[6]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, models

# Load and preprocess dataset
full_ds = tf.keras.utils.image_dataset_from_directory(
    r'cat_dog',         # Root folder with subfolders as class names
    image_size=(228, 228),
    color_mode='grayscale'                      # Any batch size works
).map(lambda x, y: (x / 255.0, y))

# Convert dataset to numpy arrays
x_full, y_full = [], []
for images, labels in full_ds:
    x_full.append(images.numpy())
    y_full.append(labels.numpy())
x_full = np.concatenate(x_full, axis=0)
y_full = np.concatenate(y_full, axis=0)

# Train-test split
x_train, x_test, y_train, y_test = train_test_split(x_full, y_full, test_size=0.3, random_state=42)

# CNN model definition
def build_cnn_model():
    inputs = tf.keras.Input(shape=(228, 228, 1))

    x = layers.Conv2D(16, 3, activation='relu')(inputs)
    x = layers.MaxPooling2D()(x)
    x = layers.Conv2D(32, 3, activation='relu')(x)
    x = layers.MaxPooling2D()(x)
    x = layers.Conv2D(64, 3, activation='relu')(x)
    x = layers.Flatten()(x)
    outputs = layers.Dense(10, activation='softmax')(x)

    model = models.Model(inputs=inputs, outputs=outputs)
    return model

# Create the model
model = build_cnn_model()
model.summary()
# Compile and train the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
```

[6]:
```
Found 132 files belonging to 2 classes.
```

[6]:
```
Model: "functional_2"
```

[6]:

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 228, 228, 1)	0
conv2d_3 (Conv2D)	(None, 226, 226, 16)	160
max_pooling2d_2 (MaxPooling2D)	(None, 113, 113, 16)	0
conv2d_4 (Conv2D)	(None, 111, 111, 32)	4,640
max_pooling2d_3 (MaxPooling2D)	(None, 55, 55, 32)	0
conv2d_5 (Conv2D)	(None, 53, 53, 64)	18,496
flatten_1 (Flatten)	(None, 179776)	0
dense_1 (Dense)	(None, 10)	1,797,770

[6]:
```
Total params: 1,821,066 (6.95 MB)
```

[6]:
```
Trainable params: 1,821,066 (6.95 MB)
```

[6]:
```
Non-trainable params: 0 (0.00 B)
```

[6]:
```
Epoch 1/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 7s 1s/step - accuracy: 0.2794 - loss: 1.7068
Epoch 2/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 1s/step - accuracy: 0.4744 - loss: 0.9120
Epoch 3/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 1s/step - accuracy: 0.6192 - loss: 0.6659
Epoch 4/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 1s/step - accuracy: 0.5467 - loss: 0.7084
Epoch 5/5
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 1s/step - accuracy: 0.7836 - loss: 0.5992
```

[6]:

GradCAM

[7]:
```
from tf_keras_vis.utils.scores import CategoricalScore
from tf_keras_vis.utils.model_modifiers import ReplaceToLinear
```

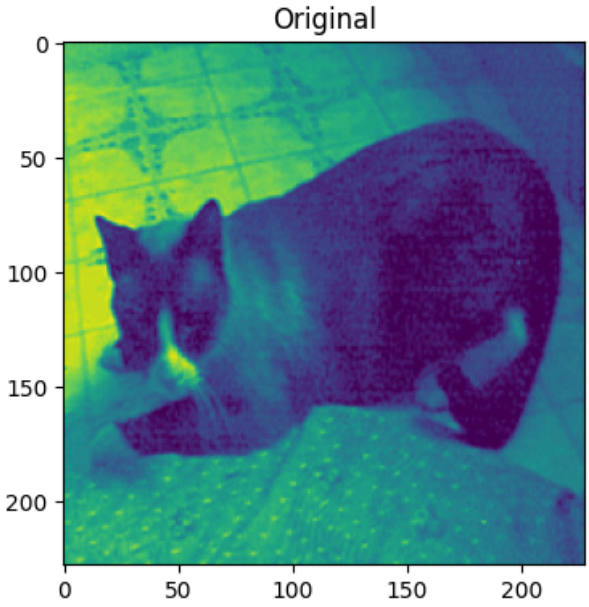
```
from tf_keras_vis.gradcam import Gradcam
from tensorflow.keras.preprocessing.image import load_img,img_to_array
```

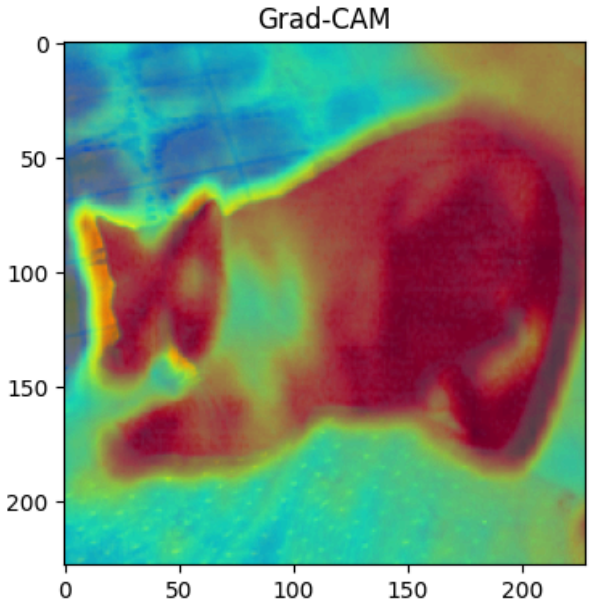
[8]:
```
# img_path = x_test[0]
img = x_test[0]
img_array = img_to_array(img) / 255.0
img_input = np.expand_dims(img_array, axis=0)

# 4. Apply Grad-CAM
score = CategoricalScore([np.argmax(model.predict(img_input))])
gm = Gradcam(model, model_modifier=ReplaceToLinear())
cam = gm(score, img_input, penultimate_layer=-1)
print(np.argmax(model.predict(img_input)),f"Original:{y_test[0]}")
```

[8]:
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 316ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 123ms/step
1 Original:0
```

[9]:
```
import matplotlib.pyplot as plt
f, ax = plt.subplots(1, 2, figsize=(12, 4))
ax[0].imshow(img)
ax[0].set_title("Original")
ax[1].imshow(img_array)
ax[1].imshow(cam[0], cmap='jet', alpha=0.5)
ax[1].set_title("Grad-CAM")
plt.tight_layout()
plt.show()
```

[9]:



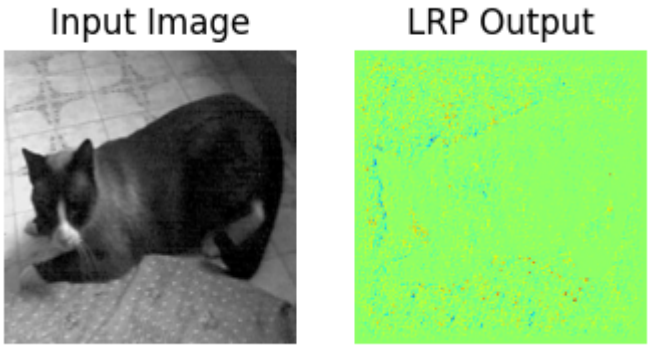


LRP AND SALIENCY MAP

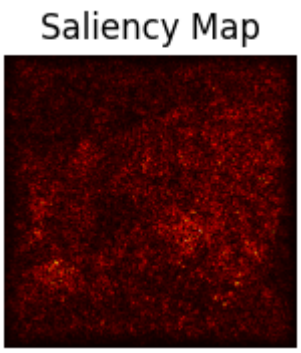
From Test Data

11]:
```
img = x_test[0]
img_tensor = tf.convert_to_tensor([img])  # Make it a batch
with tf.GradientTape() as tape:
    tape.watch(img_tensor)
    preds = model(img_tensor)
    class_idx = tf.argmax(preds[0])
    loss = preds[0, class_idx]

grads = tape.gradient(loss, img_tensor)[0].numpy()
relevance = grads * img  # Gradient × Input relevance
saliency = np.max(np.abs(grads),axis=-1)

# Visualize input and relevance map
plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title("Input Image")
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(relevance,cmap='jet')
plt.title("LRP Output")
plt.axis('off')
plt.show()

plt.subplot(1, 3, 3)
plt.imshow(saliency,cmap='hot')
plt.title("Saliency Map")
plt.axis('off')
plt.show()
```

11]:



11]:



BY Custom Image

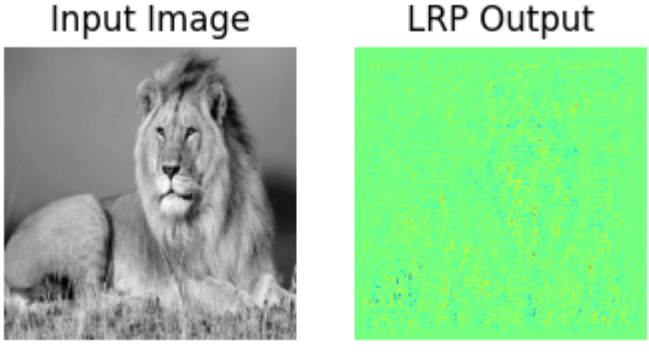
13]:

```
from tensorflow.keras.preprocessing.image import load_img,img_to_array
img=load_img('lion.jpg',target_size=(228, 228) # same image shape as in the model
             ,color_mode='grayscale' #if (x,x,1) or if (x,x,3) no need
              )
img=img_to_array(img)/255.0
img_batch=np.expand_dims(img,axis=0)
```

14]:

```
img_tensor = tf.convert_to_tensor(img_batch)  # Make it a batch
with tf.GradientTape() as tape:
    tape.watch(img_tensor)
    preds = model(img_tensor)
    class_idx = tf.argmax(preds[0])
    loss = preds[0, class_idx]

grads = tape.gradient(loss, img_tensor)[0].numpy()
relevance = grads * img  # Gradient × Input relevance
saliency = np.max(np.abs(grads), axis=-1)

# Visualize input and relevance map
plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title("Input Image")
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(relevance,cmap='jet')
plt.title("LRP Output")
plt.axis('off')
plt.show()

plt.subplot(1, 3, 3)
plt.imshow(saliency,cmap='hot')
plt.title("Saliency Map")
plt.axis('off')
plt.show()
```

14]:



14]: