[ ]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

[ ]:
```
import keras
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
import random
from sklearn.metrics import roc_curve, auc, confusion_matrix
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras import mixed_precision
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.callbacks import ReduceLROnPlateau, TensorBoard
```

[ ]:
```
img_size = (224, 224)
batch_size = 18

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    brightness_range=[0.5, 1.5],
    channel_shift_range=20,
)

train_dir = "/kaggle/input/feb-25-702010-trvaltest/3.tr70val20te10/train"
val_dir = "/kaggle/input/feb-25-702010-trvaltest/3.tr70val20te10/val"
test_dir = "/kaggle/input/feb-25-702010-trvaltest/3.tr70val20te10/test"

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True,
)

val_datagen = ImageDataGenerator(rescale=1./255)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    shuffle=False
)

test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    shuffle=False
)
```

[ ]:
```
base_model = EfficientNetB0(include_top=False, weights=None, input_shape=(224, 224, 3))
```

```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(1, activation='sigmoid')(x)

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

[ ]:
```
for layer in base_model.layers:
    layer.trainable = True
```

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

[ ]:
```
callbacks1 = [
    ModelCheckpoint(filepath='model_weights.h5', save_best_only=True),
    EarlyStopping(patience=5,monitor='val_loss', restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=0.001, verbose=1)]
```

[ ]:
```
history = model.fit(
    train_generator,
    epochs=20,
    validation_data=val_generator,
    callbacks=callbacks1
)
```

[ ]:
```

```

[ ]:
```
model.save('/kaggle/working/EffNetB0_702010_no_weights.h5')
```

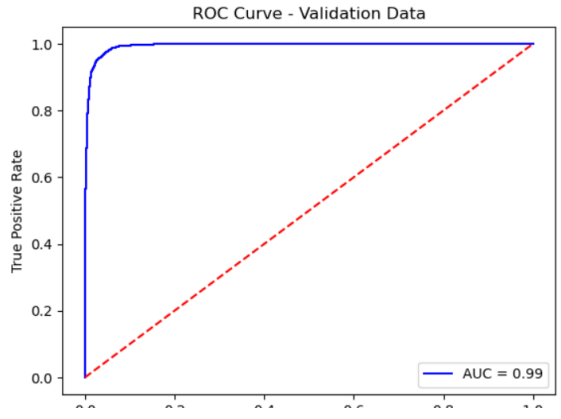
[ ]:
```

```

[35]:
```
# Evaluate on validation data for model
val_loss, val_acc = model.evaluate(val_generator)
print('Validation loss:', val_loss)
print('Validation accuracy:', val_acc)
```

```
331/331 [==============================] - 18s 55ms/step - loss: 0.0962 - accuracy: 0.9620
Validation loss: 0.0962304025888443
Validation accuracy: 0.9620295763015747
```

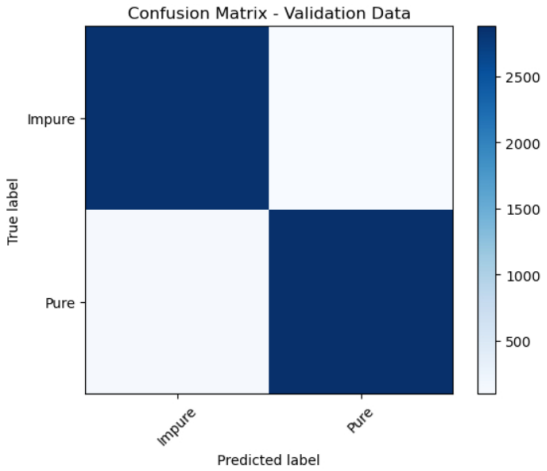
[36]:
```
# Generate ROC curve for validation data
val_pred = model.predict(val_generator)
val_fpr, val_tpr, val_thresholds = roc_curve(val_generator.classes, val_pred)
val_roc_auc = auc(val_fpr, val_tpr)

plt.plot(val_fpr, val_tpr, 'b', label='AUC = %0.2f' % val_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.title('ROC Curve - Validation Data')
plt.legend(loc='lower right')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
331/331 [==============================] - 18s 56ms/step
```



[37]:

```
# Generate confusion matrix for validation data
val_pred_classes = np.round(val_pred)
val_cm = confusion_matrix(val_generator.classes, val_pred_classes)
plt.imshow(val_cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix - Validation Data')
plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, ['Impure', 'Pure'], rotation=45)
plt.yticks(tick_marks, ['Impure', 'Pure'])
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```




```
print(val_cm)
```

```
[[2850  101]
 [ 125 2876]]
```

+ Code + Markdown

[ ]:

[38]:

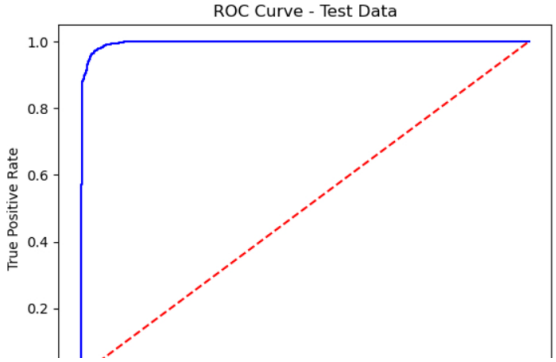
```
# Evaluate on test data for model
test_loss, test_acc = model.evaluate(test_generator)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

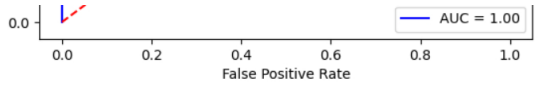
```
166/166 [==============================] - 10s 57ms/step - loss: 0.0833 - accuracy: 0.9687
Test loss: 0.08326282352209091
Test accuracy: 0.9687395095825195
```

[40]:

```
# Generate ROC curve for test data for model
test_pred = model.predict(test_generator)
test_fpr, test_tpr, test_thresholds = roc_curve(test_generator.classes, test_pred)
test_roc_auc = auc(test_fpr, test_tpr)

plt.plot(test_fpr, test_tpr, 'b', label='AUC = %0.2f' % test_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.title('ROC Curve - Test Data')
plt.legend(loc='lower right')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

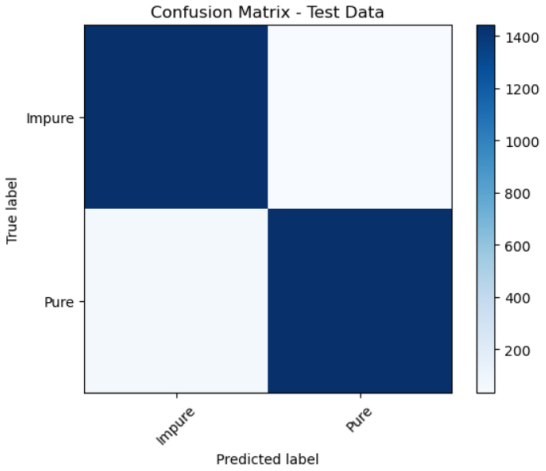
```
166/166 [==============================] - 9s 52ms/step
```





[41]:

```
# Generate confusion matrix for test data
test_pred_classes = np.round(test_pred)
test_cm = confusion_matrix(test_generator.classes, test_pred_classes)
plt.imshow(test_cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix - Test Data')
plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, ['Impure', 'Pure'], rotation=45)
plt.yticks(tick_marks, ['Impure', 'Pure'])
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



[42]:

```
print(test_cm)
```

```
[[1441   34]
 [  59 1441]]
```

[ ]: