

```

# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'real-life-industrial-dataset-of-casting-product:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F487456%2

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
    os.symlink(KAGGLE_INPUT_PATH, os.path.join(".", 'input'), target_is_directory=True)
except FileExistsError:
    pass
try:
    os.symlink(KAGGLE_WORKING_PATH, os.path.join(".", 'working'), target_is_directory=True)
except FileExistsError:
    pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}]{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
                with ZipFile(tfile) as zfile:
                    zfile.extractall(destination_path)
            else:
                with tarfile.open(tfile.name) as tarfile:
                    tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')

```

```

➡ Downloading real-life-industrial-dataset-of-casting-product, 104965698 bytes compressed
[=====] 104965698 bytes downloaded
Downloaded and uncompressed: real-life-industrial-dataset-of-casting-product
Downloading our-uploaded-dataset-name, 87541466 bytes compressed
[=====] 87541466 bytes downloaded
Downloaded and uncompressed: our-uploaded-dataset-name
Downloading your-uploaded-dataset, 217513846 bytes compressed
[=====] 217513846 bytes downloaded

```

Downloaded and uncompressed: your-uploaded-dataset
Data source import complete.

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 and Run All"
You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session

```

/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_6449.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_9202.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_3304.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_6922.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_9990.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_4081.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_3214.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_7240.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_1102.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_6390.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_7072.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_6119.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_2992.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_5077.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_3371.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_9397.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_2235.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_9732.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_2394.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_6149.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_9850.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_93.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_4168.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_5370.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_2816.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_3870.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_9136.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_1780.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_4038.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_4405.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_645.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_563.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_9389.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_5588.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_5045.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_6239.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_5310.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_5548.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_1450.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_8127.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_9341.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_7939.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_2999.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_6189.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_1445.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_9818.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_8460.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_9235.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_8762.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_3868.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_3984.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_1715.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_5818.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_8833.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_1644.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_5814.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_9516.jpeg
/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_512x512/casting_512x512/ok_front/cast_ok_0_6424.jpeg

```

```
import os
import cv2
import time
import random
import numpy as np
from PIL import Image
```

```
import matplotlib.pyplot as plt
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.metrics import AUC, Precision, Recall
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
from tensorflow.keras.applications import Xception, VGG19, ResNet50, InceptionResNetV2, ResNet152V2, EfficientNetB2, ConvNeXtTiny
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, GlobalMaxPooling2D, Dropout
random.seed(555)
```

✓ Read and Preparation of Data

```
dir_train = '/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/train'
dir_test = '/kaggle/input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/test'
```

```
# Train
dir_train_def = dir_train + '/def_front/'
dir_train_ok = dir_train + '/ok_front/'
# Test
dir_test_def = dir_test + '/def_front/'
dir_test_ok = dir_test + '/ok_front/'
```

```
image_files_train_def = os.listdir(dir_train_def)
image_files_train_ok = os.listdir(dir_train_ok)
```

```
n = len(image_files_train_def)
m = len(image_files_train_ok)
print(f'the number of all the images in the training set is {n+m}')
print(f'number of def imgs is {n}')
print(f'number of ok imgs is {m}')
print(f'the ratio between ok and def imgs is {m/n}')
```

```
🔍 the number of all the images in the training set is 6633
number of def imgs is 3758
number of ok imgs is 2875
the ratio between ok and def imgs is 0.7650345928685471
```

✓ Example of def and ok folders

```
# Function to get a list of random image files from a directory
def get_random_image_files(directory, num_files):
    files = os.listdir(directory)
    random.shuffle(files)
    return files[:num_files]
```

```
# Create a 2x3 grid for "ok_front" images
plt.figure(figsize=(12, 8))
plt.suptitle('Examples From The Training Dataset')
```

```
for i in range(3):
    plt.subplot(2, 3, i + 1)
    image_files_ok = get_random_image_files(dir_train_ok, 3)
    img = Image.open(os.path.join(dir_train_ok, image_files_ok[i]))
    plt.imshow(img)
    plt.title('ok_front')
```

```
# Create a 2x3 grid for "def_front" images
for i in range(3):
    plt.subplot(2, 3, i + 4)
    image_files_def = get_random_image_files(dir_train_def, 3)
    img = Image.open(os.path.join(dir_train_def, image_files_def[i]))
    plt.imshow(img)
    plt.title('def_front')
```

```
plt.tight_layout()
plt.show()
```



Examples From The Training Dataset

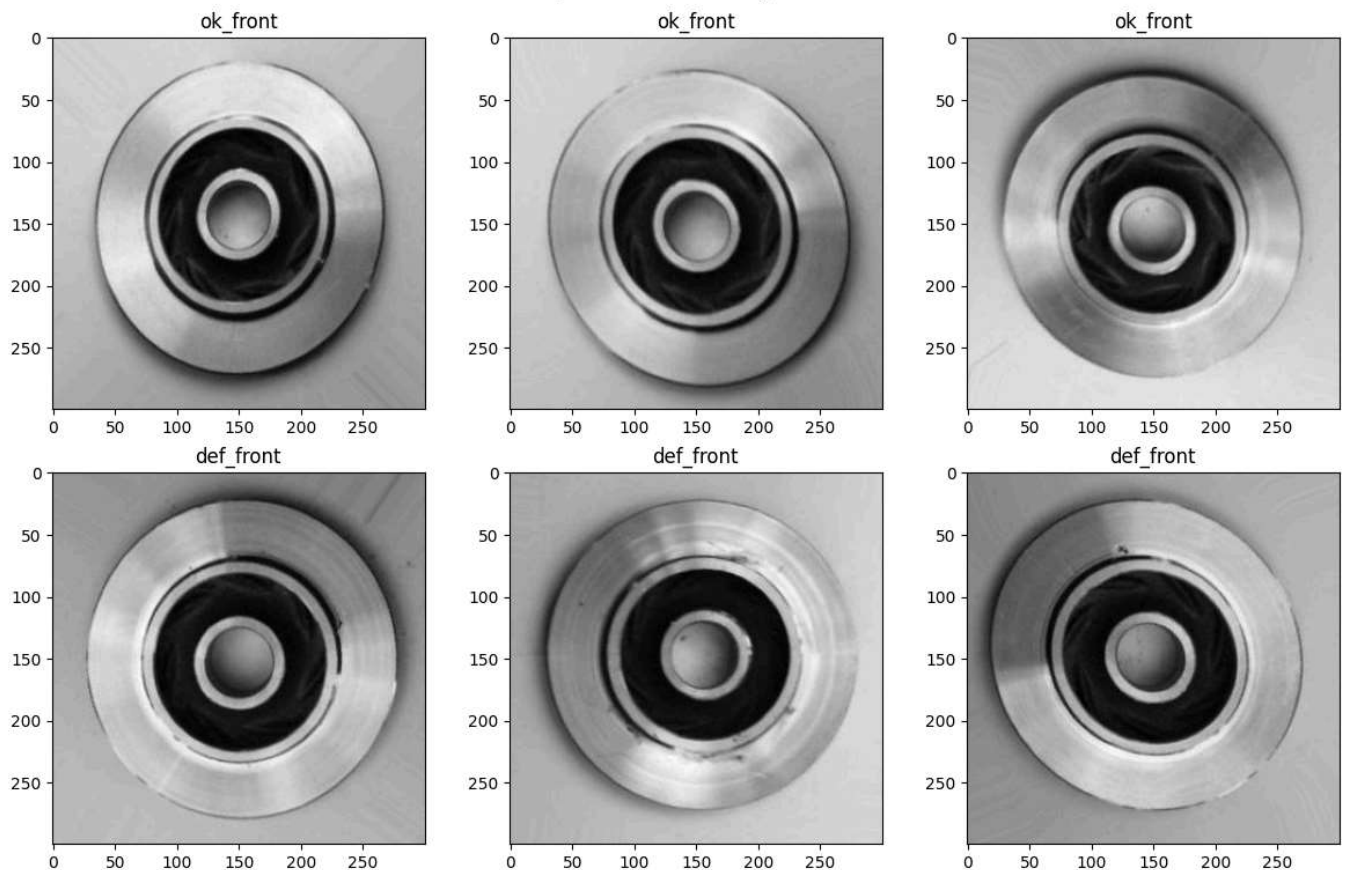


Image Details Extraction

```
# same for the ok_front
img = Image.open(os.path.join(dir_train_def, image_files_def[0]))
img.size, img.mode
```

```
((300, 300), 'RGB')
```

Data Augmentation

```
img_size = (300,300)
rand_seed = 555
batch_size = 32
epochs = 5
```

```
train_gen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    vertical_flip=True,
    rotation_range=40,
    brightness_range=[0.2, 1.5],
    validation_split=0.4,
)
```

```
test_gen = ImageDataGenerator(rescale=1./255)
```

Splitting data into training and validation set

```
arg_train = {'target_size': img_size,
             'color_mode': 'rgb',
             'classes': {'ok_front': 0,
```

```

        'def_front': 1},
        'class_mode': 'binary',
        'batch_size': batch_size,
        'seed': rand_seed}


arg_test = {'target_size': img_size,
            'color_mode': 'rgb',
            'classes': {'ok_front': 0,
                        'def_front': 1},
            'class_mode': 'binary',
            'batch_size': batch_size,
            'seed': rand_seed,
            'shuffle': False}

# 80%
train_set = train_gen.flow_from_directory(directory=dir_train,
                                          subset='training',
                                          **arg_train)

#20%
valid_set = train_gen.flow_from_directory(directory=dir_train,
                                          subset='validation',
                                          **arg_train)

# for the 0 and 1 ...etc
test_set = test_gen.flow_from_directory(directory=dir_test,
                                       **arg_test)

```

 Found 3980 images belonging to 2 classes.
 Found 2653 images belonging to 2 classes.
 Found 715 images belonging to 2 classes.

✓ Training the Model

```
weights_path = '/kaggle/input/your-uploaded-dataset/resnet152v2_weights_tf_dim_ordering_tf_kernels_notop.h5'
```

```
from keras import regularizers
```

```

def load_pretrained_model(model_name):
    if model_name == 'Xception':
        return Xception(weights='imagenet', include_top=False)
    elif model_name == 'ResNet50':
        return ResNet50(weights='imagenet', include_top=False)
    elif model_name == 'InceptionResNetV2':
        return InceptionResNetV2(weights='imagenet', include_top=False)
    elif model_name == 'ResNet152V2':
        return ResNet152V2(weights=weights_path, include_top=False)

def create_and_compile_model(base_model, learning_rate=0.01):
    x = base_model.output
    x = GlobalMaxPooling2D()(x)

    # Correct usage of regularizer
    x = Dense(256, activation='relu', kernel_regularizer=regularizers.l1_l2()(x))
    x = Dense(128, activation='relu', kernel_regularizer=regularizers.l1_l2()(x))

    predictions = Dense(1, activation='sigmoid')(x) # Sigmoid for binary classification

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

    # Freeze the base model layers
    for layer in base_model.layers:
        layer.trainable = False

    # Compile the model with the optimizer and the loss/metrics
    model.compile(optimizer=Adam(learning_rate=learning_rate),
                  loss='binary_crossentropy', # Binary classification
                  metrics=['accuracy', AUC(), Precision(), Recall()])

    return model

def train_and_evaluate_model(model, model_name, train_set, valid_set, test_set, epochs=10, batch_size=32):
    model_checkpoint = ModelCheckpoint(
        filepath='best_model.keras', # Updated file extension to `.keras`
        monitor='val_auc',
        save_best_only=True,

```

```

        save_weights_only=False,
        mode='max',
        verbose=1
    )

    # Start the timer
    start_time = time.time()

    history = model.fit(
        train_set,
        steps_per_epoch=train_set.samples // batch_size,
        epochs=epochs,
        validation_data=valid_set,
        validation_steps=valid_set.samples // batch_size,
        callbacks=[model_checkpoint]
    )

    # End the timer
    end_time = time.time()
    training_time = end_time - start_time

    # Evaluate the model on the test set
    test_loss, *test_metrics = model.evaluate(test_set, steps=test_set.samples // batch_size)
    test_acc = test_metrics[0]
    test_auc = test_metrics[1]
    test_precision = test_metrics[2]
    test_recall = test_metrics[3]

    # Start the evaluation timer
    evaluation_time = time.time() - end_time

    # Print the results
    print(f"model: {model_name}")
    print(f"Test accuracy: {test_acc * 100:.2f}%")
    print(f"Test loss: {test_loss:.4f}")
    print(f"Test AUC: {test_auc:.4f}")
    print(f"Test Precision: {test_precision:.4f}")
    print(f"Test Recall: {test_recall:.4f}")
    print(f"Training time: {training_time:.2f} seconds")
    print(f"Evaluation time: {evaluation_time:.2f} seconds")

    y_true = test_set.classes
    y_pred = model.predict(test_set)
    y_pred_classes = (y_pred > 0.5).astype(int) # Convert to 0 or 1 based on a threshold
    cm = confusion_matrix(y_true, y_pred_classes)
    report = classification_report(y_true, y_pred_classes)
    auc = roc_auc_score(y_true, y_pred)

    return test_acc, cm, report, auc, test_precision, test_recall, history.history

```

✓ Running Epochs

```

%%time

#model_names = ['Xception', 'InceptionResNetV2', 'ResNet152V2']
model_names = ['ResNet152V2']
results = {}

for model_name in model_names:
    print(f"Training model: {model_name}")
    base_model = load_pretrained_model(model_name)
    model = create_and_compile_model(base_model)
    test_acc, cm, report, auc, precision, recall, history = train_and_evaluate_model(model,
                                                                                      model_name,
                                                                                      train_set,
                                                                                      valid_set,
                                                                                      test_set,
                                                                                      epochs,
                                                                                      batch_size)

    results[model_name] = {
        'test_accuracy': test_acc,
        'confusion_matrix': cm,
        'classification_report': report,
        'roc_auc': auc,
        'history': history
    }

```

```

Training model: ResNet152V2
Epoch 1/5
124/124 ----- 215s 2s/step - accuracy: 0.7328 - auc_1: 0.7512 - loss: 13.5059 - precision_1: 0.7568 - recall_1: 0.7568
Epoch 2/5
/usr/local/lib/python3.10/dist-packages/keras/src/callbacks/model_checkpoint.py:206: UserWarning: Can save best model only with val_
self._save_model(epoch=epoch, batch=None, logs=logs)
124/124 ----- 4s 30ms/step - accuracy: 0.9062 - auc_1: 1.0000 - loss: 0.1722 - precision_1: 0.8500 - recall_1: 1.0000
Epoch 3/5
124/124 ----- 174s 1s/step - accuracy: 0.9764 - auc_1: 0.9961 - loss: 0.0699 - precision_1: 0.9832 - recall_1: 0.9741
Epoch 4/5
124/124 ----- 1s 5ms/step - accuracy: 1.0000 - auc_1: 1.0000 - loss: 0.0017 - precision_1: 1.0000 - recall_1: 1.0000
Epoch 5/5
124/124 ----- 174s 1s/step - accuracy: 0.9888 - auc_1: 0.9972 - loss: 0.0404 - precision_1: 0.9903 - recall_1: 0.9891
22/22 ----- 8s 368ms/step - accuracy: 0.9854 - auc_1: 0.6508 - loss: 0.0328 - precision_1: 0.6522 - recall_1: 0.6181
model: ResNet152V2
Test accuracy: 97.16%
Test loss: 0.0680
Test AUC: 0.9987
Test Precision: 1.0000
Test Recall: 0.9548
Training time: 582.95 seconds
Evaluation time: 8.35 seconds
23/23 ----- 28s 898ms/step
CPU times: user 12min 32s, sys: 30.9 s, total: 13min 3s
Wall time: 10min 22s

```

✓ Inferencing the Model on validation set

```

best_model_name = max(results, key=lambda k: results[k]['roc_auc'])
best_model_results = results[best_model_name]

# Access the performance metrics
best_test_acc = best_model_results['test_accuracy']
best_cm = best_model_results['confusion_matrix']
best_report = best_model_results['classification_report']
best_auc = best_model_results['roc_auc']
best_history = best_model_results['history']

print(f"Best Model: {best_model_name}")
print(f"Test Accuracy: {best_test_acc * 100:.2f}%")
print("Confusion Matrix:")
print(best_cm)
print("Classification Report:")
print(best_report)
print(f"ROC AUC: {best_auc:.2f}")

# Plot training and validation accuracy and loss

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(best_history['accuracy'], label='Training Accuracy')
plt.plot(best_history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(best_history['loss'], label='Training Loss')
plt.plot(best_history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

```



Best Model: ResNet152V2

Test Accuracy: 97.16%

Confusion Matrix:

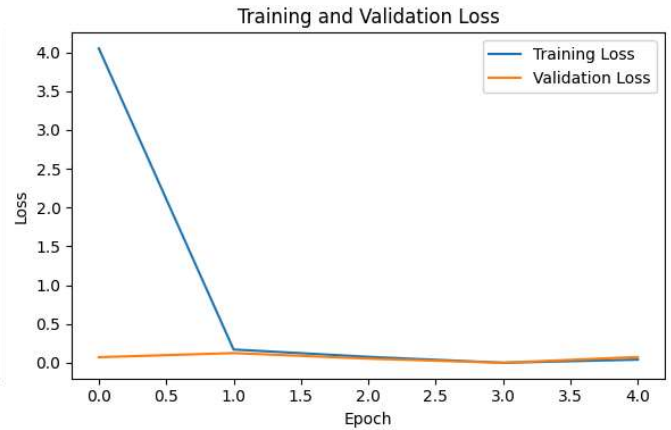
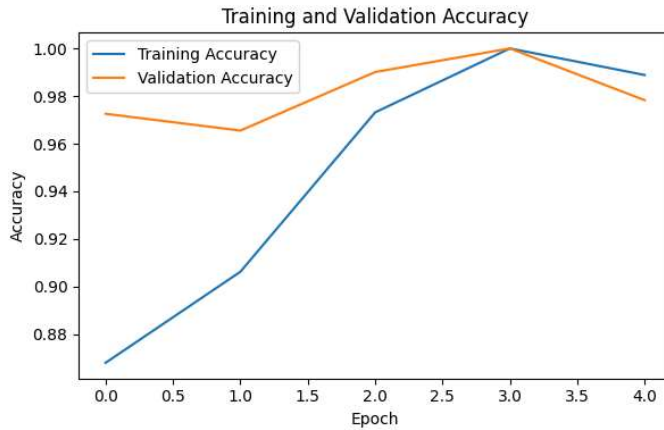
[[262 0]

[21 432]]

Classification Report:

	precision	recall	f1-score	support
0	0.93	1.00	0.96	262
1	1.00	0.95	0.98	453
accuracy			0.97	715
macro avg	0.96	0.98	0.97	715
weighted avg	0.97	0.97	0.97	715

ROC AUC: 1.00



✓ Validation Accuracy

```
best_model = load_model("/content/best_model.keras")
# Assuming you have set up the 'test_set' as mentioned earlier
predictions = best_model.predict(test_set)
predicted_labels = (predictions > 0.5).astype(int)
true_labels = test_set.classes
```

```
# Calculate classification report and confusion matrix
print(classification_report(true_labels, predicted_labels))
conf_matrix = confusion_matrix(true_labels, predicted_labels)
```



23/23 ————— 23s 688ms/step

	precision	recall	f1-score	support
0	0.99	0.99	0.99	262
1	0.99	0.99	0.99	453
accuracy			0.99	715
macro avg	0.99	0.99	0.99	715
weighted avg	0.99	0.99	0.99	715

✓ Confusion Matrix

```
# Assuming you have 'conf_matrix' calculated as the confusion matrix
total_samples = np.sum(conf_matrix)
```

```
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.get_cmap('Blues'))
plt.title(f'Confusion Matrix\nTotal Samples: {total_samples}')
plt.colorbar()
tick_marks = np.arange(2) # Assuming you have 2 classes (0 and 1)
plt.xticks(tick_marks, ['Ok Front', 'Defected Front'])
plt.yticks(tick_marks, ['Ok Front', 'Defected Front'])
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
# Add numbers inside the boxes with black color and percentages
for i in range(2):
    for j in range(2):
```



```
count = conf_matrix[i, j]
percentage = (count / total_samples) * 100
plt.text(j, i, f'{count}\n({percentage:.2f}%', horizontalalignment='center', color='black', fontsize=12)
```

```
plt.show()
```

