EE 604
Assignment 2
Dhruv - 210338

Q1 -

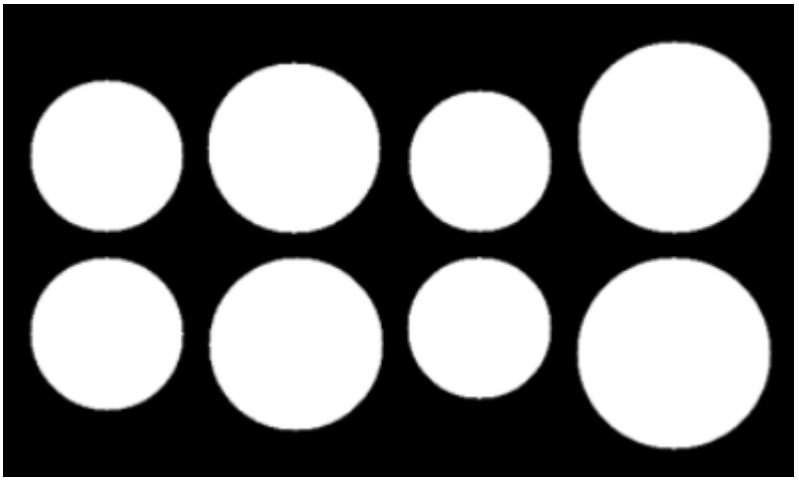
Original Image

Steps Followed -

1. **Image Preprocessing:** Convert the image to grayscale, apply Gaussian blur for noise reduction, and use Canny edge detection to find edges representing potential coin boundaries.
2. **Hough Circle Transform:**
    - Define parameter space with potential radii and center coordinates.
    - Initialize a 3D accumulator array to store votes for circle centers.
    - For each edge pixel, compute possible circle centers across all angles and update the accumulator with votes for valid centers.
3. **Peak Detection**: Identify peaks in the accumulator using a threshold and non-maximum suppression to detect prominent circles while avoiding duplicate detections.

4. **Circle Extraction:** Extract circle parameters (center coordinates and radii) from accumulator peaks.
5. **Binary Mask Creation:** Generate a binary mask highlighting detected circles by drawing filled circles at the detected coin locations.

Binary Mask of Detected Circles -



Time To Run the code -
~7 - 10 minutes

```
import cv2
import numpy as np

from google.colab import drive
drive.mount('/content/drive')

img = cv2.imread('')

# Grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Apply Gaussian blur to reduce noise
gray_blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# Edge detection
edges = cv2.Canny(gray_blurred, 50, 150)

rows, cols = edges.shape
```

```
# Radii range
min_radius = 15
max_radius = 80
radii_range = range(min_radius, max_radius + 1, 1)

# Accumulator array for each radius
accumulators = {}
for r in radii_range:
    accumulators[r] = np.zeros((rows, cols), dtype=np.uint64)

# Pre-compute circle perimeters for each radius
circle_perimeters = {}
for r in radii_range:
    circle_perimeters[r] = []
    for angle in range(0, 360):
        theta = np.deg2rad(angle)
        dx = int(r * np.cos(theta))
        dy = int(r * np.sin(theta))
        circle_perimeters[r].append((dx, dy))

# For each edge point, update the accumulator arrays
edge_points = np.argwhere(edges != 0)

for x, y in edge_points:
    for r in radii_range:
        for dx, dy in circle_perimeters[r]:
            a = x - dy
            b = y - dx
            if 0 <= a < rows and 0 <= b < cols:
                accumulators[r][a, b] += 1

# Find peaks in the accumulator arrays
# Set a threshold for detecting circles
threshold = 0.99

# Create an empty mask for the output
mask = np.zeros_like(gray, dtype=np.uint8)

for r in radii_range:
    accumulator = accumulators[r]
    accumulator_normalized = accumulator / accumulator.max()
```

```
    accumulator_thresholded = accumulator_normalized > threshold
    centers = np.argwhere(accumulator_thresholded)
    for center in centers:
        a, b = center
        cv2.circle(mask, (b, a), r, (255), thickness=-1)

cv2.imwrite('coins_mask.png', mask)
# Find peaks in the accumulator arrays
# Set a threshold for detecting circles
threshold = 0.99

mask = np.zeros_like(gray, dtype=np.uint8)

for r in radii_range:
    accumulator = accumulators[r]
    accumulator_normalized = accumulator / accumulator.max()
    accumulator_thresholded = accumulator_normalized > threshold
    centers = np.argwhere(accumulator_thresholded)
    for center in centers:
        a, b = center
        cv2.circle(mask, (b, a), r, (255), thickness=-1)

cv2.imwrite('coins_mask.png', mask)
```

Q 2 -

This code implements a binary image classification model using a pre-trained VGG16 convolutional neural network.

Data Preparation:

- The code defines directories for training and testing images, each labeled as "Normal" or "Defective."
- It applies data augmentation to the training images, including rescaling, rotation, zooming, and flipping. The test set is only rescaled.
- The images are loaded with a target size of 224x224 pixels and a batch size of 32.

Model Setup:

- A pre-trained VGG16 model is loaded, excluding the top (classification) layers, allowing it to serve as a feature extractor.
- The VGG16 layers are frozen to prevent them from being updated during training.
- Custom layers are added on top of VGG16, including a Flatten layer, a Dense layer with 512 units and ReLU activation, a dropout layer to reduce overfitting, and a final output layer with sigmoid activation for binary classification.

Model Compilation and Training:

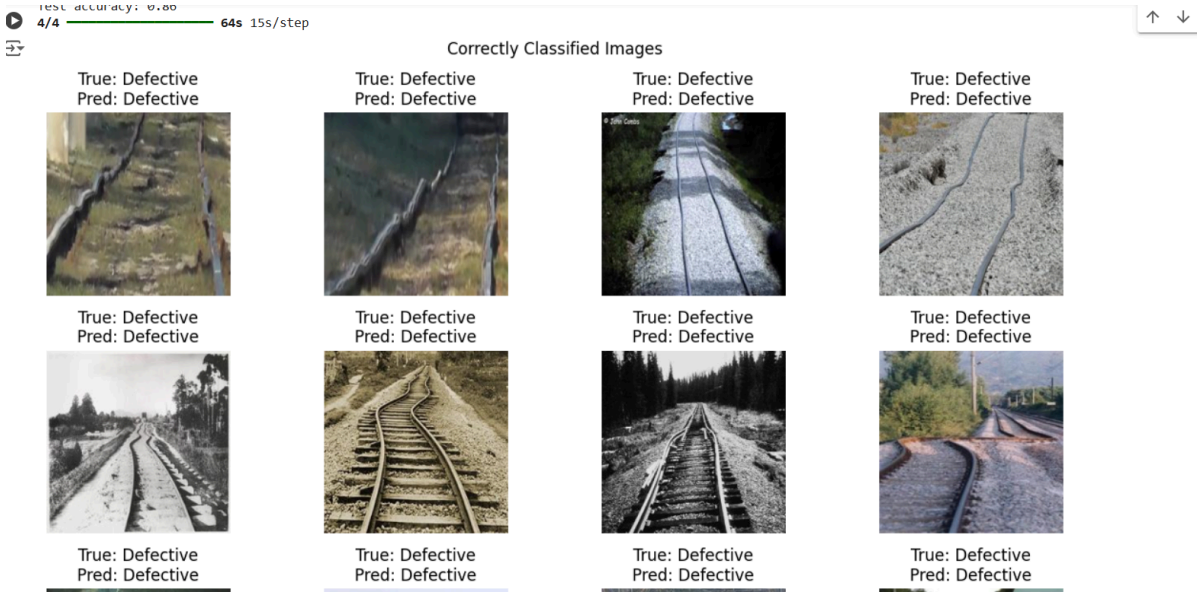
- The model is compiled with the Adam optimizer and binary cross-entropy loss.
- It is trained for 5 epochs on the augmented training data, with validation on the test data.

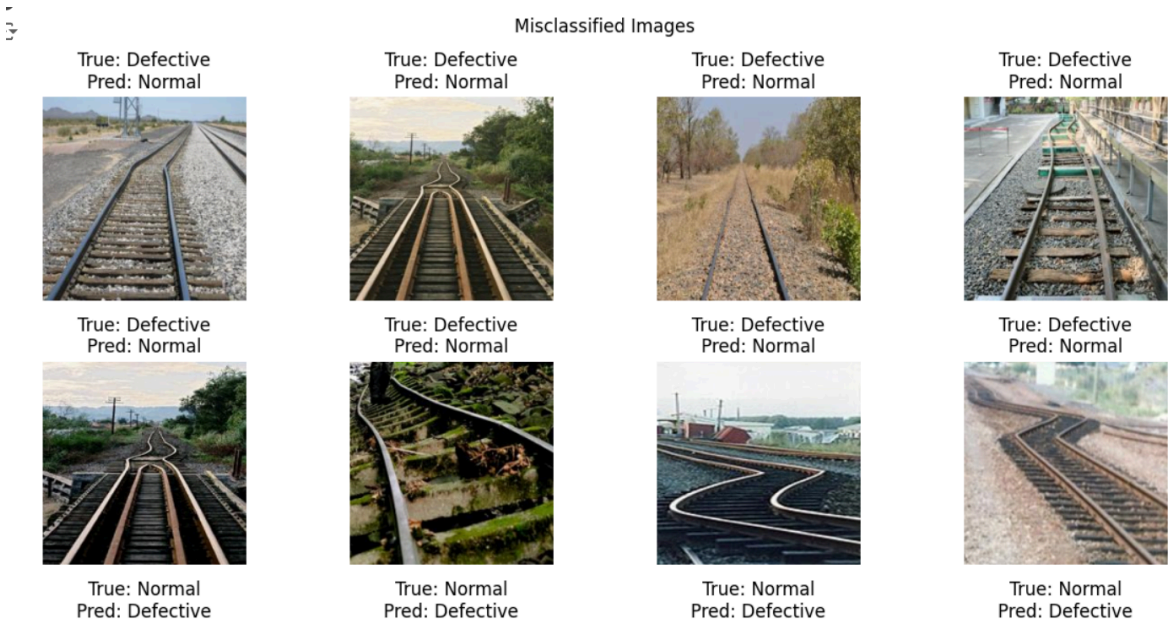
Model Evaluation:

- The model's accuracy on the test set is computed and printed.
- Predictions are generated for the test images, and each image is classified as "Defective" or "Normal" based on a threshold of 0.5.

Visualization:

- Correctly and incorrectly classified images from the test set are identified.
- A custom function display_images visualizes a selection of correctly and incorrectly classified images, displaying each with its true and predicted labels.





Accuracy - 93.91%

```python
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Dropout
import matplotlib.pyplot as plt
import numpy as np
import os

import logging
logging.getLogger('tensorflow').setLevel(logging.ERROR)

train_dir = '/content/drive/MyDrive/604 assign 2/TMD_SIBGRAP-2021/NewR22/train'
test_dir = '/content/drive/MyDrive/604 assign 2/TMD_SIBGRAP-2021/NewR22/test'

# Image dimensions
IMG_HEIGHT = 224
IMG_WIDTH = 224

# Data augmentation for the training set
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    zoom_range=0.15,
    horizontal_flip=True,
    vertical_flip=True
)

# No augmentation for the testing set, only rescaling
test_datagen = ImageDataGenerator(rescale=1./255)

# Load training data
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=32,
    class_mode='binary'
)

# Load testing data
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=32,
    class_mode='binary',
    shuffle=False
)
```

```
# Load the pre-trained VGG16 model without the top layers
base_model = VGG16(weights='imagenet', include_top=False,
input_shape=(IMG_HEIGHT, IMG_WIDTH, 3))

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

# Add custom layers on top of the base model
x = base_model.output
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(1, activation='sigmoid')(x)

# Define the full model
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=test_generator
)

# Evaluate the model
test_loss, test_acc = model.evaluate(test_generator)
print(f'Test accuracy: {test_acc:.2f}')

# Get the filenames, true labels, and predictions
filenames = test_generator.filenames
true_labels = test_generator.classes
predictions = model.predict(test_generator)
predicted_labels = (predictions > 0.5).astype(int).reshape(-1)

# Function to display images
def display_images(images, labels, preds, title):
    plt.figure(figsize=(12, 8))
    for i in range(len(images)):
        plt.subplot(3, 4, i+1)
        plt.imshow(images[i])
        plt.title(f'True: {labels[i]}\nPred: {preds[i]}')
        plt.axis('off')
```

```
    plt.suptitle(title)
    plt.tight_layout()
    plt.show()

# Indices of correctly classified images
correct_indices = np.where(predicted_labels == true_labels)[0]
# Select a few examples
correct_samples = correct_indices[:12]
# Load images
correct_images = [plt.imread(os.path.join(test_dir, filenames[i])) for i in
correct_samples]
# Get labels
correct_true_labels = ['Defective' if l == 0 else 'Normal' for l in
true_labels[correct_samples]]
correct_pred_labels = ['Defective' if l == 0 else 'Normal' for l in
predicted_labels[correct_samples]]

display_images(correct_images, correct_true_labels, correct_pred_labels,
'Correctly Classified Images')


incorrect_indices = np.where(predicted_labels != true_labels)[0]

incorrect_samples = incorrect_indices[:12]

incorrect_images = [plt.imread(os.path.join(test_dir, filenames[i])) for i in
incorrect_samples]

incorrect_true_labels = ['Defective' if l == 0 else 'Normal' for l in
true_labels[incorrect_samples]]
incorrect_pred_labels = ['Defective' if l == 0 else 'Normal' for l in
predicted_labels[incorrect_samples]]

display_images(incorrect_images, incorrect_true_labels, incorrect_pred_labels,
'Misclassified Images')
```