**Code 1:**

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import matplotlib.pyplot as plt

railroad\_inspection/

train/

defect/

no\_defect/

validation/

defect/

no\_defect/

python

Copy

# Define paths

train\_dir = 'railroad\_inspection/train'

validation\_dir = 'railroad\_inspection/validation'

# Image dimensions

img\_height, img\_width = 150, 150

batch\_size = 32

# Data generators with data augmentation

train\_datagen = ImageDataGenerator(

rescale=1./255,

rotation\_range=40,

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\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size=(img\_height, img\_width),

batch\_size=batch\_size,

class\_mode='binary'

)

validation\_generator = validation\_datagen.flow\_from\_directory(

validation\_dir,

target\_size=(img\_height, img\_width),

batch\_size=batch\_size,

class\_mode='binary'

)

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(img\_height, img\_width, 3)),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(128, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(128, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

layers.Flatten(),

layers.Dense(512, activation='relu'),

layers.Dense(1, activation='sigmoid')

])

model.compile(

optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy']

)

model.summary()

history = model.fit(

train\_generator,

steps\_per\_epoch=train\_generator.samples // batch\_size,

epochs=30,

validation\_data=validation\_generator,

validation\_steps=validation\_generator.samples // batch\_size

)

# Plot training and validation accuracy and loss

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training accuracy')

plt.plot(epochs, val\_acc, 'b', label='Validation accuracy')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

model.save('railroad\_inspection\_model.h5')

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing import image

import numpy as np

# Load the model

model = load\_model('railroad\_inspection\_model.h5')

# Load an image for prediction

img\_path = 'path\_to\_test\_image.jpg'

img = image.load\_img(img\_path, target\_size=(img\_height, img\_width))

img\_array = image.img\_to\_array(img)

img\_array = np.expand\_dims(img\_array, axis=0)

img\_array /= 255.

# Make prediction

prediction = model.predict(img\_array)

if prediction[0] > 0.5:

print("Defect detected!")

else:

print("No defect detected.")

**Code 2:**

**##Step 1: Import libraries**

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import numpy as np

import matplotlib.pyplot as plt

**##Step 2: Load and Preprocess Data**

**## Create ImageDataGenerators for training and validation**

train\_datagen = ImageDataGenerator(rescale=1./255,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True)

test\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_directory('path/to/train\_data',

target\_size=(128, 128),

batch\_size=32,

class\_mode='binary')

validation\_generator = test\_datagen.flow\_from\_directory('path/to/test\_data',

target\_size=(128, 128),

batch\_size=32,

class\_mode='binary')

**##Step 3: Build the Deep Neural Network (CNN)**

**##This is a simple CNN architecture that could be used for track inspection.**

model = models.Sequential()

# Convolutional layer 1

model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(128, 128, 3)))

model.add(layers.MaxPooling2D((2, 2)))

# Convolutional layer 2

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

# Convolutional layer 3

model.add(layers.Conv2D(128, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

# Flatten the layers

model.add(layers.Flatten())

# Fully connected layers

model.add(layers.Dense(128, activation='relu'))

model.add(layers.Dropout(0.5)) # Dropout to prevent overfitting

model.add(layers.Dense(1, activation='sigmoid')) # Binary classification (defect or no defect)

# Compile the model

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

# Summarize the model architecture

model.summary()

**##Step 4: Train the Model**

**##Now we can train the model using the data generators created earlier.**

history = model.fit(

train\_generator,

steps\_per\_epoch=train\_generator.samples // train\_generator.batch\_size,

epochs=10,

validation\_data=validation\_generator,

validation\_steps=validation\_generator.samples // validation\_generator.batch\_size

)

**##Step 5: Evaluate the Model**

**##After training, you can evaluate the model's performance on the test set:**

test\_loss, test\_acc = model.evaluate(validation\_generator)

print(f"Test accuracy: {test\_acc\*100:.2f}%")

**##Step 6: Visualize Training History**

**##You can plot the training and validation accuracy and loss to monitor overfitting.**

**# Plotting training & validation accuracy values**

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

**# Plotting training & validation loss values**

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

**##Step 7: Make Predictions**

**##After the model is trained, you can use it to make predictions on new images:**

from tensorflow.keras.preprocessing import image

# Load a new image to test

img\_path = 'path/to/new\_image.jpg'

img = image.load\_img(img\_path, target\_size=(128, 128))

img\_array = image.img\_to\_array(img) # Convert image to array

img\_array = np.expand\_dims(img\_array, axis=0) # Add batch dimension

# Predict using the trained model

prediction = model.predict(img\_array)

print("Defect Detected" if prediction[0] > 0.5 else "No Defect")