In [3]:
```
# CSC 732 Final Part 1 Problem 2 Relu Softmax Adam Test Case
# Dominic Klusek, Jonathan Rozen
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:98% !important; }</style>"))
```

Dataset Information

Beginner set of 16,000 custom images for categorizing polyhedral dice. Polyhedral dice vary in color, and images are taken at different angles over different background types: blank, colored, and wood. All images were created, edited, and sorted by Mario Lurig. Fixed camera positions (minimum 2 angles) used to capture video on a rotating platform with two white lights Minimum 5 different dice used on 6 different backgrounds (white and various colors) Video was then exported as images and then batch cropped to 480x480 Handheld camera moved over 5+ dice on various wood surfaces (minimum 2) using natural lighting Video edited and exported to images then batch cropped to 480x480 Images that were partially out of crop were manually removed

- **Number of Instances:** 16,000
- **Number of Classes:** 6
- **Image Dimensions:** 480x480
- **Number of Channels:** 3

In [4]:
```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers.core import Flatten, Dropout, Dense, Activation
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from keras import backend as K
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import pickle
```

In [5]:
```
# set some hyper parameters for training
batch_size = 32
num_classes = 6
num_epochs = 40
```

In [6]:
```
train_dir = 'Datasets/Dice/train/'
test_dir = 'Datasets/Dice/test/'
validation_dir = 'Datasets/Dice/val/'
```

In [7]:
```
# build the model
model = Sequential()
model.add(Conv2D(32, (3, 3), kernel_initializer='glorot_uniform', input
_shape=(150, 150, 3)))
#model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(rate=0.2))
model.add(Conv2D(64, (3, 3), kernel_initializer='glorot_uniform'))
#model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(rate=0.2))
model.add(Conv2D(128, (3, 3), kernel_initializer='glorot_uniform'))
#model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(rate=0.2))
model.add(Conv2D(128, (3, 3), kernel_initializer='glorot_uniform'))
#model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(rate=0.2))
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_initializer='glorot_unif
orm', use_bias=False))
model.add(BatchNormalization())
model.add(Dropout(rate=0.2))
model.add(Dense(6, activation='softmax', kernel_initializer='glorot_uni
form'))
```

In [8]:
```
# compile the model to use binary crossentropy and use RMSProp optimize
r
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=
['acc'])
```

In [9]:
```
# create image generators
# each using the images from their respective directories
train_datagen = ImageDataGenerator(
rescale=1./255.,
rotation_range=40,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.2,
zoom_range=0.2)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
train_dir,
target_size=(150, 150),
batch_size=batch_size,
class_mode='categorical')

test_generator = train_datagen.flow_from_directory(
test_dir,
target_size=(150, 150),
batch_size=batch_size,
class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
validation_dir,
target_size=(150, 150),
batch_size=batch_size,
class_mode='categorical')
```

```
Found 6600 images belonging to 6 classes.
Found 1200 images belonging to 6 classes.
Found 1800 images belonging to 6 classes.
```

In [16]:
```
# fit the model to the image created by the generators
history = model.fit_generator(
train_generator,
steps_per_epoch=100,
epochs=num_epochs,
validation_data=validation_generator,
validation_steps=100, callbacks=[keras.callbacks.ModelCheckpoint('best_
model_less.h5', save_best_only=True)])
```

```
Epoch 1/40
100/100 [==============================] - 69s 691ms/step - loss: 1.9132 - acc: 0.2465 - val_loss:
1.9256 - val_acc: 0.1662
Epoch 2/40
100/100 [==============================] - 63s 635ms/step - loss: 1.6551 - acc: 0.3350 - val_loss:
1.6538 - val_acc: 0.2725
Epoch 3/40
100/100 [==============================] - 30s 304ms/step - loss: 1.4068 - acc: 0.4307 - val_loss:
1.8535 - val_acc: 0.3033
Epoch 4/40
100/100 [==============================] - 29s 292ms/step - loss: 1.3154 - acc: 0.4791 - val_loss:
1.2651 - val_acc: 0.5289
Epoch 5/40
100/100 [==============================] - 30s 298ms/step - loss: 1.0694 - acc: 0.5828 - val_loss:
1.3178 - val_acc: 0.5120
Epoch 6/40
100/100 [==============================] - 29s 294ms/step - loss: 0.9052 - acc: 0.6527 - val_loss:
1.4970 - val_acc: 0.4676
Epoch 7/40
100/100 [==============================] - 29s 288ms/step - loss: 0.7718 - acc: 0.7185 - val_loss:
0.9255 - val_acc: 0.6396
Epoch 8/40
100/100 [==============================] - 29s 295ms/step - loss: 0.6786 - acc: 0.7487 - val_loss:
0.4575 - val_acc: 0.7703
Epoch 9/40
100/100 [==============================] - 29s 290ms/step - loss: 0.5811 - acc: 0.7784 - val_loss:
1.2711 - val_acc: 0.7009
Epoch 10/40
100/100 [==============================] - 30s 300ms/step - loss: 0.4980 - acc: 0.8149 - val_loss:
0.6706 - val_acc: 0.7430
Epoch 11/40
100/100 [==============================] - 28s 282ms/step - loss: 0.4550 - acc: 0.8338 - val_loss:
0.5862 - val_acc: 0.8328
Epoch 12/40
100/100 [==============================] - 27s 272ms/step - loss: 0.3921 - acc: 0.8609 - val_loss:
0.6823 - val_acc: 0.8553
Epoch 13/40
100/100 [==============================] - 27s 271ms/step - loss: 0.3130 - acc: 0.8942 - val_loss:
0.2195 - val_acc: 0.8898
Epoch 14/40
100/100 [==============================] - 28s 275ms/step - loss: 0.2849 - acc: 0.8966 - val_loss:
0.4177 - val_acc: 0.8725
Epoch 15/40
100/100 [==============================] - 27s 272ms/step - loss: 0.2548 - acc: 0.9078 - val_loss:
0.4719 - val_acc: 0.8959
Epoch 16/40
100/100 [==============================] - 27s 270ms/step - loss: 0.2914 - acc: 0.9059 - val_loss:
0.3769 - val_acc: 0.8912
Epoch 17/40
100/100 [==============================] - 27s 275ms/step - loss: 0.2081 - acc: 0.9284 - val_loss:
0.2444 - val_acc: 0.9181
Epoch 18/40
100/100 [==============================] - 27s 272ms/step - loss: 0.2124 - acc: 0.9281 - val_loss:
0.1637 - val_acc: 0.8725
Epoch 19/40
100/100 [==============================] - 28s 277ms/step - loss: 0.1891 - acc: 0.9355 - val_loss:
0.1588 - val_acc: 0.9334
Epoch 20/40
100/100 [==============================] - 30s 298ms/step - loss: 0.1812 - acc: 0.9403 - val_loss:
0.1133 - val_acc: 0.9410
Epoch 21/40
100/100 [==============================] - 29s 293ms/step - loss: 0.1494 - acc: 0.9509 - val_loss:
0.1178 - val_acc: 0.9408
```

In [17]:

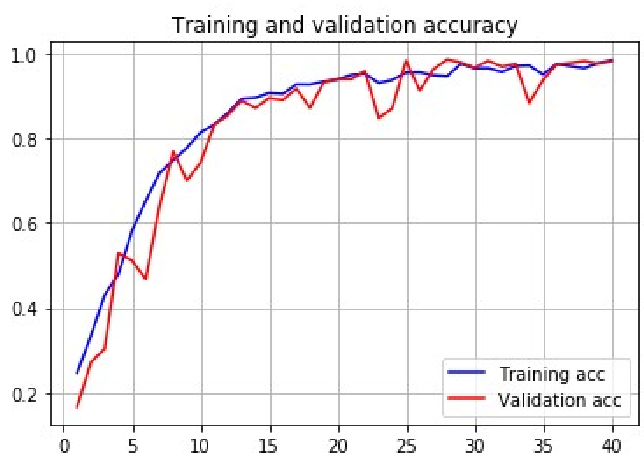
```
# save the model for future use
model.save('dice_relu_softmax_Adam_less_steps.h5')
```

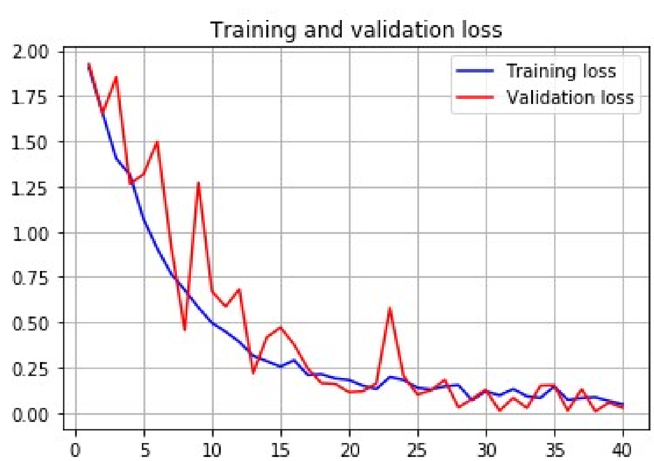
In [19]:

```
#model.load_weights('cats_and_dogs_relu_sigmoid_Adam.h5')
# Evaluate classifier with test image generator
evaluation = model.evaluate_generator(test_generator, steps=400, verbos
e=1)
print("Test Loss: %.4f" % evaluation[0])
print("Test Accuracy: %.4f" % evaluation[1])
```

```
400/400 [==============================] - 104s 259ms/step
Test Loss: 0.1644
Test Accuracy: 0.9777
```

In [20]:
```
# plot accuracy and loss plots
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.figure(facecolor='w')
plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.grid()
plt.figure(facecolor='w')
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.grid()
plt.show()
```





Visual Activation Maps

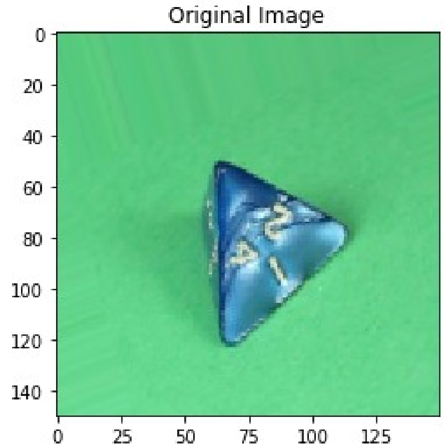
In [10]:
```
from keras.preprocessing import image
import numpy as np

#create list of randomly augmented images of dice
dice_images = []

for i in range(2):
    img = image.load_img("val/d4/d4_angle_color015.jpg", target_size=(1
50,150))
    img_tensor = image.img_to_array(img)
    img_tensor = train_datagen.apply_transform(img_tensor, train_datage
n.get_random_transform(img_tensor.shape))
    img_tensor = np.expand_dims(img_tensor,axis=0)
    img_tensor/=255
    dice_images.append(img_tensor)

for i in range(2):
    img = image.load_img("val/d6/d6_off-angle_181.jpg", target_size=(15
0,150))
    img_tensor = image.img_to_array(img)
    img_tensor = train_datagen.apply_transform(img_tensor, train_datage
n.get_random_transform(img_tensor.shape))
    img_tensor = np.expand_dims(img_tensor,axis=0)
    img_tensor/=255
    dice_images.append(img_tensor)

for i in range(2):
    img = image.load_img("val/d8/d8_wood940.jpg", target_size=(150,15
0))
    img_tensor = image.img_to_array(img)
    img_tensor = train_datagen.apply_transform(img_tensor, train_datage
n.get_random_transform(img_tensor.shape))
    img_tensor = np.expand_dims(img_tensor,axis=0)
    img_tensor/=255
    dice_images.append(img_tensor)

for i in range(2):
    img = image.load_img("val/d12/d12_top202.jpg", target_size=(150,15
0))
    img_tensor = image.img_to_array(img)
    img_tensor = train_datagen.apply_transform(img_tensor, train_datage
n.get_random_transform(img_tensor.shape))
    img_tensor = np.expand_dims(img_tensor,axis=0)
    img_tensor/=255
    dice_images.append(img_tensor)

for i in range(2):
    img = image.load_img("val/d20/d20_color498.jpg", target_size=(150,1
```

In [16]:
```
model = models.load_model('dice_relu_softmax_Adam_less_steps.h5')
```

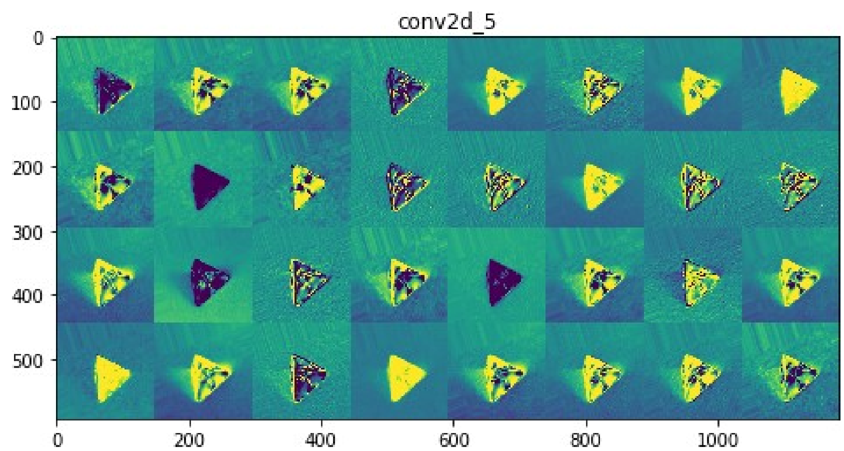
In [17]:
```
from keras import models
layer_outputs = [layer.output for layer in model.layers]
activation_model = models.Model(inputs=model.input, outputs=layer_outpu
ts)
```

In [18]:

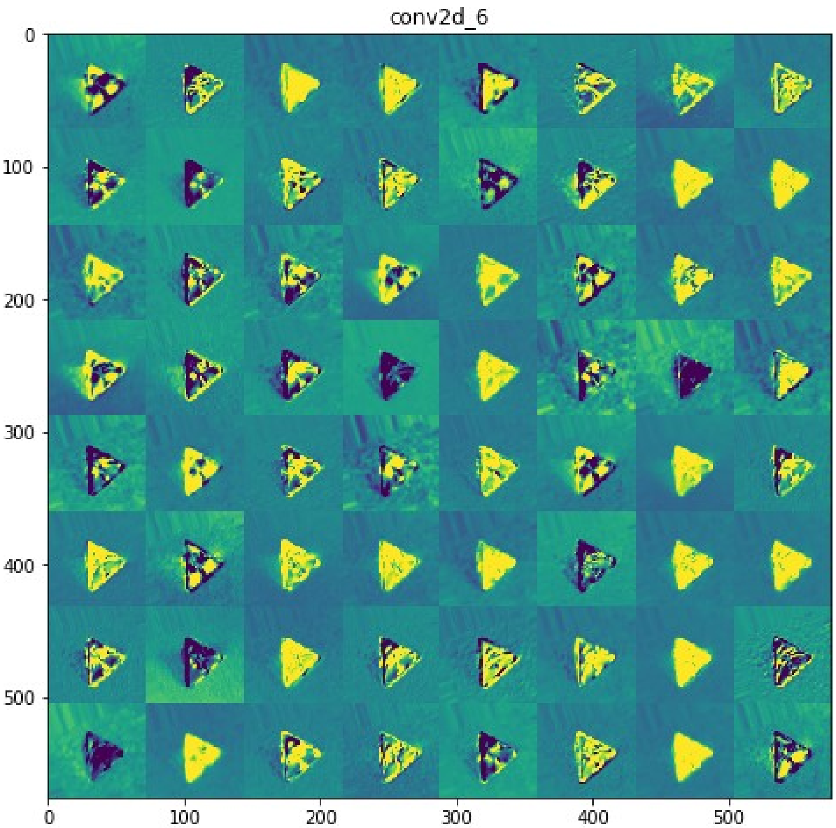
```
import numpy as np
import time
for img in dice_images:
    plt.figure(facecolor='w')
    plt.title("Original Image")
    plt.imshow(img[0])
    activations = activation_model.predict(img)
    layer_names = [layer.name for layer in model.layers]
    images_per_row = 8

    for layer_name, layer_activation in zip(layer_names, activations):
        # skiip none convolutional layers
        if(layer_name.find('conv') == -1):
            continue
        n_features = layer_activation.shape[-1]

        size = layer_activation.shape[1]

        n_cols = n_features // images_per_row
        display_grid = np.zeros((size * n_cols, images_per_row * size))

        for col in range(n_cols):
            for row in range(images_per_row):
                channel_image = layer_activation[0,:, :,col * images_pe
r_row + row]
                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 : (col + 1) * size, row * size
: (row + 1) * size] = channel_image

        scale = 1. / size
        plt.figure(figsize=(scale * display_grid.shape[1],scale * displ
ay_grid.shape[0]), facecolor='w')
        plt.title(layer_name)
        plt.grid(False)
        plt.imshow(display_grid, aspect='auto', cmap='viridis')
    plt.show()
```

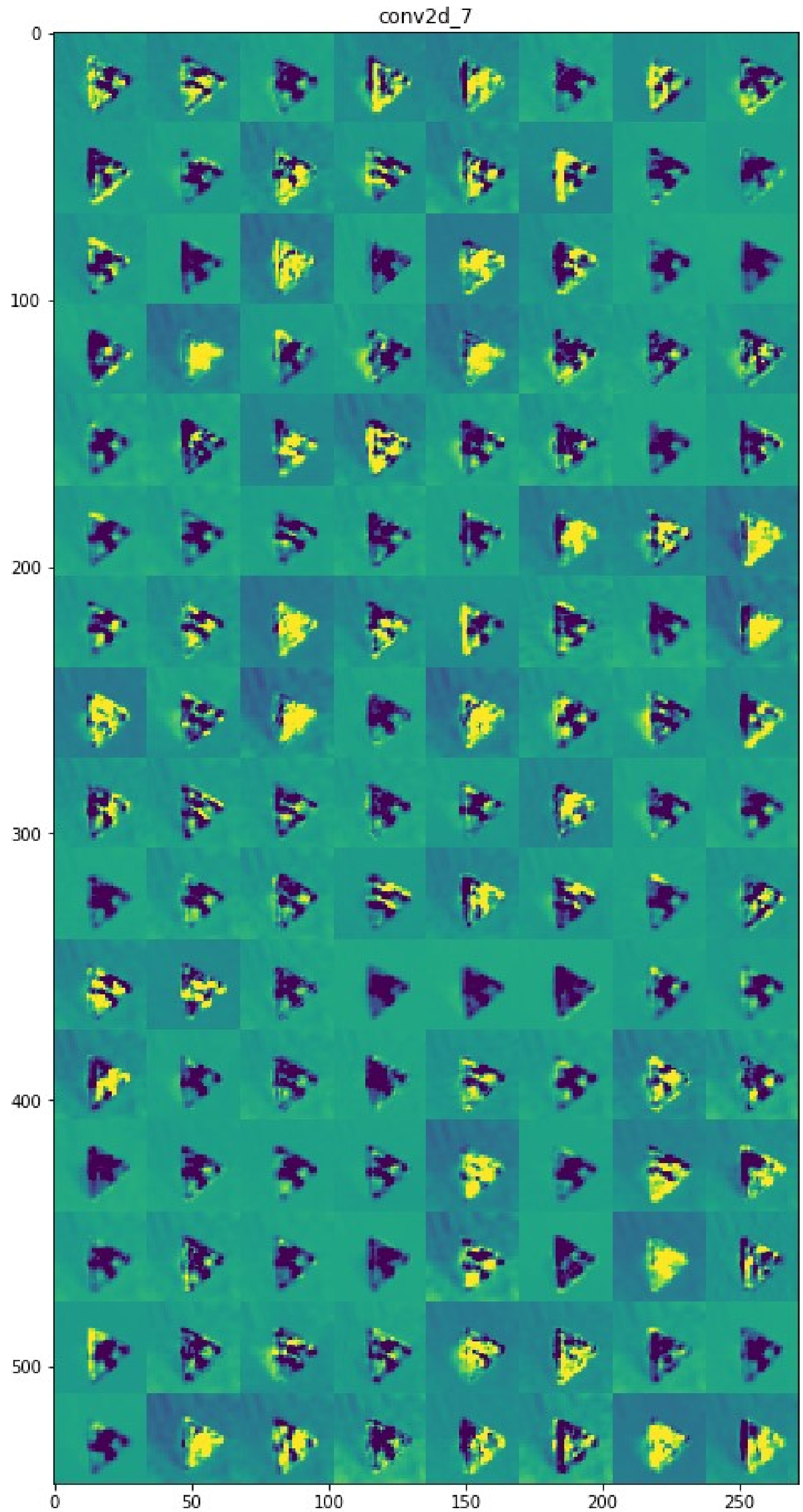


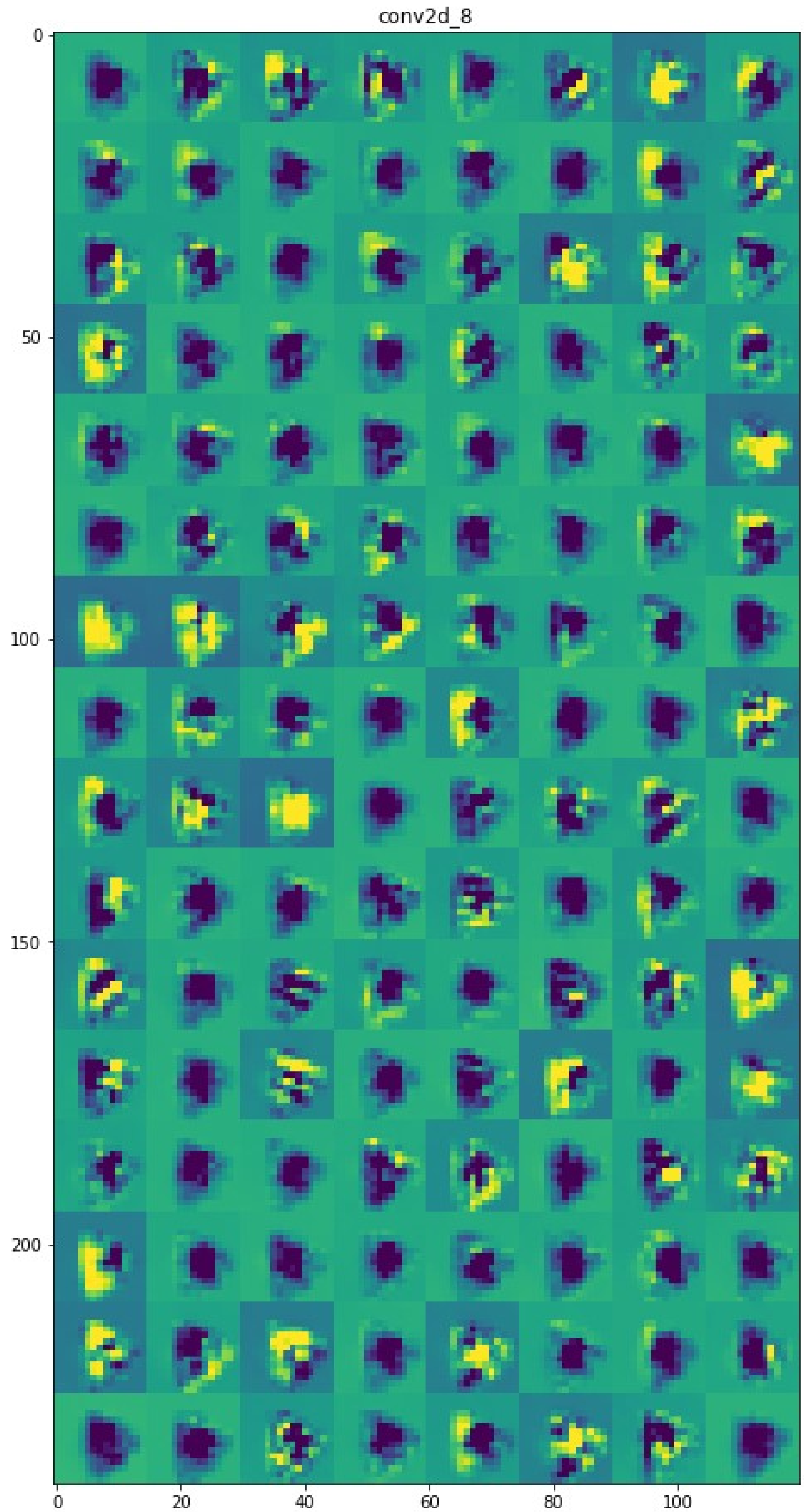
Original Image
0
20
40
60
80
100
120
140
0
25
50
75
100
125

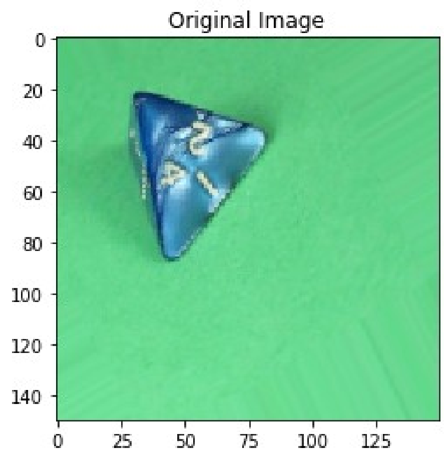


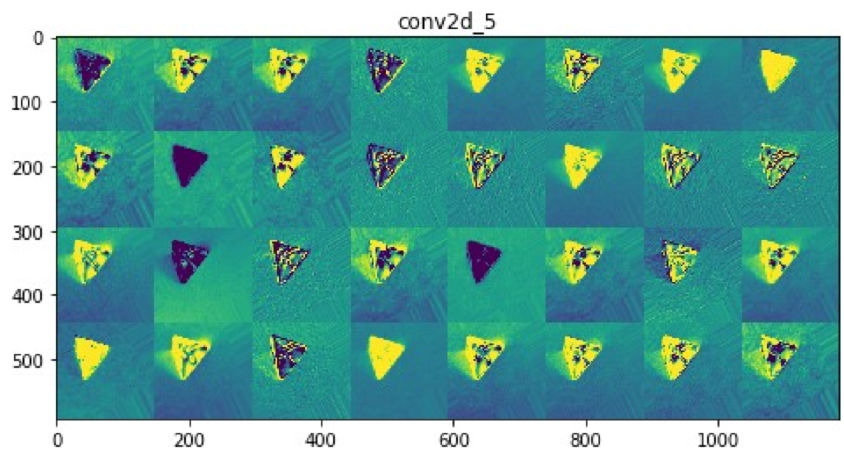
conv2d_5
0
100
200
300
400
500
0
200
400
600
800
1000

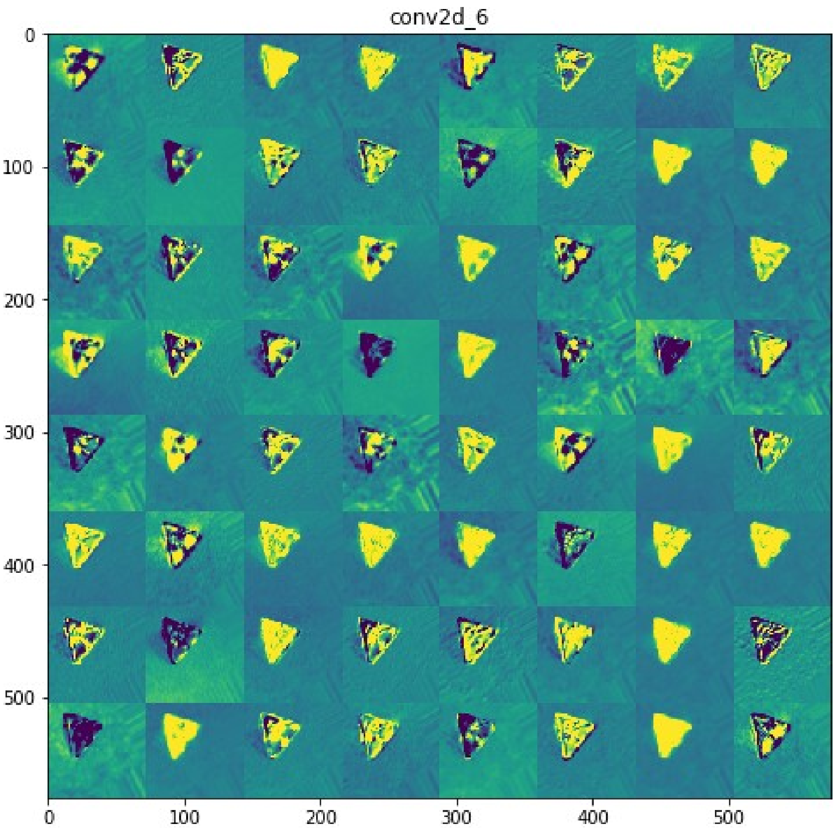

conv2d_6
0
100
200
300
400
500
0
100
200
300
400
500

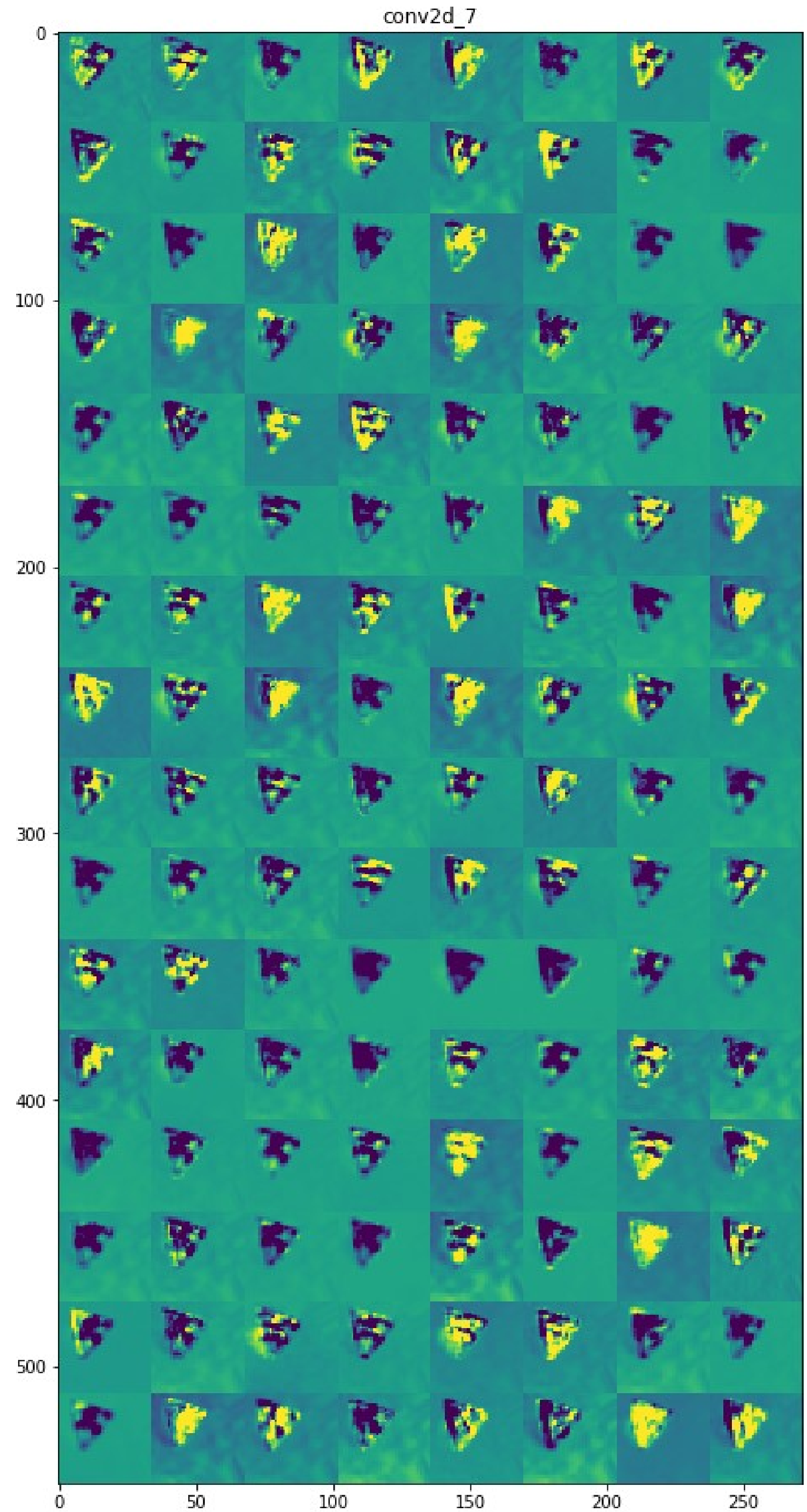

conv2d_7
0
100
200
300
400
500
0
50
100
150
200
250

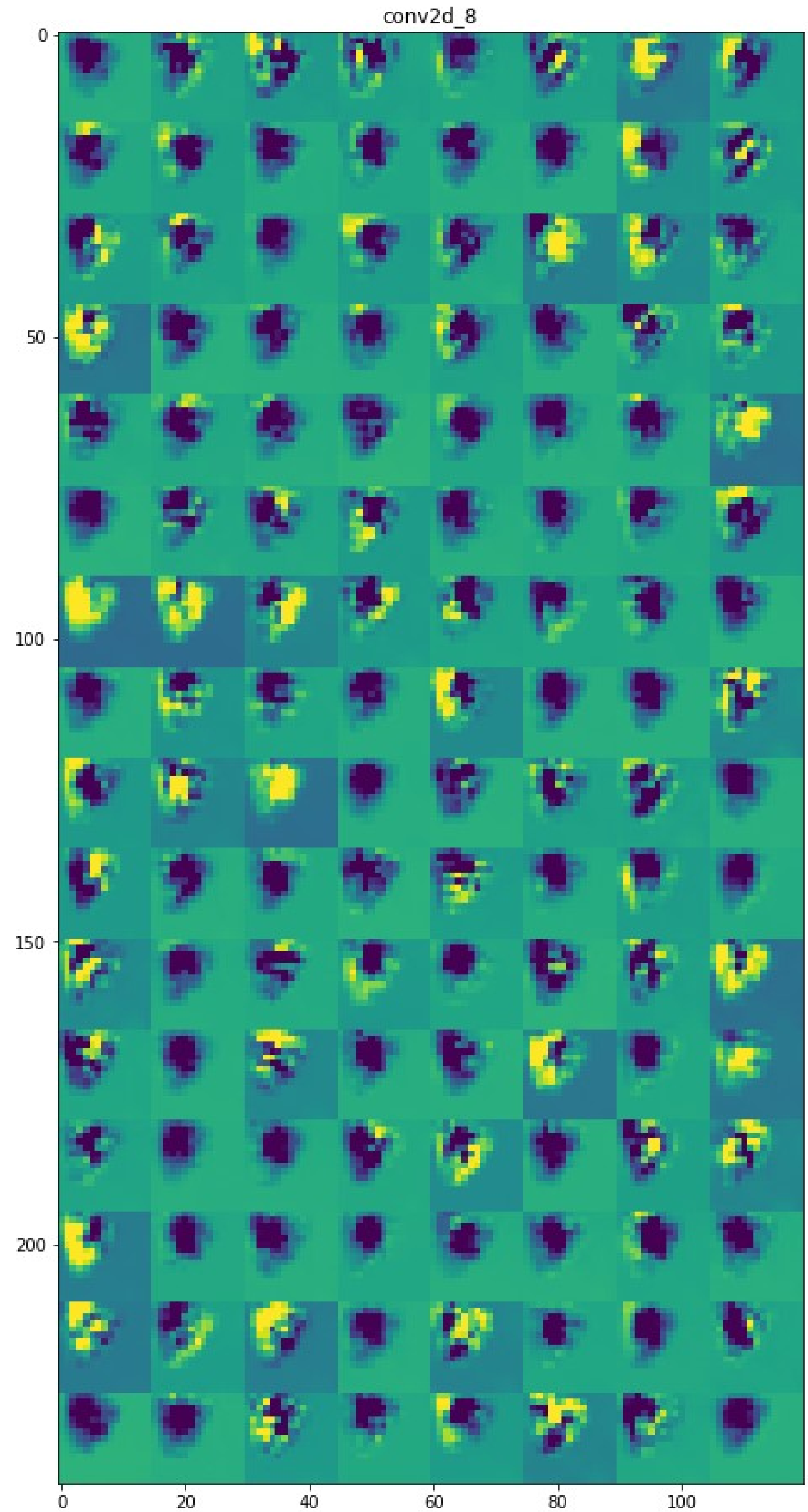

conv2d_8
0
50
100
150
200
0
20
40
60
80
100

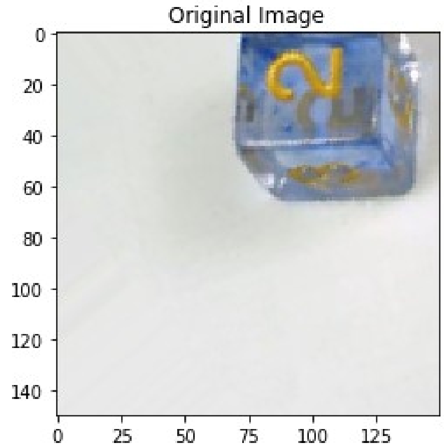

Original Image
0
20
40
60
80
100
120
140
0
25
50
75
100
125

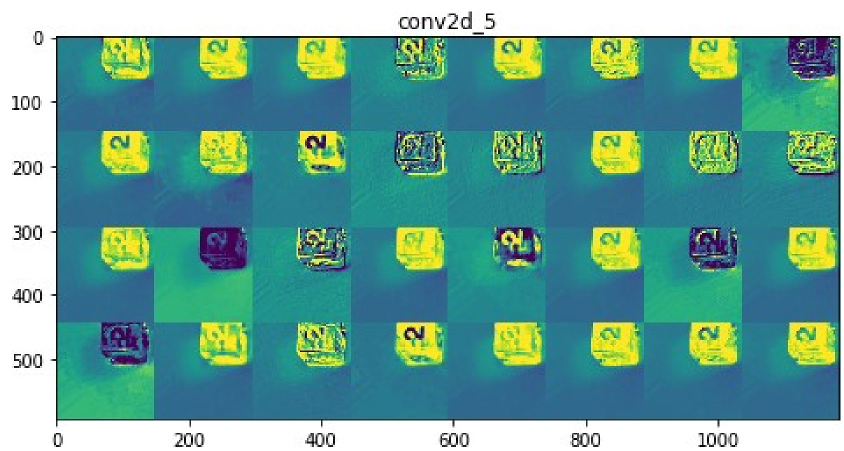

conv2d_5
0
100
200
300
400
500
0
200
400
600
800
1000

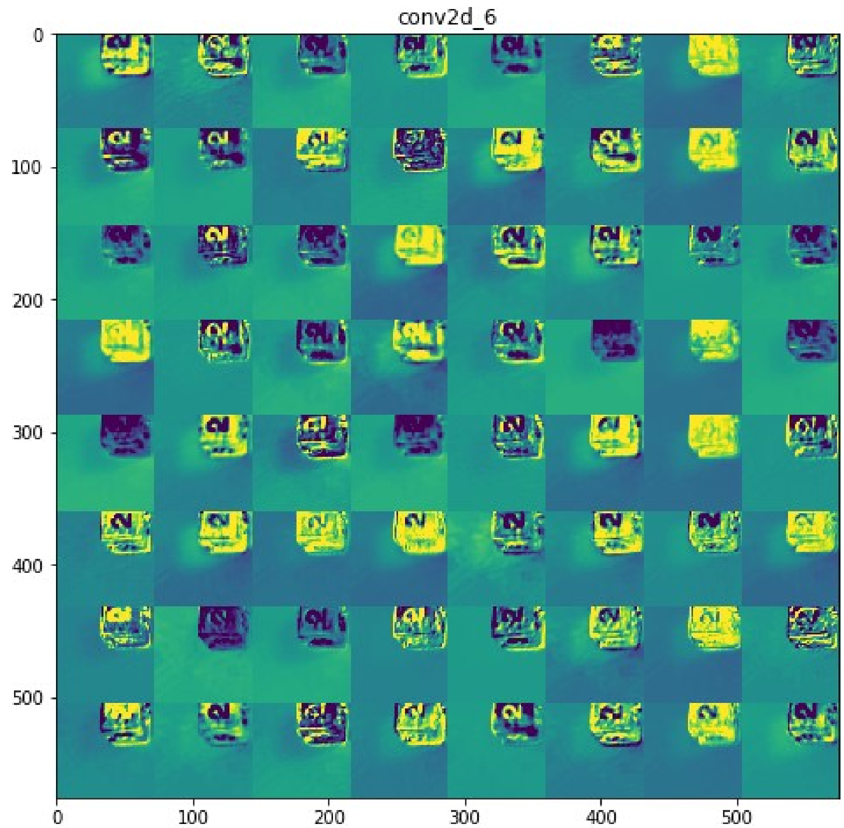

conv2d_6
0
100
200
300
400
500
0
100
200
300
400
500

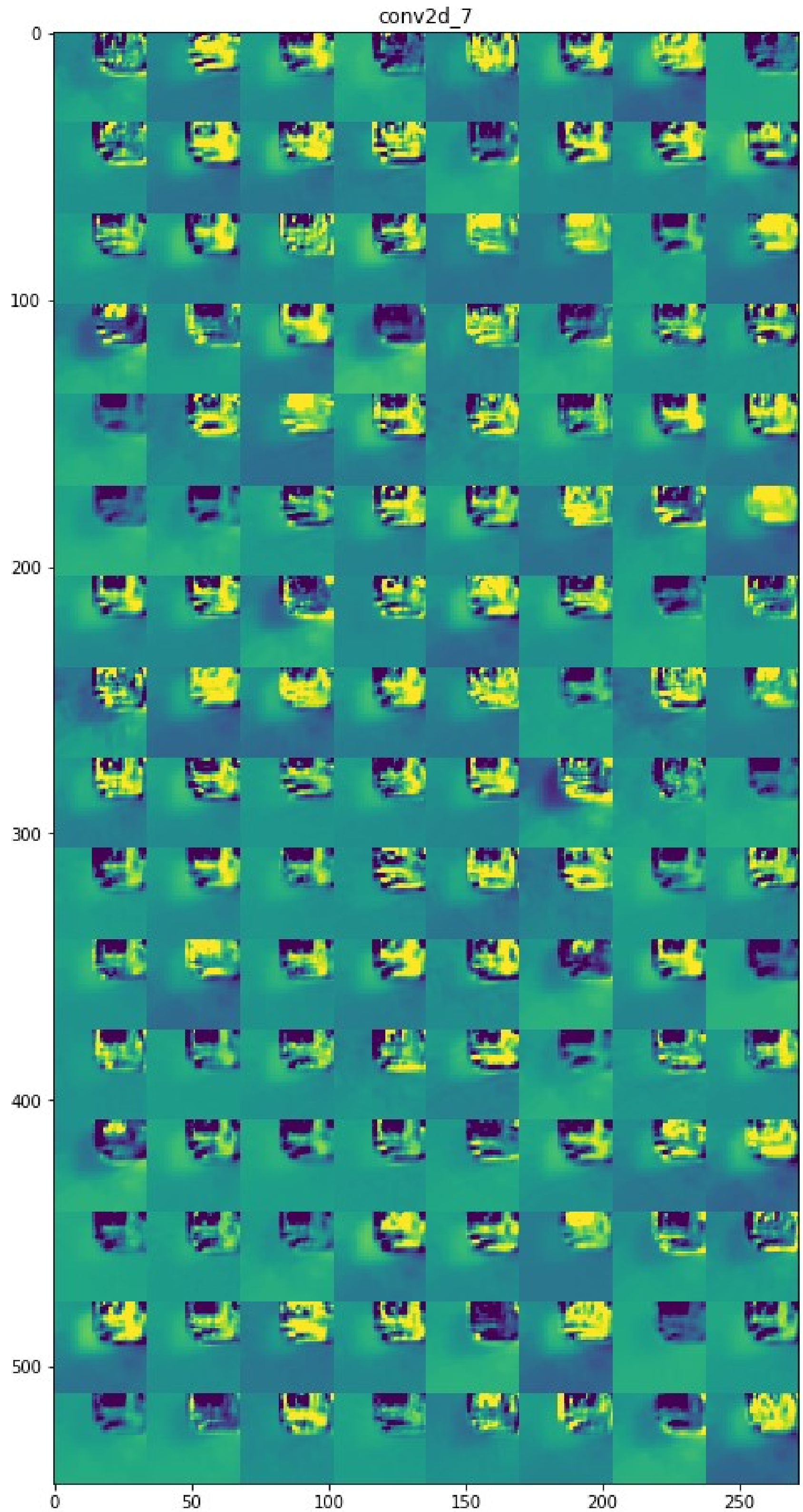

conv2d_7
0
100
200
300
400
500
0
50
100
150
200
250

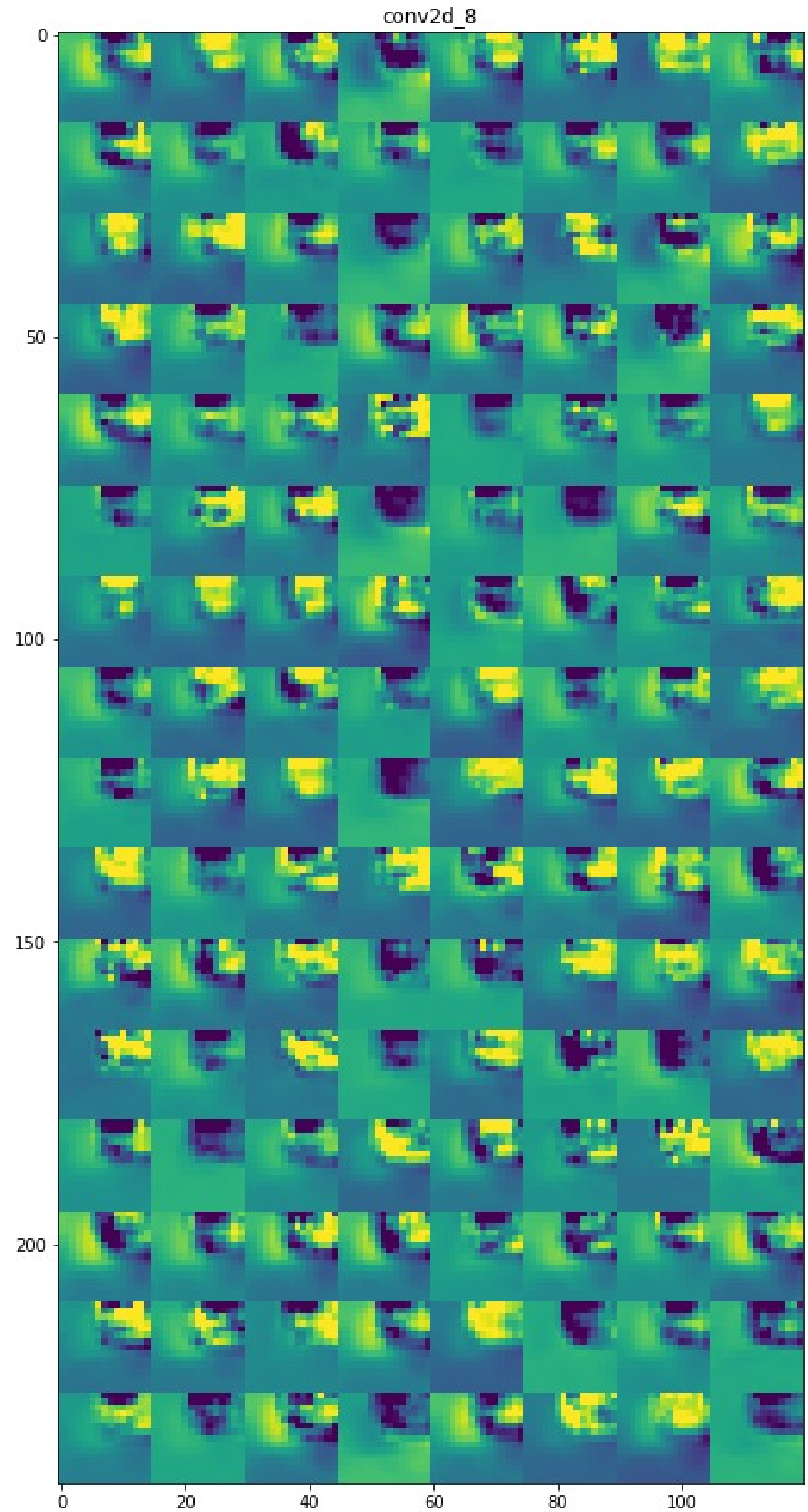

conv2d_8
0
50
100
150
200
0
20
40
60
80
100

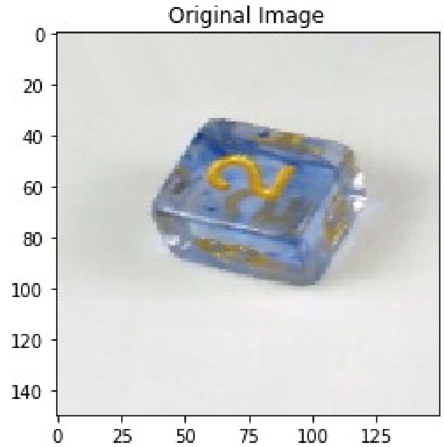

Original Image
0
20
40
60
80
100
120
140
0
25
50
75
100
125

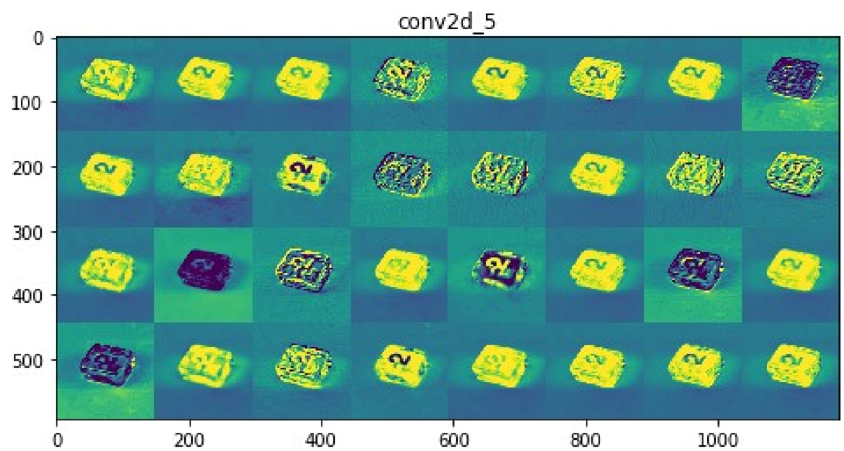

conv2d_5
0
100
200
300
400
500
0
200
400
600
800
1000

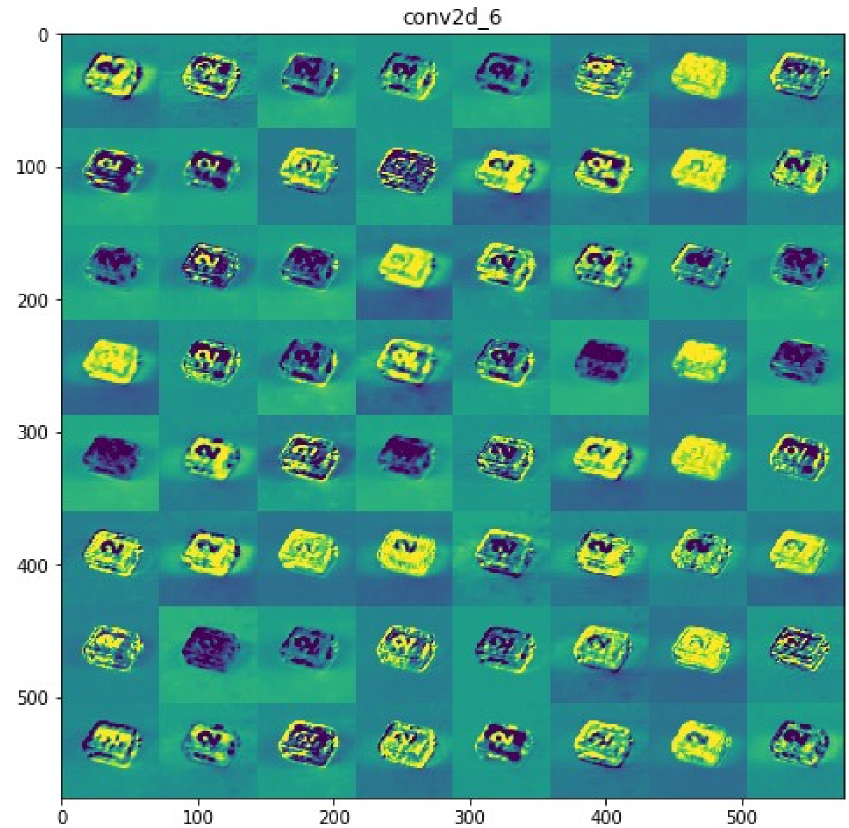

conv2d_6
0
100
200
300
400
500
0
100
200
300
400
500

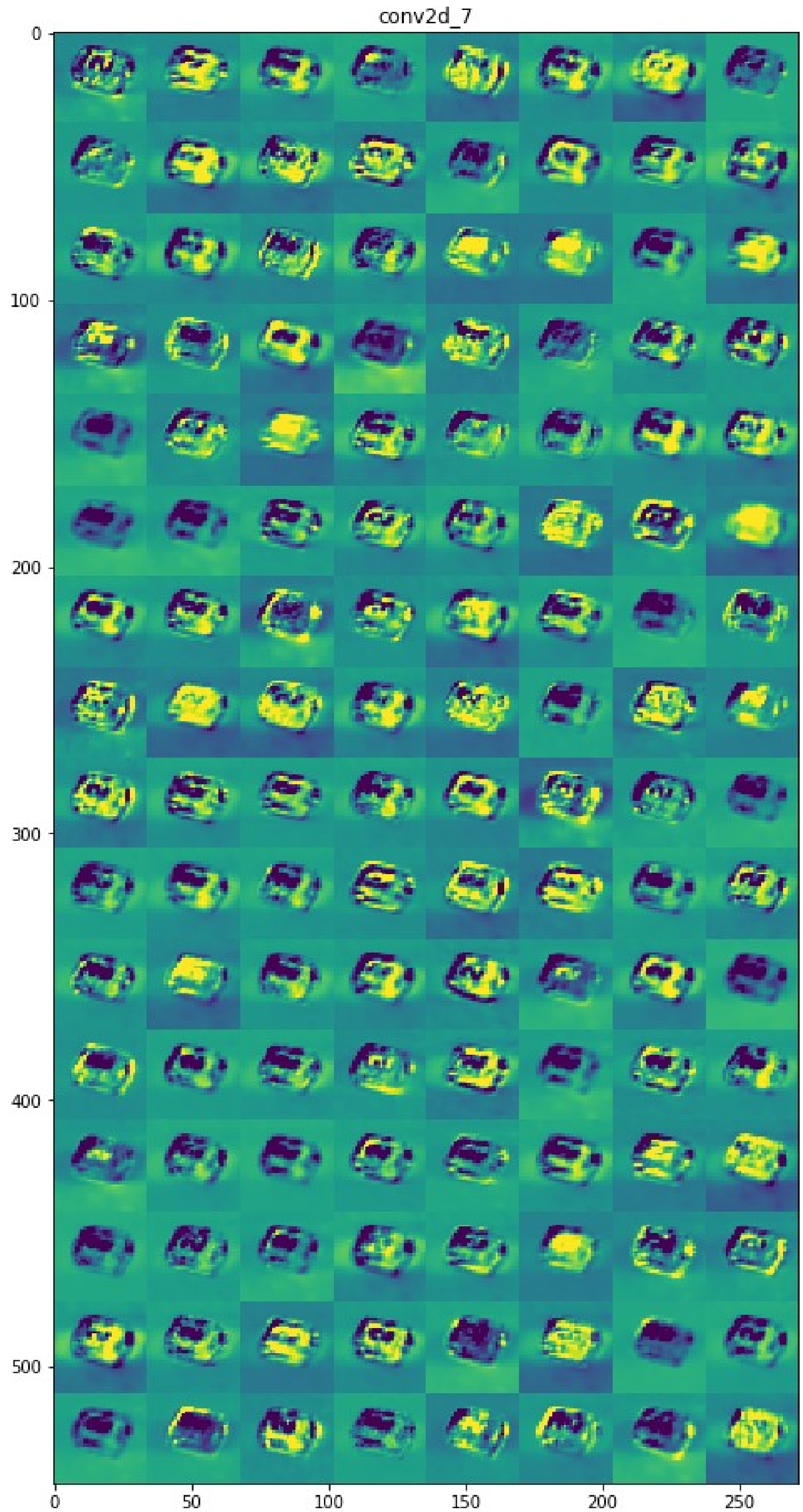

conv2d_7
0
100
200
300
400
500
0
50
100
150
200
250

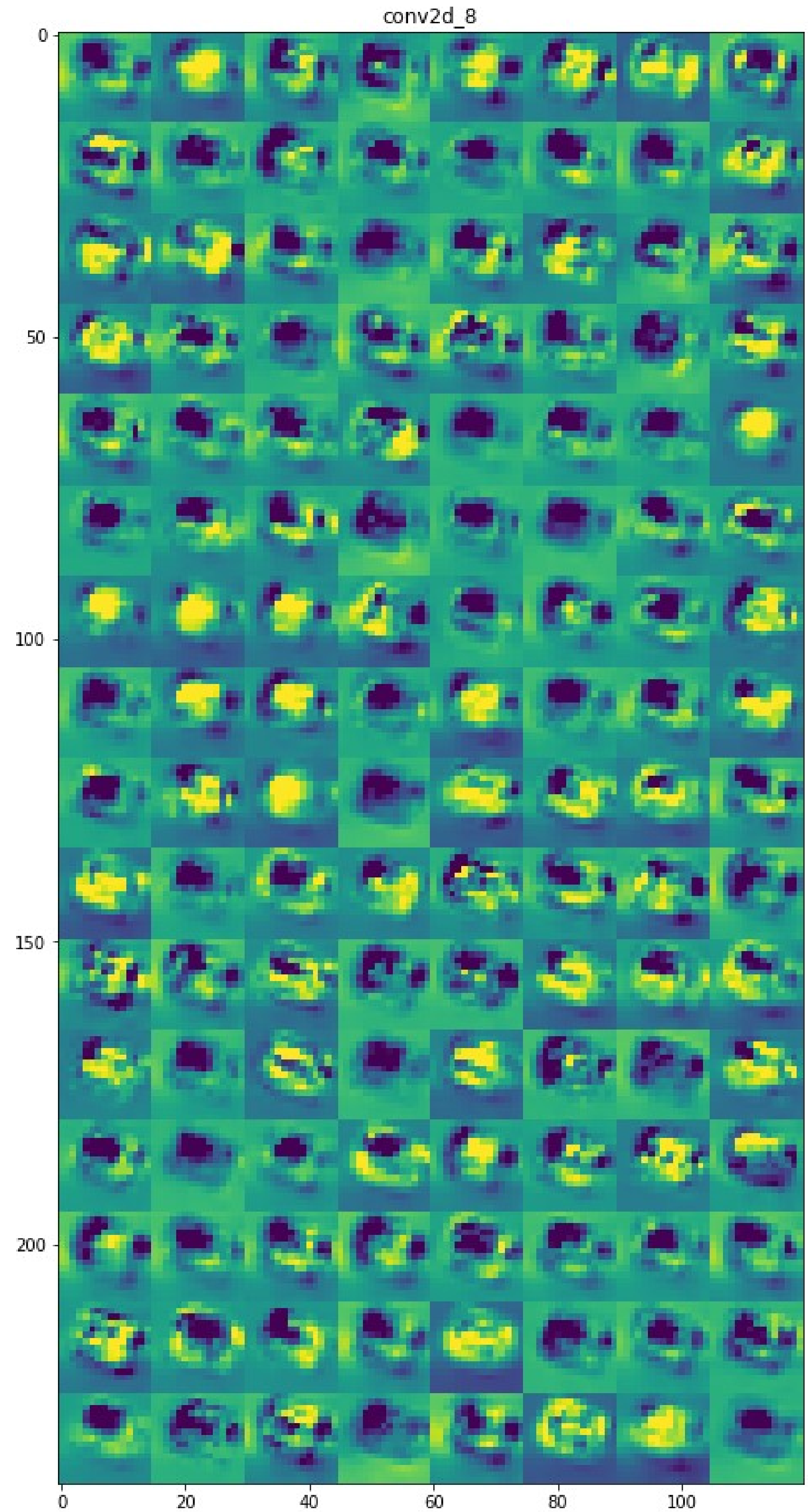

conv2d_8
0
50
100
150
200
0
20
40
60
80
100

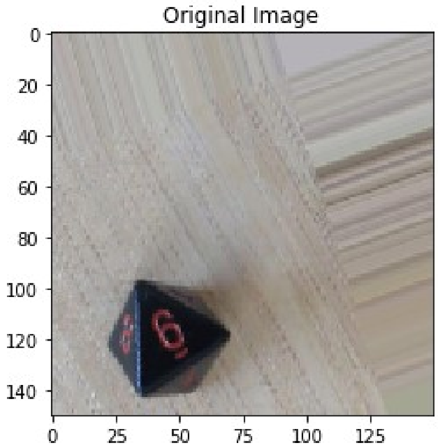

Original Image
0
20
40
60
80
100
120
140
0
25
50
75
100
125

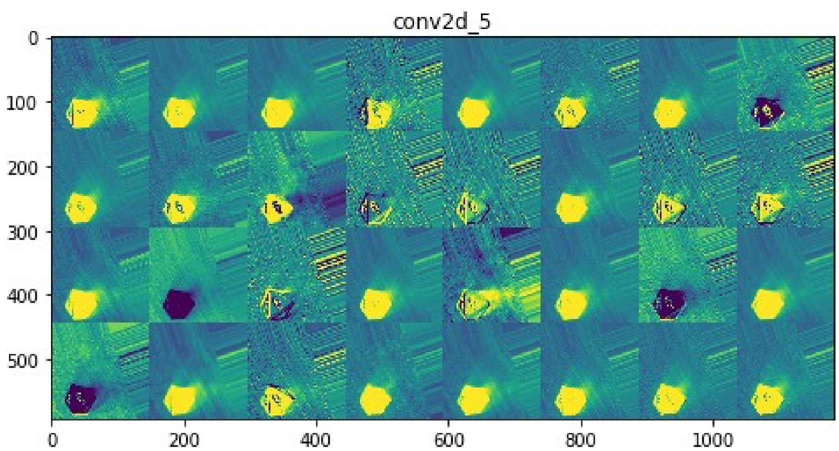

conv2d_5
0
100
200
300
400
500
0
200
400
600
800
1000

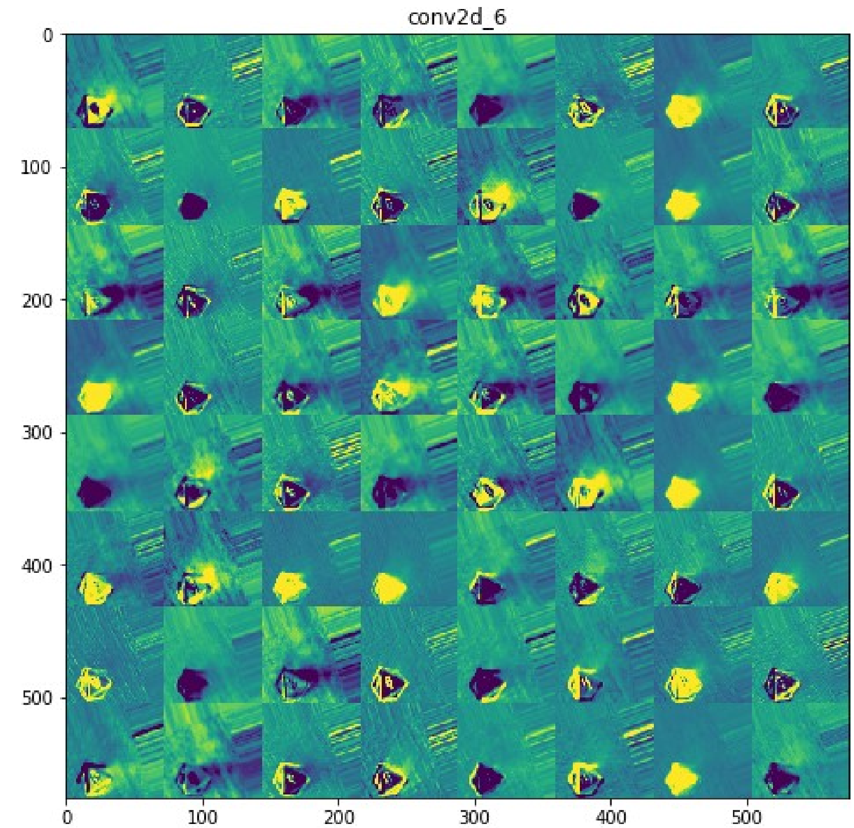

conv2d_6
0
100
200
300
400
500
0
100
200
300
400
500

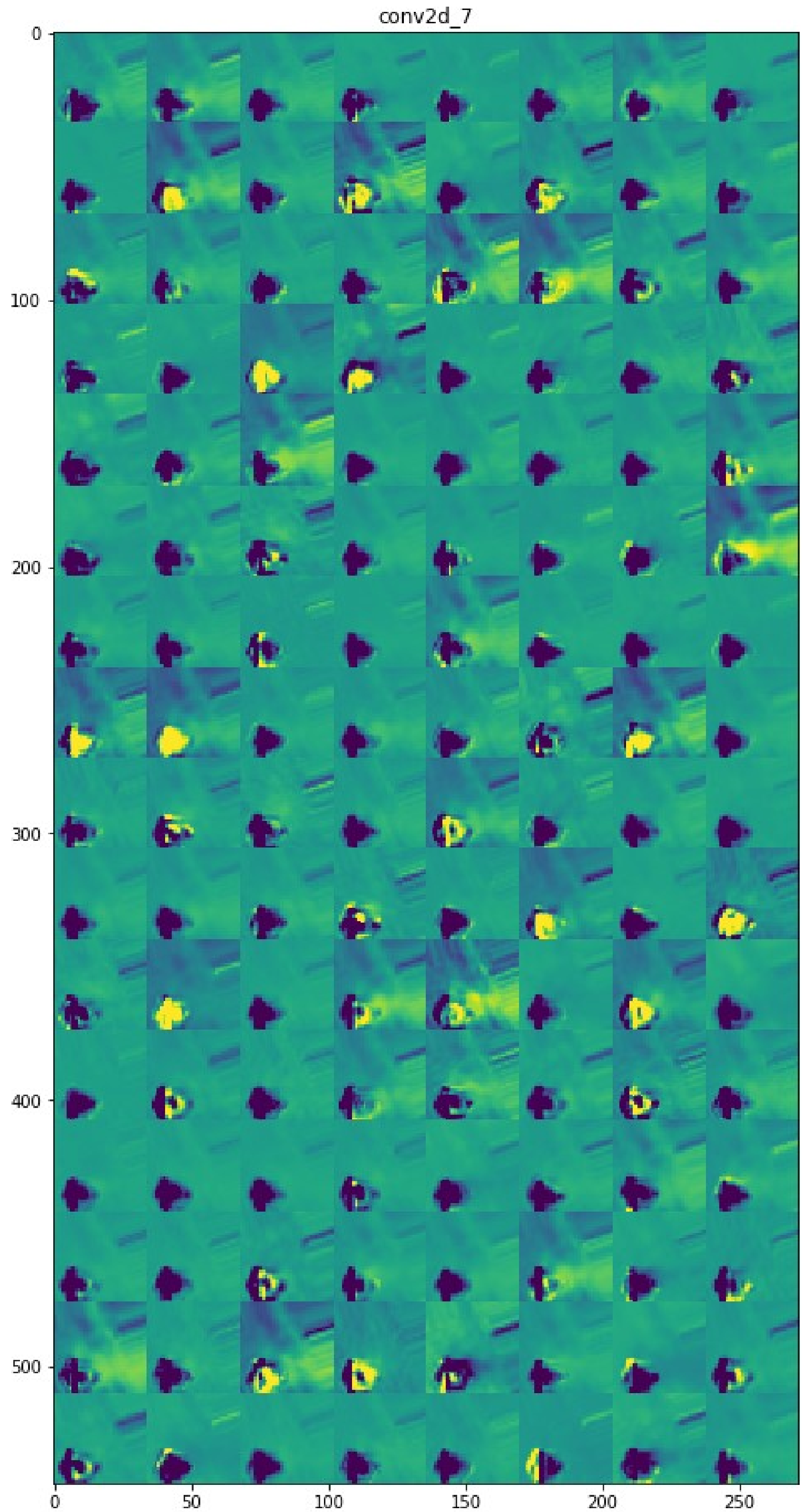

conv2d_7
0
100
200
300
400
500
0
50
100
150
200
250


conv2d_8
0
50
100
150
200
0
20
40
60
80
100

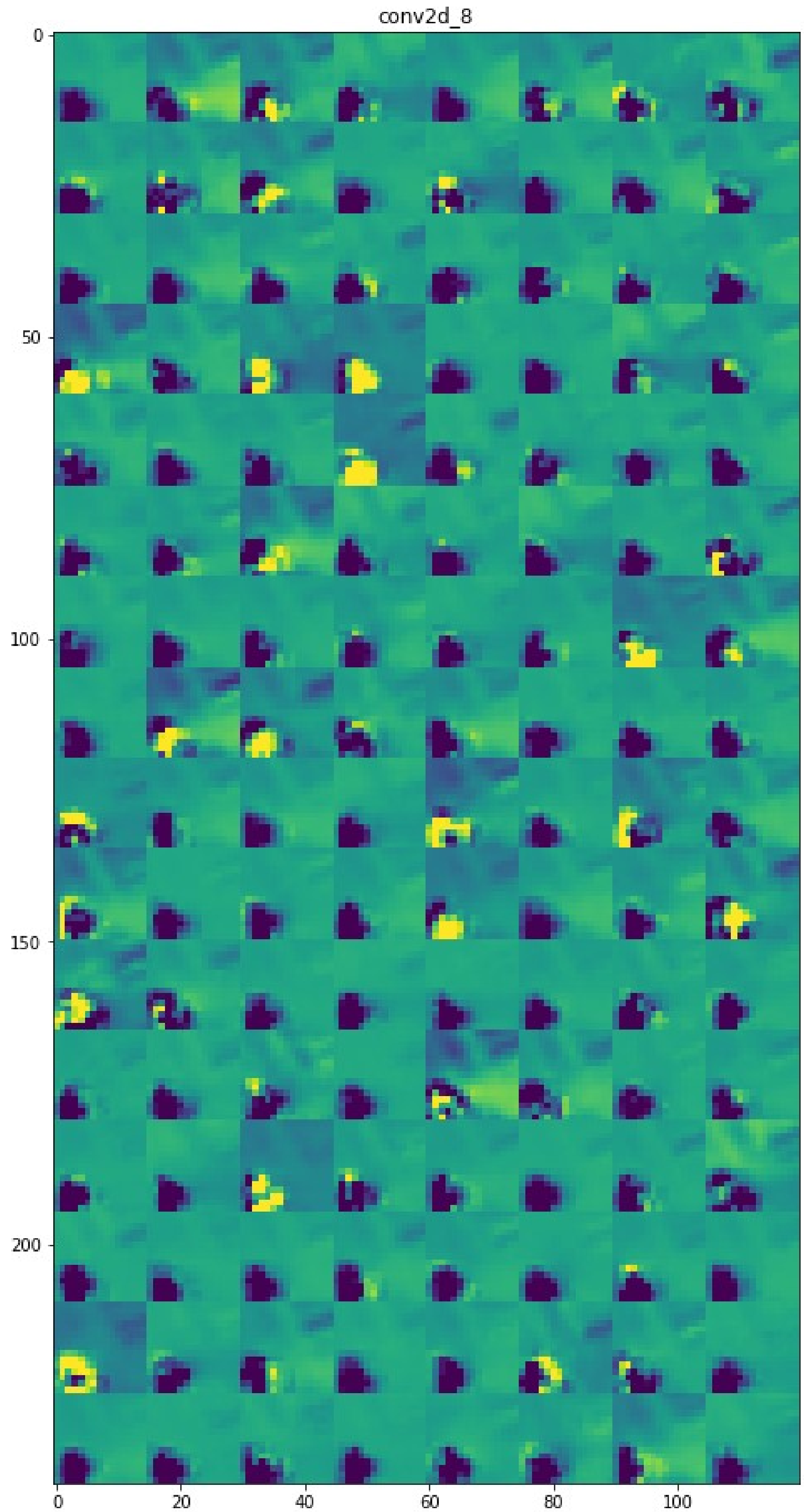

Original Image
0
20
40
60
80
100
120
140
0 25 50 75 100 125

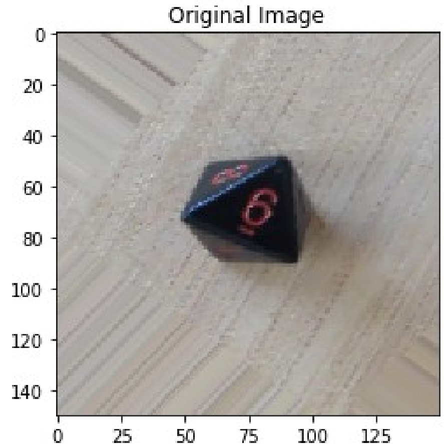

conv2d_5
0
100
200
300
400
500
0 200 400 600 800 1000

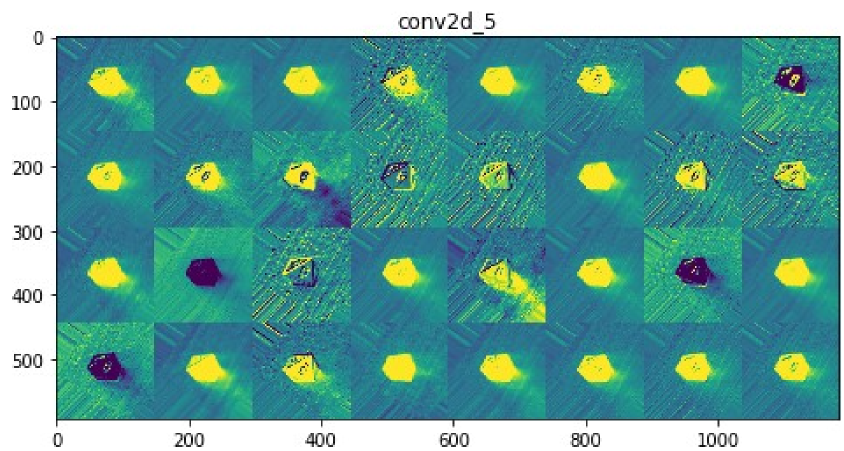

conv2d_6
0
100
200
300
400
500
0
100
200
300
400
500

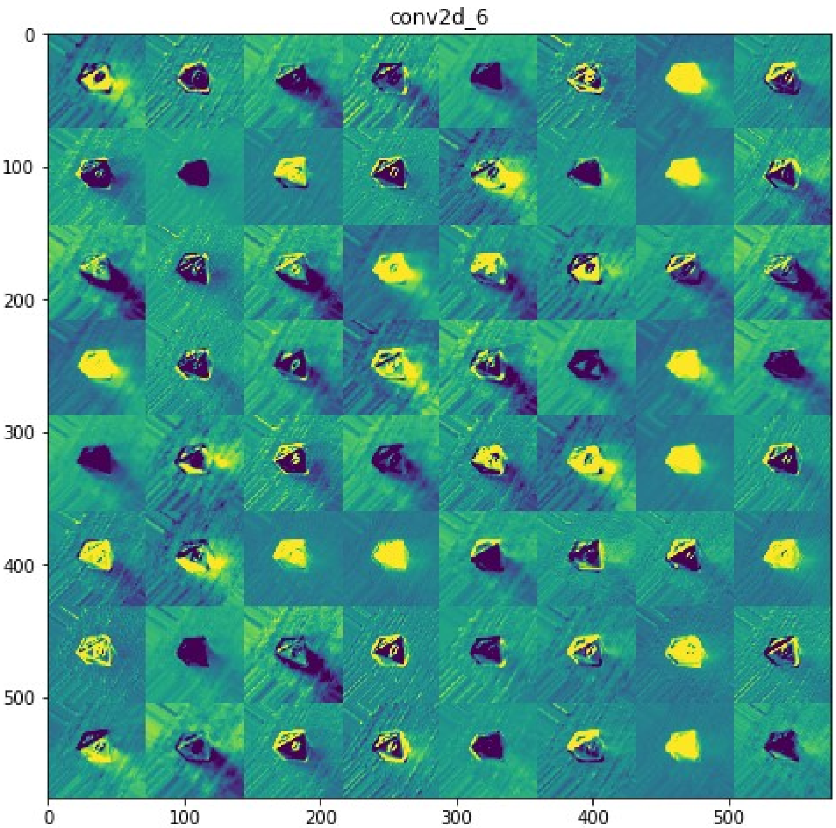


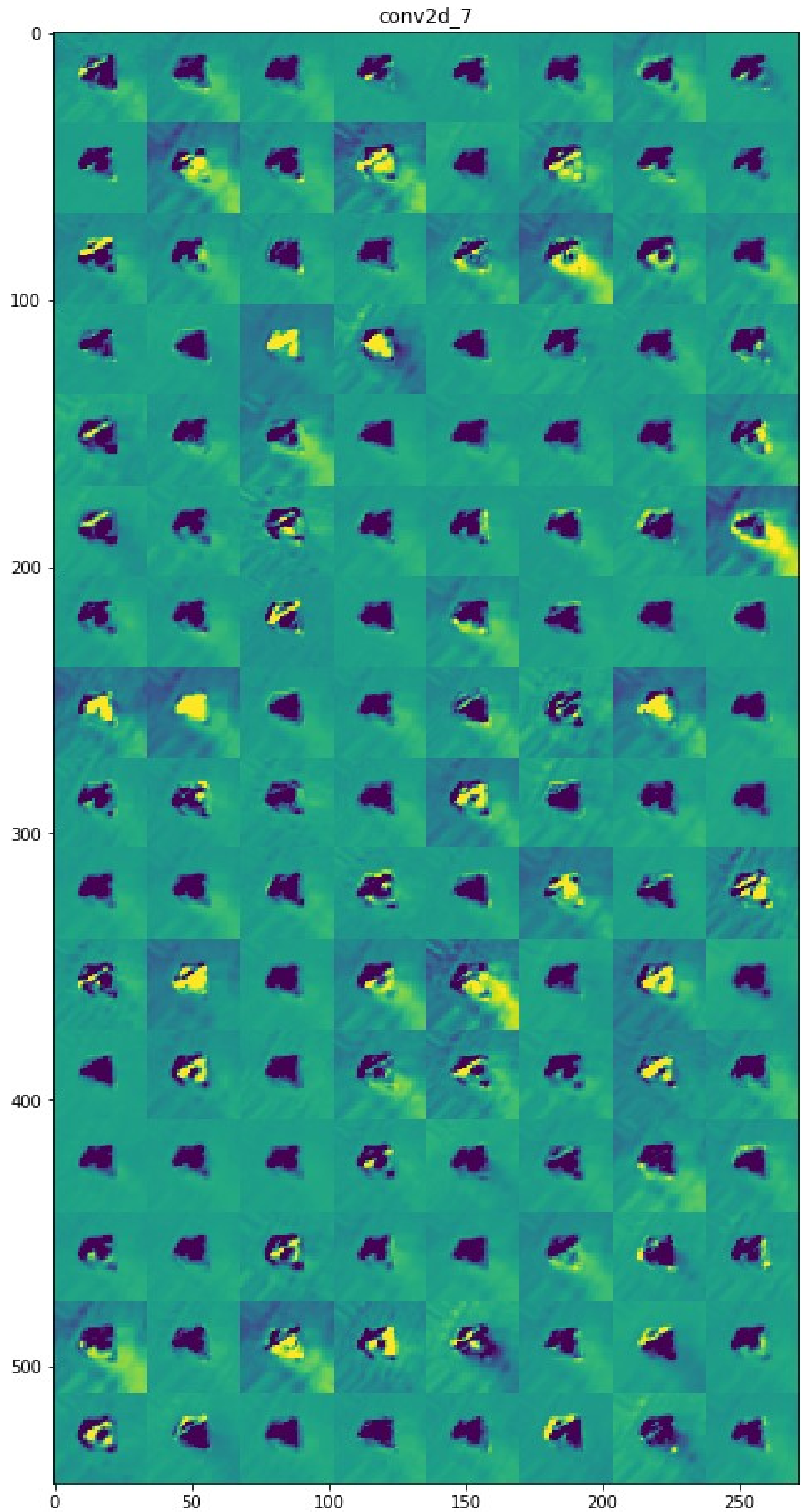
conv2d_7
0
100
200
300
400
500
0
50
100
150
200
250

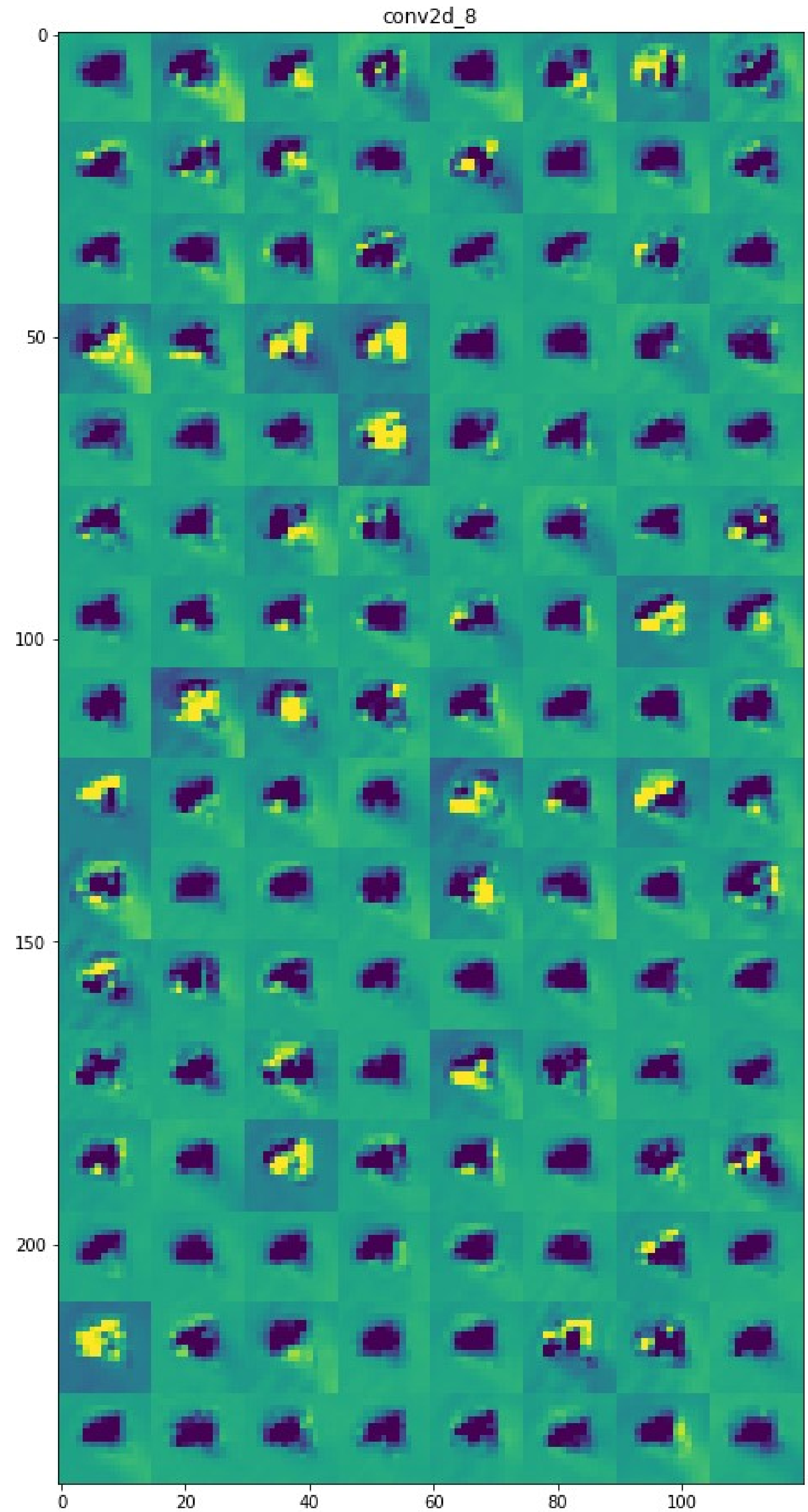

conv2d_8
0
50
100
150
200
0
20
40
60
80
100

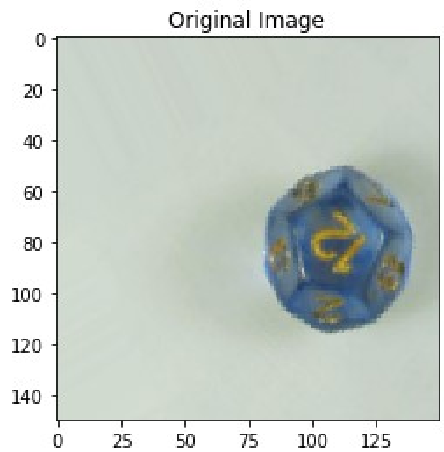

Original Image
0
20
40
60
80
100
120
140
0
25
50
75
100
125

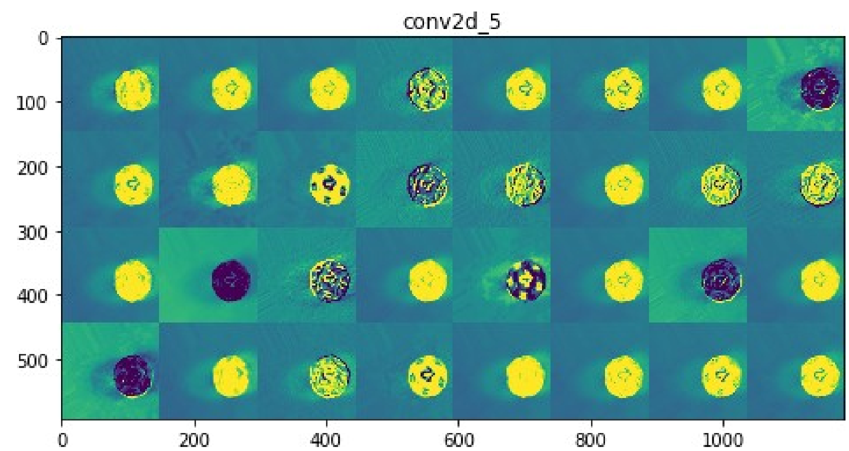

conv2d_5
0
100
200
300
400
500
0
200
400
600
800
1000

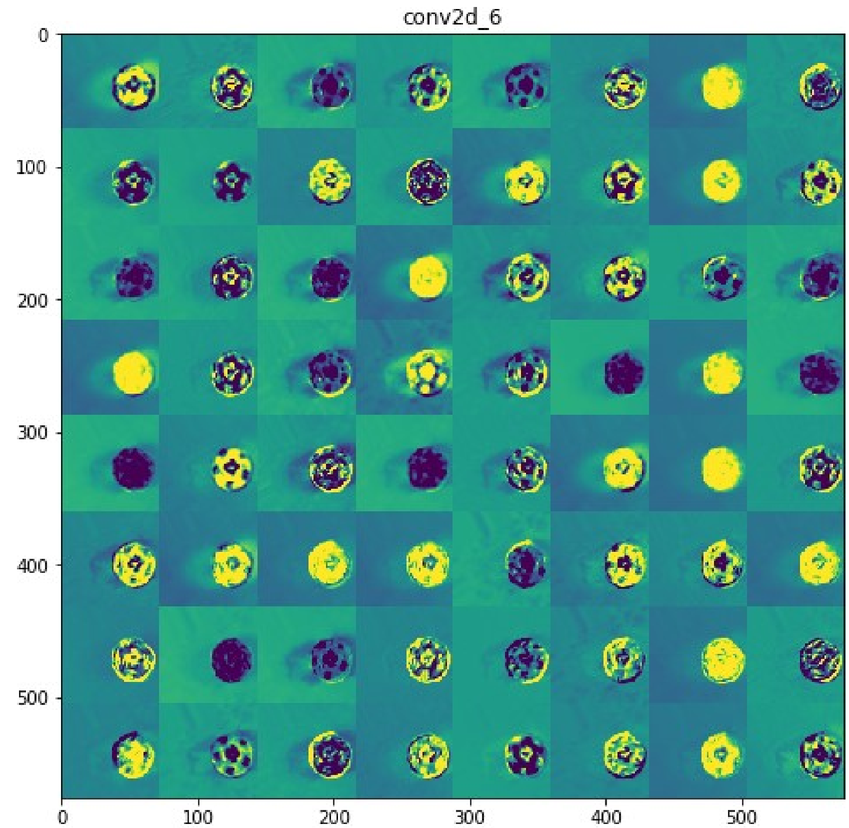

conv2d_6
0
100
200
300
400
500
0
100
200
300
400
500

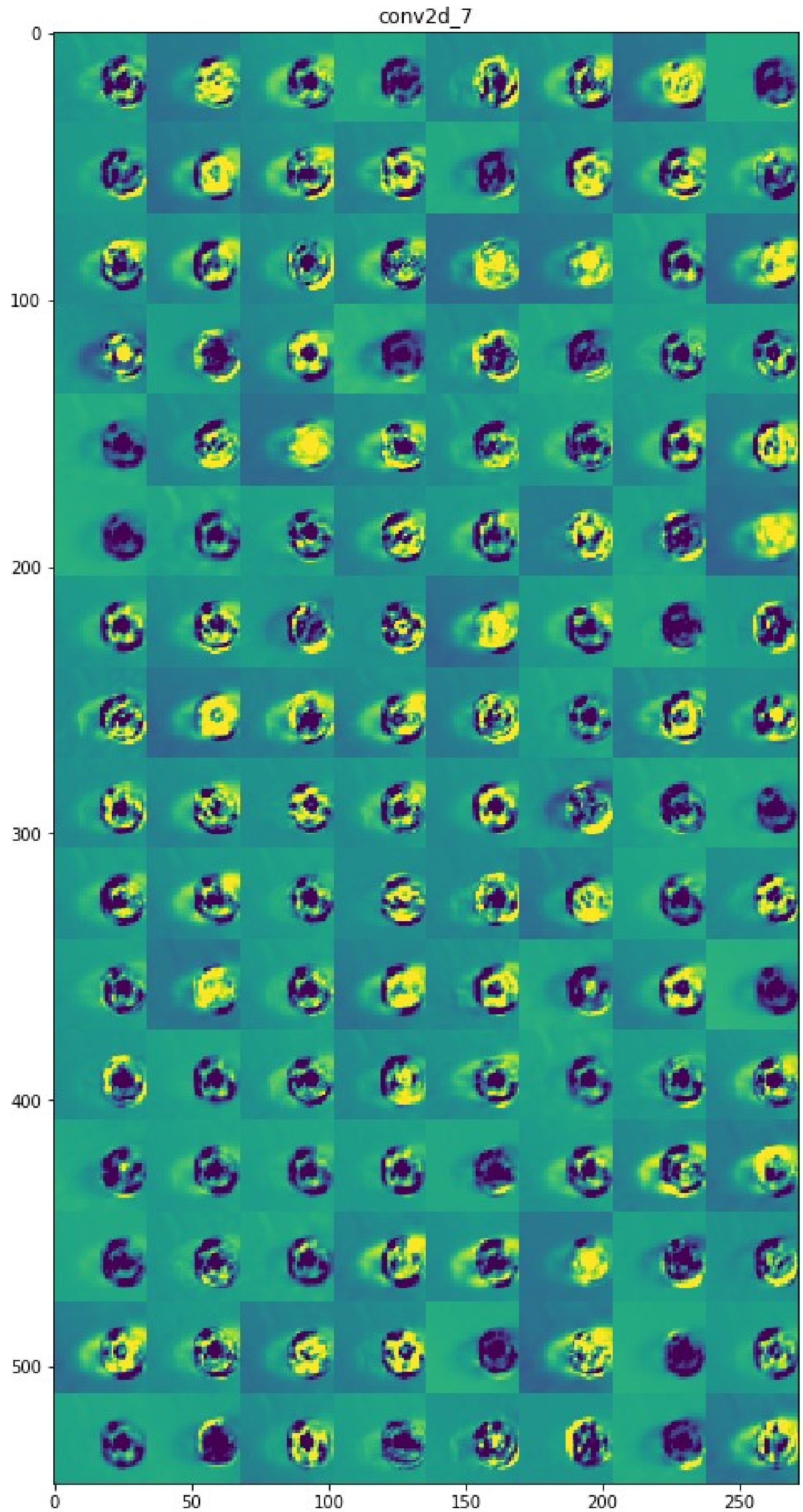

conv2d_7
0
100
200
300
400
500
0
50
100
150
200
250

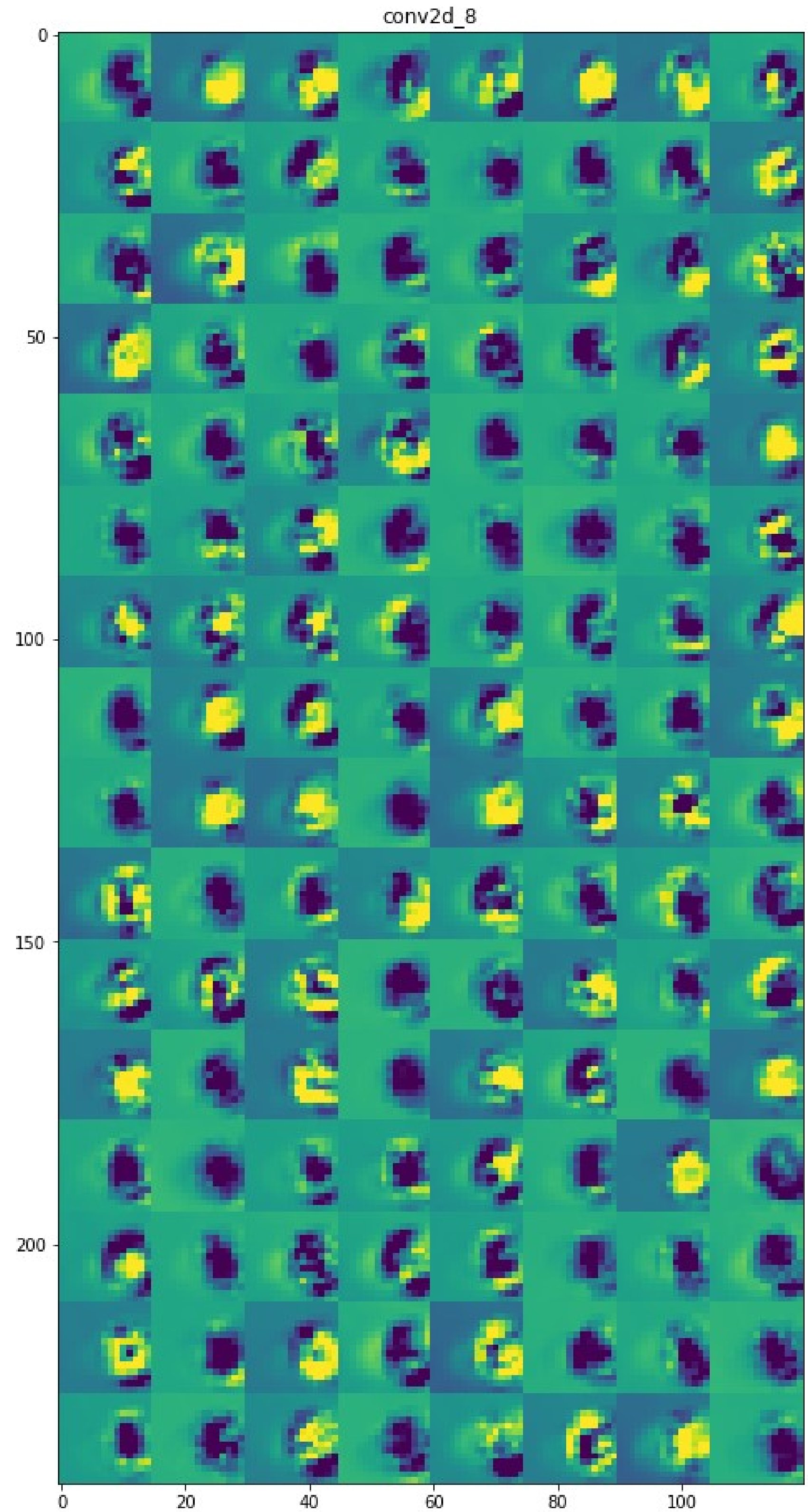

conv2d_8
0
50
100
150
200
0
20
40
60
80
100

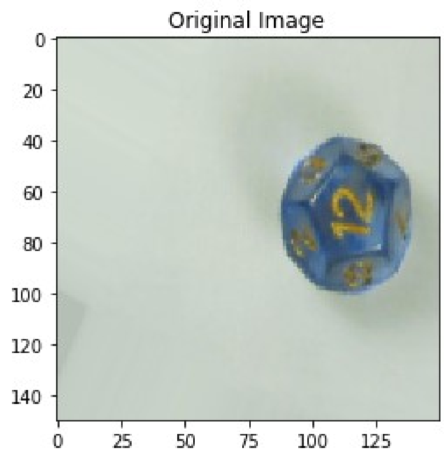

Original Image
0
20
40
60
80
100
120
140
0
25
50
75
100
125

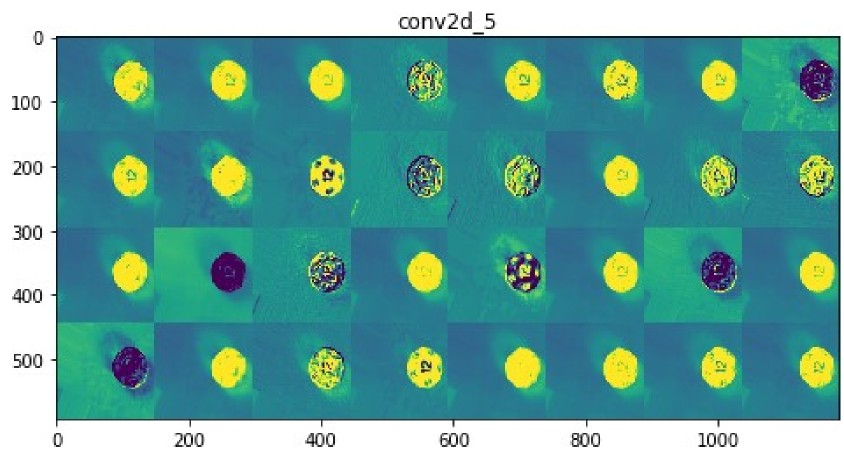

conv2d_5
0
100
200
300
400
500
0
200
400
600
800
1000

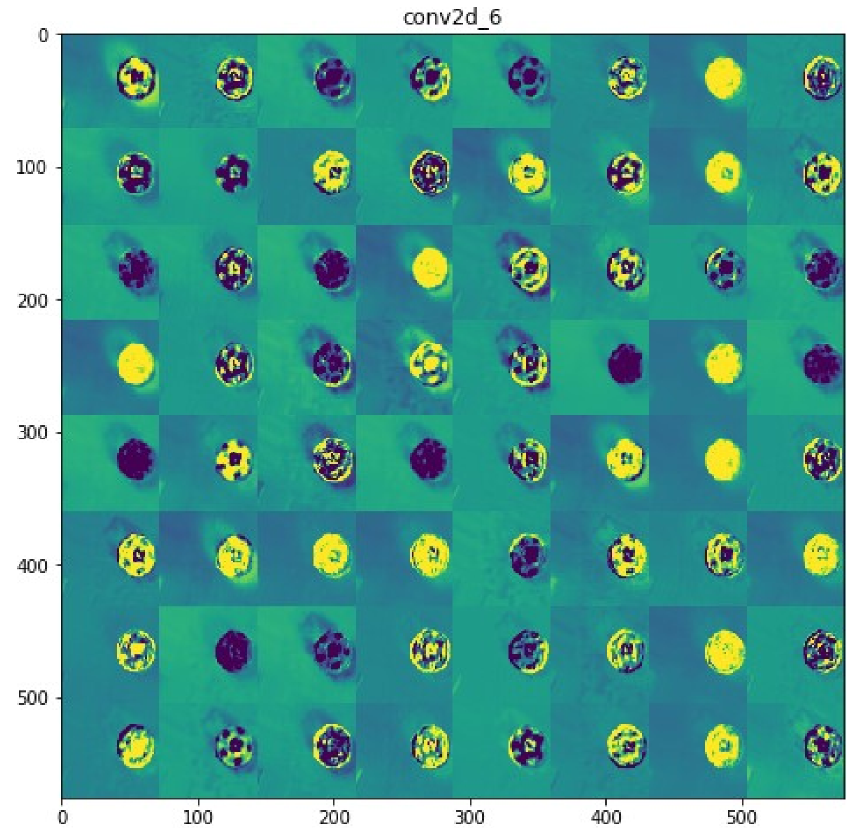

conv2d_6
0
100
200
300
400
500
0
100
200
300
400
500

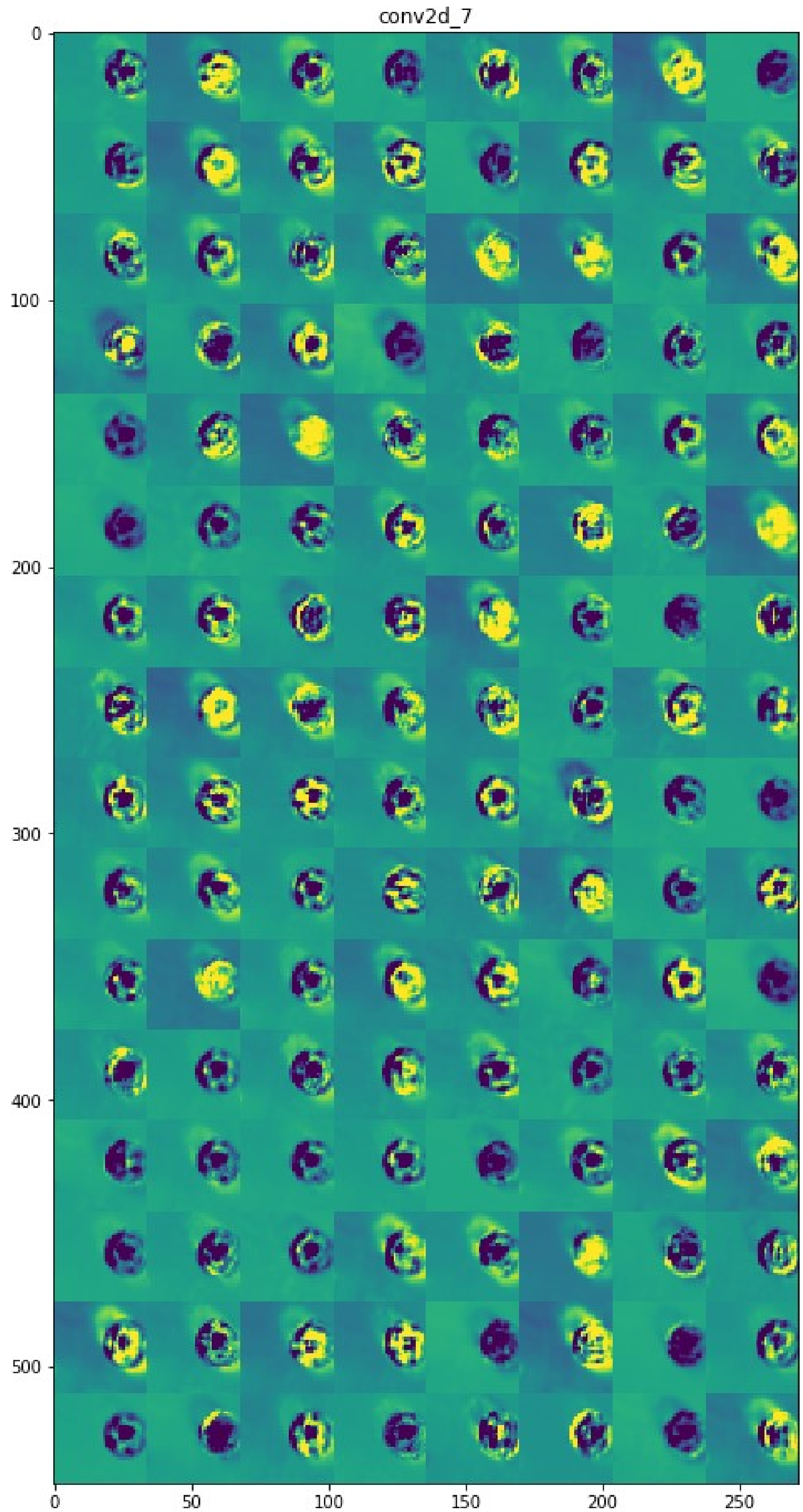

conv2d_7
0
100
200
300
400
500
0
50
100
150
200
250

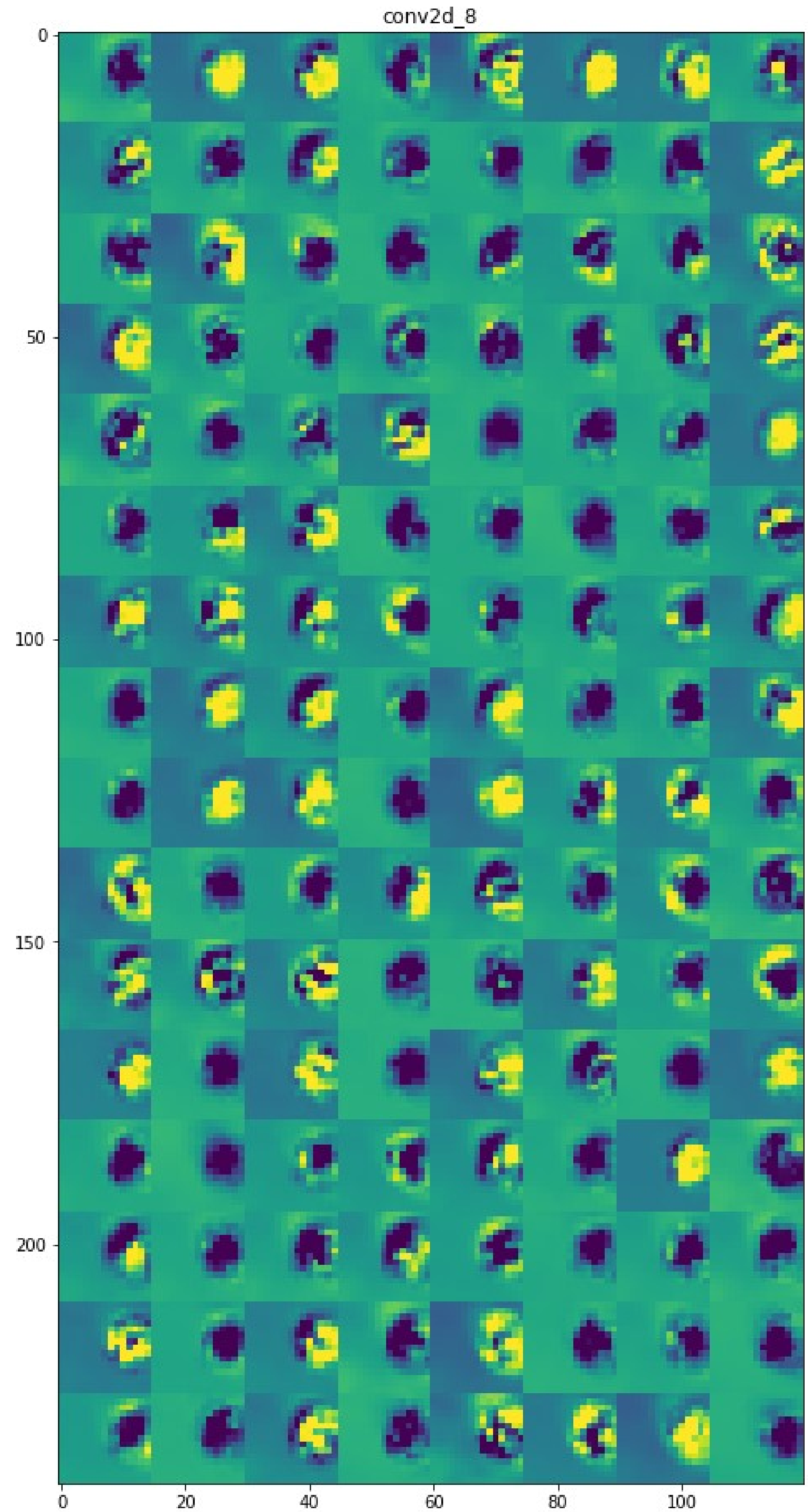

conv2d_8
0
50
100
150
200
0
20
40
60
80
100

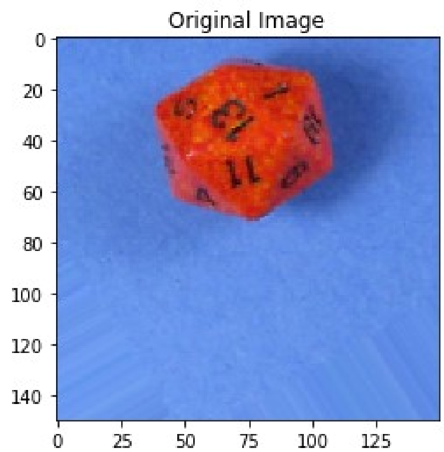

Original Image
0
20
40
60
80
100
120
140
0
25
50
75
100
125

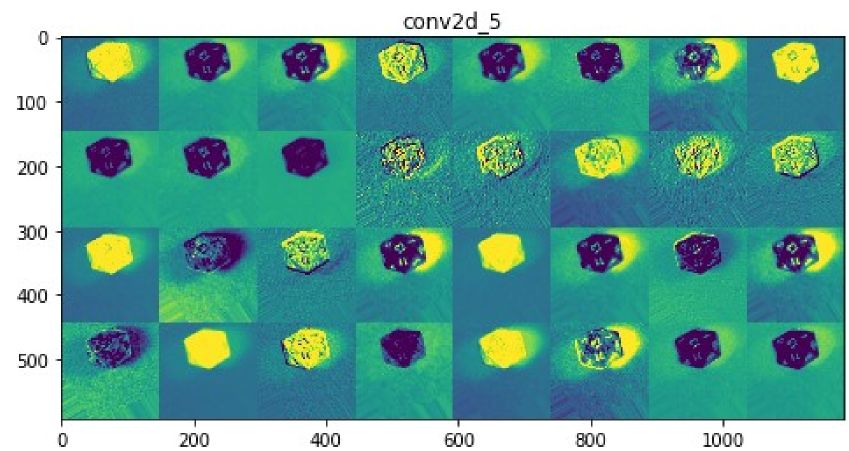

conv2d_5
0
100
200
300
400
500
0
200
400
600
800
1000

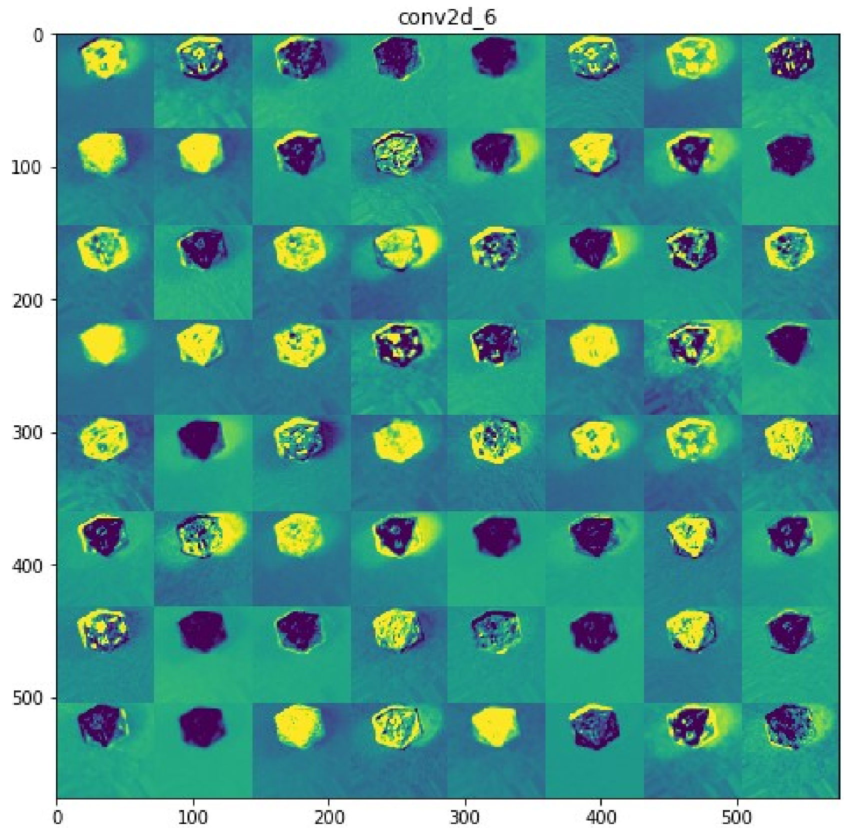

conv2d_6
0
100
200
300
400
500
0
100
200
300
400
500


conv2d_7
0
100
200
300
400
500
0
50
100
150
200
250

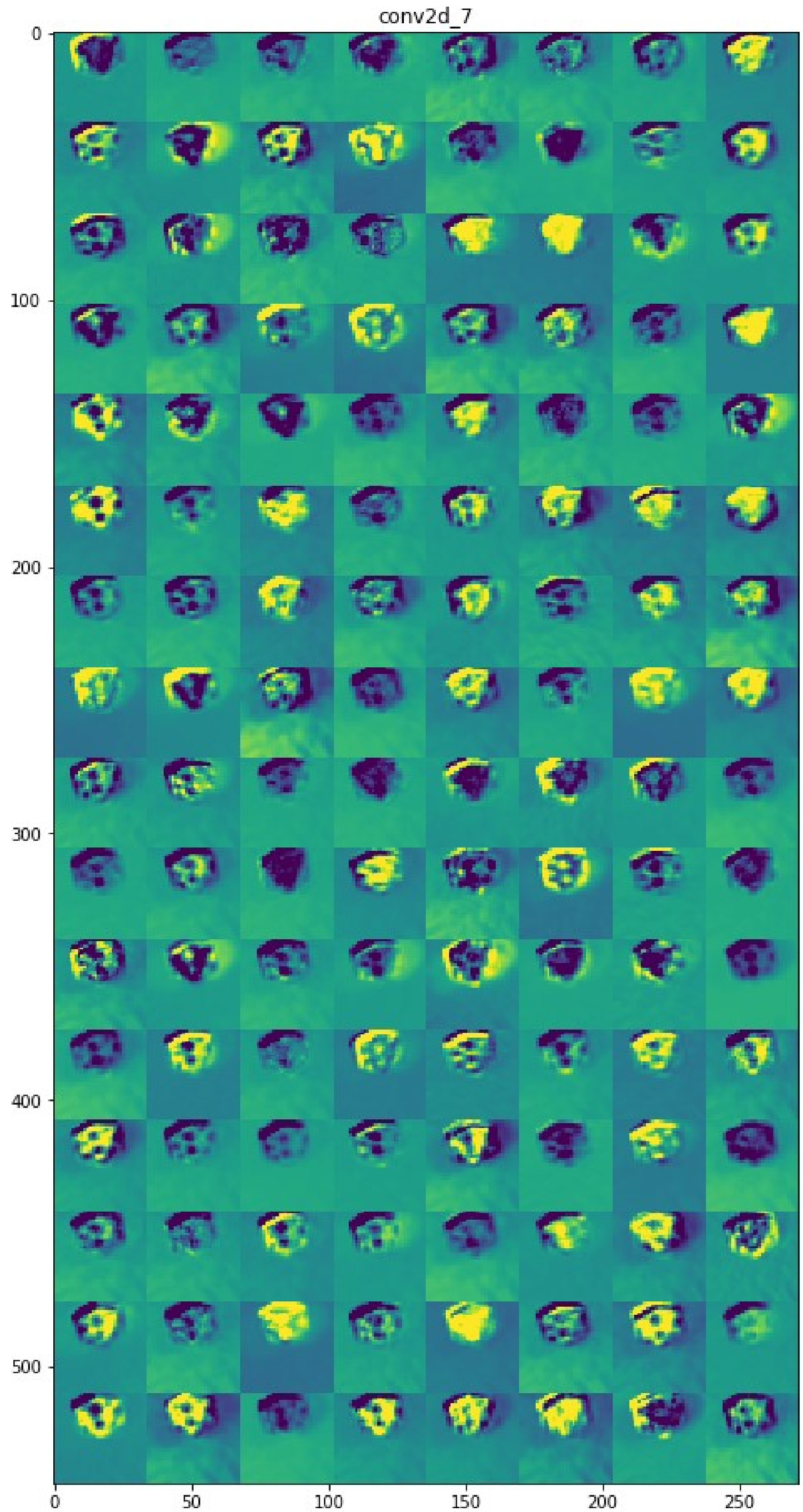

conv2d_8
0
50
100
150
200
0
20
40
60
80
100

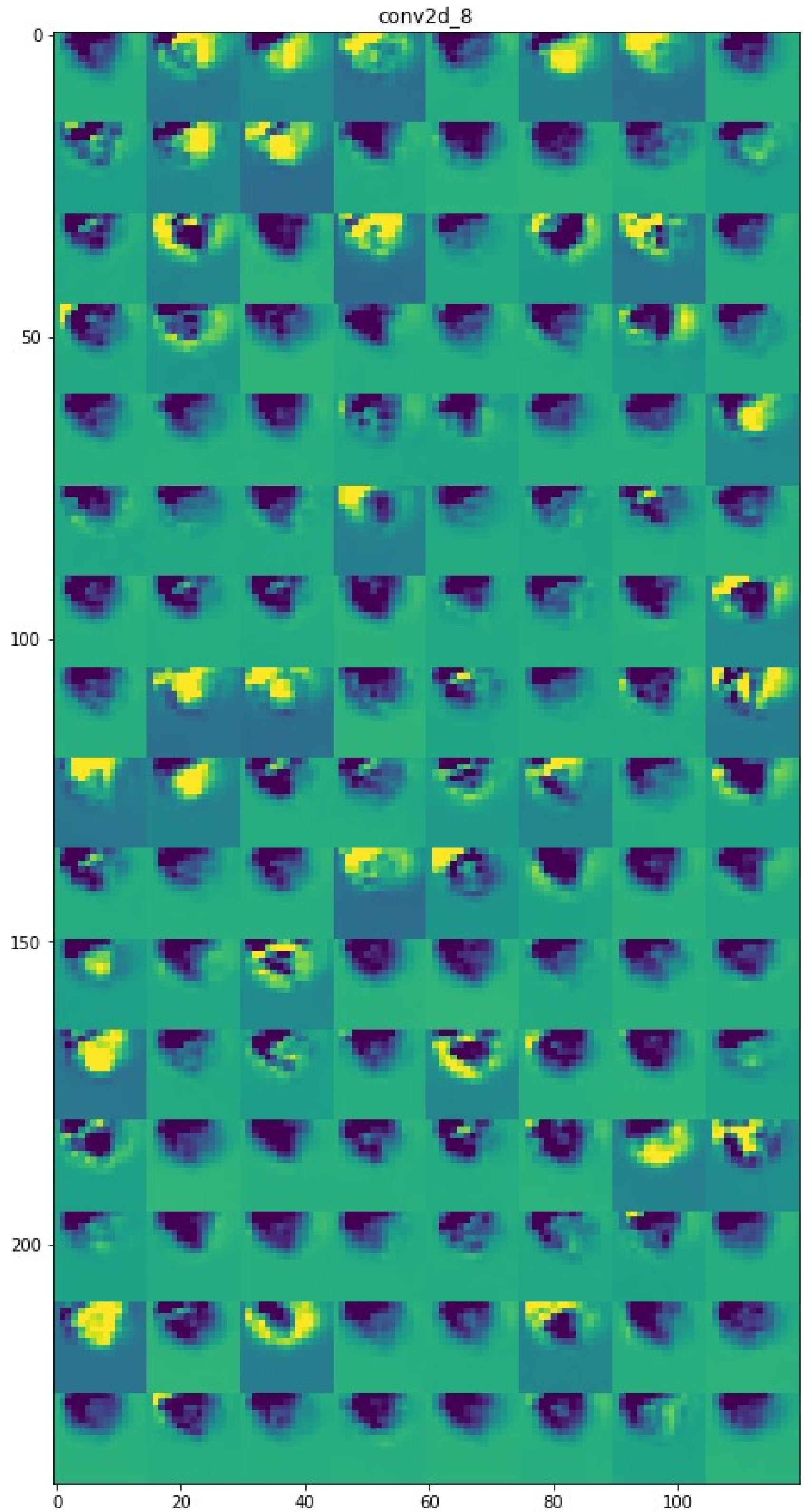

Original Image
0
20
40
60
80
100
120
140
0
25
50
75
100
125

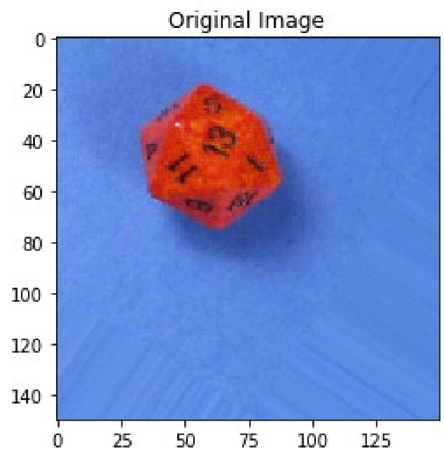

conv2d_5
0
100
200
300
400
500
0
200
400
600
800
1000

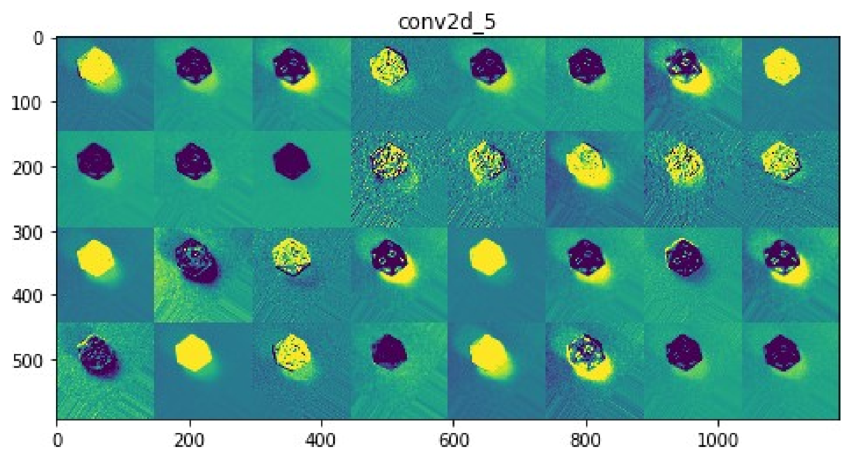


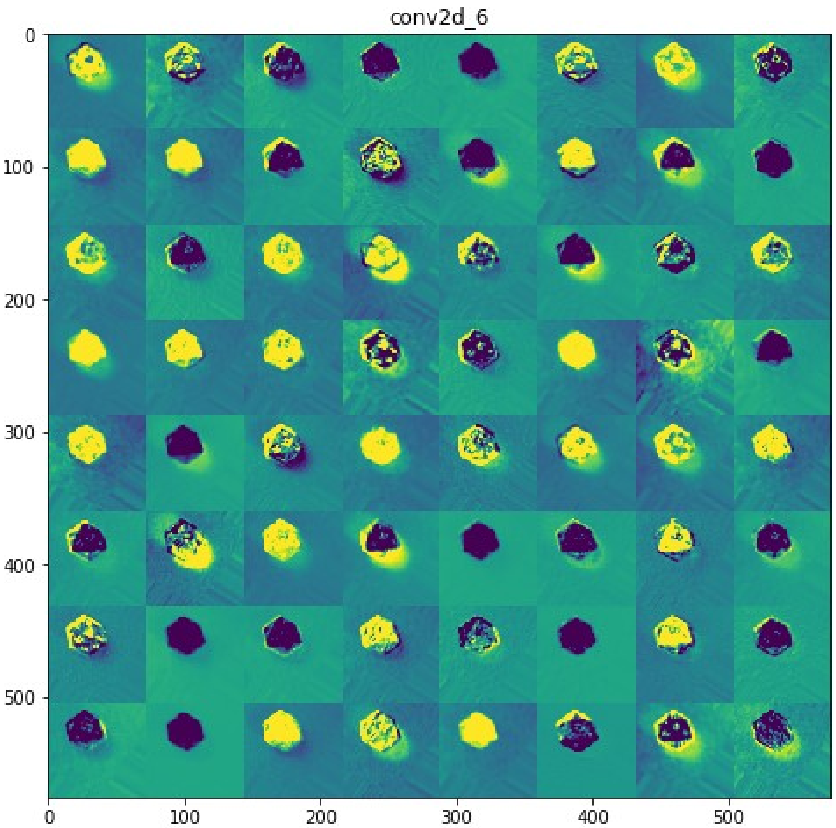
conv2d_6
0
100
200
300
400
500
0
100
200
300
400
500

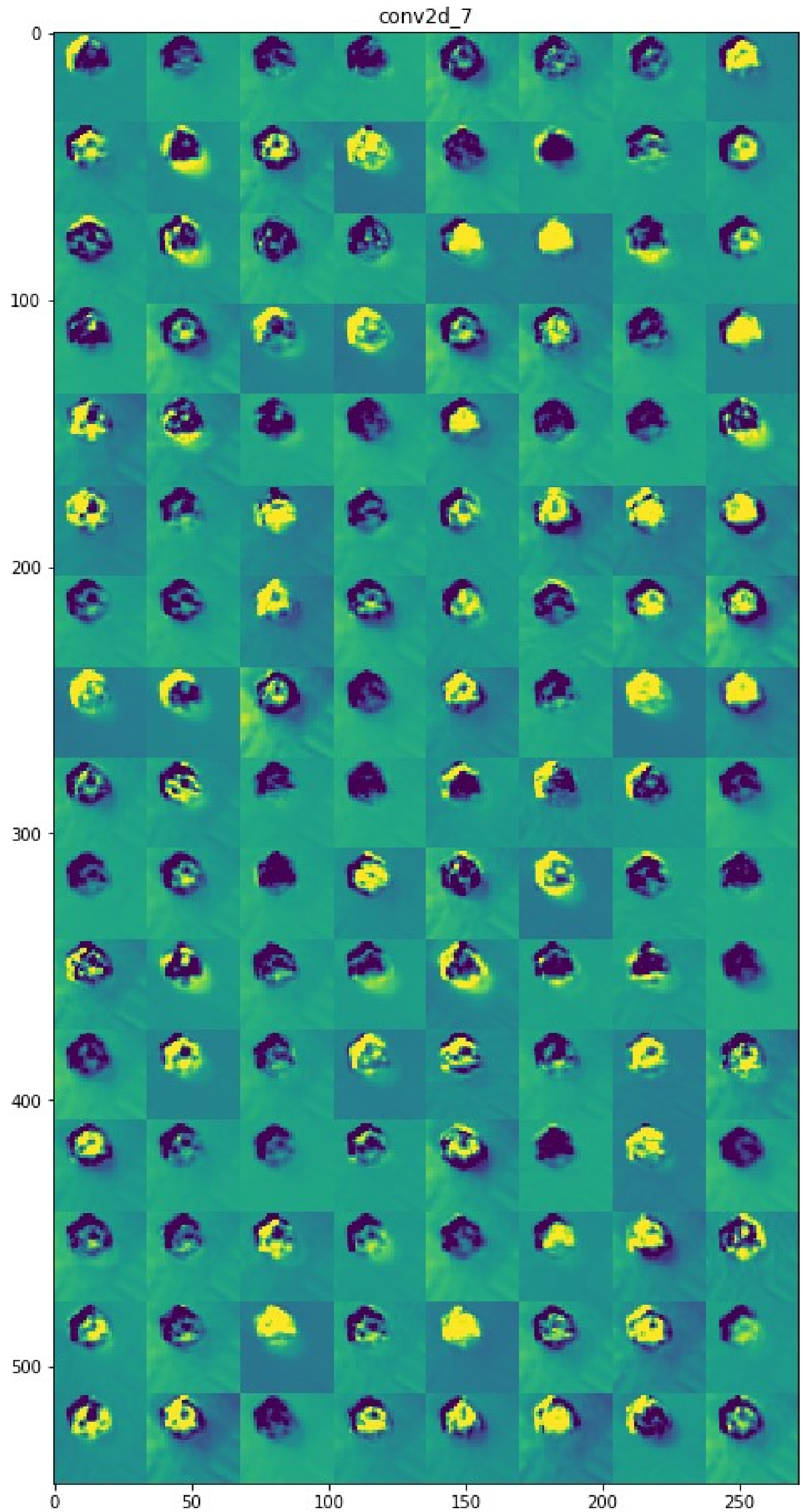

conv2d_7
0
100
200
300
400
500
0
50
100
150
200
250

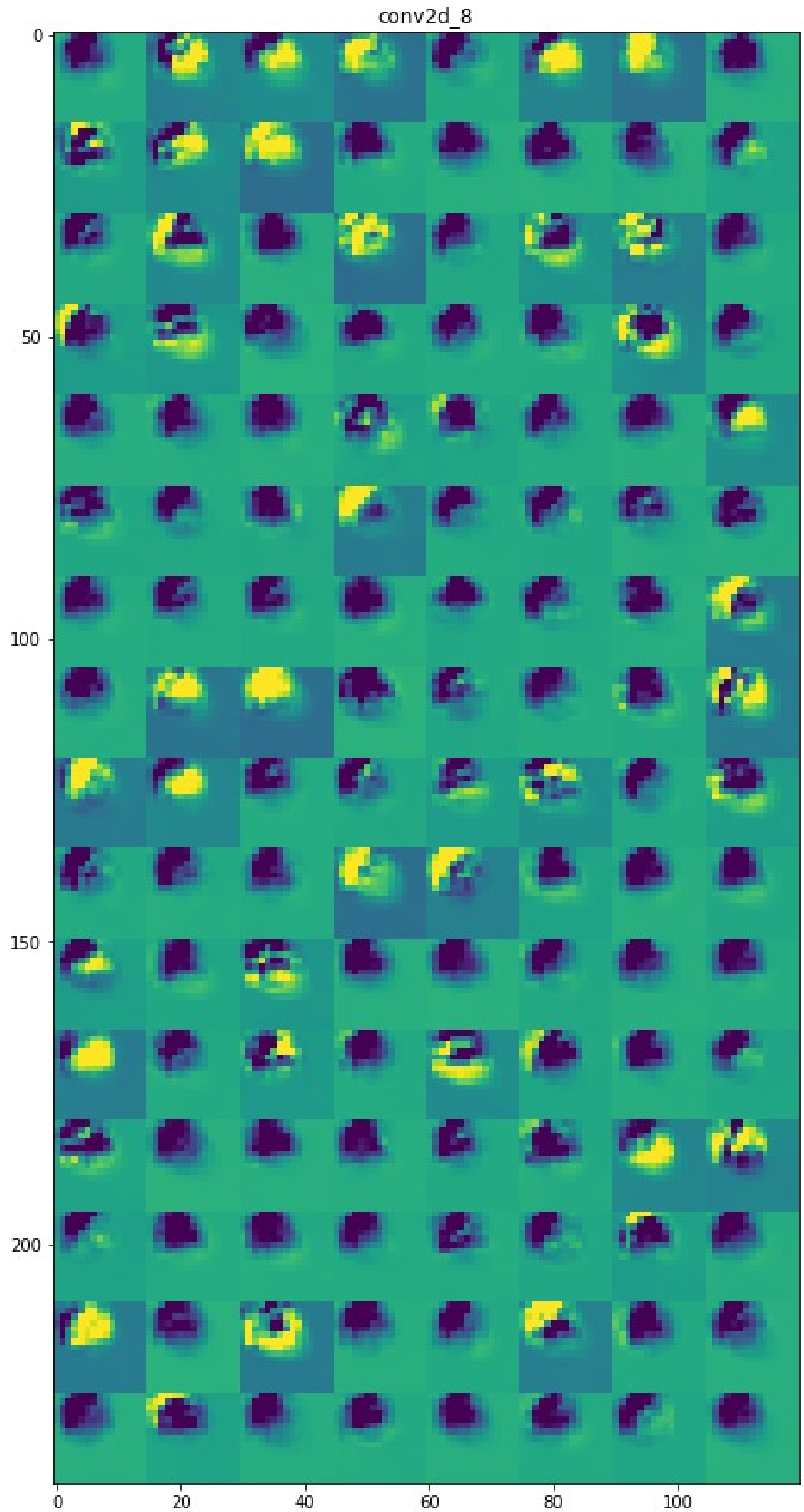

conv2d_8
0
50
100
150
200
0
20
40
60
80
100


Original Image
0
20
40
60
80
100
120
140
0
25
50
75
100
125


conv2d_5
0
100
200
300
400
500
0
200
400
600
800
1000


conv2d_6
0
100
200
300
400
500
0
100
200
300
400
500


conv2d_7
0
100
200
300
400
500
0
50
100
150
200
250


conv2d_8
0
50
100
150
200
0
20
40
60
80
100

Visualizing the activation maps when the CNN makes a prediction is a good way to visualize the type of data that is being collected and how it is abstracted by later layers. For instance when we look at the first layer conv2d_1 seems to collect some fur texture, silhouette, contours, and other such data; then conv2d_2 collects similar data, then for other layers the data becomes more generalized and abstract compared to the first 2 layers due to the dimensionality reduction. We can also see all the filters that do not contribute much or those that have been set to 0 by the dropout function.

Visualizing Filters

In [26]:
```
def deprocess_image(x):
    """utility function to convert a float array into a valid uint8 ima
ge.

    # Arguments
        x: A numpy-array representing the generated image.

    # Returns
        A processed numpy-array, which could be used in e.g. imshow.
    """
    x -= x.mean()
    x /= (x.std() + 1e-5)
    x *= 0.1
    x += 0.5
    x = np.clip(x, 0, 1)
    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

In [27]:
```
def generate_pattern(layer_name, filter_index, size=150):
    layer_output = model.get_layer(layer_name).output
    #print(layer_output.shape)
    loss = K.mean(layer_output[:, :, :, filter_index])
    grads = K.gradients(loss, model.input)[0]
    grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)
    iterate = K.function([model.input], [loss, grads])
    input_img_data = np.random.random((1, size, size, 3)) * 20 + 128.
    #print(input_img_data.shape)
    step = 1.
    for i in range(40):
        loss_value, grads_value = iterate([input_img_data])
        input_img_data += grads_value * step
    img = input_img_data[0]
    return deprocess_image(img)
```

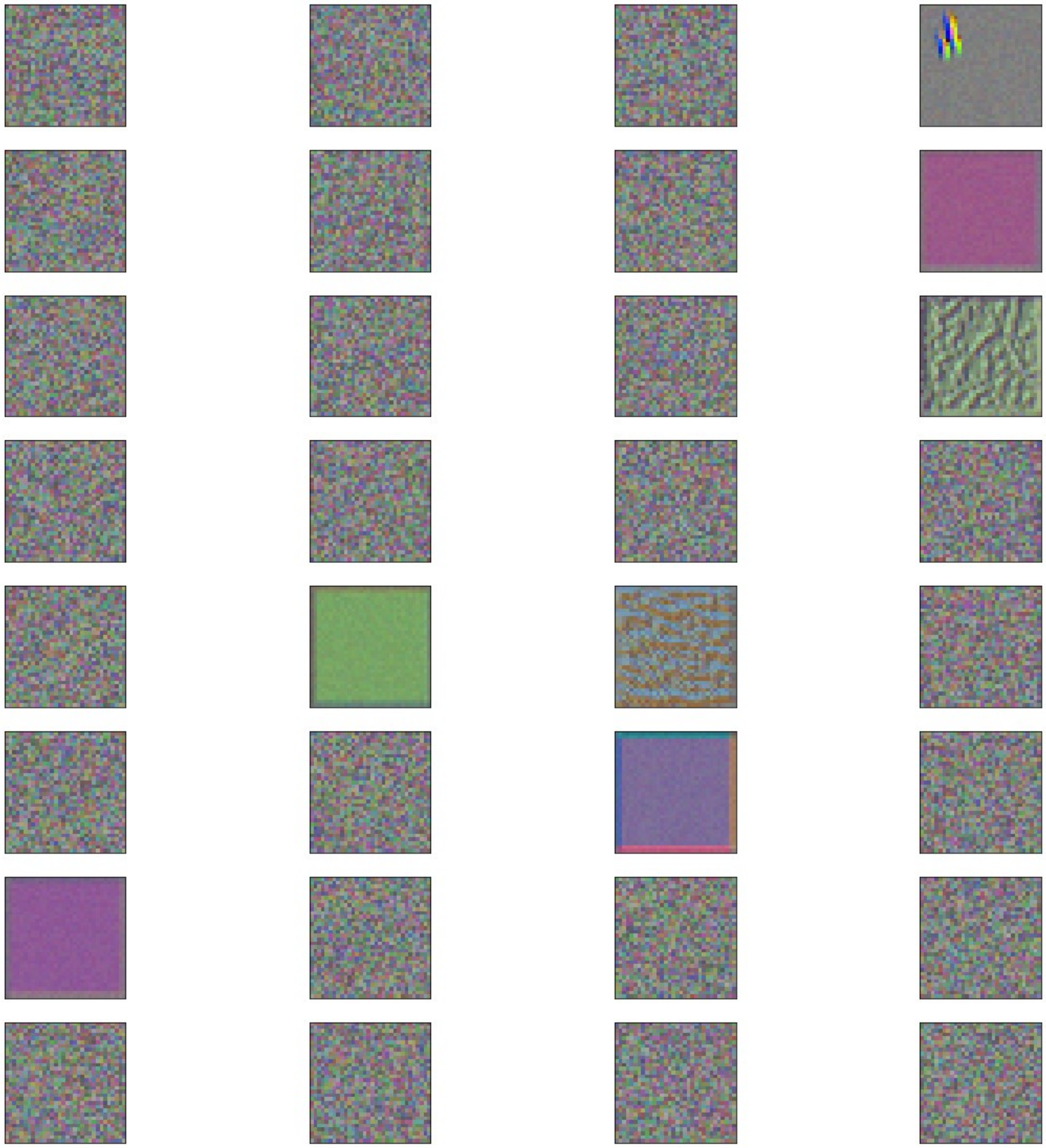
In [28]:
```
#model = load_model('cats_and_dogs_small_2.h5')
layer_names = [layer.name for layer in model.layers]
layer_output = model.get_layer(layer_names[0]).output
print(layer_names)
print(len(layer_names))
```

```
['conv2d_5', 'activation_5', 'max_pooling2d_5', 'dropout_6', 'conv2d_6', 'activation_6', 'max_poolin
g2d_6', 'dropout_7', 'conv2d_7', 'activation_7', 'max_pooling2d_7', 'dropout_8', 'conv2d_8', 'activa
tion_8', 'max_pooling2d_8', 'dropout_9', 'flatten_2', 'dense_3', 'batch_normalization_2', 'dropout_1
0', 'dense_4']
21
```

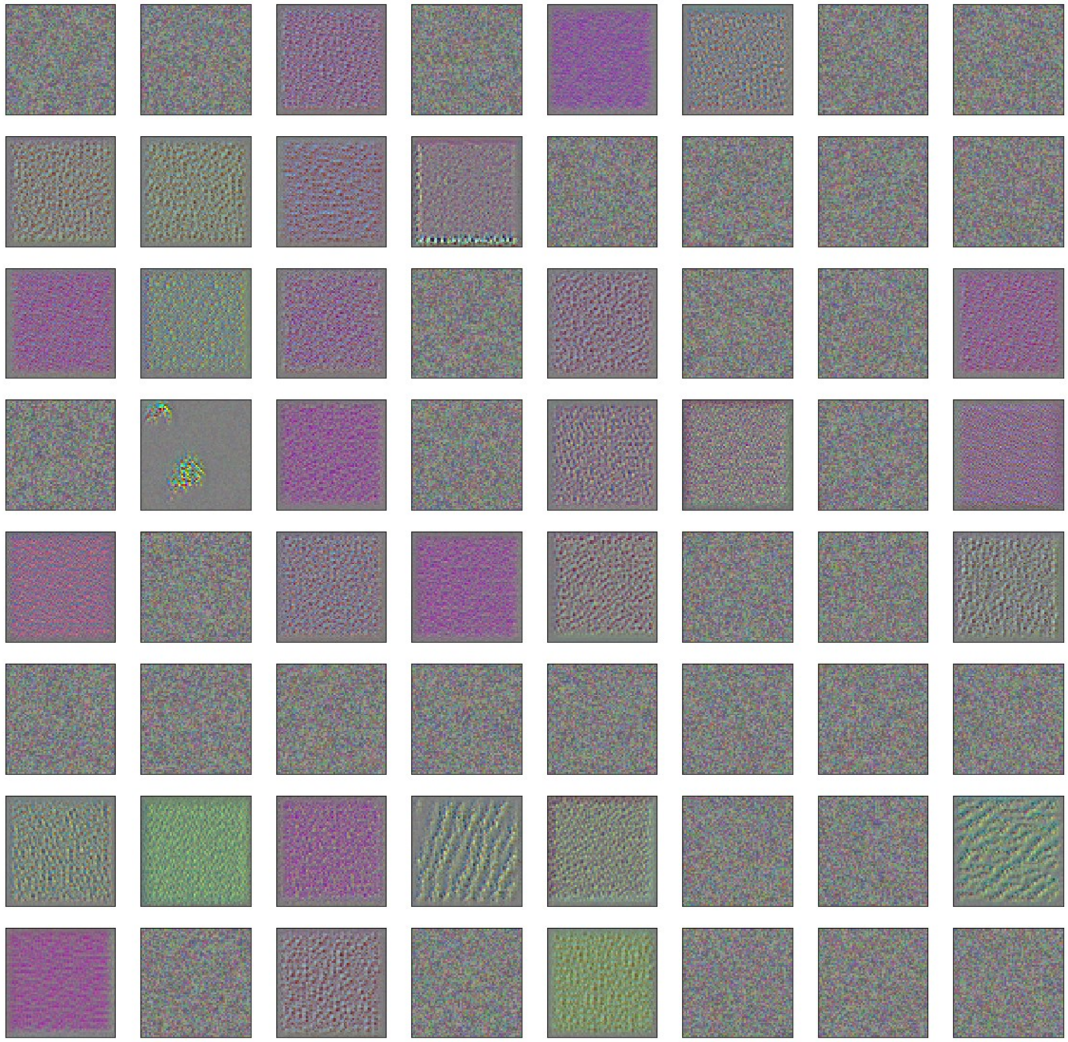
In [30]:

```
# plot the filters for convolutional layers utilizing gradient decompos
itiong
sizes = [32,64,128,128]
num_rows_and_columns = [(8,4),(8,8),(16,8),(16,8),(16,8),(16,8)]
index = 1
for layer_name, size, c_and_n in zip(layer_names[1:14:4], sizes, num_ro
ws_and_columns):
    print(model.get_layer(layer_name).output)
    print(size)
    print(c_and_n)
    plt.figure(figsize=(20,20), facecolor='w')
    plt.title(layer_name)
    for i in range(size):
        filter_img = generate_pattern(layer_name, i, size=size)
        plt.subplot(c_and_n[0],c_and_n[1],index)
        plt.imshow(filter_img)
        plt.xticks([])
        plt.yticks([])
        index = index + 1
    index = 1
    plt.show()
```

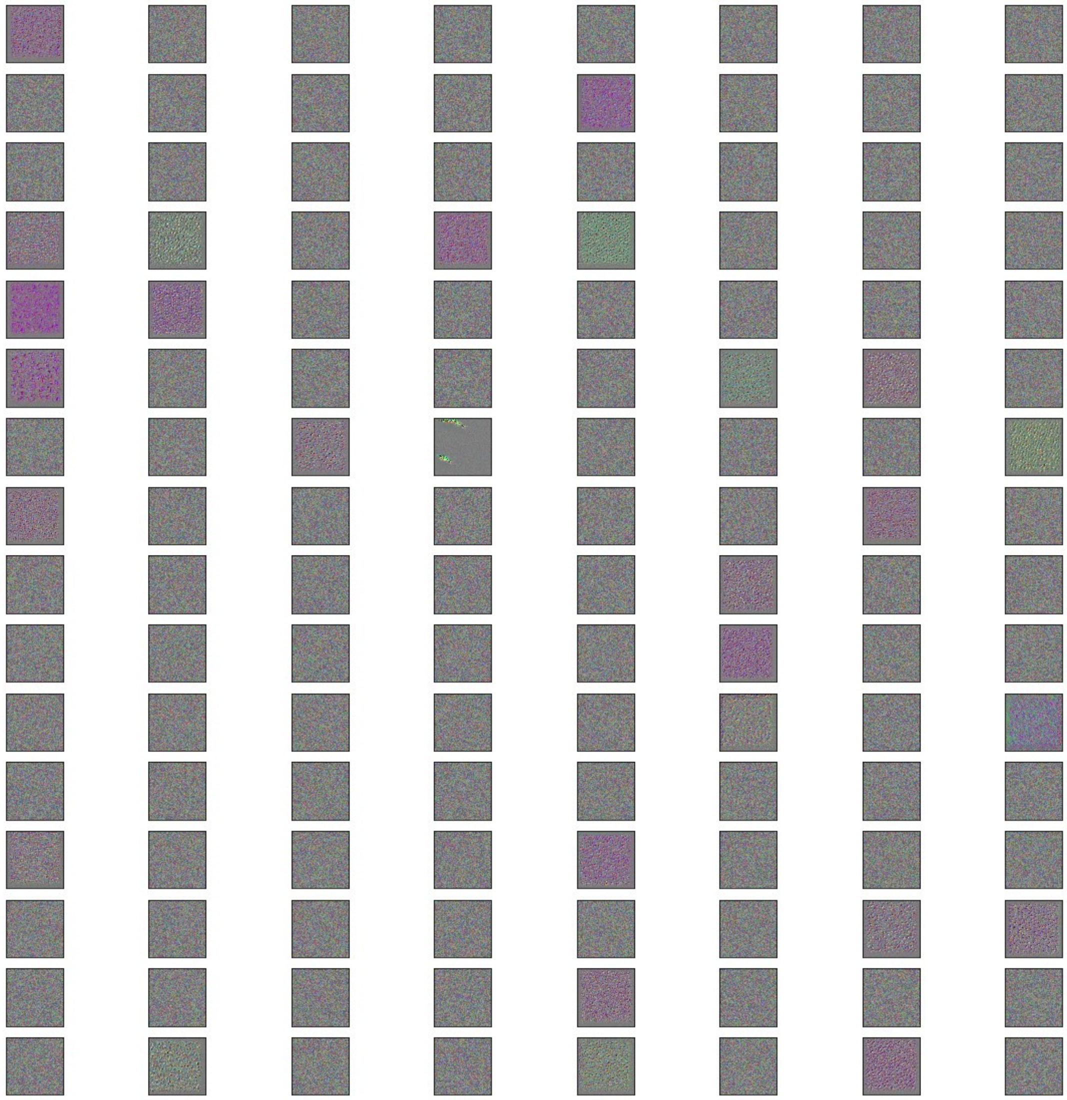
```
Tensor("activation_5/Relu:0", shape=(None, 148, 148, 32), dtype=float32)
32
(8, 4)
```



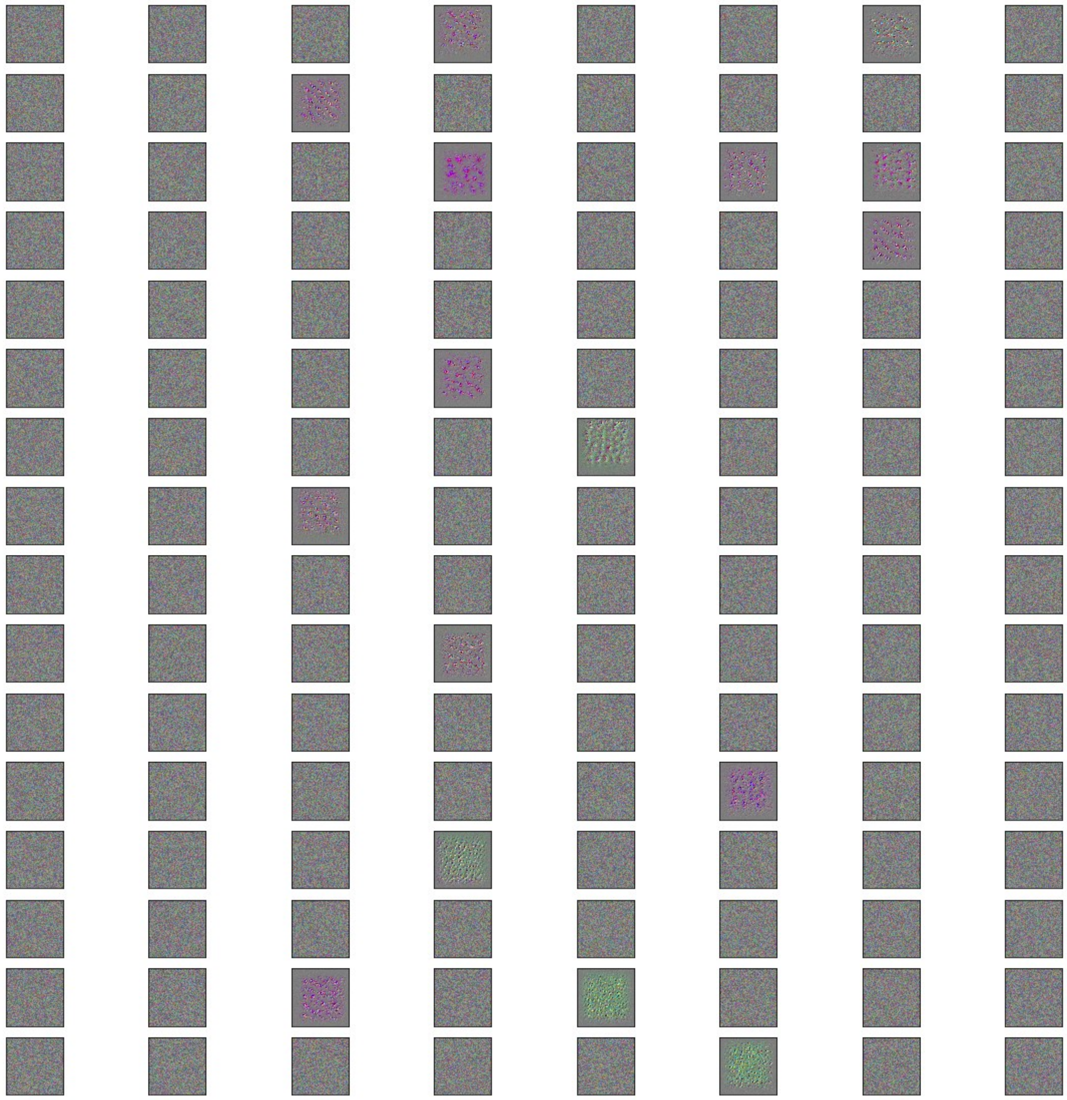
```
Tensor("activation_6/Relu:0", shape=(None, 72, 72, 64), dtype=float32)
64
(8, 8)
```



```
Tensor("activation_7/Relu:0", shape=(None, 34, 34, 128), dtype=float32)
128
(16, 8)
```



```
Tensor("activation_8/Relu:0", shape=(None, 15, 15, 128), dtype=float32)
128
(16, 8)
```



In [ ]: