SMARTBRIDGE
Let's Bridge the Gap
a Veranda Enterprise

Smart Internz

Model Optimization and Tuning Phase Template

Date	15 july 2024
Team ID	SWTID1720093035
Project Title	**TechPart Vision: Personal Computer Parts Image Classification Using EfficientNet Transfer 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
CNN	**Convolutional Layers** • **Number of Filters**: 32 • **Kernel Size**: 3 (first layer), 2 (second layer) • **Activation Function**: ReLU • **Padding**: Same **Pooling Layers** • **Pooling Size**: 2 **Regularization**

- **Dropout Rate**: 0.5

Dense Layers

- **Units**: 128
- **Kernel Initializer**: HeNormal

Output Layer

- **Units**: 14

Training Parameters

- **Batch Size**: 32
- **Number of Epochs**: 10

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten
from keras.layers import Convolution2D, MaxPooling2D
import tensorflow as tf
```

```
initializer = tf.keras.initializers.HeNormal()
```

```
model=Sequential()
model.add(Convolution2D(filters=32, kernel_size=3, padding='same', activation="relu",
                        input_shape=(255, 255, 3)))
model.add(MaxPooling2D(strides=2, pool_size=2, padding= "valid"))
model.add(Convolution2D(filters=32, kernel_size=2, padding='same', activation="relu"))
model.add(MaxPooling2D(strides=2, pool_size=2, padding="valid"))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_initializer=initializer))
model.add(Dropout(0.5))
model.add(Dense(14, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model.fit(train_gen, validation_data= val_gen, batch_size= 32, epochs = 10, verbose = 1)
```

EfficientNetV2 B1 Model Initialisation	Tuned Hyperparameters: Base Model: EfficientNetV2B1 **Base Model** • **Base Model**: EfficientNetV2B1 (pretrained on ImageNet, used as a feature extractor) **Pooling** • **Pooling**: Max (global max pooling applied to the output of the base model) **Batch Normalization** • **Axis**: -1 • **Momentum**: 0.99 • **Epsilon**: 0.001 **Dense Layer** • **Units**: 256 **Regularization** • **Kernel Regularization**: l2(0.016) • **Activity Regularization**: l1(0.006) • **Bias Regularization**: l1(0.006) **Activation** • **Activation Function**: ReLU

	Dropout
	
	• **Rate**: 0.4
	
	Output Layer
	
	• **Units**: 14 (softmax activation for multi-class classification)
	
	Optimizer
	
	• **Optimizer**: Adamax
	
	• **Learning Rate**: 0.001
	
	Callbacks
	
	• **ReduceLROnPlateau**:
	    ○ **Monitor**: "val_loss"
	    ○ **Factor**: 0.4
	    ○ **Patience**: 2
	    ○ **Min LR**: 0.0
	
	• **EarlyStopping**:
	    ○ **Monitor**: "val_loss"
	    ○ **Patience**: 2
	    ○ **Restore Best Weights**: True
	
	Training Parameters
	
	• **Number of Epochs**: 5

```python
base_model=tf.keras.applications.EfficientNetV2B1(include_top=False, weights="imagenet",input_shape=(255,255,3), pooling='max')
print('Created EfficientNetV2 B1 model')
```

```
ownloading data from https://storage.googleapis.com/tensorflow/keras-applications/efficientnet_v2/efficientnetv2-b1_notop.h5
8456008/28456008 [==============================] - 2s 0us/step
reated EfficientNetV2 B1 model
```

```python
base_model.trainable=True
x=base_model.output
x=BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)
x = Dense(256, kernel_regularizer = regularizers.l2(l = 0.016),activity_regularizer=regularizers.l1(0.006),
                bias_regularizer=regularizers.l1(0.006) ,activation='relu')(x)
x=Dropout(rate=.4, seed=123)(x)
output=Dense(14, activation='softmax')(x)
model=Model(inputs=base_model.input, outputs=output)
model.compile(Adamax(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```python
rlronp=keras.callbacks.ReduceLROnPlateau( monitor="val_loss", factor=0.4, patience=2, verbose=1, mode="auto",  min_delta=0.00001, cooldown=0, min_lr=0.0
estop=keras.callbacks.EarlyStopping( monitor="val_loss", min_delta=0,  patience=2, verbose=1, mode="auto", baseline=None, restore_best_weights=True)
callbacks=[rlronp, estop]
```

```python
history=model.fit(x=train_gen,   epochs=5, verbose=1, callbacks=callbacks,  validation_data=val_gen,
                    validation_steps=None,  shuffle=True)
```

Vgg19

Hyperparameter Tuning:

Base Model

- **Base Model**: VGG19 (pretrained on ImageNet, used as a feature extractor)
- **Input Shape**: (224, 224, 3)

Pooling

- **Pooling**: Global Average Pooling

Dense Layers

- **Units**: 1024
- **Activation Function**: ReLU

Output Layer

- **Units**: len(train_gen.class_indices)

- **Activation Function**: Softmax

Optimizer

- **Optimizer**: Adam
- **Learning Rate**: 0.001

Loss Function

- **Loss Function**: Categorical Crossentropy

Metrics

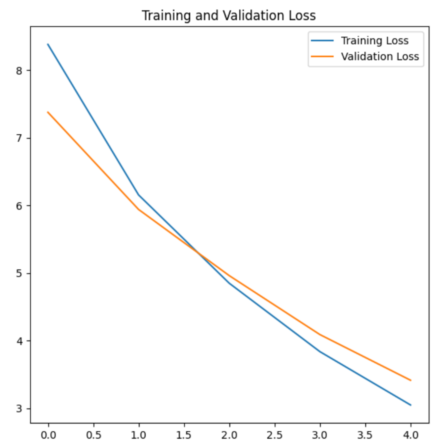
- **Metrics**: Accuracy

Freezing Layers

- **Freezing Layers**: Initially freeze all base model layers

```python
from tensorflow.keras.applications import VGG19
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(len(train_gen.class_indices), activation='softmax')(x)

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

for layer in base_model.layers:
    layer.trainable = False

model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

Final Model Selection Justification (2 Marks):

Final Model	**Reasoning**
EfficientNetV2B1 Model Initialisation	**EfficientNetV2B1:**  1. **Training Accuracy:** ○ The training accuracy starts around 0.55 and increases rapidly, reaching around 0.95 by epoch 4. 2. **Validation Accuracy:** ○ The validation accuracy starts around 0.70 and remains relatively stable, fluctuating slightly but generally around 0.75 by epoch 4. 3. **Training Loss:** ○ The training loss decreases significantly from around 8 to below 3.

4. **Validation Loss:**

    - The validation loss also decreases from around 7 to slightly above 3.

VGG19:

1. **Training Accuracy:**

    - The training accuracy starts around 0.4 and increases steadily, reaching around 0.9 by epoch 17.5.

2. **Validation Accuracy:**

    - The validation accuracy starts around 0.4 and shows fluctuations, peaking around 0.7 but generally staying lower than the training accuracy.

3. **Training Loss:**

    - The training loss decreases significantly from around 2 to nearly 0.2.

4. **Validation Loss:**

	◦ The validation loss decreases initially from around 1.8 to around 1.2 but then fluctuates and increases slightly. **Comparison:** • **Training Accuracy:** ◦ EfficientNetV2B1 reaches higher training accuracy more quickly than VGG19. • **Validation Accuracy:** ◦ EfficientNetV2B1 maintains a relatively stable validation accuracy around 0.75, while VGG19's validation accuracy fluctuates and generally stays lower. • **Training Loss:** ◦ Both models show a significant decrease in training loss, but EfficientNetV2B1 starts with a higher value and decreases more sharply initially. • **Validation Loss:** ◦ EfficientNetV2B1 has a more consistent decrease in validation loss, whereas VGG19's validation loss fluctuates after the initial decrease. **Conclusion:** EfficientNetV2B1 seems to perform better overall, with higher and more stable validation accuracy and a consistent decrease in validation loss compared to VGG19. While both models improve over time,

EfficientNetV2B1 shows quicker and more consistent improvements in the metrics provided.

```
#### EfficientNetV2B1 Model Initialisation

base_model=tf.keras.applications.EfficientNetV2B1(include_top=False, weights="imagenet",input_shape=(255,255,3), pooling='max')
print('Created EfficientNetV2 B1 model')

ownloading data from https://storage.googleapis.com/tensorflow/keras-applications/efficientnet_v2/efficientnetv2-b1_notop.h5
8456008/28456008 [==============================] - 2s 0us/step
reated EfficientNetV2 B1 model

base_model.trainable=True
x=base_model.output
x=BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)
x = Dense(256, kernel_regularizer = regularizers.l2(l = 0.016),activity_regularizer=regularizers.l1(0.006),
                    bias_regularizer=regularizers.l1(0.006) ,activation='relu')(x)
x=Dropout(rate=.4, seed=123)(x)
output=Dense(14, activation='softmax')(x)
model=Model(inputs=base_model.input, outputs=output)
model.compile(Adamax(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
rlronp=keras.callbacks.ReduceLROnPlateau( monitor="val_loss", factor=0.4, patience=2, verbose=1, mode="auto",  min_delta=0.00001, cooldown=0, min_lr=0.0
estop=keras.callbacks.EarlyStopping( monitor="val_loss", min_delta=0,  patience=2, verbose=1, mode="auto", baseline=None, restore_best_weights=True)
callbacks=[rlronp, estop]
```

Model fitting

```
history=model.fit(x=train_gen,   epochs=5, verbose=1, callbacks=callbacks,  validation_data=val_gen,
                    validation_steps=None,  shuffle=True)

Epoch 1/5
100/100 [==============================] - 94s 423ms/step - loss: 8.3989 - accuracy: 0.5595 - val_loss: 7.3195 - val_accuracy: 0.7043 - lr: 0.0010
Epoch 2/5
100/100 [==============================] - 33s 330ms/step - loss: 6.1665 - accuracy: 0.8233 - val_loss: 5.9759 - val_accuracy: 0.7134 - lr: 0.0010
Epoch 3/5
100/100 [==============================] - 33s 334ms/step - loss: 4.8552 - accuracy: 0.9032 - val_loss: 4.9639 - val_accuracy: 0.7317 - lr: 0.0010
Epoch 4/5
100/100 [==============================] - 34s 335ms/step - loss: 3.8550 - accuracy: 0.9258 - val_loss: 4.1046 - val_accuracy: 0.7256 - lr: 0.0010
Epoch 5/5
100/100 [==============================] - 34s 337ms/step - loss: 3.0294 - accuracy: 0.9486 - val_loss: 3.4238 - val_accuracy: 0.7195 - lr: 0.0010
```