**Grad-CAM Visualization for Fake Color Detection**

import cv2

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Model

import matplotlib.pyplot as plt

# Load the trained model

model = tf.keras.models.load\_model('fake\_colorized\_image\_detection\_model.h5')

# Load an image for Grad-CAM analysis

img\_path = 'fake\_colorized\_images/fake\_image\_1.jpg' # Example image path

img = cv2.imread(img\_path)

img = cv2.resize(img, (224, 224))

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

# Grad-CAM for visualization

grad\_model = Model([model.inputs], [model.get\_layer('conv2d\_2').output, model.output])

with tf.GradientTape() as tape:

conv\_outputs, predictions = grad\_model(img)

loss = predictions[:, 0]

# Compute the gradients

grads = tape.gradient(loss, conv\_outputs)[0]

# Average the gradients

pooled\_grads = tf.reduce\_mean(grads, axis=(0, 1, 2))

# Multiply the gradients by the feature map

conv\_outputs = conv\_outputs[0]

for i in range(128): # Assuming 128 filters in the last Conv layer

conv\_outputs[:, :, i] \*= pooled\_grads[i]

# Generate the heatmap

heatmap = np.mean(conv\_outputs, axis=-1)

# Normalize the heatmap between 0 and 1

heatmap = np.maximum(heatmap, 0)

heatmap /= np.max(heatmap)

# Convert heatmap to RGB

heatmap = cv2.resize(heatmap, (224, 224))

heatmap = np.uint8(255 \* heatmap)

heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP\_JET)

# Superimpose heatmap on the original image

superimposed\_img = heatmap \* 0.4 + cv2.imread(img\_path)

# Plot the result

plt.imshow(superimposed\_img[:, :, ::-1])

plt.title('Grad-CAM for Fake Color Detection')

plt.axis('off')

plt.show()