Model Optimization and Tuning Phase Template

Date	20 July 2024
Name	Sourabh Sanjay Dabhade
Project Title	Greenclassify: Deep Learning-Based Approach For Vegetable Image Classification
Maximum Marks	10 Marks

Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase involves refining neural network models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Hyperparameter Tuning Documentation (8 Marks):

Model	Tuned Hyperparameters

CNN

```
## Add layers to cnn model

# INPUT AND HIDDEN LAYERS

# Convolutional Layer
model.add(Conv2D(filters = 32,
                 kernel_size = 3,
                 padding = "same",
                 activation = "relu",
                 input_shape = [224, 224, 3])
          )

# Pooling Layer
model.add(MaxPooling2D(pool_size = (2,2)))

# Convolutional Layer
model.add(Conv2D(filters = 64,
                 kernel_size = 3,
                 padding = "same",
                 activation = "relu",)
          )

# Pooling Layer
model.add(MaxPooling2D())
```

```
# CLASSIFICATION

# Flatten Layer
model.add(Flatten())

# Fully Connected Layer
model.add(Dense(128, activation = "relu"))

# Output Layer
model.add(Dense(15, activation = "softmax"))
```

`filters:` The number of filters (or feature maps) to be learned by the convolutional layer. In this case, the first layer has 32 filters, and the second layer has 64 filters.

`kernel_size:` The size of the convolution window, which is 3x3 in this case.

`padding:` The padding strategy used, which is "same" in this case, meaning the output feature map will have the same spatial dimensions as the input.

	`activation:` The activation function used, which is ReLU (Rectified Linear Unit) in this case. **`input_shape:`** The shape of the input data, which is [224, 224, 3] (height, width, channels) for the first layer. **`pool_size`**: The size of the pooling window, which is 2x2 in this case, meaning the spatial dimensions of the feature maps will be reduced by a factor of 2. **`ACCURACY:93.51`**

VGG16

```
vgg = VGG16(include_top=False,input_shape=(224,224,3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
```

```
for layer in vgg.layers:
  layer.trainable=False
```

```
x = Flatten()(vgg.output)
```

```
output = Dense(15,activation='softmax')(x)
```

```
vgg16 = Model(vgg.input,output)
```

`VGG16(include_top=False, input_shape=(224, 224, 3))`: This creates a pre-trained VGG16 model, a popular convolutional neural network architecture, without the top (fully connected) layers. The input_shape parameter specifies the expected input size of the model, which is 224x224 pixels with 3 color channels (RGB).

`x = Flatten()(vgg.output)`: This applies a Flatten layer to the output of the VGG16 model, converting the 2D feature maps into a 1D vector.

`output = Dense(15, activation='softmax')(x)`: This adds a new fully connected layer with 15 units (corresponding to the number of classes) and a softmax activation function. The softmax activation ensures that the output represents a probability distribution over the classes.

ACCURACY:97.80

Resnet50

BASE MODEL:

```
resnet50 = ResNet50(include_top=False,input_shape=(224,224,3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 5s 0us/step
```

```
for layer in resnet50.layers:
  layer.trainable=False
```

```
x = Flatten()(resnet50.output)
output = Dense(15,activation='softmax')(x)
resnet50 = Model(resnet50.input,output)
```

`include_top=False`: Excludes the fully connected layers to allow customization for specific tasks.

`input_shape=(224,224,3)`: Specifies the input image dimensions and color channels.

`layer.trainable=False`: Freezes the weights of the pre-trained layers to retain learned features.

`x = Flatten()(resnet50.output)`: Converts the output of the last convolutional layer to a 1D tensor.

`output = Dense(15, activation='softmax')(x)`: Adds a dense layer with 15 units and softmax activation for classification.

ACCURACY: 59.27

HYPERPARAMETER TUNING:

```
import keras_tuner as kt
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define a model-building function
def build_model(hp):
    inception = InceptionV3(include_top=False, input_shape=(224, 224, 3))

    # Freeze the layers in the base model
    for layer in inception.layers:
        layer.trainable = False

    x = Flatten()(inception.output)

    # Tune the number of units in the Dense layer
    units = hp.Int('units', min_value=32, max_value=512, step=32)
    x = Dense(units, activation='relu')(x)

    # Tune the learning rate for the optimizer
    learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

    output = Dense(15, activation='softmax')(x)

    model = Model(inception.input, output)
```

```
    model.compile(
        loss='categorical_crossentropy',
        optimizer=Adam(learning_rate=learning_rate),
        metrics=['accuracy']
    )

    return model

# Set up the Keras Tuner
tuner = kt.RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=3,  # Number of hyperparameter combinations to try
    executions_per_trial=1,  # Number of models to train per trial for robustness
    directory='kt_dir',
    project_name='inceptionv3_tuning'
)


# Search for the best hyperparameters
tuner.search(train_data, validation_data=validation_data, epochs=5)

# Get the best hyperparameters and model
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
best_model = tuner.get_best_models(num_models=1)[0]

print(f"""
The hyperparameter search is complete.
The optimal number of units in the dense layer is {best_hps.get('units')}.
The optimal learning rate for the optimizer is {best_hps.get('learning_rate')}.
""")
```

```
Trial 3 Complete [00h 17m 31s]
val_accuracy: 0.9933333396911621

Best val_accuracy So Far: 0.996666669845581
Total elapsed time: 00h 50m 33s

The hyperparameter search is complete.
The optimal number of units in the dense layer is 32.
The optimal learning rate for the optimizer is 0.0001.
```

The code uses Keras Tuner to optimize the hyperparameters of an InceptionV3-based model. It tunes the number of units in the Dense layer and the learning rate of the Adam optimizer. The search is performed over 3 trials, and the best hyperparameters found are 32 units in the Dense layer and a learning rate of 0.0001. The final validation accuracy achieved is 0.9933.

RETRAIN THE MODEL:

```
best_model.fit(train_data, validation_data=validation_data, epochs=5)
```

```
Epoch 1/5
235/235 [==============================] - 212s 862ms/step - loss: 0.0144 - accuracy: 0.9963 - val_loss: 0.0144 - val_accuracy:
0.9970
Epoch 2/5
235/235 [==============================] - 190s 809ms/step - loss: 0.0130 - accuracy: 0.9971 - val_loss: 0.0160 - val_accuracy:
0.9960
Epoch 3/5
235/235 [==============================] - 195s 828ms/step - loss: 0.0091 - accuracy: 0.9974 - val_loss: 0.0108 - val_accuracy:
0.9973
Epoch 4/5
235/235 [==============================] - 193s 822ms/step - loss: 0.0100 - accuracy: 0.9974 - val_loss: 0.0229 - val_accuracy:
0.9913
Epoch 5/5
235/235 [==============================] - 195s 828ms/step - loss: 0.0104 - accuracy: 0.9966 - val_loss: 0.0377 - val_accuracy:
0.9863
```

ACCURACY:98.63

Inception

```
train = train_gen.flow_from_directory(train_path,target_size=(299,299),batch_size=64)
val = val_gen.flow_from_directory(validation_path,target_size=(299,299),batch_size=64)
```

```
Found 15000 images belonging to 15 classes.
Found 3000 images belonging to 15 classes.
```

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
tf.random.set_seed(1234)
```

```
inceptionV3 = InceptionV3(include_top=False,input_shape=(299 ,299,3))
for layer in inceptionV3.layers:
  print(layer)
```

```
for layer in inceptionV3.layers:
  layer.trainable=False
```

```
x = Flatten()(inceptionV3.output)
```

```
output = Dense(15,activation='softmax')(x)
```

```
inceptionV3 = Model(inceptionV3.input,output)
```

`include_top=False`: Excludes the fully connected layers to allow customization for specific tasks.

`input_shape=(299,299,3)`: Specifies the input image dimensions and color channels.

`layer.trainable=False`: Freezes the weights of the pre-trained layers to retain learned features.

`x = Flatten()(inceptionV3.output)`: Converts the output of the last convolutional layer to a 1D tensor.

`output = Dense(15, activation='softmax')(x)`: Adds a dense layer with 15 units and softmax activation for classification.

ACCURACY:98.37

Xception	``` train = train_gen.flow_from_directory(train_path,target_size=(299,299),batch_size=64) val = val_gen.flow_from_directory(validation_path,target_size=(299,299),batch_size=64) ``` ``` Found 15000 images belonging to 15 classes. Found 3000 images belonging to 15 classes. ``` ``` from tensorflow.keras.applications.xception import Xception tf.random.set_seed(1234) ``` ``` Xception1 = Xception(include_top=False,input_shape=(299,299,3)) ``` ``` Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5 83683744/83683744 [==============================] - 5s 0us/step ``` ``` for layer in Xception1.layers: layer.trainable=False ``` ``` x = Flatten()(Xception1.output) ``` ``` output = Dense(15,activation='softmax')(x) ``` ``` Xception1 = Model(Xception1.input,output) ``` **`include_top=False`**: Excludes the fully connected layers to allow customization for specific tasks. **`input_shape=(299,299,3)`** : Specifies the input image dimensions and color channels. **`layer.trainable=False`**: Freezes the weights of the pre-trained layers to retain learned features. **`x = Flatten()(Xception1.output)`** : Converts the output of the last convolutional layer to a 1D tensor. **`output = Dense(15, activation='softmax')(x)`** : Adds a dense layer with 15 units and softmax activation for classification. **ACCURACY: 99.00**

Final Model Selection Justification (2 Marks):

Final Model	Reasoning

Xception	**ACCURACY: 99.00** The Xception model is chosen for vegetable image classification because it uses depthwise separable convolutions to efficiently capture fine details, leading to the highest accuracy among all tested models.