Model Optimization and Tuning Phase Template

Date	28 November 2024
Team ID	739771
Project Title	Deep Fruit Veg: Automated Fruit And Veg Identification
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
Model 1: EfficientNetB3	1. **Learning Rate (lr)**: The learning rate is set to 0.001, which controls the step size during model optimization. It's an important parameter for convergence speed and stability. **2. Dropout Rate**: The dropout rate is set to 0.2. This helps regularize the model and reduce overfitting by randomly setting a fraction of input units to 0 during training. **3. Regularization (L1 and L2)**: L1 and L2 regularization strengths are set to 0.0006 and 0.0016, respectively, to penalize large weights and encourage simpler models.

```python
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adamax
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, MaxPooling2D, Flatten

img_shape=(img_size[0],img_size[1],3)
model_name='EfficientNetB3'
base_model=tf.keras.applications.EfficientNetB3(include_top=False,weights='imagenet',input_shape=img_shape)
base_model.trainable=True
x=base_model.output
x=BatchNormalization(axis=-1,momentum=0.99,epsilon=0.001)(x)
x=MaxPooling2D()(x)
# Flatten the output before the Dense layer
x = Flatten()(x)
# The l2 regularizer expects the regularization strength as a positional argument.
x=Dense(256,kernel_regularizer=regularizers.l2( 0.0016),activity_regularizer=regularizers.l1(0.0006),bias_regularizer=regularizers.l1(0.0006),activa
x=Dropout(rate=0.2,seed=123)(x)
output=Dense(class_count,activation='softmax')(x)
# Use a different variable name to avoid overwriting the Model class
model = Model(inputs=base_model.input,outputs=output) # Changed 'Model' to 'model'

# Define the learning rate
lr = 0.001
```

<table>
<tr>
<td>Model 2:

Adamax Optimizer</td>
<td>1. Learning Rate (lr): The learning rate for the Adamax optimizer is set to 0.001. It governs how much the model weights are adjusted with respect to the loss gradient during training.
2. Beta1 and Beta2: These are default values for Adamax's momentum terms, which are used to control the moving averages of past gradients. Beta1 is typically set to 0.9, and Beta2 to 0.999. These values are not explicitly set in your code but are key in controlling momentum.
<pre><code># Define the learning rate
lr = 0.001

model.compile(Adamax(learning_rate=lr),loss='categorical_crossentropy',metrics=['accuracy'])
</code></pre>
</td>
</tr>
</table>

Final Model Selection Justification (2 Marks):

Final Model	**Reasoning**
Adamax and EfficientNetB3	**Efficiency and Performance:** EfficientNetB3 has been shown to provide state-of-the-art performance for a variety of image classification tasks while being computationally efficient. It uses a compound scaling method that balances network depth, width, and resolution, which helps achieve better accuracy with fewer parameters compared to traditional architectures. This makes it a suitable choice for the project, especially with a relatively large dataset and a goal of achieving high accuracy in fruit and vegetable classification. **Optimizer Choice**: **Adamax**, a variant of the Adam optimizer, is chosen because it has demonstrated good performance in handling sparse gradients, which can be especially beneficial for image classification tasks. The learning rate of 0.001 strikes a good balance between convergence speed and stability during training. **But** The combination of **EfficientNetB3** with **Adamax**, along with proper regularization, provides a model that is scalable and generalizes well across unseen data, which is crucial for ensuring that the model performs well in real-world deployment scenarios (e.g., automated sorting in food processing plants or quality control in supermarkets).