```
from tensorflow import keras
# model = keras.models.load_model("convnet_from_scratch_with_augmentation.keras")
model = keras.models.load_model("oxford_segmentation.keras")
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 200, 200, 3)]     0

 rescaling (Rescaling)       (None, 200, 200, 3)       0

 conv2d (Conv2D)             (None, 100, 100, 64)      1792

 conv2d_1 (Conv2D)           (None, 100, 100, 64)      36928

 conv2d_2 (Conv2D)           (None, 50, 50, 128)       73856

 conv2d_3 (Conv2D)           (None, 50, 50, 128)       147584

 conv2d_4 (Conv2D)           (None, 25, 25, 256)       295168

 conv2d_5 (Conv2D)           (None, 25, 25, 256)       590080

 conv2d_transpose (Conv2DTra  (None, 25, 25, 256)      590080
 nspose)

 conv2d_transpose_1 (Conv2DT  (None, 50, 50, 256)      590080
 ranspose)

 conv2d_transpose_2 (Conv2DT  (None, 50, 50, 128)      295040
 ranspose)

 conv2d_transpose_3 (Conv2DT  (None, 100, 100, 128)    147584
 ranspose)

 conv2d_transpose_4 (Conv2DT  (None, 100, 100, 64)     73792
 ranspose)

 conv2d_transpose_5 (Conv2DT  (None, 200, 200, 64)     36928
 ranspose)

 conv2d_6 (Conv2D)           (None, 200, 200, 3)       1731

=================================================================
Total params: 2,880,643
Trainable params: 2,880,643
Non-trainable params: 0
_________________________________________________________________
```

```
from tensorflow import keras
import numpy as np

img_path = keras.utils.get_file(
    fname="cat.jpg",
    origin="https://img-datasets.s3.amazonaws.com/cat.jpg")

def get_img_array(img_path, target_size):
    img = keras.utils.load_img(
        img_path, target_size=target_size)
    array = keras.utils.img_to_array(img)
    array = np.expand_dims(array, axis=0)
    return array

img_tensor = get_img_array(img_path, target_size=(180, 180))
```

```
Downloading data from https://img-datasets.s3.amazonaws.com/cat.jpg
80329/80329 [==============================] - 0s 5us/step
```

Displaying the test picture

```
import matplotlib.pyplot as plt
plt.axis("off")
plt.imshow(img_tensor[0].astype("uint8"))
plt.show()
```

```
from tensorflow.keras import layers

layer_outputs = []
layer_names = []
for layer in model.layers:
    if isinstance(layer, (layers.Conv2D, layers.MaxPooling2D)):
        layer_outputs.append(layer.output)
        layer_names.append(layer.name)
activation_model = keras.Model(inputs=model.input, outputs=layer_outputs)
```

Using the model to compute layer activations

```
activations = activation_model.predict(img_tensor)
```

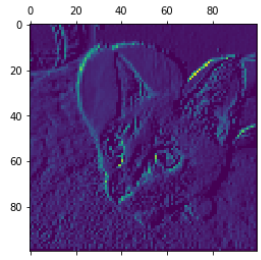
```
WARNING:tensorflow:5 out of the last 5 calls to .predict_function at 0x0000025D0EA14790> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 [==============================] - 0s 150ms/step
```

```
first_layer_activation = activations[0]
print(first_layer_activation.shape)
```

```
(1, 100, 100, 64)
```

Visualizing the fifth channel

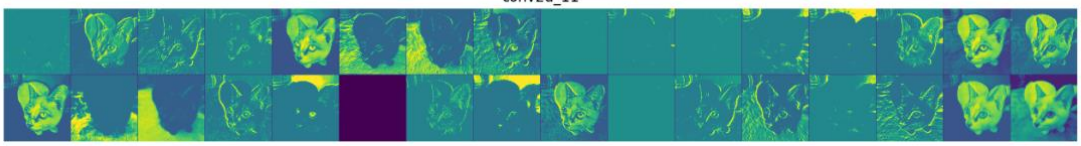
```
import matplotlib.pyplot as plt
plt.matshow(first_layer_activation[0, :, :, 5], cmap="viridis")
```



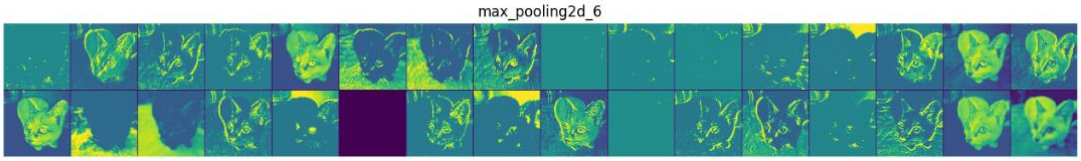
Visualizing every channel in every intermediate activation

```
images_per_row = 16
for layer_name, layer_activation in zip(layer_names, activations):
    n_features = layer_activation.shape[-1]
    size = layer_activation.shape[1]
    n_cols = n_features // images_per_row

    if n_cols==0:
        continue

    display_grid = np.zeros(((size + 1) * n_cols - 1,
                             images_per_row * (size + 1) - 1))
    for col in range(n_cols):
        for row in range(images_per_row):
            channel_index = col * images_per_row + row
            channel_image = layer_activation[0, :, :, channel_index].copy()
            if channel_image.sum() != 0:
                channel_image -= channel_image.mean()
                channel_image /= channel_image.std()
                channel_image *= 64
                channel_image += 128
            channel_image = np.clip(channel_image, 0, 255).astype("uint8")
            display_grid[
                col * (size + 1): (col + 1) * size + col,
                row * (size + 1) : (row + 1) * size + row] = channel_image
    scale = 1. / size
    plt.figure(figsize=(scale * display_grid.shape[1],
                        scale * display_grid.shape[0]))
    plt.title(layer_name)
    plt.grid(False)
    plt.axis("off")
    plt.imshow(display_grid, aspect="auto", cmap="viridis")
```

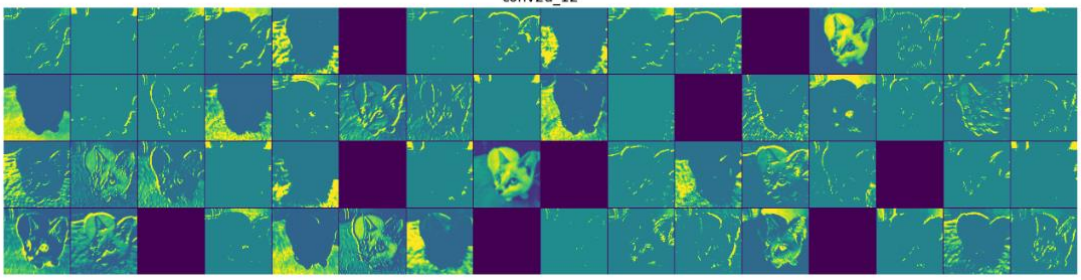
conv2d_11



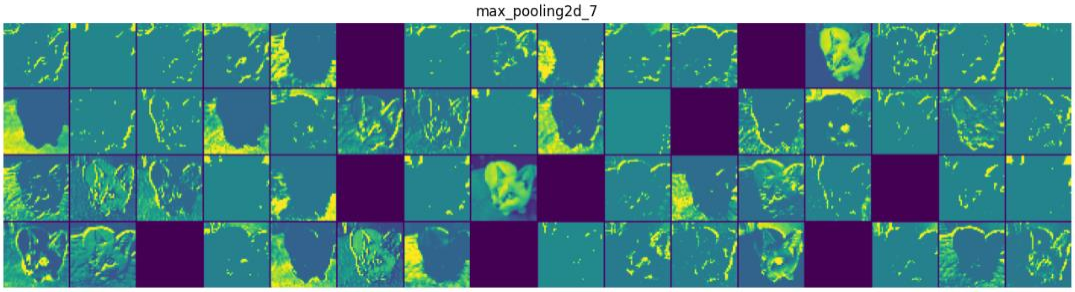
max_pooling2d_6



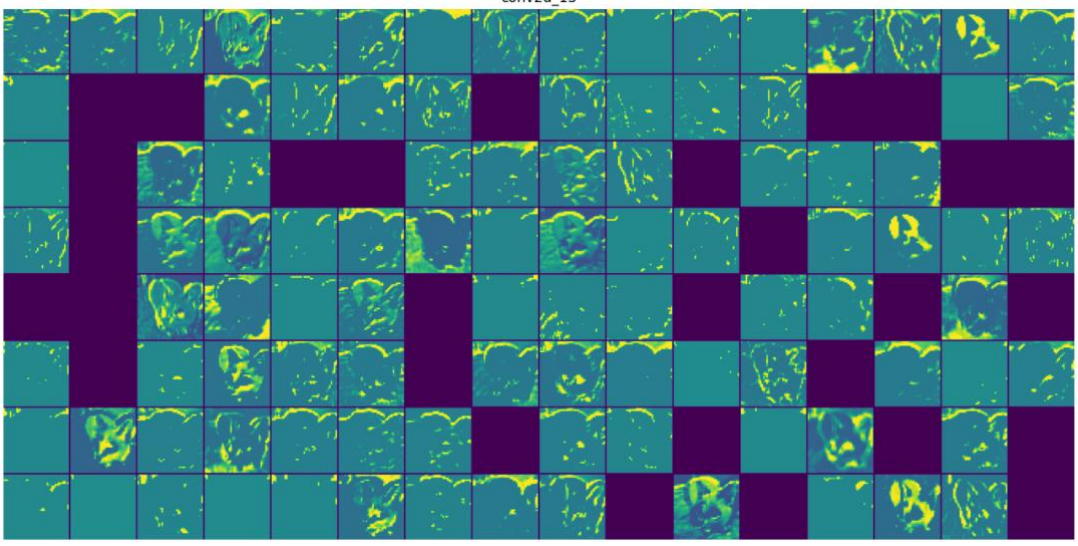
conv2d_12



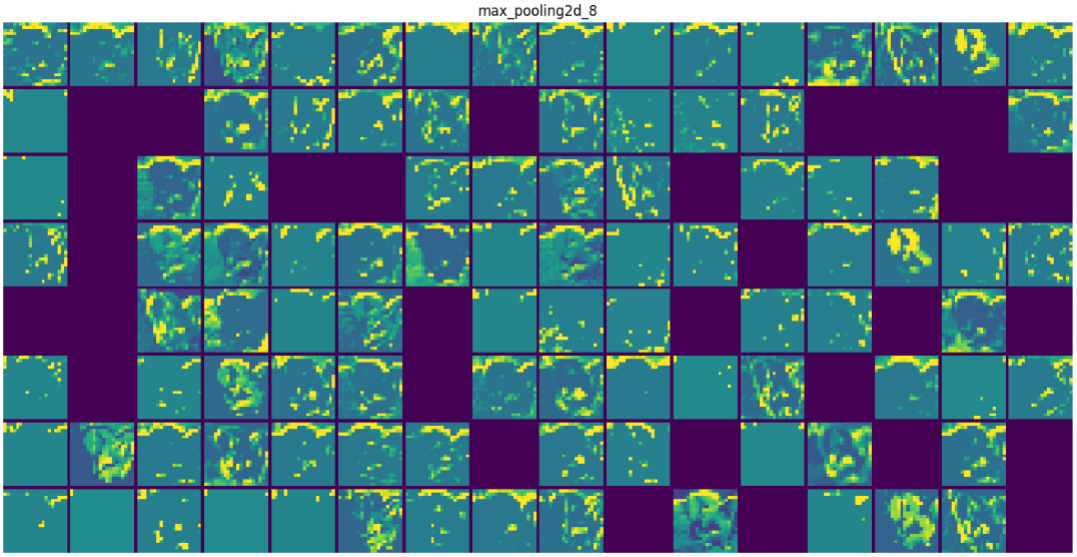
max_pooling2d_7



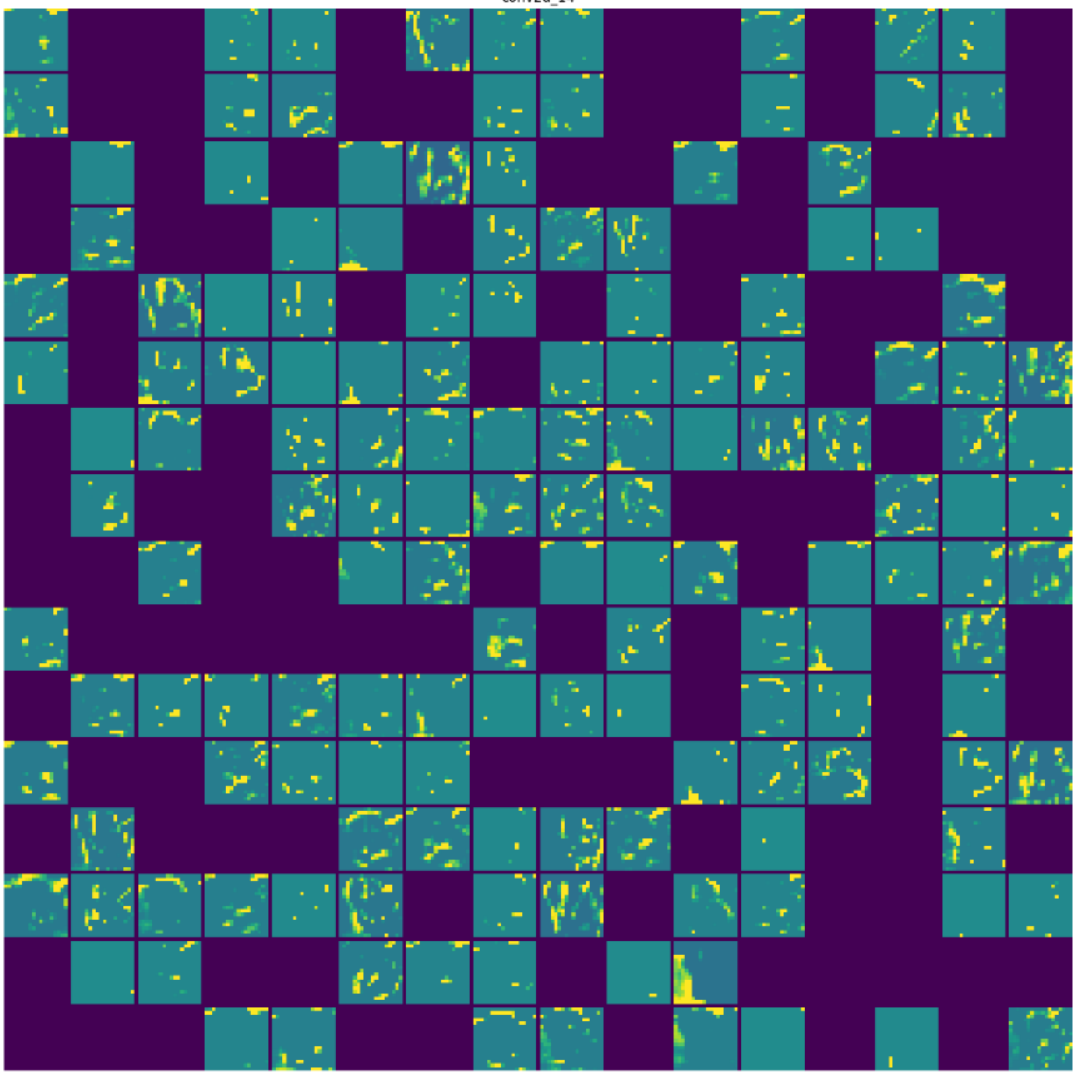
conv2d_13



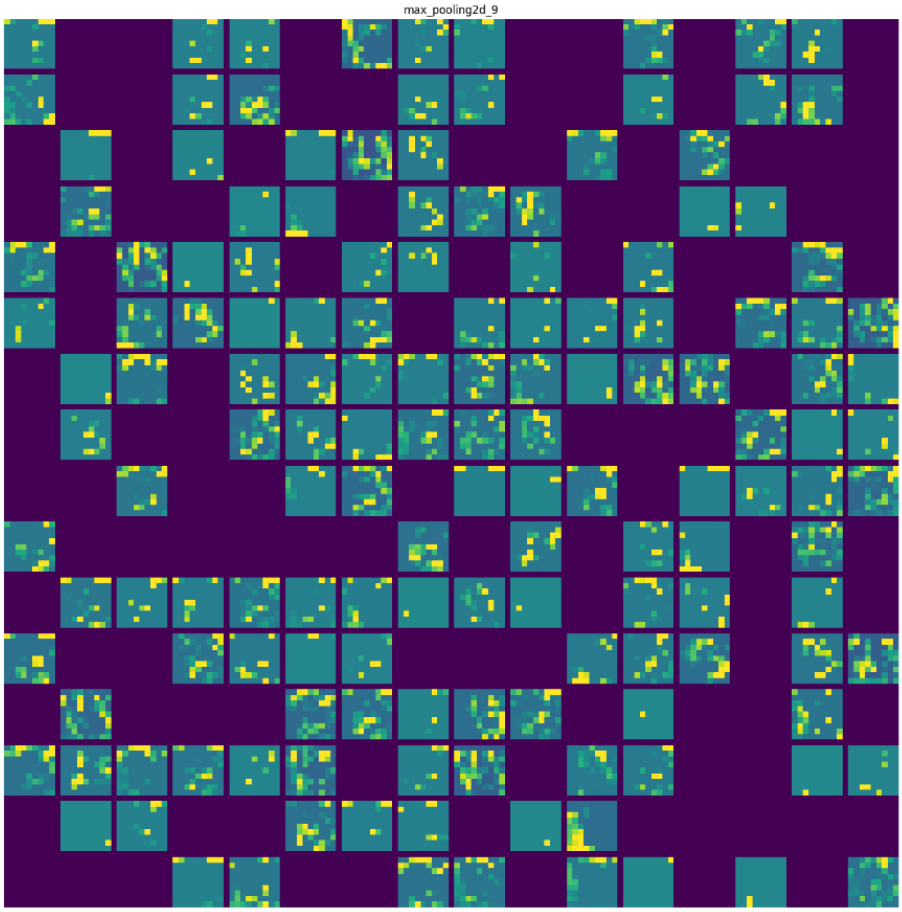
max_pooling2d_8



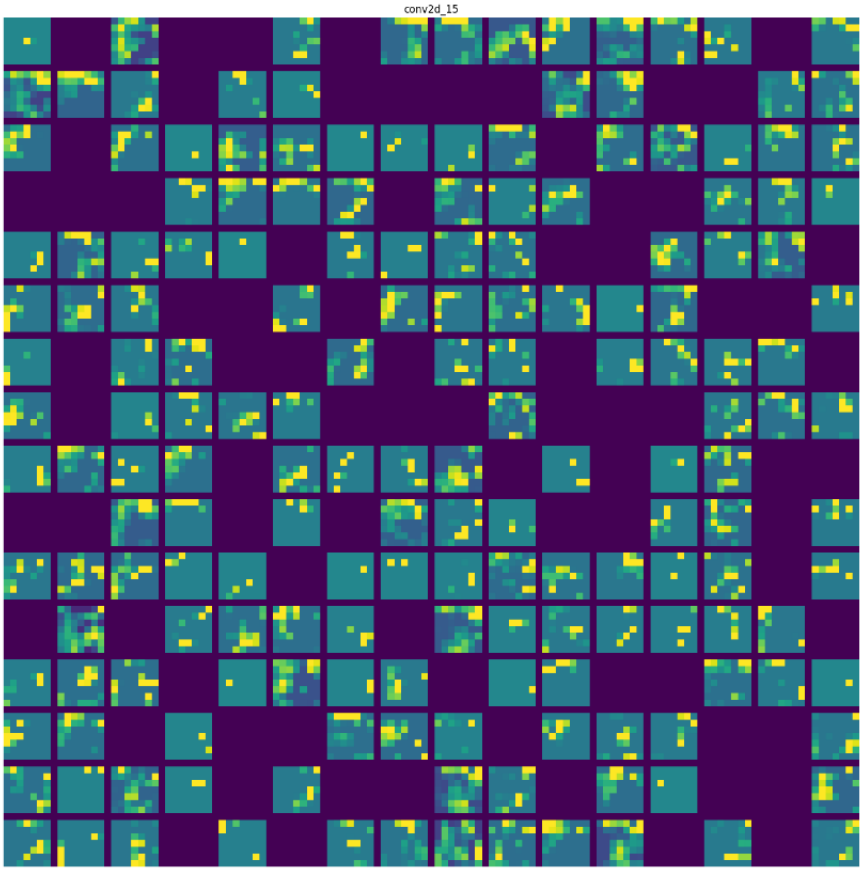
conv2d_14



max_pooling2d_9



conv2d_15



```
model = keras.applications.xception.Xception(
    weights="imagenet",
    include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83683744/83683744 [==============================] - 47s 1us/step
```

Printing the names of all convolutional layers in Xception

```
for layer in model.layers:
    if isinstance(layer, (keras.layers.Conv2D, keras.layers.SeparableConv2D)):
        print(layer.name)
```

```
block1_conv1
block1_conv2
block2_sepconv1
block2_sepconv2
conv2d
block3_sepconv1
block3_sepconv2
conv2d_1
block4_sepconv1
block4_sepconv2
conv2d_2
block5_sepconv1
block5_sepconv2
block5_sepconv3
block6_sepconv1
block6_sepconv2
block6_sepconv3
block7_sepconv1
block7_sepconv2
block7_sepconv3
block8_sepconv1
block8_sepconv2
block8_sepconv3
block9_sepconv1
block9_sepconv2
block9_sepconv3
block10_sepconv1
block10_sepconv2
block10_sepconv3
block11_sepconv1
block11_sepconv2
block11_sepconv3
block12_sepconv1
block12_sepconv2
block12_sepconv3
block13_sepconv1
block13_sepconv2
conv2d_3
block14_sepconv1
block14_sepconv2
```

Creating a feature extractor model

```
layer_name = "block3_sepconv1"
layer = model.get_layer(name=layer_name)
feature_extractor = keras.Model(inputs=model.input, outputs=layer.output)
```

Using the feature extractor

```
activation = feature_extractor(
    keras.applications.xception.preprocess_input(img_tensor)
)
```

```
import tensorflow as tf

def compute_loss(image, filter_index):
    activation = feature_extractor(image)
    filter_activation = activation[:, 2:-2, 2:-2, filter_index]
    return tf.reduce_mean(filter_activation)
```

Loss maximization via stochastic gradient ascent

```
@tf.function
def gradient_ascent_step(image, filter_index, learning_rate):
    with tf.GradientTape() as tape:
        tape.watch(image)
        loss = compute_loss(image, filter_index)
    grads = tape.gradient(loss, image)
    grads = tf.math.l2_normalize(grads)
    image += learning_rate * grads
    return image
```

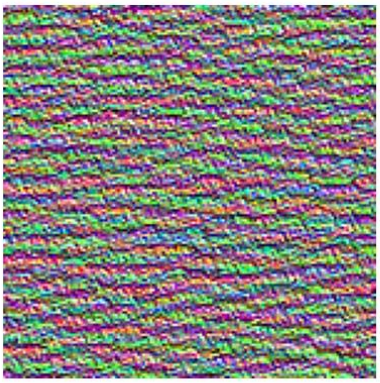
Function to generate filter visualizations

```
img_width = 200
img_height = 200

def generate_filter_pattern(filter_index):
    iterations = 30
    learning_rate = 10.
    image = tf.random.uniform(
        minval=0.4,
        maxval=0.6,
        shape=(1, img_width, img_height, 3))
    for i in range(iterations):
        image = gradient_ascent_step(image, filter_index, learning_rate)
    return image[0].numpy()
```

Utility function to convert a tensor into a valid image

```
def deprocess_image(image):
    image -= image.mean()
    image /= image.std()
    image *= 64
    image += 128
    image = np.clip(image, 0, 255).astype("uint8")
    image = image[25:-25, 25:-25, :]
    return image
```

```
plt.axis("off")
plt.imshow(deprocess_image(generate_filter_pattern(filter_index=2)))
```



Generating a grid of all filter response patterns in a layer

```
all_images = []
for filter_index in range(64):
    print(f"Processing filter {filter_index}")
    image = deprocess_image(
        generate_filter_pattern(filter_index)
    )
    all_images.append(image)

margin = 5
n = 8
cropped_width = img_width - 25 * 2
cropped_height = img_height - 25 * 2
width = n * cropped_width + (n - 1) * margin
height = n * cropped_height + (n - 1) * margin
stitched_filters = np.zeros((width, height, 3))

for i in range(n):
    for j in range(n):
        image = all_images[i * n + j]
        stitched_filters[
            (cropped_width + margin) * i : (cropped_width + margin) * i + cropped_width,
            (cropped_height + margin) * j : (cropped_height + margin) * j
            + cropped_height,
            :,
        ] = image

keras.utils.save_img(
    f"filters_for_layer_{layer_name}.png", stitched_filters)
```

```
Processing filter 0
Processing filter 1
Processing filter 2
Processing filter 3
Processing filter 4
Processing filter 5
Processing filter 6
Processing filter 7
Processing filter 8
Processing filter 9
Processing filter 10
Processing filter 11
Processing filter 12
Processing filter 13
Processing filter 14
Processing filter 15
Processing filter 16
Processing filter 17
Processing filter 18
Processing filter 19
Processing filter 20
Processing filter 21
Processing filter 22
Processing filter 23
Processing filter 24
Processing filter 25
Processing filter 26
Processing filter 27
Processing filter 28
Processing filter 29
Processing filter 30
Processing filter 31
Processing filter 32
Processing filter 33
Processing filter 34
Processing filter 35
Processing filter 36
Processing filter 37
Processing filter 38
Processing filter 39
Processing filter 40
Processing filter 41
Processing filter 42
Processing filter 43
Processing filter 44
Processing filter 45
Processing filter 46
Processing filter 47
Processing filter 48
Processing filter 49
Processing filter 50
Processing filter 51
Processing filter 52
Processing filter 53
Processing filter 54
Processing filter 55
Processing filter 56
Processing filter 57
Processing filter 58
Processing filter 59
Processing filter 60
Processing filter 61
Processing filter 62
Processing filter 63
```

Visualizing heatmaps of class activation

Loading the Xception network with pretrained weights

```
model = keras.applications.xception.Xception(weights="imagenet")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kernels.h5
91884032/91884032 [==============================] - 8s 0us/step
```

Preprocessing an input image for Xception

```
img_path = keras.utils.get_file(
    fname="elephant.jpg",
    origin="https://img-datasets.s3.amazonaws.com/elephant.jpg")

def get_img_array(img_path, target_size):
    img = keras.utils.load_img(img_path, target_size=target_size)
    array = keras.utils.img_to_array(img)
    array = np.expand_dims(array, axis=0)
    array = keras.applications.xception.preprocess_input(array)
    return array

img_array = get_img_array(img_path, target_size=(299, 299))
```

```
Downloading data from https://img-datasets.s3.amazonaws.com/elephant.jpg
733657/733657 [==============================] - 1s 1us/step
```

```
preds = model.predict(img_array)
print(keras.applications.xception.decode_predictions(preds, top=3)[0])
```

```
1/1 [==============================] - 1s 502ms/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
35363/35363 [==============================] - 0s 0us/step
[('n02504458', 'African_elephant', 0.8699397), ('n01871265', 'tusker', 0.07695623), ('n02504013', 'Indian_elephant', 0.023541799)]
```

```
np.argmax(preds[0])
```

386

Setting up a model that returns the last convolutional output

```
last_conv_layer_name = "block14_sepconv2_act"
classifier_layer_names = [
    "avg_pool",
    "predictions",
]
last_conv_layer = model.get_layer(last_conv_layer_name)
last_conv_layer_model = keras.Model(model.inputs, last_conv_layer.output)
```

Reapplying the classifier on top of the last convolutional output

```
classifier_input = keras.Input(shape=last_conv_layer.output.shape[1:])
x = classifier_input
for layer_name in classifier_layer_names:
    x = model.get_layer(layer_name)(x)
classifier_model = keras.Model(classifier_input, x)
```

Retrieving the gradients of the top predicted class

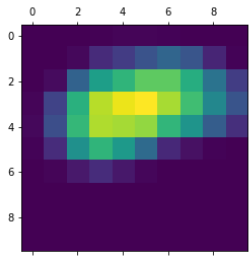
```
import tensorflow as tf

with tf.GradientTape() as tape:
    last_conv_layer_output = last_conv_layer_model(img_array)
    tape.watch(last_conv_layer_output)
    preds = classifier_model(last_conv_layer_output)
    top_pred_index = tf.argmax(preds[0])
    top_class_channel = preds[:, top_pred_index]

grads = tape.gradient(top_class_channel, last_conv_layer_output)
```

Gradient pooling and channel-importance weighting

```
pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2)).numpy()
last_conv_layer_output = last_conv_layer_output.numpy()[0]
for i in range(pooled_grads.shape[-1]):
    last_conv_layer_output[:, :, i] *= pooled_grads[i]
heatmap = np.mean(last_conv_layer_output, axis=-1)
```

Heatmap post-processing

```
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)
```



Superimposing the heatmap on the original picture

```
import matplotlib.cm as cm

img = keras.utils.load_img(img_path)
img = keras.utils.img_to_array(img)

heatmap = np.uint8(255 * heatmap)

jet = cm.get_cmap("jet")
jet_colors = jet(np.arange(256))[:, :3]
jet_heatmap = jet_colors[heatmap]

jet_heatmap = keras.utils.array_to_img(jet_heatmap)
jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
jet_heatmap = keras.utils.img_to_array(jet_heatmap)

superimposed_img = jet_heatmap * 0.4 + img
superimposed_img = keras.utils.array_to_img(superimposed_img)

save_path = "elephant_cam.jpg"
superimposed_img.save(save_path)
```