

```
In [1]: # CSC 732 Final Part 2 Problem 1 Relu Softmax RMSProp 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

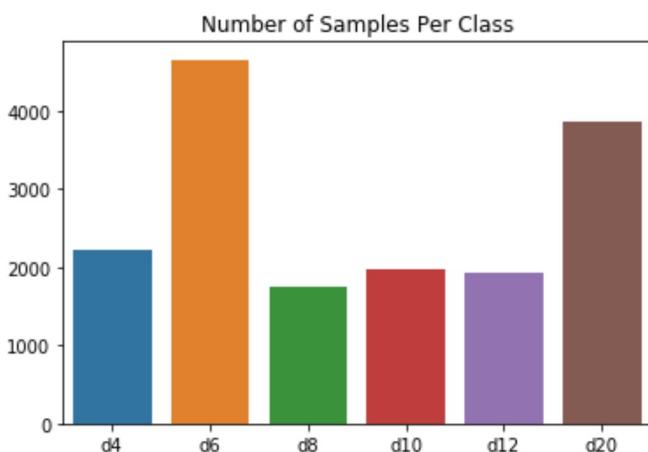
```
In [2]: import keras  
from keras.datasets import mnist  
from keras.models import Sequential  
from keras.layers.normalization import BatchNormalization  
from keras.layers.convolutional import Conv2D, MaxPooling2D  
from keras.layers.core import Flatten, Dropout, Dense, Activation  
from keras.preprocessing.image import ImageDataGenerator  
from keras.utils import to_categorical  
from keras import backend as K  
from skimage import transform  
from sklearn.utils import shuffle  
import matplotlib.image as mpimg  
import matplotlib.pyplot as plt  
import seaborn as sns  
import numpy as np  
import tqdm  
import pickle  
import os  
import glob
```

Using TensorFlow backend.

```
In [3]: # set some hyper parameters for training  
batch_size = 32  
num_classes = 6  
num_epochs = 50
```

View Dataset Frequencies

```
In [4]: # find files in each folder of dataset  
d4_list = os.listdir('Datasets/Dice/dataset/d4')  
d6_list = os.listdir('Datasets/Dice/dataset/d6')  
d8_list = os.listdir('Datasets/Dice/dataset/d8')  
d10_list = os.listdir('Datasets/Dice/dataset/d10')  
d12_list = os.listdir('Datasets/Dice/dataset/d12')  
d20_list = os.listdir('Datasets/Dice/dataset/d20')  
d4_size = len(d4_list)  
d6_size = len(d6_list)  
d8_size = len(d8_list)  
d10_size = len(d10_list)  
d12_size = len(d12_list)  
d20_size = len(d20_list)  
  
In [5]: sns.barplot([0,1,2,3,4,5], [d4_size, d6_size, d8_size, d10_size, d12_size, d20_size])  
plt.xticks(range(6), ['d4', 'd6', 'd8', 'd10', 'd12', 'd20'])  
plt.title("Number of Samples Per Class")  
plt.show()
```

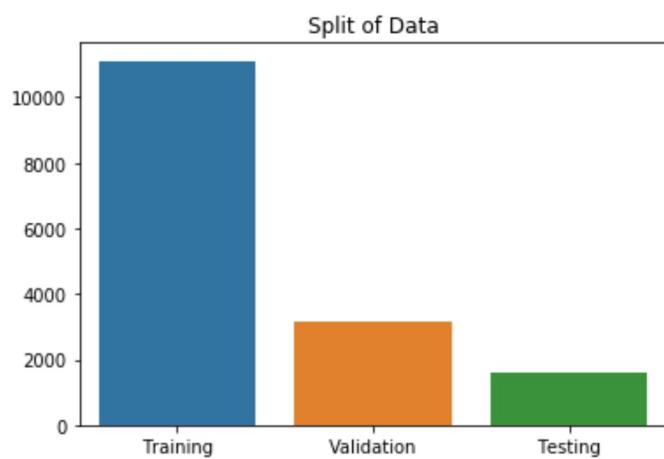


```
In [6]: train_size = 0
validation_size = 0
test_size = 0
# find size of training set
d4_list = os.listdir('Datasets/train/d4')
d6_list = os.listdir('Datasets/train/d6')
d8_list = os.listdir('Datasets/train/d8')
d10_list = os.listdir('Datasets/train/d10')
d12_list = os.listdir('Datasets/train/d12')
d20_list = os.listdir('Datasets/train/d20')
d4_size = len(d4_list)
d6_size = len(d6_list)
d8_size = len(d8_list)
d10_size = len(d10_list)
d12_size = len(d12_list)
d20_size = len(d20_list)
train_size = d4_size + d6_size + d8_size + d10_size + d12_size + d20_size

# find size of validation set
d4_list = os.listdir('Datasets/val/d4')
d6_list = os.listdir('Datasets/val/d6')
d8_list = os.listdir('Datasets/val/d8')
d10_list = os.listdir('Datasets/val/d10')
d12_list = os.listdir('Datasets/val/d12')
d20_list = os.listdir('Datasets/val/d20')
d4_size = len(d4_list)
d6_size = len(d6_list)
d8_size = len(d8_list)
d10_size = len(d10_list)
d12_size = len(d12_list)
d20_size = len(d20_list)
validation_size = d4_size + d6_size + d8_size + d10_size + d12_size + d20_size

# find size of test set
d4_list = os.listdir('Datasets/test/d4')
d6_list = os.listdir('Datasets/test/d6')
d8_list = os.listdir('Datasets/test/d8')
d10_list = os.listdir('Datasets/test/d10')
d12_list = os.listdir('Datasets/test/d12')
d20_list = os.listdir('Datasets/test/d20')
d4_size = len(d4_list)
d6_size = len(d6_list)
d8_size = len(d8_list)
d10_size = len(d10_list)
d12_size = len(d12_list)
```

```
In [7]: sns.barplot([0,1,2], [train_size, validation_size, test_size])
plt.xticks(range(3), ['Training', 'Validation', 'Testing'])
plt.title("Split of Data")
plt.show()
```

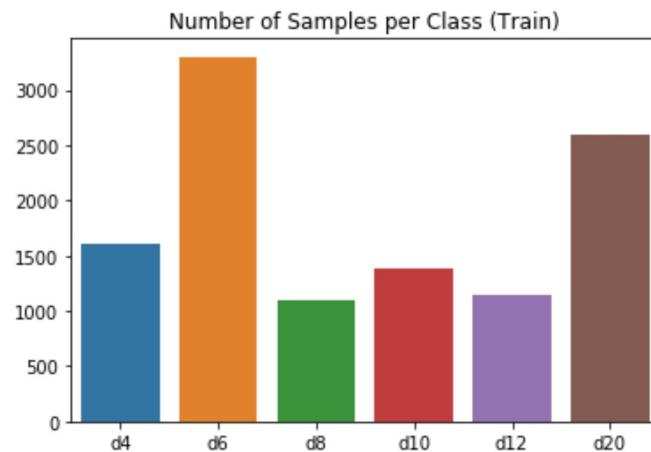


```
In [8]: train_size = 0
validation_size = 0
test_size = 0
# find size of training set
d4_list = os.listdir('Datasets/train/d4')
d6_list = os.listdir('Datasets/train/d6')
d8_list = os.listdir('Datasets/train/d8')
d10_list = os.listdir('Datasets/train/d10')
d12_list = os.listdir('Datasets/train/d12')
d20_list = os.listdir('Datasets/train/d20')
d4_size = len(d4_list)
d6_size = len(d6_list)
d8_size = len(d8_list)
d10_size = len(d10_list)
d12_size = len(d12_list)
d20_size = len(d20_list)
print("Samples for Training Set")
print("D4 : ", d4_size)
print("D6 : ", d6_size)
print("D8 : ", d8_size)
print("D10 : ", d10_size)
print("D12 : ", d12_size)
print("D20 : ", d20_size)
train_size = d4_size + d6_size + d8_size + d10_size + d12_size + d20_size
sns.barplot([0,1,2,3,4,5], [d4_size, d6_size, d8_size, d10_size, d12_size, d20_size])
plt.xticks(range(6), ['d4', 'd6', 'd8', 'd10', 'd12', 'd20'])
plt.title("Number of Samples per Class (Train)")
plt.show()

# find size of validation set
d4_list = os.listdir('Datasets/val/d4')
d6_list = os.listdir('Datasets/val/d6')
d8_list = os.listdir('Datasets/val/d8')
d10_list = os.listdir('Datasets/val/d10')
d12_list = os.listdir('Datasets/val/d12')
d20_list = os.listdir('Datasets/val/d20')
d4_size = len(d4_list)
d6_size = len(d6_list)
d8_size = len(d8_list)
d10_size = len(d10_list)
d12_size = len(d12_list)
d20_size = len(d20_list)
print("Samples for Validation Set")
print("D4 : ", d4_size)
print("D6 : ", d6_size)
print("D8 : ", d8_size)
print("D10 : ", d10_size)
print("D12 : ", d12_size)
print("D20 : ", d20_size)
```

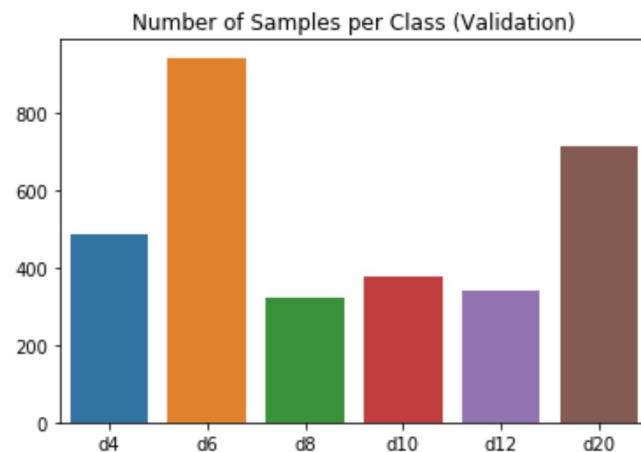
Samples for Training Set

D4 : 1603
D6 : 3301
D8 : 1096
D10 : 1379
D12 : 1142
D20 : 2590



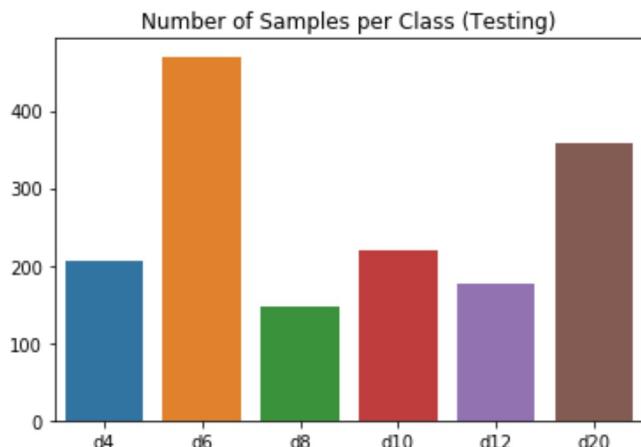
Samples for Validation Set

D4 : 484
D6 : 942
D8 : 321
D10 : 378
D12 : 338
D20 : 712



Samples for Testing Set

D4 : 205
D6 : 470
D8 : 147
D10 : 219
D12 : 176
D20 : 359



Load Dataset

```
In [9]: # create image generator
datagenerator = 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)
```

```
In [10]: # lists to hold train images and train labels
X_train = []
Y_train = []

# load in images and labels based on how generators label data
# for each file in class folder
for file in tqdm.tqdm(glob.glob('Datasets/train/d4/*.jpg')):
    # append original image
    # load image
    image = plt.imread(file)
    # resize to 100x100
    image = transform.resize(image, (100,100))
    # append image and label to appropriate lists
    X_train.append(image)
    Y_train.append(3)

    # create some random images, and append to list
    img_tensor = image
    img_tensor = datagenerator.apply_transform(img_tensor, datagenerator
r.get_random_transform(img_tensor.shape))
    X_train.append(img_tensor)
    Y_train.append(3)

# for each file in class folder
for file in tqdm.tqdm(glob.glob('Datasets/train/d6/*.jpg')):
    # load image
    image = plt.imread(file)
    # resize to 100x100
    image = transform.resize(image, (100,100))
    # append image and label to appropriate lists
    X_train.append(image)
    Y_train.append(4)

# for each file in class folder
for file in tqdm.tqdm(glob.glob('Datasets/train/d8/*.jpg')):
    # load image
    image = plt.imread(file)
    # resize to 100x100
    image = transform.resize(image, (100,100))
    # append image and label to appropriate lists
    X_train.append(image)
    Y_train.append(5)

    # create some random images, and append to list
    img_tensor = image
    img_tensor = datagenerator.apply_transform(img_tensor, datagenerator
r.get_random_transform(img_tensor.shape))
    X_train.append(img_tensor)
    Y_train.append(5)
```

```
100%|██████████| 1603/1603 [00:3
5<00:00, 45.38it/s]
100%|██████████| 3301/3301 [01:0
8<00:00, 48.15it/s]
100%|██████████| 1096/1096 [00:2
3<00:00, 45.74it/s]
100%|██████████| 1379/1379 [00:3
1<00:00, 43.43it/s]
100%|██████████| 1142/1142 [00:2
6<00:00, 43.04it/s]
100%|██████████| 2590/2590 [00:5
8<00:00, 44.45it/s]
```

```
In [11]: # lists to hold val images and val labels
X_val = []
Y_val = []

# load in images and labels based on how generators label data
# for each file in class folder
for file in tqdm.tqdm(glob.glob('Datasets/val/d4/*.jpg')):
    # append original image
    # load image
    image = plt.imread(file)
    # resize to 100x100
    image = transform.resize(image, (100,100))
    # append image and label to appropriate lists
    X_val.append(image)
    Y_val.append(3)

    # create some random images, and append to list
    img_tensor = image
    img_tensor = datagenerator.apply_transform(img_tensor, datagenerator
r.get_random_transform(img_tensor.shape))
    X_val.append(img_tensor)
    Y_val.append(3)

# for each file in class folder
for file in tqdm.tqdm(glob.glob('Datasets/val/d6/*.jpg')):
    # load image
    image = plt.imread(file)
    # resize to 100x100
    image = transform.resize(image, (100,100))
    # append image and label to appropriate lists
    X_val.append(image)
    Y_val.append(4)

# for each file in class folder
for file in tqdm.tqdm(glob.glob('Datasets/val/d8/*.jpg')):
    # load image
    image = plt.imread(file)
    # resize to 100x100
    image = transform.resize(image, (100,100))
    # append image and label to appropriate lists
    X_val.append(image)
    Y_val.append(5)

    # create some random images, and append to list
    img_tensor = image
    img_tensor = datagenerator.apply_transform(img_tensor, datagenerator
r.get_random_transform(img_tensor.shape))
    X_val.append(img_tensor)
    Y_val.append(5)
```

100%| 484/484 [00:1
1<00:00, 43.82it/s]
100%| 942/942 [00:1
9<00:00, 47.49it/s]
100%| 321/321 [00:0
7<00:00, 43.08it/s]
100%| 378/378 [00:0
8<00:00, 42.51it/s]
100%| 338/338 [00:0
8<00:00, 41.98it/s]
100%| 712/712 [00:1
6<00:00, 43.75it/s]

```
In [12]: # lists to hold test images and test labels
X_test = []
Y_test = []

# load in images and labels based on how generators label data
# for each file in class folder
for file in tqdm.tqdm(glob.glob('Datasets/test/d4/*.jpg')):
    # load image
    image = plt.imread(file)
    # resize to 100x100
    image = transform.resize(image, (100,100))
    # append image and label to appropriate lists
    X_test.append(image)
    Y_test.append(3)

# for each file in class folder
for file in tqdm.tqdm(glob.glob('Datasets/test/d6/*.jpg')):
    # load image
    image = plt.imread(file)
    # resize to 100x100
    image = transform.resize(image, (100,100))
    # append image and label to appropriate lists
    X_test.append(image)
    Y_test.append(4)

# for each file in class folder
for file in tqdm.tqdm(glob.glob('Datasets/test/d8/*.jpg')):
    # load image
    image = plt.imread(file)
    # resize to 100x100
    image = transform.resize(image, (100,100))
    # append image and label to appropriate lists
    X_test.append(image)
    Y_test.append(5)

# for each file in class folder
for file in tqdm.tqdm(glob.glob('Datasets/test/d10/*.jpg')):
    # load image
    image = plt.imread(file)
    # resize to 100x100
    image = transform.resize(image, (100,100))
    # append image and label to appropriate lists
    X_test.append(image)
    Y_test.append(0)

# for each file in class folder
for file in tqdm.tqdm(glob.glob('Datasets/test/d12/*.jpg')):
    # load image
```

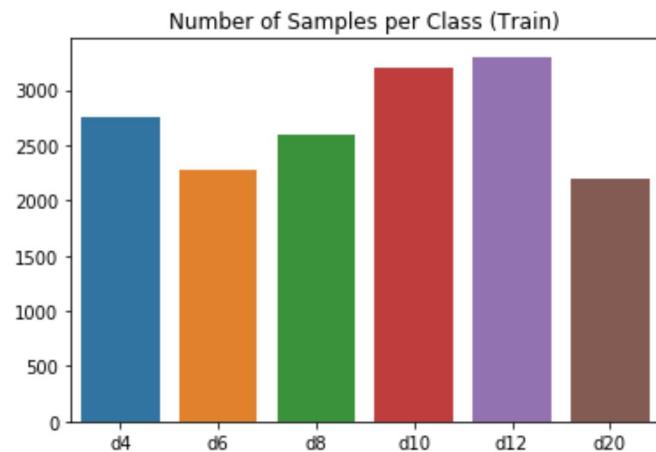
100%| 205/205 [00:0
4<00:00, 45.91it/s]
100%| 470/470 [00:0
9<00:00, 47.08it/s]
100%| 147/147 [00:0
2<00:00, 49.49it/s]
100%| 219/219 [00:0
4<00:00, 48.54it/s]
100%| 176/176 [00:0
3<00:00, 47.16it/s]
100%| 359/359 [00:0
8<00:00, 44.24it/s]

```
In [13]: # find size of training set
unique, counts = np.unique(Y_train, return_counts=True)
d4_size, d6_size, d8_size, d10_size, d12_size, d20_size = counts
print("Samples for Training Set")
print("D4 : ", d4_size)
print("D6 : ", d6_size)
print("D8 : ", d8_size)
print("D10 : ", d10_size)
print("D12 : ", d12_size)
print("D20 : ", d20_size)
train_size = d4_size + d6_size + d8_size + d10_size + d12_size + d20_size
sns.barplot([0,1,2,3,4,5], [d4_size, d6_size, d8_size, d10_size, d12_size, d20_size])
plt.xticks(range(6), ['d4', 'd6', 'd8', 'd10', 'd12', 'd20'])
plt.title("Number of Samples per Class (Train)")
plt.show()

# find size of validation set
unique, counts = np.unique(Y_val, return_counts=True)
# find size of training set
d4_size, d6_size, d8_size, d10_size, d12_size, d20_size = counts
print("Samples for Validation Set")
print("D4 : ", d4_size)
print("D6 : ", d6_size)
print("D8 : ", d8_size)
print("D10 : ", d10_size)
print("D12 : ", d12_size)
print("D20 : ", d20_size)
validation_size = d4_size + d6_size + d8_size + d10_size + d12_size + d20_size
sns.barplot([0,1,2,3,4,5], [d4_size, d6_size, d8_size, d10_size, d12_size, d20_size])
plt.xticks(range(6), ['d4', 'd6', 'd8', 'd10', 'd12', 'd20'])
plt.title("Number of Samples per Class (Validation)")
plt.show()
```

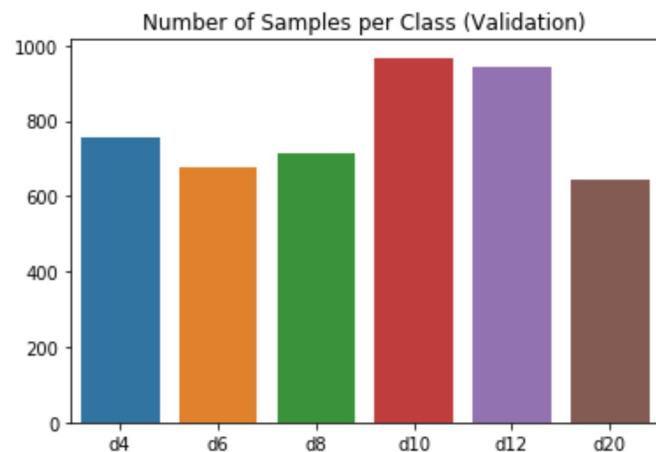
Samples for Training Set

D4 : 2758
D6 : 2284
D8 : 2590
D10 : 3206
D12 : 3301
D20 : 2192



Samples for Validation Set

D4 : 756
D6 : 676
D8 : 712
D10 : 968
D12 : 942
D20 : 642



In [14]: `from sklearn.preprocessing import OneHotEncoder`

```
In [16]: #shuffle datasets
X_train, Y_train = shuffle(X_train, Y_train)
X_val, Y_val = shuffle(X_val, Y_val)
X_test, Y_test = shuffle(X_test, Y_test)
# convert class labels to categorical
encoder = OneHotEncoder(sparse=False)
encoder.fit(np.expand_dims(Y_train, axis=1))
Y_train_cat = encoder.transform(np.expand_dims(Y_train, axis=1))
Y_val_cat = encoder.transform(np.expand_dims(Y_val, axis=1))
Y_test_cat = encoder.transform(np.expand_dims(Y_test, axis=1))
```

Build/Train Model

```
In [17]: # build the model
model = Sequential()
model.add(Conv2D(32, (3, 3), kernel_initializer='glorot_uniform', input_shape=(100, 100, 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(Flatten())
model.add(Dense(512, kernel_initializer='glorot_uniform', use_bias=False))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Dropout(rate=0.2))
model.add(Dense(6, kernel_initializer='glorot_uniform'))
model.add(Activation('softmax'))
```

```
In [18]: # compile the model to use binary crossentropy and use RMSProp optimize
r
model.compile(loss='categorical_crossentropy',optimizer='rmsprop',metrics=['acc'])
```

```
In [19]: history = model.fit(X_train,  
                           Y_train_cat,  
                           batch_size=batch_size,  
                           epochs=num_epochs,  
                           verbose=1, shuffle=True, validation_data=[X_val, Y_  
                           val_cat])
```

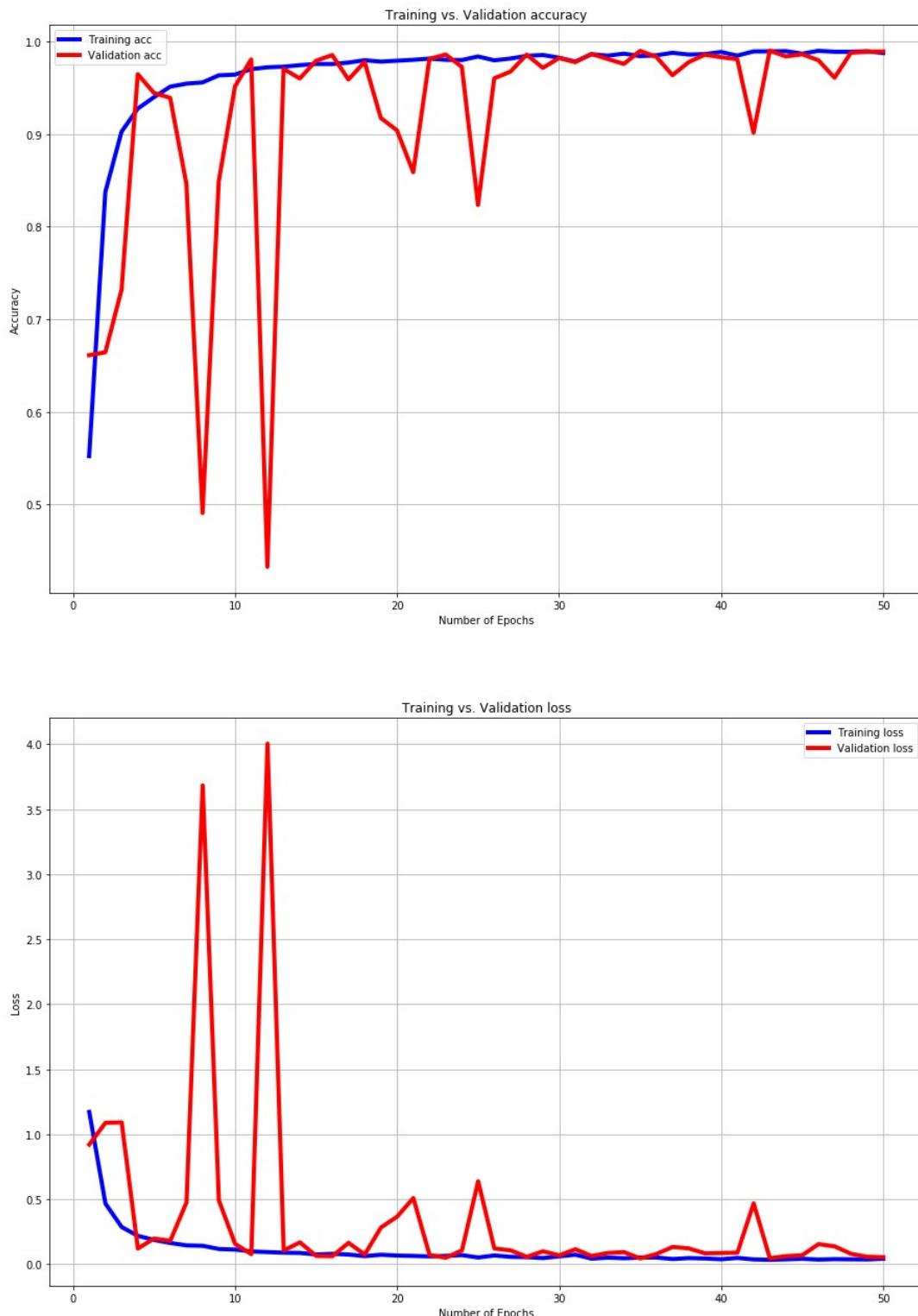
```
Train on 16331 samples, validate on 4696 samples
Epoch 1/50
16331/16331 [=====] - 15s 922us/step - loss: 1.1713 - acc: 0.5520 - val_loss: 0.9205 - val_acc: 0.6610
Epoch 2/50
16331/16331 [=====] - 13s 789us/step - loss: 0.4664 - acc: 0.8374 - val_loss: 1.0879 - val_acc: 0.6642
Epoch 3/50
16331/16331 [=====] - 13s 778us/step - loss: 0.2865 - acc: 0.9023 - val_loss: 1.0909 - val_acc: 0.7317
Epoch 4/50
16331/16331 [=====] - 13s 769us/step - loss: 0.2187 - acc: 0.9273 - val_loss: 0.1212 - val_acc: 0.9644
Epoch 5/50
16331/16331 [=====] - 13s 766us/step - loss: 0.1870 - acc: 0.9396 - val_loss: 0.1979 - val_acc: 0.9442
Epoch 6/50
16331/16331 [=====] - 12s 762us/step - loss: 0.1638 - acc: 0.9510 - val_loss: 0.1808 - val_acc: 0.9389
Epoch 7/50
16331/16331 [=====] - 13s 767us/step - loss: 0.1456 - acc: 0.9542 - val_loss: 0.4756 - val_acc: 0.8463
Epoch 8/50
16331/16331 [=====] - 13s 770us/step - loss: 0.1421 - acc: 0.9555 - val_loss: 3.6821 - val_acc: 0.4906
Epoch 9/50
16331/16331 [=====] - 12s 762us/step - loss: 0.1176 - acc: 0.9632 - val_loss: 0.4905 - val_acc: 0.8488
Epoch 10/50
16331/16331 [=====] - 12s 755us/step - loss: 0.1134 - acc: 0.9640 - val_loss: 0.1570 - val_acc: 0.9512
Epoch 11/50
16331/16331 [=====] - 12s 761us/step - loss: 0.0988 - acc: 0.9699 - val_loss: 0.0782 - val_acc: 0.9804
Epoch 12/50
16331/16331 [=====] - 12s 762us/step - loss: 0.0938 - acc: 0.9718 - val_loss: 4.0032 - val_acc: 0.4325
Epoch 13/50
16331/16331 [=====] - 12s 759us/step - loss: 0.0890 - acc: 0.9726 - val_loss: 0.1063 - val_acc: 0.9702
Epoch 14/50
16331/16331 [=====] - 12s 763us/step - loss: 0.0871 - acc: 0.9742 - val_loss: 0.1686 - val_acc: 0.9598
Epoch 15/50
16331/16331 [=====] - 12s 760us/step - loss: 0.0755 - acc: 0.9755 - val_loss: 0.0644 - val_acc: 0.9789
Epoch 16/50
16331/16331 [=====] - 12s 758us/step - loss: 0.0803 - acc: 0.9754 - val_loss: 0.0620 - val_acc: 0.9851
Epoch 17/50
16331/16331 [=====] - 13s 786us/step - loss: 0.0752 - acc: 0.9770 - val_loss: 0.1650 - val_acc: 0.9587
Epoch 18/50
16331/16331 [=====] - 13s 766us/step - loss: 0.0626 - acc: 0.9795 - val_loss: 0.0750 - val_acc: 0.9776
Epoch 19/50
16331/16331 [=====] - 12s 762us/step - loss: 0.0734 - acc: 0.9780 - val_loss: 0.2828 - val_acc: 0.9172
Epoch 20/50
16331/16331 [=====] - 12s 751us/step - loss: 0.0671 - acc: 0.9790 - val_loss: 0.3664 - val_acc: 0.9037
Epoch 21/50
16331/16331 [=====] - 12s 761us/step - loss: 0.0644 - acc: 0.9800 - val_loss:
```

```
In [20]: # save the model for future use  
model.save('dice_relu_softmax_RMSProp_less_steps.h5')
```

Evaluate Performance

```
In [21]: # Evaluate classifier with test image generator  
evaluation = model.evaluate(X_test, Y_test_cat, verbose=1)  
print("Test Loss: %.4f" % evaluation[0])  
print("Test Accuracy: %.4f" % (evaluation[1]*100))  
  
1576/1576 [=====] - 1s 421us/step  
Test Loss: 0.0375  
Test Accuracy: 99.4924
```

```
In [22]: # 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', figsize=(15,10))
plt.plot(epochs, acc, 'b', label='Training acc', lw=4)
plt.plot(epochs, val_acc, 'r', label='Validation acc', lw=4)
plt.title('Training vs. Validation accuracy')
plt.xlabel('Number of Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid()
plt.figure(facecolor='w', figsize=(15,10))
plt.plot(epochs, loss, 'b', label='Training loss', lw=4)
plt.plot(epochs, val_loss, 'r', label='Validation loss', lw=4)
plt.title('Training vs. Validation loss')
plt.xlabel('Number of Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid()
plt.show()
```



```
In [23]: # shuffle dataset manually because model did it before
X_test, Y_test_cat = shuffle(X_test, Y_test_cat)
# predict labels
predictions = model.predict(X_test)
```

```
In [24]: from sklearn.metrics import r2_score, confusion_matrix, classification_
report
from sklearn.metrics import mean_squared_error, roc_curve, auc, confusi
on_matrix
from scipy import interp
from itertools import cycle
```

```
In [25]: # Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(6):
    fpr[i], tpr[i], _ = roc_curve(Y_test_cat[:, i], predictions[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(Y_test_cat.ravel(), predictions.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# First aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(6)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(6):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= 6

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

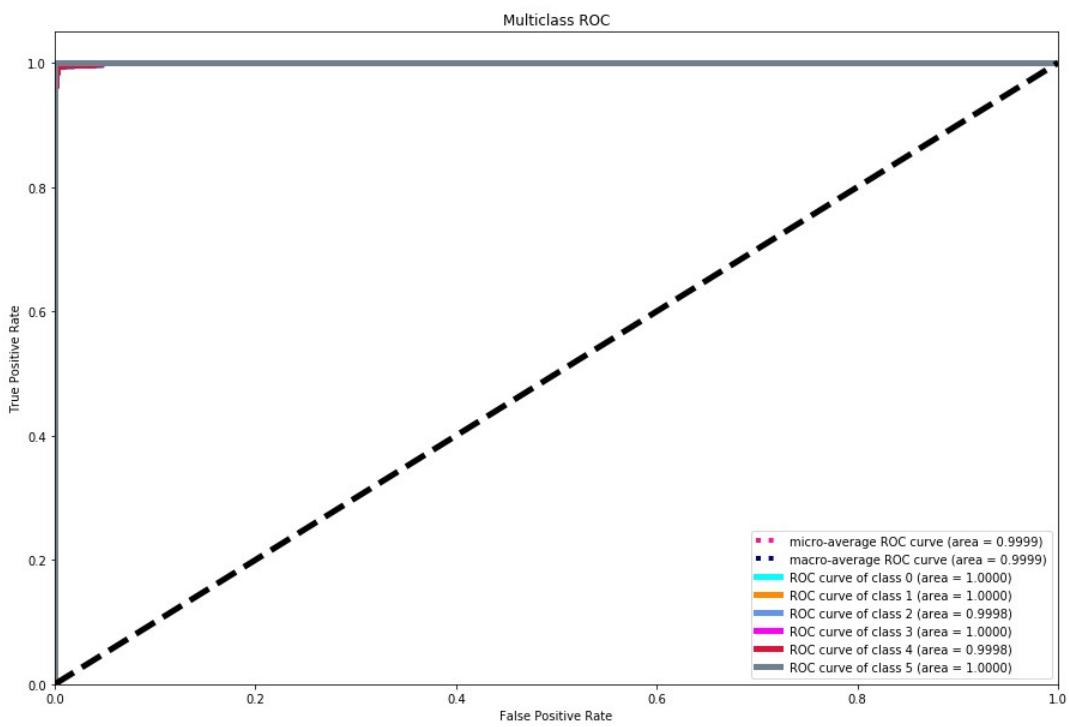
# set line width of graphs
lw = 5

# Plot all ROC curves
plt.figure(figsize=(15,10), facecolor='w')
plt.plot(fpr["micro"], tpr["micro"],
          label='micro-average ROC curve (area = {0:0.4f})'.format(roc_auc["micro"]),
          color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
          label='macro-average ROC curve (area = {0:0.4f})'.format(roc_auc["macro"]),
          color='navy', linestyle=':', linewidth=4)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'fuchsia', 'crimson', 'slategray'])
for i, color in zip(range(6), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
              label='ROC curve of class {0} (AUC = {1:0.4f})'.format(i, roc_auc[i]))
```

C:\Users\kluse\Anaconda3\envs\MLearning\lib\site-packages\ipykernel_launcher.py:19: DeprecationWarning: scipy.interp is deprecated and will be removed in SciPy 2.0.0, use numpy.interp instead



```
In [26]: # plot confusion matrix
plt.figure(figsize=(15,10), facecolor='w')
numerical_Y_test = encoder.inverse_transform(Y_test_cat)[:,0]
numerical_predictions = np.argmax(predictions, axis=-1)
sns.heatmap(confusion_matrix(numerical_Y_test, numerical_predictions, normalize='all'), fmt='.4f', annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x238eb822630>
```



```
In [27]: # print classification report to find Precision, Recall, and F1-score
print(classification_report(numerical_Y_test, numerical_predictions, digits=4))
```

	precision	recall	f1-score	support
0	0.9864	0.9954	0.9909	219
1	0.9944	1.0000	0.9972	176
2	1.0000	0.9889	0.9944	359
3	1.0000	1.0000	1.0000	205
4	0.9915	0.9936	0.9926	470
5	1.0000	1.0000	1.0000	147
accuracy			0.9949	1576
macro avg	0.9954	0.9963	0.9958	1576
weighted avg	0.9950	0.9949	0.9949	1576

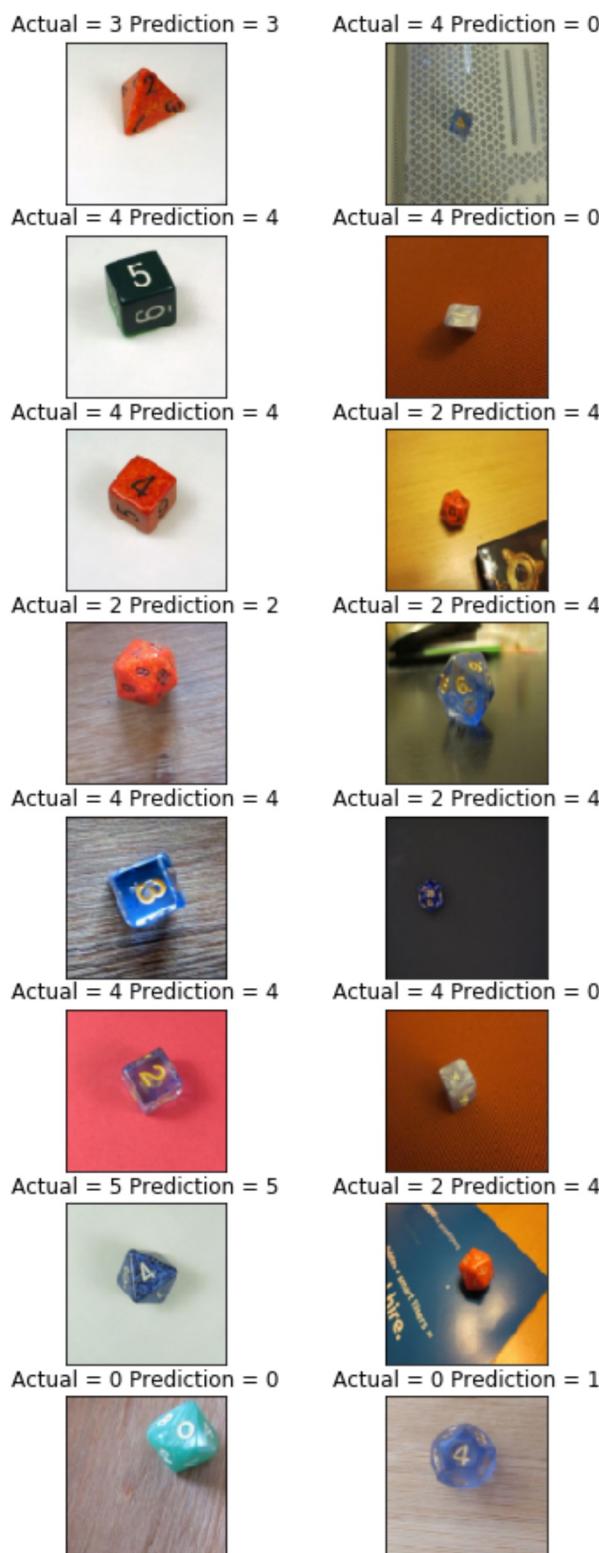
Visualize Predictions

```
In [28]: # lists to hold some correct and incorrectly classified images
correct_indices = []
incorrect_indices = []

for i in range(numerical_Y_test.shape[0]):
    # check if we have a correct match if so append to list
    if(numerical_Y_test[i] == numerical_predictions[i]):
        correct_indices.append(i)
    # stop when we reach 12 examples
    if(len(correct_indices) == 12):
        break

for i in range(numerical_Y_test.shape[0]):
    # check if we have a incorrect match if so append to list
    if(numerical_Y_test[i] != numerical_predictions[i]):
        incorrect_indices.append(i)
    # stop when we reach 12 examples
    if(len(incorrect_indices) == 12):
        break
```

```
In [29]: index = 1
plt.figure(figsize=(6,24))
for correct, incorrect in zip(correct_indices, incorrect_indices):
    plt.subplot(12, 2, index)
    plt.imshow(X_test[correct])
    plt.title("Actual = %s Prediction = %s" % (numerical_Y_test[correct], numerical_predictions[correct]))
    plt.xticks([])
    plt.yticks([])
    plt.subplot(12, 2, index+1)
    plt.imshow(X_test[incorrect])
    plt.title("Actual = %s Prediction = %s" % (numerical_Y_test[incorrect], numerical_predictions[incorrect]))
    plt.xticks([])
    plt.yticks([])
    index = index + 2
```



Visual Activation Maps

```
In [30]: 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("Datasets/val/d4/13136.jpg", target_size=(100,
100))
    img_tensor = image.img_to_array(img)
    img_tensor = datagenerator.apply_transform(img_tensor, datagenerator
r.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("Datasets/val/d6/13123.jpg", target_size=(100,
100))
    img_tensor = image.img_to_array(img)
    img_tensor = datagenerator.apply_transform(img_tensor, datagenerator
r.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("Datasets/val/d8/13541.jpg", target_size=(100,
100))
    img_tensor = image.img_to_array(img)
    img_tensor = datagenerator.apply_transform(img_tensor, datagenerator
r.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("Datasets/val/d12/15317.jpg", target_size=(10
0,100))
    img_tensor = image.img_to_array(img)
    img_tensor = datagenerator.apply_transform(img_tensor, datagenerator
r.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("Datasets/val/d17/220/14000.jpg", target_size=(10
0,100))
    img_tensor = image.img_to_array(img)
    img_tensor = datagenerator.apply_transform(img_tensor, datagenerator
r.get_random_transform(img_tensor.shape))
    img_tensor = np.expand_dims(img_tensor, axis=0)
    img_tensor/=255
    dice_images.append(img_tensor)
```

```
In [31]: 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 [32]: 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

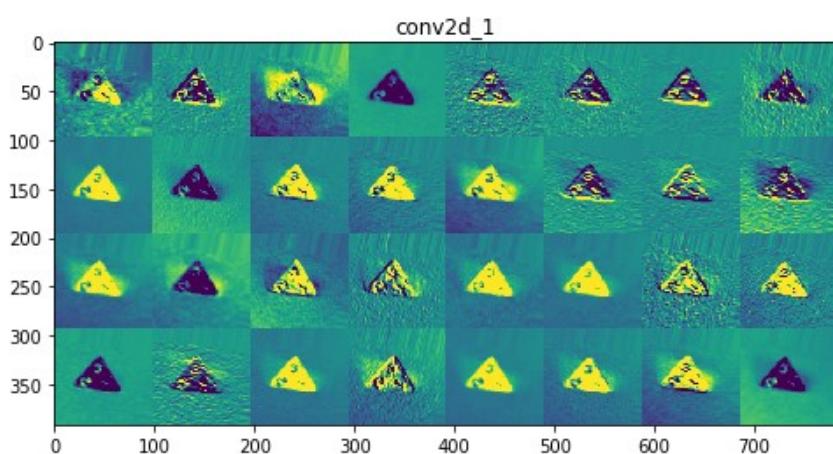
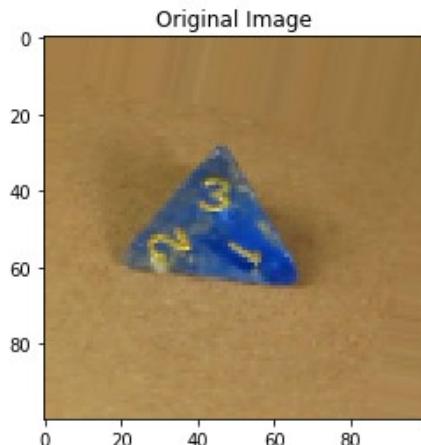
    for layer_name, layer_activation in zip(layer_names, activations):
        # skip none convolutional layers
        if(layer_name.find('conv') == -1):
            continue
        n_features = layer_activation.shape[-1]

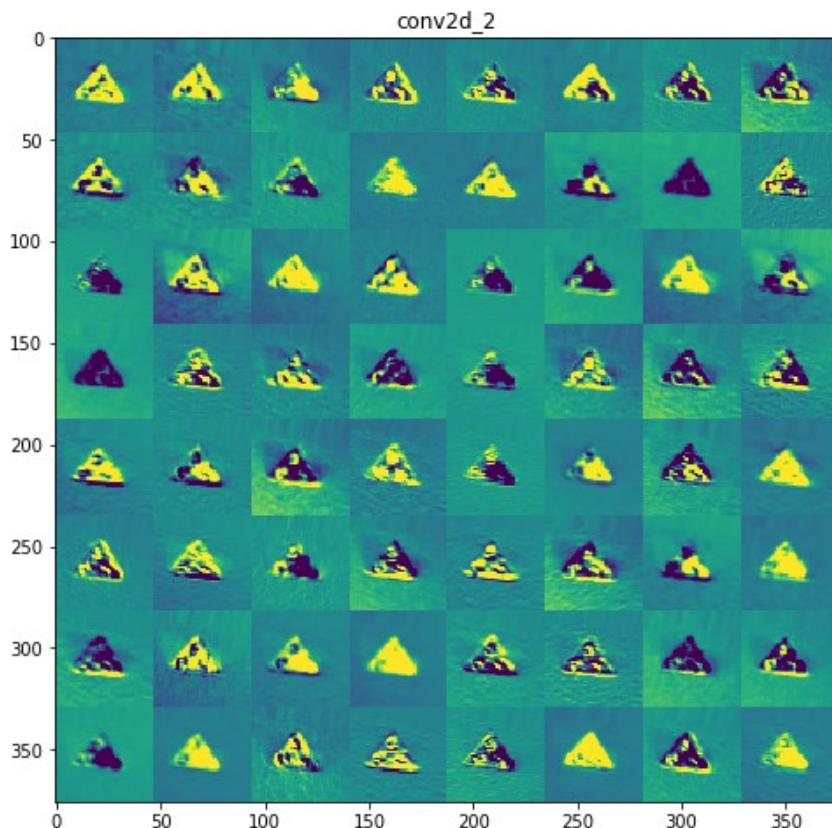
        size = layer_activation.shape[1]

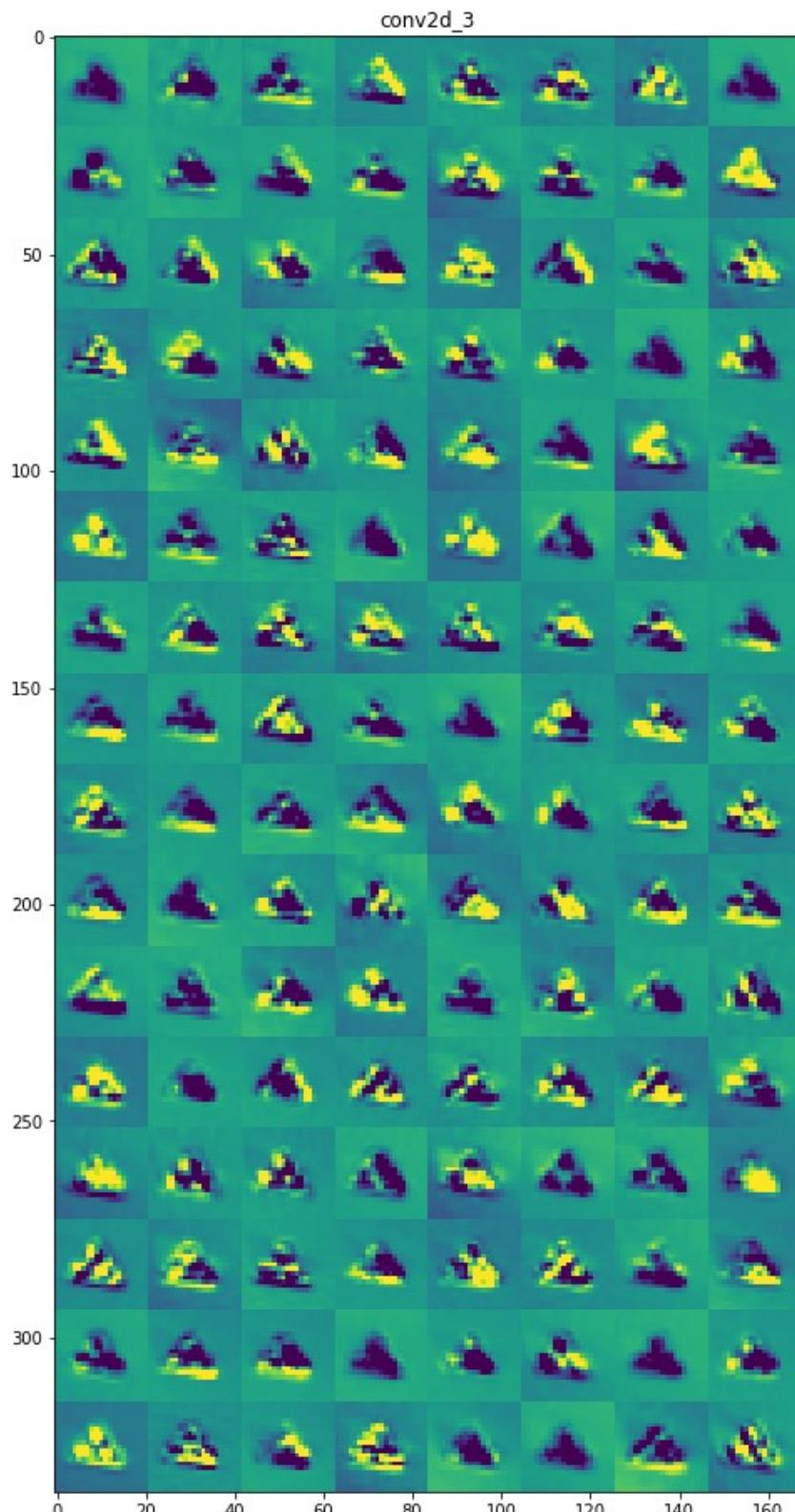
        n_cols = n_features // images_per_row
        display_grid = np.zeros((size * n_cols, images_per_row * size))

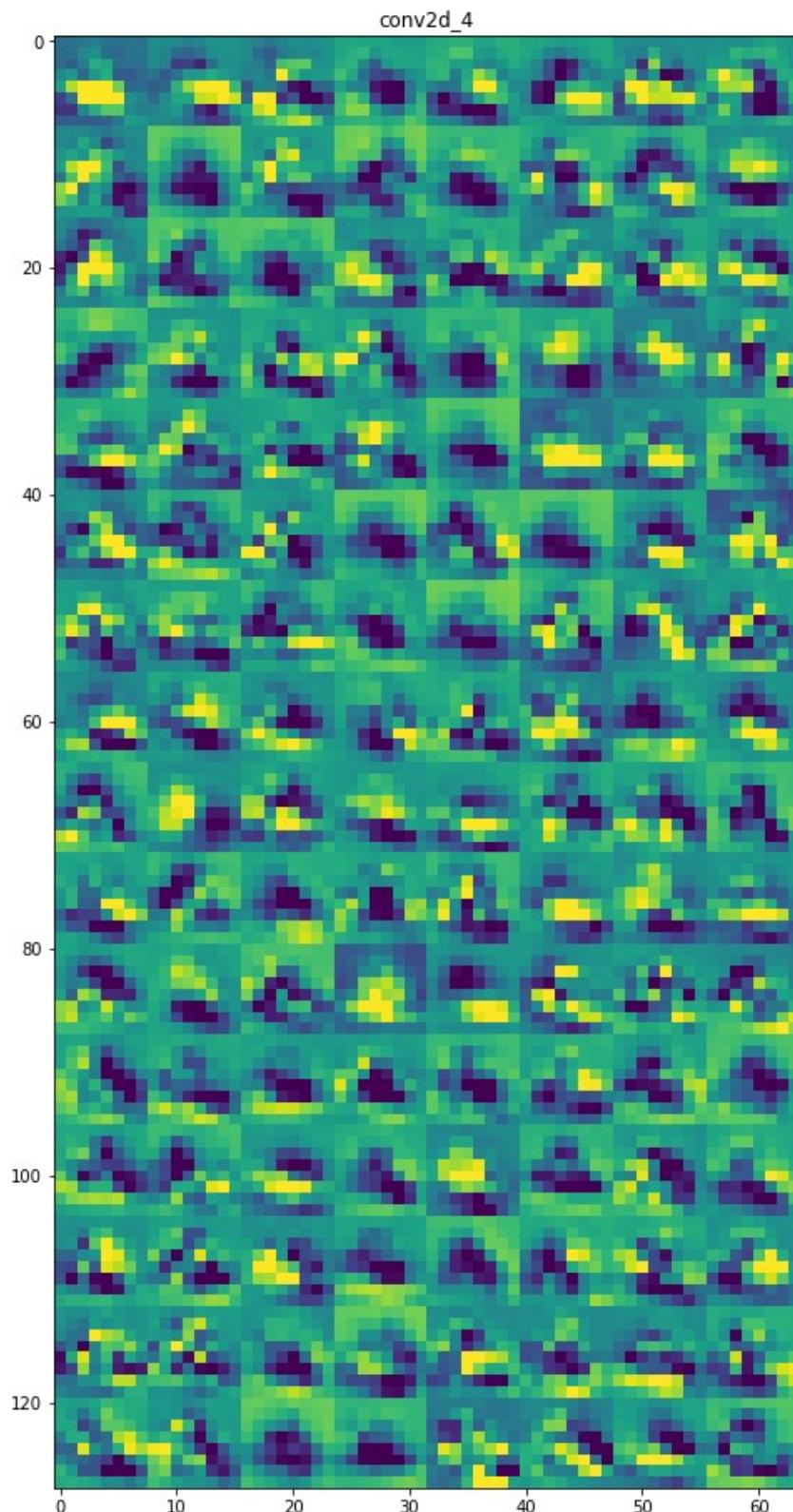
        for col in range(n_cols):
            for row in range(images_per_row):
                channel_image = layer_activation[0, :, :, col * images_per_row + row]
                channel_image -= channel_image.mean()
                channel_image /= channel_image.std()
                channel_image *= 64
                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

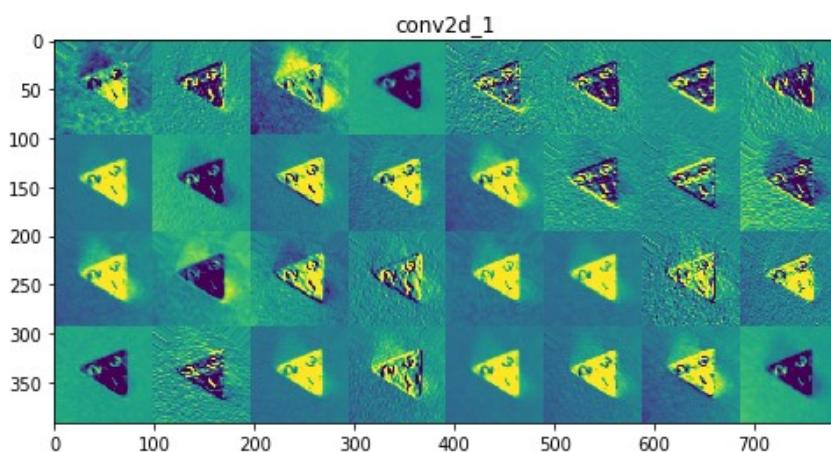
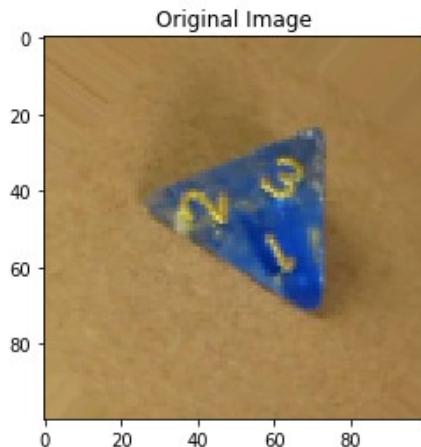
        scale = 1. / size
        plt.figure(figsize=(scale * display_grid.shape[1], scale * display_grid.shape[0]), facecolor='w')
        plt.title(layer_name)
        plt.grid(False)
        plt.imshow(display_grid, aspect='auto', cmap='viridis')
        plt.show()
```

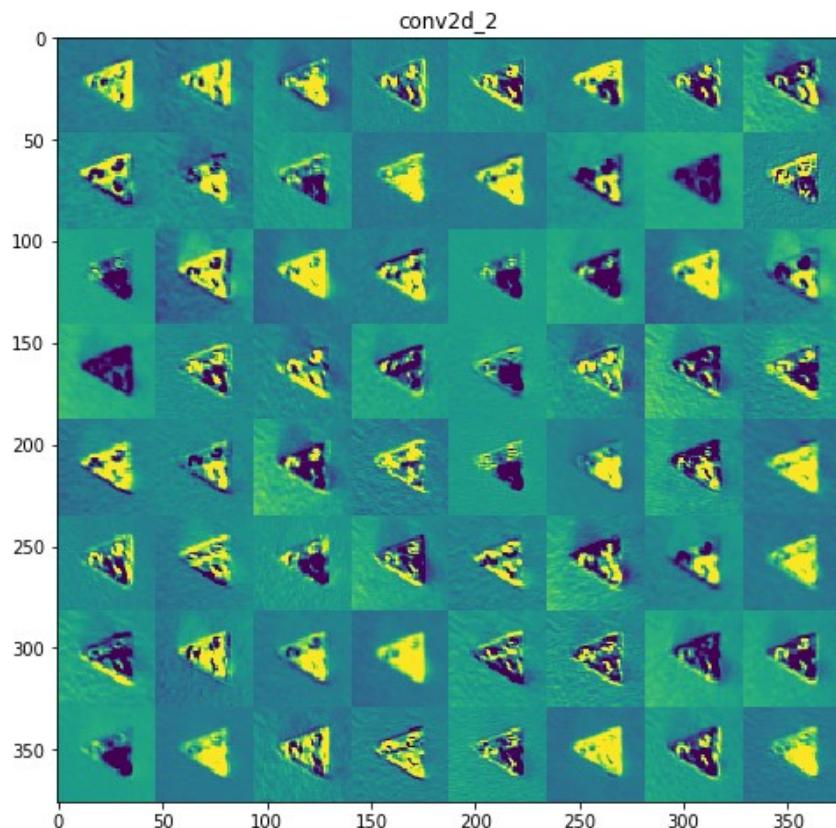


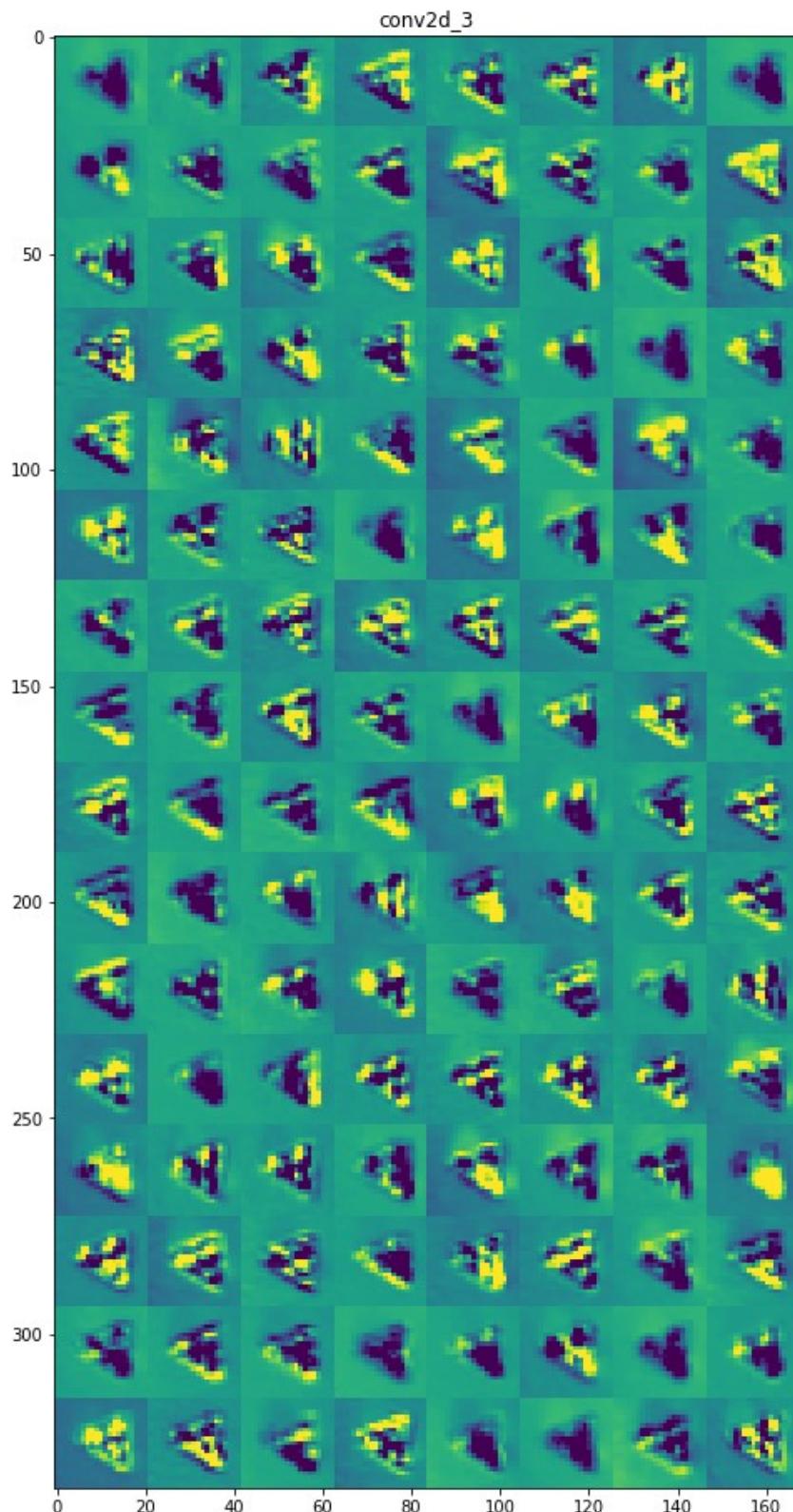


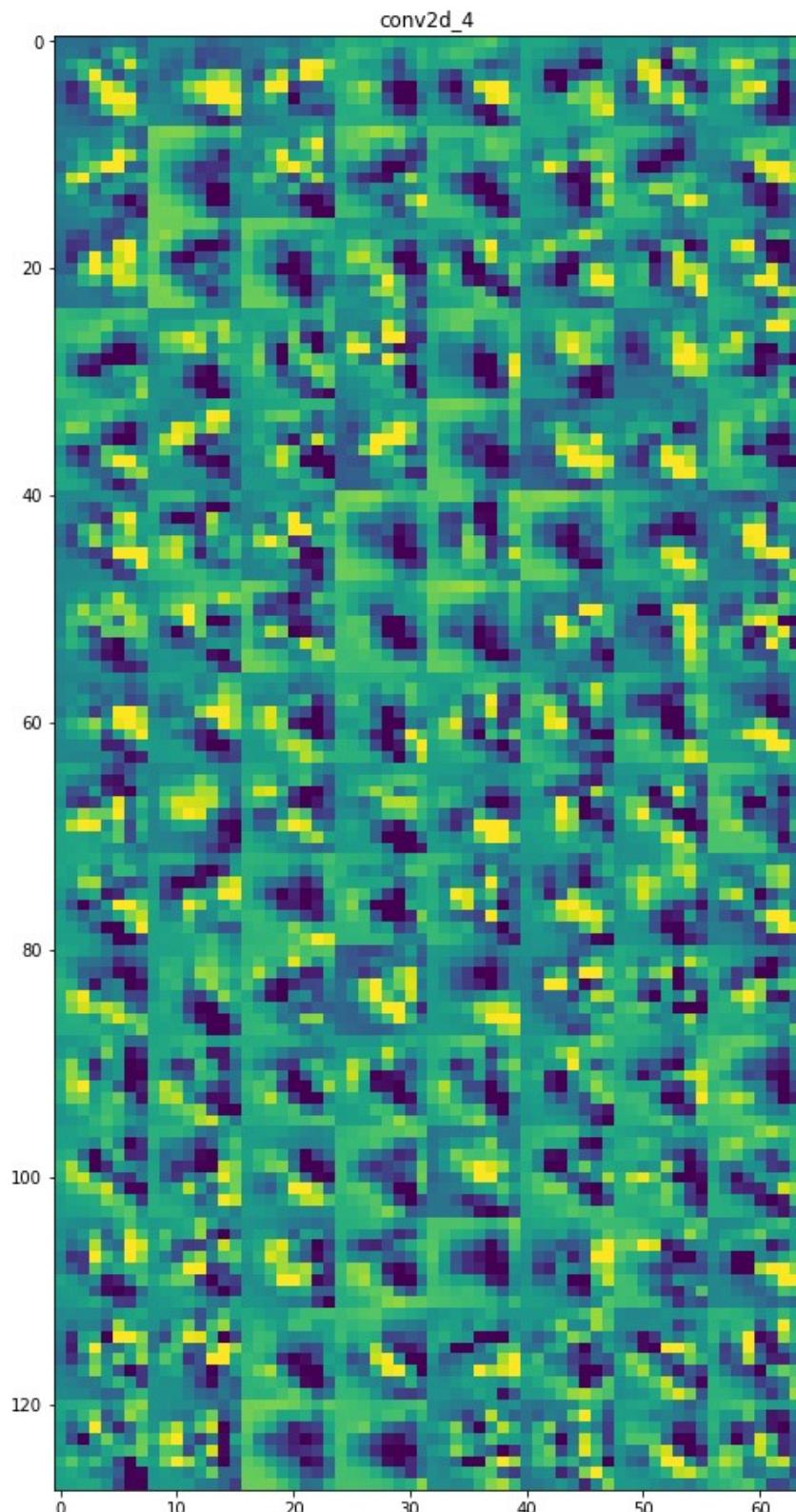


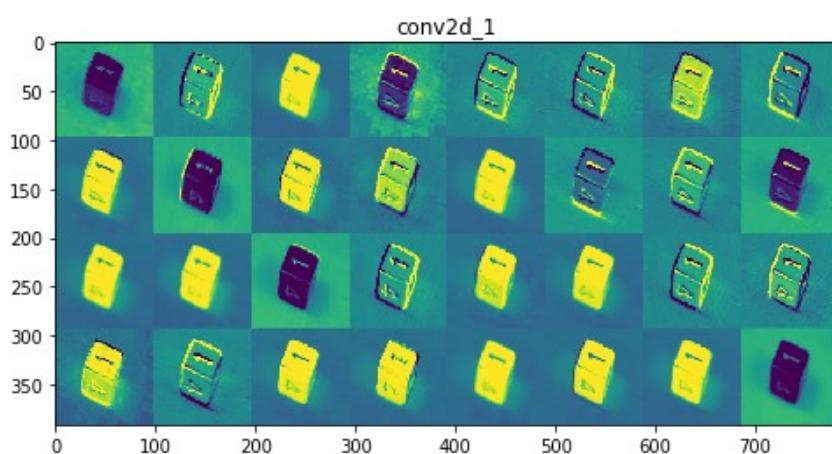
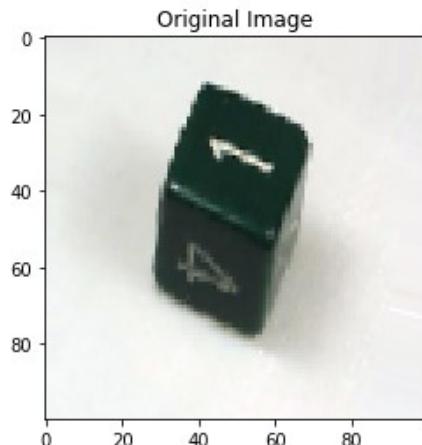


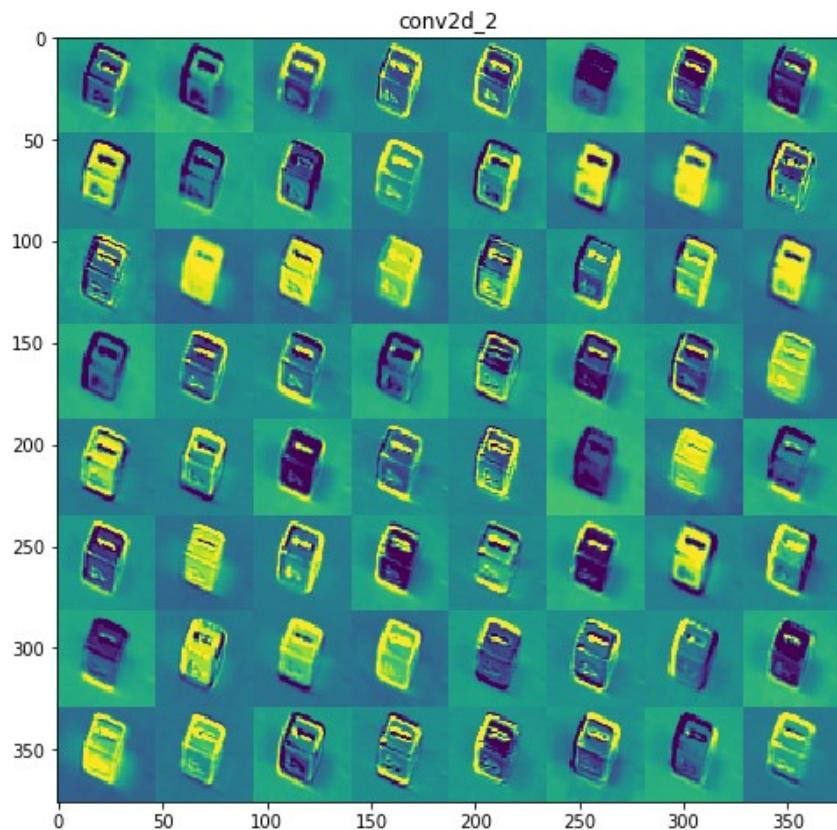


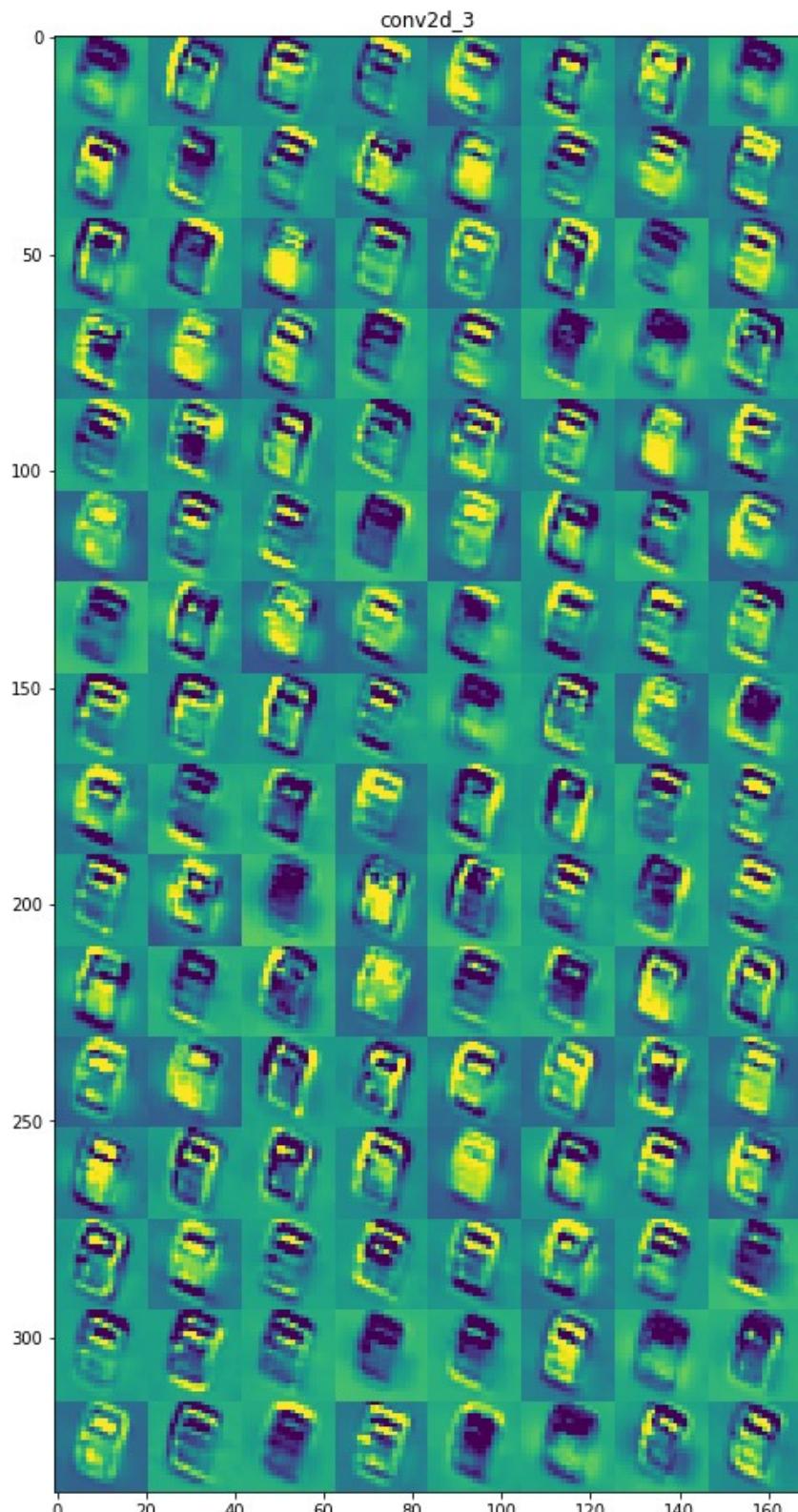


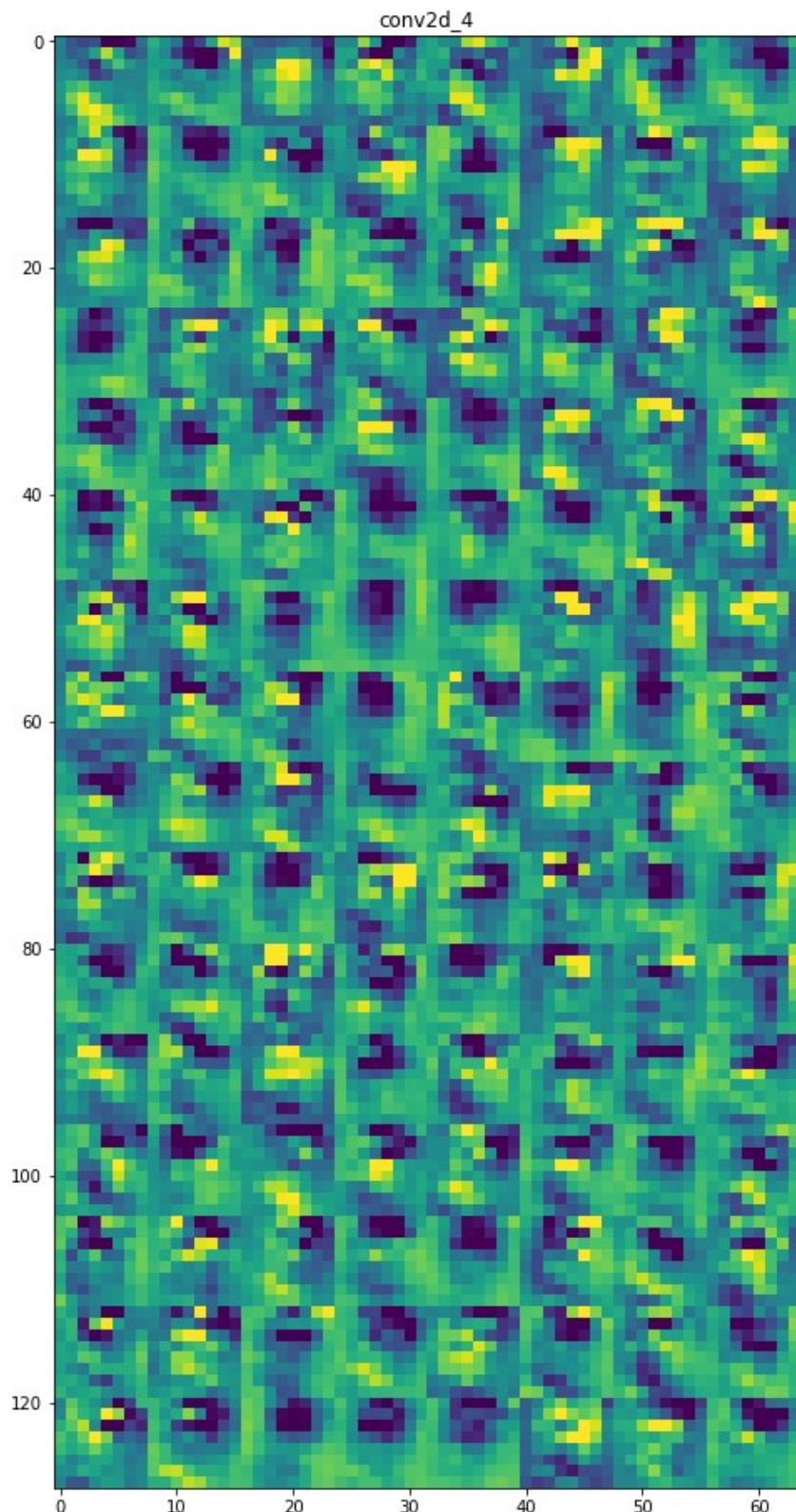


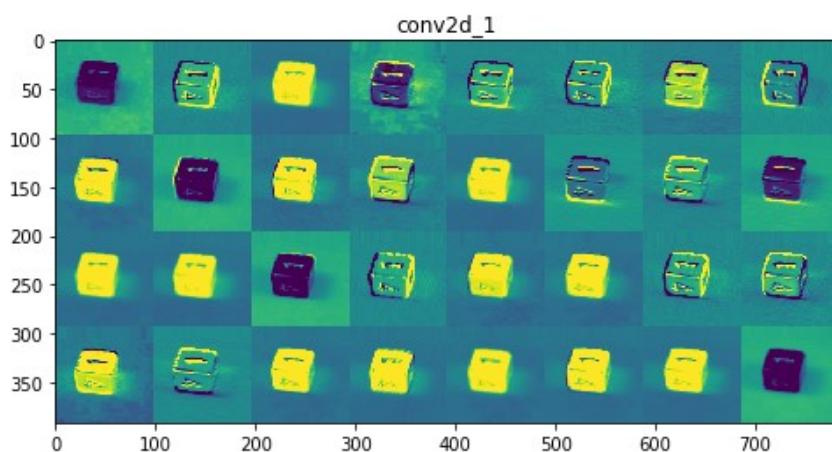
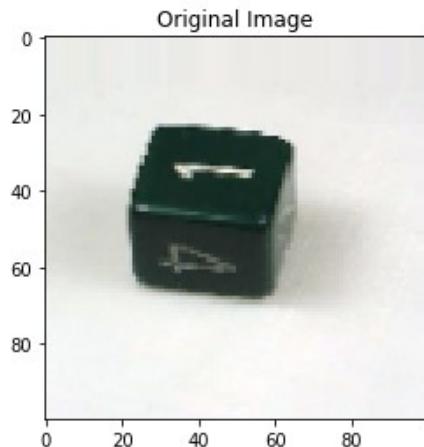


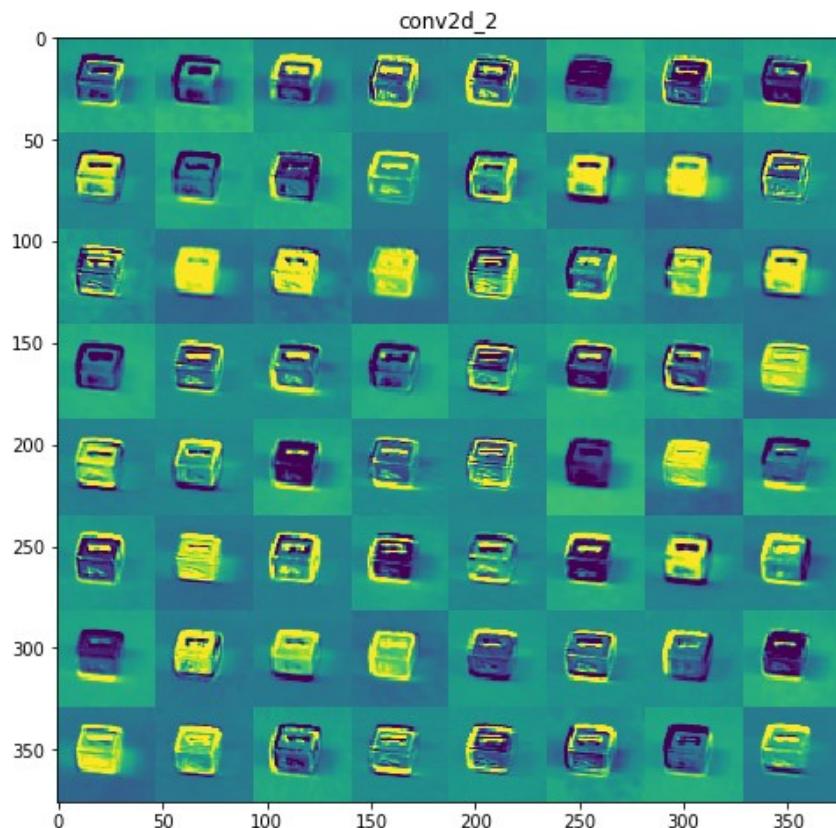


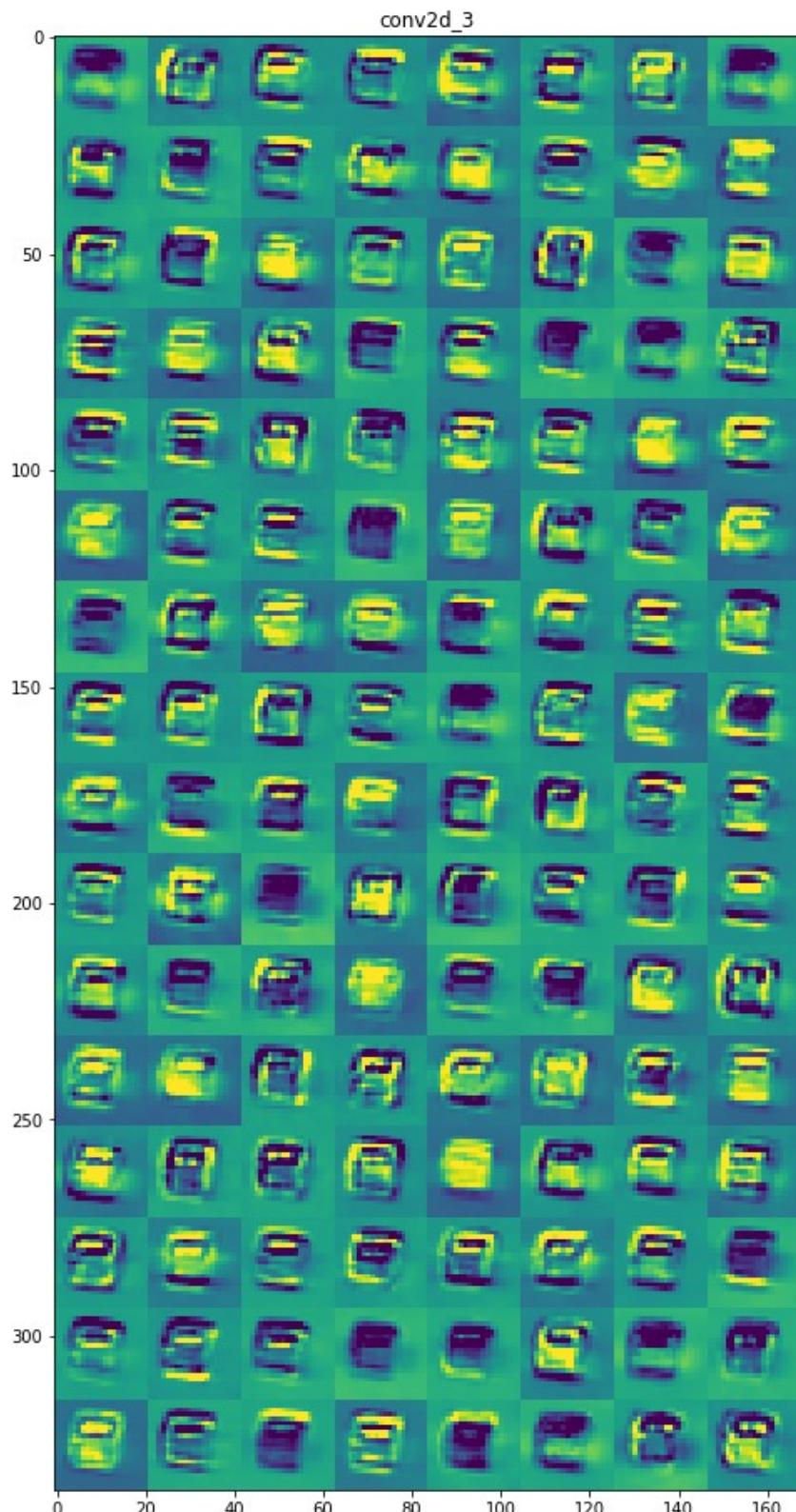


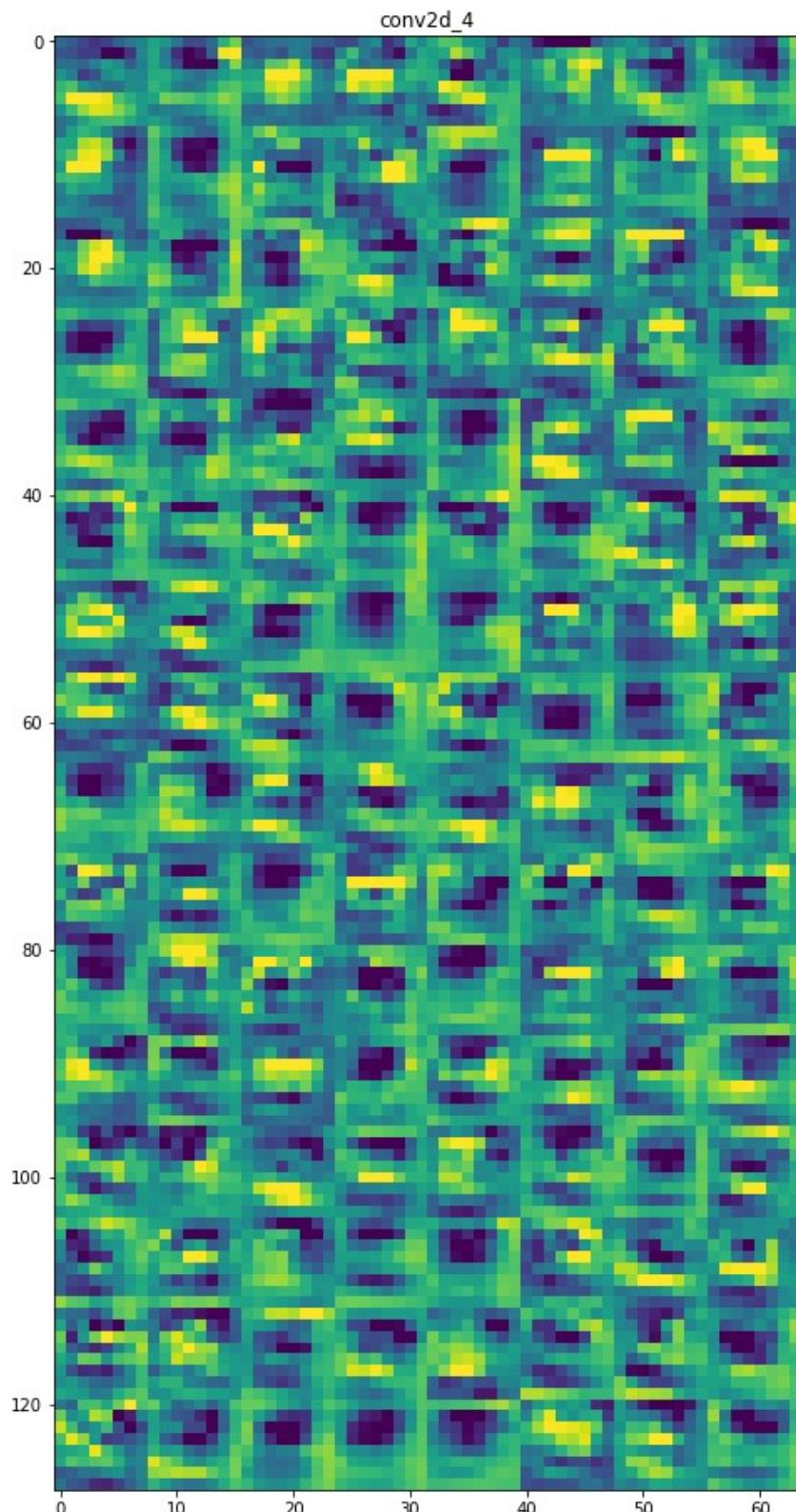


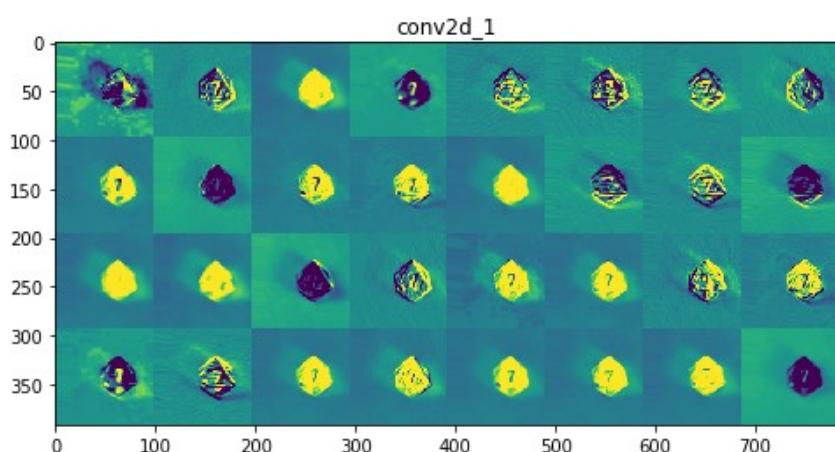
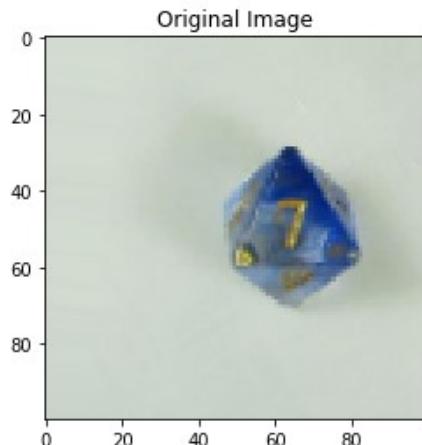


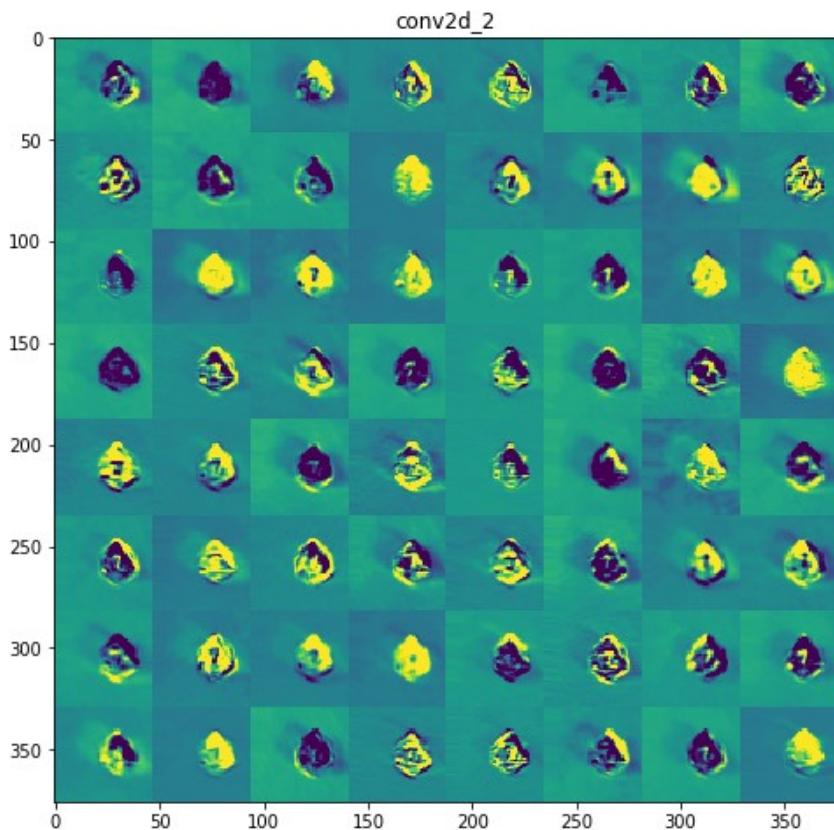


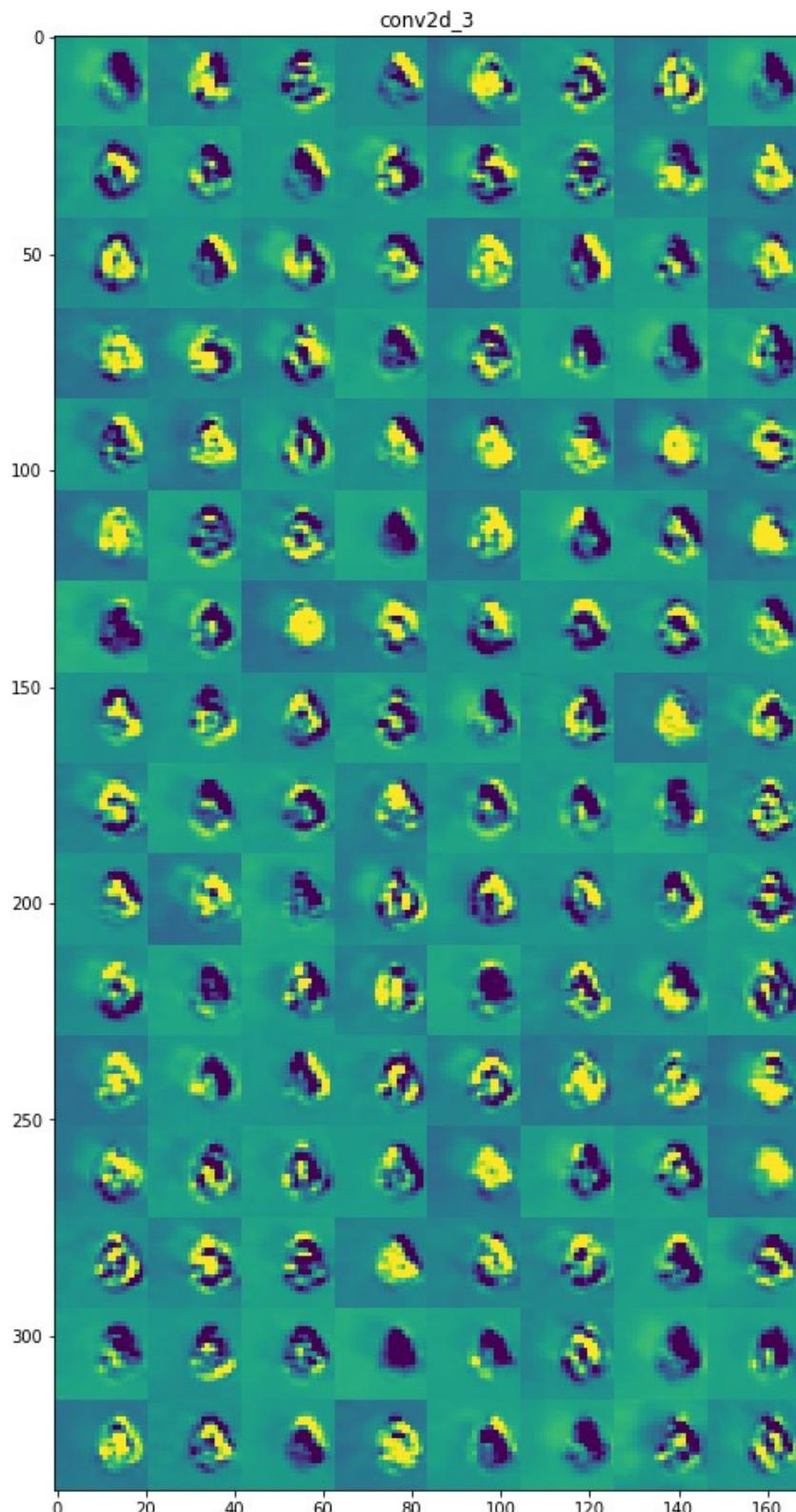


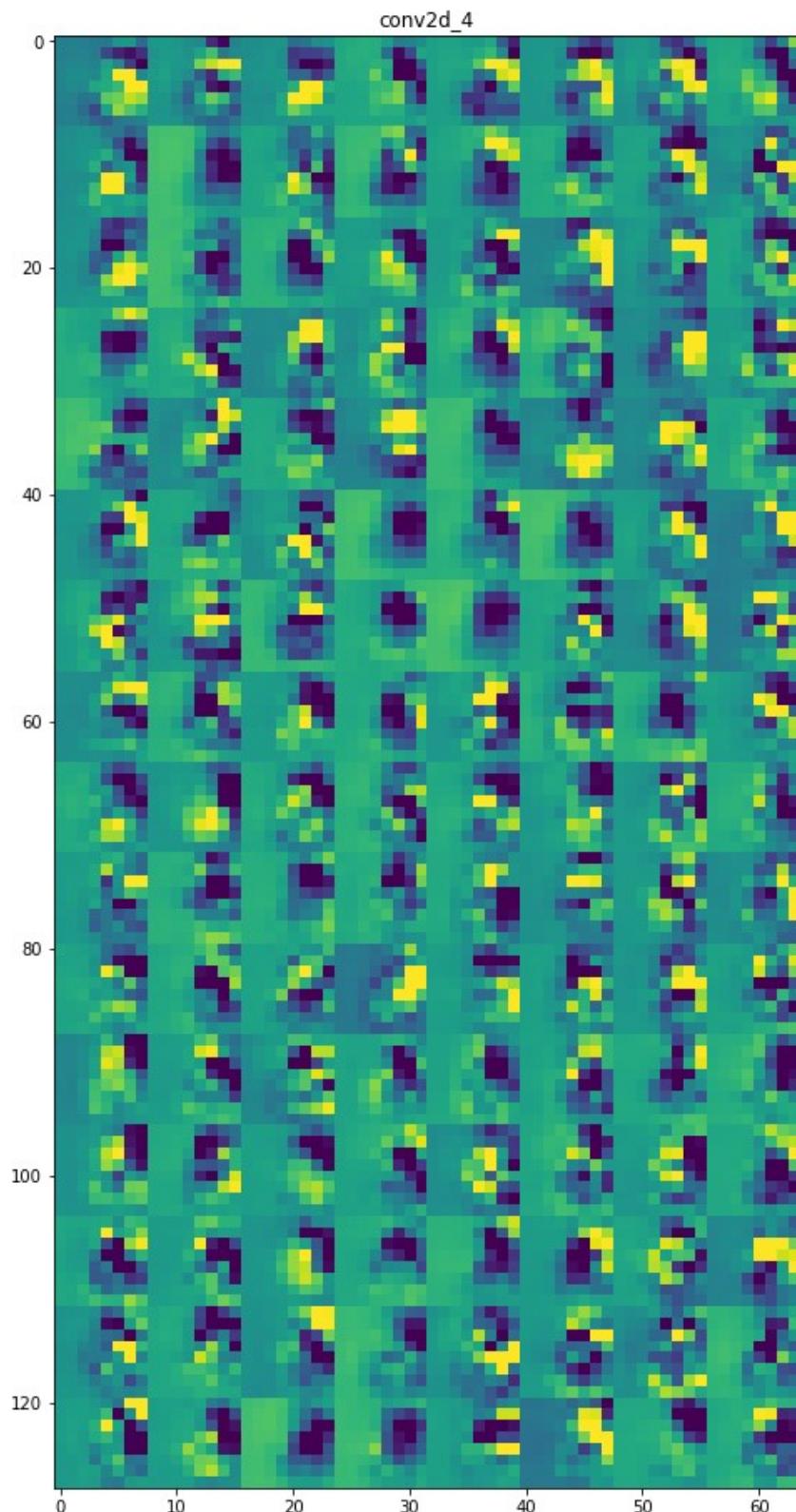


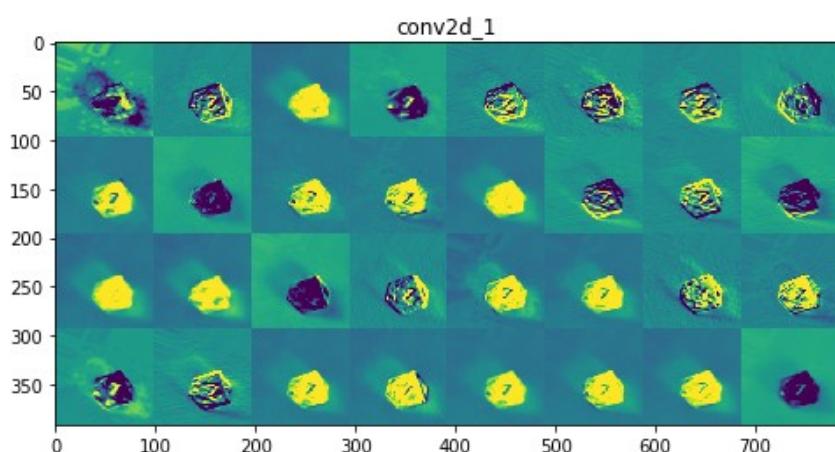
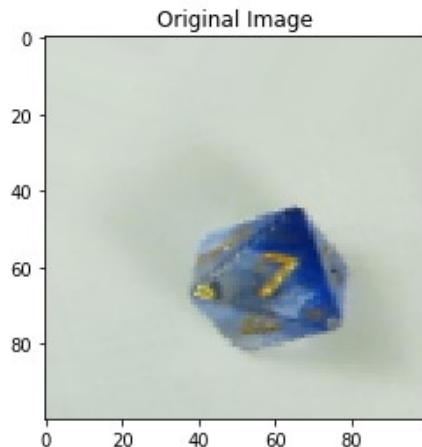


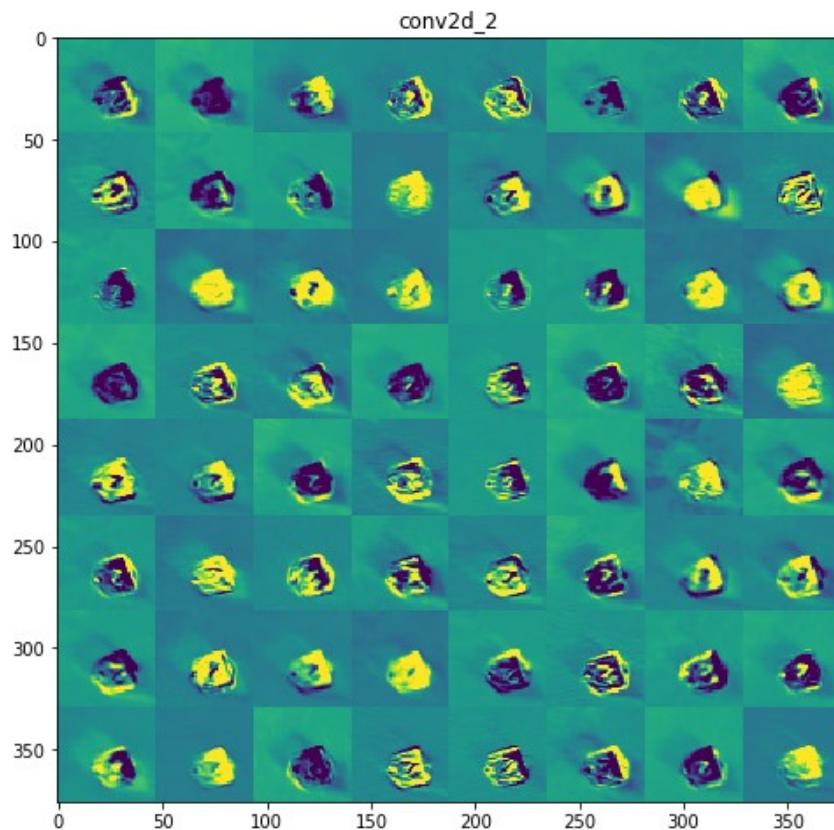


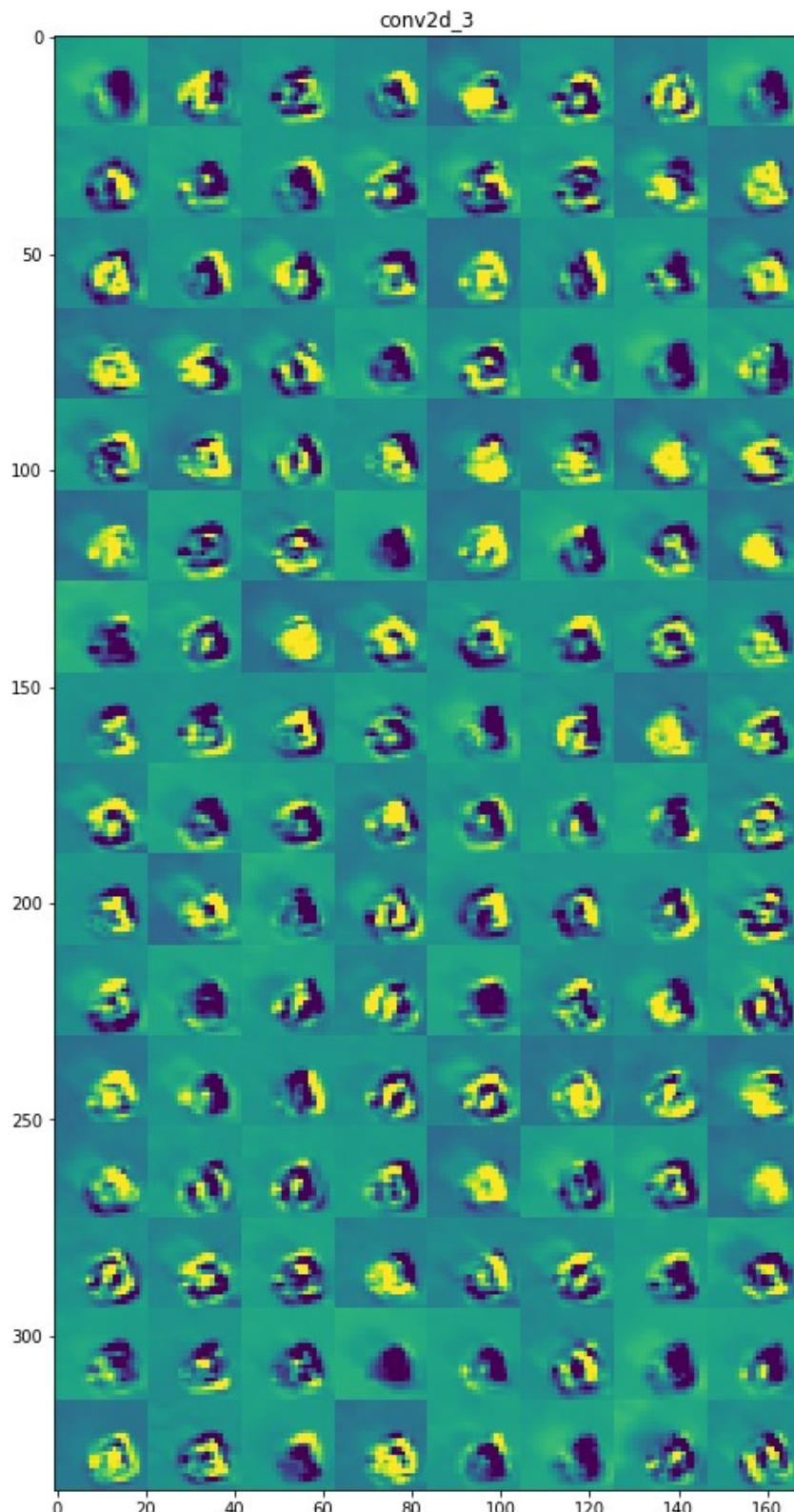


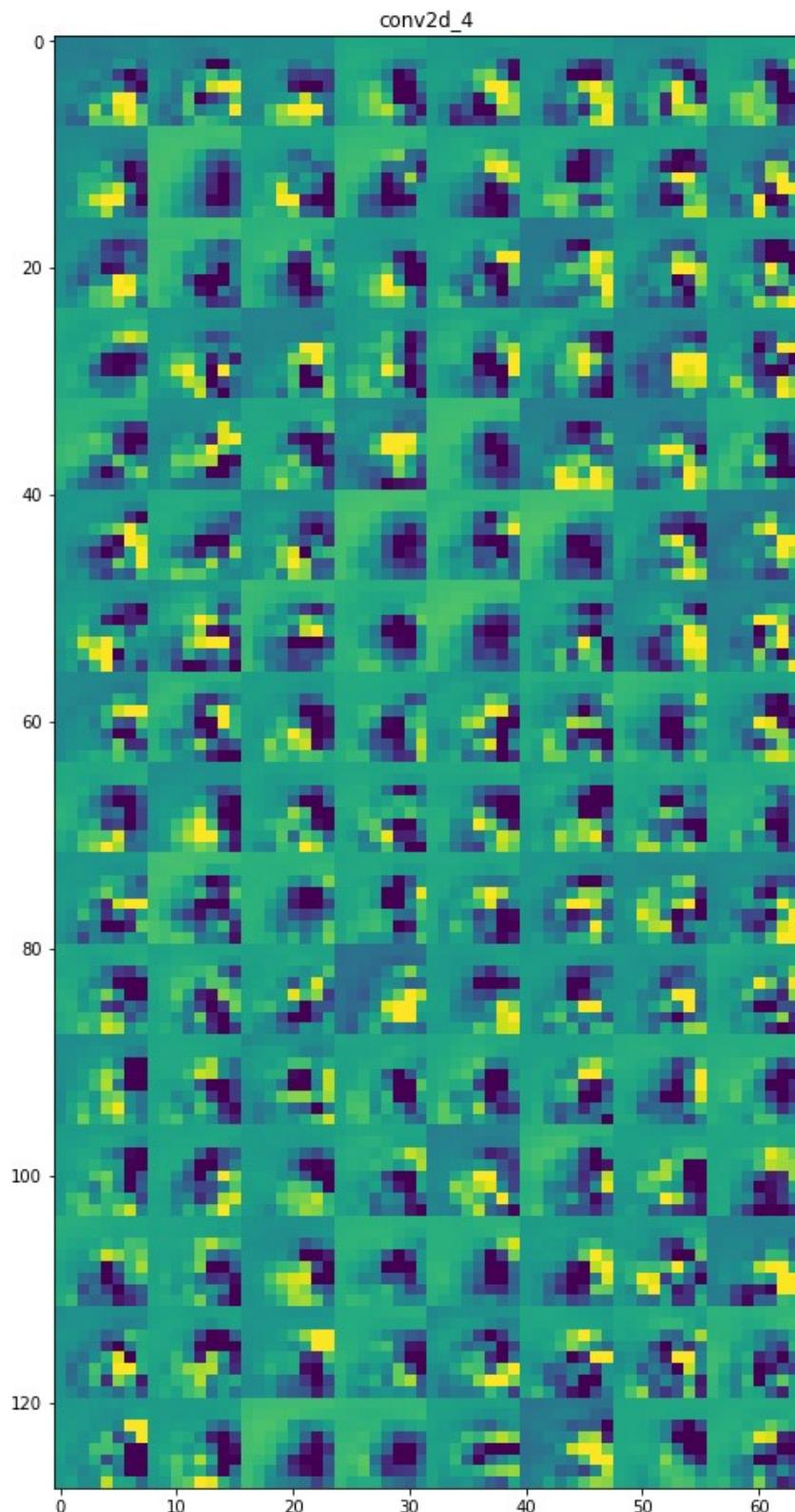


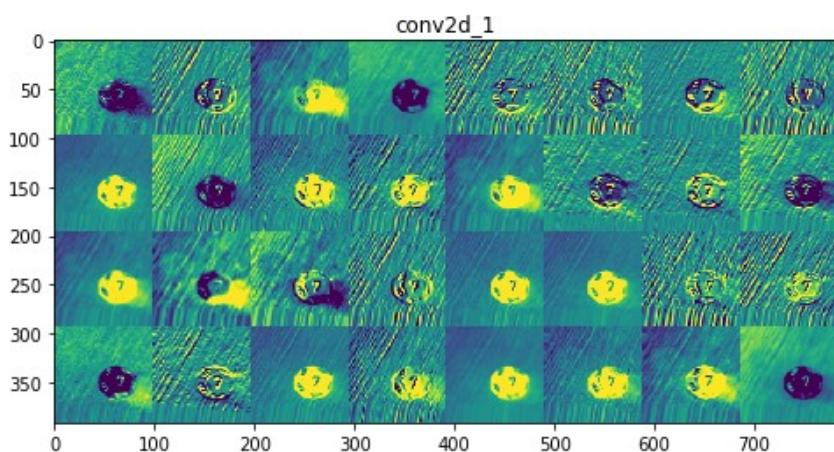
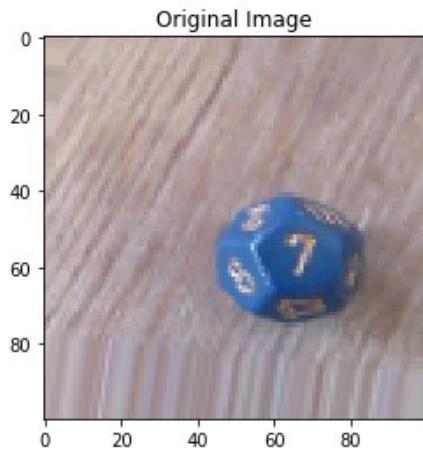


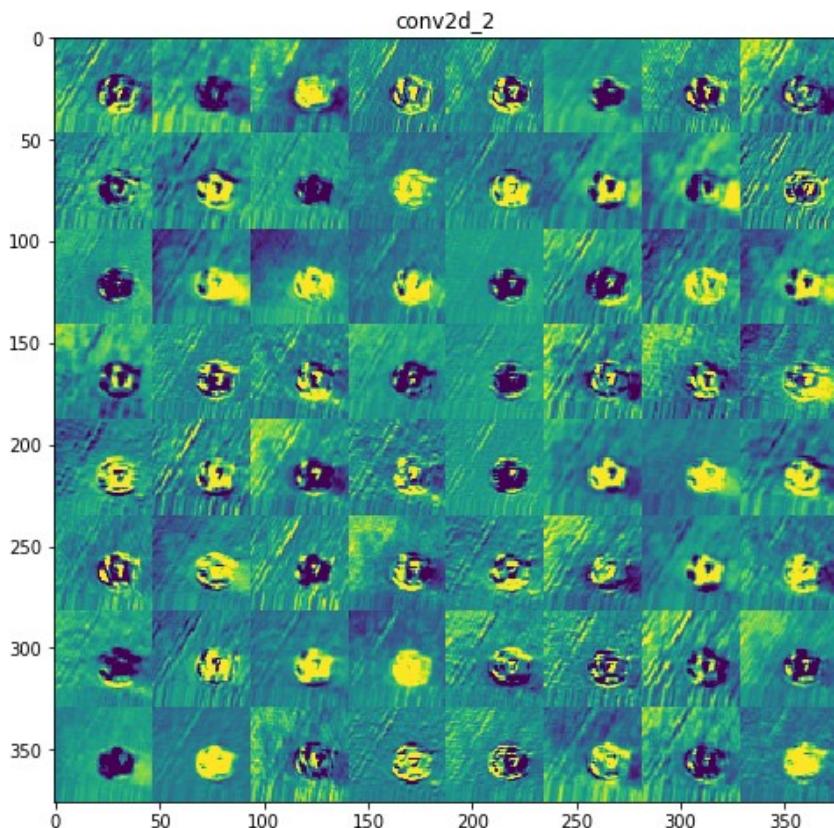


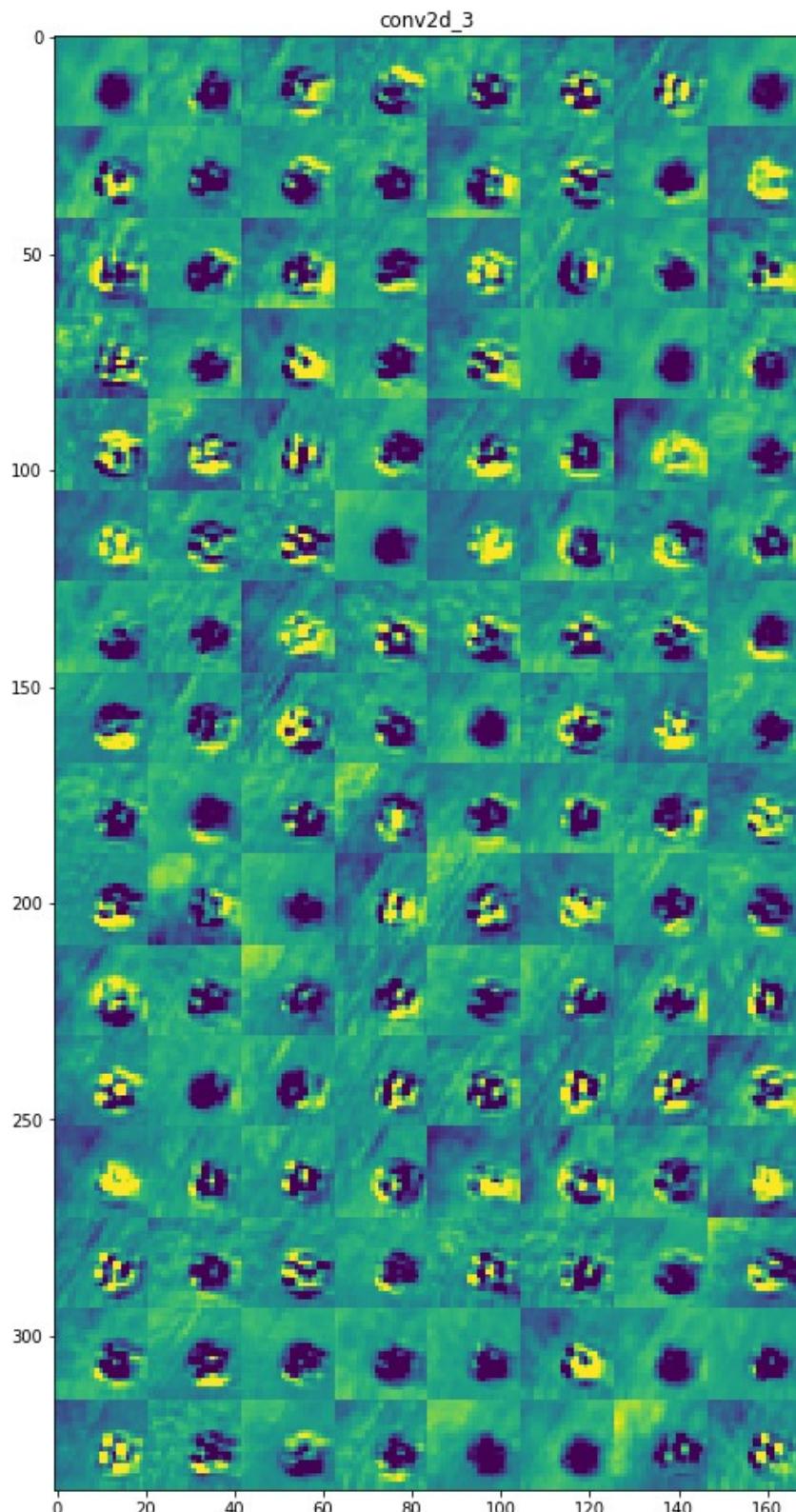


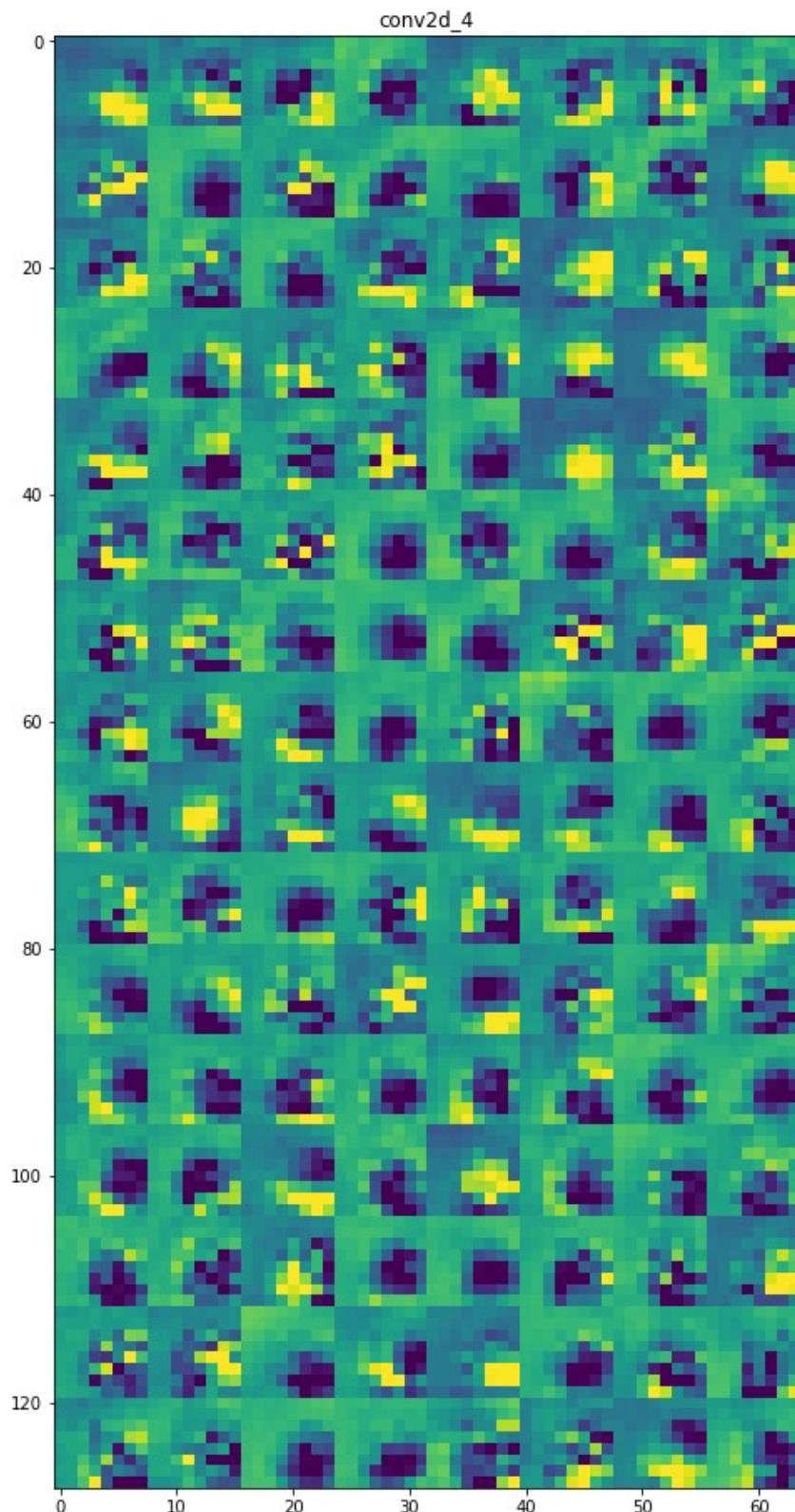


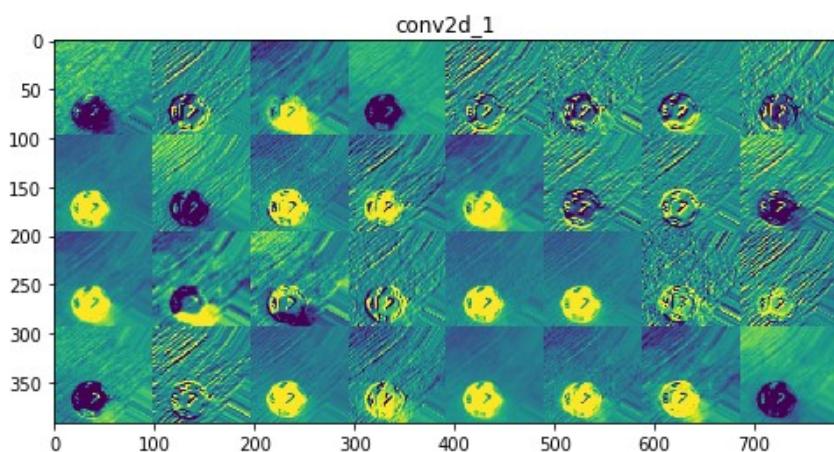
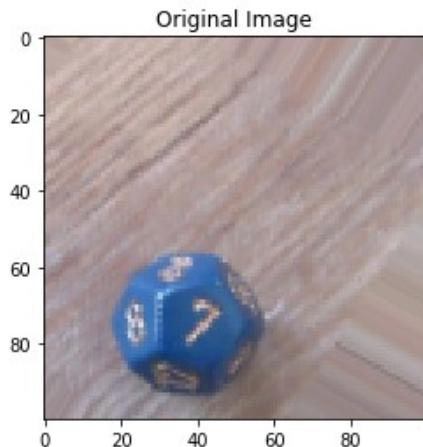


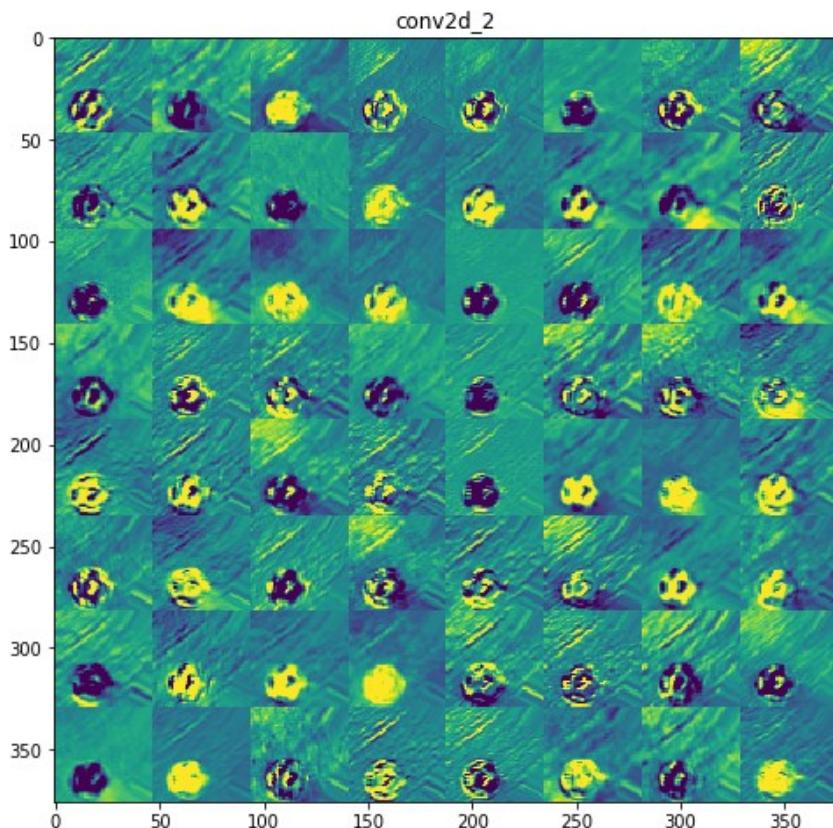


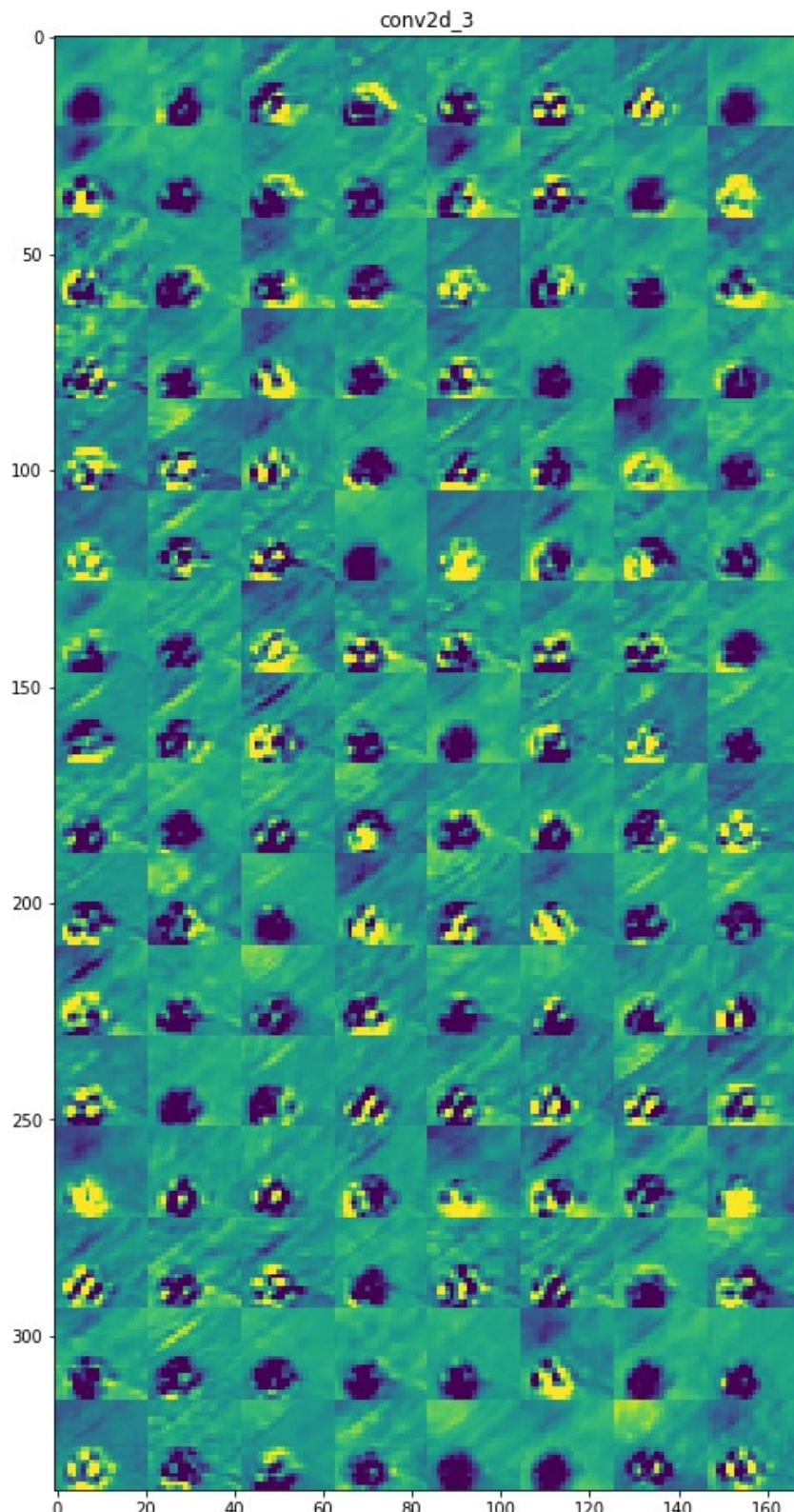


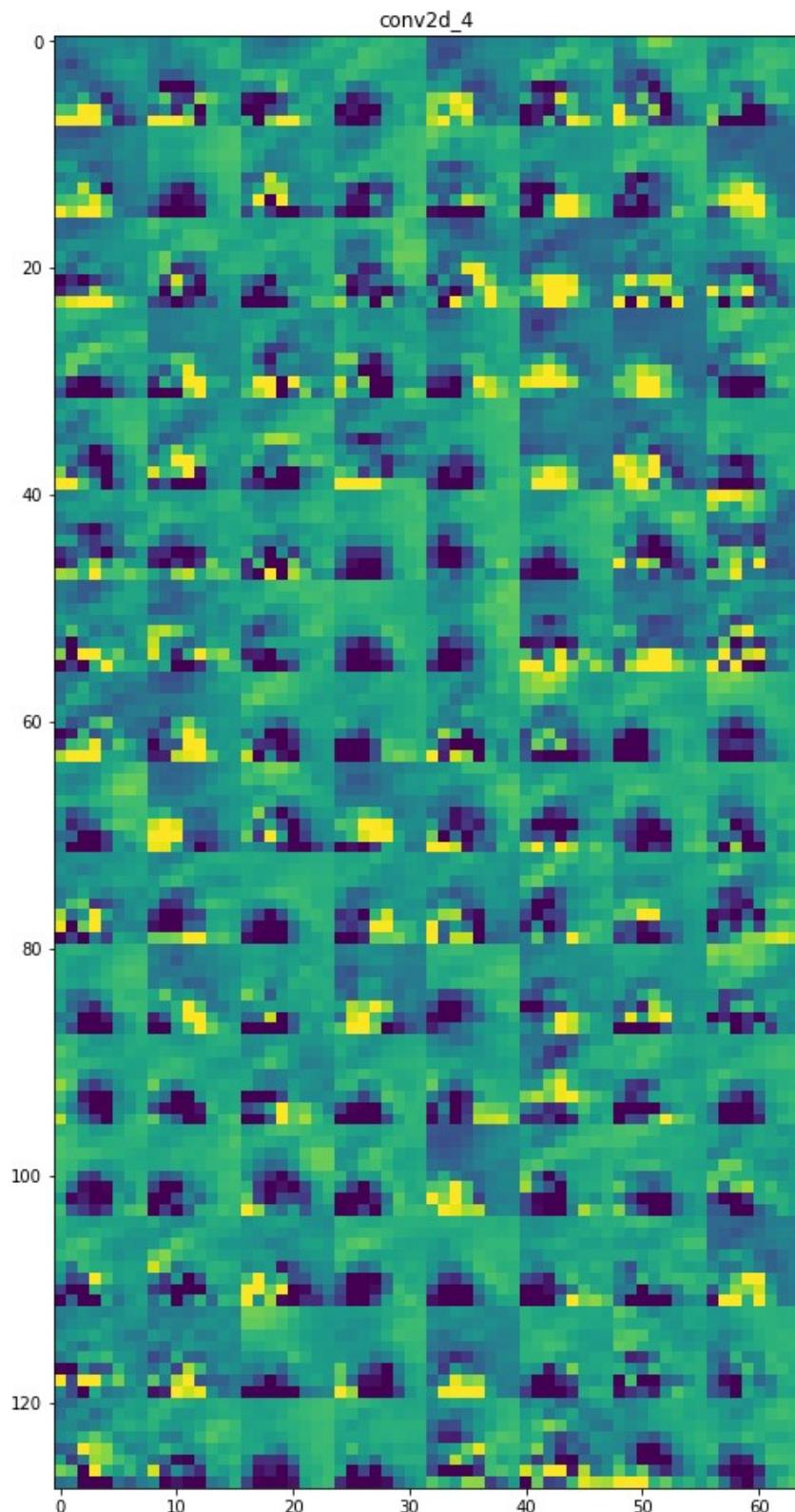


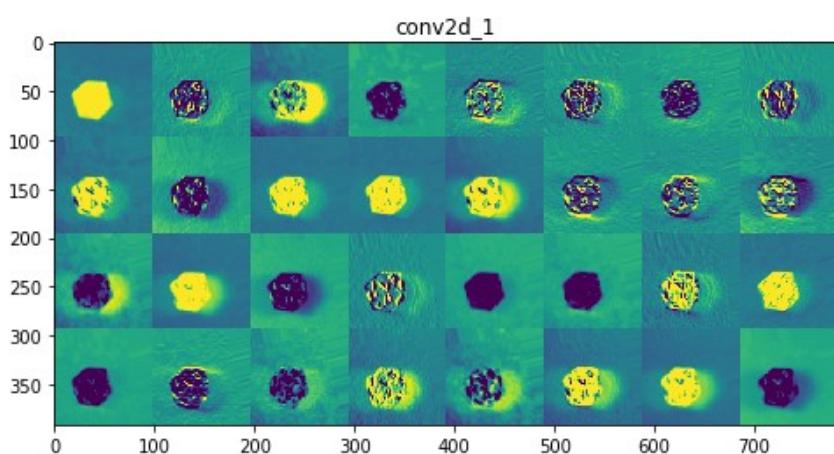
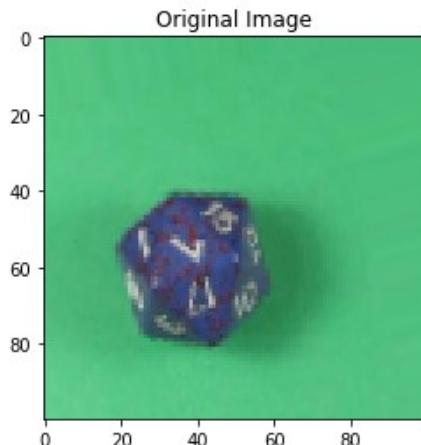


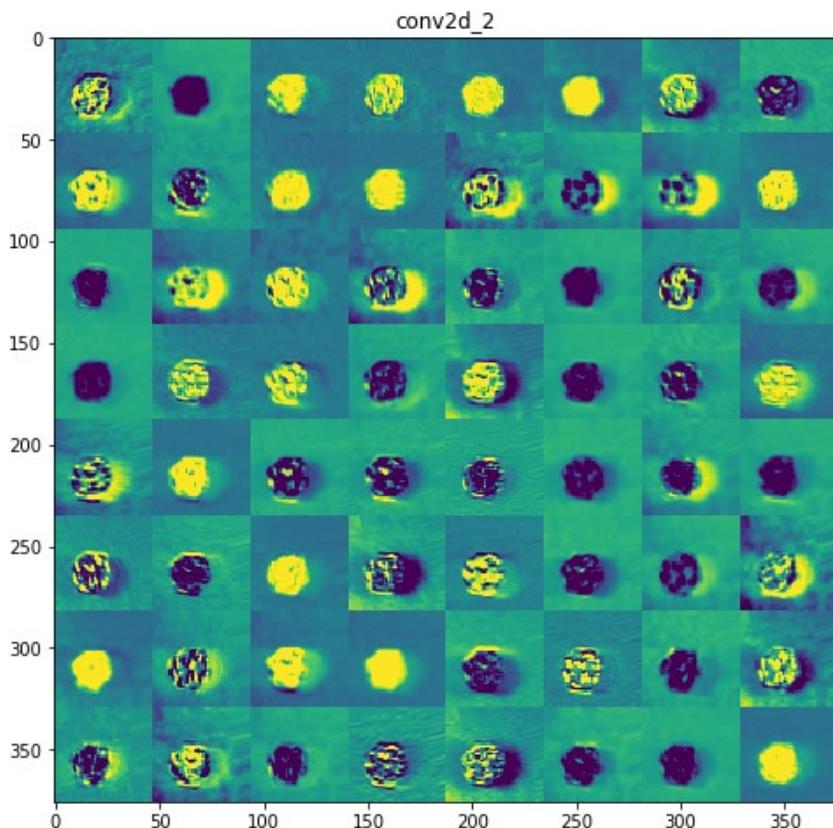


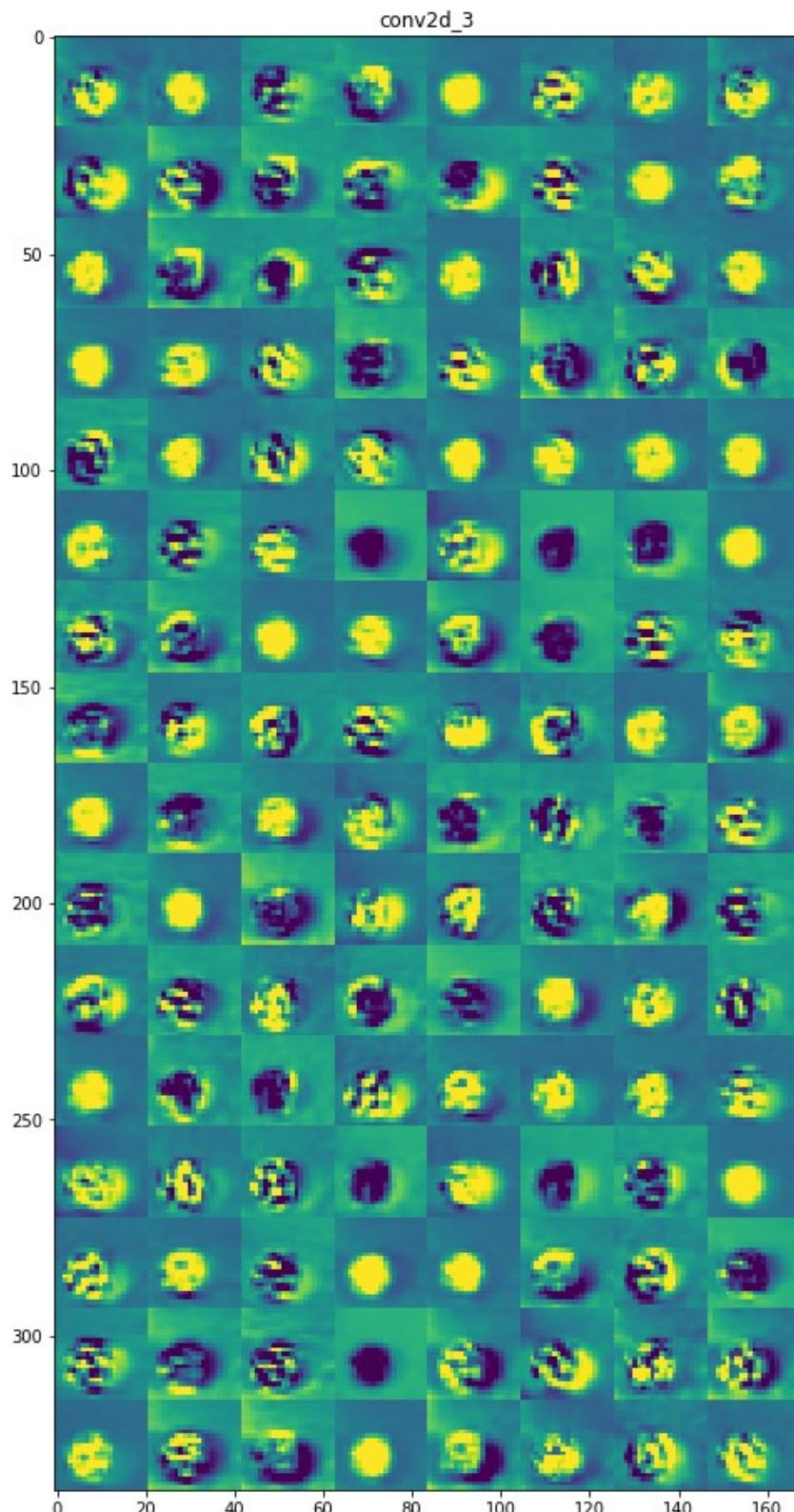


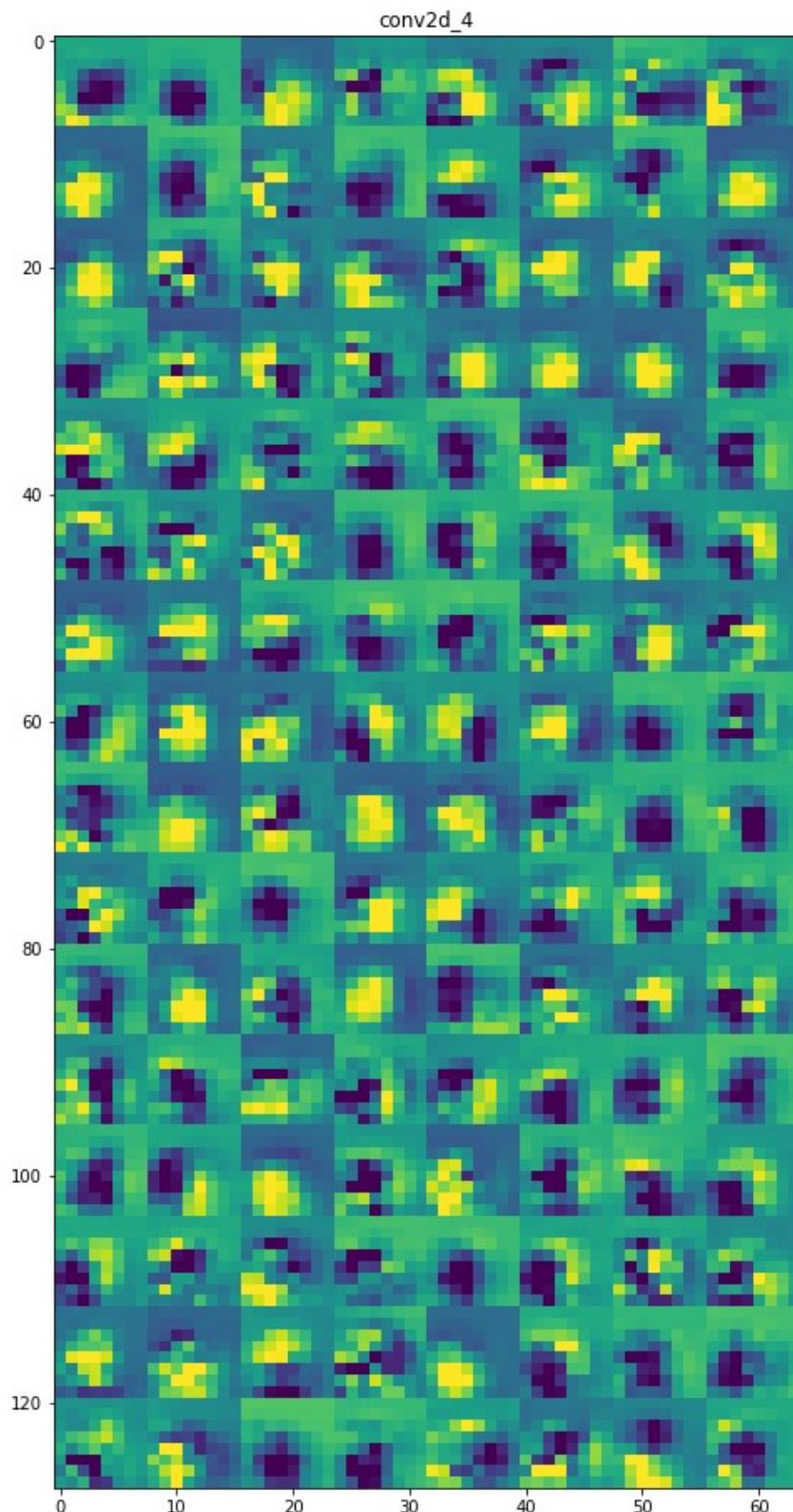


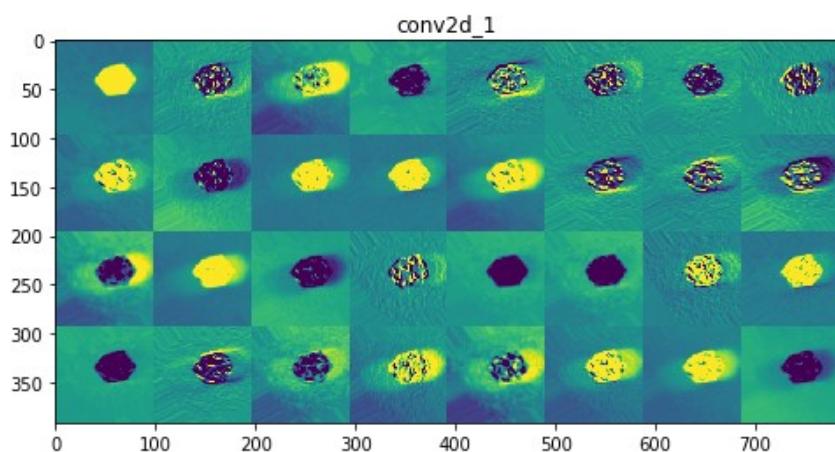
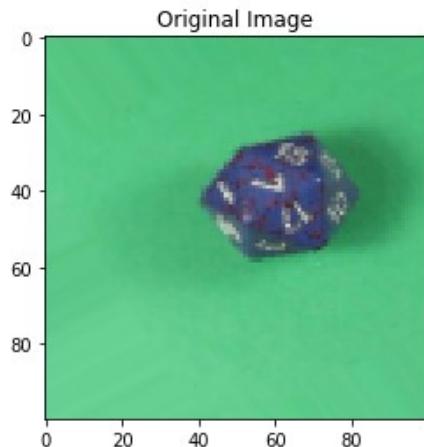


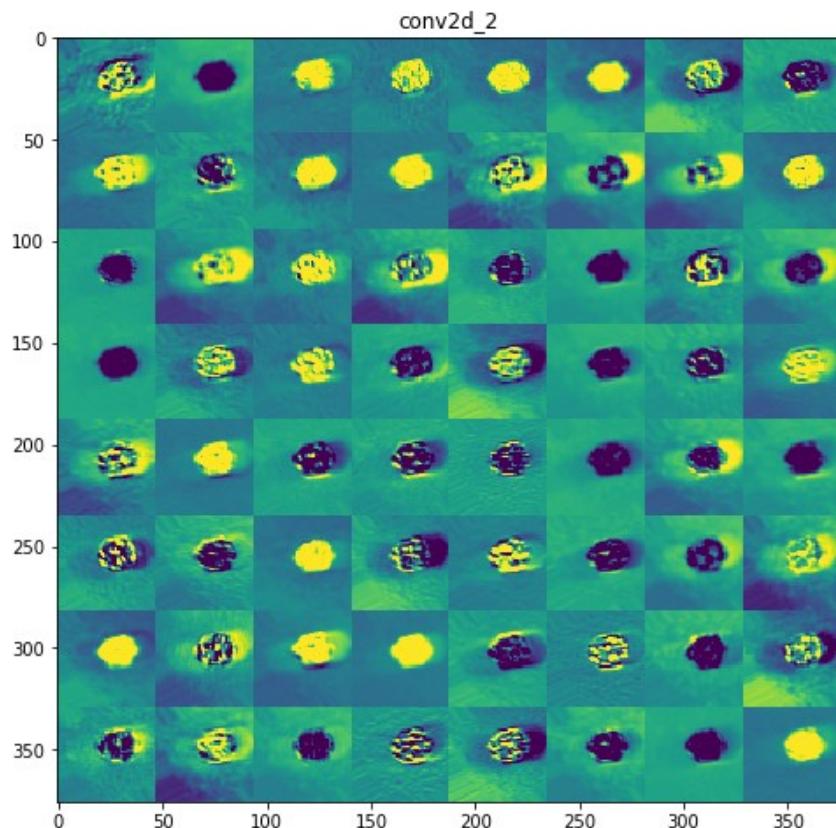


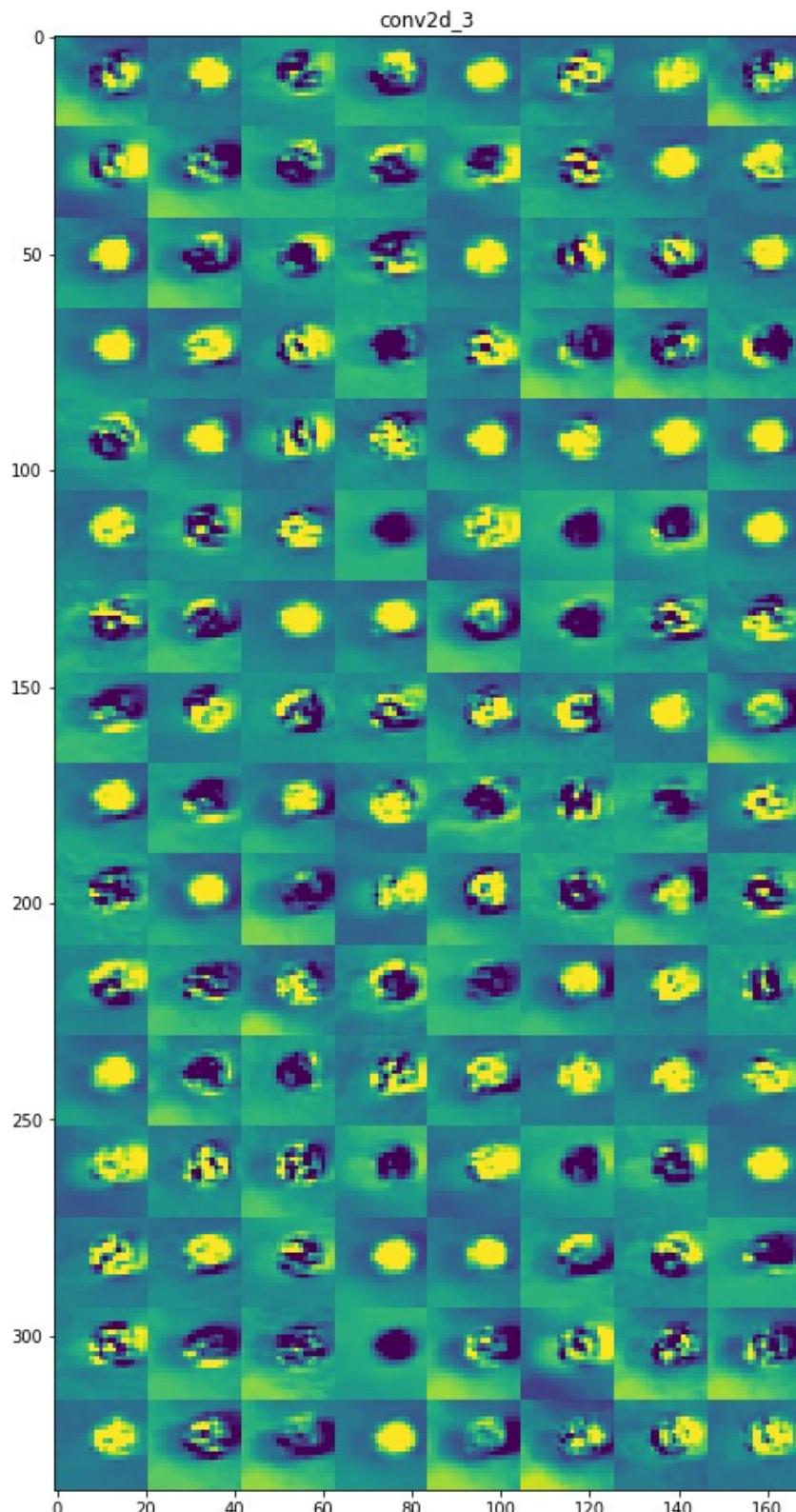


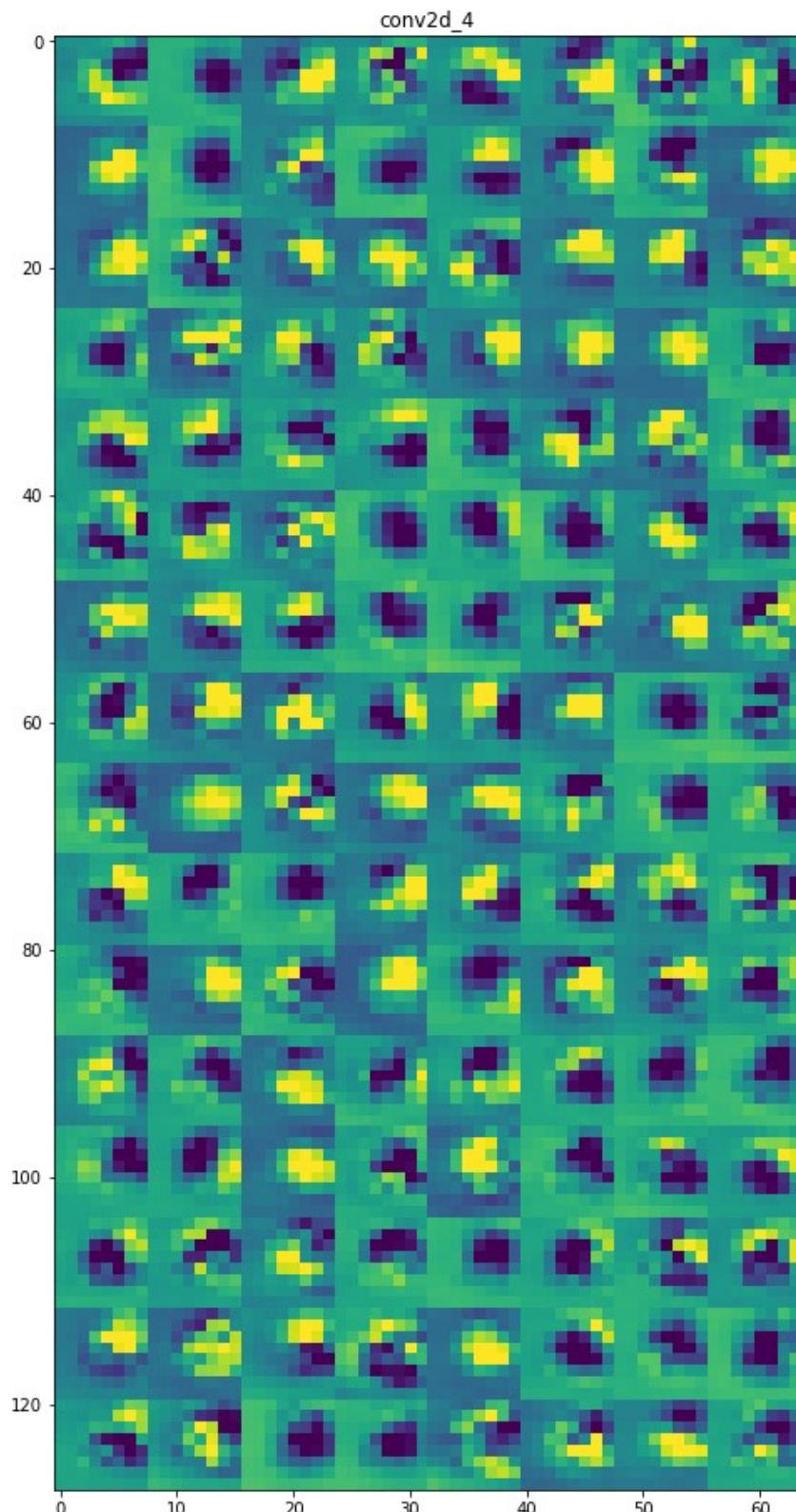












Visualizing the activation maps when the CNN makes a prediction is a good way to visualize the type of data that is being collected and how it is abstracted by later layers. For instance when we look at the first layer conv2d_1 seems to collect some fur texture, silhouette, contours, and other such data; 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 [33]: def deprocess_image(x):
    """utility function to convert a float array into a valid uint8 image.

    # 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

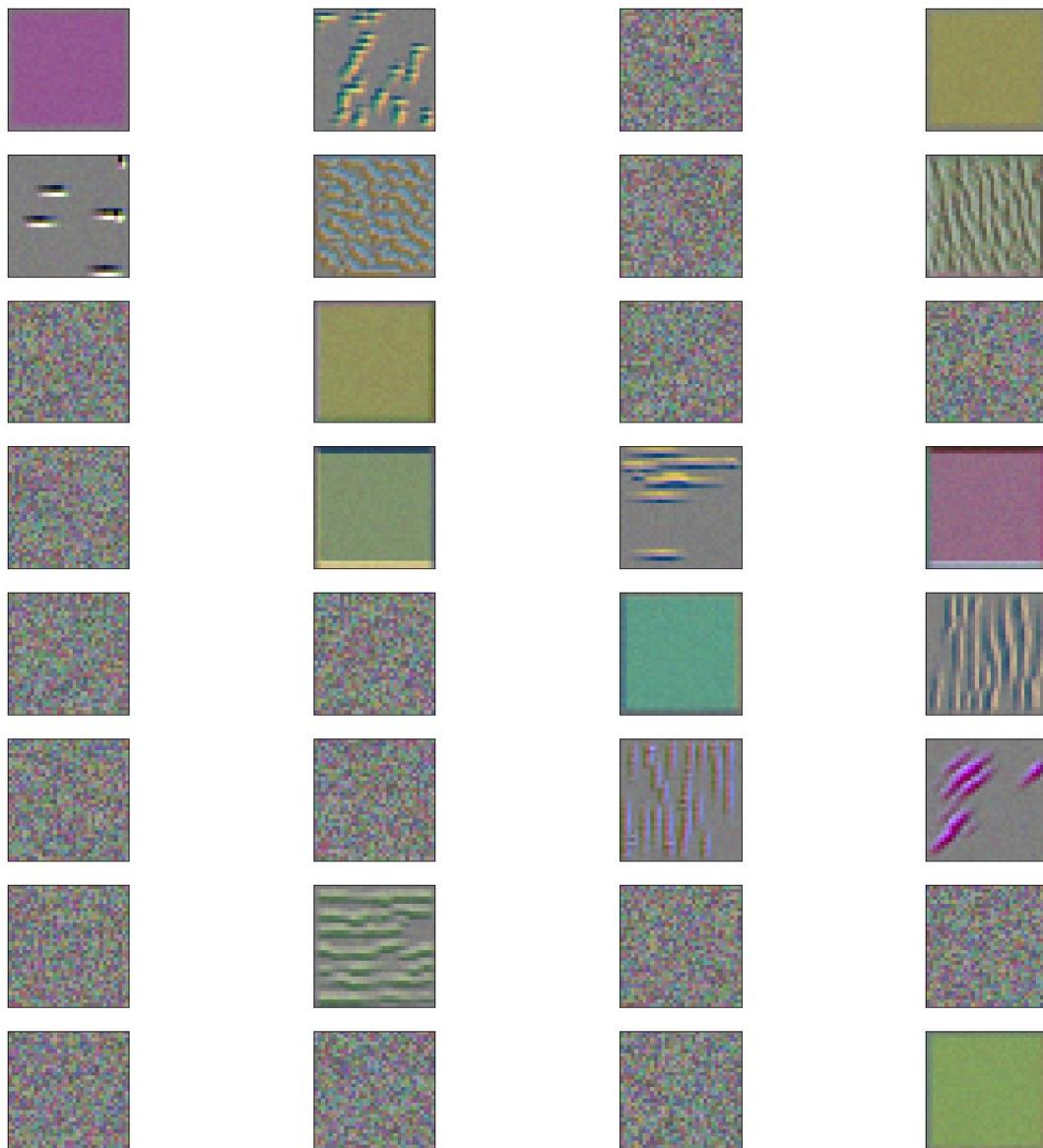
In [34]: def generate_pattern(layer_name, filter_index, size=100):
    layer_output = model.get_layer(layer_name).output
    #print(layer_output.shape)
    loss = K.mean(layer_output[:, :, :, filter_index])
    grads = K.gradients(loss, model.input)[0]
    grads /= (K.sqrt(K.mean(K.square(grads))) + 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 [35]: #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))

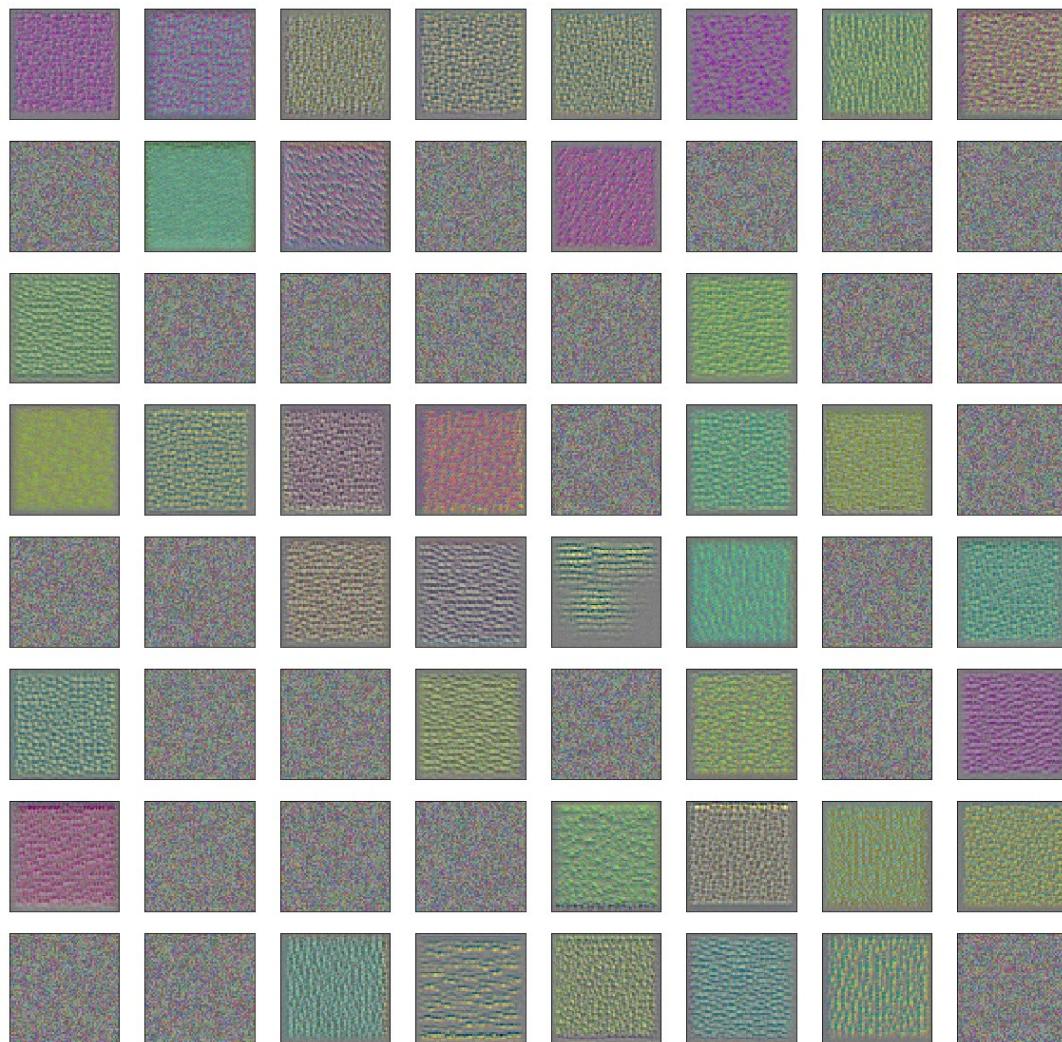
['conv2d_1', 'activation_1', 'max_pooling2d_1', 'dropout_1', 'conv2d_2', 'activation_2', 'max_poolin
g2d_2', 'dropout_2', 'conv2d_3', 'activation_3', 'max_pooling2d_3', 'dropout_3', 'conv2d_4', 'activa
tion_4', 'max_pooling2d_4', 'dropout_4', 'flatten_1', 'dense_1', 'activation_5', 'batch_normalizatio
n_1', 'dropout_5', 'dense_2', 'activation_6']
23
```

```
In [36]: # plot the filters for convolutional layers utilizing gradient decomposition
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_rows_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()
```

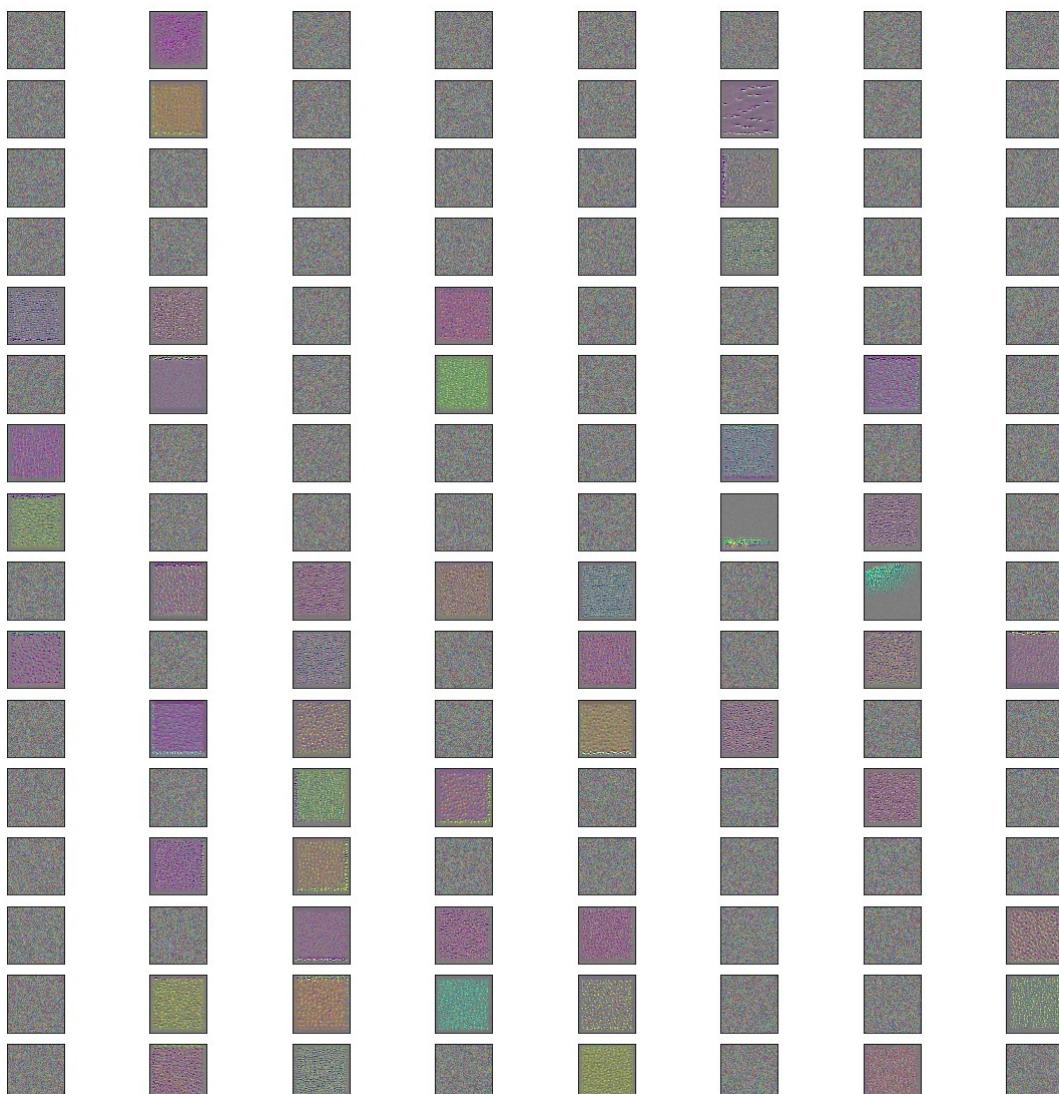
```
Tensor("activation_1/Relu:0", shape=(None, 98, 98, 32), dtype=float32)
32
(8, 4)
```



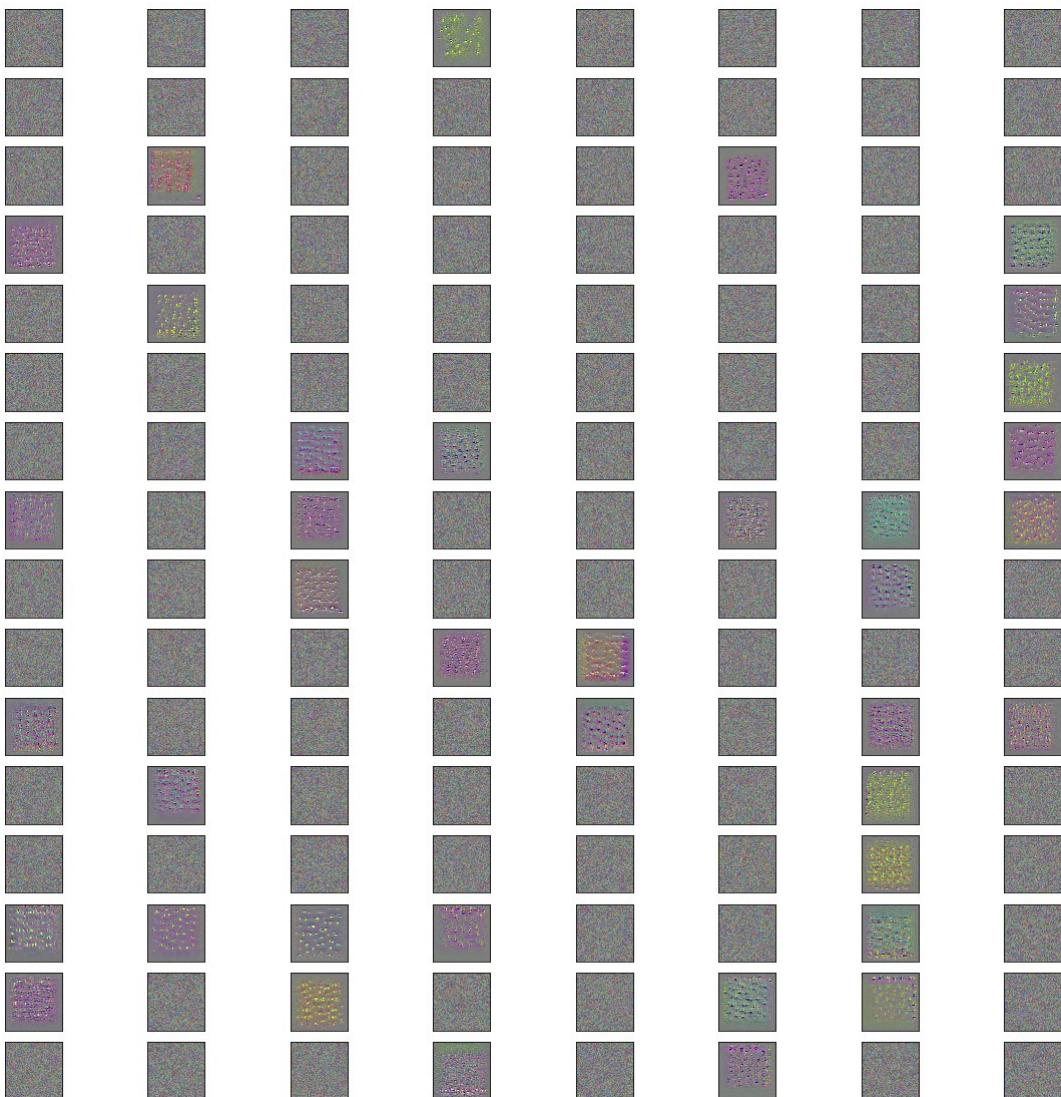
```
Tensor("activation_2/Relu:0", shape=(None, 47, 47, 64), dtype=float32)
64
(8, 8)
```



```
Tensor("activation_3/Relu:0", shape=(None, 21, 21, 128), dtype=float32)
128
(16, 8)
```



```
Tensor("activation_4/Relu:0", shape=(None, 8, 8, 128), dtype=float32)
128
(16, 8)
```



In []: