Model Optimization and Tuning Phase

Date	14th July 2024
Team ID	SWTID1720195938
Project Title	**CovidVision: Advanced COVID-19 Detection from Lung X-Rays 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 hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Hyperparameter Tuning Documentation (8 Marks):

Model	Tuned Hyperparameters
VGG16	1. Learning Rate (0.0001): This is the step size that the optimizer takes while performing gradient descent during training. A smaller learning rate means the model learns slowly and needs more epochs to train, while a larger learning rate makes the model learn faster but it might overshoot the optimal point. 2. Loss Function (Binary Crossentropy): This is the function that the model tries to minimize during training. It measures how well the model's predictions match the true values. Binary crossentropy is used for binary classification problems. 3. Optimizer (Adam): This is the algorithm used to update the weights of the model based on the gradients of the loss function. Adam is a

popular optimizer that adapts the learning rate for each weight individually.

4. Batch Size (32): This is the number of training examples used in one iteration of model training. Using a larger batch size means the model makes fewer updates to its weights, but those updates are based on a larger amount of data.
5. Number of Epochs (30): This is the number of times the entire training dataset is passed forward and backward through the neural network.
6. Validation Split (0.3): This is the fraction of the data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch.
7. Early Stopping Parameters: Early stopping is a form of regularization used to avoid overfitting when training a learner with an iterative method. Here, it monitors the validation loss ('val_loss'), with the mode set to 'min', meaning training will stop when the validation loss stops decreasing. Patience is set to 4, meaning training will stop if the validation loss doesn't improve for 4 epochs.
8. Dropout Rate (0.2): This is the fraction of the inputs to drop out during training, which is a regularization technique to prevent overfitting.
9. Number of Filters in Convolution Layers (8 and 16): These are the number of filters used in the convolutional layers. Each filter extracts different features from the input data.
10. Kernel Size ((3,3)): This is the size of the filters in the convolutional layers.
11. Pool Size ((2,2)): This is the size of the pooling window in the max pooling layers.
12. Activation Functions (ReLU and Sigmoid): These determine the output of a neuron given an input or set of inputs. ReLU (Rectified Linear Unit) is used in the hidden layers and Sigmoid is used in the output layer for binary classification.
13. Number of Neurons in Dense Layer (32**)**: This is the number of neurons in the dense layer, which directly influences the capacity/complexity of the model

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
img_height, img_width= IMAGE_SIZE
batch_size=32
train_datagen = ImageDataGenerator(validation_split=0.3)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='training')

validation_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='validation')
```

```
data_augmentation = keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomRotation(
        factor=(-0.2, 0.3),
        fill_mode='reflect',
        interpolation='bilinear',
        seed=None
    ),
    tf.keras.layers.experimental.preprocessing.Rescaling(
        scale=1/.255,
        offset=0.0
    ),
])
```

```
VGG16_model=tf.keras.Sequential([
    data_augmentation,
    tf.keras.layers.Conv2D(8, (3,3), activation='relu', input_shape=IMAGE_SHAPE),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding="valid"),
    tf.keras.layers.Conv2D(16, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding="valid"),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(32),
    tf.keras.layers.Dropout(.2, input_shape=(32,)),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
VGG16_model.compile(optimizer=Adam(learning_rate=0.0001),
                    loss='binary_crossentropy',
                    metrics = ['accuracy'])
```

```
es = tf.keras.callbacks.EarlyStopping(monitor = 'val_loss', mode = 'min', verbose = 2, patience = 4)

trainer=VGG16_model.fit(train_generator,validation_data=validation_generator,epochs=30, callbacks = [es])
```

CNN Model

1. Learning Rate (0.0001): This is the step size that the optimizer takes while performing gradient descent during training. A smaller learning rate means the model learns slowly and needs more epochs to train, while a larger learning rate makes the model learn faster but it might overshoot the optimal point.
2. Loss Function (Binary Crossentropy): This is the function that the model tries to minimize during training. It measures how well the model's predictions match the true values. Binary crossentropy is used for binary classification problems.
3. Optimizer (Adam): This is the algorithm used to update the weights of the model based on the gradients of the loss function. Adam is a popular optimizer that adapts the learning rate for each weight individually.

4. Batch Size (32): This is the number of training examples used in one iteration of model training. Using a larger batch size means the model makes fewer updates to its weights, but those updates are based on a larger amount of data.
5. Number of Epochs (30): This is the number of times the entire training dataset is passed forward and backward through the neural network.
6. Validation Split (0.3): This is the fraction of the data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch.
7. Early Stopping Parameters: Early stopping is a form of regularization used to avoid overfitting when training a learner with an iterative method. Here, it monitors the validation loss ('val_loss'), with the mode set to 'min', meaning training will stop when the validation loss stops decreasing. Patience is set to 4, meaning training will stop if the validation loss doesn't improve for 4 epochs.
8. Dropout Rate (0.2): This is the fraction of the inputs to drop out during training, which is a regularization technique to prevent overfitting.
9. Number of Filters in Convolution Layers (8 and 16): These are the number of filters used in the convolutional layers. Each filter extracts different features from the input data.
10. Kernel Size ((3,3)): This is the size of the filters in the convolutional layers.
11. Pool Size ((2,2)): This is the size of the pooling window in the max pooling layers.
12. Activation Functions (ReLU and Sigmoid): These determine the output of a neuron given an input or set of inputs. ReLU (Rectified Linear Unit) is used in the hidden layers and Sigmoid is used in the output layer for binary classification.
13. Number of Neurons in Dense Layer (32): This is the number of neurons in the dense layer, which directly influences the capacity/complexity of the model.

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

```
data_augmentation = keras.Sequential([
  tf.keras.layers.experimental.preprocessing.RandomRotation(
        factor=(-0.2, 0.3),
        fill_mode='reflect',
        interpolation='bilinear',
        seed=None
    ),
    tf.keras.layers.experimental.preprocessing.Rescaling(
        scale=1/.255,
        offset=0.0
    ),
])
```

```
CNN_model=tf.keras.Sequential([
    data_augmentation,
    tf.keras.layers.Conv2D(8, (3,3), activation='relu', input_shape=IMAGE_SHAPE),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding="valid"),
    tf.keras.layers.Conv2D(16, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding="valid"),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(32),
    tf.keras.layers.Dropout(.2, input_shape=(32,)),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
CNN_model.compile(optimizer=Adam(learning_rate=0.0001),
                  loss='binary_crossentropy',
                  metrics = ['accuracy'])
```

```
img_height, img_width= IMAGE_SIZE
batch_size=32
train_datagen = ImageDataGenerator(validation_split=0.3)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='training')
validation_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='validation')
```

```
es = tf.keras.callbacks.EarlyStopping(monitor = 'val_loss', mode = 'min', verbose = 2, patience = 4)

trainer=CNN_model.fit(train_generator,validation_data=validation_generator,epochs=30, callbacks = [es])
```

</td>
</tr>
<tr>
<td>RESNET 50</td>
<td>

1. Learning Rate (0.0001): This is the step size that the optimizer takes while performing gradient descent during training. A smaller learning rate means the model learns slowly and needs more epochs to train, while a larger learning rate makes the model learn faster but it might overshoot the optimal point.
2. Loss Function (Binary Crossentropy): This is the function that the model tries to minimize during training. It measures how well the model's predictions match the true values. Binary crossentropy is used for binary classification problems.
3. Optimizer (Adam): This is the algorithm used to update the weights of the model based on the gradients of the loss function. Adam is a popular optimizer that adapts the learning rate for each weight individually.
4. Batch Size (32): This is the number of training examples used in one iteration of model training. Using a larger batch size means the

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

model makes fewer updates to its weights, but those updates are based on a larger amount of data.

5. Number of Epochs (30): This is the number of times the entire training dataset is passed forward and backward through the neural network.
6. Validation Split (0.3): This is the fraction of the data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch.
7. Early Stopping Parameters: Early stopping is a form of regularization used to avoid overfitting when training a learner with an iterative method. Here, it monitors the validation loss ('val_loss'), with the mode set to 'min', meaning training will stop when the validation loss stops decreasing. Patience is set to 4, meaning training will stop if the validation loss doesn't improve for 4 epochs.
8. Number of Neurons in Dense Layer (512**)**: This is the number of neurons in the dense layer, which directly influences the capacity/complexity of the model.
9. Activation Functions (ReLU and Sigmoid): These determine the output of a neuron given an input or set of inputs. ReLU (Rectified Linear Unit) is used in the hidden layers and Sigmoid is used in the output layer for binary classification.
10. Trainable Parameters in Pretrained Model (False**)**: This indicates whether the weights of the pretrained ResNet50 model are updated during training. If False, the weights are frozen and not updated, which is useful for transfer learning.
11. ImageDataGenerator Parameters: These are the parameters for the data augmentation process. They include 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'). These parameters control how the input images are randomly transformed to augment the training dataset and reduce overfitting.

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

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
img_height, img_width= IMAGE_SIZE
batch_size=32
train_datagen = ImageDataGenerator(validation_split=0.3)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='training')

validation_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='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_split=0.3)
```

```
resnet50_model= Sequential()

pre_model= tf.keras.applications.ResNet50(
    include_top=False,
    input_shape=(256,256,3),
    pooling='avg',
    classes=5,
    weights='imagenet'
)
for layer in pre_model.layers:
  layer.trainable= False

resnet50_model.add(pre_model)
resnet50_model.add(Flatten())
resnet50_model.add(Dense(512, activation='relu'))
resnet50_model.add(Dense(1, activation='sigmoid'))
```

```
resnet50_model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='binary_crossentropy',
              metrics = ['accuracy'])
```

```
es = tf.keras.callbacks.EarlyStopping(monitor = 'val_loss', mode = 'min', verbose = 2, patience = 4)

trainer=resnet50_model.fit(train_generator,validation_data=validation_generator,epochs=30, callbacks = [es])
```

</td>
</tr>
<tr>
<td>Xception</td>
<td>

1. Learning Rate (0.0001): This is the step size that the optimizer takes while performing gradient descent during training. A smaller learning rate means the model learns slowly and needs more epochs to train, while a larger learning rate makes the model learn faster but it might overshoot the optimal point.
2. Loss Function (Binary Crossentropy): This is the function that the model tries to minimize during training. It measures how well the model's predictions match the true values. Binary crossentropy is used for binary classification problems.
3. Optimizer (Adam): This is the algorithm used to update the weights of the model based on the gradients of the loss function. Adam is a popular optimizer that adapts the learning rate for each weight individually.
4. Batch Size (32): This is the number of training examples used in one iteration of model training. Using a larger batch size means the model makes fewer updates to its weights, but those updates are based on a larger amount of data.

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

5. Number of Epochs (30): This is the number of times the entire training dataset is passed forward and backward through the neural network.
6. Validation Split (0.3): This is the fraction of the data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch.
7. Early Stopping Parameters: Early stopping is a form of regularization used to avoid overfitting when training a learner with an iterative method. Here, it monitors the validation loss ('val_loss'), with the mode set to 'min', meaning training will stop when the validation loss stops decreasing. Patience is set to 4, meaning training will stop if the validation loss doesn't improve for 4 epochs.
8. Number of Neurons in Dense Layer (512): This is the number of neurons in the dense layer, which directly influences the capacity/complexity of the model.
9. Activation Functions (ReLU and Sigmoid): These determine the output of a neuron given an input or set of inputs. ReLU (Rectified Linear Unit) is used in the hidden layers and Sigmoid is used in the output layer for binary classification.
10. Trainable Parameters in Pretrained Model (False): This indicates whether the weights of the pretrained Xception model are updated during training. If False, the weights are frozen and not updated, which is useful for transfer learning.

```
img_height, img_width=IMAGE_SIZE
batch_size=16
train_datagen = ImageDataGenerator(validation_split=0.3)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='training')
validation_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='training')
```

```
xception_model = Sequential()
pre_model = tf.keras.applications.Xception(
    include_top=False,
    input_shape=(299, 299, 3),
    pooling='avg',
    weights='imagenet'
)
for layer in pre_model.layers:
    layer.trainable = False
xception_model.add(pre_model)
xception_model.add(Flatten())
xception_model.add(Dense(512, activation='relu'))
xception_model.add(Dense(1, activation='sigmoid'))
```

```
xception_model.compile(optimizer=Adam(learning_rate=0.0001),
                       loss='binary_crossentropy',
                       metrics = ['accuracy'])
```

```
[ ]  es = tf.keras.callbacks.EarlyStopping(monitor = 'val_loss', mode = 'min', verbose = 2, patience = 4)

     trainer=xception_model.fit(train_generator,validation_data=validation_generator,epochs=30)
```

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
Xception	Xception was chosen as the final optimized model for its efficient architecture, well-suited for image classification tasks like binary classification. Leveraging transfer learning with pre-trained weights from ImageNet, the model benefits from learned features and accelerated convergence, especially with frozen layers for specific task adaptation. Utilizing ReLU activation in hidden layers and sigmoid in the output layer ensures efficient gradient flow and probabilistic output suitable for binary decisions. The Adam optimizer optimizes learning rates individually, enhancing training efficiency, while early stopping with validation loss monitoring prevents overfitting. Incorporating robust data augmentation via ImageDataGenerator further improves model generalization by simulating diverse training scenarios. Overall, Xception balances model complexity with computational efficiency, making it a strong choice for achieving high accuracy and deployment readiness in your binary classification application.