""" Model Interpretation """

model = tf.keras.models.load\_model('VGG16\_with\_CLAHE.h5')

model.summary()

img\_path = 'E:/Software/practice\_projects/seat\_belt\_detection/04-35-47.jpg'

img = read\_img(img\_path, 0)

clahe = cv2.createCLAHE(clipLimit=2, tileGridSize=(8,8))

final\_img = clahe.apply(img)

final\_img = final\_img / 255

plt.figure(figsize=(10, 5))

# plt.suptitle("Contrast Limited Adaptive Histogram Equalization (CLAHE) on Gray scale image ")

plt.imshow(final\_img, cmap='gray')

# Expand dimensions

processed\_img = final\_img[np.newaxis, :, :, np.newaxis]

processed\_img = np.repeat(processed\_img, 3, -1)

pred\_label = np.argmax(model.predict(processed\_img))

print(pred\_label)

def get\_CAM(processed\_image, actual\_label, layer\_name='block5\_conv3'):

model\_grad = Model([model.inputs],

[model.get\_layer(layer\_name).output, model.output])

with tf.GradientTape() as tape:

conv\_output\_values, predictions = model\_grad(processed\_image)

# watch the conv\_output\_values

tape.watch(conv\_output\_values)

## Use binary cross entropy loss

## actual\_label is 0 if cat, 1 if dog

# get prediction probability of dog

# If model does well,

# pred\_prob should be close to 0 if cat, close to 1 if dog

pred\_prob = predictions[:,1]

# make sure actual\_label is a float, like the rest of the loss calculation

actual\_label = tf.cast(actual\_label, dtype=tf.float32)

# add a tiny value to avoid log of 0

smoothing = 0.00001

# Calculate loss as binary cross entropy

loss = -1 \* (actual\_label \* tf.math.log(pred\_prob + smoothing) + (1 - actual\_label) \* tf.math.log(1 - pred\_prob + smoothing))

print(f"binary loss: {loss}")

# get the gradient of the loss with respect to the outputs of the last conv layer

grads\_values = tape.gradient(loss, conv\_output\_values)

grads\_values = K.mean(grads\_values, axis=(0,1,2))

conv\_output\_values = np.squeeze(conv\_output\_values.numpy())

grads\_values = grads\_values.numpy()

# weight the convolution outputs with the computed gradients

for i in range(512):

conv\_output\_values[:,:,i] \*= grads\_values[i]

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

heatmap = np.maximum(heatmap, 0)

heatmap /= heatmap.max()

del model\_grad, conv\_output\_values, grads\_values, loss

return heatmap

sample\_label = 1

heatmap = get\_CAM(processed\_img, sample\_label)

plt.imshow(heatmap)

cnt = -1

def save\_and\_display\_gradcam(img\_path, heatmap, cam\_path="cam.jpg", alpha=0.4):

# Load the original image

img = tf.keras.preprocessing.image.load\_img(img\_path)

img = tf.keras.preprocessing.image.img\_to\_array(img)

# img = img[:, :, np.newaxis]

# img = np.repeat(img, 3, -1)

# Rescale heatmap to a range 0-255

heatmap = np.uint8(255 \* heatmap)

# Use jet colormap to colorize heatmap

jet = cm.get\_cmap("jet")

# Use RGB values of the colormap

jet\_colors = jet(np.arange(256))[:, :3]

jet\_heatmap = jet\_colors[heatmap]

# Create an image with RGB colorized heatmap

jet\_heatmap = tf.keras.preprocessing.image.array\_to\_img(jet\_heatmap)

jet\_heatmap = jet\_heatmap.resize((img.shape[1], img.shape[0]))

jet\_heatmap = tf.keras.preprocessing.image.img\_to\_array(jet\_heatmap)

# Superimpose the heatmap on original image

superimposed\_img = jet\_heatmap \* alpha + img

superimposed\_img = tf.keras.preprocessing.image.array\_to\_img(superimposed\_img)

# Save the superimposed image

global cnt

cnt += 1

cam\_path = 'cam' + str(cnt) + '.jpg'

superimposed\_img.save(cam\_path)

# Display Grad CAM

display(Image(cam\_path))

save\_and\_display\_gradcam(img\_path, heatmap)