Model Development Phase Template

Date	18 JUNE 2025
Team ID	SWTID1749825524
Project Title	Deepfruitveg: Automated Fruit and Veg Identification
Maximum Marks	4 Marks

Initial Model Training Code, Model Validation and Evaluation Report

1. Model Training

- The dataset is loaded using TensorFlow's image_dataset_from_directory() method.
- All images are resized to **64×64 pixels** and normalized (pixel values scaled to 0–1).
- A **CNN model** is created using:
    - Conv2D layers for feature extraction
    - MaxPooling2D layers for spatial reduction
    - Dropout for regularization
    - Dense layers for classification
- The model is compiled with:
    - **Optimizer**: Adam
    - **Loss Function**: Categorical Cross-Entropy
    - **Metric**: Accuracy
- The training process uses model.fit() for **50 epochs**, with real-time validation.

-

2. Model Validation

- **20% of the dataset** is used as the **validation set**.
- Validation accuracy and loss are tracked during each epoch.
- Helps monitor **overfitting** and **generalization** during training.

3. Model Evaluation

- The final model is tested on unseen data using model.evaluate().
- Metrics analysed:
  - **Test Accuracy**
  - **Test Loss**
  - **Confusion Matrix**
- **Accuracy/loss curves** are plotted using matplotlib for visual insight into training dynamics.

Initial Model Training Code:

building Model

In [ ]:
```
cnn = tf.keras.models.Sequential()
```

Building Convolution Layer

In [ ]:
```
cnn.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,padding='same',activation='relu',input_shape=[64,64,3]))
cnn.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [ ]:
```
cnn.add(tf.keras.layers.Dropout(0.25))
```

In [ ]:
```
cnn.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,padding='same',activation='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

In [ ]:
```
cnn.add(tf.keras.layers.Dropout(0.25))


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


cnn.add(tf.keras.layers.Dense(units=512,activation='relu'))


cnn.add(tf.keras.layers.Dense(units=256,activation='relu'))
```

```
cnn.add(tf.keras.layers.Dense(units=512,activation='relu'))


cnn.add(tf.keras.layers.Dense(units=256,activation='relu'))


cnn.add(tf.keras.layers.Dropout(0.5)) #To avoid overfitting
```

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

Compiling and Training Phase

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

In [ ]:
```
cnn.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 32)	896
conv2d_1 (Conv2D)	(None, 62, 62, 32)	9,248
max_pooling2d (MaxPooling2D)	(None, 31, 31, 32)	0
dropout (Dropout)	(None, 31, 31, 32)	0
conv2d_2 (Conv2D)	(None, 31, 31, 64)	18,496
conv2d_3 (Conv2D)	(None, 29, 29, 64)	36,928
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 64)	0

conv2d_2 (Conv2D)	(None, 31, 31, 64)	18,496
conv2d_3 (Conv2D)	(None, 29, 29, 64)	36,928
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 64)	0
dropout_1 (Dropout)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 512)	6,423,040
dense_1 (Dense)	(None, 256)	131,328
dropout_2 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 36)	9,252

Total params: 6,629,188 (25.29 MB)

Trainable params: 6,629,188 (25.29 MB)

Non-trainable params: 0 (0.00 B)

[ ]:
```
training_history = cnn.fit(x=training_set,validation_data=validation_set,epochs=50)
```

```
Epoch 1/50
98/98 ━━━━━━━━━━━━━━━━━━━━ 613s 6s/step - accuracy: 0.0322 - loss: 41.8640 - val_accuracy: 0.0655 - val_loss: 3.4824
Epoch 2/50
98/98 ━━━━━━━━━━━━━━━━━━━━ 172s 2s/step - accuracy: 0.0573 - loss: 3.4807 - val_accuracy: 0.1026 - val_loss: 3.3357
Epoch 3/50
98/98 ━━━━━━━━━━━━━━━━━━━━ 204s 2s/step - accuracy: 0.0686 - loss: 3.3621 - val_accuracy: 0.1852 - val_loss: 3.1095
Epoch 4/50
98/98 ━━━━━━━━━━━━━━━━━━━━ 190s 2s/step - accuracy: 0.1046 - loss: 3.2237 - val_accuracy: 0.3390 - val_loss: 2.8057
Epoch 5/50
98/98 ━━━━━━━━━━━━━━━━━━━━ 146s 1s/step - accuracy: 0.1528 - loss: 3.0567 - val_accuracy: 0.4786 - val_loss: 2.4621
Epoch 6/50
98/98 ━━━━━━━━━━━━━━━━━━━━ 155s 2s/step - accuracy: 0.2228 - loss: 2.8047 - val_accuracy: 0.6211 - val_loss: 1.7898
```

Model Validation and Evaluation Report:

Evaluating Model

In [ ]:

```
#Training set Accuracy
train_loss, train_acc = cnn.evaluate(training_set)
print('Training accuracy:', train_acc)
```

```
98/98 ━━━━━━━━━━━━━━━━━━━━ 90s 913ms/step - accuracy: 0.9870 - loss: 0.0466
Training accuracy: 0.9878399968147278
```

In [ ]:

```
#Validation set Accuracy
val_loss, val_acc = cnn.evaluate(validation_set)
print('Validation accuracy:', val_acc)
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 12s 1s/step - accuracy: 0.9537 - loss: 0.2568
Validation accuracy: 0.9572649598121643
```

Saving Model

In [ ]:

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

In [ ]:

```
training_history.history #Return Dictionary of history
```

In [ ]:

```
#Recording History in json
import json
with open('training_hist.json','w') as f:
  json.dump(training_history.history,f)
```

In [ ]:

```
print(training_history.history.keys())
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

Calculating Accuracy of Model Achieved on Validation set

In [ ]:

```
print("Validation set Accuracy: {} %".format(training_history.history['val_accuracy'][-1]*1
```

```
Validation set Accuracy: 95.72649598121643 %
```

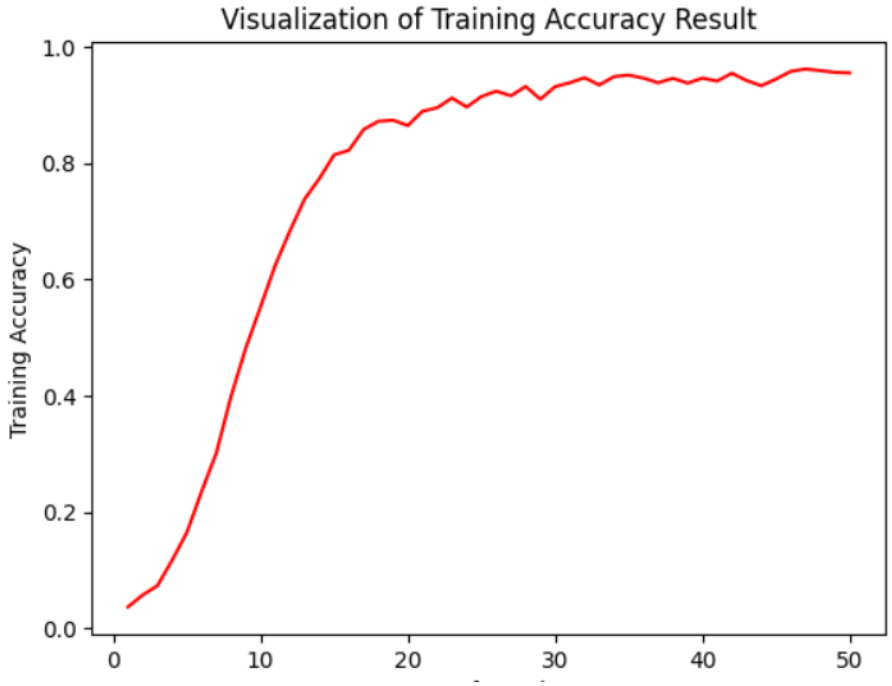
Training Visualization

In [ ]:

```
#training_history.history['accuracy']
```

In [ ]:

```
epochs = [i for i in range(1,51)]
plt.plot(epochs,training_history.history['accuracy'],color='red')
plt.xlabel('No. of Epochs')
plt.ylabel('Training Accuracy')
plt.title('Visualization of Training Accuracy Result')
plt.show()
```



Validation Accuracy

In [ ]:

```
plt.plot(epochs,training_history.history['val_accuracy'],color='blue')
plt.xlabel('No. of Epochs')
plt.ylabel('Validation Accuracy')
plt.title('Visualization of Validation Accuracy Result')
plt.show()
```