In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
import keras
import numpy as np
import tensorflow as tf
```

```
Mounted at /content/drive
```

Visualizing CNN model initial convolutional layer

In [2]:
```
# The dimensions of our input image
img_width = 200
img_height = 200
# Our target layer: we will visualize the filters from this layer.
# See `model.summary()` for list of layer names, if you want to change this.
model = keras.saving.load_model('/content/drive/MyDrive/Colab Notebooks/Automa
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 197, 197, 32)      1568

 flatten (Flatten)           (None, 1241888)           0

 dense (Dense)               (None, 32)                39740448

 dense_1 (Dense)             (None, 2)                 66

=================================================================
Total params: 39742082 (151.60 MB)
Trainable params: 39742082 (151.60 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [3]:
```
layer_name = "conv2d"
```

In [4]:
```
# Set up a model that returns the activation values for our target layer
layer = model.get_layer(name=layer_name)
feature_extractor = keras.Model(inputs=model.inputs, outputs=layer.output)
```

In [5]:
```
def compute_loss(input_image, filter_index):
    activation = feature_extractor(input_image)
    # We avoid border artifacts by only involving non-border pixels in the loss
    filter_activation = activation[:, 2:-2, 2:-2, filter_index]
    return tf.reduce_mean(filter_activation)
```

In [6]:
```
@tf.function
def gradient_ascent_step(img, filter_index, learning_rate):
    with tf.GradientTape() as tape:
        tape.watch(img)
        loss = compute_loss(img, filter_index)
    # Compute gradients.
    grads = tape.gradient(loss, img)
    # Normalize gradients.
    grads = tf.math.l2_normalize(grads)
```

```
    img += learning_rate * grads
    return loss, img
```

In [7]:
```
def initialize_image():
    # We start from a gray image with some random noise
    img = tf.random.uniform((1, img_width, img_height, 3))
    # ResNet50V2 expects inputs in the range [-1, +1].
    # Here we scale our random inputs to [-0.125, +0.125]
    return (img - 0.5) * 0.25


def visualize_filter(filter_index):
    # We run gradient ascent for 20 steps
    iterations = 30
    learning_rate = 10.0
    img = initialize_image()
    for iteration in range(iterations):
        loss, img = gradient_ascent_step(img, filter_index, learning_rate)

    # Decode the resulting input image
    img = deprocess_image(img[0].numpy())
    return loss, img


def deprocess_image(img):
    # Normalize array: center on 0., ensure variance is 0.15
    img -= img.mean()
    img /= img.std() + 1e-5
    img *= 0.15

    # Center crop
    img = img[25:-25, 25:-25, :]

    # Clip to [0, 1]
    img += 0.5
    img = np.clip(img, 0, 1)

    # Convert to RGB array
    img *= 255
    img = np.clip(img, 0, 255).astype("uint8")
    return img
```

In [8]:
```
from IPython.display import Image, display

loss, img = visualize_filter(0)
keras.utils.save_img("0.png", img)
```

In [9]:
```
# Compute image inputs that maximize per-filter activations
# for the first 64 filters of our target layer
all_imgs = []
for filter_index in range(10):
    print("Processing filter %d" % (filter_index,))
    loss, img = visualize_filter(filter_index)
    all_imgs.append(img)

# Build a black picture with enough space for
# our 8 x 8 filters of size 128 x 128, with a 5px margin in between
margin = 5
n_height = 5
```

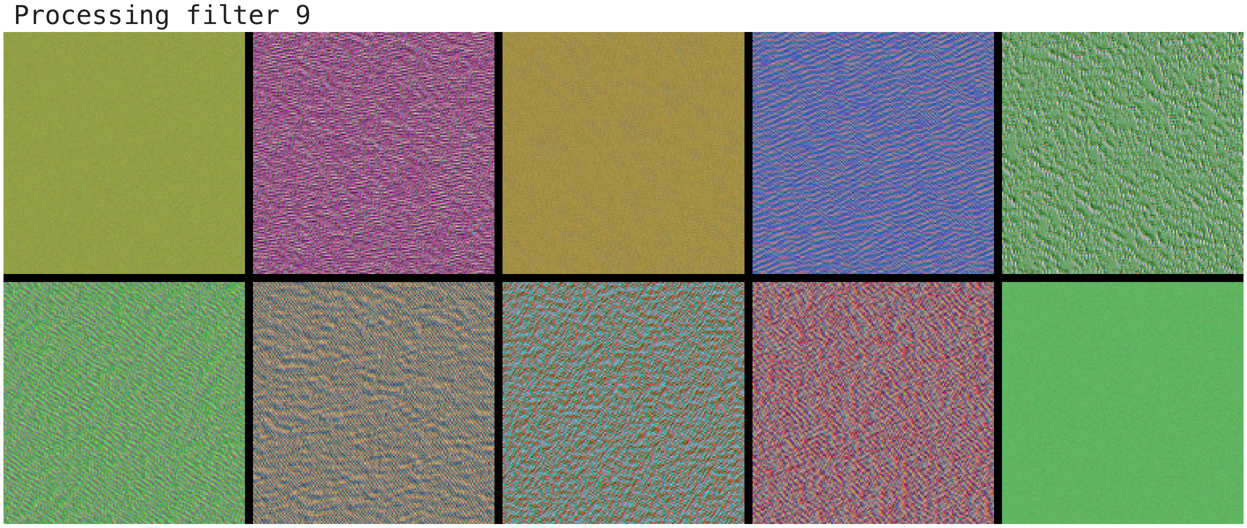
```
n_width = 2
cropped_width = img_width - 25 * 2
cropped_height = img_height - 25 * 2
width = n_width * cropped_width + (n_width - 1) * margin
height = n_height * cropped_height + (n_height - 1) * margin
stitched_filters = np.zeros((width, height, 3))

# Fill the picture with our saved filters
for i in range(n_width):
    for j in range(n_height):
        img = all_imgs[i * n_height + j]
        stitched_filters[
            (cropped_width + margin) * i : (cropped_width + margin) * i + cropp
            (cropped_height + margin) * j : (cropped_height + margin) * j
            + cropped_height,
            :,
        ] = img
keras.utils.save_img("stiched_filters.png", stitched_filters)

from IPython.display import Image, display

display(Image("stiched_filters.png"))
```

```
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
```



In [10]:
```
from google.colab import drive
drive.mount('/content/drive')
!jupyter nbconvert --to html "/content/drive/MyDrive/Colab Notebooks/Automated
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab Notebooks/Auto
mated Decision Systems/visualization.ipynb to html
[NbConvertApp] Writing 596769 bytes to /content/drive/MyDrive/Colab Notebooks/
Automated Decision Systems/visualization.html
```