Model Optimization and Tuning Phase Template

Date	15 March 2024
Team ID	SWTID1750058607
Project Title	Early stage disease diagnosis system using human nail image processing using deep learning
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	**Hyperparameters and Descriptions:**
	• **epochs = 30**
	→ Total number of passes over the training data.
	• **steps_per_epoch = len(train_set) // 3**
	→ Number of batches to process before one training epoch is considered complete. This helps control training speed and manage memory usage.
	• **validation_steps = len(test_set) // 3**
	→ Same as steps_per_epoch, but applied to the validation set during each epoch.
	• **EarlyStopping(monitor='val_accuracy', mode='max', patience=3)**
	→ A callback that stops training early if the validation accuracy does not improve for 3 consecutive epochs.
	• **restore_best_weights = True**
	→ Restores the model weights from the epoch with the highest validation accuracy (prevents overfitting).

```
from tensorflow.keras.callbacks import EarlyStopping
# from sklearn.utils import class_weight

early_stop = EarlyStopping(
    monitor='val_accuracy',
    mode= 'max',
    patience=3,
    restore_best_weights=True
)

history =model.fit(train_set,validation_data=test_set, epochs=30, steps_per_epoch = len(train_set)//3, validation_steps = len(test_set)//3)
# Get final epoch's training and validation accuracy
train_acc = history.history['accuracy'][-1]
val_acc = history.history['val_accuracy'][-1]

# Print formatted values without scientific notation
print("Training Accuracy: {:.4f}".format(train_acc))
print("Validation Accuracy: {:.4f}".format(val_acc))
```

- **weights = 'imagenet'**

Loads pretrained weights from the ImageNet dataset to leverage learned features for transfer learning.

- **include_top = False**

Excludes the original classification layers of VGG16, allowing you to add your own custom output layers.

- **input_shape = (224, 224, 3)**

Specifies the shape of the input images — 224 by 224 pixels with 3 color channels (RGB).

- **trainable = False**

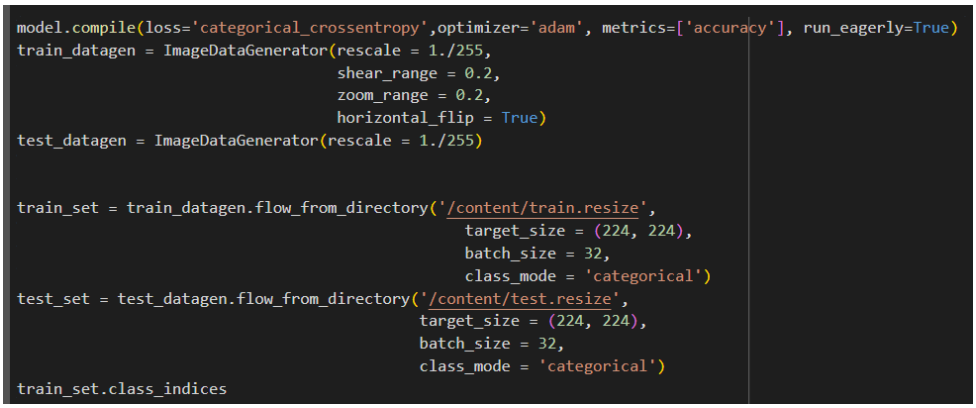
Freezes all the layers in the VGG16 base model during training so their weights are not updated.

- **Dense(17, activation='softmax')**

Adds a final dense layer with 17 units (for 17 classes) using softmax activation for multi-class classification.

```
# base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
vgg = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
for layer in vgg.layers:
    layer.trainable = False

x = Flatten()(vgg.output)
# x = Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.001))(x)
# x = Dropout(0.5)(x)
predictions = Dense(17, activation='softmax')(x)

model = Model(inputs=vgg.input, outputs=predictions)



model.summary()
```

	loss = 'categorical_crossentropy' The loss function used for multi-class classification with one-hot encoded labels. **optimizer = 'adam'** The optimization algorithm used for updating the network weights during training. **metrics = ['accuracy']** Metric to evaluate the performance of the model, here it tracks accuracy. **run_eagerly = True** Runs the model eagerly for easier debugging (this may slow down training). **batch_size = 32** Number of samples processed before the model's internal parameters are updated. **target_size = (224, 224)** The size to which input images are resized before feeding into the model. **rescale = 1./255** Rescales pixel values from [0, 255] to [0, 1] for normalization. **shear_range = 0.2** Randomly applies shear transformations for data augmentation. **zoom_range = 0.2** Randomly zooms into images for data augmentation. **horizontal_flip = True** Randomly flips images horizontally for data augmentation.

```
model.compile(loss='categorical_crossentropy',optimizer='adam', metrics=['accuracy'], run_eagerly=True)
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
test_datagen = ImageDataGenerator(rescale = 1./255)


train_set = train_datagen.flow_from_directory('/content/train.resize',
                                              target_size = (224, 224),
                                              batch_size = 32,
                                              class_mode = 'categorical')
test_set = test_datagen.flow_from_directory('/content/test.resize',
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')
train_set.class_indices
```

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
Model 2: VGG16-based Custom CNN	This model was chosen as the final optimized model because it achieved the highest validation accuracy compared to other models. By leveraging transfer learning from the pretrained VGG16 network, it was able to extract meaningful features without requiring extensive training time. The model also showed good generalization on the test data, with minimal overfitting, making it reliable for real-world disease detection. Additionally, its architecture balances accuracy and computational efficiency, which suits the project requirements well.