Model Optimization and Tuning Phase Template

Date	15 April 2024
Team ID	738181
Project Title	CRIME VISION: ADVANCED CRIME CLASSIFICATION WITH 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 hyper parameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Hyperparameter Tuning Documentation (8 Marks):

Model	Tuned Hyperparameters
DenseNet 121	Batch size = 250, `batch_size = 250,` Drop out = (0.2,0.2), Activation function = (relu, relu), Number of Layers in CNN = 4 Layers (GlobalAveragePooling, (256, 512, 1024)),

	`` def create_model(): model=Sequential() base_model=transfer_learning() model.add(base_model) model.add(GlobalAveragePooling2D()) model.add(Dense(256, activation="relu")) model.add(Dropout(0.2)) model.add(Dense(512, activation="relu")) model.add(Dropout(0.2)) model.add(Dense(1024, activation="relu")) model.add(Dense(14, activation="softmax")) model.summary() return model ``` Optimizer = Adam, ``` model.compile(optimizer="adam",loss='categorical_crossentropy',metrics=['accuracy']) ``` Filter shape = (224,224)
Xception	Batch size = 250, `batch_size = 250,` Drop out = (0.2,0.2), Activation function = (relu, relu), Number of Layers in CNN = 4 Layers (GlobalAveragePooling, (256, 512, 1024)),

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

```
def create_model_2():
    model = Sequential()
    xception=transfer_learning()
    model.add(xception)
    model.add(GlobalAveragePooling2D())
    model.add(Dense(256,activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(512,activation = 'relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1024,activation = 'relu'))
    x = Flatten()(xception.output)
    output = Dense(14, activation ='softmax')(x)
    xception = Model(xception.input, output)
    xception.summary()
    return xception
```

Optimizer = Adam,

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

Filter shape = (299,299)
</td>
</tr>
<tr>
<td>VGG16</td>
<td>

Batch size = 250, `batch_size = 250,`

```
def create_model():
  model=Sequential()
  vgg=transfer_learning()
  model.add(vgg)
  model.add(GlobalAveragePooling2D())
  x = Flatten()(vgg.output)
  output = Dense(14, activation ='softmax')(x)
  vgg16 = Model(vgg.input, output)
  vgg16.summary()
  return vgg16
```

Optimizer = Adam,

```
model.compile(optimizer="adam",loss='categorical_crossentropy',metrics=['accuracy'])
```
</td>
</tr>
</table>

<table>
<tr><td></td><td>Filter shape = (224,224)</td></tr>
<tr><td>ResNet50</td><td>Batch size = 250, <code>batch_size = 250,</code>

Drop out = (0.2,0.2),

Activation function = (relu, relu),

Number of Layers in CNN = = 4 Layers (GlobalAveragePooling,

(256, 512, 1024)),

<pre>def create_model_2():
    model = Sequential()
    resnet=transfer_learning()
    model.add(resnet)
    model.add(GlobalAveragePooling2D())
    model.add(Dense(256,activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(512,activation = 'relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1024,activation = 'relu'))
    x = Flatten()(resnet.output)
    output = Dense(14, activation ='softmax')(x)
    resnet50 = Model(resnet.input, output)
    resnet50.summary()
    return resnet50</pre>

Optimizer = Adam,

<pre>2 model.compile(optimizer="adam",loss='categorical_crossentropy',metrics=['accuracy'])</pre>

Filter shape = (224,224)</td></tr>
</table>

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
DenseNet121	• Efficient Parameter Usage: Here each layer receives feature maps from all preceding layers, facilitating feature reuse and reducing the number of parameters. This leads to more efficient training and inference compared to traditional CNN architectures. • Feature Propagation: DenseNet's dense connectivity pattern enables direct connections between layers at different depths in the network. This facilitates feature propagation and gradient flow, allowing information to be passed more effectively through the network. • Pre-Trained Weights: DenseNet121 comes with pre-trained weights on large-scale image datasets such as ImageNet. Leveraging pre-trained weights allows the model to capture rich hierarchical features from a diverse range of images.

	• Proven Performance: DenseNet architectures have been widely used and validated across various computer vision tasks, including image classification, object detection, and segmentation.