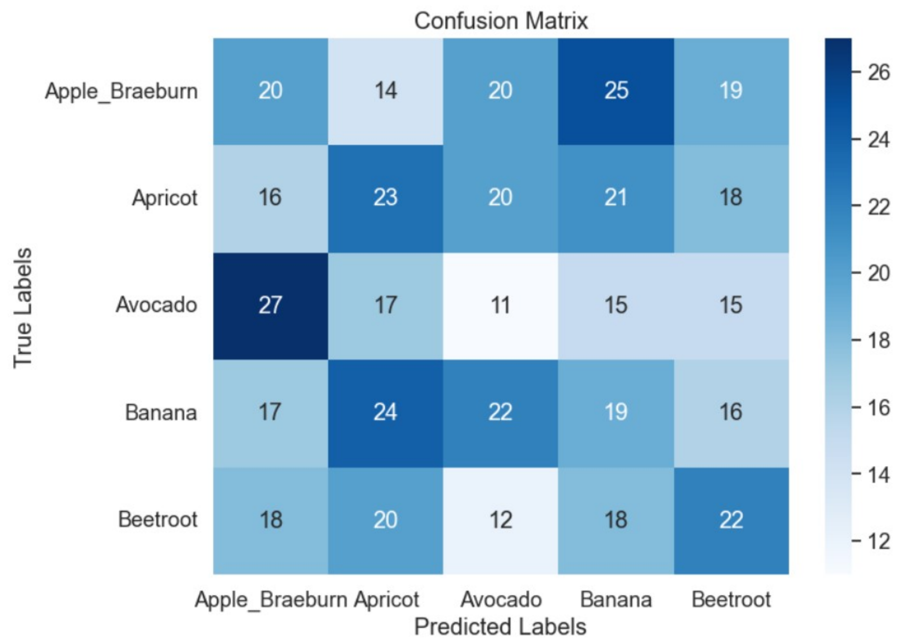


Figure 21: RCNN confusion matrix

3.2.3 AlexNet

The AlexNet model built for image classification uses TensorFlow and Keras. The data augmentation technique is implemented to address the dataset shortage issue and it uses ReLU activation functions and dropout regularization to prevent overfitting. Then images are loaded, resized to a consistent size (224x224), and normalized by dividing pixel values by 255.0 to ensure consistent input to the model. One hot encoding is used to label the fruit classes and made them suitable for categorical classification. AlexNet architecture built with multiple convolutional and pooling layers, followed by fully connected layers. The model is compiled with the Adam optimizer and categorical cross-entropy loss function, to make it suitable for multi-class classification tasks. The model is trained for 10 epochs using a batch size of 32. Finally, the summary of the model performance is evaluated by classification report, confusion matrix, 10 fold cross-validation and per epoch accuracy as shown in below Figures from 20 to 24.

```
In [43]: # AlexNet

In [3]: import os
        import cv2
        import numpy as np
        import tensorflow as tf
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
        from tensorflow.keras.optimizers import Adam
        from tensorflow.keras.preprocessing.image import ImageDataGenerator
        from sklearn.model_selection import train_test_split

        # Define fruit classes
        fruit_classes = ["Apple_Braeburn", "Apricot", "Avocado", "Banana", "Beetroot"]

        # Define data directory
        data_dir = r'C:\Users\DELL\Documents\NCI\Semister_3\Thesis\dataset\fruits-360_dataset\fruits-360\Train'

        # Define image size
        img_size = (224, 224)

        # Initialize lists for images and labels
        images = []
        labels = []

        # Data Augmentation
        datagen = ImageDataGenerator(
            rotation_range=40,
            width_shift_range=0.2,
            height_shift_range=0.2,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True,
            fill_mode='nearest'
        )
```

Figure 22: RCNN code

```
# Load and augment images
for fruit_class in fruit_classes:
    path = os.path.join(data_dir, fruit_class)
    class_num = fruit_classes.index(fruit_class)
    for img_name in os.listdir(path):
        img = cv2.imread(os.path.join(path, img_name))
        img = cv2.resize(img, img_size)

        # Apply data augmentation
        img = datagen.random_transform(img)

        images.append(img)
        labels.append(class_num)

# Convert lists to NumPy arrays
images = np.array(images)
labels = np.array(labels)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)

# Normalize pixel values
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# One-hot encode labels
y_train = tf.keras.utils.to_categorical(y_train, num_classes=len(fruit_classes))
y_test = tf.keras.utils.to_categorical(y_test, num_classes=len(fruit_classes))

# Create the AlexNet model
model = Sequential([
    Conv2D(filters=96, kernel_size=(11, 11), strides=(4, 4), activation='relu', input_shape=(224, 224, 3),
           kernel_regularizer=tf.keras.regularizers.l2(1e-4)),
    MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
    Conv2D(filters=256, kernel_size=(5, 5), activation='relu', padding="same", kernel_regularizer=tf.keras.regularizers.l2(1e-4)
    MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
    Conv2D(filters=384, kernel_size=(3, 3), activation='relu', padding="same", kernel_regularizer=tf.keras.regularizers.l2(1e-4)
    Conv2D(filters=384, kernel_size=(3, 3), activation='relu', padding="same", kernel_regularizer=tf.keras.regularizers.l2(1e-4)
    Conv2D(filters=256, kernel_size=(3, 3), activation='relu', padding="same", kernel_regularizer=tf.keras.regularizers.l2(1e-4)
```

Figure 23: AlexNet code

```
    Conv2D(filters=384, kernel_size=(3, 3), activation='relu', padding="same", kernel_regularizer=tf.keras.regularizers.l2(1e-4)
    Conv2D(filters=384, kernel_size=(3, 3), activation='relu', padding="same", kernel_regularizer=tf.keras.regularizers.l2(1e-4)
    Conv2D(filters=256, kernel_size=(3, 3), activation='relu', padding="same", kernel_regularizer=tf.keras.regularizers.l2(1e-4)
    MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
    Flatten(),
    Dense(4096, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(1e-4)),
    Dropout(0.5),
    Dense(4096, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(1e-4)),
    Dropout(0.5),
    Dense(len(fruit_classes), activation='softmax')
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

# Train the model
history = model.fit(X_train, y_train, batch_size=32, epochs=10, validation_data=(X_test, y_test))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 54, 54, 96)        34944

 max_pooling2d (MaxPooling2D  (None, 26, 26, 96)       0
 )

 conv2d_1 (Conv2D)           (None, 26, 26, 256)       614656

 max_pooling2d_1 (MaxPooling  (None, 12, 12, 256)      0
 2D)

 conv2d_2 (Conv2D)           (None, 12, 12, 384)       885120

 conv2d_3 (Conv2D)           (None, 12, 12, 384)       1327488

 conv2d_4 (Conv2D)           (None, 12, 12, 256)       884992
```

Figure 24: AlexNet code

```
 dropout_1 (Dropout)         (None, 4096)              0

 dense_2 (Dense)             (None, 5)                 20485

=================================================================
Total params: 46,767,493
Trainable params: 46,767,493
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
59/59 [==============================] - 143s 2s/step - loss: 1.8307 - accuracy: 0.5846 - val_loss: 1.1418 - val_accuracy: 0.89
36
Epoch 2/10
59/59 [==============================] - 139s 2s/step - loss: 1.0829 - accuracy: 0.9106 - val_loss: 0.9241 - val_accuracy: 0.95
96
Epoch 3/10
59/59 [==============================] - 142s 2s/step - loss: 0.8476 - accuracy: 0.9739 - val_loss: 0.7959 - val_accuracy: 0.98
09
Epoch 4/10
59/59 [==============================] - 139s 2s/step - loss: 0.7808 - accuracy: 0.9851 - val_loss: 0.7177 - val_accuracy: 1.00
00
Epoch 5/10
59/59 [==============================] - 136s 2s/step - loss: 0.7081 - accuracy: 0.9968 - val_loss: 0.6836 - val_accuracy: 1.00
00
Epoch 6/10
59/59 [==============================] - 140s 2s/step - loss: 0.7075 - accuracy: 0.9888 - val_loss: 0.6728 - val_accuracy: 0.99
15
Epoch 7/10
59/59 [==============================] - 142s 2s/step - loss: 0.7005 - accuracy: 0.9840 - val_loss: 0.6510 - val_accuracy: 1.00
00
Epoch 8/10
59/59 [==============================] - 156s 3s/step - loss: 0.6569 - accuracy: 0.9904 - val_loss: 0.6637 - val_accuracy: 0.98
30
Epoch 9/10
59/59 [==============================] - 150s 3s/step - loss: 0.6168 - accuracy: 0.9968 - val_loss: 0.5984 - val_accuracy: 1.00
00
Epoch 10/10
59/59 [==============================] - 159s 3s/step - loss: 0.5903 - accuracy: 0.9995 - val_loss: 0.5840 - val_accuracy: 0.99
79
```

Figure 25: AlexNet code

```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

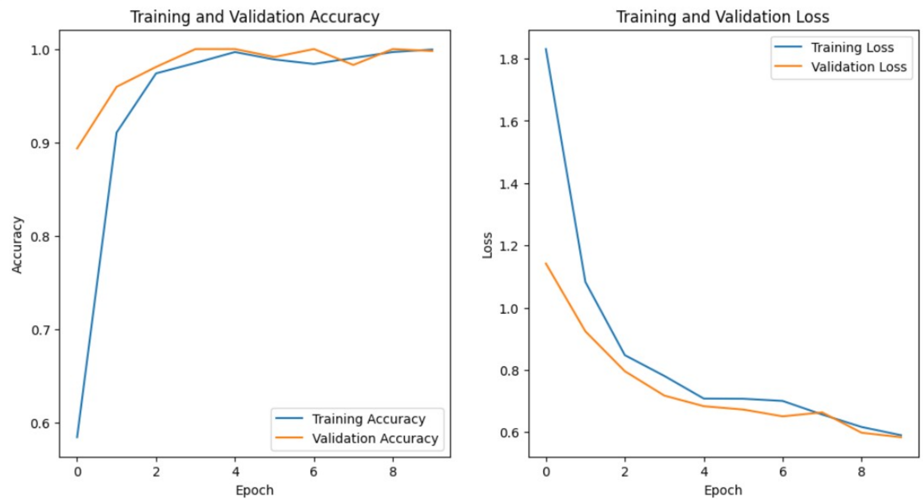


Figure 26: AlexNet per epoch accuracy

```
sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```

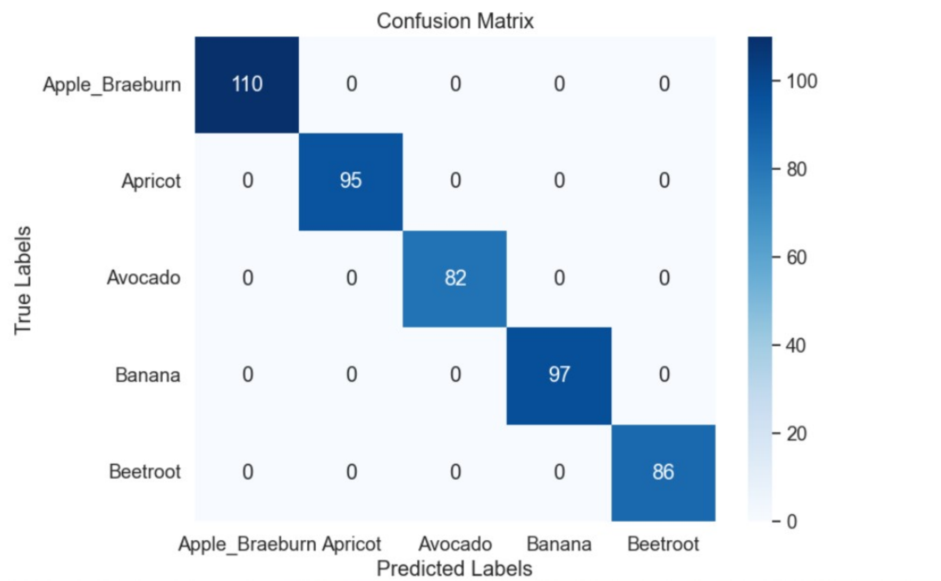


Figure 27: AlexNet confusion matrix

```
plt.text(0.5, 80, f'Standard Deviation: {std_deviation:.2f}', fontsize=12, color='purple')

# Display the plot
plt.show()
```

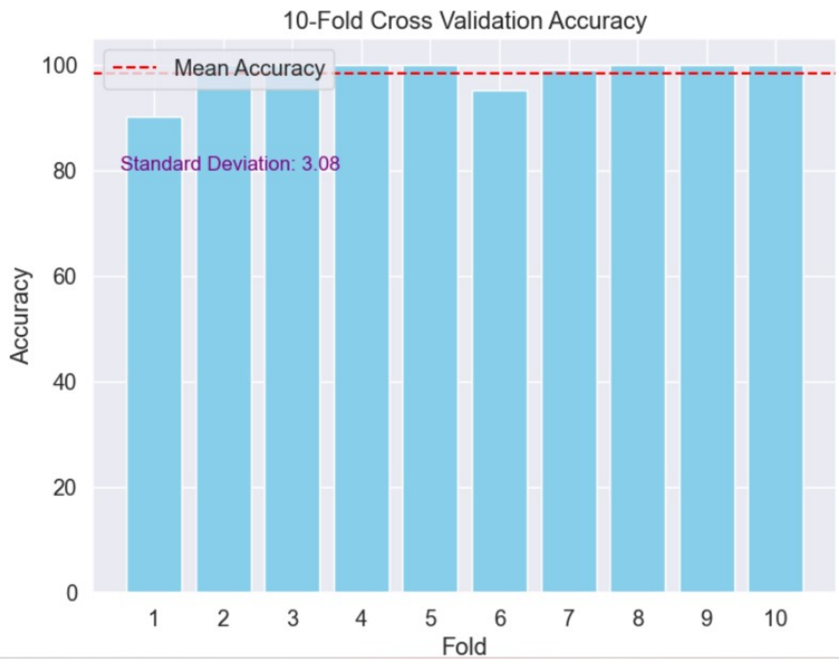


Figure 28: AlexNet 10 fold cross accuracy

3.3 Machine learning

The machine learning algorithms like SVM, Logistic regression, Naive Bayes and KNN are considered for image processing and fruits classification. All algorithms on top of the given dataset performed well, which can be confirmed by observing highest values for precision, recall, and F1-score in Table 6. Hyperparameters considered for each of the machine learning algorithms are as follows. Number of neighbours considered as default is 5 in KNN, in case of logistic regression maximum number of iterations for solver to converge is 1000 as default. Further in random forest number of decision trees in the forest as default is 100 and in SVM the type of kernel used is linear and C = 1.0 as regularization parameter. Apart from these, data augmentation and regularization with 10 pca components is considered to address overfitting issue. After build and execution of these algorithms, it is found that confusion matrix with non zero values diagonally indicates best performance of all algorithms as shown in Table 7. Algorithms like SVM, Logistic regression and KNN have high mean accuracy from 51% to 100% while Naive Bayes showed a lower cross validation score of 27.91shown in Table 8. For Naive Bayes, the standard deviation is also high in comparison with other algorithms. This indicates that the Naive Bayes shows more variability in performance. Further, it is observable that SVM, Logistic Regression, and KNN achieve higher mean cross-validation scores and lower standard deviation in comparison with Naive Bayes. By which it is clearly possible to say that SVM, Logistic Regression, and KNN perform better on top of the considered dataset than Naive Bayes. Below figures indicates code implemented for each machine learning algorithms and their results.

3.3.1 Naive Bayes

```
In [ ]: #Naive Bayes
```

```
In [1]: import os
        import cv2
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split, cross_val_score, KFold
        from sklearn.naive_bayes import GaussianNB
        from sklearn.metrics import classification_report, confusion_matrix
        from sklearn.decomposition import PCA

        def load_images(input_path, augment=True, apply_pca=True, pca_components=50):
            labels = []
            images = []
            class_names = sorted(os.listdir(input_path))

            for class_name in class_names:
                class_path = os.path.join(input_path, class_name)

                for image_file in os.listdir(class_path):
                    image_path = os.path.join(class_path, image_file)

                    image = cv2.imread(image_path)
                    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
                    image = cv2.resize(image, (100, 100))

                    if augment:
                        image = apply_augmentation(image)

                    images.append(image.flatten())
                    labels.append(class_name)

            images = np.array(images)

            if apply_pca:
                pca = PCA(n_components=pca_components)
                images = pca.fit_transform(images)
```

Figure 29: Naive bayes code

```python
    return images, np.array(labels)

def apply_augmentation(image):
    angle = np.random.randint(-15, 16)
    rows, cols = image.shape
    rotation_matrix = cv2.getRotationMatrix2D((cols/2, rows/2), angle, 1)
    image = cv2.warpAffine(image, rotation_matrix, (cols, rows))

    if np.random.rand() > 0.5:
        image = cv2.flip(image, 1)

    return image

def split_dataset(images, labels, test_size=0.2, random_state=42):
    return train_test_split(images, labels, test_size=test_size, random_state=random_state)

def build_naive_bayes_classifier():
    return GaussianNB()

def evaluate_classifier(classifier, X_train, y_train, X_test, y_test):
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)

    print("Classification Report:")
    print(classification_report(y_test, y_pred))

    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

def cross_validate(classifier, images, labels, n_splits=10):
    kfold = KFold(n_splits=n_splits)
    scores = cross_val_score(classifier, images, labels, cv=kfold)

    print("Cross-validation Scores (%):")
    for fold, score in enumerate(scores, start=1):
        score_percentage = score * 100  # Convert accuracy to percentage
        print(f"Fold {fold}: {score_percentage:.2f}%")
```

Figure 30: Naive bayes code

```python
    mean_score_percentage = np.mean(scores) * 100
    std_score_percentage = np.std(scores) * 100
    print(f"Mean Cross-validation : {mean_score_percentage:.2f}%")
    print(f"Standard Deviation : {std_score_percentage:.2f}%")

if __name__ == '__main__':
    input_path = r'C:\Users\DELL\Documents\NCI\Semister_3\Thesis\dataset\fruits-360_dataset\fruits-360\Train'

    images, labels = load_images(input_path)

    X_train, X_test, y_train, y_test = split_dataset(images, labels)

    naive_bayes_classifier = build_naive_bayes_classifier()

    evaluate_classifier(naive_bayes_classifier, X_train, y_train, X_test, y_test)

    cross_validate(naive_bayes_classifier, images, labels, n_splits=10)
```

Figure 31: Naive bayes code

3.3.2 KNN

```
In [1]: # KNN
```

```
In [30]: import os
import numpy as np
import cv2
import random
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

def load_images(input_path, augment=True, apply_pca=True, pca_components=10):
    labels = []
    images = []
    class_names = sorted(os.listdir(input_path))

    for i, class_name in enumerate(class_names):
        class_path = os.path.join(input_path, class_name)

        for image_file in os.listdir(class_path):
            image_path = os.path.join(class_path, image_file)

            image = cv2.imread(image_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            image = cv2.resize(image, (100, 100))

            if augment:
                image = apply_augmentation(image)

            images.append(image.flatten())
            labels.append(i)

    images = np.array(images)

    if apply_pca:
        pca = PCA(n_components=pca_components)
        images = pca.fit_transform(images)
```

Figure 32: KNN code

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

def apply_augmentation(image):
    angle = random.randint(-15, 15)
    rows, cols = image.shape
    rotation_matrix = cv2.getRotationMatrix2D((cols/2, rows/2), angle, 1)
    image = cv2.warpAffine(image, rotation_matrix, (cols, rows))

    if random.random() < 0.5:
        image = cv2.flip(image, 1)

    return image

def split_dataset(images, labels, test_size=0.2, random_state=42):
    return train_test_split(images, labels, test_size=test_size, random_state=random_state)

def build_knn_classifier(n_neighbors=5):
    return KNeighborsClassifier(n_neighbors=n_neighbors)

def cross_validate(classifier, images, labels, n_splits=10):
    kfold = KFold(n_splits=n_splits)
    scores = cross_val_score(classifier, images, labels, cv=kfold)

    print("Cross-validation Scores (%):")
    for fold, score in enumerate(scores, start=1):
        score_percentage = score * 100  # Convert accuracy to percentage
        print(f"Fold {fold}: {score_percentage:.2f}%")

    mean_score_percentage = np.mean(scores) * 100
    print(f"Mean Cross-validation : {mean_score_percentage:.2f}%")
    std_score_percentage = np.std(scores) * 100
    print(f"Standard Deviation : {std_score_percentage:.2f}%")

if __name__ == '__main__':
    input_path = r'C:\Users\DELL\Documents\NCI\Semister_3\Thesis\dataset\fruits-360_dataset\fruits-360\train_five_fruits'

    images, labels = load_images(input_path)
```

Figure 33: KNN code

```python
if __name__ == '__main__':
    input_path = r'C:\Users\DELL\Documents\NCI\Semister_3\Thesis\dataset\fruits-360_dataset\fruits-360\train_five_fruits'

    images, labels = load_images(input_path)

    X_train, X_test, y_train, y_test = split_dataset(images, labels)

    knn_classifier = build_knn_classifier(n_neighbors=5)

    # Fit the model on the training data
    knn_classifier.fit(X_train, y_train)

    # Rename class labels
    class_names = ["Apple_Braeburn", "Apricot", "Avocado", "Banana", "Beetroot"]
    y_test_mapped = [class_names[i] for i in y_test]
    y_pred = knn_classifier.predict(X_test)
    y_pred_mapped = [class_names[i] for i in y_pred]

    # Print classification report with renamed classes
    print("Classification Report:")
    print(classification_report(y_test_mapped, y_pred_mapped))

    train_and_plot(knn_classifier, X_train, y_train, X_test, y_test, epochs=100)

    cross_validate(knn_classifier, images, labels, n_splits=10)
```

Figure 34: KNN code

3.4 Logistic regression

```python
In [ ]: # Logistic regression

In [10]: import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.decomposition import PCA

def load_images(input_path, augment=True, apply_pca=True, pca_components=50):
    class_names = sorted(os.listdir(input_path))
    labels = []
    images = []

    for i, class_name in enumerate(class_names):
        class_path = os.path.join(input_path, class_name)

        for image_file in os.listdir(class_path):
            image_path = os.path.join(class_path, image_file)

            image = cv2.imread(image_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            image = cv2.resize(image, (100, 100))

            if augment:
                image = apply_augmentation(image)

            images.append(image.flatten())
            labels.append(class_name)  # Use class_name as the label

    images = np.array(images)
```

Figure 35: Logistic regression code

```
    if apply_pca:
        pca = PCA(n_components=pca_components)
        images = pca.fit_transform(images)

    return images, np.array(labels)

def apply_augmentation(image):
    angle = np.random.randint(-15, 16)
    rows, cols = image.shape
    rotation_matrix = cv2.getRotationMatrix2D((cols/2, rows/2), angle, 1)
    image = cv2.warpAffine(image, rotation_matrix, (cols, rows))

    if np.random.rand() > 0.5:
        image = cv2.flip(image, 1)

    return image

def split_dataset(images, labels, test_size=0.2, random_state=42):
    return train_test_split(images, labels, test_size=test_size, random_state=random_state)

def build_logistic_regression_classifier():
    return LogisticRegression(max_iter=1000)

def train_and_plot(classifier, X_train, y_train, X_test, y_test, epochs=100):
    train_accuracy = []
    test_accuracy = []

    for epoch in range(1, epochs + 1):
        classifier.fit(X_train, y_train)
        train_accuracy.append(classifier.score(X_train, y_train))
        test_accuracy.append(classifier.score(X_test, y_test))
```

Figure 36: Logistic regression code

```
def train_and_plot(classifier, X_train, y_train, X_test, y_test, epochs=100):
    train_accuracy = []
    test_accuracy = []

    for epoch in range(1, epochs + 1):
        classifier.fit(X_train, y_train)
        train_accuracy.append(classifier.score(X_train, y_train))
        test_accuracy.append(classifier.score(X_test, y_test))

    # Plot accuracy per epoch
    plt.figure(figsize=(8, 6))
    plt.plot(range(1, epochs + 1), train_accuracy, label="Train Accuracy")
    plt.plot(range(1, epochs + 1), test_accuracy, label="Test Accuracy")
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.title('Accuracy per Epoch')
    plt.grid()
    plt.show()

def evaluate_classifier(classifier, X_train, y_train, X_test, y_test):
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

def cross_validate(classifier, images, labels, n_splits=10):
    kfold = KFold(n_splits=n_splits)
    scores = cross_val_score(classifier, images, labels, cv=kfold)

    print("Cross-validation Scores (%):")
    for fold, score in enumerate(scores, start=1):
        score_percentage = score * 100  # Convert accuracy to percentage
        print(f"Fold {fold}: {score_percentage:.2f}%")
```

Figure 37: Logistic regression code

```python
def cross_validate(classifier, images, labels, n_splits=10):
    kfold = KFold(n_splits=n_splits)
    scores = cross_val_score(classifier, images, labels, cv=kfold)

    print("Cross-validation Scores (%):")
    for fold, score in enumerate(scores, start=1):
        score_percentage = score * 100  # Convert accuracy to percentage
        print(f"Fold {fold}: {score_percentage:.2f}%")

    mean_score_percentage = np.mean(scores) * 100
    std_score_percentage = np.std(scores) * 100
    print(f"Mean Cross-validation : {mean_score_percentage:.2f}%")
    print(f"Standard Deviation : {std_score_percentage:.2f}%")

if __name__ == '__main__':
    input_path = r'C:\Users\DELL\Documents\NCI\Semister_3\Thesis\dataset\fruits-360_dataset\fruits-360\Train'

    images, labels = load_images(input_path)

    X_train, X_test, y_train, y_test = split_dataset(images, labels)

    logistic_regression_classifier = build_logistic_regression_classifier()

    evaluate_classifier(logistic_regression_classifier, X_train, y_train, X_test, y_test)

    train_and_plot(logistic_regression_classifier, X_train, y_train, X_test, y_test, epochs=100)

    cross_validate(logistic_regression_classifier, images, labels, n_splits=10)
```

Figure 38: Logistic regression code

3.5 SVM

```python
In [3]: # SVM
```

```python
In [2]: import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV, KFold, cross_val_score
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.decomposition import PCA

input_path = r'C:\Users\DELL\Documents\NCI\Semister_3\Thesis\dataset\fruits-360_dataset\fruits-360\Train'
classes = ['Apple_Braeburn', 'Apricot', 'Avocado', 'Banana', 'Beetroot']

def load_images(path, classes, augment=True, apply_pca=True, pca_components=50):
    images = []
    labels = []
    for class_idx, class_name in enumerate(classes):
        class_path = os.path.join(path, class_name)
        for img_name in os.listdir(class_path):
            img_path = os.path.join(class_path, img_name)
            img = cv2.imread(img_path)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            img = cv2.resize(img, (100, 100))

            if augment:
                img = apply_augmentation(img)

            images.append(img.flatten())
            labels.append(class_idx)

    images = np.array(images)

    if apply_pca:
        pca = PCA(n_components=pca_components)
        images = pca.fit_transform(images)
```

Figure 39: SVM code

```python
    return images, np.array(labels)

def apply_augmentation(image):
    angle = np.random.randint(-15, 16)
    rows, cols = image.shape
    rotation_matrix = cv2.getRotationMatrix2D((cols/2, rows/2), angle, 1)
    image = cv2.warpAffine(image, rotation_matrix, (cols, rows))

    if np.random.rand() > 0.5:
        image = cv2.flip(image, 1)

    return image

X, y = load_images(input_path, classes, augment=True, apply_pca=True, pca_components=50)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Use GridSearchCV to find the optimal hyperparameters
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1]}
grid_search = GridSearchCV(SVC(kernel='rbf'), param_grid, cv=10)
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_

# Train the model with the best hyperparameters
best_model = SVC(kernel='rbf', C=best_params['C'], gamma=best_params['gamma'], random_state=42)

# Initialize lists to store accuracy values per epoch
train_accuracy = []
test_accuracy = []

# Training loop
epochs = 100
for epoch in range(epochs):
    best_model.fit(X_train, y_train)
    train_accuracy.append(best_model.score(X_train, y_train))
    test_accuracy.append(best_model.score(X_test, y_test))
```

Figure 40: SVM code

```python
# Training loop
epochs = 100
for epoch in range(epochs):
    best_model.fit(X_train, y_train)
    train_accuracy.append(best_model.score(X_train, y_train))
    test_accuracy.append(best_model.score(X_test, y_test))

# Plot accuracy per epoch
plt.figure(figsize=(8, 6))
plt.plot(range(1, epochs + 1), train_accuracy, label="Train Accuracy")
plt.plot(range(1, epochs + 1), test_accuracy, label="Test Accuracy")
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy per Epoch')
plt.grid()
plt.show()

# Make predictions
y_pred = best_model.predict(X_test)

# Rename class labels
class_names = ['Apple_Braeburn', 'Apricot', 'Avocado', 'Banana', 'Beetroot']
y_test_mapped = [class_names[i] for i in y_test]
y_pred_mapped = [class_names[i] for i in y_pred]

print("Classification Report:")
report = classification_report(y_test_mapped, y_pred_mapped)
print(report)

print("Confusion Matrix:")
confusion = confusion_matrix(y_test_mapped, y_pred_mapped)
print(confusion)

kfold = KFold(n_splits=10, shuffle=True, random_state=42)
results = cross_val_score(best_model, X, y, cv=kfold)
```

Figure 41: SVM code

Fig. 42 indicates precision, recall, F1 score for Naive Bayes is around 95%, for KNN it is 93%, logistic it is 98% and for SVM it is 100%. These accuracy marks shows machine learning algorithms considered have best accuracy. Fig. 43 consists of comparison of machine learning classification report. In which we can find diagonal value populated indicates machine learning algorithms considered performing better. Further the Fig. 44 gives comparison of mean accuracy of all machine learnign algorithms considered in which we can see that Naiver Bayes performed least with 27.91% and SVM as best with 100%.

Table 8: Cross-fold Validation Scores Comparison

Algorithm	Mean Cross-validation Score
Naive Bayes	27.91%
KNN	51.86%
Logistic Regression	72.81%
SVM	100.00%

Figure 42: Machine learning 10 fold cross validation

Table 6: Machine learning Classification Report Comparison

Algorithm	Precision	Recall	F1-Score	Accuracy
Naive Bayes	0.95	0.95	0.95	0.95
KNN	0.93	0.93	0.93	0.93
Logistic Regression	0.98	0.98	0.98	0.98
SVM	1.00	1.00	1.00	1.00

Figure 43: machine learning classification

Table 7: Confusion Matrix Comparison

Algorithm	Apple Braeburn	Apricot	Avocado	Banana	Beetroot
Naive Bayes	110	95	82	97	86
KNN	103	88	72	83	91
Logistic Regression	104	95	81	96	84
SVM	110	95	82	97	86

Figure 44: Machine learning confusion matrix

3.6 Algorithm selection

Though most of the deep learning and machine learning algorithms considered providing best accuracy, actually failed in accurate detection of fruits. For example KNN model built failed in Avocado fruit detection as shown in Fig. 45, Naive Bayes wrongly predicted Avocado as Apricot as shown in figure 46. Whereas AlexNet outperformed all other algorithms by accurately detecting fruits. For example AlexNet correctly detected fruit Avocado as shown in Fig. 47. So, AlexNet is choosed as best deep learning algorithm for fruit detection.

Figure 45: Wrong prediction by KNN

Figure 46: Wrong prediction by Naive bayes

Figure 47: Accurate prediction by AlexNet

3.7 Bill generation

A local database called PostgreSQL is considered and a table called fruit and price is created. Within which fruit and their respective prices are maintained as whown in Fig. 48.

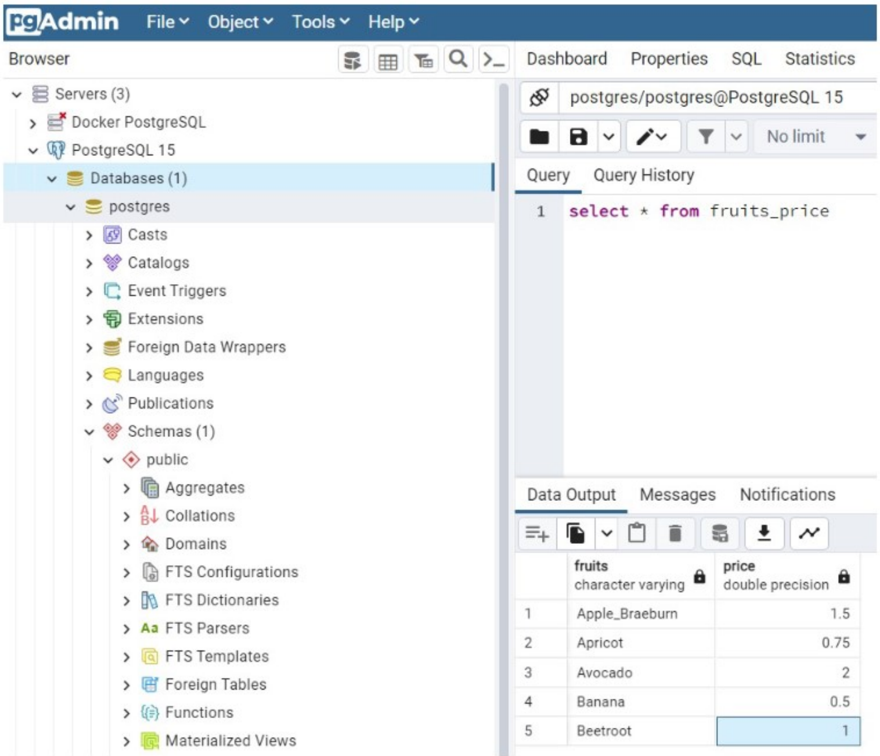


Figure 19: fruits and their respective price in dollar

Figure 48: Fruit and price table

Once the fruit is detected by the AlexNet algorithm built by taking the image. Then the respective price from the database as shown in Fig. 48 will be fetched by using the code as shown in Fig. 49 and 50. Within which sql query is maintained to tech respective price. Then by multiplying with their respective quantities total amount will be found and bill is generated as shown in Fig. 51 without using internet and cloud server.

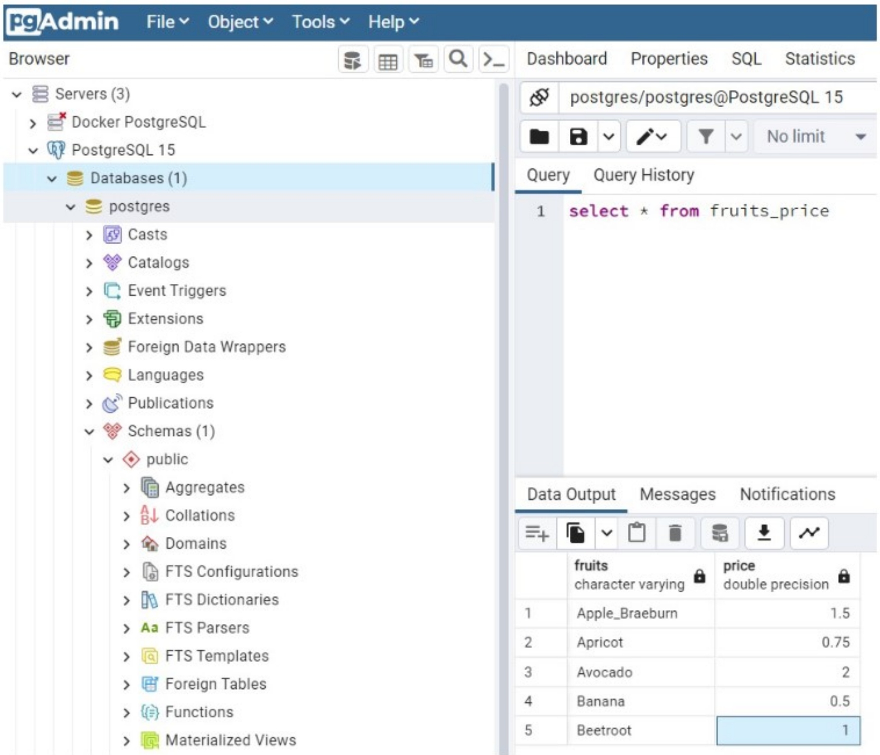


Figure 19: fruits and their respective price in dollar

Figure 49: Fruit and price table

```
Enter the quantity: 8
***********************
Fruit Emporium
123 Main Street, City
2023-09-16 22:06:27
Total price for 8.00 Banana: $4.00
***********************
Time taken: 2.044621 seconds
```

Figure 50: Bill generation code

```
In [19]: import time
         from datetime import datetime
         start_time = time.time()

         shop_name = "Fruit Emporium"
         shop_address = "123 Main Street, City"
         current_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

         while True:
             fruit_name = predicted_fruit

             if fruit_name.lower() == 'exit':
                 break
             sample_fruit_price = ["Banana","Avocaddo","Apple_Braeburn","Apricot","Beetroot"]
             if fruit_name in sample_fruit_price:
                 try:
                     quantity = float(input("Enter the quantity: "))
                     if quantity < 0:
                         print("Quantity cannot be negative.")
                     else:
                         price = fetched_value * quantity
                         print("**********************")
                         print(shop_name)
                         print(shop_address)
                         print(current_date)
                         print(f"Total price for {quantity:.2f} {fruit_name}: ${price:.2f}")
                         print("**********************")
                     break
                 except ValueError:
                     print("Invalid quantity. Please enter a valid number.")
             else:
                 print("Fruit not found in the list.")

         end_time = time.time()

         elapsed_time = end_time - start_time

         print(f"Time taken: {elapsed_time:.6f} seconds")
```

Figure 51: Bill generation code

```
Enter the quantity: 8
**********************
Fruit Emporium
123 Main Street, City
2023-09-16 22:06:27
Total price for 8.00 Banana: $4.00
**********************
Time taken: 2.044621 seconds
```

Figure 52: Bill generated

References

Federation, N. R. (2020). The 2020 national retail security survey. Accessed on September 1, 2023.
URL: *https://cdn.nrf.com/sites/default/files/2020-07/RS-105905*$_2$020$_N$*ationalRetailSecuritySurvey.pdf*