

Amir Hossein Eyvazkhani - 1747696

Ex - 6

Task 1

```
In [ ]: import tensorflow as tf
        from tensorflow import keras
        from keras import layers
        import numpy as np
        from PIL import Image
        import matplotlib.pyplot as plt
        from matplotlib.colors import ListedColormap
```

```
In [ ]: resnet_model = keras.applications.ResNet50(
        include_top=True,
        weights="imagenet",
        classes=1000,
        classifier_activation="softmax",
    )
    image_file = tf.image.decode_png(tf.io.read_file('ouzel.jpg'))
    image = np.expand_dims(image_file, axis=0)
    image = tf.keras.applications.resnet50.preprocess_input(image)
    # Make a prediction
    result = resnet_model.predict(image)
    prediction_class = np.argmax(result)
    decoded_prediction = keras.applications.resnet50.decode_predictions(result, top=1)[0][0][1]
    print(f'The prediction class number is {prediction_class} which is a {decoded_prediction}')
    def gradient_input(model, inp, output_index=0):
        inp_tensor = tf.cast(inp, dtype=tf.float32)
        inp_tensor = tf.convert_to_tensor(inp_tensor)
        with tf.GradientTape() as t:
            t.watch(inp_tensor) # enable gradient recording w.r.t. to this tensor
            output = tf.squeeze(model(inp_tensor)) # forward inference
            if output.ndim>0:
                my_output = output[output_index] # pick right element from output
            else:
```

```

        my_output = output
        gradient = t.gradient(my_output, inp_tensor) # get gradient @my_output/@input
        gradient = np.squeeze(np.array(gradient)) # convert from tensor to numpy array
        return gradient

gredient = gradient_input(resnet_model, image, prediction_class)
# Compute the saliency map
saliency_map = tf.reduce_max(tf.abs(gredient), axis=-1)
# Normalize the saliency map between 0 and 255 for visualization
normalized_saliency_map = np.uint8(255 * (saliency_map - tf.reduce_min(saliency_map)) / (tf.reduce_max(saliency_map)
# Expand dimensions to add a channel for a grayscale image
normalized_saliency_map_with_channel = np.expand_dims(normalized_saliency_map, axis=-1)
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(image_file)
plt.title(f'Original Image: {decoded_prediction}')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(normalized_saliency_map, cmap='gray')
plt.title('Saliency Map')
plt.axis('off')

plt.tight_layout()
plt.show()

# Save the saliency map as a greyscale image
saliency_map_image = tf.keras.preprocessing.image.array_to_img(normalized_saliency_map_with_channel)
saliency_map_image.save('saliency_map.png')

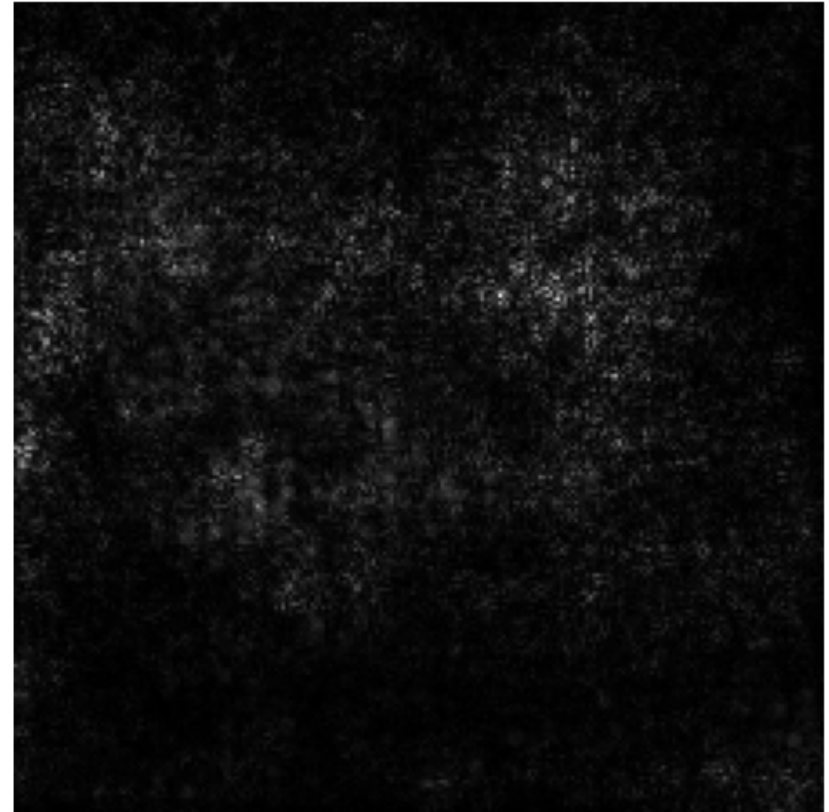
```

1/1 [=====] - 1s 1s/step
The prediction class number is 20 which is a water_ouzel
(224, 224)

Original Image: water_ouzel



Saliency Map



Task 2

```
In [ ]: # Build a simple CNN with strided convolution layers
def define_model():
    inputs = tf.keras.Input(shape=(128,128,1),name='Inputs')
    x = layers.Conv2D(16,kernel_size=(5,5),activation='tanh',strides=1,name='L1')(inputs)
    x = layers.Conv2D(16,kernel_size=(5,5),activation='tanh',strides=2,name='L2')(x)
    x = layers.Conv2D(16,kernel_size=(3,3),activation='tanh',strides=1,name='L3')(x)
    x = layers.Conv2D(16,kernel_size=(3,3),activation='tanh',strides=2,name='L4')(x)
    x = layers.Conv2D(16,kernel_size=(3,3),activation='tanh',strides=1,name='L5')(x)
    x = layers.Conv2D(16,kernel_size=(3,3),activation='tanh',strides=2,name='L6')(x)
    x = layers.Flatten()(x)
    outputs = layers.Dense(10,activation='softmax')(x)
    model = keras.Model(inputs=inputs, outputs=outputs)
```

```

return model

# For an output node in <model> at the intermediate layer <layer_name>
# and spatial position <x>,<y> and channel index <c>, compute the gradient of
# that output node with respect to the input <inp>
def get_gradient(model, layer_name, x, y, c, inp):
    input_tensor = tf.convert_to_tensor(inp, dtype=tf.float32)
    # Define a gradient tape
    with tf.GradientTape() as tape:
        tape.watch(input_tensor)
        intermediate_model = tf.keras.Model(inputs=model.input,
                                             outputs=model.get_layer(layer_name).output)
        output = tf.squeeze(intermediate_model(input_tensor))
        if output.ndim > 0:
            new_output = output[x, y, c]
        else:
            new_output = output
        gradient = tape.gradient(new_output, input_tensor)
    gradient = np.squeeze(np.array(gradient))
    return gradient

model = define_model()
layer_names = [layer.name for layer in model.layers]
layer_names_without_input_and_output = layer_names[1:-2]
for layer in layer_names_without_input_and_output:
    layer_name = layer
    x = 0 # Spatial position x
    y = 0 # Spatial position y
    c = 0 # Channel index

    inp = np.random.uniform(0,1,size=(1,128,128,1)) # Input tensor
    inp = tf.convert_to_tensor(inp, dtype=tf.float32)

    gradient = get_gradient(model, layer_name, x, y, c, inp)
    gradient = gradient * 10000
    size_of_receptive_field = np.max(np.nonzero(gradient)) + 1
    print(f'for the layer {layer}, the receptive field is {size_of_receptive_field}x{size_of_receptive_field}')
    # Convert gradient to image
    image = Image.fromarray(gradient.astype('uint8'))

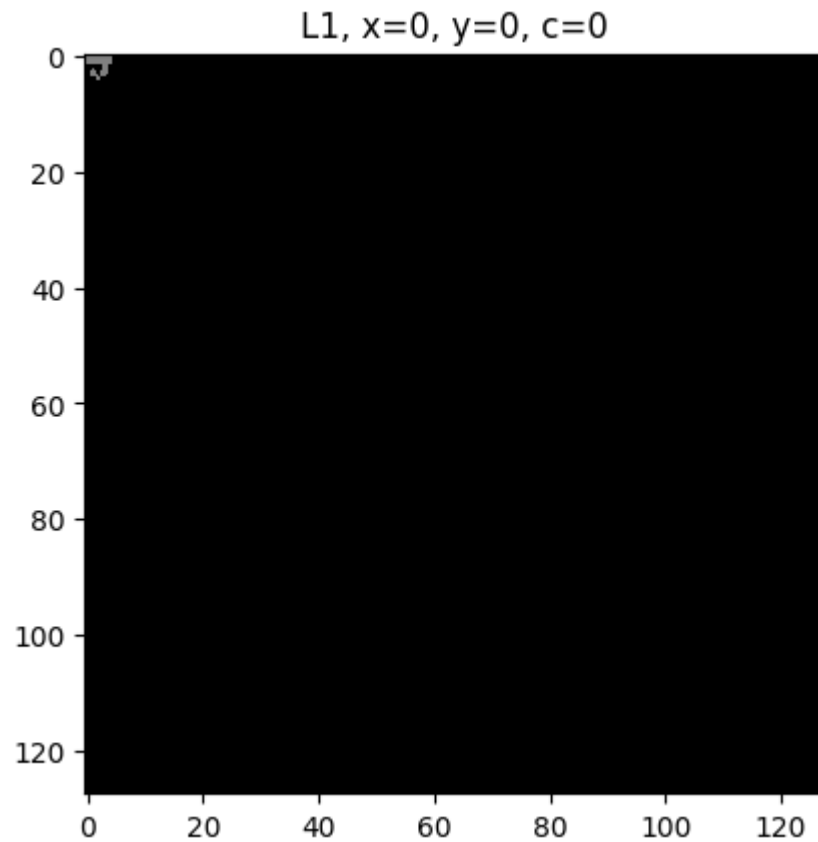
    # Define your custom black and blue color palette

```

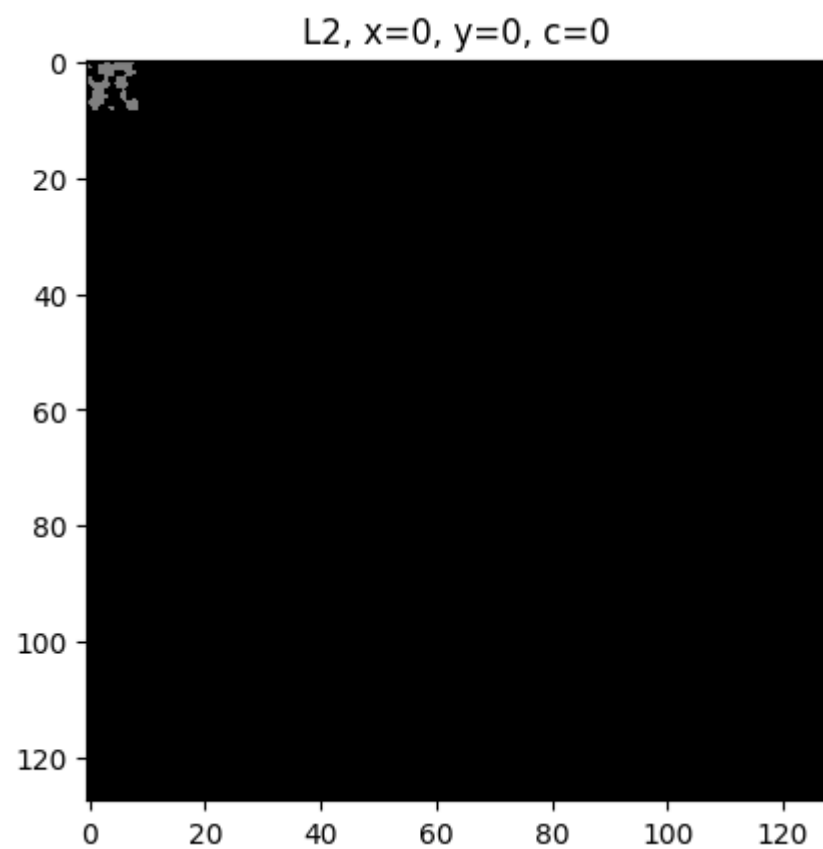
```
colors = ['black', 'grey']
custom_cmap = ListedColormap(colors)

# Display the sample data using the custom color palette
plt.title(f'{layer}, x={x}, y={y}, c={c}')
plt.imshow(image, cmap=custom_cmap)
# plt.colorbar() # Add a color bar for reference
plt.show()
```

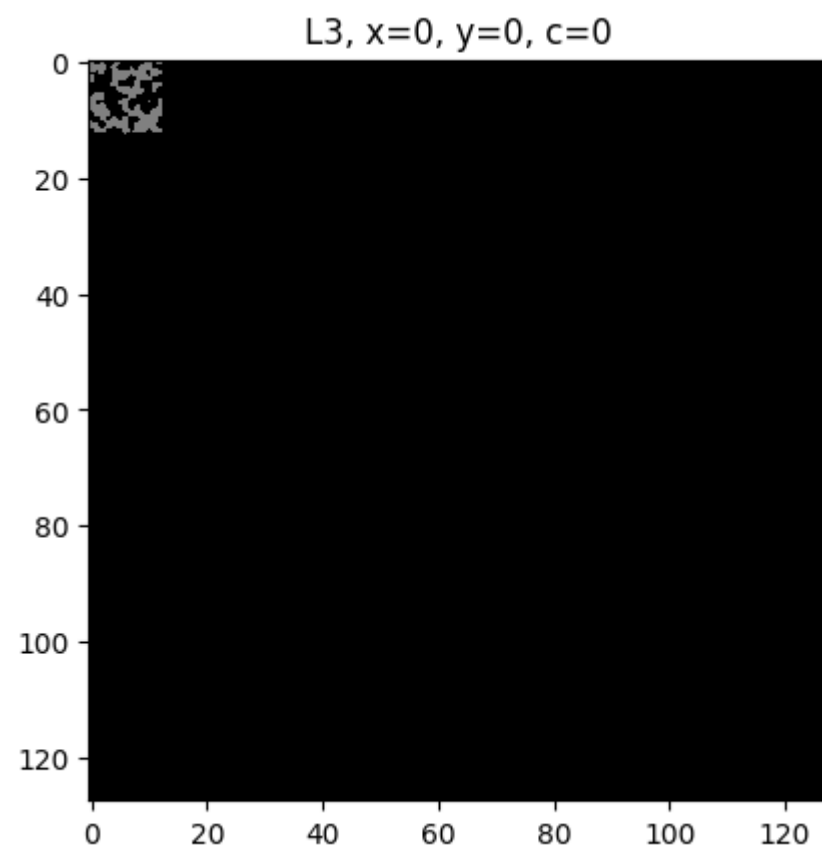
for the layer L1, the receptive field is 5x5



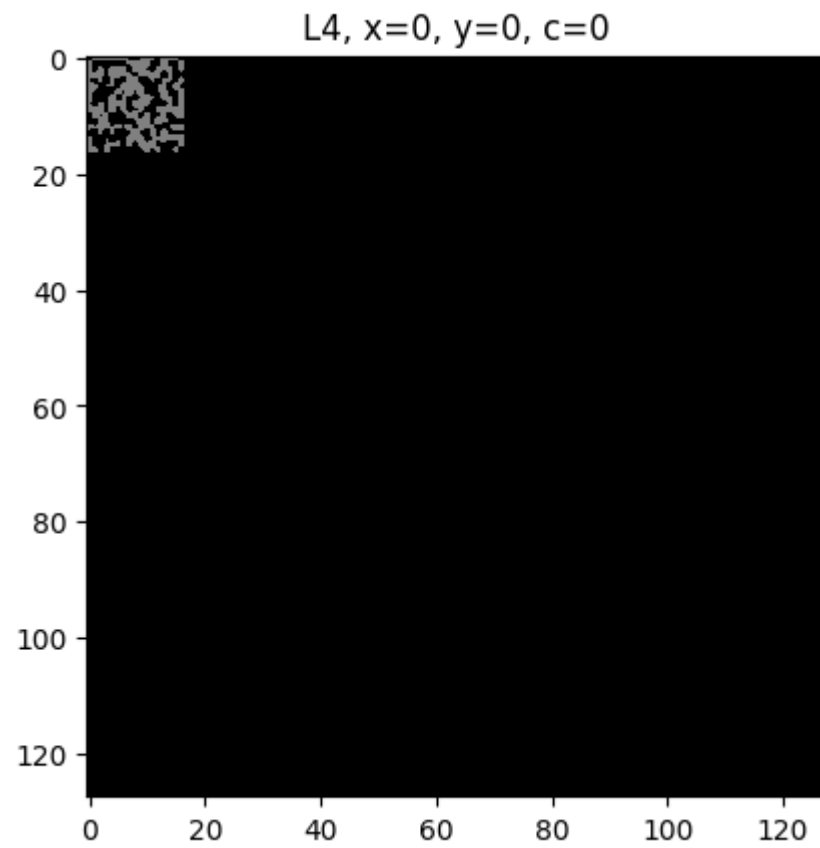
for the layer L2, the receptive field is 9x9



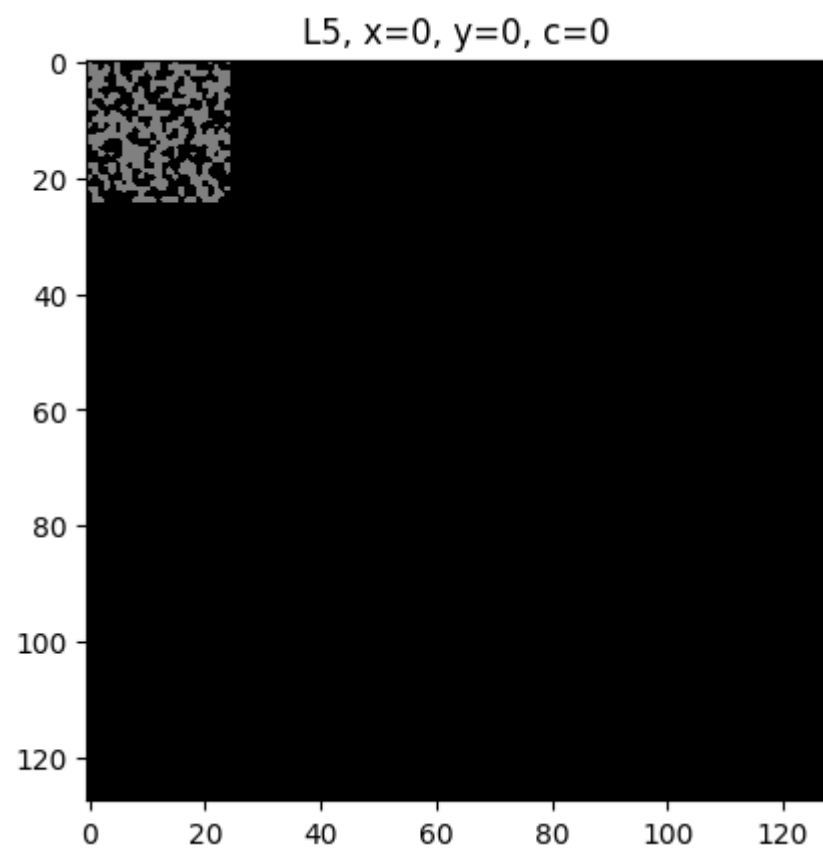
for the layer L3, the receptive field is 13x13



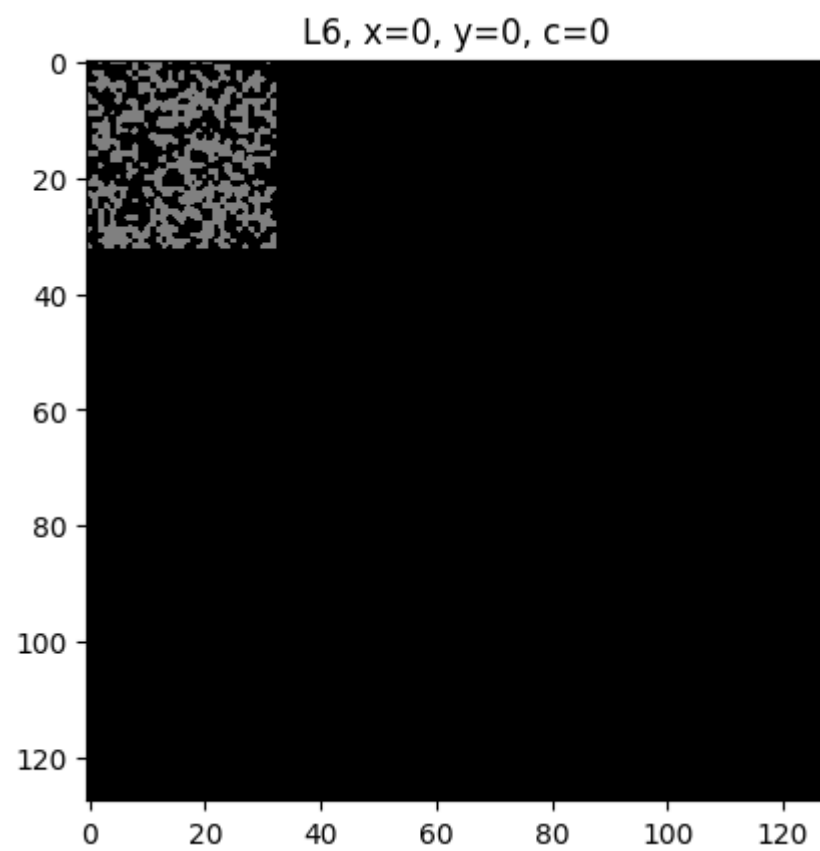
for the layer L4, the receptive field is 17x17



for the layer L5, the receptive field is 25x25



for the layer L6, the receptive field is 33x33



In []: