Computational Intelligence Lab

Assignment 5

Karolina Kotlowska czwartek 9:30 IO

5.1 Reuse horses or humans classifier

In [1]:
```
import tensorflow_datasets as tfds
import tensorflow as tf
ds_train = tfds.load('horses_or_humans', split='train', as_supervised=True, shuffle_fi
ds_test = tfds.load('horses_or_humans', split='test', as_supervised=True, shuffle_file
```

```
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Un
known size, total: Unknown size) to C:\Users\krzyc\tensorflow_datasets\horses_or_huma
ns\3.0.0...
Dl Completed...: 0 url [00:00, ? url/s]
Dl Size...: 0 MiB [00:00, ? MiB/s]
Generating splits...:   0%|          | 0/2 [00:00<?, ? splits/s]
Generating train examples...: 0 examples [00:00, ? examples/s]
Shuffling C:\Users\krzyc\tensorflow_datasets\horses_or_humans\3.0.0.incompleteKCL9C8
\horses_or_humans-train.tf…
Generating test examples...: 0 examples [00:00, ? examples/s]
Shuffling C:\Users\krzyc\tensorflow_datasets\horses_or_humans\3.0.0.incompleteKCL9C8
\horses_or_humans-test.tfr…
Dataset horses_or_humans downloaded and prepared to C:\Users\krzyc\tensorflow_dataset
s\horses_or_humans\3.0.0. Subsequent calls will reuse this data.
```

In [79]:
```
from keras import models
from keras import layers

model1 = models.Sequential(
    [layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(300,
     layers.Conv2D(filters=32, kernel_size=(1,1), activation='relu', input_shape=(300,
     layers.MaxPooling2D(2,2),
     layers.Dropout(0.2),
     layers.Flatten(),
     layers.Dense(2, activation='softmax')]
)
```

TODO 5.1.1 Use your own previously developed classifier model. Train it (without augmentation) and save with model.save(filename).

For demonstration puproses we will download a previously trained model (which was used in lab 4),

In [5]:
```
!python -m wget https://dysk.agh.edu.pl/s/x46rDRkLjeaGKz5/download/horses_or_humans_cr
```

```
Saved under horses_or_humans_cnn.h5
```

In [6]:
```
from keras.models import load_model
model = load_model('horses_or_humans_cnn.h5')

model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)          (None, 298, 298, 16)      448

 max_pooling2d_20 (MaxPoolin  (None, 149, 149, 16)     0
 g2D)

 dropout_16 (Dropout)        (None, 149, 149, 16)      0

 conv2d_21 (Conv2D)          (None, 147, 147, 32)      4640

 max_pooling2d_21 (MaxPoolin  (None, 73, 73, 32)       0
 g2D)

 dropout_17 (Dropout)        (None, 73, 73, 32)        0

 conv2d_22 (Conv2D)          (None, 71, 71, 64)        18496

 max_pooling2d_22 (MaxPoolin  (None, 35, 35, 64)       0
 g2D)

 dropout_18 (Dropout)        (None, 35, 35, 64)        0

 conv2d_23 (Conv2D)          (None, 33, 33, 64)        36928

 max_pooling2d_23 (MaxPoolin  (None, 16, 16, 64)       0
 g2D)

 dropout_19 (Dropout)        (None, 16, 16, 64)        0

 conv2d_24 (Conv2D)          (None, 14, 14, 64)        36928

 max_pooling2d_24 (MaxPoolin  (None, 7, 7, 64)         0
 g2D)

 flatten_4 (Flatten)         (None, 3136)              0

 dense_9 (Dense)             (None, 512)               1606144

 dense_10 (Dense)            (None, 2)                 1026

=================================================================
Total params: 1,704,610
Trainable params: 1,704,610
Non-trainable params: 0
_________________________________________________________________
```

TODO 5.1.2 Test it on ds_test. Don't forget the preprocessing part. At least indicate that you intend to uses batches! This model expects batches (the additional first dimension).

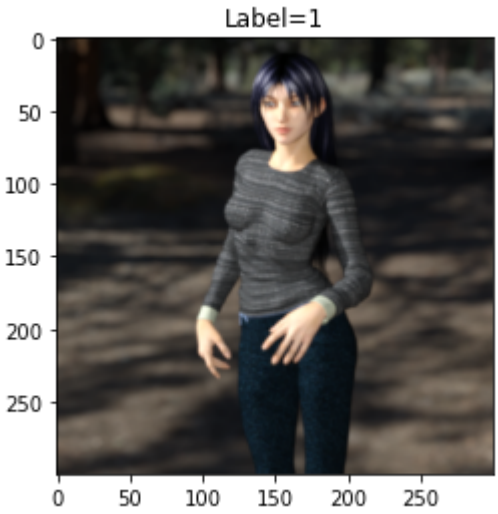
In [7]:
```
loss, acc = model.evaluate(ds_test.batch(64))
print("Accuracy", acc)
```

```
4/4 [==============================] - 3s 553ms/step - loss: 2.3288 - accuracy: 0.828
1
Accuracy 0.828125
```

TODO 5.1.3 Test it on an image taken from ds_train

In [8]:
```
import matplotlib.pyplot as plt
import numpy as np
# This was the 8-th image in ds_train


it = ds_train.take(9).it.as_numpy_iterator()
for i in range(0,9):
  image,label = it.next()
plt.title(f'Label={label}')
plt.imshow(image)
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x15e80a21b80>
```



Predict the output label... Be sure that the image has additional batch dimension

In [9]:
```
print(image.shape)
image = np.expand_dims(image,axis=0)
print(image.shape)
```

```
(300, 300, 3)
(1, 300, 300, 3)
```

In [10]:
```
preds = model.predict(image)
print(preds)
y_pred = np.argmax(preds, axis=1)
print(f'Predicted:{y_pred}')
```

```
1/1 [==============================] - 0s 436ms/step
[[3.625678e-09 1.000000e+00]]
Predicted:[1]
```

5.2 Display layer activations

Collect layer names. **Those are the layer names in the model used as an example. In your model the names are probably different**

In [88]:
```
layer_names = [l.name for l in model1.layers]
print(layer_names)
```

```
['conv2d', 'conv2d_1', 'max_pooling2d', 'dropout', 'flatten', 'dense']
```

Collect layer outputs of the model into a new activation_model, which for a given input image produces multiple outputs

In [12]:
```
from keras import models

# Extracts the outputs of the first 14 layers:
# The exaxct number depends on the model!
layer_outputs = [layer.output for layer in model.layers[:14]]

# Creates a model that will return these outputs, given the model input:
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
```

Supply the image and read activations... Then inspect the result.

In [13]:
```
activations = activation_model.predict(image)
print(len(activations))

for i,a in enumerate(activations):
  print(f'{layer_names[i]} : {a.shape}') #ostatni paramtetr to channels
```

```
1/1 [==============================] - 0s 158ms/step
14
conv2d_20 : (1, 298, 298, 16)
max_pooling2d_20 : (1, 149, 149, 16)
dropout_16 : (1, 149, 149, 16)
conv2d_21 : (1, 147, 147, 32)
max_pooling2d_21 : (1, 73, 73, 32)
dropout_17 : (1, 73, 73, 32)
conv2d_22 : (1, 71, 71, 64)
max_pooling2d_22 : (1, 35, 35, 64)
dropout_18 : (1, 35, 35, 64)
conv2d_23 : (1, 33, 33, 64)
max_pooling2d_23 : (1, 16, 16, 64)
dropout_19 : (1, 16, 16, 64)
conv2d_24 : (1, 14, 14, 64)
max_pooling2d_24 : (1, 7, 7, 64)
```

We will define a function that will display a grid of activations for each channel... As you may observe the numbers of filters, hence channels, differ. Use grid_shape to adjust layout and number of filters.

In [14]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
import numpy as np

def display(tensor,grid_shape=(4,4),start=0,title=None):
  fig = plt.figure(figsize=(grid_shape[1]*2, grid_shape[0]*2))
```

```
  grid = ImageGrid(fig, 111,  # similar to subplot(111)
                   nrows_ncols=grid_shape,  # creates 2x2 grid of axes
                   axes_pad=0.05,  # pad between axes in inch.
                   )
  images = [tensor[0, :, :, start+i] for i in range(grid_shape[0]*grid_shape[1])]
  for ax, im in zip(grid, images):
      # Iterating over the grid returns the Axes.
      ax.imshow(im,cmap='viridis')
      ax.axis('off')

  # fig.tight_layout()
  # fig.subplots_adjust(top=0.95)
  if title is not None:
    plt.suptitle(title)
    print('-----')
  plt.show()
```

TODO 5.2.1 Display activations for all convolution layers

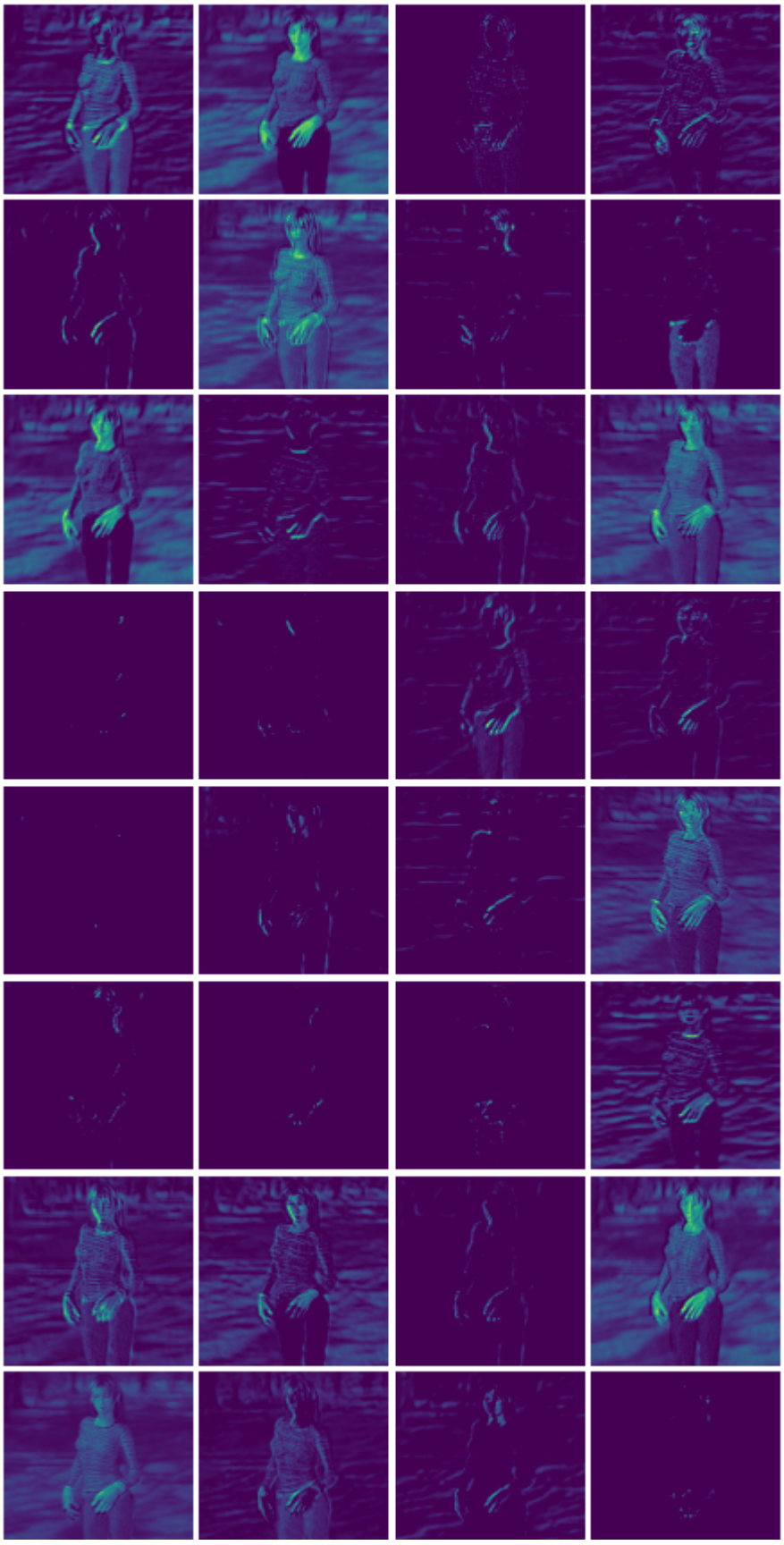
In [15]:
```
display(activations[0],grid_shape=(4,4),title=layer_names[0])
display(activations[3],grid_shape=(8,4),title=layer_names[3])
display(activations[6],grid_shape=(8,8),title=layer_names[6])

display(activations[9],grid_shape=(8,4),title=layer_names[9])
display(activations[12],grid_shape=(8,4),title=layer_names[12])
```

```
-----
```

conv2d_20

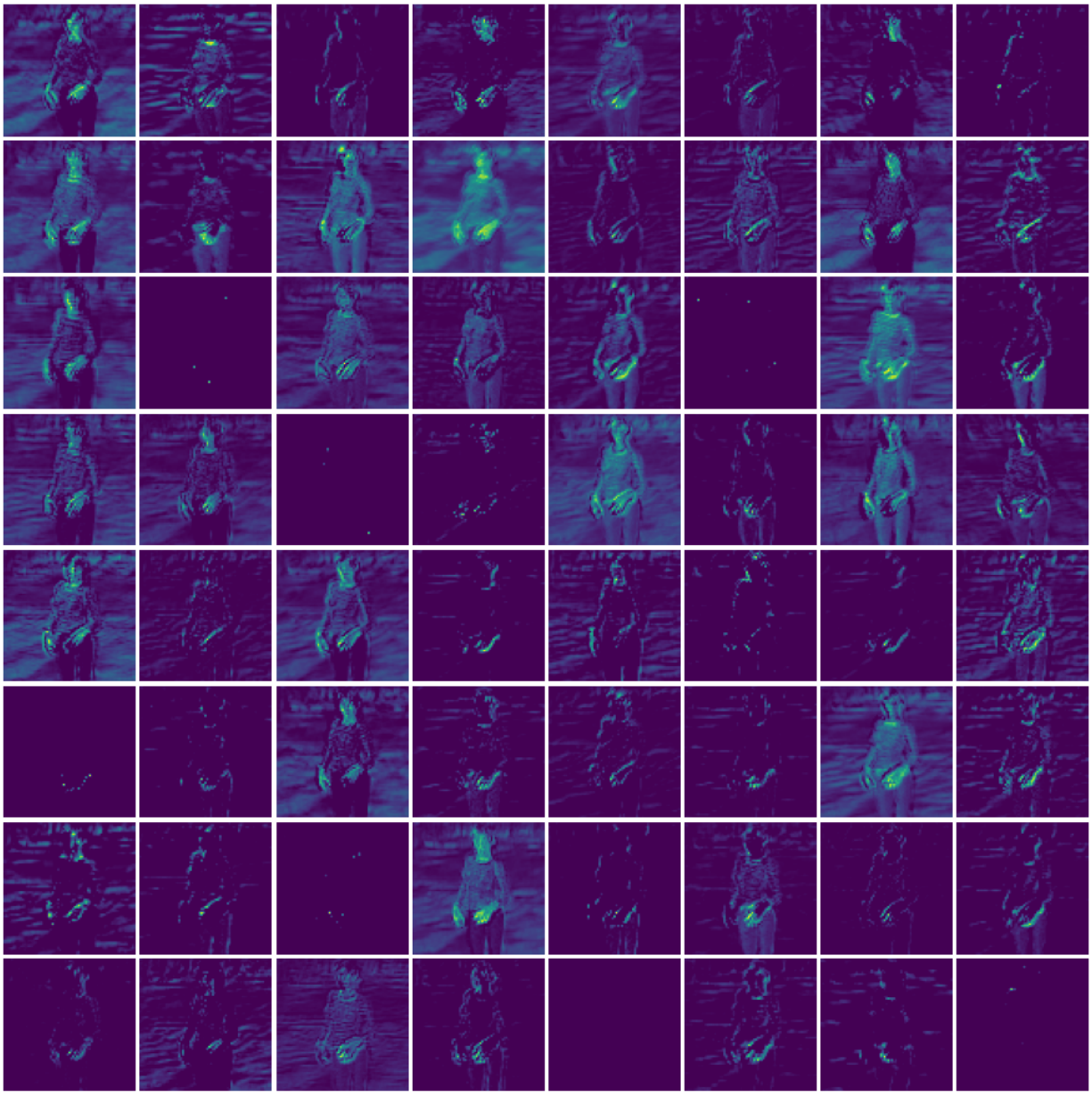
```
-----
```

conv2d_21



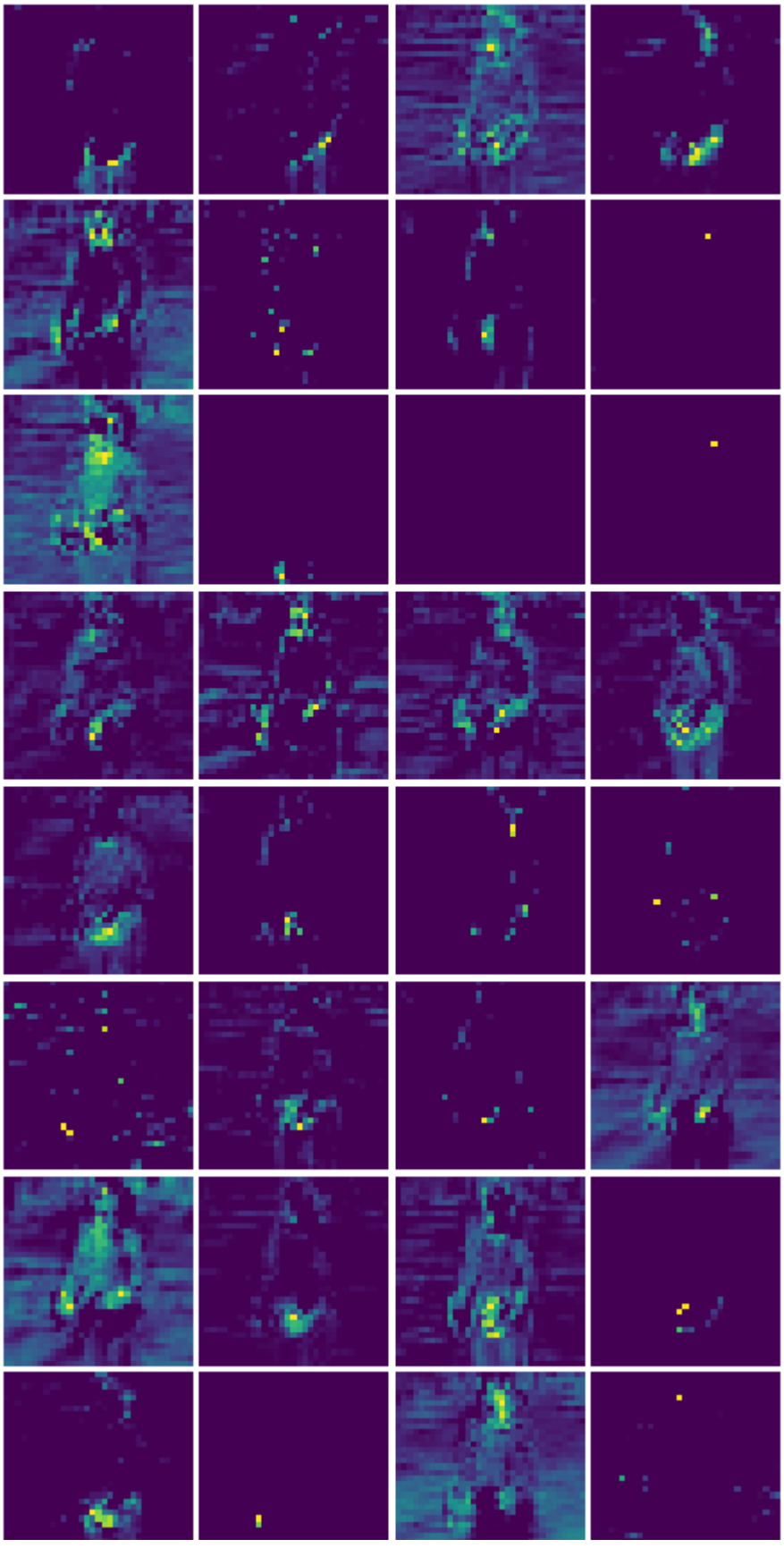
- - - - -

conv2d_22



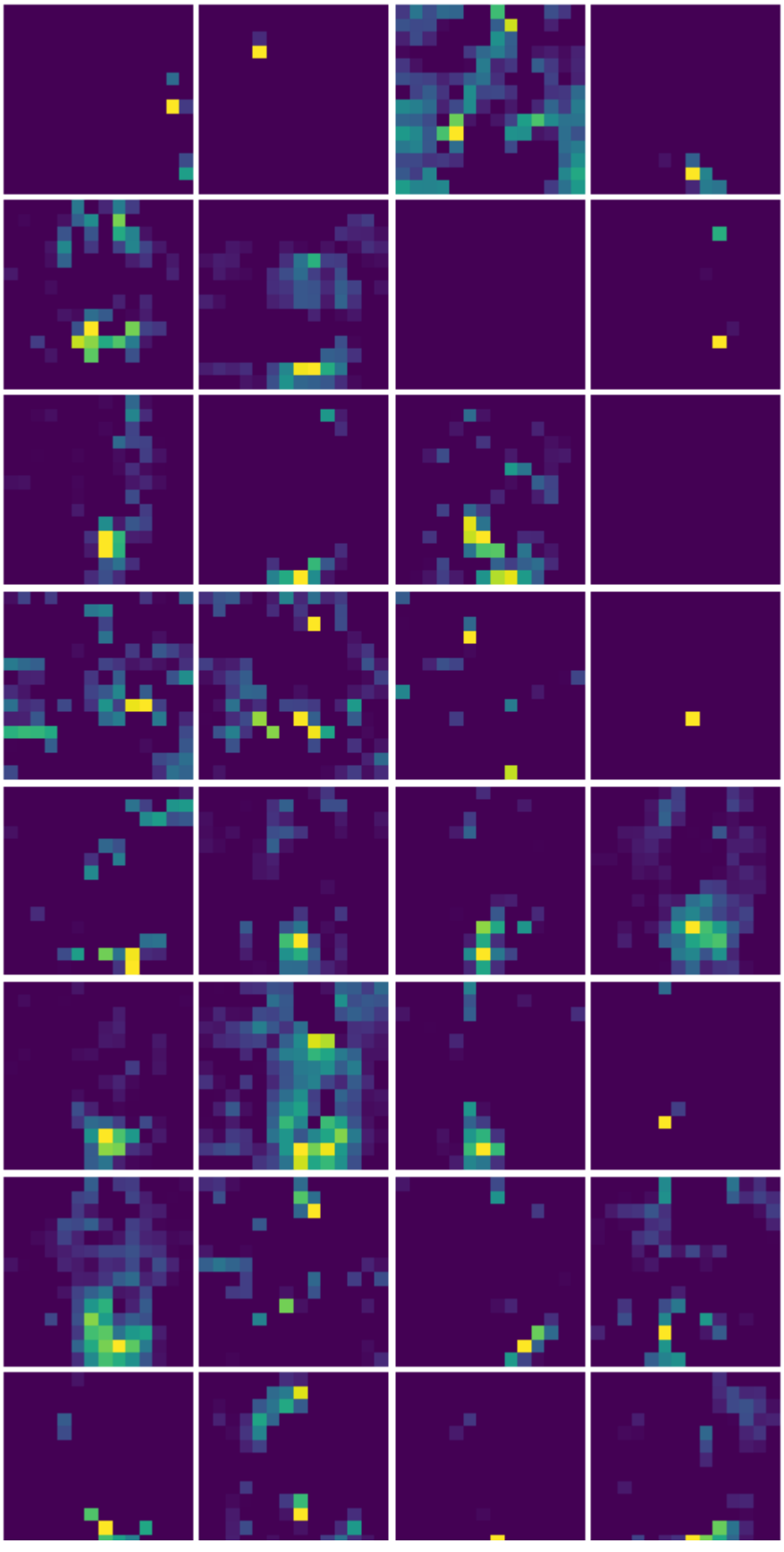
- - - - -

conv2d_23



- - - - -

conv2d_24



5.2.1 Test on a horse

TODO 5.2.2 Load a horse photo, perform classification and display activations

In [16]:
```
from PIL import Image
import requests
from io import BytesIO
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (10, 10)


response = requests.get('https://www.helpfulhorsehints.com/wp-content/uploads/quarter-
img = Image.open(BytesIO(response.content))


plt.matshow(img)
```

Out[16]:
```
<matplotlib.image.AxesImage at 0x15e85848d90>
```

In [17]:
```
horse = np.array(img)
print(horse.shape)
```

```
(597, 1024, 3)
```

The image should be transformed.

- Select a region
- Cut out a part
- Resize

In [18]:
```
# To see which part will be delivered to the classifier...

horse[30:900,200:900,0] = np.clip(horse[30:900,200:900,0],0,205)
horse[30:900,200:900,0]+=50
plt.matshow(horse)
```

Out[18]:
```
<matplotlib.image.AxesImage at 0x15e859a70d0>
```

In [19]:
```
from keras import layers
X = horse[100:600,100:600,:]
resize = layers.Resizing(300,300)
X=resize(X).numpy()
print(X.shape)
```

```
(300, 300, 3)
```

Classify

In [20]:
```
X_as_batch = X.reshape(1,300,300,3)
preds = model.predict(X_as_batch)
print(preds)
y_pred = np.argmax(preds,axis=1)
print(f'Predicted:{y_pred}')
```

```
1/1 [==============================] - 0s 281ms/step
[[0.99046844 0.00953153]]
Predicted:[0]
```

Display activations

In [21]:
```
activations = activation_model.predict(X_as_batch)
display(activations[0],grid_shape=(4,4),title=layer_names[0])
display(activations[3],grid_shape=(8,4),title=layer_names[3])
display(activations[6],grid_shape=(8,8),title=layer_names[6])
```

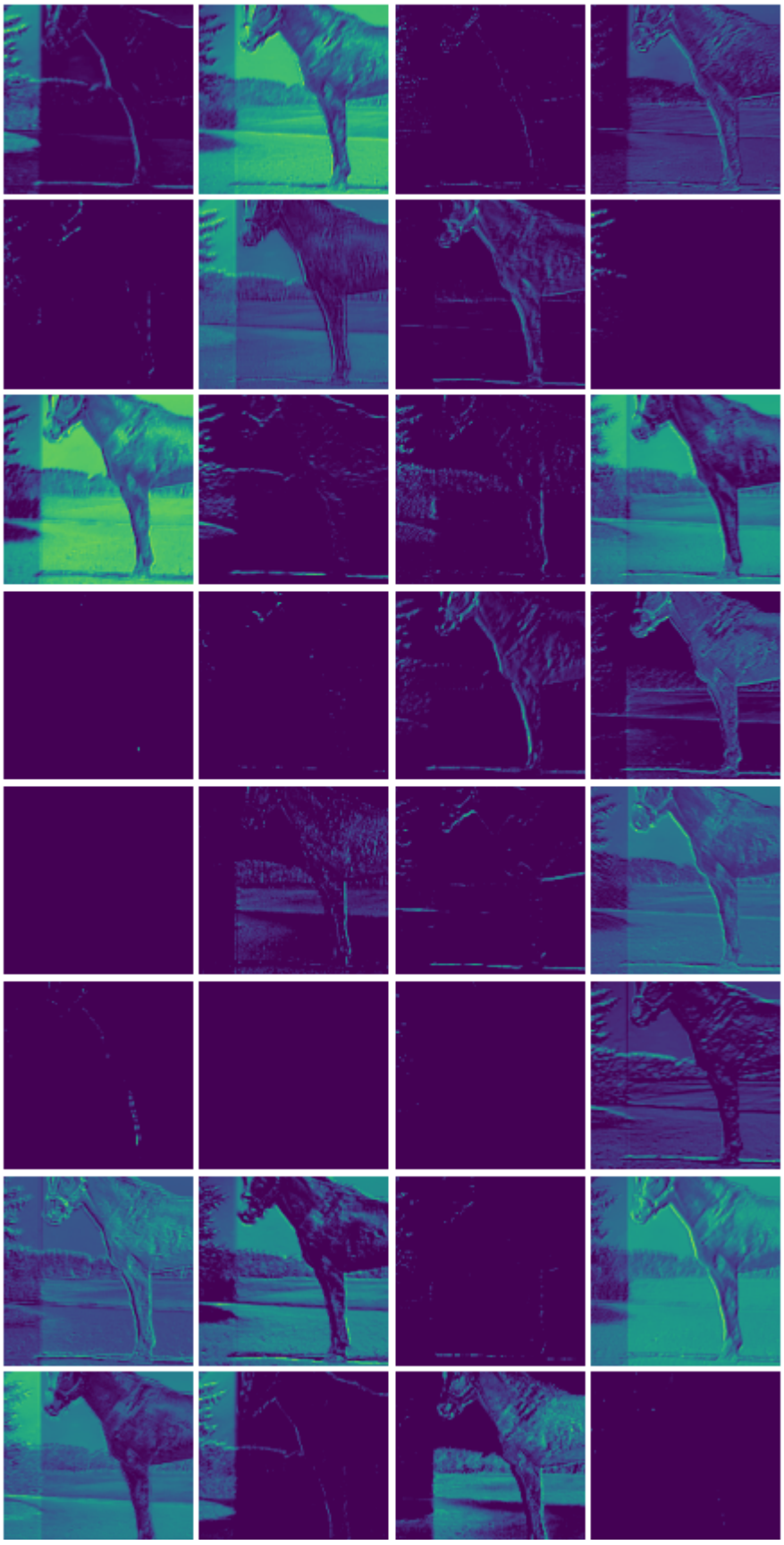
```
display(activations[9],grid_shape=(8,4),title=layer_names[9])
display(activations[12],grid_shape=(8,4),title=layer_names[12])
```

```
1/1 [==============================] - 0s 238ms/step
-----
```

conv2d_20

```
-----
```

conv2d_21

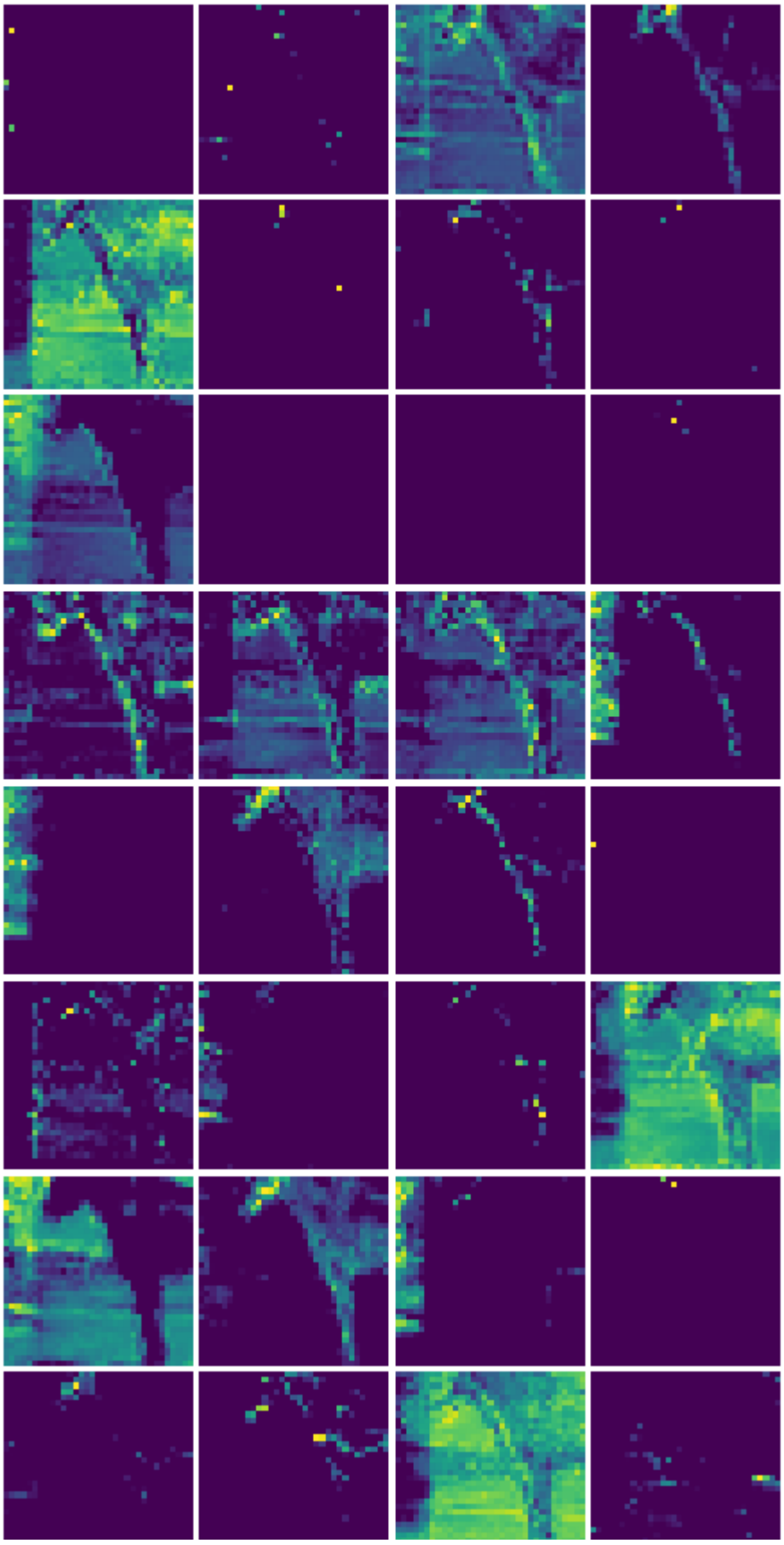


---

conv2d_22

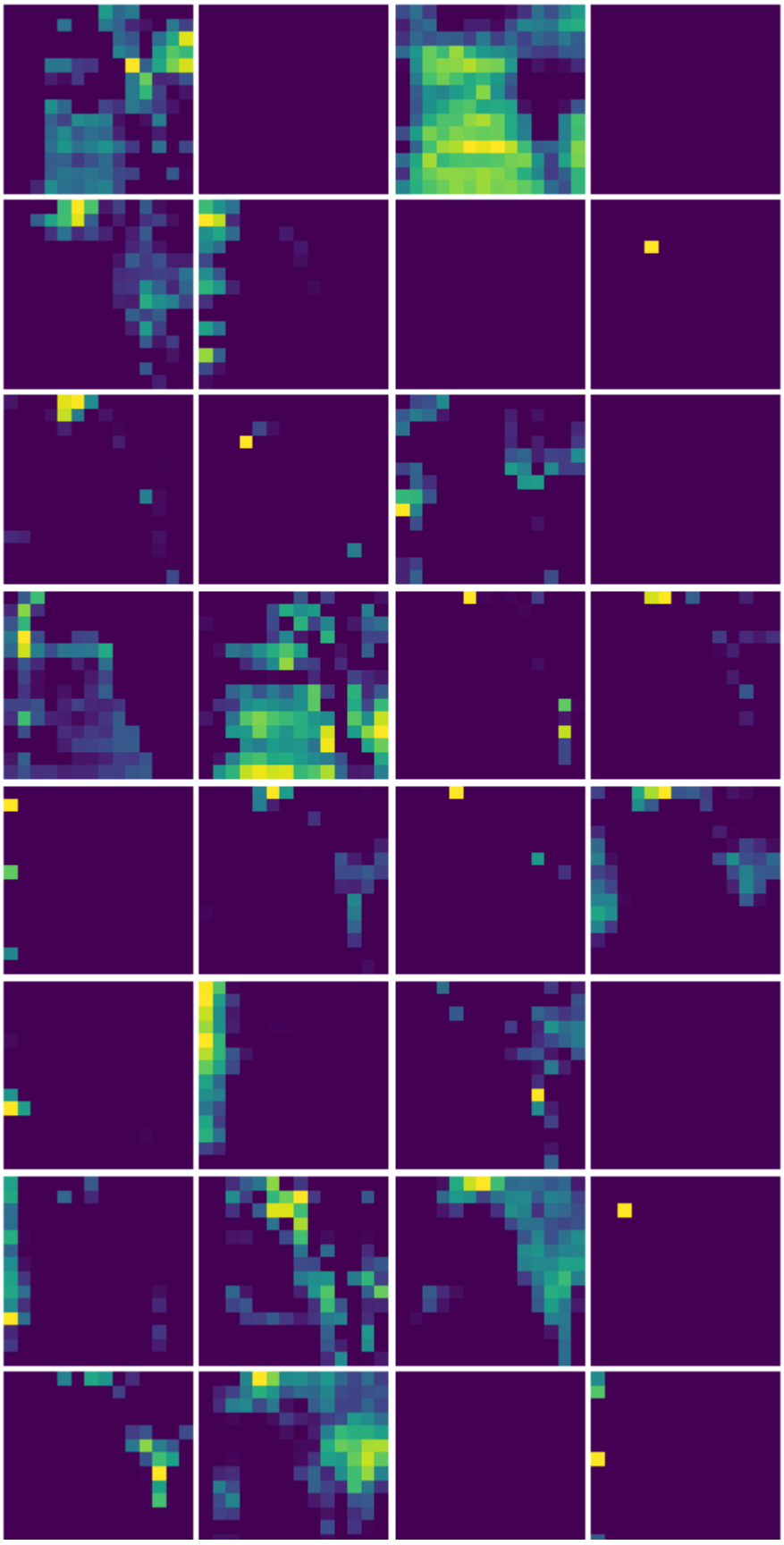
---

conv2d_23



- - - - -

conv2d_24



5.2.2 Test on a human

TODO 5.2.3 Select and load a photo of a human, perform classification and display activations

The first image seems similar to these in the training set

In [22]:
```
from PIL import Image
import requests
from io import BytesIO
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (10, 10)

response = requests.get('https://i.pinimg.com/564x/ab/ea/47/abea47528a8810ebe709cdee0f
img = Image.open(BytesIO(response.content))
plt.imshow(img)
```

Out[22]:
```
<matplotlib.image.AxesImage at 0x15e82caa100>
```

The second might be more demanding...

In [23]:
```
response = requests.get('https://cdn.shopify.com/s/files/1/0333/9653/products/adidas-c
img = Image.open(BytesIO(response.content))
plt.imshow(img)
```

Out[23]:
```
<matplotlib.image.AxesImage at 0x15e82c65ac0>
```

In [24]:
```
image = np.array(img)
print(image.shape)
```

```
(240, 240, 3)
```

In [25]:
```
from keras import layers
resize = layers.Resizing(300,300)
X=resize(image).numpy()
print(X.shape)
```

```
(300, 300, 3)
```

In [26]:
```
X_as_batch = X.reshape(1,300,300,3)
preds = model.predict(X_as_batch)
print(preds)
```

```
y_pred = np.argmax(preds,axis=1)
print(f'Predicted:{y_pred}')
```

```
1/1 [==============================] - 0s 59ms/step
[[0.38133964 0.61866033]]
Predicted:[1]
```

Display activations

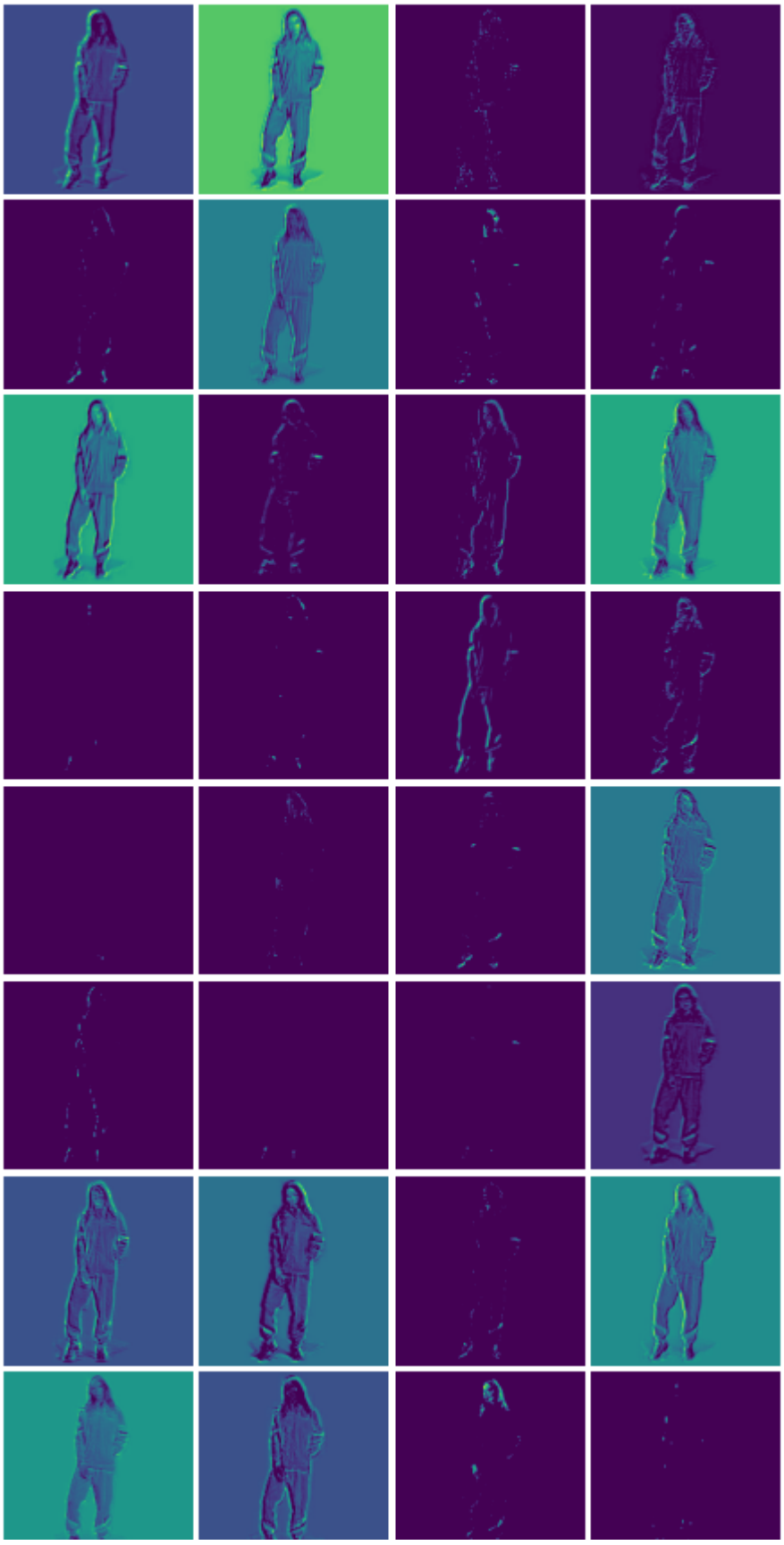
In [27]:
```
activations = activation_model.predict(X_as_batch)
display(activations[0],grid_shape=(4,4),title=layer_names[0])
display(activations[3],grid_shape=(8,4),title=layer_names[3])
display(activations[6],grid_shape=(8,8),title=layer_names[6])

display(activations[9],grid_shape=(8,4),title=layer_names[9])
display(activations[12],grid_shape=(8,4),title=layer_names[12])
```

```
1/1 [==============================] - 0s 52ms/step
-----
```

conv2d_20

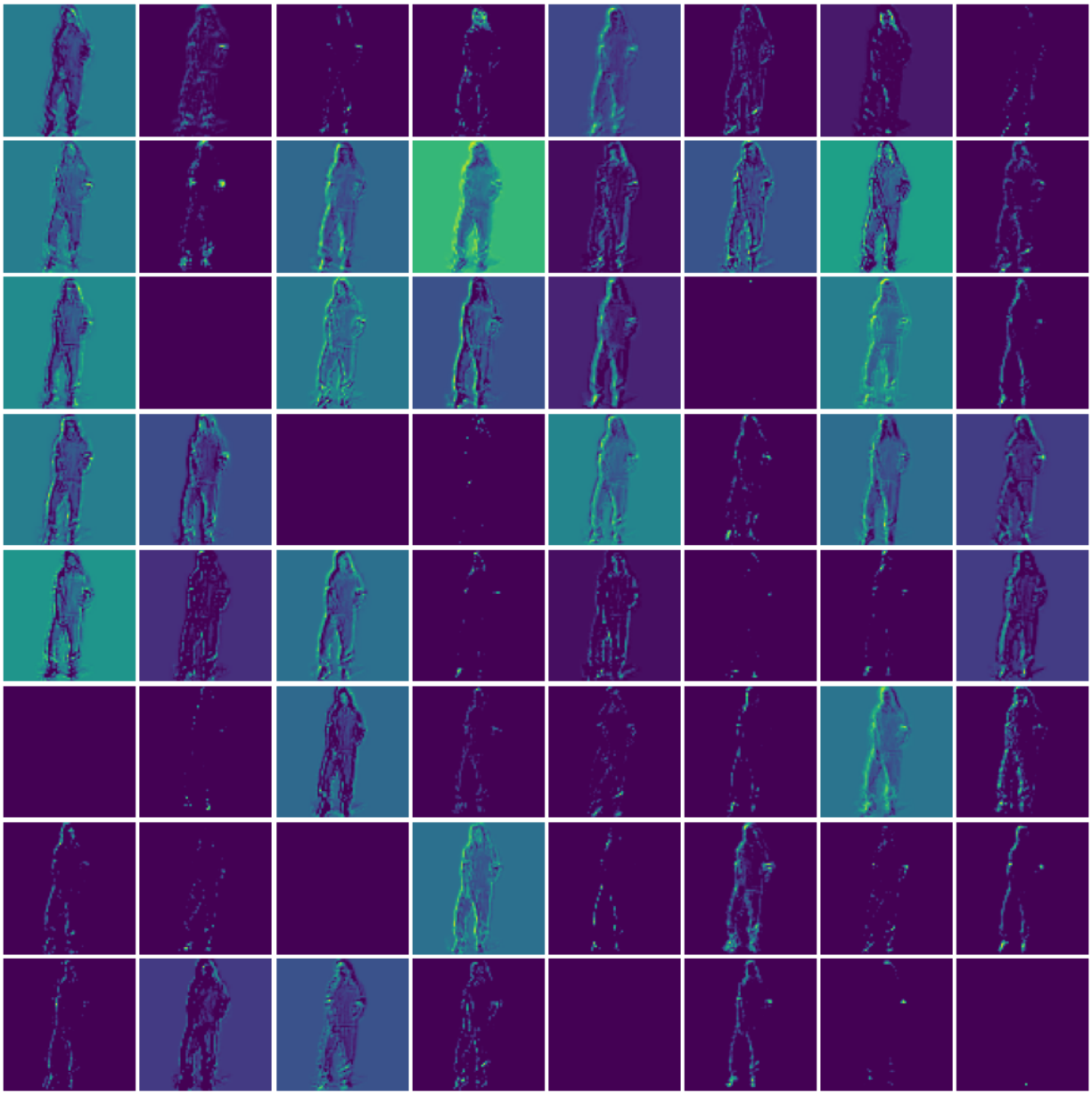
```
-----
```

conv2d_21



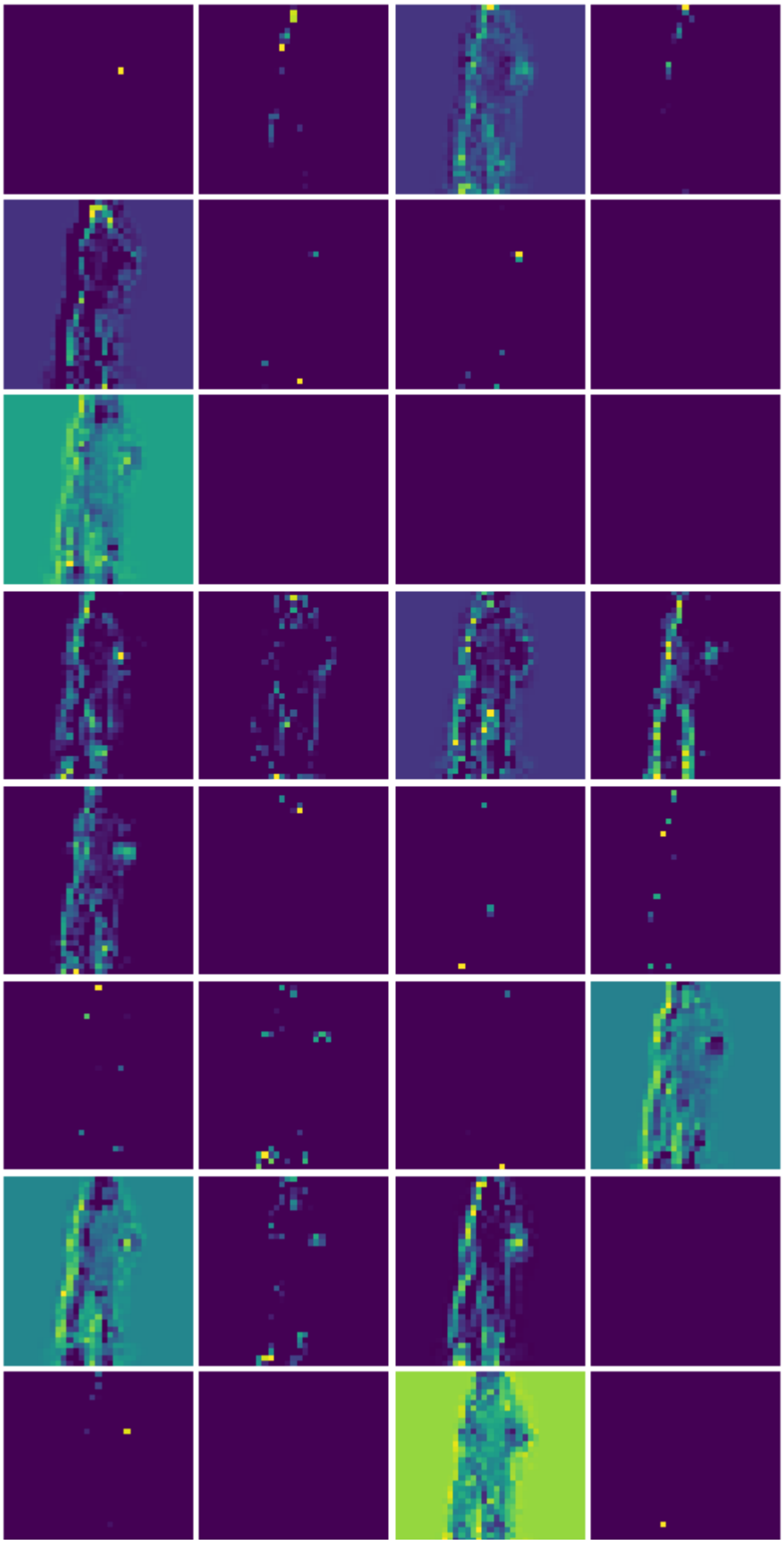
- - - - -

conv2d_22



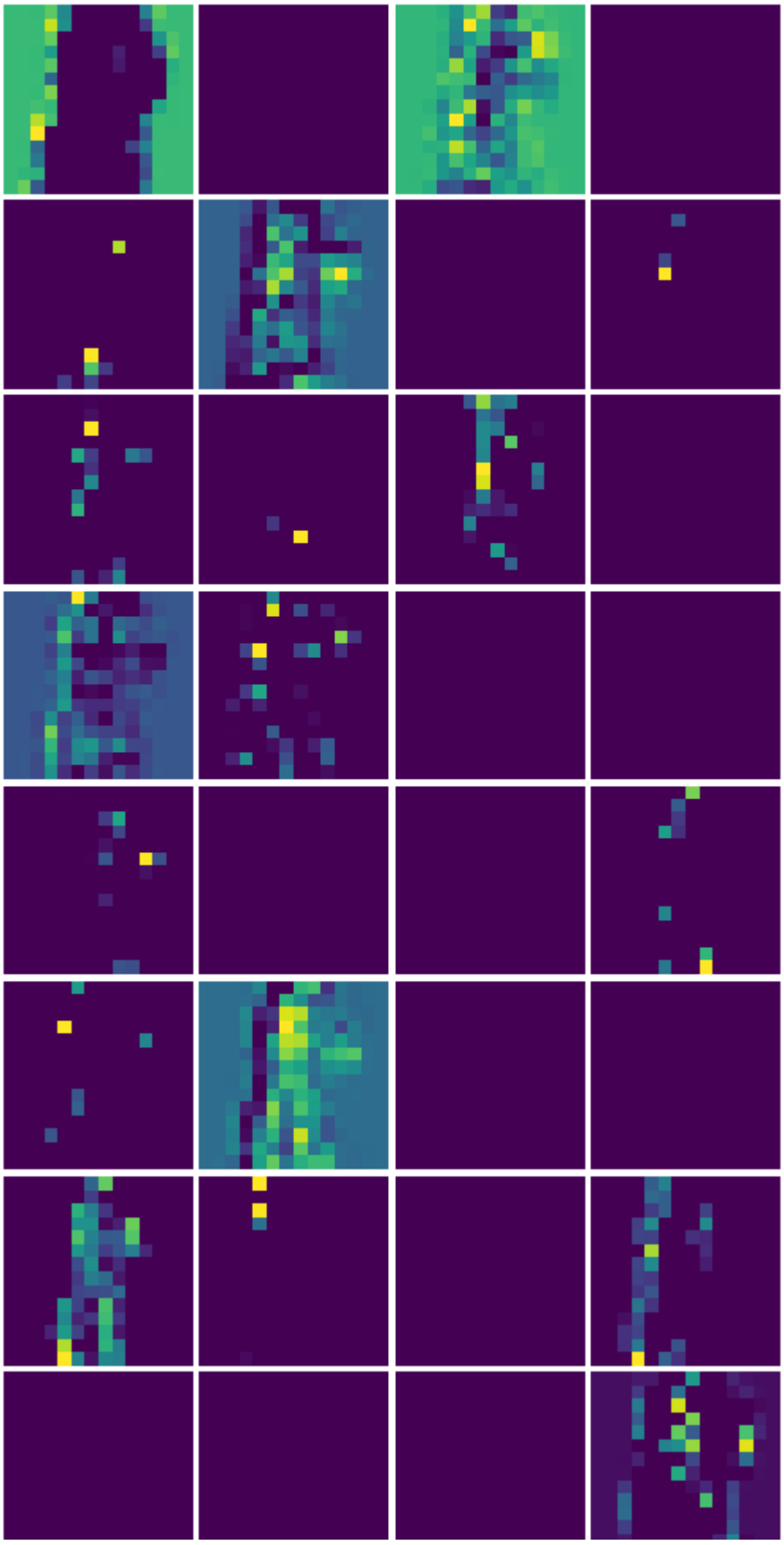
- - - - -

conv2d_23



- - - - -

conv2d_24



5.3 Visualize filters

The goal is to find pixel values and locations that contribute the most to activations of a given layer.

This is an optimization problem.

- Let X be an image (3D input array)
- Let $f(X)_i$ be an output of a particular i-th filter
- Let $F(f(X)_i)$ be a goal function to be optimized

The maximum filter response occurs if the input is equal $X^* = \arg\max_X F(f(X)_i)$

Remark The code below is not executed in the eager mode. Please restart the execution environment and reload the model.

In [28]:
```
import tensorflow as tf
from keras import backend as K
tf.compat.v1.disable_eager_execution()

from keras.models import load_model
model = load_model('horses_or_humans_cnn.h5')

model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)          (None, 298, 298, 16)      448

 max_pooling2d_20 (MaxPoolin  (None, 149, 149, 16)     0
 g2D)

 dropout_16 (Dropout)        (None, 149, 149, 16)      0

 conv2d_21 (Conv2D)          (None, 147, 147, 32)      4640

 max_pooling2d_21 (MaxPoolin  (None, 73, 73, 32)       0
 g2D)

 dropout_17 (Dropout)        (None, 73, 73, 32)        0

 conv2d_22 (Conv2D)          (None, 71, 71, 64)        18496

 max_pooling2d_22 (MaxPoolin  (None, 35, 35, 64)       0
 g2D)

 dropout_18 (Dropout)        (None, 35, 35, 64)        0

 conv2d_23 (Conv2D)          (None, 33, 33, 64)        36928

 max_pooling2d_23 (MaxPoolin  (None, 16, 16, 64)       0
 g2D)

 dropout_19 (Dropout)        (None, 16, 16, 64)        0

 conv2d_24 (Conv2D)          (None, 14, 14, 64)        36928

 max_pooling2d_24 (MaxPoolin  (None, 7, 7, 64)         0
 g2D)

 flatten_4 (Flatten)         (None, 3136)              0

 dense_9 (Dense)             (None, 512)               1606144

 dense_10 (Dense)            (None, 2)                 1026

=================================================================
Total params: 1,704,610
Trainable params: 1,704,610
Non-trainable params: 0
_________________________________________________________________
```

In [29]:
```
layer_name = 'conv2d_20'
filter_index = 0

layer_output = model.get_layer(layer_name).output
goal = K.mean(layer_output[:, :, :, filter_index])
```

In [30]:
```
# The call to `gradients` returns a list of tensors (of size 1 in this case)
# hence we only keep the first element, which is a tensor.
grads = tf.gradients(goal, model.input)[0]
```

```
# Then gradients are scaled
grads /= (tf.math.sqrt(K.mean(tf.square(grads))) + 1e-5)
```

We define compute_goal_grads function.

As per the Keras/Tensorflow manual, this function [K.function] runs the computation graph that we have created in the code, taking input from the first parameter and extracting the number of outputs as per the layers mentioned in the second parameter.

In [31]:
```
compute_goal_grads = K.function([model.input], [goal, grads])

# Let's test it:
import numpy as np
goal_value, grads_value = compute_goal_grads([np.zeros((1, 300, 300, 3))])
print(goal_value)
```

```
0.010476801
```

Simple implementation of gradient ascent algorithm (suitable for maximization)

In [32]:
```
# We start from a gray image with some noise
input_img_data = np.random.random((1, 300, 300, 3)) * 20 + 128.

# Run gradient ascent for 40 steps
step = 1.  # this is the magnitude of each gradient update
for i in range(40):
    # Compute the loss value and gradient value
    goal_value, grads_value = compute_goal_grads([input_img_data])
    # Here we adjust the input image in the direction that maximizes the loss
    input_img_data += grads_value * step
```

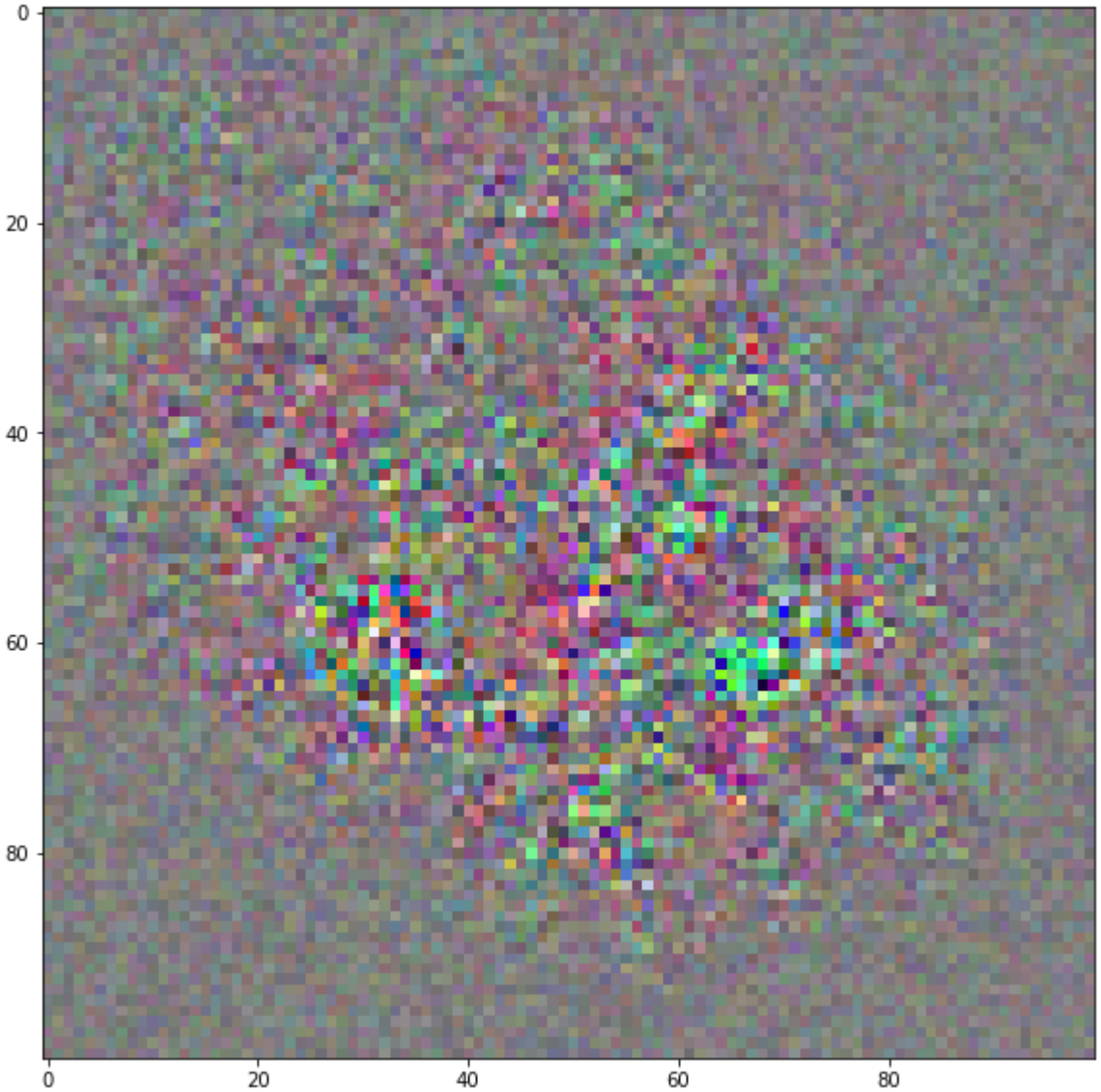
In [33]:
```
def convert_to_image(x):
    # normalize tensor: center on 0., ensure std is 0.1
    x -= x.mean()
    x /= (x.std() + 1e-5)
    x *= 0.1

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

    # convert to RGB array
    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

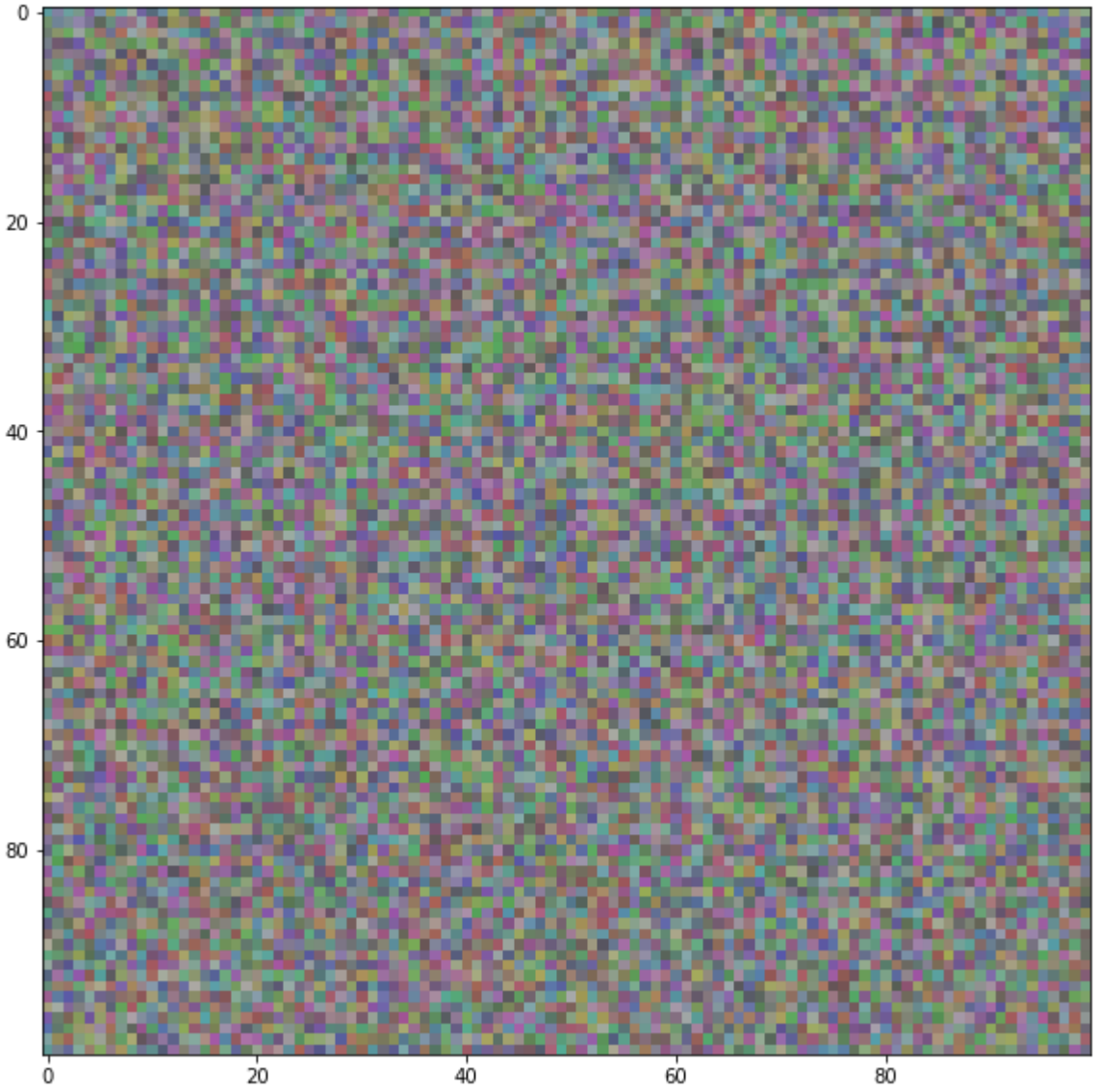
Finally, we arrange this into a function

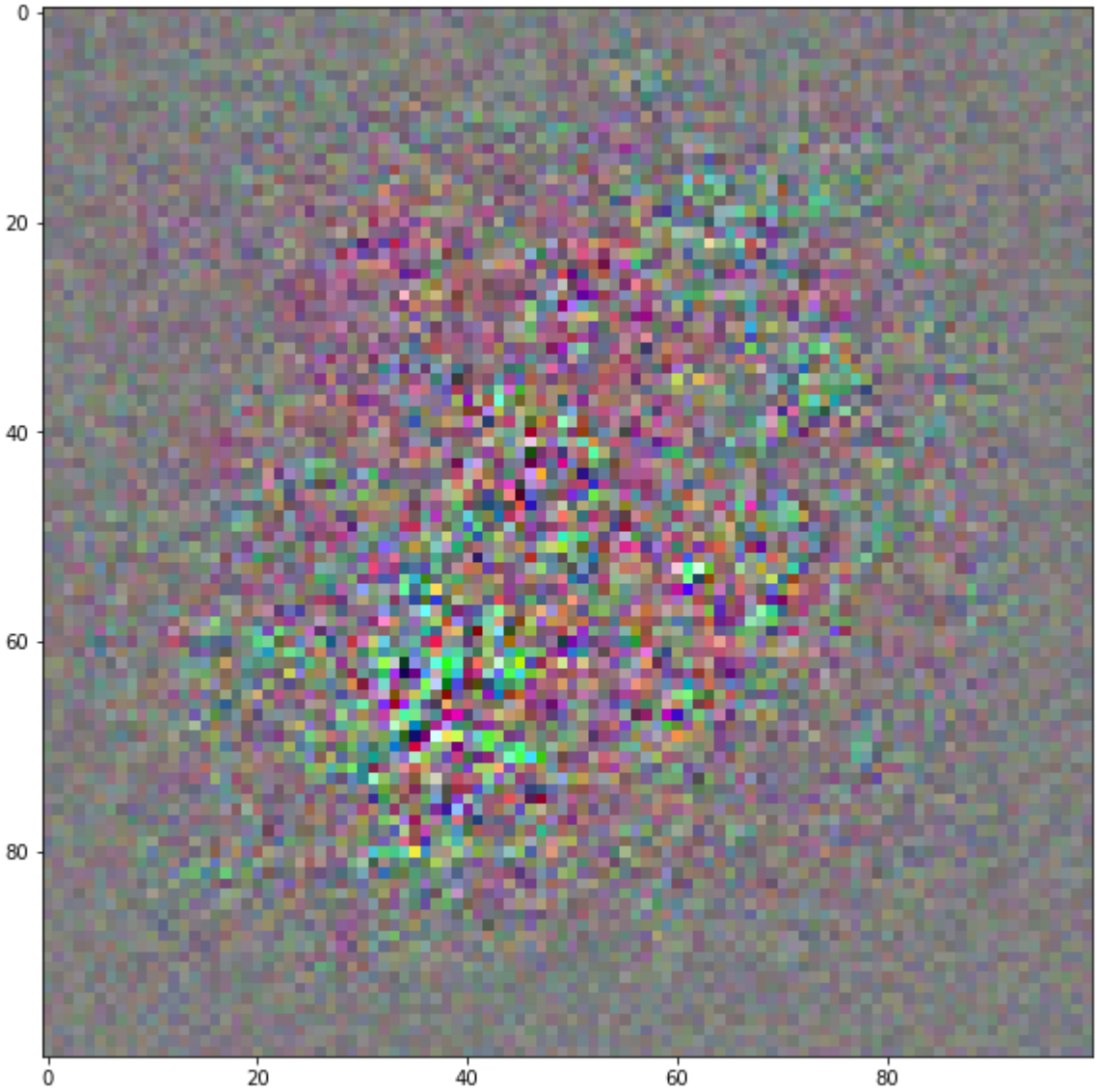
In [34]:
```
def generate_pattern(layer_name, filter_index, size=100):
    # Build a loss function that maximizes the activation of the nth filter of the lay
    layer_output = model.get_layer(layer_name).output
    goal = K.mean(layer_output[:, :, :, filter_index])

    # Compute the gradient of the input picture with this loss
    grads = K.gradients(goal, model.input)[0]

    # Normalization trick: we normalize the gradient
```

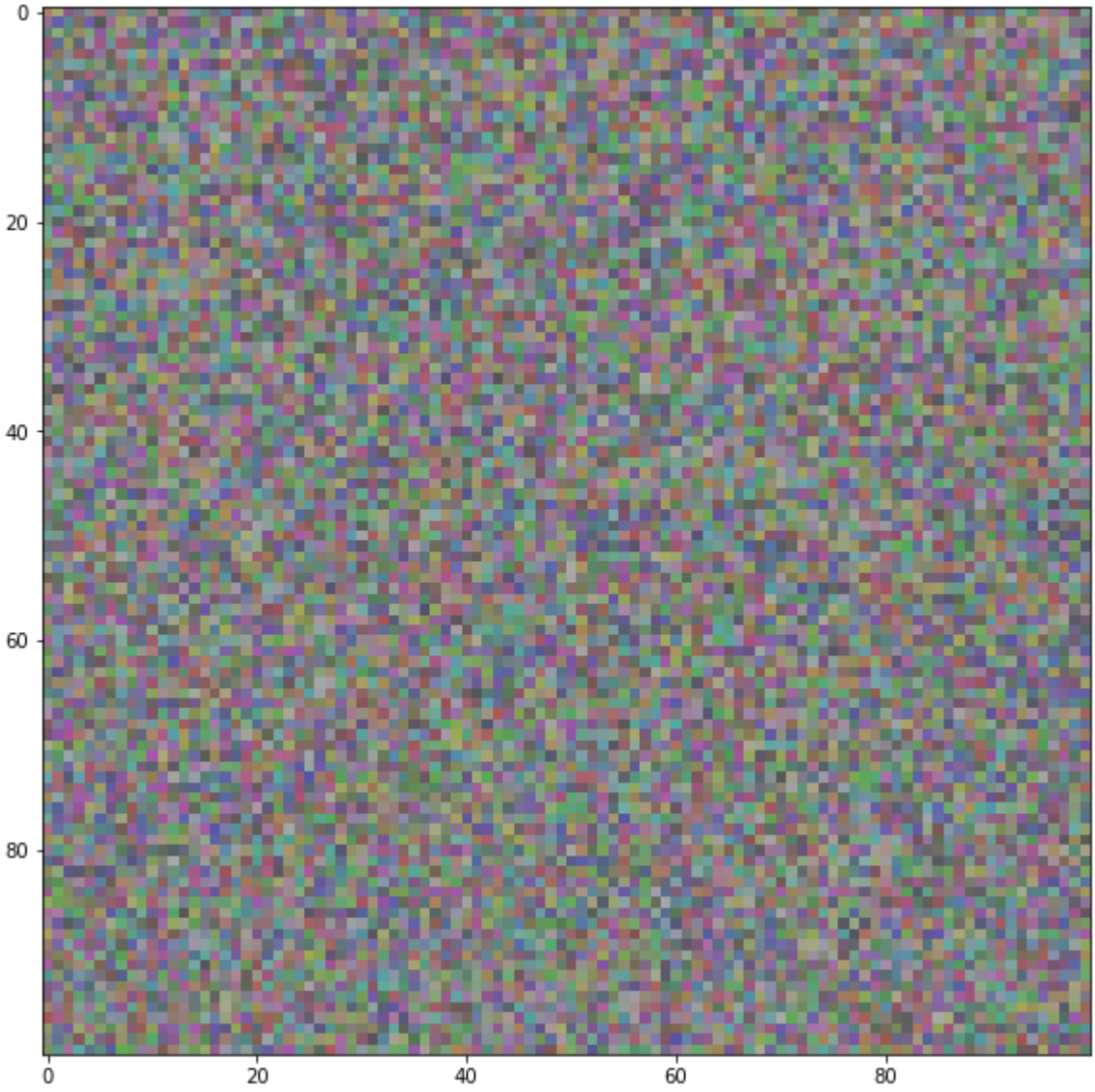
```
    grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)

    # This function returns the goal and grads given the input picture
    compute_goal_grads = K.function([model.input], [goal, grads])

    # We start from a gray image with some noise
    input_img_data = np.random.random((1, size, size, 3)) * 20 + 128.

    # Run gradient ascent for 40 steps
    step = 1.
    for i in range(40):
        goal_value, grads_value = compute_goal_grads([input_img_data])
        input_img_data += grads_value * step

    img = input_img_data[0]
    return convert_to_image(img)
```

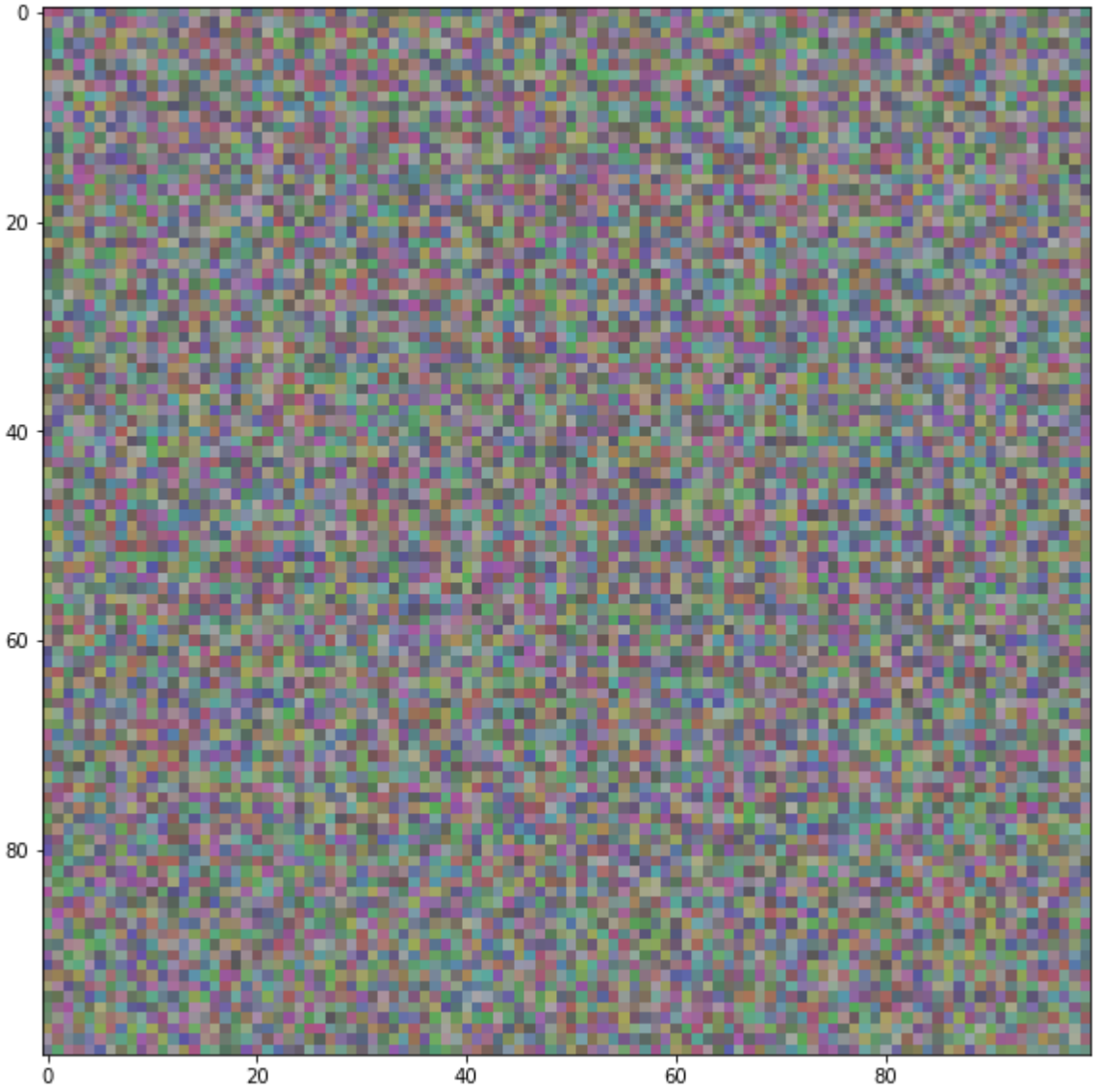
In [35]:
```
import matplotlib.pyplot as plt
for i in range(16):
    plt.imshow(generate_pattern('conv2d_24', i))
    plt.show()
```

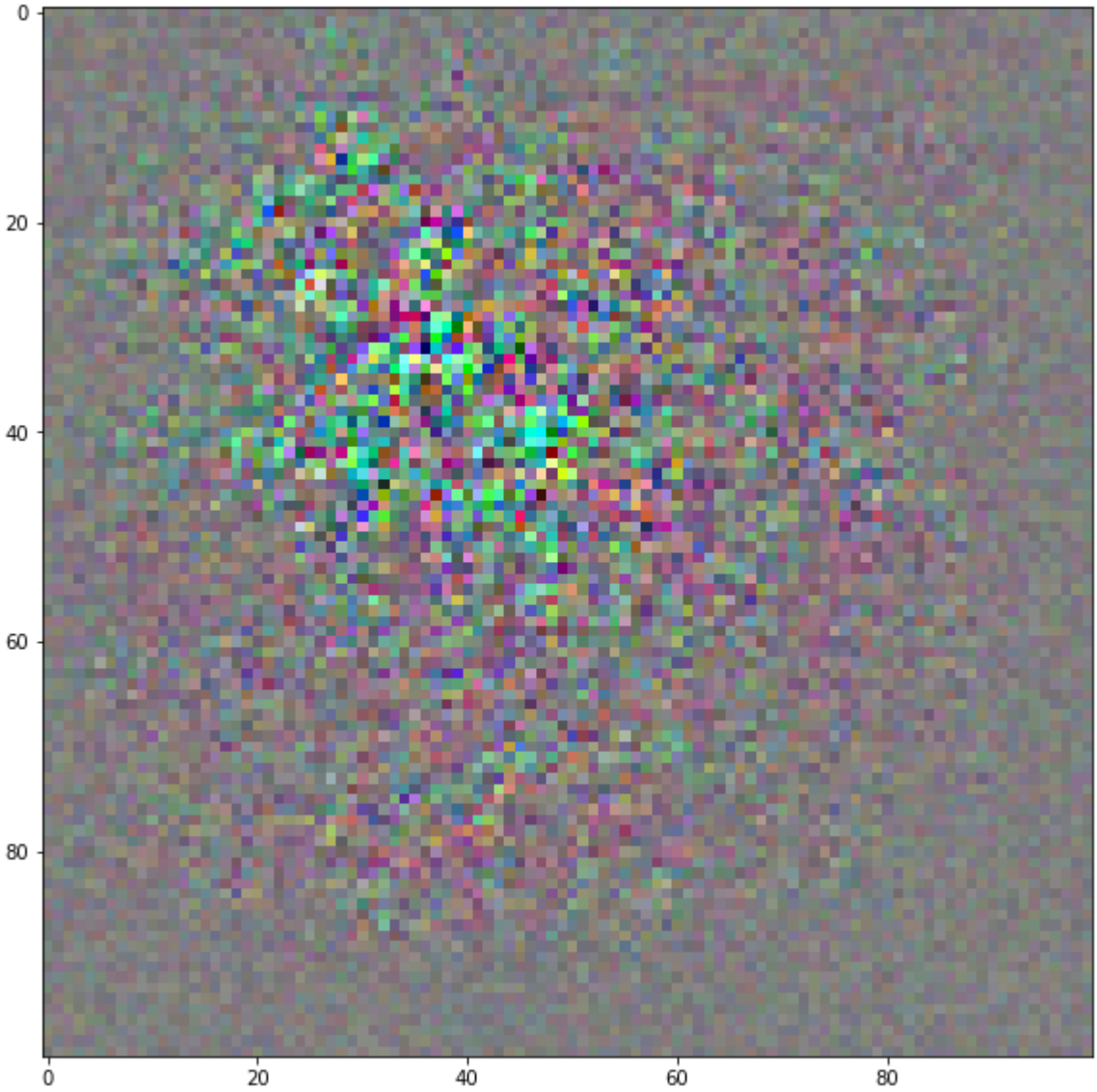


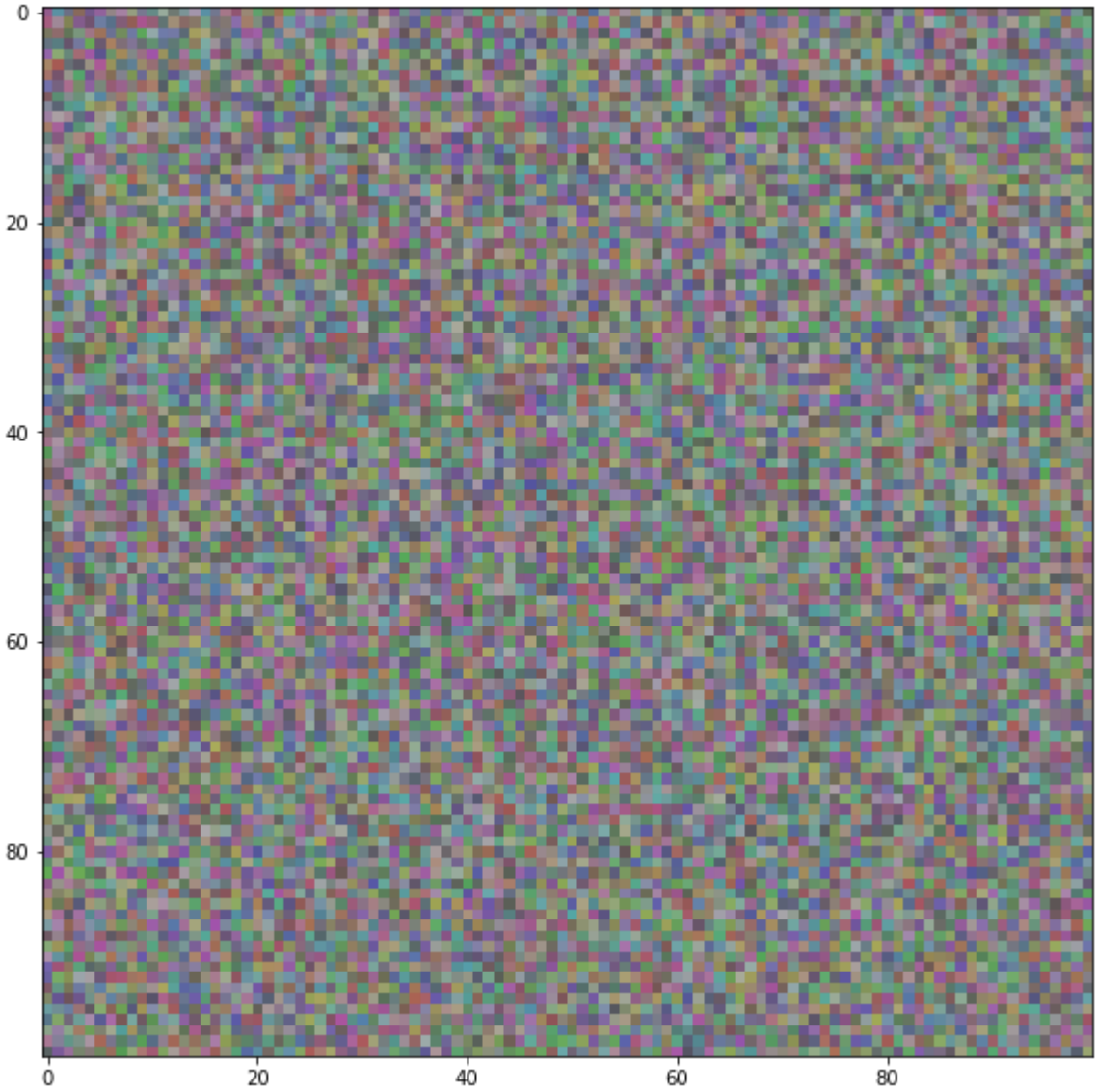

0
20
40
60
80
0
20
40
60
80

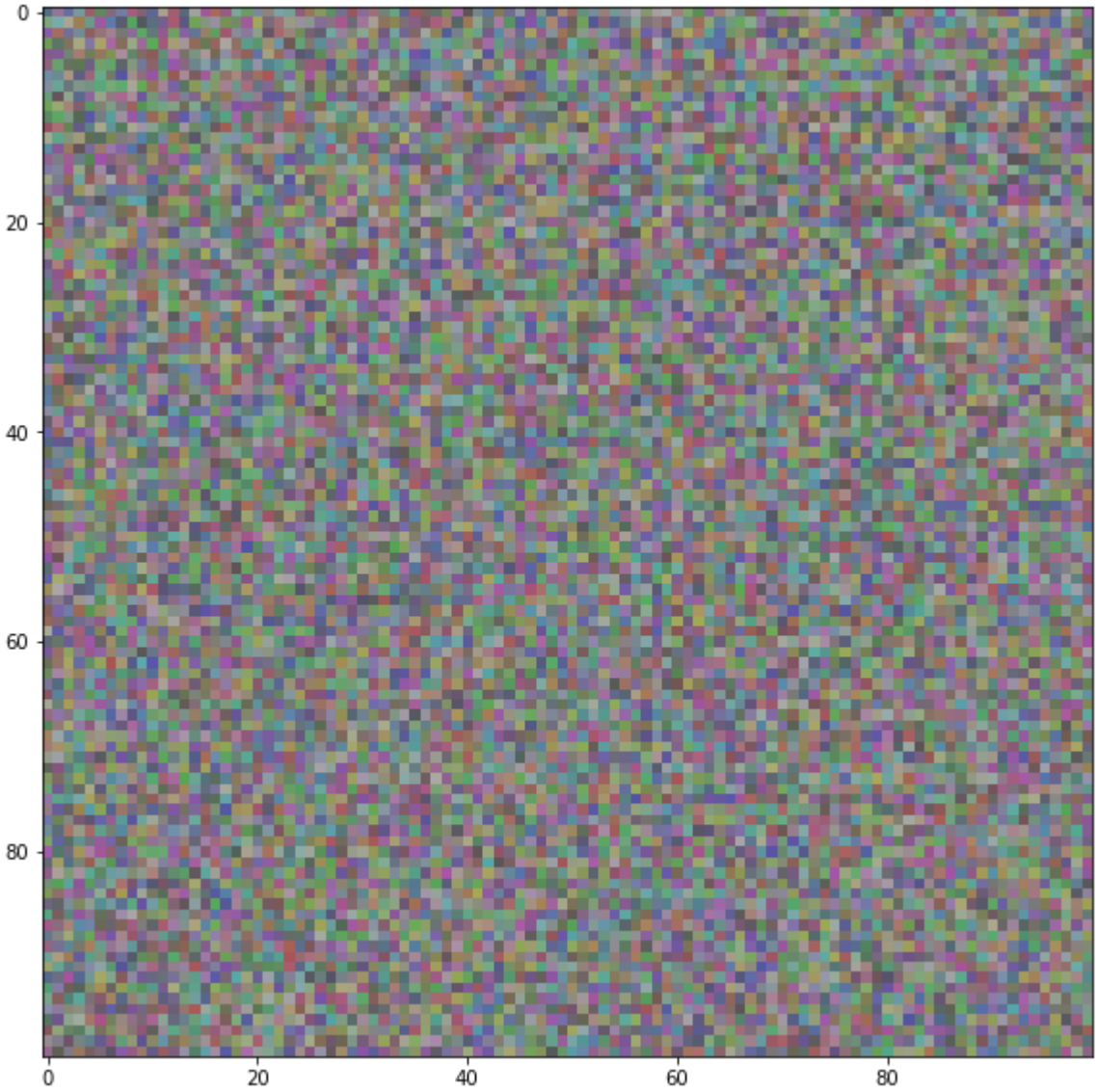

0
20
40
60
80
0
20
40
60
80

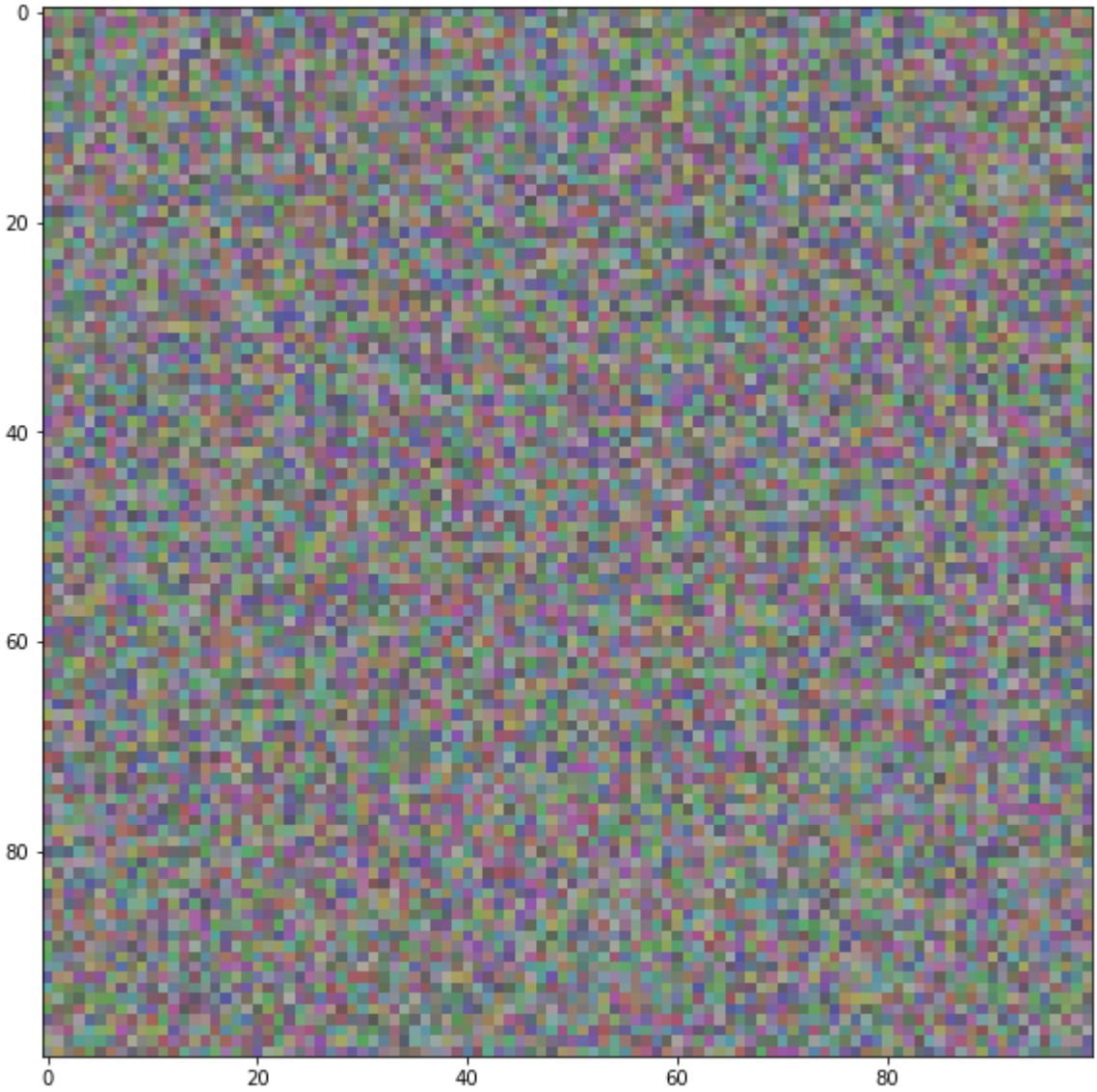

0
20
40
60
80
0
20
40
60
80

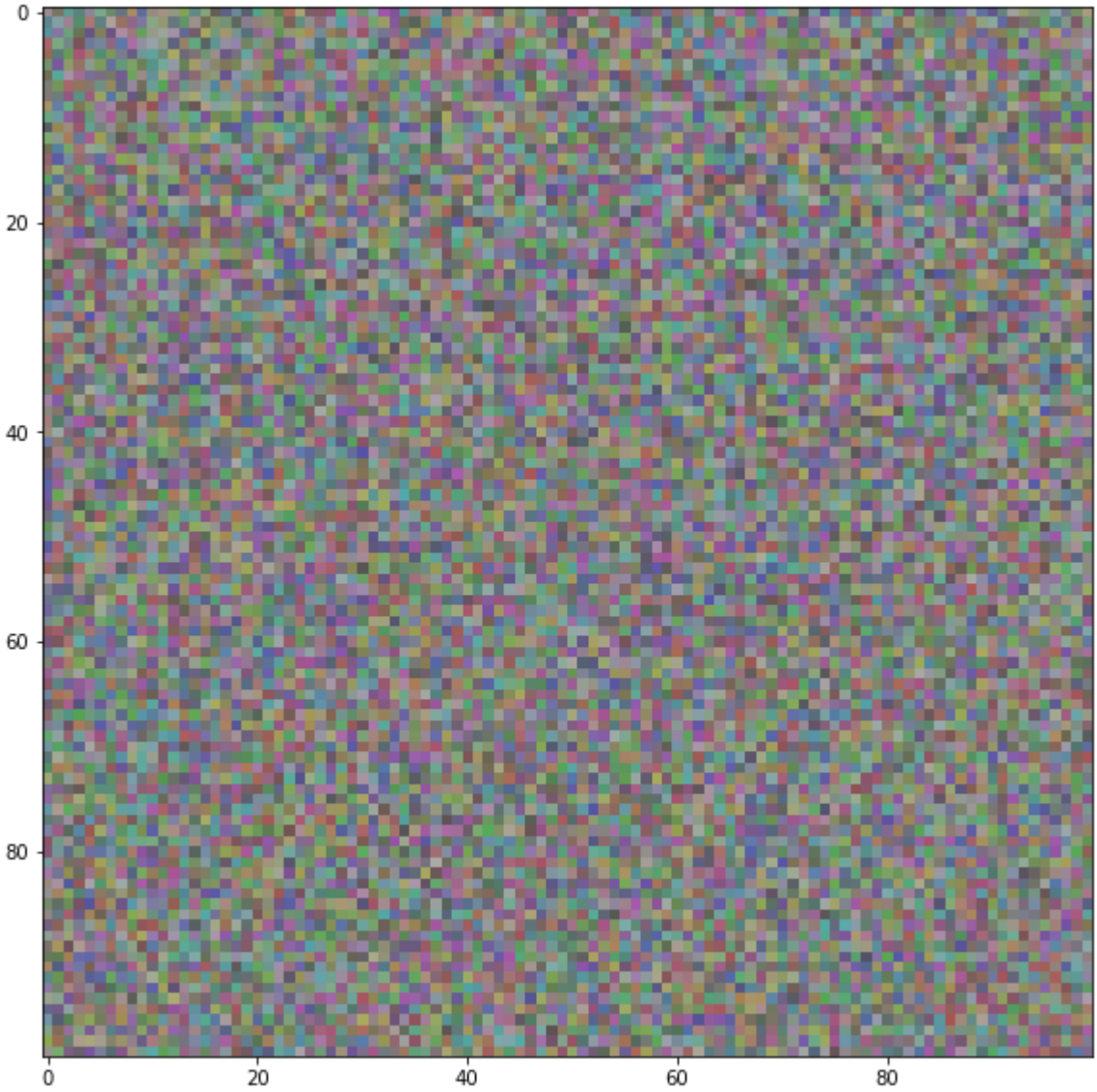

0
20
40
60
80
0
20
40
60
80

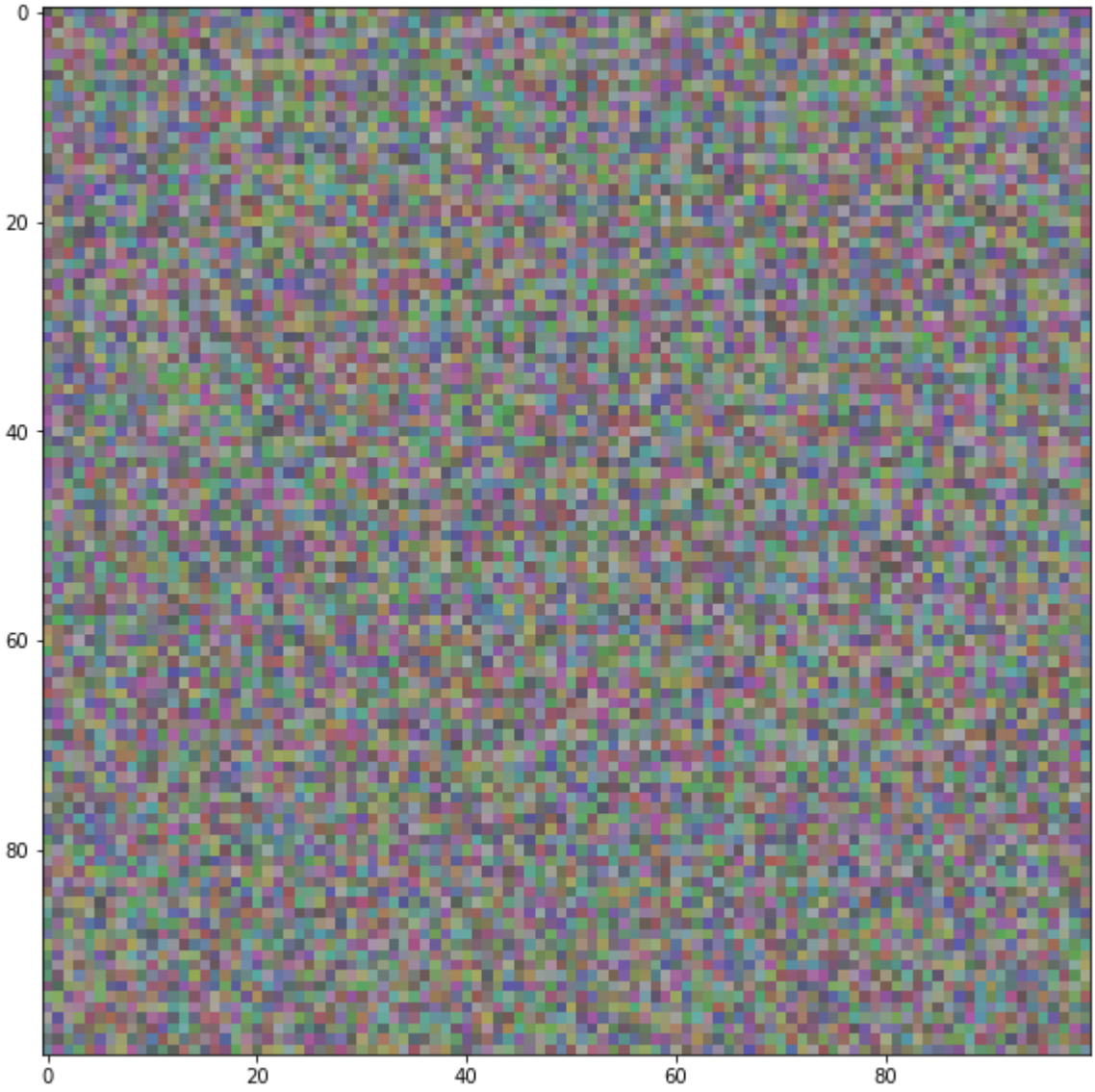

0
20
40
60
80
0
20
40
60
80

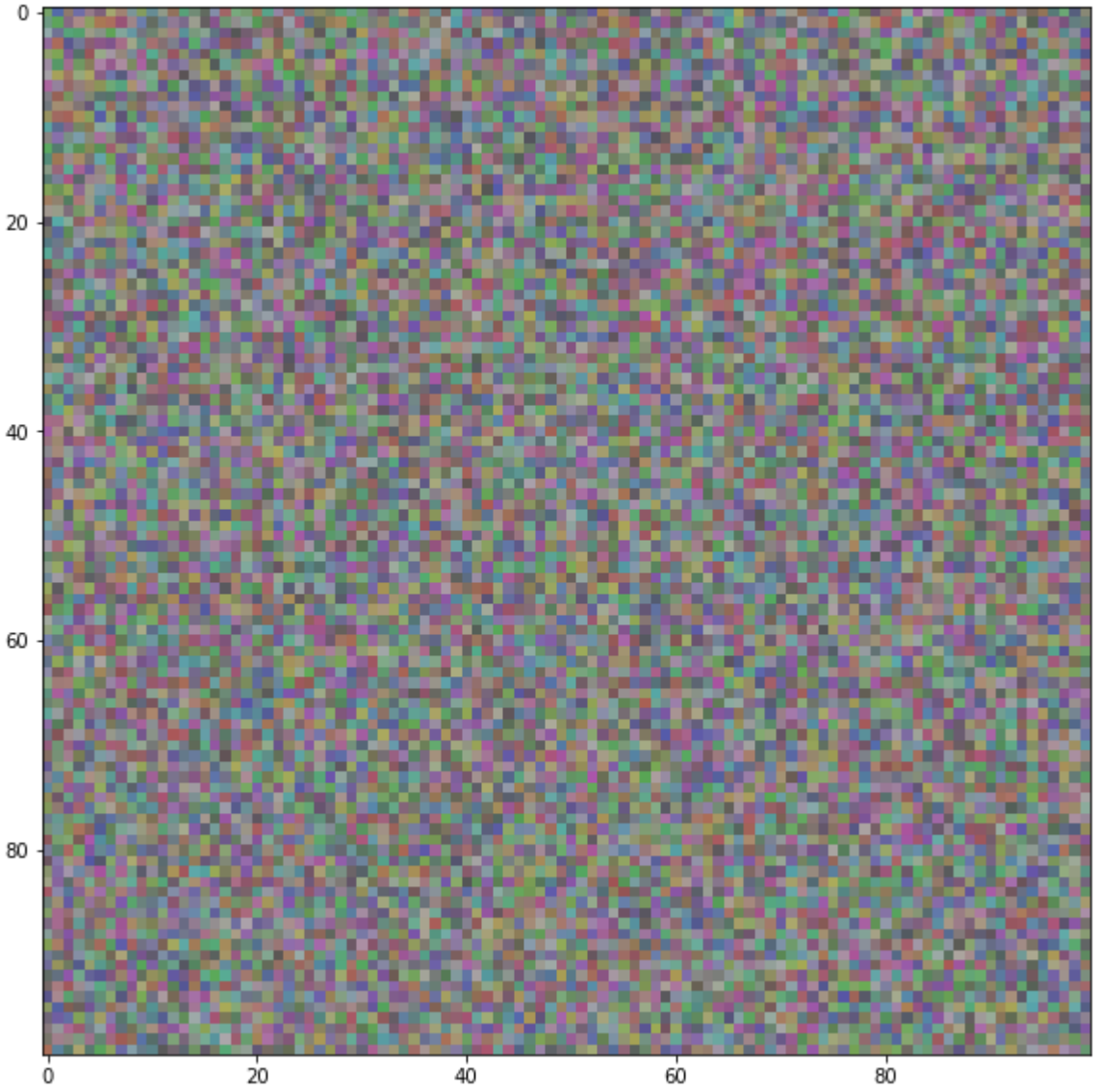

0
20
40
60
80
0
20
40
60
80

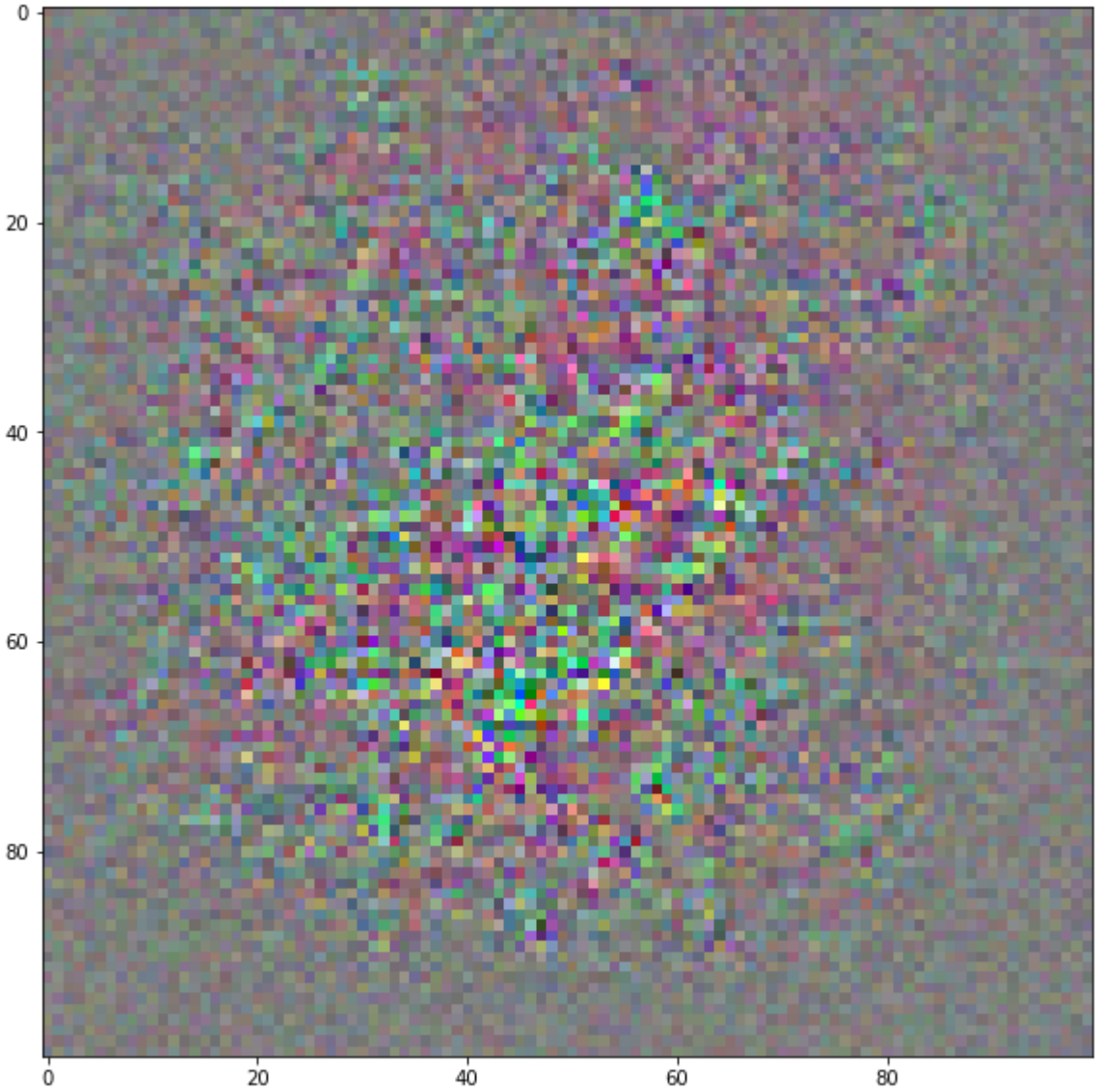

0
20
40
60
80
0
20
40
60
80

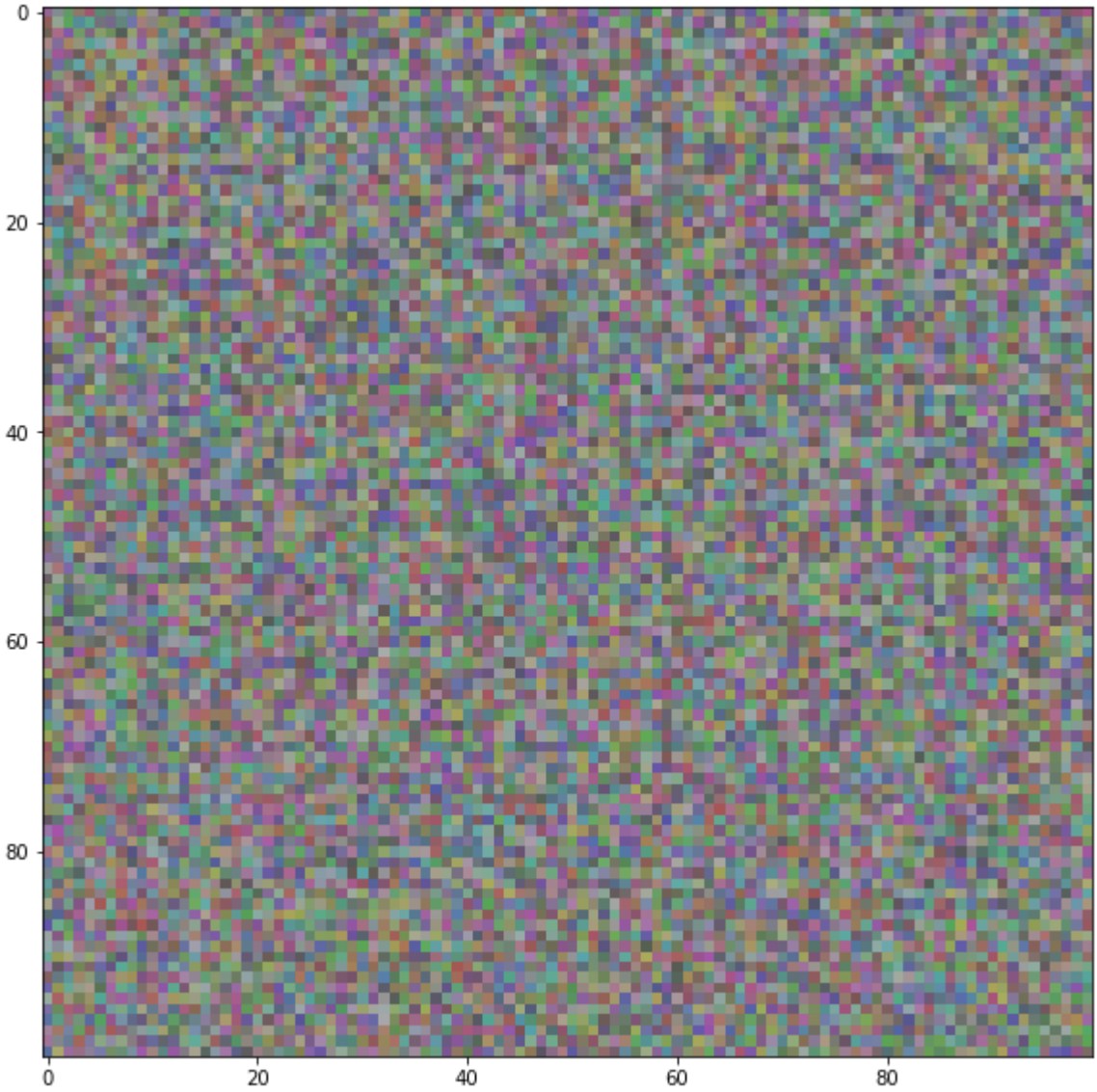

0
20
40
60
80
0
20
40
60
80

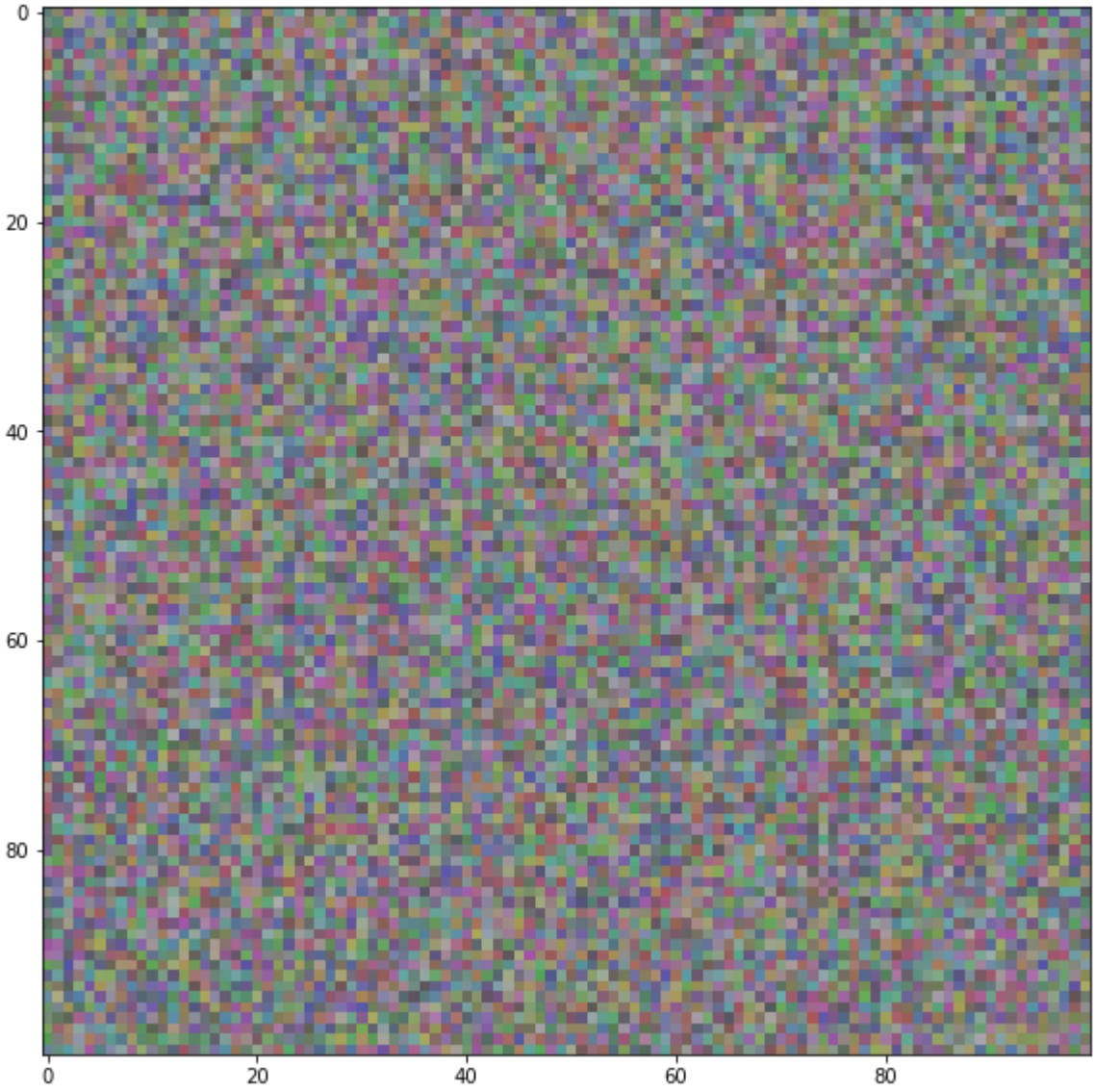

0
20
40
60
80
0
20
40
60
80

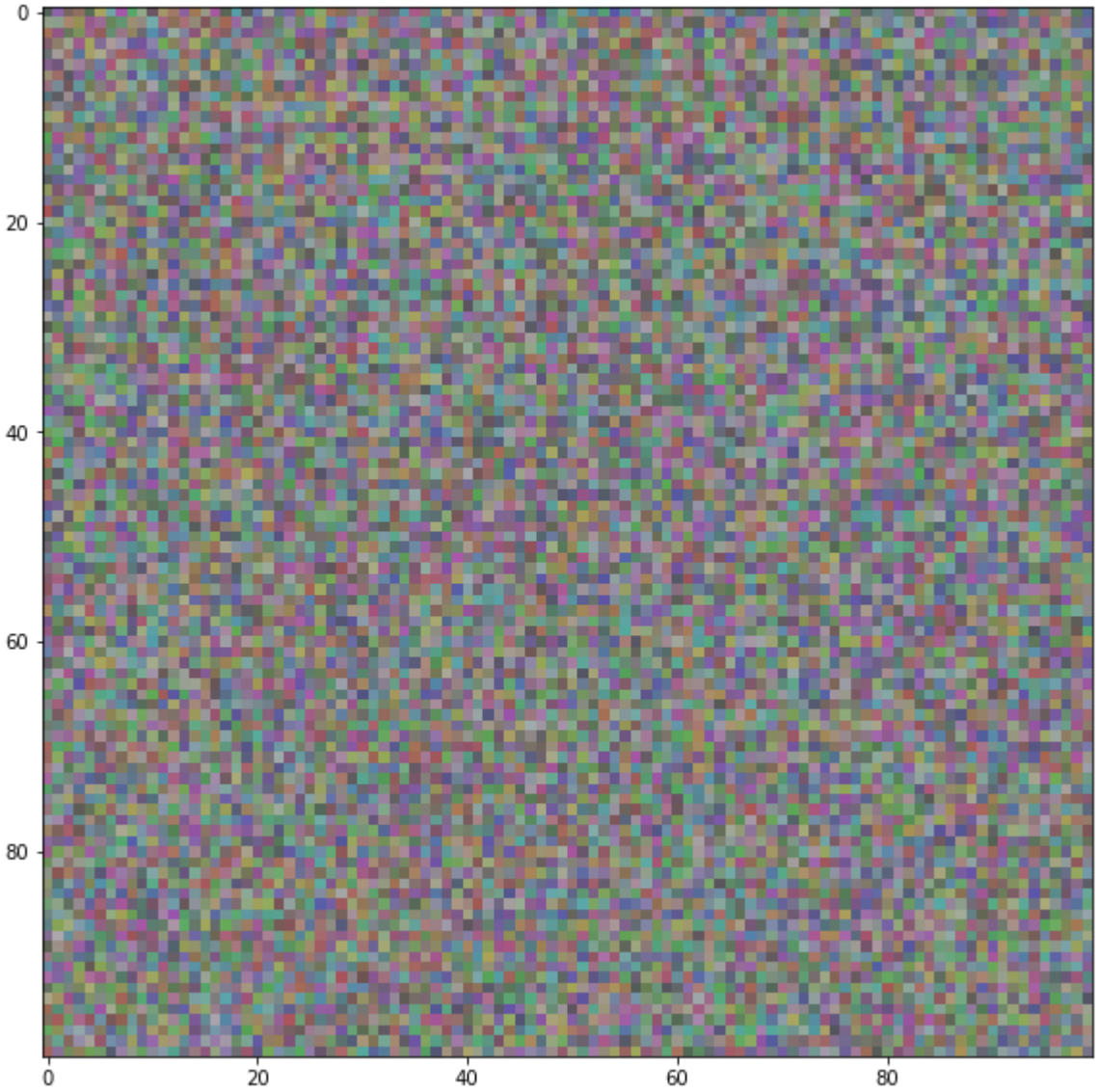

0
20
40
60
80
0
20
40
60
80


0
20
40
60
80
0
20
40
60
80


0
20
40
60
80
0
20
40
60
80


0
20
40
60
80
0
20
40
60
80


0
20
40
60
80
0
20
40
60
80

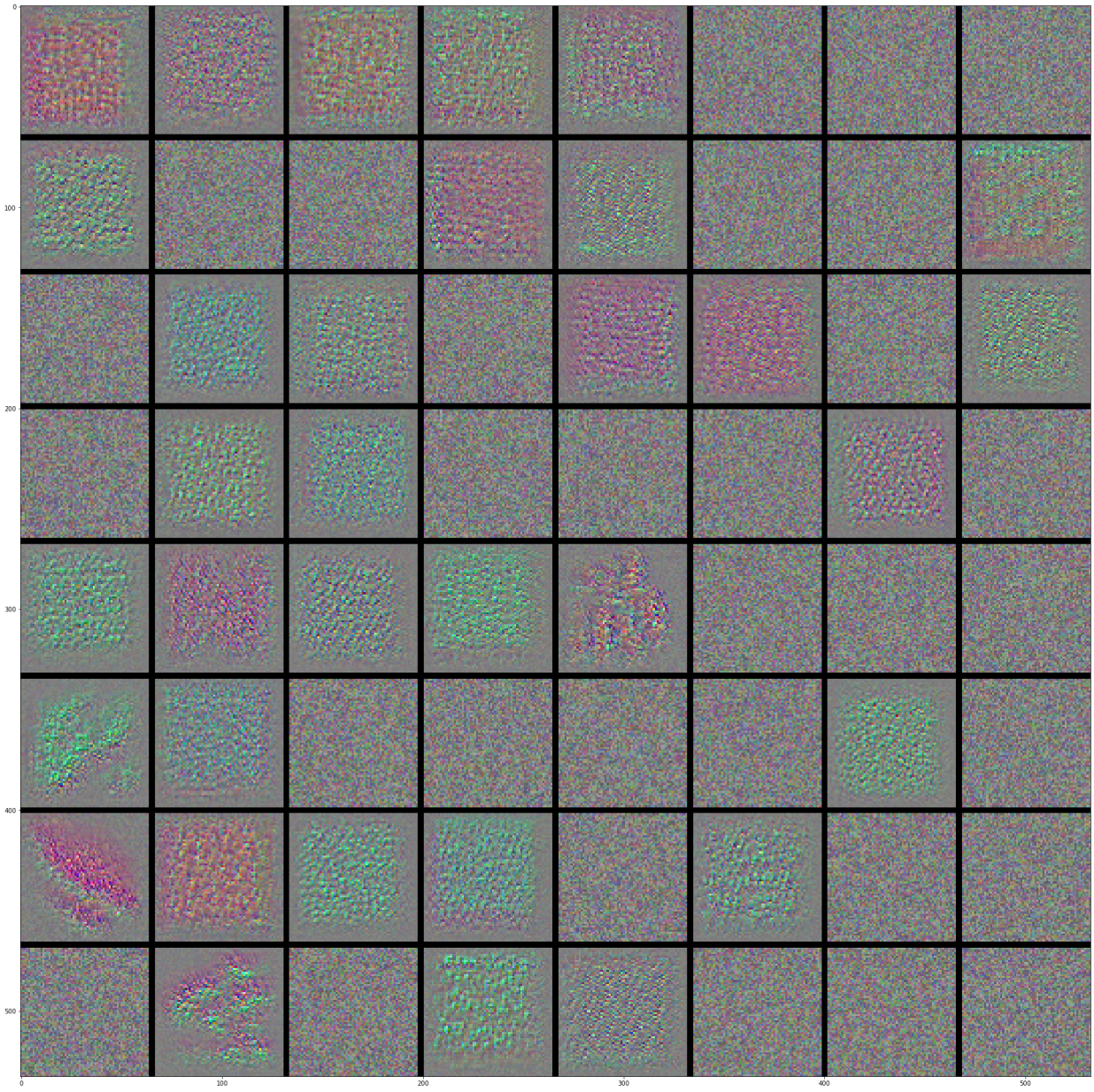


Visualization of several filters

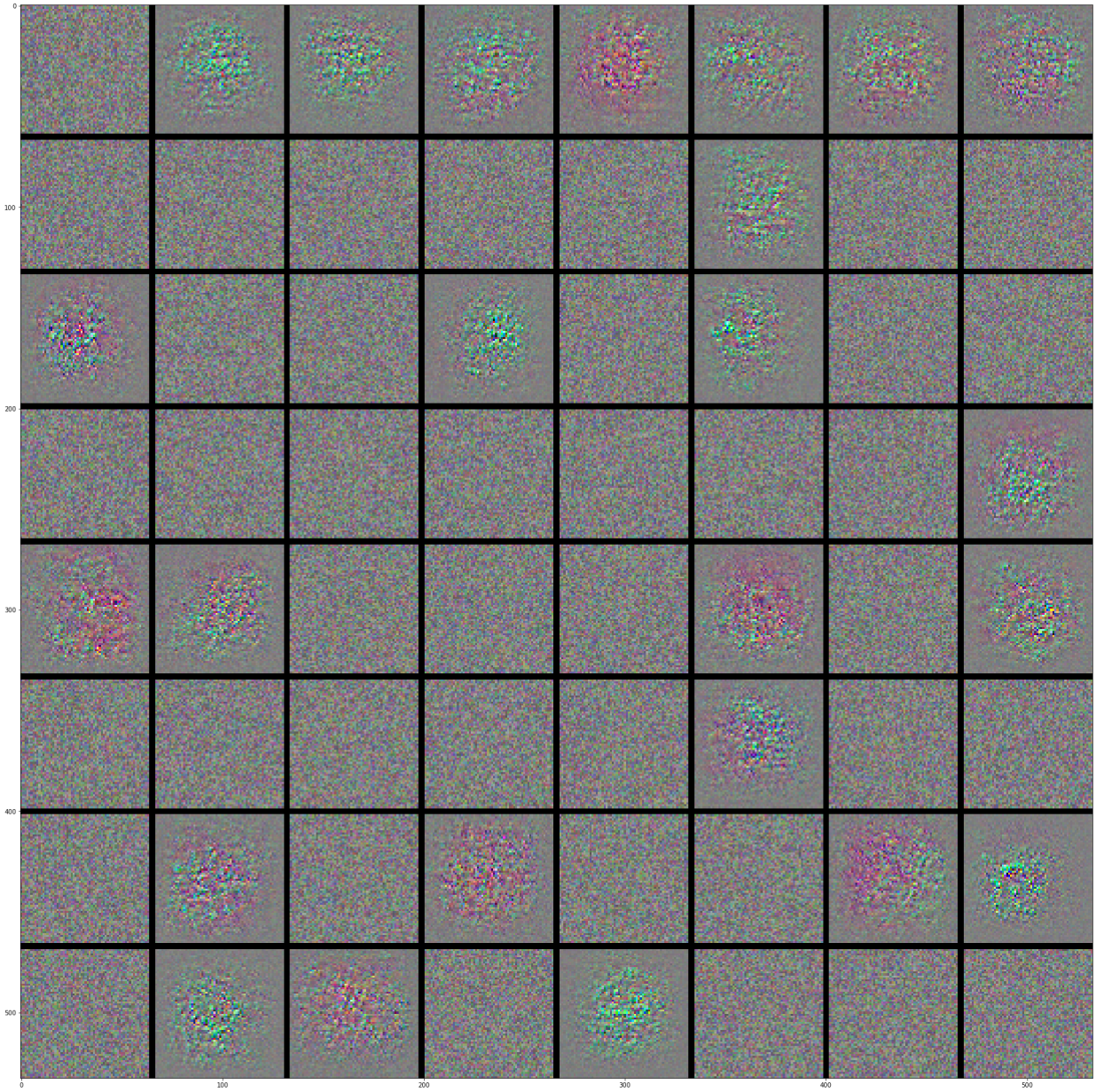
In [36]:
```
size = 64
margin = 3
no = 8

for layer_name in ['conv2d_22', 'conv2d_23',]:

    # This a empty (black) image where we will store our results.
    results = np.zeros((no * size + 7 * margin, no * size + 7 * margin, 3), dtype=np.u

    for i in range(no):  # iterate over the rows of our results grid
        for j in range(no):  # iterate over the columns of our results grid
            # Generate the pattern for filter `i + (j * no)` in `layer_name`
            filter_img = generate_pattern(layer_name, i + (j * no), size=size)
            # Put the result in the square `(i, j)` of the results grid
            horizontal_start = i * size + i * margin
            horizontal_end = horizontal_start + size
            vertical_start = j * size + j * margin
            vertical_end = vertical_start + size
            results[horizontal_start : horizontal_end, vertical_start : vertical_end,

    # Display the results grid
```

```
    print("Layer name: " + layer_name)
    plt.figure(figsize=(size / 2, size / 2))
    plt.imshow(results)
    plt.show()
```

Layer name: conv2d_22



Layer name: conv2d_23



Visualization of VGG16 filters response is much more appealing, because the network was trained on 1.2M images. However, some patterns can be also observed for this model.

Observe the effect of padding (lack of padding). Regions at the borders do not contribute. Many filters just produce noise.

TODO 5.3.1 Visualize filters in a selected layer of your model.

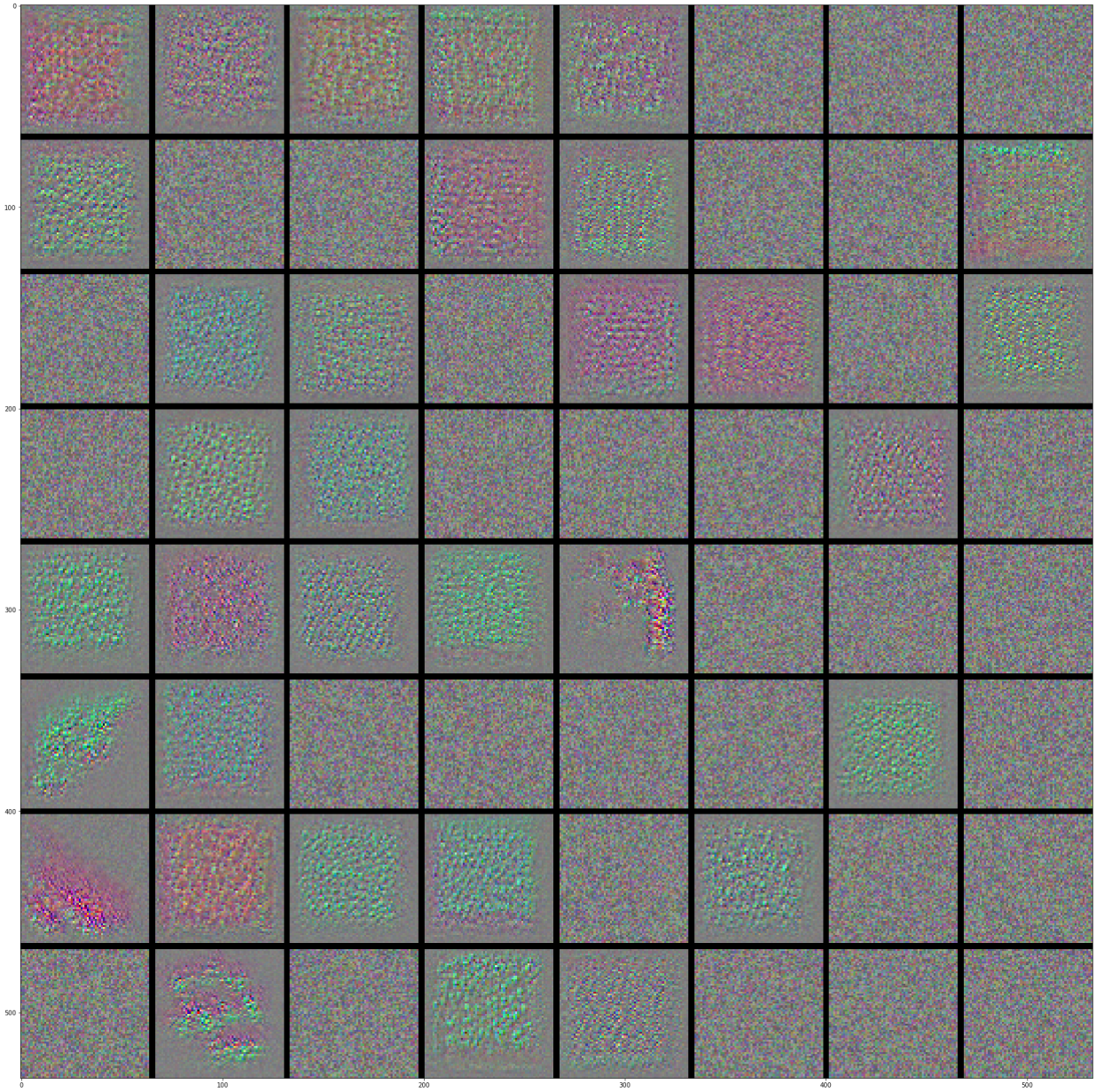
In [80]:
```
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 298, 298, 32)      896

 conv2d_1 (Conv2D)           (None, 298, 298, 32)      1056

 max_pooling2d (MaxPooling2D  (None, 149, 149, 32)     0
 )

 dropout (Dropout)           (None, 149, 149, 32)      0

 flatten (Flatten)           (None, 710432)            0

 dense (Dense)               (None, 2)                 1420866

=================================================================
Total params: 1,422,818
Trainable params: 1,422,818
Non-trainable params: 0
_________________________________________________________________
```

In [85]:

```
size = 64
margin = 3
no = 8

for layer_name in ['conv2d_22']:

    # This a empty (black) image where we will store our results.
    results = np.zeros((no * size + 7 * margin, no * size + 7 * margin, 3), dtype=np.u

    for i in range(no):  # iterate over the rows of our results grid
        for j in range(no):  # iterate over the columns of our results grid
            # Generate the pattern for filter `i + (j * no)` in `layer_name`
            filter_img = generate_pattern(layer_name, i + (j * no), size=size)
            # Put the result in the square `(i, j)` of the results grid
            horizontal_start = i * size + i * margin
            horizontal_end = horizontal_start + size
            vertical_start = j * size + j * margin
            vertical_end = vertical_start + size
            results[horizontal_start : horizontal_end, vertical_start : vertical_end,

    # Display the results grid
    print("Layer name: " + layer_name)
    plt.figure(figsize=(size / 2, size / 2))
    plt.imshow(results)
    plt.show()
```

```
Layer name: conv2d_22
```



5.4 Heatmaps

Heatmaps are used to mark parts of the image that contributed the most to the final classifier decision.

Algorithm

1. Select an image
2. Classify it
3. Select a neuron y_k corresponding to the predicted class label
4. Select z an output of a convolutional layer. Tensor z has the shape: $h \times w \times d$, where d (depth) corresponds to the number of filters
5. Compute gradients $\frac{dy_k}{dz_{i,j,k}}$

6. Compute how much each filter contributes to the output value - by calculating mean values of gradients for each filter (pooled_gradients). The vector size is equal to d.
7. Pooled gradients act as weights. Multiply z by pooled_gradients and compute means along the filter axis.

The shape of resulting heatmap matrix is $h \times w$. It can be then scaled to the size of the original image and superimposed. (Actually it works better with padding=same.)

Remark The eager mode is disabled.

5.4.1 Compute a heatmap

In [37]:
```
import tensorflow as tf
from keras import backend as K
tf.compat.v1.disable_eager_execution()
K.clear_session()
```

TODO 5.4.1 Load your own model, then use one of the previously processed images.

In [87]:
```
from keras.models import load_model

model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 298, 298, 32)      896

 conv2d_1 (Conv2D)           (None, 298, 298, 32)      1056

 max_pooling2d (MaxPooling2D  (None, 149, 149, 32)     0
 )

 dropout (Dropout)           (None, 149, 149, 32)      0

 flatten (Flatten)           (None, 710432)            0

 dense (Dense)               (None, 2)                 1420866

=================================================================
Total params: 1,422,818
Trainable params: 1,422,818
Non-trainable params: 0
_________________________________________________________________
```

In [59]:
```
#Load an image
from keras import layers
from PIL import Image
import requests
from io import BytesIO
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (10, 10)
```

```
response = requests.get('https://www.helpfulhorsehints.com/wp-content/uploads/quarter-
img = Image.open(BytesIO(response.content))
horse = np.array(img)
print(horse.shape)
X = horse
resize = layers.Resizing(300,300)
X=resize(X)

X=tf.reshape(X,(1,300,300,3))
print(X.shape)
```

```
(597, 1024, 3)
(1, 300, 300, 3)
```

Classify and find the activated neuron

In [60]:
```
probs = model.predict(X,steps=1)
print (probs)
idx = np.argmax(probs[0])
print(idx)
```

```
[[1.0000000e+00 3.1095356e-12]]
0
```

We will generate heatmap based on the last convolution layer.

In [61]:
```
from keras import backend as K
import tensorflow as tf

conv_layer = model.get_layer('conv2d_24')

# compute gradients
grads = K.gradients(model.output[:,0], conv_layer.output)[0]

print(grads.shape)
```

```
(None, 14, 14, 64)
```

Compute pooled_grads

In [62]:
```
# This is a vector of shape equal to the number of filters, where each entry
# is the mean intensity of the gradient over a specific feature map channel
pooled_grads = K.mean(grads, axis=(0, 1, 2))
print(pooled_grads.shape)
```

```
(64,)
```

In [63]:
```
# This function allows us to access the values of the quantities we just defined:
# `pooled_grads` and the output feature map of selected convolution layer`,
# given a sample image

get_pgrads_and_cloutput = K.function([model.input], [pooled_grads, conv_layer.output[0
```

Now we compute pooled gradients and layer output for the innput image - the X parameter

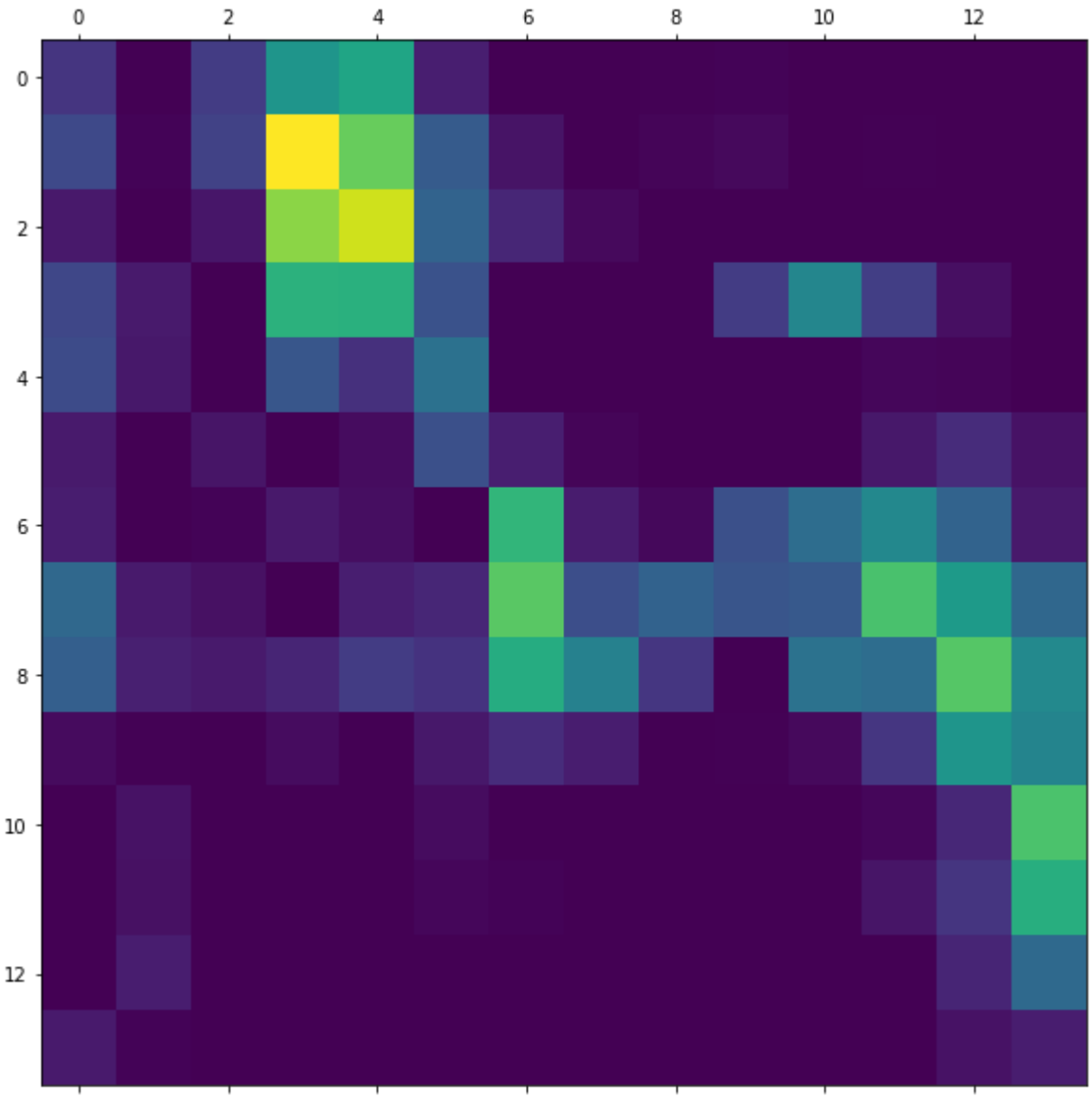
In [64]:
```
# These are the values of these two quantities, as Numpy arrays,
# given our sample image
pooled_grads_value, conv_layer_output_value = get_pgrads_and_cloutput([X])
```

In [65]:
```
# We multiply each channel in the feature map array
# by "how important this channel is" with regard to the selected class

for i in range(conv_layer_output_value.shape[2]):
    conv_layer_output_value[:, :, i] *= pooled_grads_value[i]
```

In [66]:
```
# The channel-wise mean of the resulting feature map
# is our heatmap of class activation
heatmap = np.mean(conv_layer_output_value, axis=-1)
print(heatmap.shape)
```

```
(14, 14)
```

In [67]:
```
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)
plt.show()
```



5.4.2 Packing it as a function

In [68]:
```
from keras import backend as K
import tensorflow as tf

def get_heatmap(X,layer_name):
  # Predict class label
  probs = model.predict(X,steps=1)
  idx = np.argmax(probs[0])
  conv_layer = model.get_layer(layer_name)
  # compute gradients
  grads = K.gradients(model.output[:,0], conv_layer.output)[0]
  pooled_grads = K.mean(grads, axis=(0, 1, 2))
  get_pgrads_and_cloutput = K.function([model.input], [pooled_grads, conv_layer.output

  # These are the values of these two quantities, as Numpy arrays,
  # given our sample image
  pooled_grads_value, conv_layer_output_value = get_pgrads_and_cloutput([X])
  # We multiply each channel in the feature map array
  # by "how important this channel is" with regard to the selected class

  for i in range(conv_layer_output_value.shape[2]):
      conv_layer_output_value[:, :, i] *= pooled_grads_value[i]
  # The channel-wise mean of the resulting feature map
  # is our heatmap of class activation
  heatmap = np.mean(conv_layer_output_value, axis=-1)
  heatmap = np.maximum(heatmap, 0)
  heatmap /= np.max(heatmap)
  return heatmap
```

A function to superimpose heatmap over an image region. Displays both the heatmap and the image. Uses CV2 library.

In [69]:
```
import cv2

def superimpose(X,heatmap,y_top=0,y_bottom=None,x_left=0,x_right=None,alpha=0.4):
  if y_bottom is None:
    y_bottom = X.shape[0]
  if x_right is None:
    x_right = X.shape[1]

  w = x_right - x_left
  h = y_bottom - y_top
  heatmap = cv2.resize(heatmap, (w, h))
  heatmap = np.uint8(255 * heatmap)
  heatmap = np.clip(heatmap,0,255)

  # We apply the heatmap to the original image
  heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_HOT)
  plt.matshow(heatmap)
  plt.show()

  X2=np.array(X)
  X2[y_top:y_bottom,x_left:x_right] = heatmap * alpha + X2[y_top:y_bottom,x_left:x_rig
  plt.imshow(X2)
  plt.show()
```

TODO 5.4.1 Display heatmaps of all convolutional layers using your model and the selected image.

In [70]:
```
heatmap = get_heatmap(X,'conv2d_24')
#30:900,200:900
superimpose(horse,heatmap,alpha=.6)
```

Disclaimer Due to the padding and maxpooling, regions probably should be a little bit corrected.

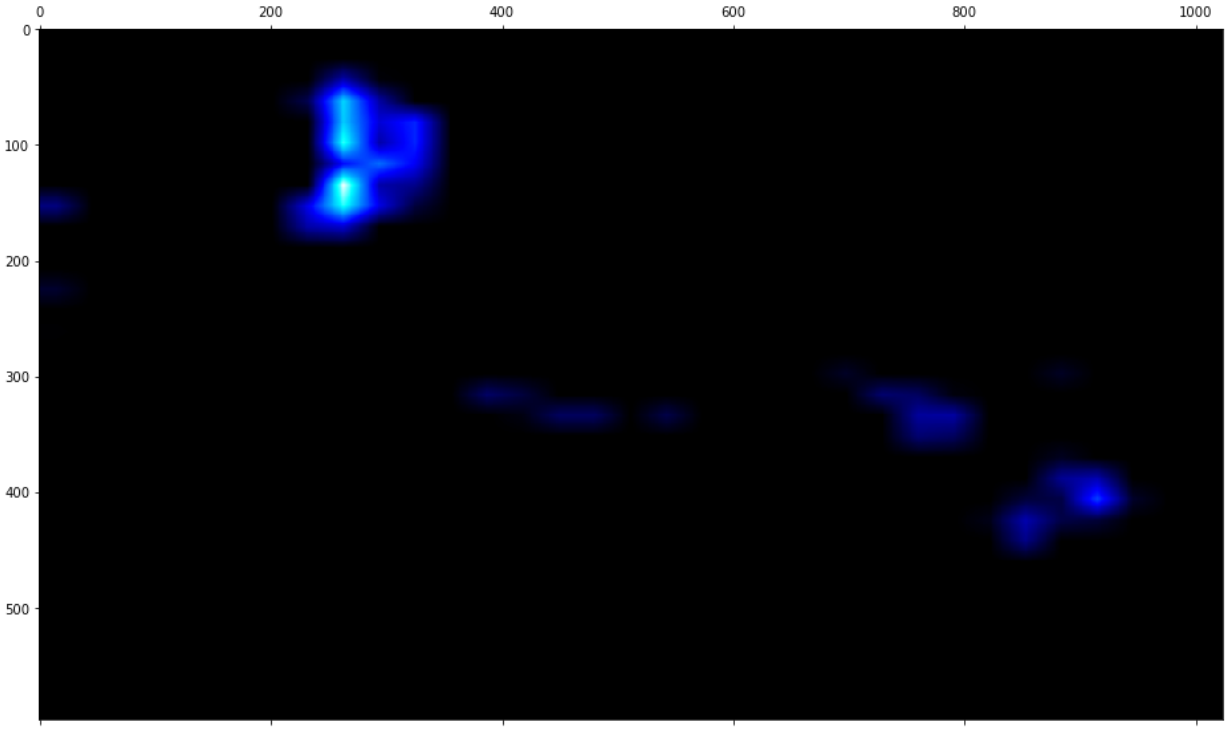
In [71]:
```
(300-14*2*2*2*2)/2
```

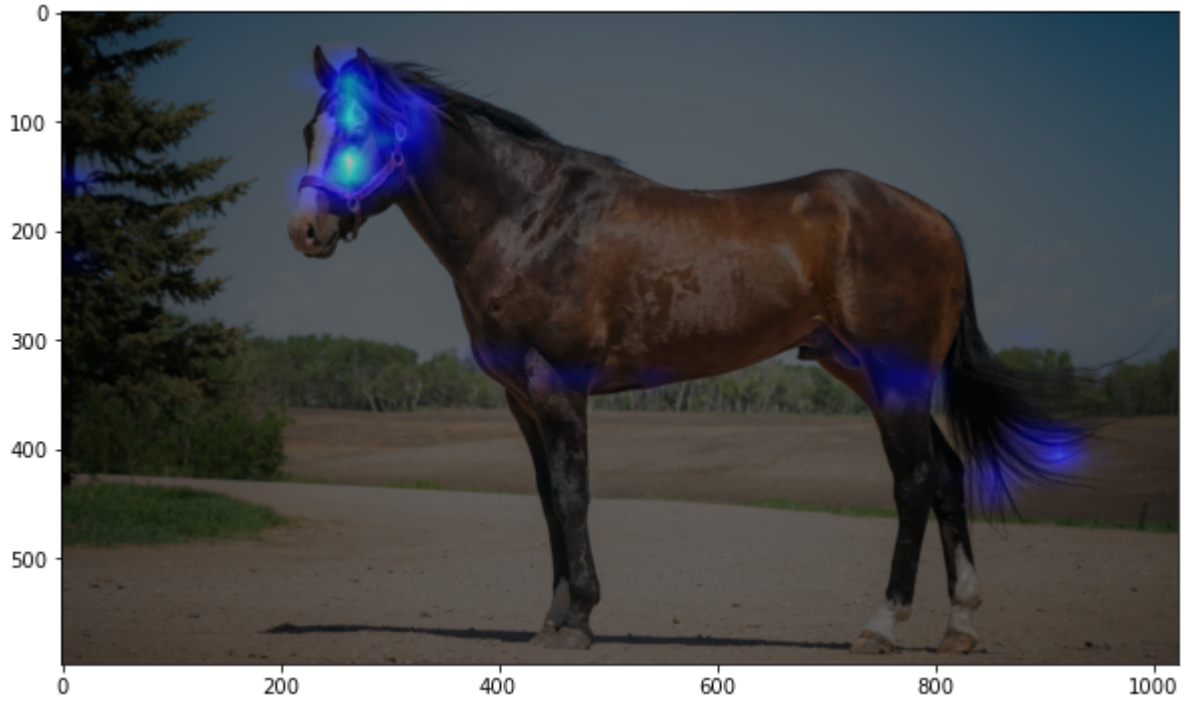
Out[71]: 38.0

In [78]:
```
heatmap = get_heatmap(X,'conv2d_24')
superimpose(horse,heatmap,alpha=.6)
```

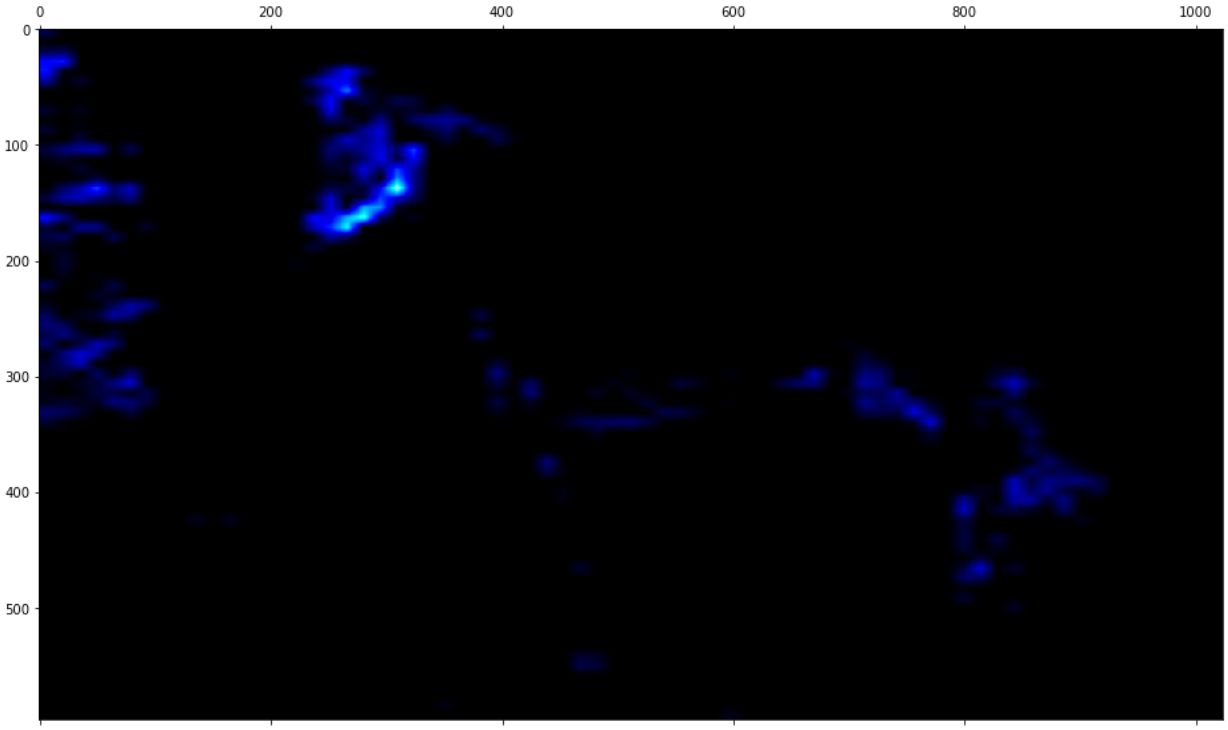
This legs position is not quite human.

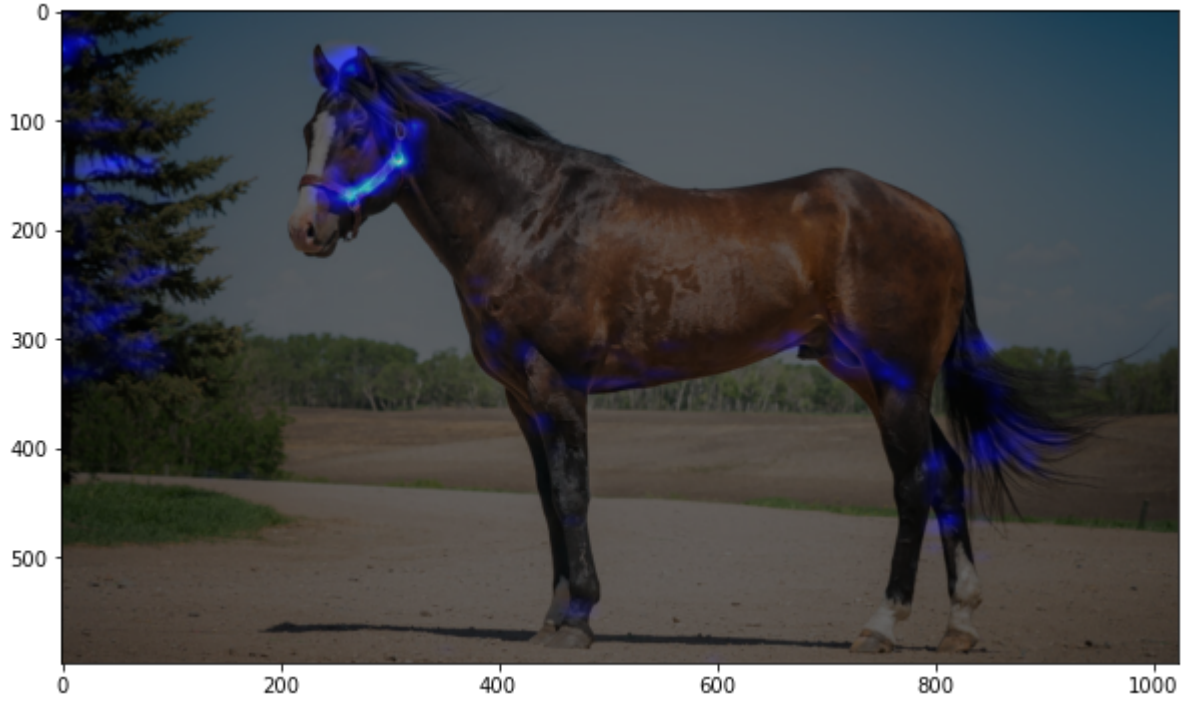
In [73]:
```
heatmap = get_heatmap(X,'conv2d_23')
superimpose(horse,heatmap,alpha=.6)
```



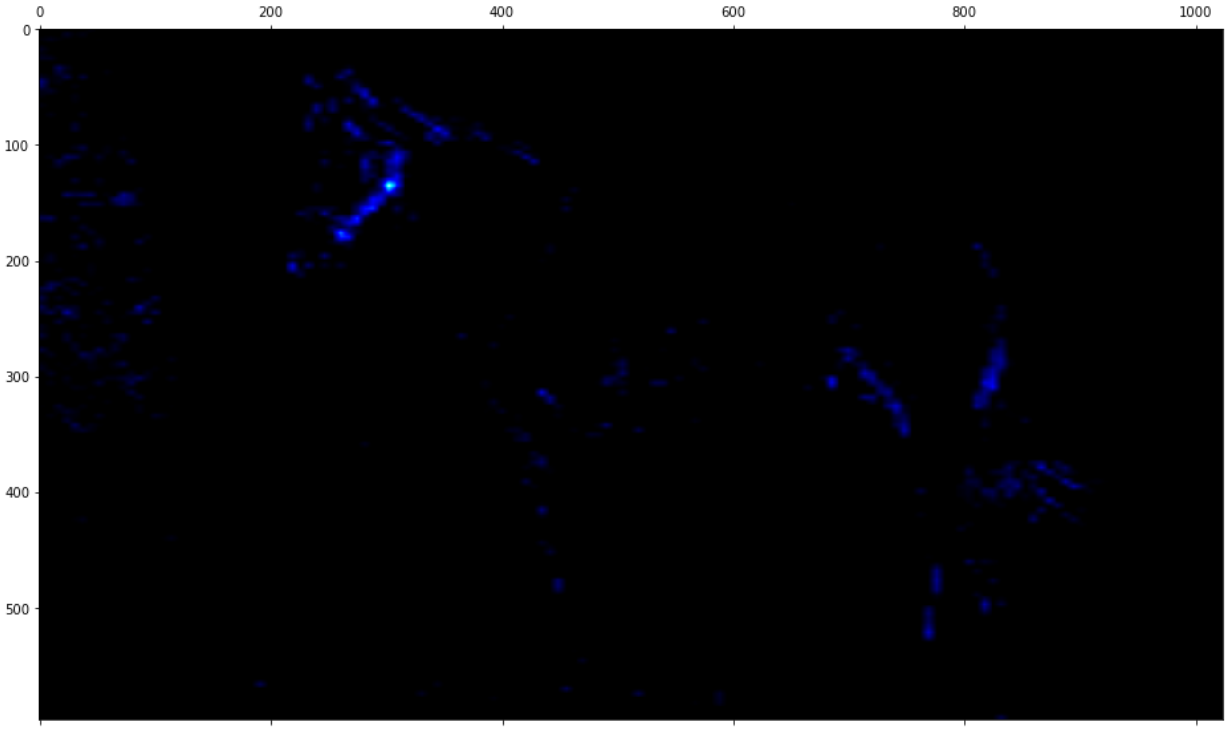


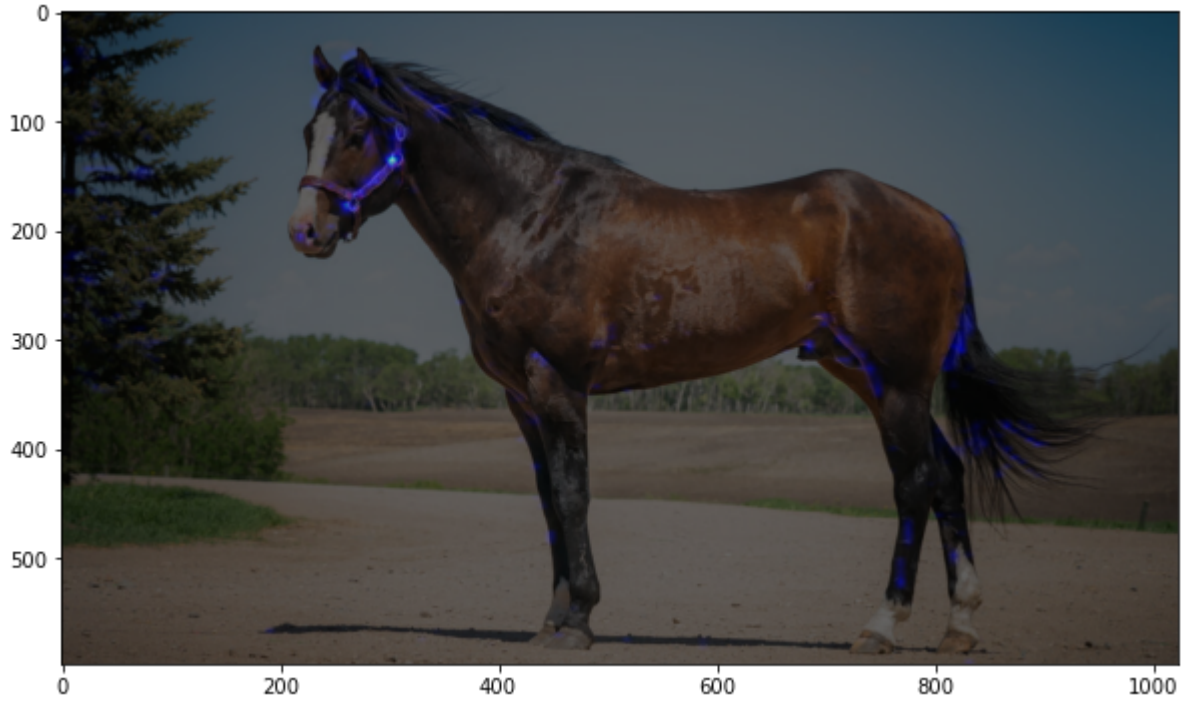
In [74]:
```
heatmap = get_heatmap(X,'conv2d_22')
superimpose(horse,heatmap,alpha=.6)
```



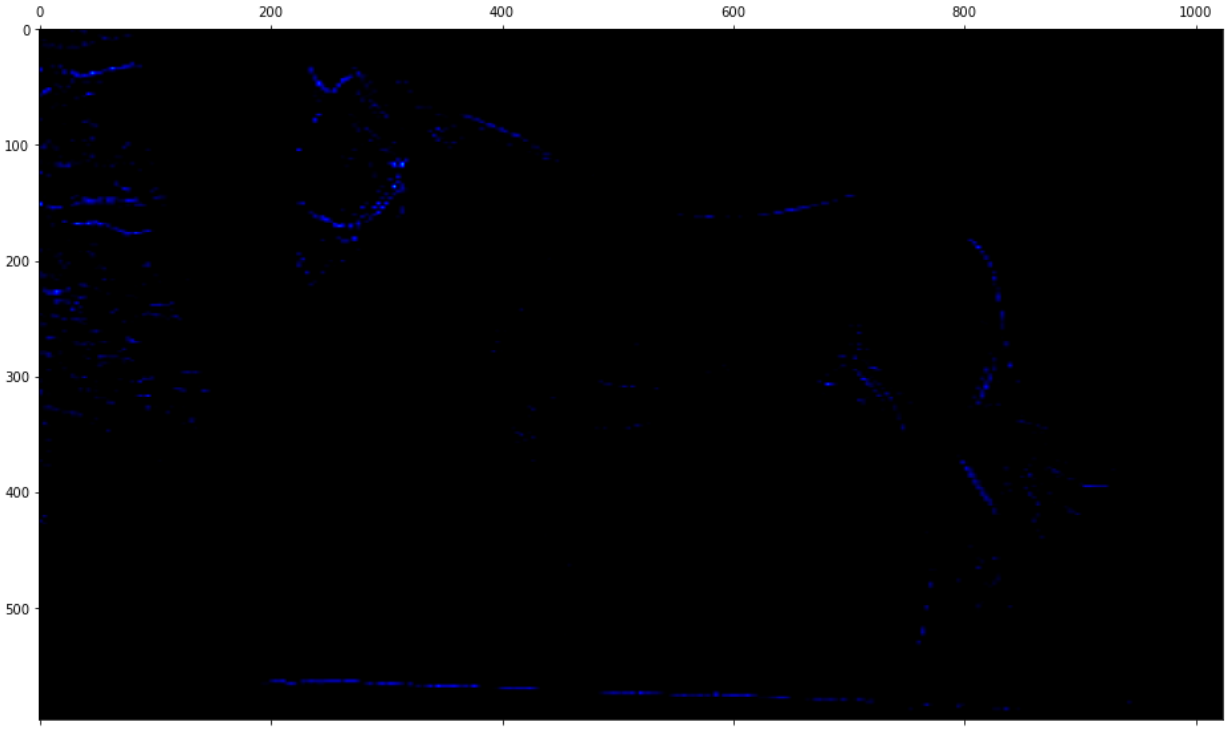


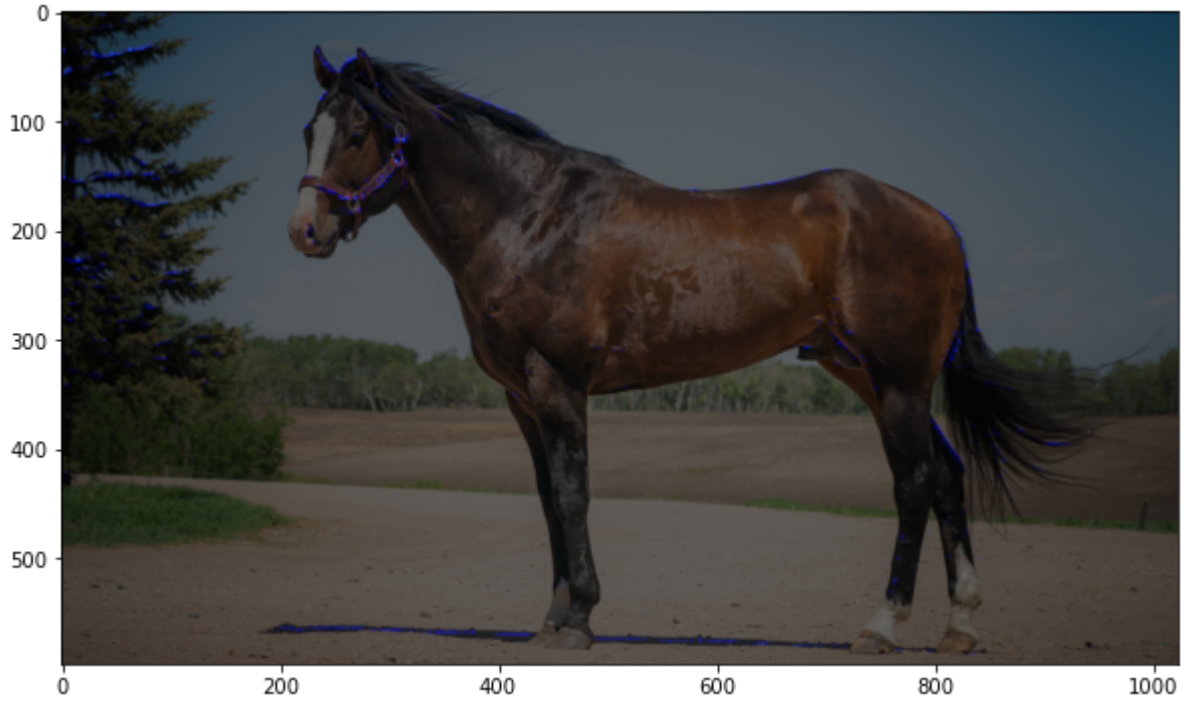
In [75]:
```
heatmap = get_heatmap(X,'conv2d_21')
superimpose(horse,heatmap,alpha=.6)
```





In [76]:
```
heatmap = get_heatmap(X,'conv2d_20')
superimpose(horse,heatmap,alpha=.6)
```


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


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