Model Optimization and Tuning Phase Template

Date	12 July 2024
Team ID	SWTID1720157891
Project Title	Rice Classification using CNN
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	This code creates a standard CNN model using a predefined architecture. It includes three convolutional layers, each followed by a max-pooling layer, and ends with a flatten layer, a dense layer, and a dropout layer before the output layer. Additionally, it sets up ImageDataGenerator instances for data augmentation during training and rescaling during validation. The model is compiled and then trained using the data generators. Finally, the model is evaluated on the validation data, and the validation accuracy is printed. **Key Components:** • Predefined static model architecture.

<table>
<tr>
<td></td>
<td>

Image augmentation using ImageDataGenerator.
Model training and evaluation without hyperparameter tuning


```
from tensorflow.keras.layers import BatchNormalization

# Create ImageDataGenerator instances for training and validation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    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'
)

validation_datagen = ImageDataGenerator(rescale=1./255)



# Build the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_height, img_width, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(train_generator.class_indices), activation='softmax')
])

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


# Train the model
history = model.fit(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator
)
# Evaluate the model and print the accuracy
validation_loss, validation_accuracy = model.evaluate(validation_generator)
print(f'Validation Accuracy: {validation_accuracy * 100:.2f}%')
```


</td>
</tr>
<tr>
<td>Model 2</td>
<td>This code defines a function to build a Convolutional Neural Network (CNN) model with hyperparameters that can be tuned using KerasTuner. The model structure includes several convolutional layers, followed by max-pooling layers, and a fully connected dense layer before the output layer. The hyperparameters such as the number of filters, kernel sizes, number of convolutional layers, dense units, and dropout rate are</td>
</tr>
</table>

dynamically set using the KerasTuner's hp object. The RandomSearch tuner is used to find the best hyperparameters by maximizing validation accuracy.

Key Components:

- Hyperparameter tuning using KerasTuner.
- Dynamic model architecture.
- Tuner setup for searching optimal hyperparameters.
-

```
import kerastuner as kt

# Define a hypermodel function
def build_model1(hp):
    model = Sequential()
    model.add(Conv2D(
        filters=hp.Int('conv_1_filters', min_value=32, max_value=128, step=32),
        kernel_size=hp.Choice('conv_1_kernel', values=[3, 5]),
        activation='relu',
        input_shape=(img_height, img_width, 3)
    ))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))

    for i in range(hp.Int('num_conv_layers', 1, 3)):
        model.add(Conv2D(
            filters=hp.Int(f'conv_{i+2}_filters', min_value=32, max_value=128, step=32),
            kernel_size=hp.Choice(f'conv_{i+2}_kernel', values=[3, 5]),
            activation='relu'
        ))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(units=hp.Int('dense_units', min_value=32, max_value=128, step=32), activation='relu'))
    model.add(Dropout(rate=hp.Float('dropout_rate', min_value=0.0, max_value=0.5, step=0.1)))
    model.add(BatchNormalization())
    model.add(Dense(len(train_generator.class_indices), activation='softmax'))

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

    return model

# Define the tuner
tuner = kt.RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=2,
    executions_per_trial=1,
    directory='my_dir',
    project_name='rice_grain_classification'
)
```

	
…	…

Final Model Selection Justification (2 Marks):

Final Model	Reasoning

Model 2	**Hyperparameter Optimization:** This approach allows you to automatically search for the best set of hyperparameters, which can lead to better model performance. **Flexibility:** You can explore a wide range of model architectures and configurations without manually coding each one. **Automation:** KerasTuner handles the process of training multiple models with different hyperparameters, saving you time and effort.