Develop and Implement - CNN based application

# PLANT PALETTE

## A Fruit and Vegetable Prediction using Deep Neural Networks

### Problem Statement

Traditional methods for fruit and vegetable classification rely on manually engineered features, which often struggle with large, diverse datasets. These methods include decision trees, SVMs, and k-NN, which lack robustness for complex patterns. The proposed solution employs Convolutional Neural Networks (CNNs), known for their superior ability to automatically learn hierarchical features directly from image data. This approach aims to enhance classification accuracy for fruits and vegetables, addressing challenges in agricultural automation and food quality inspection.

Key Challenges:

- Manual feature engineering is time-consuming and error-prone.
- Limited generalization with traditional models.
- Need for an automated, accurate system for diverse datasets.

### Dataset Used

The dataset consists of images of 36 classes of fruits and vegetables. Images were gathered from various sources to ensure diversity in appearance, environmental conditions, and lighting.

**Data Splitting**:

- Training Data: For model learning.
- Validation Data: To tune hyperparameters and prevent overfitting.
- Test Data: For final model evaluation.

## Implementation

The implementation involves building a CNN using TensorFlow and Keras frameworks. Here's a summary of the process:

1. **Model Architecture**:
    - **Convolutional Layers**: Extract spatial features.
    - **MaxPooling Layers**: Reduce dimensionality.
    - **Dropout Layers**: Prevent overfitting by randomly deactivating neurons.
    - **Flatten Layer**: Convert 2D feature maps to a 1D vector.
    - **Dense Layers**: Fully connected layers for classification.
    - **Softmax Activation**: Converts output logits into probabilities for each class.
2. **Training**:
    - Number of epochs: 30
    - Optimizer: Adam with learning rate scheduling.
    - Loss Function: Categorical cross-entropy, with class-weighted adjustments for imbalanced datasets.
    - Regularization: Batch normalization to accelerate convergence.
3. **Testing and Deployment**:
    - **Streamlit Framework**: Provides an interactive web interface where users can upload images and view predictions.

## Code

### 1. Loading the Dataset

```
training_set = tf.keras.utils.image_dataset_from_directory(
    'F:/ML/train',
    labels='inferred',
    label_mode='categorical',
    batch_size=32,
    image_size=(64, 64),
    shuffle=True
)
```

**2. Building the CNN Model**

```
cnn = tf.keras.models.Sequential()

# First Convolutional Layer
cnn.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, activation='relu',
input_shape=[64, 64, 3]))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# Second Convolutional Layer
cnn.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# Dropout Layer for Regularization
cnn.add(tf.keras.layers.Dropout(0.5))

# Flatten Layer
cnn.add(tf.keras.layers.Flatten())

# Fully Connected Layer
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))

# Output Layer
cnn.add(tf.keras.layers.Dense(units=36, activation='softmax'))
```

**3. Compiling the Model**

```
cnn.compile(optimizer='rmsprop',loss='categorical_crossentropy',
metrics=['accuracy'])
```

**4. Training the Model**

```
training_history = cnn.fit(x=training_set, validation_data=validation_set,
epochs=30)
```

**5. Saving the Model**

```
cnn.save('trained_model.h5')
```

**6. Visualizing the Training History**

```
import matplotlib.pyplot as plt

# Training Accuracy Plot
plt.plot(training_history.history['accuracy'], color='red')
plt.title('Training Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.show()

# Validation Accuracy Plot
plt.plot(training_history.history['val_accuracy'])
plt.title('Validation Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.show()
```

**7. Streamlit Implementation**

```
import streamlit as st
import tensorflow as tf
import numpy as np

# Function to load the model and make predictions
def model_predict_image(image):
    model = tf.keras.models.load_model('trained_model.h5')  # Load the trained model
    img = tf.keras.preprocessing.image.load_img(image, target_size=(64, 64))  # Preprocess the image
```

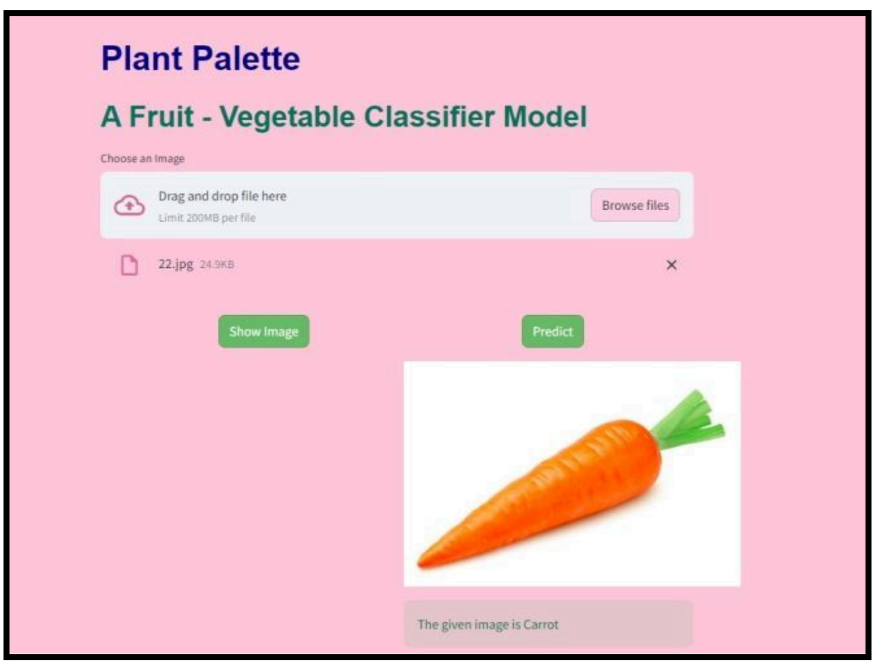
```
    input_arr = tf.keras.preprocessing.image.img_to_array(img)
    input_arr = np.array([input_arr])  # Convert image to batch format
    predictions = model.predict(input_arr)  # Predict class
    return np.argmax(predictions)

# Display the app header and description
original_title = '<h2 style="font-family:Poppins,sans-serif; color: #00008B; font-size: 40px;">Plant Palette</h2>'
st.markdown(original_title, unsafe_allow_html=True)

header = '<h3 style="font-family:Poppins,sans-serif; color: #337357; font-size: 35px;">A Fruit - Vegetable Classifier Model</h3>'
st.markdown(header, unsafe_allow_html=True)

# File uploader for the image input
image = st.file_uploader("Choose an Image")

# Layout for Show Image and Predict buttons
button_col1, button_col2 = st.columns(2)

with button_col1:
    if st.button('Show Image') and image:
        st.image(image, width=400)  # Display the uploaded image

with button_col2:
    if st.button('Predict'):
        if image is not None:
            st.spinner()
            result_index = model_predict_image(image)  # Predict the class
            with open("labels.txt") as f:
                labels = f.readlines()  # Load class labels
            st.success(f"The given image is {labels[result_index].capitalize()}")

# Custom CSS for button styling
st.markdown(
```

```
    """
    <style>
    .stButton > button {
        margin-top: 20px;
        margin-left: 140px;
        background-color: #7DB862 !important;
        color: white !important;
    }
    </style>
    """,
    unsafe_allow_html=True
)
```

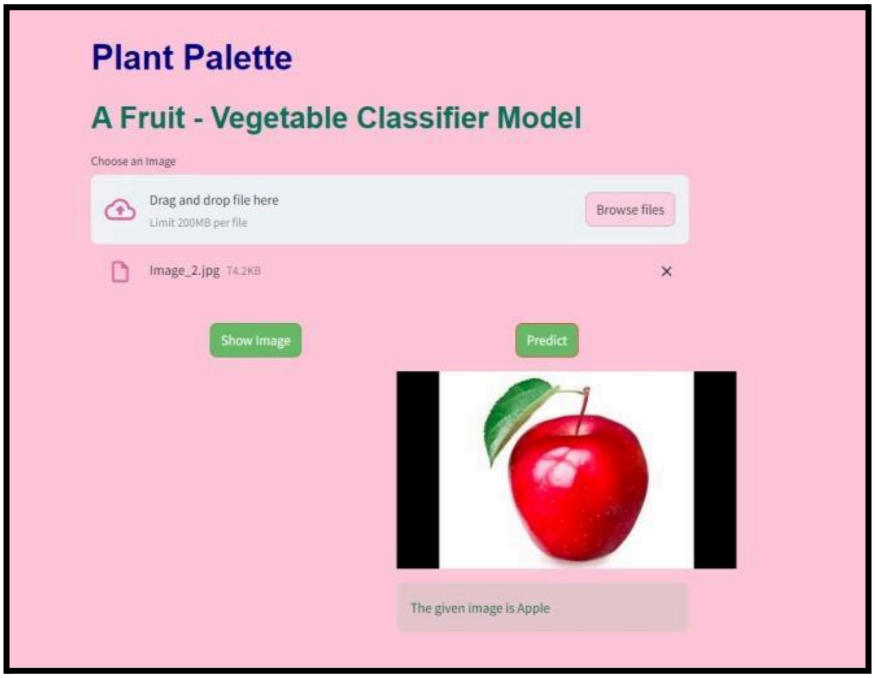
**Output**

The report includes two key screenshots from the **PlantPalette User Interface**:

1. **Prediction of a Vegetable**: Displays an image of a vegetable, followed by the model's prediction of its type.

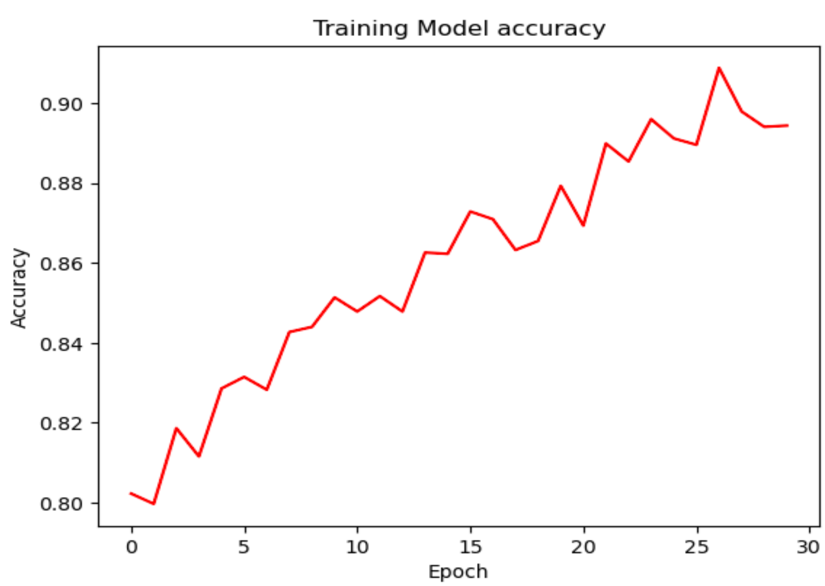


2. **Prediction of a Fruit**: Similarly, it shows a fruit image and the predicted class.

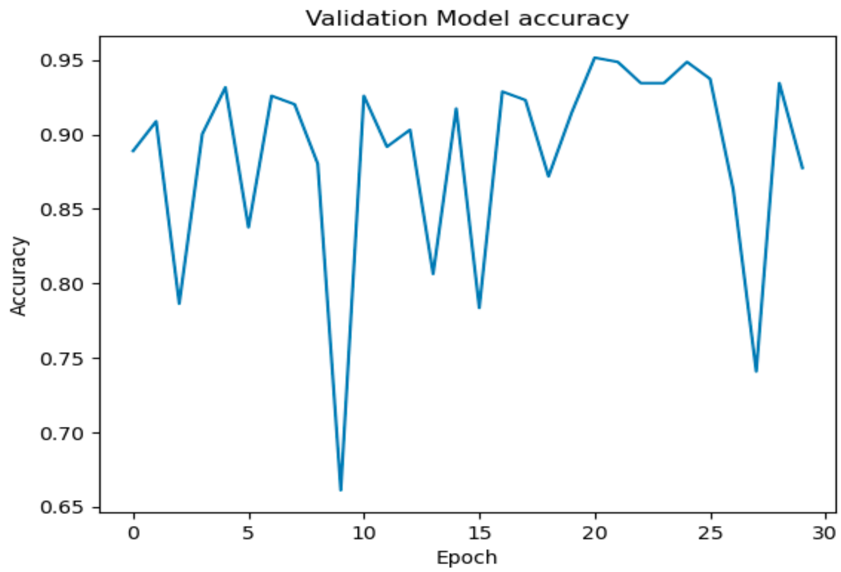



Additionally, the training and validation accuracies are visualized using line plots:

- **Training Accuracy Graph**: Shows steady improvement over epochs.



- **Validation Accuracy Graph**: Tracks the model's performance on unseen data, indicating minimal overfitting.




**Result**

The CNN model delivered:

- **Training Accuracy**: 89.4%
- **Validation Accuracy**: 87.7%

The visualizations of accuracy over epochs provide insights into model behavior:

- The training curve steadily increases, indicating effective learning.
- The validation curve closely follows, suggesting good generalization.