In [2]:
```
# Jessica Gallo & Asslidin Asliev
# CSC 732 Pattern Recognition & Neural Networks
# Final Project
# Part 1
# Image Classification with Convolutional Neural Networks
# Problem 1

# Created: 4/22/2020
# Last Modified: 4/27/2020
```

In [24]:
```
# --------
# IMPORTS |
# --------

# A
import os, shutil  # DOWNLOADING & SPLITTING DATA
# B
from keras import layers  # INSTANTIATING A SMALL CONVNET
from keras import models  # INSTANTIATING A SMALL CONVNET
from keras import optimizers  # CONFIGURING THE MODEL FOR TRAINING
from keras.preprocessing.image import ImageDataGenerator  # USING IMAGEDATAGENERATOR TO READ IMA
import matplotlib.pyplot as plt  # DISPLAYING CURVES OF LOSS & ACCURACY DURING TRAINING
# C
from keras.preprocessing import image  # DISPLAYING SOME RANDOMLY AUGMENTED TRAINGING IMAGES
from keras.models import load_model  # VISUALIZING INTERMEDIATE ACTIVATIONS
from keras.preprocessing import image  # preprocessing a single image
import numpy as np  # preprocessing a single image
import matplotlib.pyplot as plt  # displaying the test picture
from keras import models  # instantiating a model from an input tensor and a lsit of output tens
import matplotlib.pyplot as plt  # visualizing the fourth channel
from keras.applications import VGG16  # defining the loss tensor for filter visualiation
from keras import backend as K  # defining the loss tensor for filter visualization
import numpy as np  # fetching numpy output values given numpy input values
```

In [4]:
```
# ========================================================================
# A                                                                      |
# Download the original dataset and create a new one containing 3 subsets:  |
# a training set with 1400 smaples of each class, a vlaidation set with     |
# 400 samples of each class, and a test set with 200 samples for each class |
# (70%, 20%, 10%)                                                        |
# ========================================================================
```

In [16]:

```
# --------------------------------
# DOWNLOADING & SPLITTING THE DATA |
# --------------------------------

# Path to original dataset
original_dataset_dir = './Downloads/dogs_vs_cats_original_dataset/train'

# Making a new smaller dataset
base_dir = './Downloads/dogs_vs_cats_small_dataset'
os.mkdir(base_dir)

# Making directories for training, validation & test
train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)
validation_dir = os.path.join(base_dir, 'validation')
os.mkdir(validation_dir)
test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)

# Directories with training cat and dog pictures
train_cats_dir = os.path.join(train_dir, 'cats')
os.mkdir(train_cats_dir)
train_dogs_dir = os.path.join(train_dir, 'dogs')
os.mkdir(train_dogs_dir)

# Directories with validation cat and dog pictures
validation_cats_dir = os.path.join(validation_dir, 'cats')
os.mkdir(validation_cats_dir)
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
os.mkdir(validation_dogs_dir)

# Directories with test cat and dog pictures
test_cats_dir = os.path.join(test_dir, 'cats')
os.mkdir(test_cats_dir)
test_dogs_dir = os.path.join(test_dir, 'dogs')
os.mkdir(test_dogs_dir)

# Copies the first 1,000 cat images to train_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_cats_dir, fname)
    shutil.copyfile(src, dst)

# Copies the next 500 cat images to validation_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_cats_dir, fname)
    shutil.copyfile(src, dst)

# Copies the next 500 cat images to test_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_cats_dir, fname)
    shutil.copyfile(src, dst)

# Copies the first 1,000 dog images to train_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
```

```
    dst = os.path.join(train_dogs_dir, fname)
    shutil.copyfile(src, dst)

# Copies the next 500 dog images to validation_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_dogs_dir, fname)
    shutil.copyfile(src, dst)

fnames = ['dog.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

In [17]:
```
# Check how many pictures are in each train, validation and test directories
print('Total training cat images:', len(os.listdir(train_cats_dir)))
print('Total training dog images:', len(os.listdir(train_dogs_dir)))
print('Total validation cat images:', len(os.listdir(validation_cats_dir)))
print('Total validation dog images:', len(os.listdir(validation_dogs_dir)))
print('Total test cat images:', len(os.listdir(test_cats_dir)))
print('Total test dog images:', len(os.listdir(test_dogs_dir)))
```

```
Total training cat images: 1000
Total training dog images: 1000
Total validation cat images: 500
Total validation dog images: 500
Total test cat images: 500
Total test dog images: 500
```

In [18]:
```
# ================================================================================
# B                                                                              |
# Use Sequential model and build the topology of your CNN implementing          |
# augemntation by including 4 convolutional layers Conv2D (with Relu            |
# activation), 4 MaxPooling2D layers, a Dense layer of size 1 and a sigmoid     |
# activation as the number of classes is 2. Use Adam optimizer.                 |
# ================================================================================
```

In [20]:
```
# ------------------------------
# INSTANTIATING A SMALL CONVNET |
# ------------------------------

model = models.Sequential()
# tells keras to stack all layers sequentially
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
# filter size[32]: the output dimension (i.e.  number of output filters in  convolution)
# kernel_size[3,3]: specified height and weight of 2D concolution window
# activation['relu']: activation function also called non-linearity to be out neural network
# input_shape[150,,150,3]: resized image dimentions 150x150 with 3 channel
model.add(layers.MaxPooling2D((2, 2)))
# MaxPooling: reduces spacial size of incoming features to help reduce amount
# of parameters & computation in network, helping to reduce overfitting
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))  # Sigmoid function at the end because it's jus
# Want the model to output a probability of how sure an image is a dog and not a cat, which mean
# we want a probability score where higher values means the classifer believes the image is a do
# lower values means it is a cat. The sigmoid is perfect for this because it takes in a set of n
# and returns a probability distribution in the range of 0 to 1

model.summary()
# previews the arrangement and parameter size of our convet
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

In [22]:
```
# ---------------------------------
# CONFIGURING THE MODEL FOR TRAINING |
# ---------------------------------

model.compile(loss='binary_crossentropy', optimizer=optimizers.Adam(lr=1e-4), metrics=['acc'])
# Adam optimizer with learning rate of 0.0001
# Loss['binary_crossentropy']: because its a binary classification & to specify a loss function
# Accuracy metric (acc) is a good metric to use to know if our model is doing well or not
```

In [25]:
```
# ------------------------------------------------------------
# USING THE IMAGEDATAGENERATOR TO READ IMAGES FROM DIRECTORIES |
# ------------------------------------------------------------

# rescale images by 1/255
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1.255)

train_generator = train_datagen.flow_from_directory(train_dir,  # target directory
                                                    target_size=(150, 150),
                                                    batch_size=20,
                                                    class_mode='binary')
# rotation, width_shift, height_shift, shear, zoom and horizontal_flip all are image augmentatio
# tells ImageDataGen to randomly apply some transformation to image to help augment  dataset & i

# becuase of binary_crossentropy loss, you need binary labels

validation_generator = test_datagen.flow_from_directory(validation_dir,
                                                        target_size=(150, 150),
                                                        batch_size=20,
                                                        class_mode='binary')
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

In [26]:
```
# Samples from each batch
for data_batch, labels_batch in train_generator:
    print('Data batch shape:', data_batch.shape)
    print('Labels batch shape:', labels_batch.shape)
    break
```

```
Data batch shape: (20, 150, 150, 3)
Labels batch shape: (20,)
```

In [28]:
```
# ----------------------------------------
# FITTING THE MODEL USING A BATCH GENERATOR |
# ----------------------------------------

history = model.fit_generator(train_generator,
                              steps_per_epoch=100,
                              epochs=30,
                              validation_data=validation_generator,
                              validation_steps=50)
```

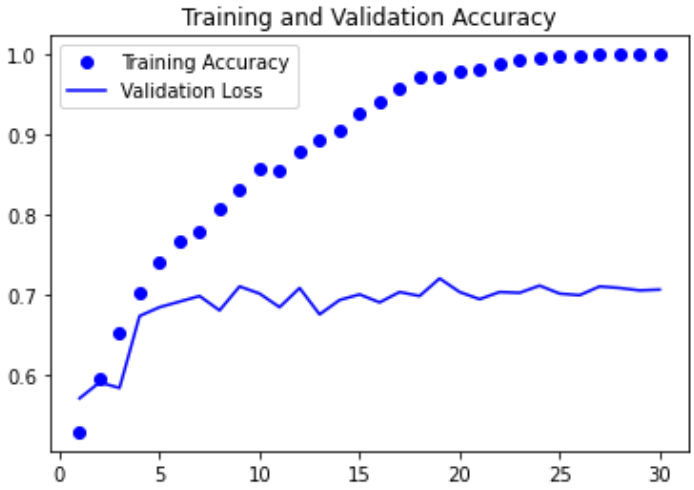
```
Epoch 1/30
100/100 [==============================] - 146s 1s/step - loss: 0.6931 - acc: 0.5280 - val_los
s: 3.3309 - val_acc: 0.5700
Epoch 2/30
100/100 [==============================] - 155s 2s/step - loss: 0.6700 - acc: 0.5945 - val_los
s: 73.7493 - val_acc: 0.5900
Epoch 3/30
100/100 [==============================] - 137s 1s/step - loss: 0.6312 - acc: 0.6510 - val_los
s: 24.2847 - val_acc: 0.5830
Epoch 4/30
100/100 [==============================] - 131s 1s/step - loss: 0.5686 - acc: 0.7030 - val_los
s: 29.4228 - val_acc: 0.6730
Epoch 5/30
100/100 [==============================] - 123s 1s/step - loss: 0.5168 - acc: 0.7405 - val_los
s: 118.1477 - val_acc: 0.6840
Epoch 6/30
100/100 [==============================] - 122s 1s/step - loss: 0.4926 - acc: 0.7675 - val_los
s: 68.6237 - val_acc: 0.6910
Epoch 7/30
100/100 [==============================] - 131s 1s/step - loss: 0.4549 - acc: 0.7790 - val_los
s: 25.4330 - val_acc: 0.6980
Epoch 8/30
100/100 [==============================] - 127s 1s/step - loss: 0.4292 - acc: 0.8075 - val_los
s: 27.7126 - val_acc: 0.6800
Epoch 9/30
100/100 [==============================] - 134s 1s/step - loss: 0.3883 - acc: 0.8320 - val_los
s: 39.7610 - val_acc: 0.7100
Epoch 10/30
100/100 [==============================] - 135s 1s/step - loss: 0.3466 - acc: 0.8565 - val_los
s: 66.9264 - val_acc: 0.7010
Epoch 11/30
100/100 [==============================] - 135s 1s/step - loss: 0.3371 - acc: 0.8555 - val_los
s: 38.9676 - val_acc: 0.6840
Epoch 12/30
100/100 [==============================] - 132s 1s/step - loss: 0.2917 - acc: 0.8785 - val_los
s: 40.1939 - val_acc: 0.7080
Epoch 13/30
100/100 [==============================] - 131s 1s/step - loss: 0.2665 - acc: 0.8930 - val_los
s: 124.3160 - val_acc: 0.6750
Epoch 14/30
100/100 [==============================] - 134s 1s/step - loss: 0.2409 - acc: 0.9040 - val_los
s: 159.5469 - val_acc: 0.6930
Epoch 15/30
100/100 [==============================] - 144s 1s/step - loss: 0.1995 - acc: 0.9270 - val_los
s: 20.1422 - val_acc: 0.7000
Epoch 16/30
100/100 [==============================] - 136s 1s/step - loss: 0.1764 - acc: 0.9410 - val_los
s: 83.0314 - val_acc: 0.6900
Epoch 17/30
100/100 [==============================] - 127s 1s/step - loss: 0.1416 - acc: 0.9580 - val_los
s: 220.4671 - val_acc: 0.7030
Epoch 18/30
```

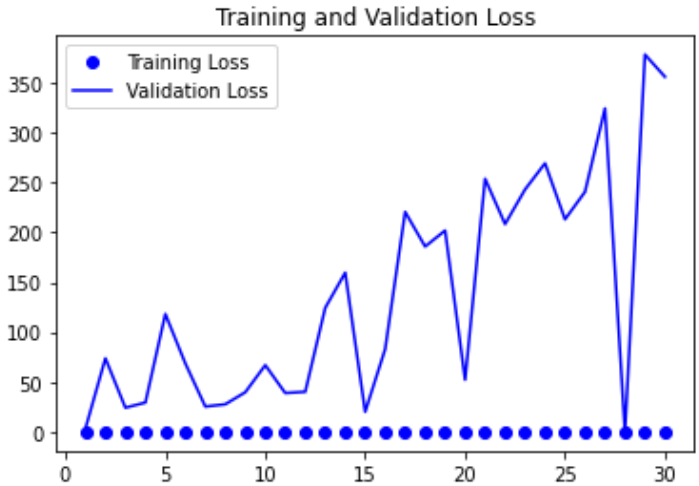
```
100/100 [==============================] - 122s 1s/step - loss: 0.1094 - acc: 0.9720 - val_los
s: 185.7634 - val_acc: 0.6980
Epoch 19/30
100/100 [==============================] - 127s 1s/step - loss: 0.1047 - acc: 0.9715 - val_los
s: 201.8043 - val_acc: 0.7200
Epoch 20/30
100/100 [==============================] - 111s 1s/step - loss: 0.0834 - acc: 0.9790 - val_los
s: 52.2085 - val_acc: 0.7030
Epoch 21/30
100/100 [==============================] - 105s 1s/step - loss: 0.0710 - acc: 0.9825 - val_los
s: 253.6445 - val_acc: 0.6940
Epoch 22/30
100/100 [==============================] - 103s 1s/step - loss: 0.0585 - acc: 0.9885 - val_los
s: 208.2495 - val_acc: 0.7030
Epoch 23/30
100/100 [==============================] - 106s 1s/step - loss: 0.0393 - acc: 0.9935 - val_los
s: 242.9491 - val_acc: 0.7020
Epoch 24/30
100/100 [==============================] - 104s 1s/step - loss: 0.0296 - acc: 0.9965 - val_los
s: 269.1779 - val_acc: 0.7110
Epoch 25/30
100/100 [==============================] - 103s 1s/step - loss: 0.0217 - acc: 0.9990 - val_los
s: 213.0229 - val_acc: 0.7010
Epoch 26/30
100/100 [==============================] - 103s 1s/step - loss: 0.0164 - acc: 0.9990 - val_los
s: 240.5958 - val_acc: 0.6990
Epoch 27/30
100/100 [==============================] - 105s 1s/step - loss: 0.0111 - acc: 1.0000 - val_los
s: 324.1751 - val_acc: 0.7100
Epoch 28/30
100/100 [==============================] - 104s 1s/step - loss: 0.0093 - acc: 1.0000 - val_los
s: 0.0000e+00 - val_acc: 0.7080
Epoch 29/30
100/100 [==============================] - 102s 1s/step - loss: 0.0063 - acc: 1.0000 - val_los
s: 378.1235 - val_acc: 0.7050
Epoch 30/30
100/100 [==============================] - 103s 1s/step - loss: 0.0053 - acc: 1.0000 - val_los
s: 355.8469 - val_acc: 0.7060
```

In [29]:

```
# ---------------
# SAVE THE MODEL |
# ---------------
model.save('cats_and_dogs_small_1.h5')
```

In [30]:
```
# ------------------------------------------------------
# DISPLAYING CURVES OF LOSS & ACCURACY DURING TRAINING |
# ------------------------------------------------------

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.plot(epochs, acc, 'bo', label='Training Accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation Loss')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()

plt.show()

# Plots show overfitting
```



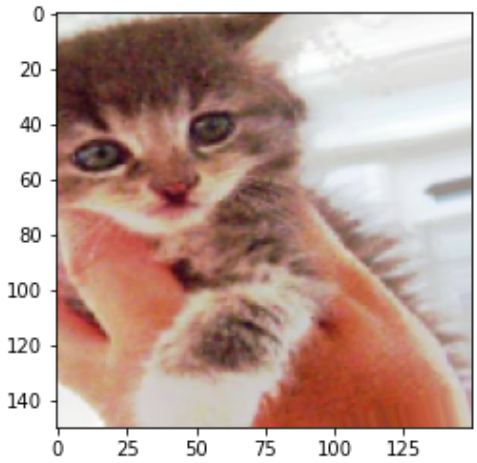


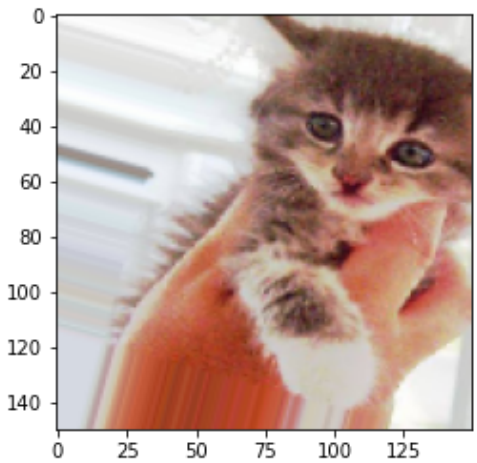
In [31]:
```
# ========================================================================
# C                                                                      |
# Display randomly augmented images(four images of two different cats and   |
# four images of two different dogs)                                     |
# ========================================================================
```

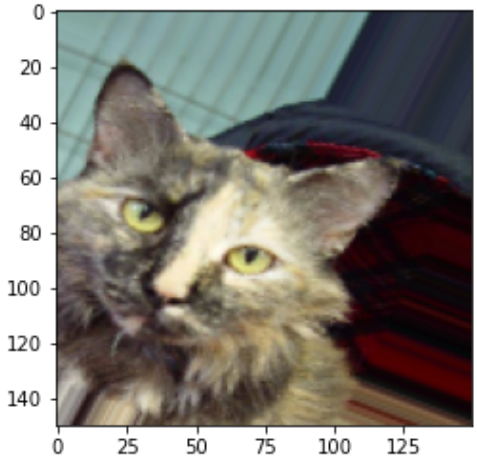
In [34]:
```
# ----------------------------------------------------------------------
# SETTING UP A DATA AUGMENTATION CONFIGURATION VIA IMAGEDATAGENERATOR |
# ----------------------------------------------------------------------

datagen = ImageDataGenerator(rotation_range=40,
                             width_shift_range=0.2,
                             height_shift_range=0.2,
                             shear_range=0.2,
                             zoom_range=0.2,
                             horizontal_flip=True,
                             fill_mode='nearest')
```

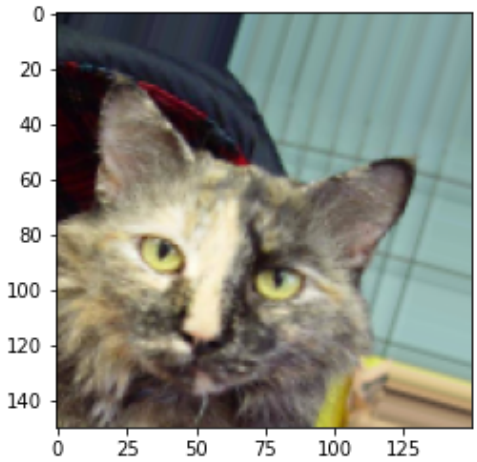
In [70]:
```
# ----------------------------------------------------
# DISPLAYING SOME RANDOMLY AUGMENTED TRAINGING IMAGES |
# ----------------------------------------------------

# 1st CAT
# ~~~~~~~
fnames = [os.path.join(train_cats_dir, fname) for fname in os.listdir(train_cats_dir)]
img_path = fnames[3]  # chooses one image augment
img = image.load_img(img_path, target_size=(150, 150))  # reads the image & resizes it

x = image.img_to_array(img)  # converts it to a numpy array with shape(150, 150, 3)

x = x.reshape((1,) + x.shape)  # reshapes it to (1, 150, 150, 3)

# generates batches of randomly transformed images
# loops indefinitely, so you need to break the loop at some point]
i = 0
for batch in datagen.flow(x, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 2 == 0:
        break

plt.show()

# 2nd CAT
# ~~~~~~~

img_path2 = fnames[2]
img2 = image.load_img(img_path2, target_size=(150, 150))
x2 = image.img_to_array(img2)
x2 = x2.reshape((1,) + x2.shape)

i = 0
for batch in datagen.flow(x2, batch_size=1):
    plt.figure(i)
    imgplot2 = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 2 == 0:
        break

plt.show()

# 1st DOG
# ~~~~~~~

fnames2 = [os.path.join(train_dogs_dir, fname) for fname in os.listdir(train_dogs_dir)]
img_path3 = fnames2[3]
img3 = image.load_img(img_path3, target_size=(150, 150))
x3 = image.img_to_array(img3)
x3 = x3.reshape((1,) + x3.shape)

i = 0
for batch in datagen.flow(x3, batch_size=1):
    plt.figure(i)
    imgplot3 = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 2 == 0:
        break

plt.show()
```

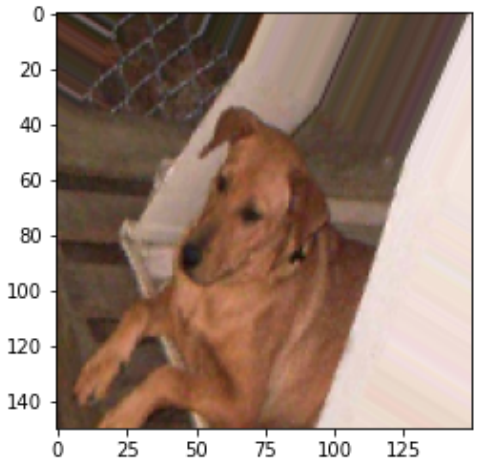
```
# 2nd DOG
# ~~~~~~~

img_path4 = fnames2[2]
img4 = image.load_img(img_path4, target_size=(150, 150))
x4 = image.img_to_array(img4)
x4 = x4.reshape((1,) + x4.shape)

i = 0
for batch in datagen.flow(x4, batch_size=1):
    plt.figure(i)
    imgplot4 = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 2 == 0:
        break

plt.show()
```

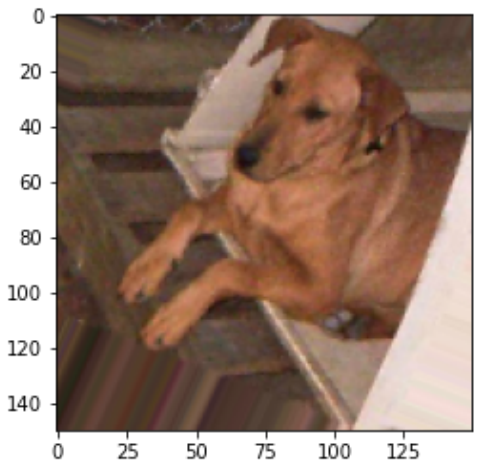


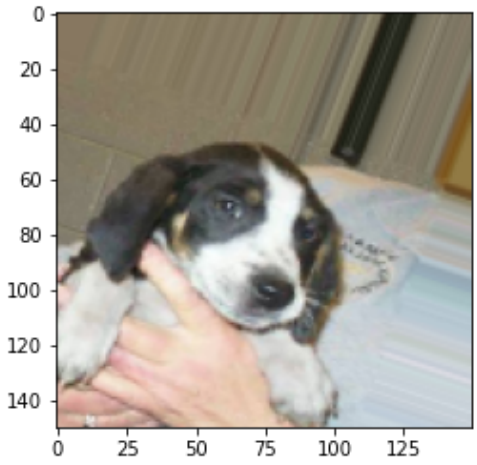


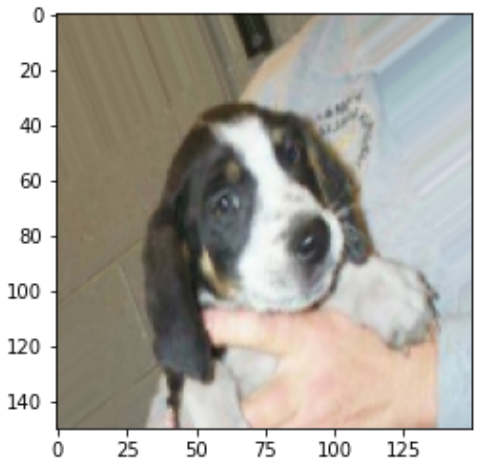

0
20
40
60
80
100
120
140
0
25
50
75
100
125


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


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


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





In [71]:
```
# ========================================================================
# D                                                                      |
# To avoid overfitting add Dropout layer and do the training of your CNN |
# using data augmentation generators. Plot training and validation accuracy |
# as well as training and validation loss                                |
# ========================================================================
```

In [73]:
```
# ------------------------------------------
# DEFINING A NEW CONVNET THAT INCLUDES DROPOUT |
# ------------------------------------------

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))  # Dropout for regularization
# Dropout randomly drops some layers in a neural networks and then learns with the reduced netwo
# The network learns to be independent and not reliable on a single layer
# Helps with overfitting
# .05 means to randomly drop half of the layers
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer=optimizers.Adam(lr=1e-4), metrics=['acc'])

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 7, 7, 128)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 6272)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6272)              0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

In [77]:
```
# ----------------------------------------------------
# TRAINING THE CONVNET USING DATA-AUGMENTATION GENERATORS |
# ----------------------------------------------------

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,
                                   horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)  # not augmented

train_generator = train_datagen.flow_from_directory(train_dir,  # target directory
                                                    target_size=(150, 150),  # resizes all images
                                                    batch_size=32,
                                                    class_mode='binary')

valdiation_generator = test_datagen.flow_from_directory(validation_dir,
                                                        target_size=(150, 150),
                                                        batch_size=32,
                                                        class_mode='binary')

history = model.fit_generator(train_generator,
                              steps_per_epoch=100,
                              epochs=100,
                              validation_data=validation_generator,
                              validation_steps=50)
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Epoch 1/100
100/100 [==============================] - 208s 2s/step - loss: 0.6914 - acc: 0.5303 - val_l
oss: 12.1340 - val_acc: 0.5900
Epoch 2/100
100/100 [==============================] - 214s 2s/step - loss: 0.6754 - acc: 0.5663 - val_l
oss: 72.1128 - val_acc: 0.5490
Epoch 3/100
100/100 [==============================] - 204s 2s/step - loss: 0.6642 - acc: 0.5891 - val_l
oss: 92.5164 - val_acc: 0.5020
Epoch 4/100
100/100 [==============================] - 236s 2s/step - loss: 0.6479 - acc: 0.6136 - val_l
oss: 60.8748 - val_acc: 0.5930
Epoch 5/100
100/100 [==============================] - 251s 3s/step - loss: 0.6263 - acc: 0.6487 - val_l
oss: 208.2860 - val_acc: 0.5190
Epoch 6/100
100/100 [==============================] - 232s 2s/step - loss: 0.6142 - acc: 0.6616 - val_l
oss: 12.0749 - val_acc: 0.5910
Epoch 7/100
100/100 [==============================] - 232s 2s/step - loss: 0.6074 - acc: 0.6731 - val_l
oss: 184.3235 - val_acc: 0.5330
Epoch 8/100
100/100 [==============================] - 234s 2s/step - loss: 0.5929 - acc: 0.6796 - val_l
oss: 116.1794 - val_acc: 0.5250
Epoch 9/100
100/100 [==============================] - 241s 2s/step - loss: 0.5839 - acc: 0.6903 - val_l
oss: 180.8572 - val_acc: 0.5610
Epoch 10/100
100/100 [==============================] - 244s 2s/step - loss: 0.5718 - acc: 0.6988 - val_l
oss: 204.1322 - val_acc: 0.5720
```

```
Epoch 11/100
100/100 [==============================] - 239s 2s/step - loss: 0.5576 - acc: 0.7139 - val_l
oss: 101.9894 - val_acc: 0.5770
Epoch 12/100
100/100 [==============================] - 221s 2s/step - loss: 0.5536 - acc: 0.7132 - val_l
oss: 236.0468 - val_acc: 0.5870
Epoch 13/100
100/100 [==============================] - 218s 2s/step - loss: 0.5402 - acc: 0.7236 - val_l
oss: 104.5838 - val_acc: 0.6140
Epoch 14/100
100/100 [==============================] - 205s 2s/step - loss: 0.5355 - acc: 0.7290 - val_l
oss: 115.1791 - val_acc: 0.6170
Epoch 15/100
100/100 [==============================] - 192s 2s/step - loss: 0.5255 - acc: 0.7367 - val_l
oss: 415.0259 - val_acc: 0.5460
Epoch 16/100
100/100 [==============================] - 213s 2s/step - loss: 0.5311 - acc: 0.7311 - val_l
oss: 190.7686 - val_acc: 0.6040
Epoch 17/100
100/100 [==============================] - 191s 2s/step - loss: 0.5214 - acc: 0.7522 - val_l
oss: 152.4332 - val_acc: 0.6310
Epoch 18/100
100/100 [==============================] - 183s 2s/step - loss: 0.5121 - acc: 0.7538 - val_l
oss: 119.8108 - val_acc: 0.5900
Epoch 19/100
100/100 [==============================] - 180s 2s/step - loss: 0.5076 - acc: 0.7472 - val_l
oss: 205.3783 - val_acc: 0.5870
Epoch 20/100
100/100 [==============================] - 184s 2s/step - loss: 0.5061 - acc: 0.7516 - val_l
oss: 94.4848 - val_acc: 0.6060
Epoch 21/100
100/100 [==============================] - 182s 2s/step - loss: 0.5134 - acc: 0.7412 - val_l
oss: 162.0881 - val_acc: 0.6170
Epoch 22/100
100/100 [==============================] - 181s 2s/step - loss: 0.4962 - acc: 0.7595 - val_l
oss: 139.1903 - val_acc: 0.6270
Epoch 23/100
100/100 [==============================] - 179s 2s/step - loss: 0.4880 - acc: 0.7579 - val_l
oss: 215.6177 - val_acc: 0.6290
Epoch 24/100
100/100 [==============================] - 148s 1s/step - loss: 0.4962 - acc: 0.7569 - val_l
oss: 161.3710 - val_acc: 0.6530
Epoch 25/100
100/100 [==============================] - 149s 1s/step - loss: 0.4821 - acc: 0.7711 - val_l
oss: 165.4077 - val_acc: 0.6550
Epoch 26/100
100/100 [==============================] - 151s 2s/step - loss: 0.4832 - acc: 0.7707 - val_l
oss: 171.4778 - val_acc: 0.6110
Epoch 27/100
100/100 [==============================] - 148s 1s/step - loss: 0.4808 - acc: 0.7667 - val_l
oss: 202.5868 - val_acc: 0.6310
Epoch 28/100
100/100 [==============================] - 148s 1s/step - loss: 0.4744 - acc: 0.7660 - val_l
oss: 68.9730 - val_acc: 0.6430
Epoch 29/100
100/100 [==============================] - 147s 1s/step - loss: 0.4675 - acc: 0.7819 - val_l
oss: 191.7742 - val_acc: 0.6520
Epoch 30/100
100/100 [==============================] - 148s 1s/step - loss: 0.4564 - acc: 0.7827 - val_l
oss: 92.1834 - val_acc: 0.6570
Epoch 31/100
100/100 [==============================] - 147s 1s/step - loss: 0.4661 - acc: 0.7721 - val_l
oss: 59.9744 - val_acc: 0.6180
```

```
Epoch 32/100
100/100 [==============================] - 147s 1s/step - loss: 0.4567 - acc: 0.7838 - val_l
oss: 105.1589 - val_acc: 0.6540
Epoch 33/100
100/100 [==============================] - 147s 1s/step - loss: 0.4429 - acc: 0.7852 - val_l
oss: 172.1588 - val_acc: 0.6900
Epoch 34/100
100/100 [==============================] - 146s 1s/step - loss: 0.4440 - acc: 0.7904 - val_l
oss: 63.6146 - val_acc: 0.6850
Epoch 35/100
100/100 [==============================] - 149s 1s/step - loss: 0.4425 - acc: 0.7942 - val_l
oss: 90.6968 - val_acc: 0.6540
Epoch 36/100
100/100 [==============================] - 149s 1s/step - loss: 0.4572 - acc: 0.7830 - val_l
oss: 58.4663 - val_acc: 0.6690
Epoch 37/100
100/100 [==============================] - 149s 1s/step - loss: 0.4398 - acc: 0.7949 - val_l
oss: 137.4117 - val_acc: 0.6900
Epoch 38/100
100/100 [==============================] - 147s 1s/step - loss: 0.4249 - acc: 0.8097 - val_l
oss: 153.2112 - val_acc: 0.6280
Epoch 39/100
100/100 [==============================] - 147s 1s/step - loss: 0.4328 - acc: 0.7999 - val_l
oss: 168.0579 - val_acc: 0.6780
Epoch 40/100
100/100 [==============================] - 147s 1s/step - loss: 0.4167 - acc: 0.8131 - val_l
oss: 18.9001 - val_acc: 0.6980
Epoch 41/100
100/100 [==============================] - 153s 2s/step - loss: 0.4301 - acc: 0.7993 - val_l
oss: 17.3868 - val_acc: 0.6720
Epoch 42/100
100/100 [==============================] - 148s 1s/step - loss: 0.4419 - acc: 0.7989 - val_l
oss: 76.4298 - val_acc: 0.7230
Epoch 43/100
100/100 [==============================] - 147s 1s/step - loss: 0.4135 - acc: 0.8094 - val_l
oss: 158.0459 - val_acc: 0.6900
Epoch 44/100
100/100 [==============================] - 146s 1s/step - loss: 0.4071 - acc: 0.8093 - val_l
oss: 122.2766 - val_acc: 0.6940
Epoch 45/100
100/100 [==============================] - 146s 1s/step - loss: 0.4244 - acc: 0.8056 - val_l
oss: 130.7091 - val_acc: 0.6670
Epoch 46/100
100/100 [==============================] - 148s 1s/step - loss: 0.4046 - acc: 0.8112 - val_l
oss: 205.6271 - val_acc: 0.6480
Epoch 47/100
100/100 [==============================] - 146s 1s/step - loss: 0.4079 - acc: 0.8109 - val_l
oss: 181.8083 - val_acc: 0.6620
Epoch 48/100
100/100 [==============================] - 146s 1s/step - loss: 0.4030 - acc: 0.8191 - val_l
oss: 284.3588 - val_acc: 0.6620
Epoch 49/100
100/100 [==============================] - 149s 1s/step - loss: 0.4030 - acc: 0.8147 - val_l
oss: 258.6402 - val_acc: 0.6960
Epoch 50/100
100/100 [==============================] - 164s 2s/step - loss: 0.3971 - acc: 0.8201 - val_l
oss: 115.2727 - val_acc: 0.6870
Epoch 51/100
100/100 [==============================] - 157s 2s/step - loss: 0.3999 - acc: 0.8223 - val_l
oss: 84.5245 - val_acc: 0.6820
Epoch 52/100
100/100 [==============================] - 155s 2s/step - loss: 0.4048 - acc: 0.8150 - val_l
oss: 114.3616 - val_acc: 0.6610
```

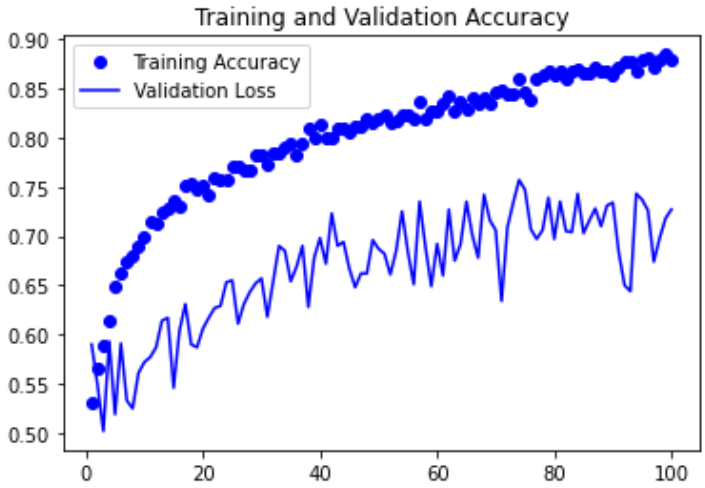
```
Epoch 53/100
100/100 [==============================] - 175s 2s/step - loss: 0.3971 - acc: 0.8179 - val_l
oss: 104.3057 - val_acc: 0.6850
Epoch 54/100
100/100 [==============================] - 175s 2s/step - loss: 0.3869 - acc: 0.8239 - val_l
oss: 127.6097 - val_acc: 0.7250
Epoch 55/100
100/100 [==============================] - 174s 2s/step - loss: 0.3823 - acc: 0.8229 - val_l
oss: 210.3990 - val_acc: 0.6840
Epoch 56/100
100/100 [==============================] - 175s 2s/step - loss: 0.3905 - acc: 0.8182 - val_l
oss: 215.3125 - val_acc: 0.6510
Epoch 57/100
100/100 [==============================] - 178s 2s/step - loss: 0.3681 - acc: 0.8364 - val_l
oss: 71.9983 - val_acc: 0.7350
Epoch 58/100
100/100 [==============================] - 175s 2s/step - loss: 0.3961 - acc: 0.8188 - val_l
oss: 133.8043 - val_acc: 0.6890
Epoch 59/100
100/100 [==============================] - 175s 2s/step - loss: 0.3826 - acc: 0.8273 - val_l
oss: 96.9491 - val_acc: 0.6490
Epoch 60/100
100/100 [==============================] - 175s 2s/step - loss: 0.3806 - acc: 0.8268 - val_l
oss: 152.7578 - val_acc: 0.6920
Epoch 61/100
100/100 [==============================] - 175s 2s/step - loss: 0.3715 - acc: 0.8354 - val_l
oss: 238.3345 - val_acc: 0.6600
Epoch 62/100
100/100 [==============================] - 192s 2s/step - loss: 0.3533 - acc: 0.8430 - val_l
oss: 96.2947 - val_acc: 0.7270
Epoch 63/100
100/100 [==============================] - 177s 2s/step - loss: 0.3738 - acc: 0.8261 - val_l
oss: 72.9868 - val_acc: 0.6750
Epoch 64/100
100/100 [==============================] - 174s 2s/step - loss: 0.3612 - acc: 0.8368 - val_l
oss: 141.8358 - val_acc: 0.6920
Epoch 65/100
100/100 [==============================] - 180s 2s/step - loss: 0.3730 - acc: 0.8279 - val_l
oss: 47.0057 - val_acc: 0.7350
Epoch 66/100
100/100 [==============================] - 177s 2s/step - loss: 0.3601 - acc: 0.8400 - val_l
oss: 81.7871 - val_acc: 0.7010
Epoch 67/100
100/100 [==============================] - 175s 2s/step - loss: 0.3669 - acc: 0.8352 - val_l
oss: 155.5197 - val_acc: 0.6780
Epoch 68/100
100/100 [==============================] - 175s 2s/step - loss: 0.3540 - acc: 0.8401 - val_l
oss: 48.5071 - val_acc: 0.7420
Epoch 69/100
100/100 [==============================] - 176s 2s/step - loss: 0.3582 - acc: 0.8346 - val_l
oss: 96.7944 - val_acc: 0.7160
Epoch 70/100
100/100 [==============================] - 178s 2s/step - loss: 0.3483 - acc: 0.8455 - val_l
oss: 256.0091 - val_acc: 0.7060
Epoch 71/100
100/100 [==============================] - 190s 2s/step - loss: 0.3460 - acc: 0.8477 - val_l
oss: 159.6587 - val_acc: 0.6340
Epoch 72/100
100/100 [==============================] - 185s 2s/step - loss: 0.3522 - acc: 0.8447 - val_l
oss: 50.8508 - val_acc: 0.7090
Epoch 73/100
100/100 [==============================] - 162s 2s/step - loss: 0.3487 - acc: 0.8438 - val_l
oss: 188.2545 - val_acc: 0.7340
```

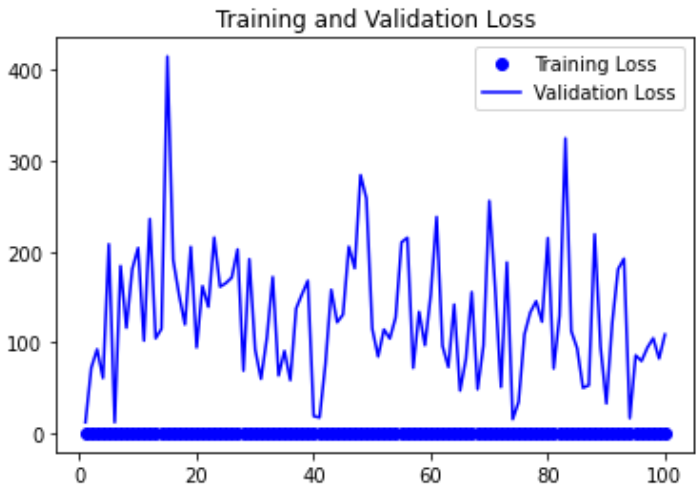
```
Epoch 74/100
100/100 [==============================] - 146s 1s/step - loss: 0.3312 - acc: 0.8592 - val_l
oss: 15.4837 - val_acc: 0.7570
Epoch 75/100
100/100 [==============================] - 147s 1s/step - loss: 0.3485 - acc: 0.8461 - val_l
oss: 34.5536 - val_acc: 0.7470
Epoch 76/100
100/100 [==============================] - 147s 1s/step - loss: 0.3557 - acc: 0.8390 - val_l
oss: 108.9379 - val_acc: 0.7070
Epoch 77/100
100/100 [==============================] - 147s 1s/step - loss: 0.3244 - acc: 0.8589 - val_l
oss: 133.2246 - val_acc: 0.6970
Epoch 78/100
100/100 [==============================] - 147s 1s/step - loss: 0.3203 - acc: 0.8634 - val_l
oss: 145.4416 - val_acc: 0.7060
Epoch 79/100
100/100 [==============================] - 147s 1s/step - loss: 0.3144 - acc: 0.8684 - val_l
oss: 122.8492 - val_acc: 0.7390
Epoch 80/100
100/100 [==============================] - 146s 1s/step - loss: 0.3188 - acc: 0.8640 - val_l
oss: 215.0446 - val_acc: 0.6970
Epoch 81/100
100/100 [==============================] - 148s 1s/step - loss: 0.3054 - acc: 0.8668 - val_l
oss: 71.3096 - val_acc: 0.7350
Epoch 82/100
100/100 [==============================] - 147s 1s/step - loss: 0.3253 - acc: 0.8599 - val_l
oss: 129.5654 - val_acc: 0.7050
Epoch 83/100
100/100 [==============================] - 146s 1s/step - loss: 0.2984 - acc: 0.8674 - val_l
oss: 324.9236 - val_acc: 0.7040
Epoch 84/100
100/100 [==============================] - 147s 1s/step - loss: 0.3062 - acc: 0.8700 - val_l
oss: 112.5832 - val_acc: 0.7430
Epoch 85/100
100/100 [==============================] - 146s 1s/step - loss: 0.3175 - acc: 0.8665 - val_l
oss: 93.6838 - val_acc: 0.7030
Epoch 86/100
100/100 [==============================] - 147s 1s/step - loss: 0.3088 - acc: 0.8662 - val_l
oss: 50.1084 - val_acc: 0.7160
Epoch 87/100
100/100 [==============================] - 146s 1s/step - loss: 0.2973 - acc: 0.8722 - val_l
oss: 52.8702 - val_acc: 0.7280
Epoch 88/100
100/100 [==============================] - 148s 1s/step - loss: 0.3028 - acc: 0.8678 - val_l
oss: 219.2124 - val_acc: 0.7100
Epoch 89/100
100/100 [==============================] - 147s 1s/step - loss: 0.3018 - acc: 0.8668 - val_l
oss: 95.1776 - val_acc: 0.7310
Epoch 90/100
100/100 [==============================] - 146s 1s/step - loss: 0.3020 - acc: 0.8643 - val_l
oss: 32.8336 - val_acc: 0.7340
Epoch 91/100
100/100 [==============================] - 145s 1s/step - loss: 0.2982 - acc: 0.8722 - val_l
oss: 121.5771 - val_acc: 0.6840
Epoch 92/100
100/100 [==============================] - 146s 1s/step - loss: 0.2824 - acc: 0.8769 - val_l
oss: 180.8146 - val_acc: 0.6500
Epoch 93/100
100/100 [==============================] - 146s 1s/step - loss: 0.2897 - acc: 0.8763 - val_l
oss: 192.2625 - val_acc: 0.6440
Epoch 94/100
100/100 [==============================] - 147s 1s/step - loss: 0.2986 - acc: 0.8684 - val_l
oss: 16.5741 - val_acc: 0.7430
```

```
Epoch 95/100
100/100 [==============================] - 147s 1s/step - loss: 0.2857 - acc: 0.8794 - val_l
oss: 85.7954 - val_acc: 0.7370
Epoch 96/100
100/100 [==============================] - 146s 1s/step - loss: 0.2799 - acc: 0.8816 - val_l
oss: 79.5412 - val_acc: 0.7260
Epoch 97/100
100/100 [==============================] - 146s 1s/step - loss: 0.2907 - acc: 0.8722 - val_l
oss: 94.3148 - val_acc: 0.6740
Epoch 98/100
100/100 [==============================] - 146s 1s/step - loss: 0.2875 - acc: 0.8797 - val_l
oss: 104.6750 - val_acc: 0.6980
Epoch 99/100
100/100 [==============================] - 146s 1s/step - loss: 0.2678 - acc: 0.8844 - val_l
oss: 82.3507 - val_acc: 0.7180
Epoch 100/100
100/100 [==============================] - 146s 1s/step - loss: 0.2760 - acc: 0.8781 - val_l
oss: 108.8086 - val_acc: 0.7270
```

In [78]:
```
model.save('cats_and_dogs_small_2.h5')
```

In [79]:
```
# ------------------------------------------------------
# DISPLAYING CURVES OF LOSS & ACCURACY DURING TRAINING |
# ------------------------------------------------------

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.plot(epochs, acc, 'bo', label='Training Accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation Loss')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()

plt.show()
```





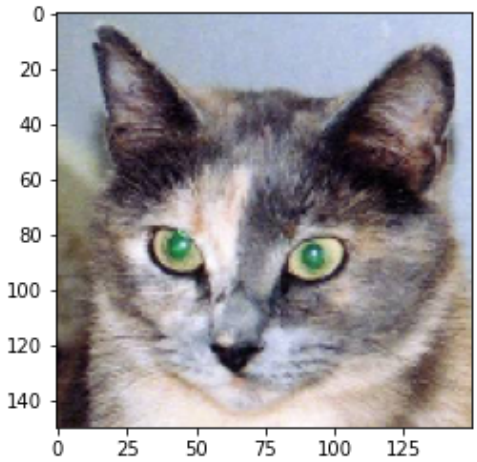
In [80]:
```
# ----------------------------------
# VISUALIZING INTERMEDIATE ACTIVATIONS |
# ----------------------------------

model = load_model('cats_and_dogs_small_2.h5')
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 7, 7, 128)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 6272)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6272)              0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

In [86]:
```
# --------------------------
# PROCESSING A SINGLE IMAGE |
# --------------------------

img_path = './Downloads/dogs_vs_cats_small_dataset/test/cats/cat.1700.jpg'

img = image.load_img(img_path, target_size=(150, 150))
img_tensor = image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis=0)
img_tensor /= 255.  # the model was trained on inputs that were preprocessed this way

print(img_tensor.shape)
```

```
(1, 150, 150, 3)
```

In [87]:
```
# ----------------------------
# DISPLAYING THE TEST PICTURE |
# ----------------------------

plt.imshow(img_tensor[0])
plt.show()
```



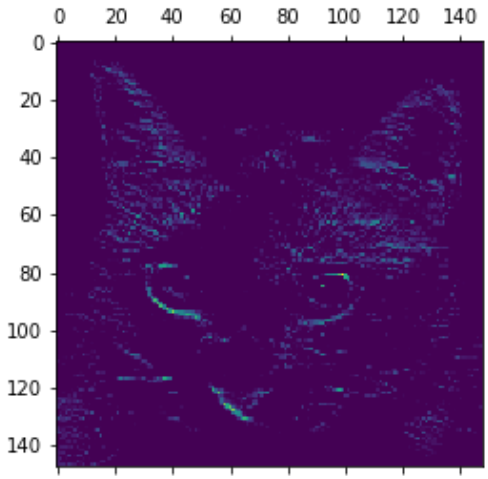
In [90]:
```
# Instantiaing a model from an input tensor and a list of output tensors

# extracts the outputs of the top 8 layers
layer_outputs = [layer.output for layer in model.layers[:8]]
# creates a model that will return these outputs, given the model input
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
```

In [91]:
```
# Running the model in predict mode

# returns a lsit of 5 numpy arrays: 1 array per layer activations
activations = activation_model.predict(img_tensor)

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

```
(1, 148, 148, 32)
```

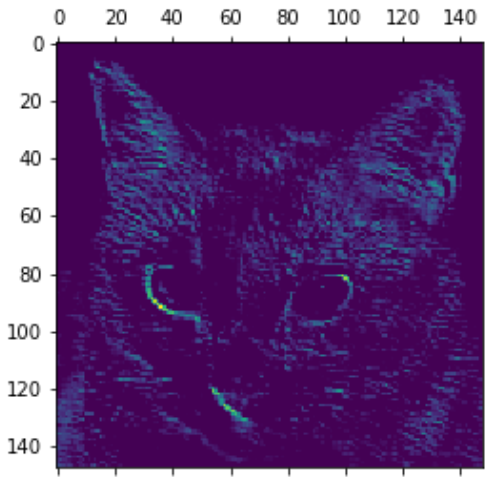
In [92]:
```
# Visualizing the fourth channel

plt.matshow(first_layer_activation[0, :, :, 4], cmap='viridis')
```

Out[92]: <matplotlib.image.AxesImage at 0x264a8197288>



In [93]:
```
# Visualizing the seventh channel

plt.matshow(first_layer_activation [0, :, :, 7], cmap='viridis')
```

Out[93]: <matplotlib.image.AxesImage at 0x264a1c33448>

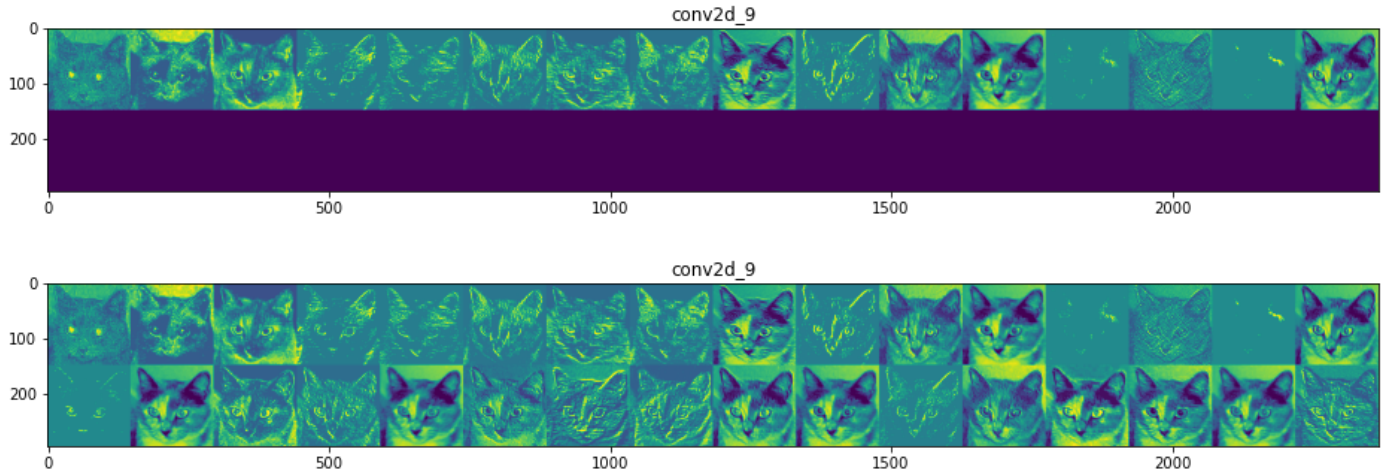


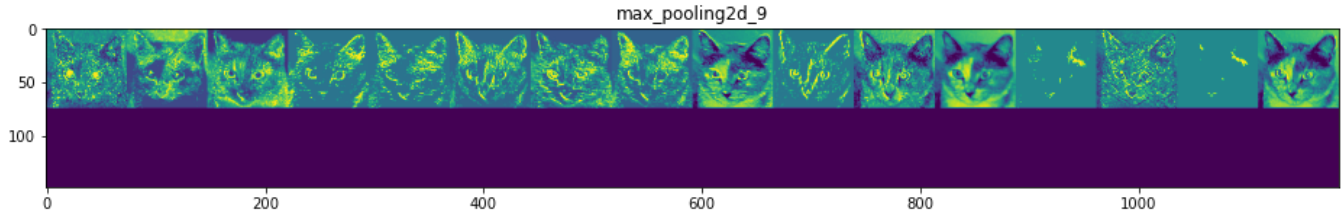
In [94]:
```
# ============================================================================
# E                                                                          |
# Visualize every channel in every intermediate activations and explain why |
# it is useful                                                               |
# ============================================================================
```

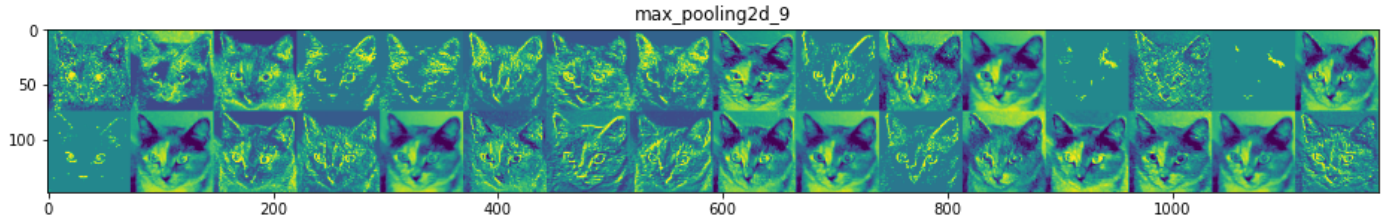
In [95]:
```
# -------------------------
# EXPLAIN WHY IT IS USEFUL |
# -------------------------

#"Intermediate activations are useful for understanding how successive
# convnet layers transform their input, and for getting a first idea of the
# meaning of indivisual convnet filters.
# Visualizing intermediate ativations consists of displaying th feature maps
# that are output by various convolution and pooling layers in a network,
# given a certain output. This gives a view into how an input is decomposed
# into different filters learned by the network. Each channel encodes
# relatively independent features, so the proper way to visualize these
# feature maps is by indepently plotting the conetnes of every channel as a
# 2D image" - Francois Chollet "DEEP LEARNING with Python"
```

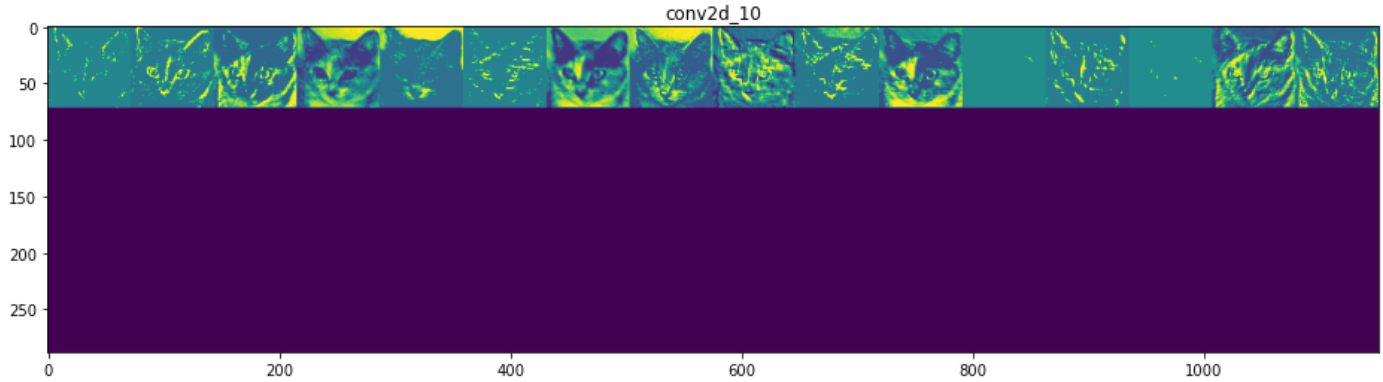
In [99]:
```
# ----------------------------------------------------------
# VISUALIZING EVERY CHANNEL IN EVERY INTERMEDIATE ACTIVATION |
# ----------------------------------------------------------

# Names of the layers so you can have them as part of your plot
layer_names = []
for layer in model.layers[:8]:
    layer_names.append(layer.name)

images_per_row = 16  # displays the feature maps

for layer_name, layer_activation in zip(layer_names, activations):
    n_features = layer_activation.shape[-1]  # number of features in the feature map

    size = layer_activation.shape[1]  # has shape (l, size, size, n_features)

    n_cols = n_features // images_per_row  # tiles the activation channels in this matrix
    display_grid = np.zeros((size * n_cols, images_per_row * size))

    for col in range(n_cols):  # tiles each filter into a big horizontal grid
        for row in range(images_per_row):
            channel_image = layer_activation[0, :, :, col * images_per_row + row]
            channel_image -= channel_image.mean()  # post processes the feature to make it visua
            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,  # displays the grid
                         row * size : (row + 1) * size] = channel_image

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

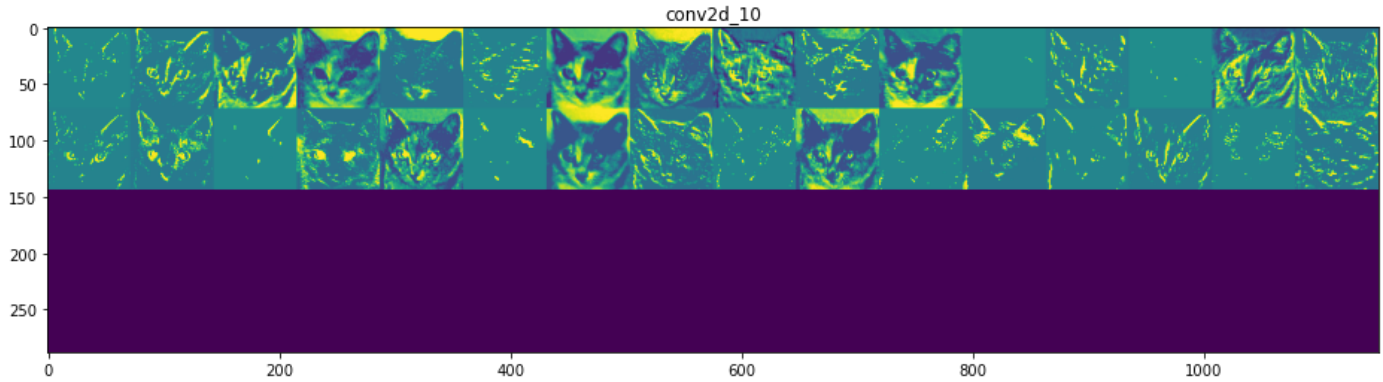
```
C:\Users\User\Anaconda3\lib\site-packages\ipykernel_launcher.py:24: RuntimeWarning: invalid val
ue encountered in true_divide
C:\Users\User\Anaconda3\lib\site-packages\ipykernel_launcher.py:33: RuntimeWarning: More than 2
0 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.fi
gure`) are retained until explicitly closed and may consume too much memory. (To control this w
arning, see the rcParam `figure.max_open_warning`).
```

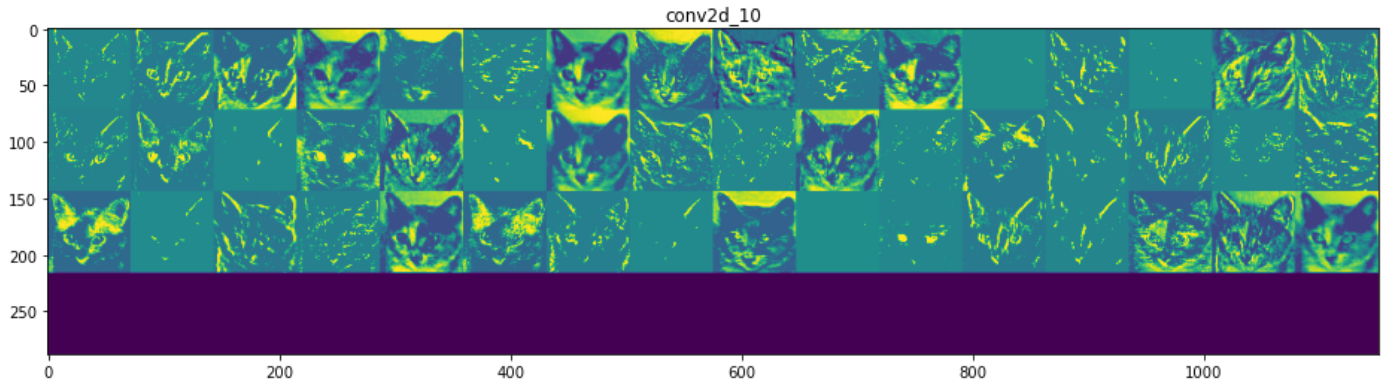


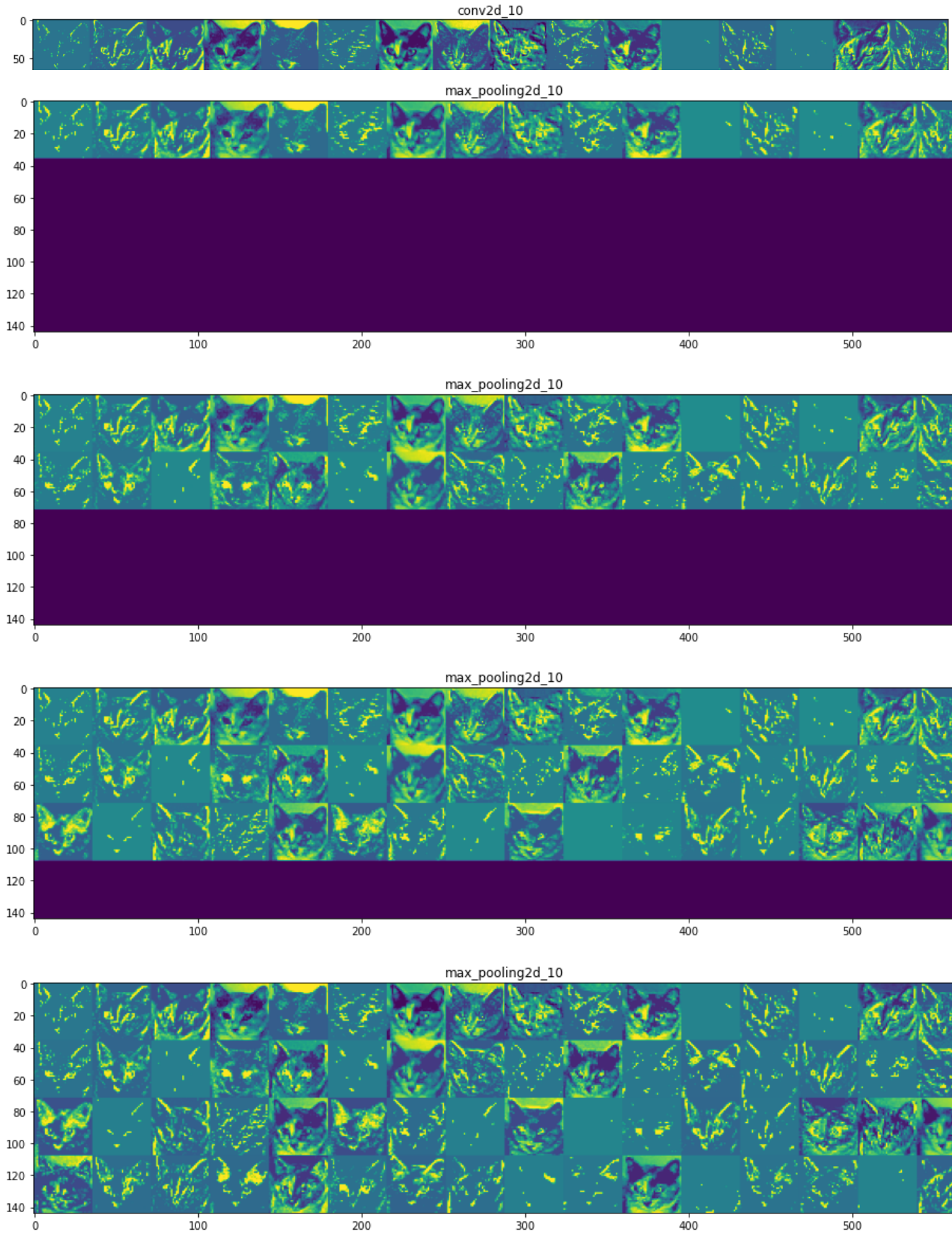

max_pooling2d_9
0
50
100
0
200
400
600
800
1000

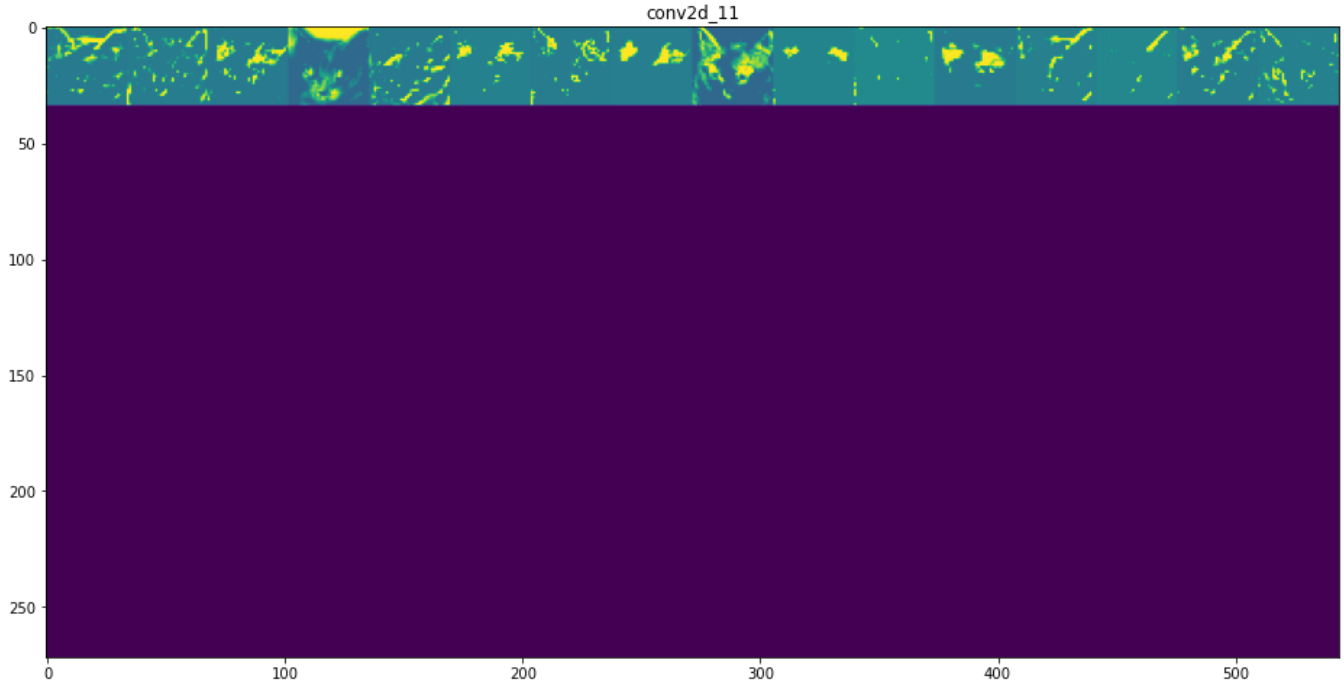

max_pooling2d_9
0
50
100
0
200
400
600
800
1000

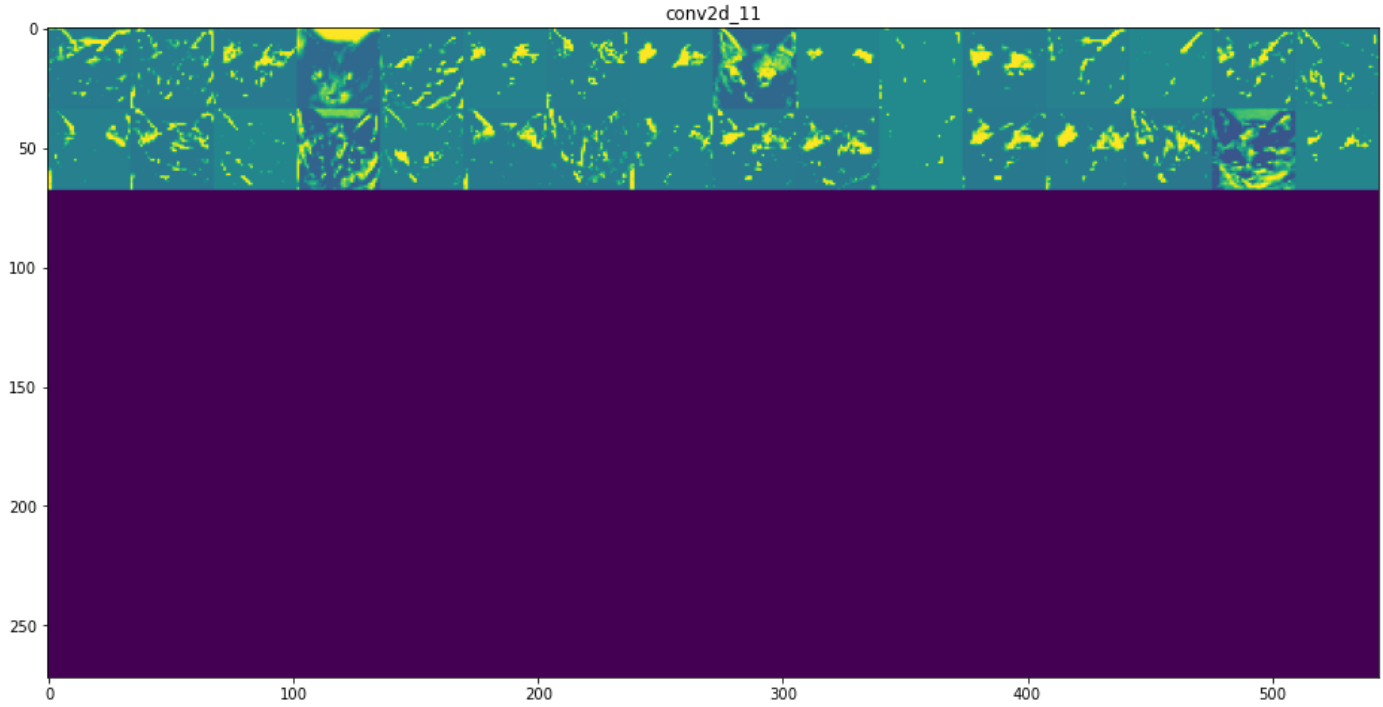

conv2d_10
0
50
100
150
200
250
0
200
400
600
800
1000

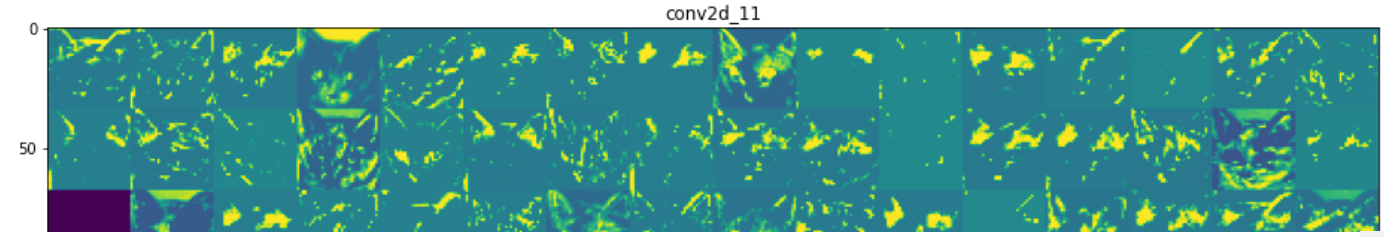

conv2d_10
0
50
100
150
200
250
0
200
400
600
800
1000

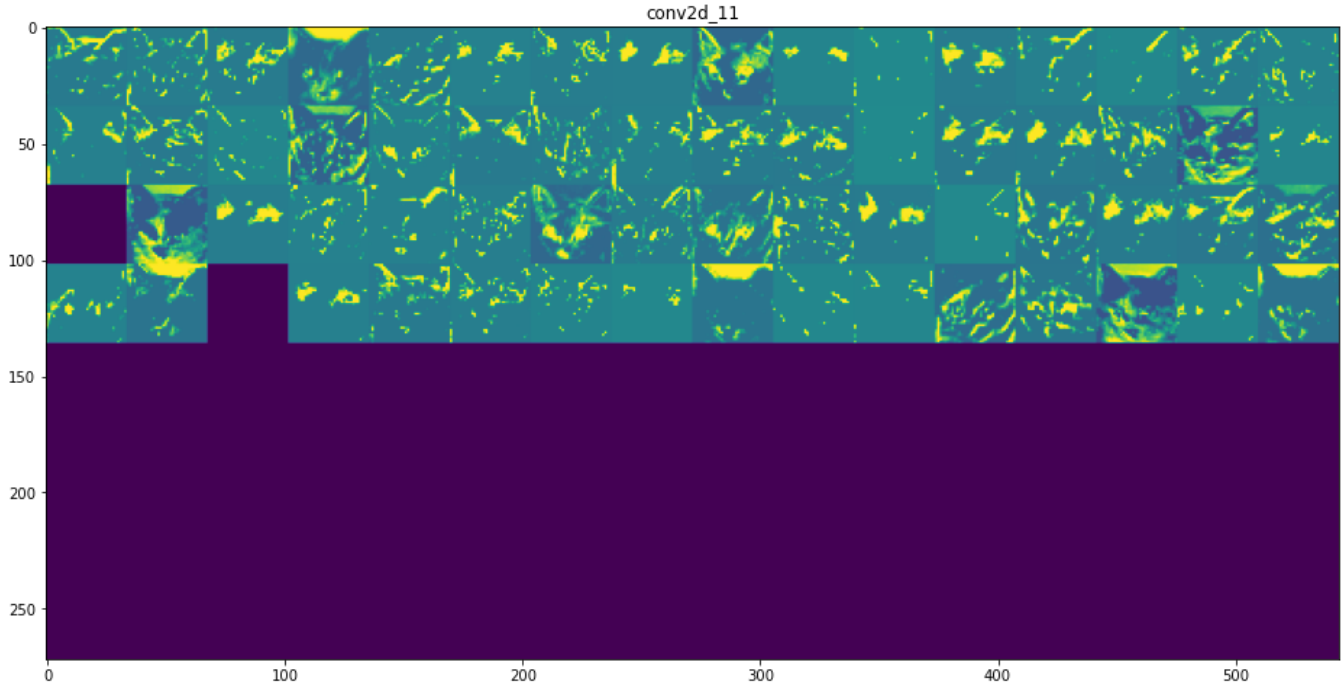

conv2d_10
0
50
100
150
200
250
0
200
400
600
800
1000

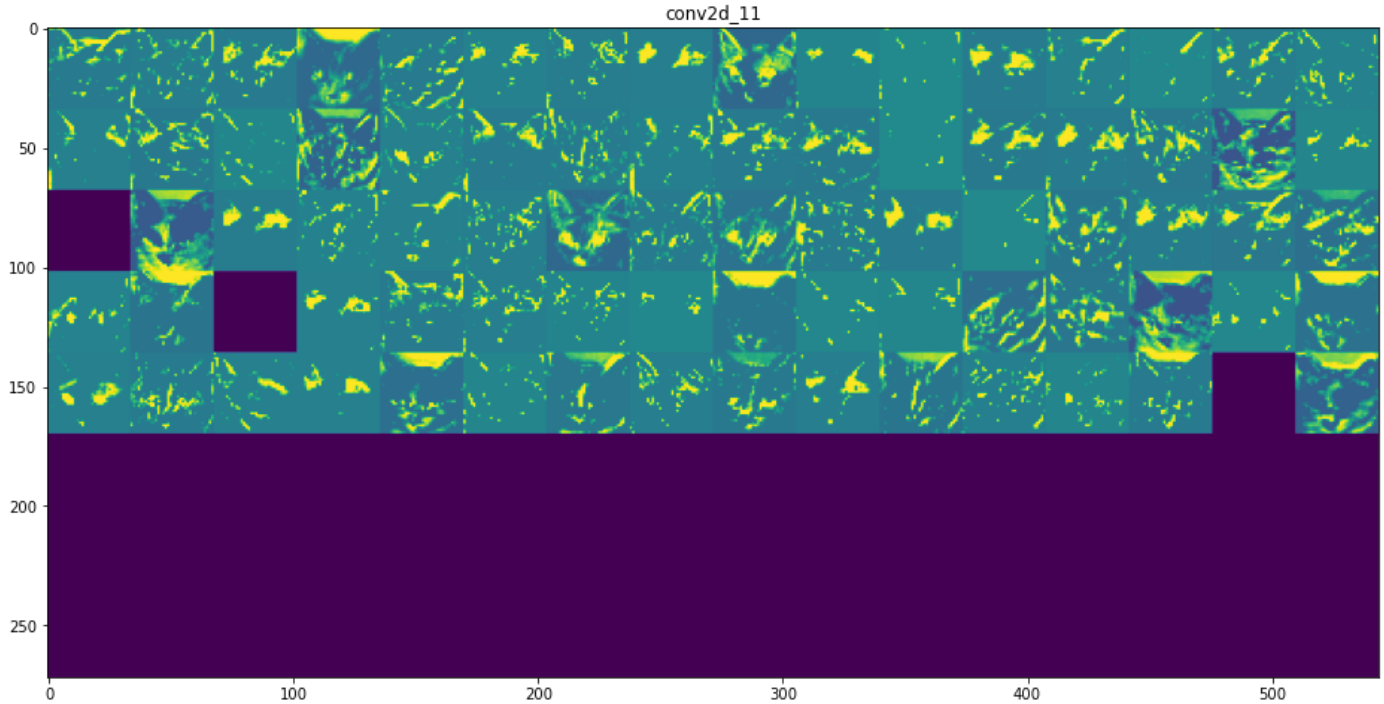

conv2d_10
max_pooling2d_10
max_pooling2d_10
max_pooling2d_10
max_pooling2d_10

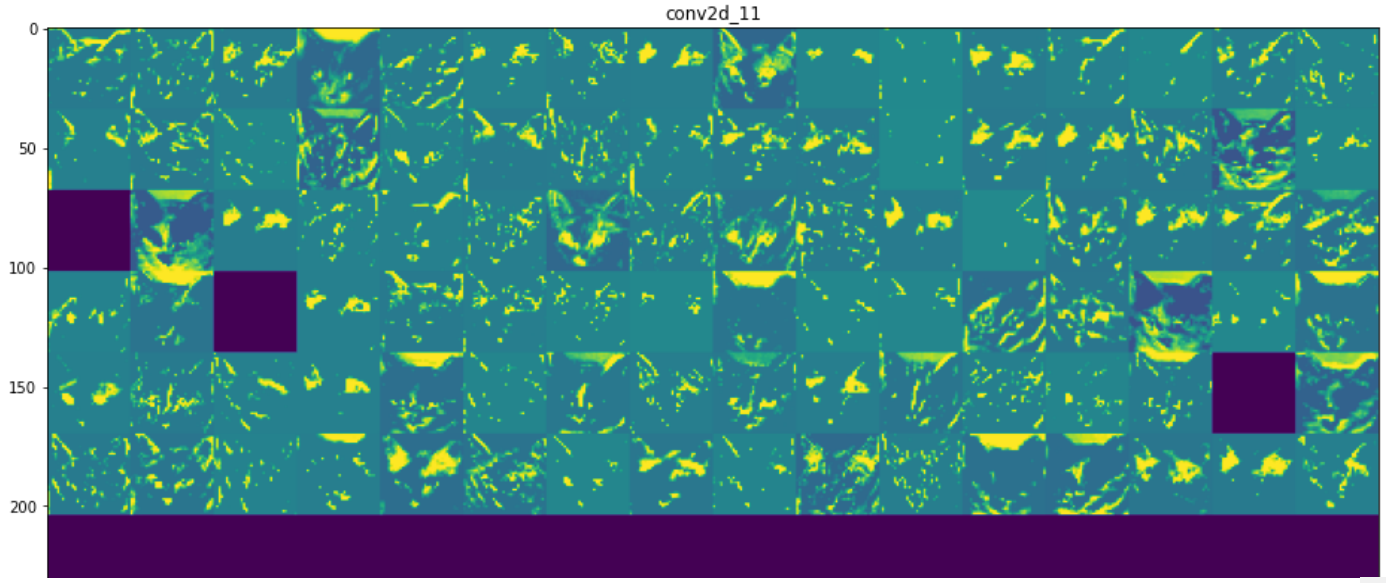

conv2d_11
0
50
100
150
200
250
0
100
200
300
400
500

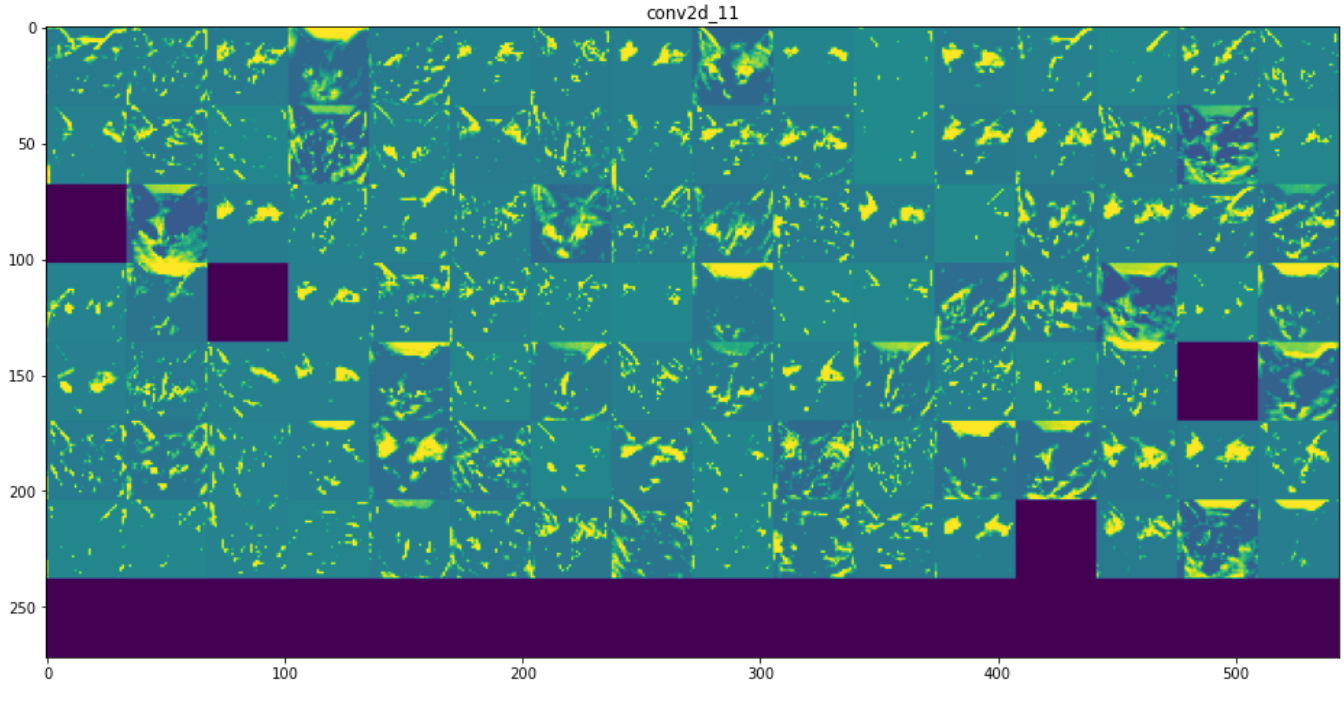

conv2d_11
0
50
100
150
200
250
0
100
200
300
400
500

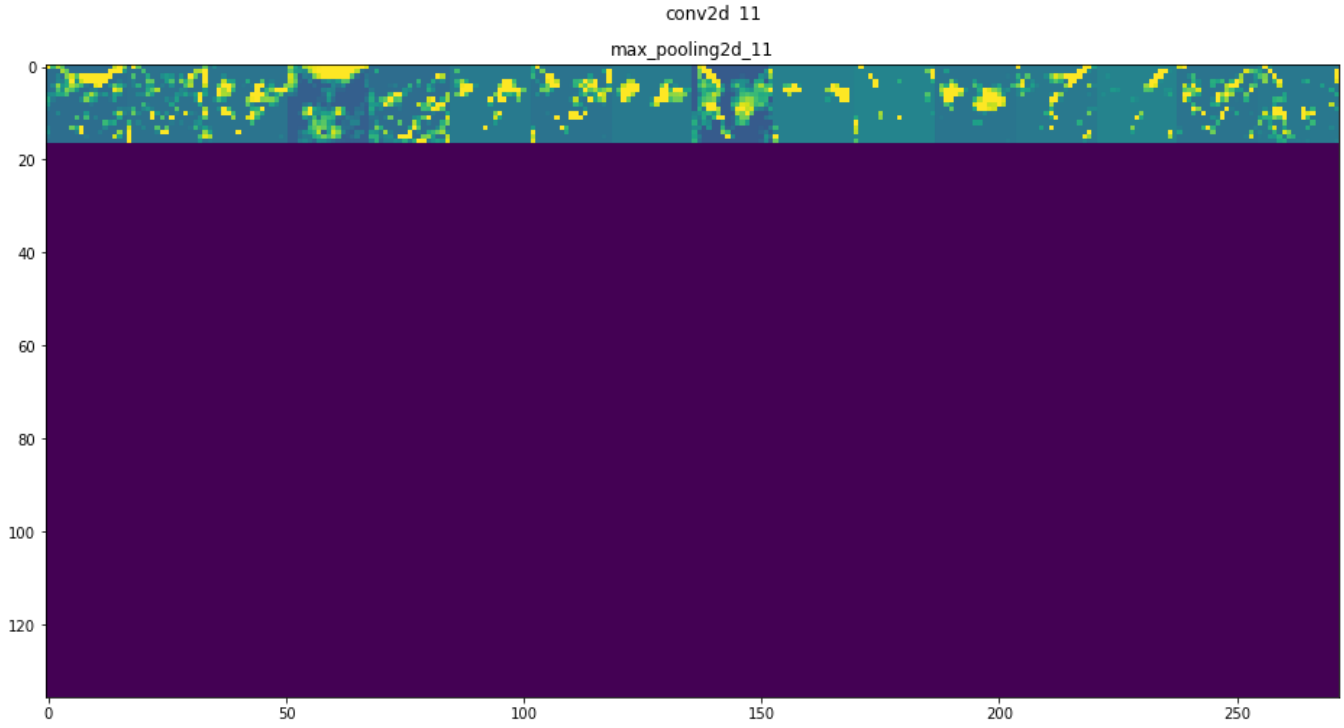

conv2d_11
0
50

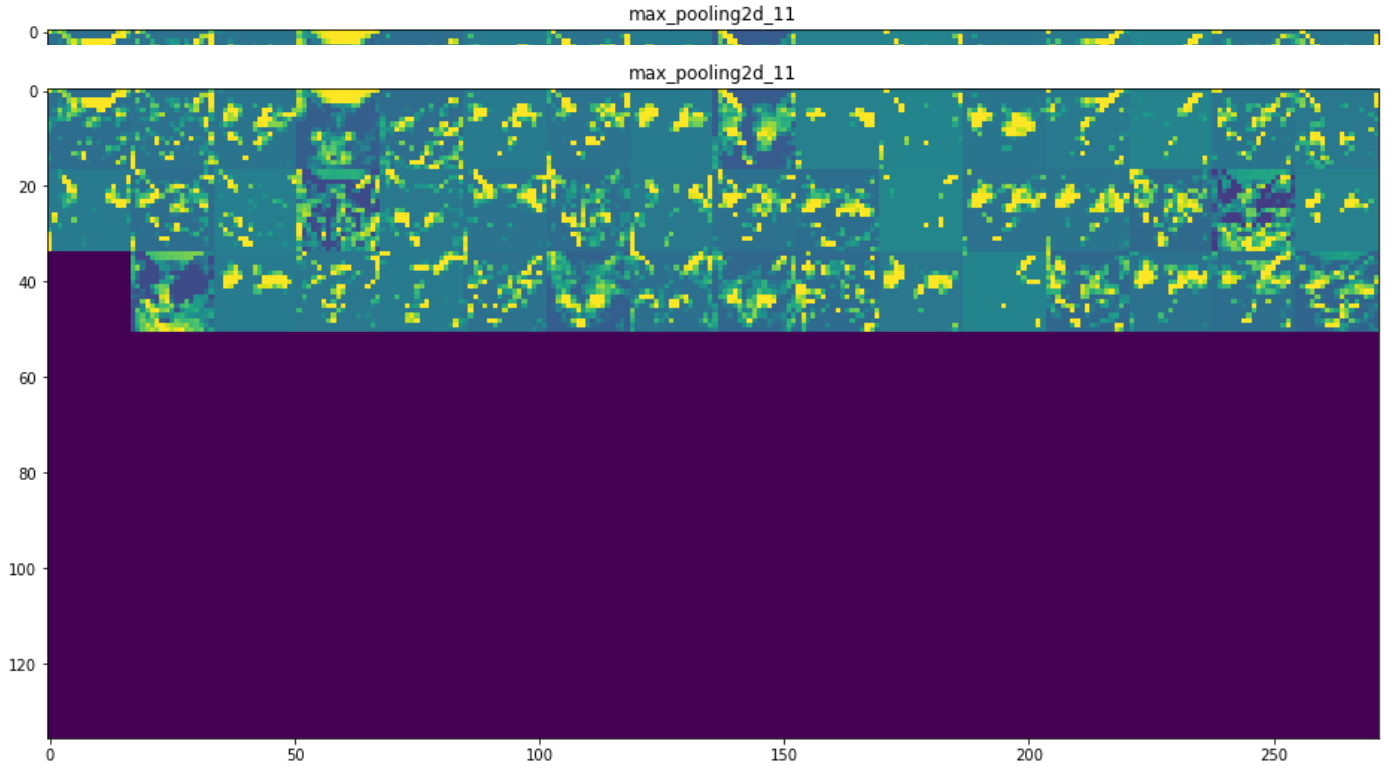

conv2d_11
0
50
100
150
200
250
0
100
200
300
400
500

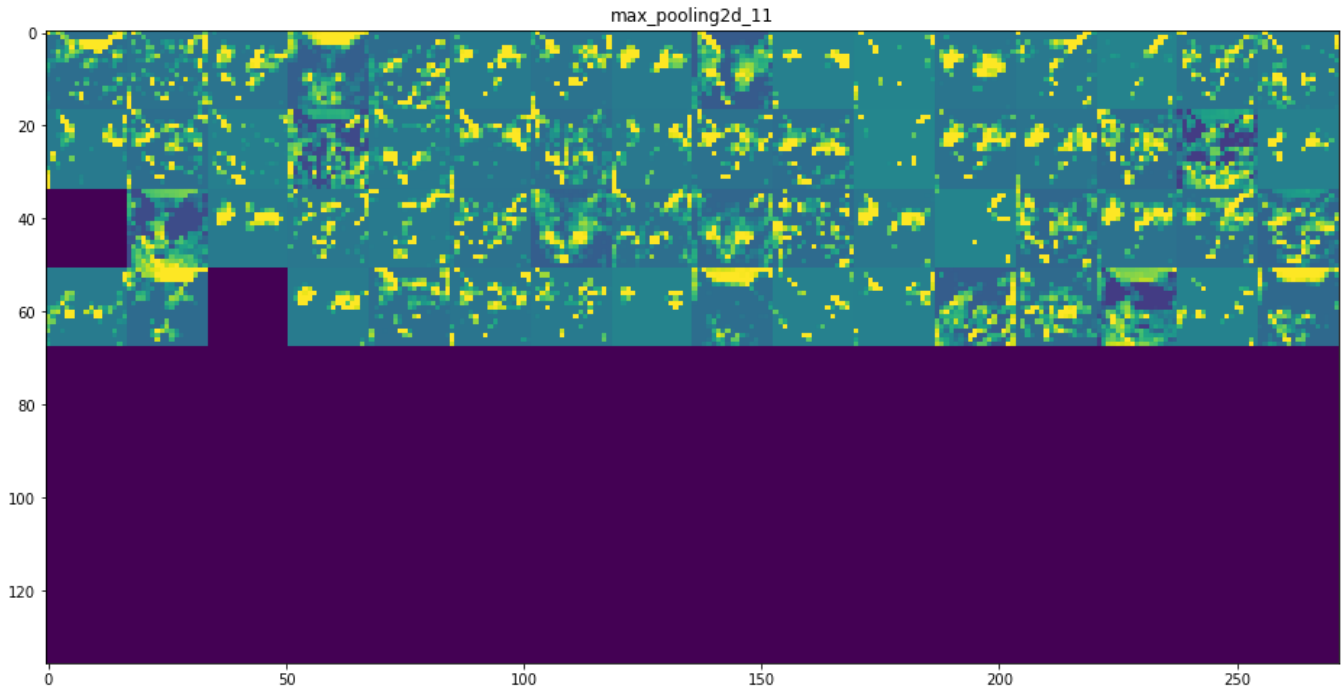

conv2d_11
0
50
100
150
200
250
0
100
200
300
400
500

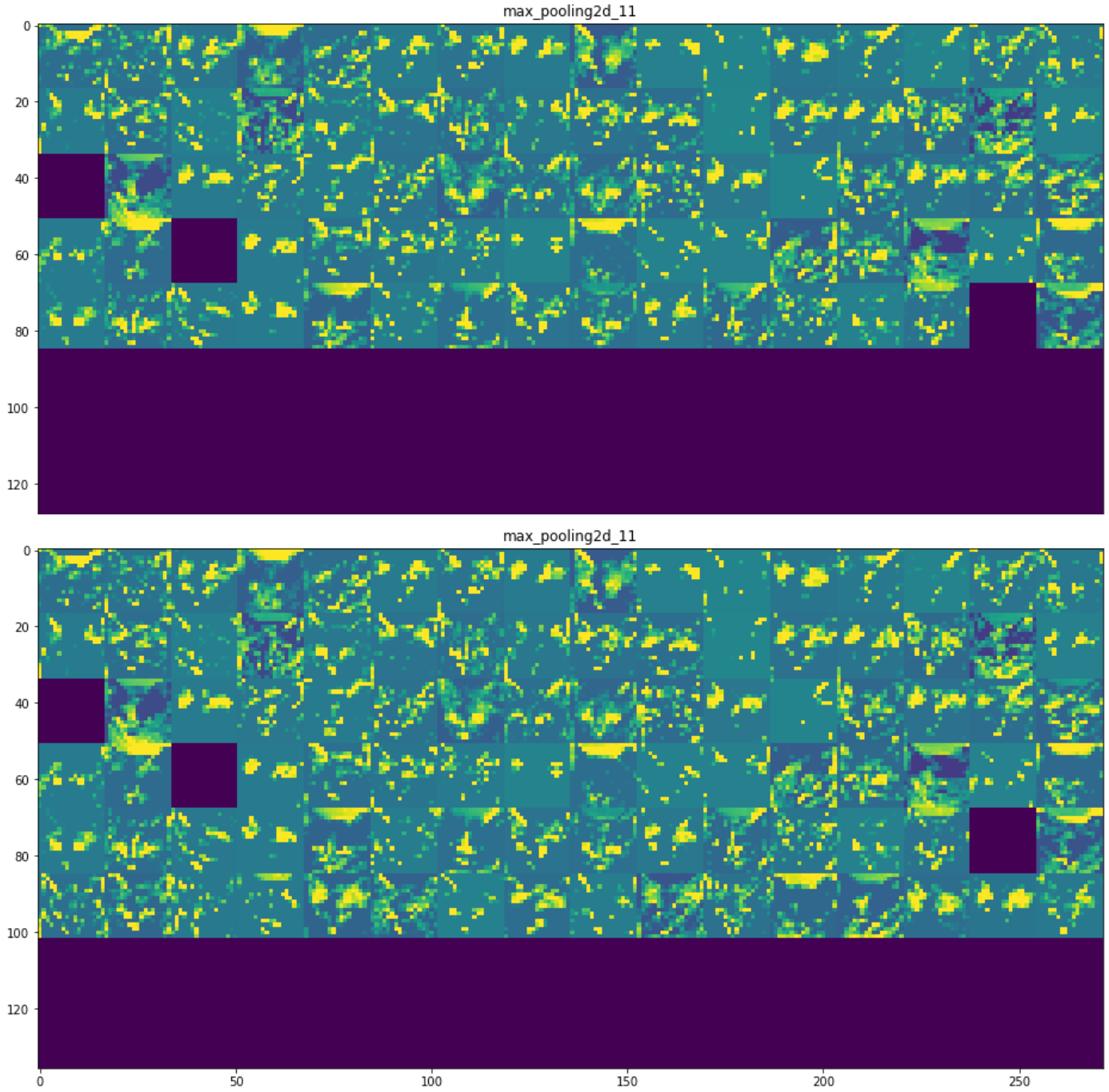

conv2d_11
0
50
100
150
200


conv2d_11
0
50
100
150
200
250
0
100
200
300
400
500

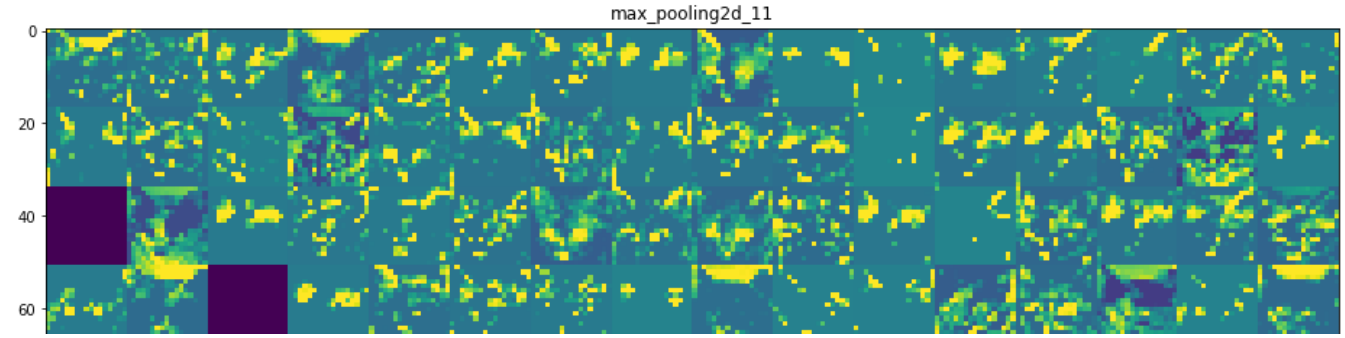

conv2d_11
max_pooling2d_11
0
20
40
60
80
100
120
0
50
100
150
200
250

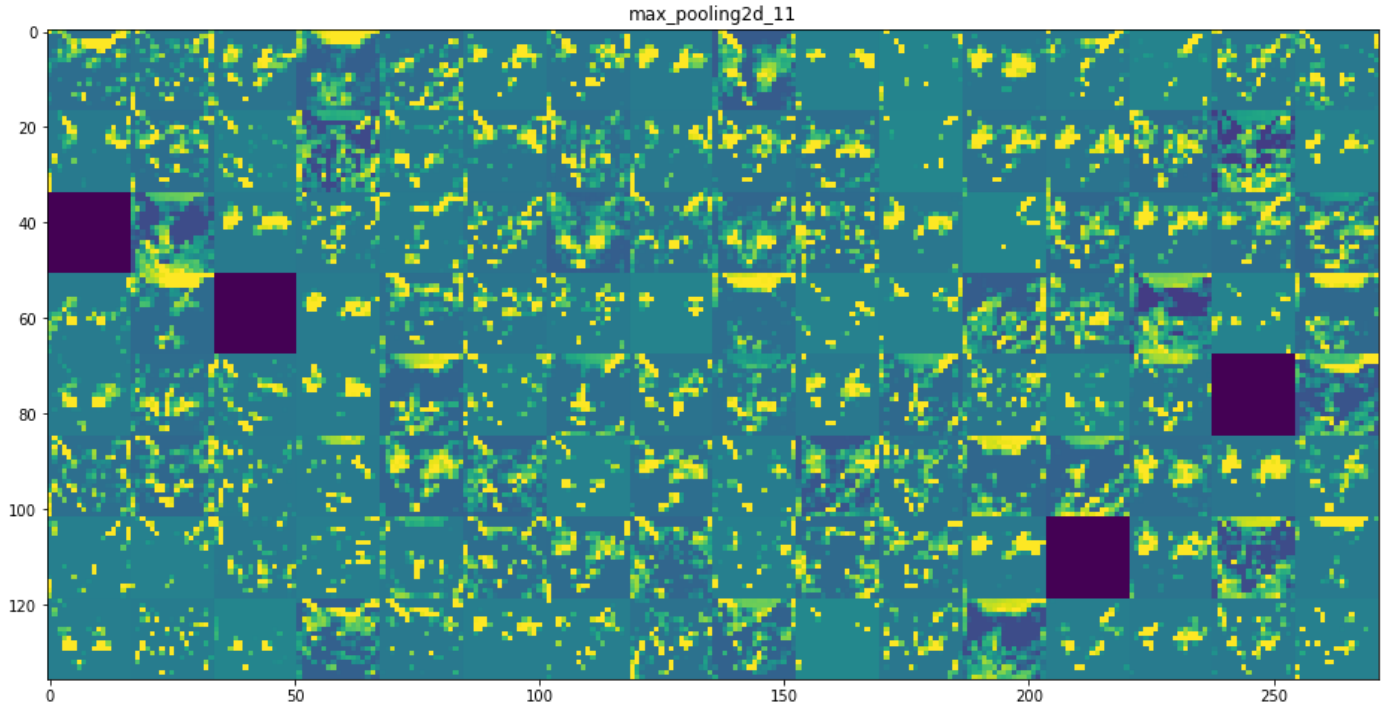

max_pooling2d_11
max_pooling2d_11
0
20
40
60
80
100
120
0
50
100
150
200
250

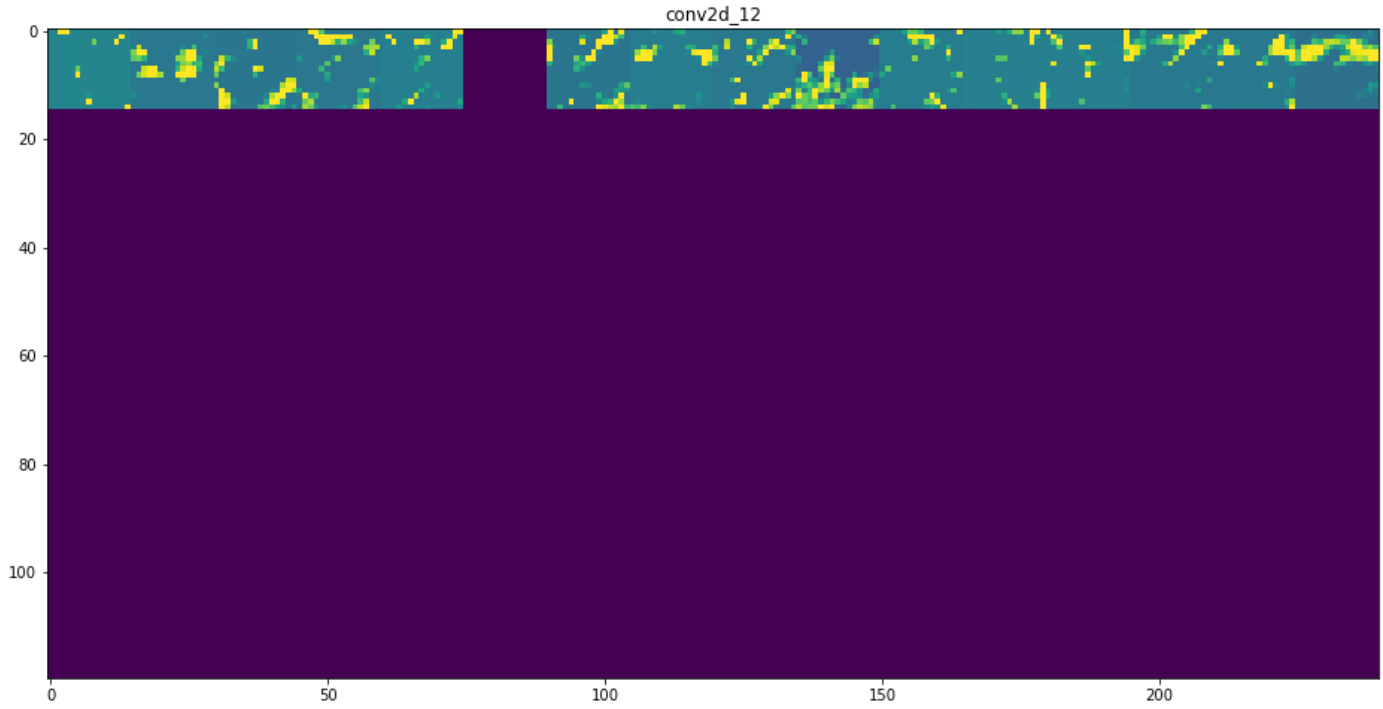

max_pooling2d_11
0
20
40
60
80
100
120
0
50
100
150
200
250

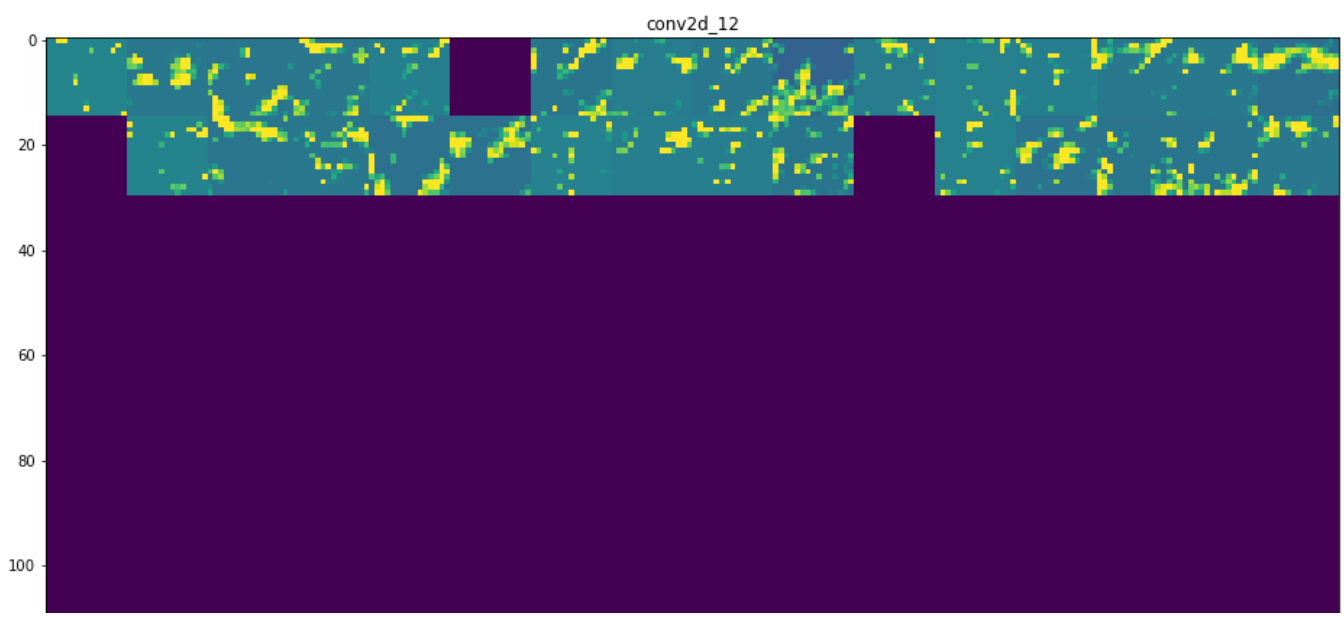

max_pooling2d_11
0
20
40
60
80
100
120

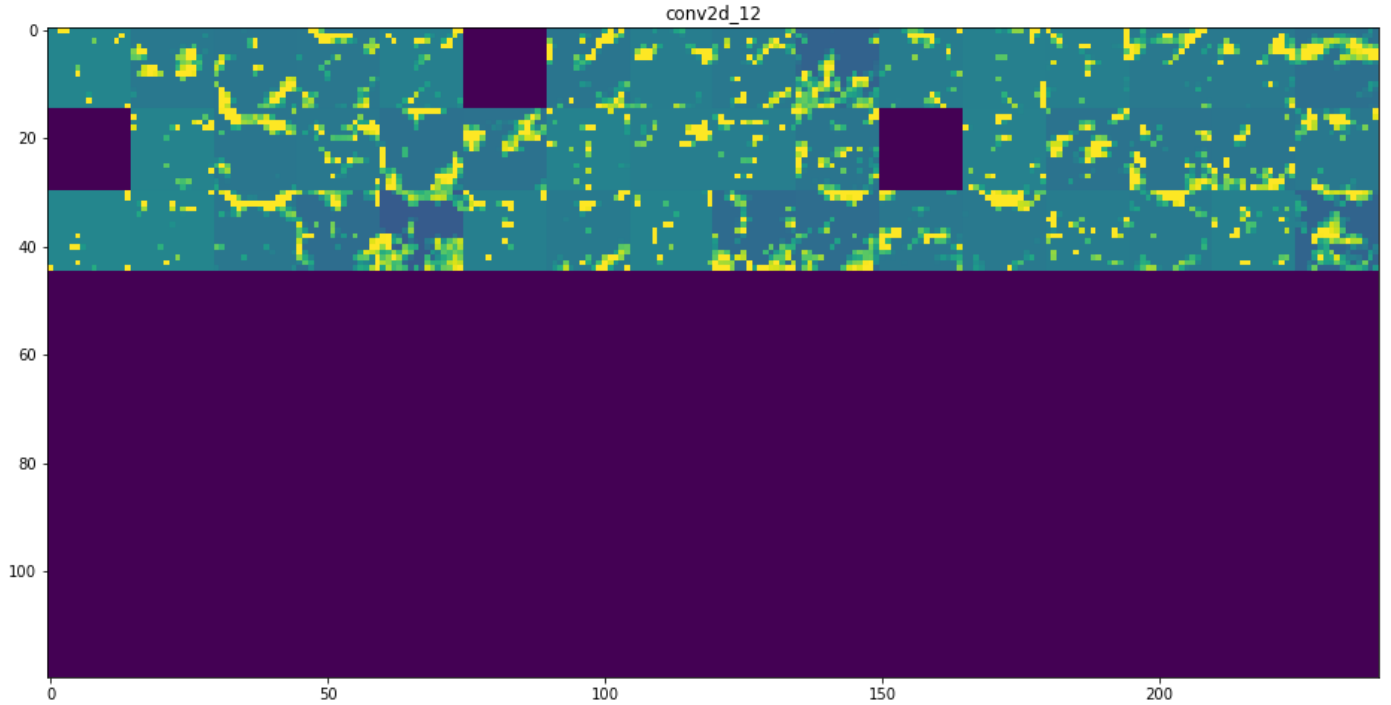
max_pooling2d_11
0
20
40
60
80
100
120
0
50
100
150
200
250

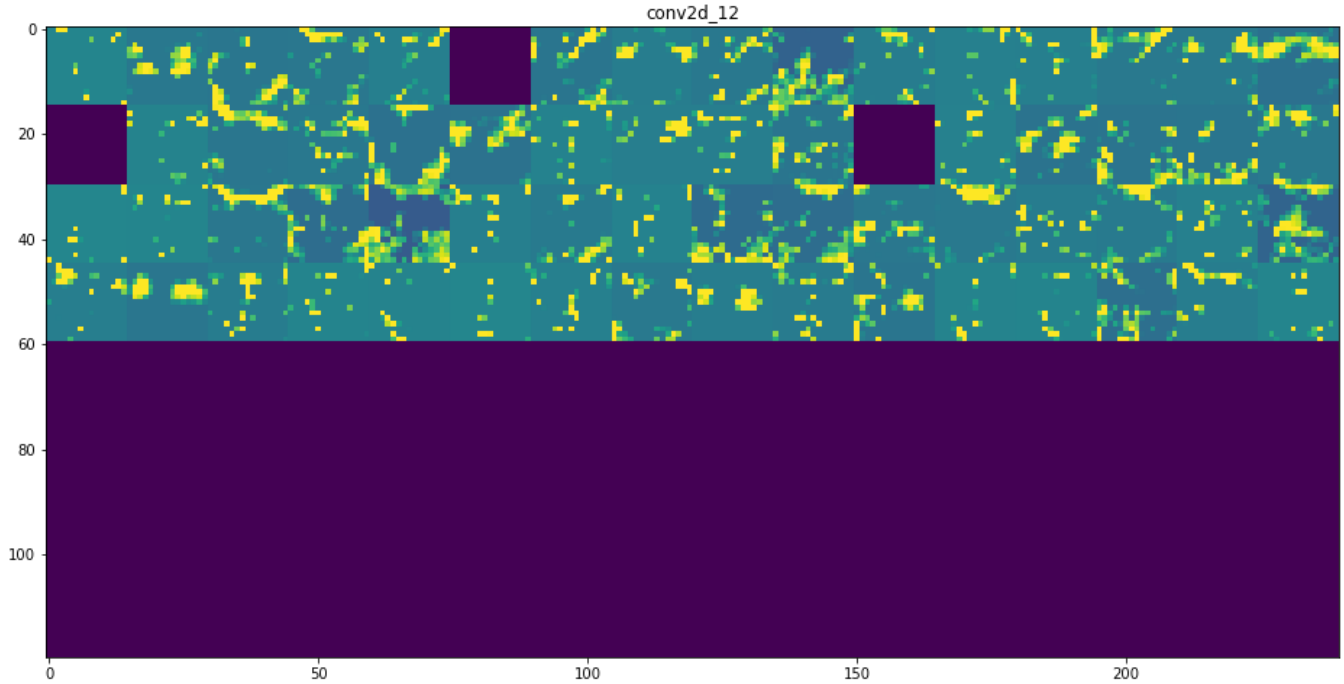

max_pooling2d_11
0
20
40
60

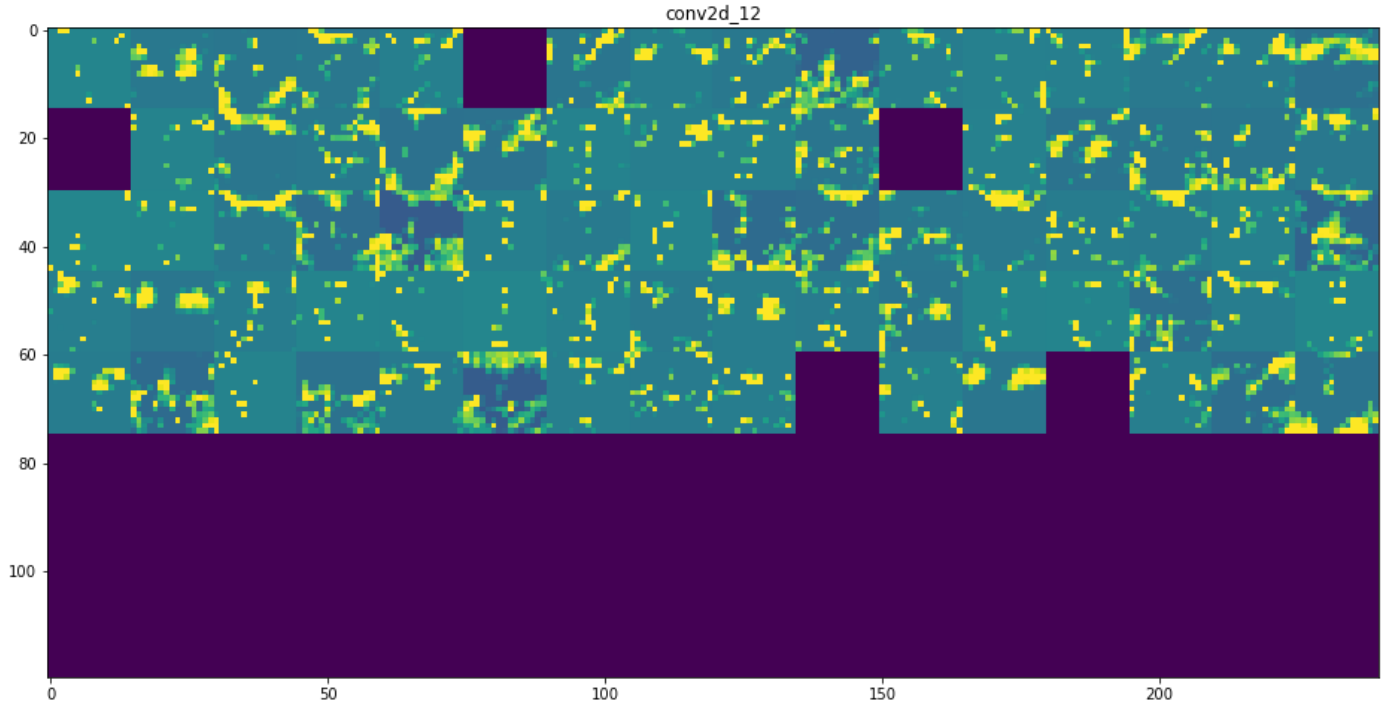

max_pooling2d_11
0
20
40
60
80
100
120
0
50
100
150
200
250

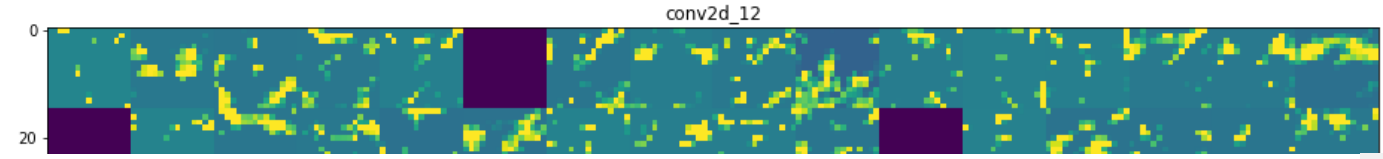

conv2d_12
0
20
40
60
80
100
0
50
100
150
200

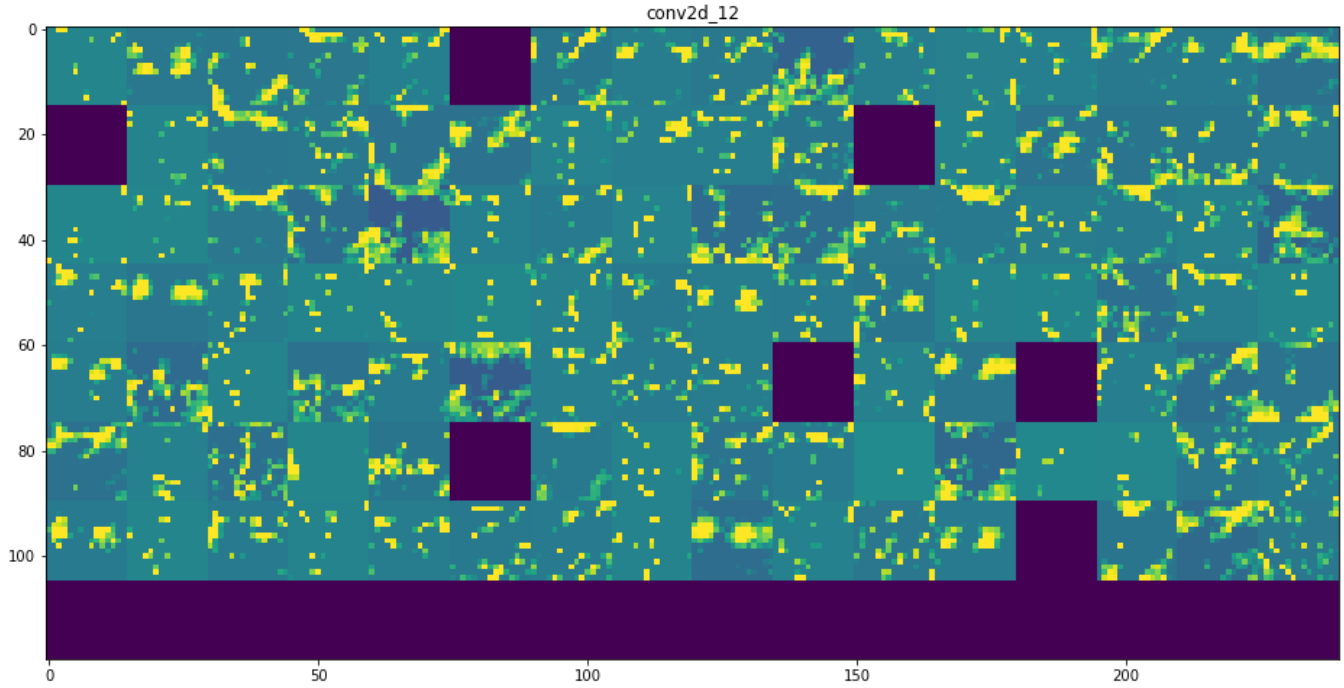

conv2d_12
0
20
40
60
80
100

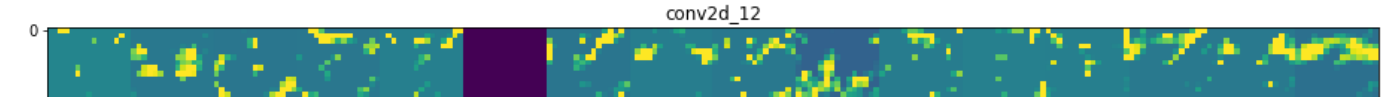

conv2d_12
0
20
40
60
80
100
0
50
100
150
200

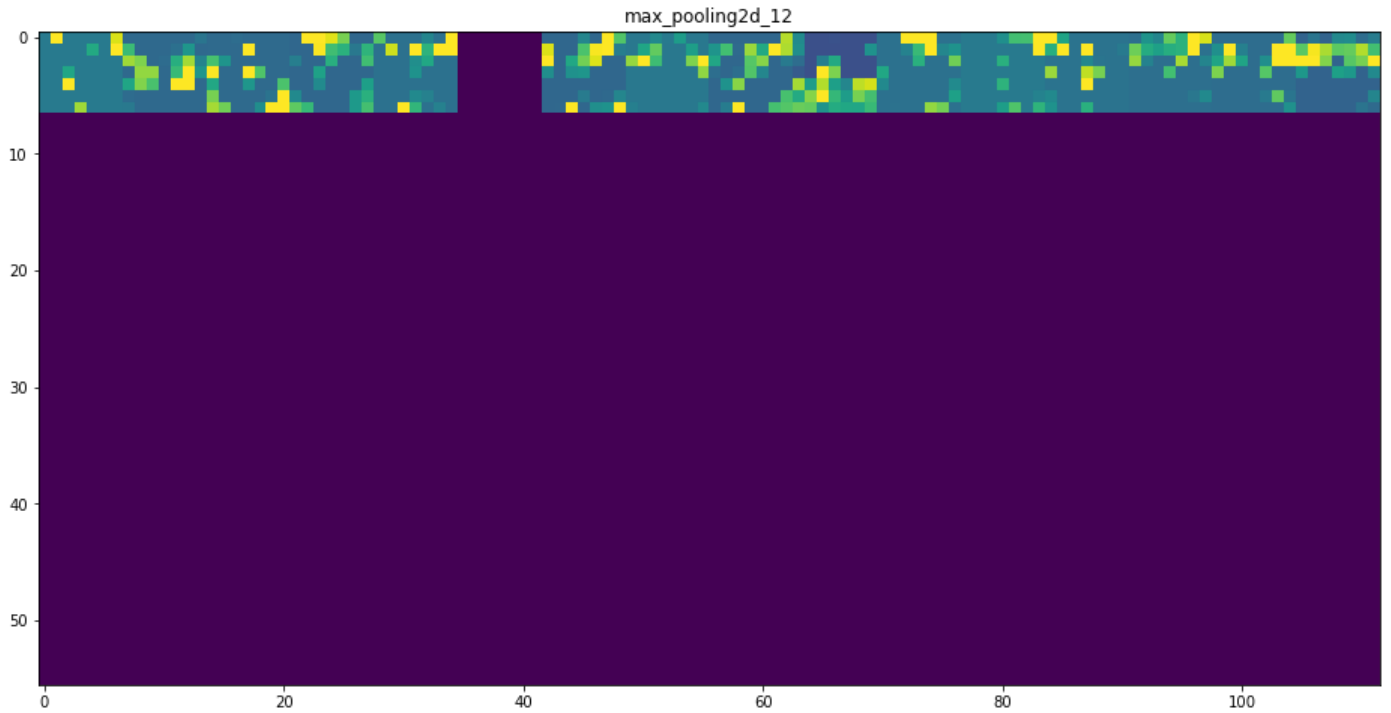

conv2d_12
0
20
40
60
80
100
0
50
100
150
200

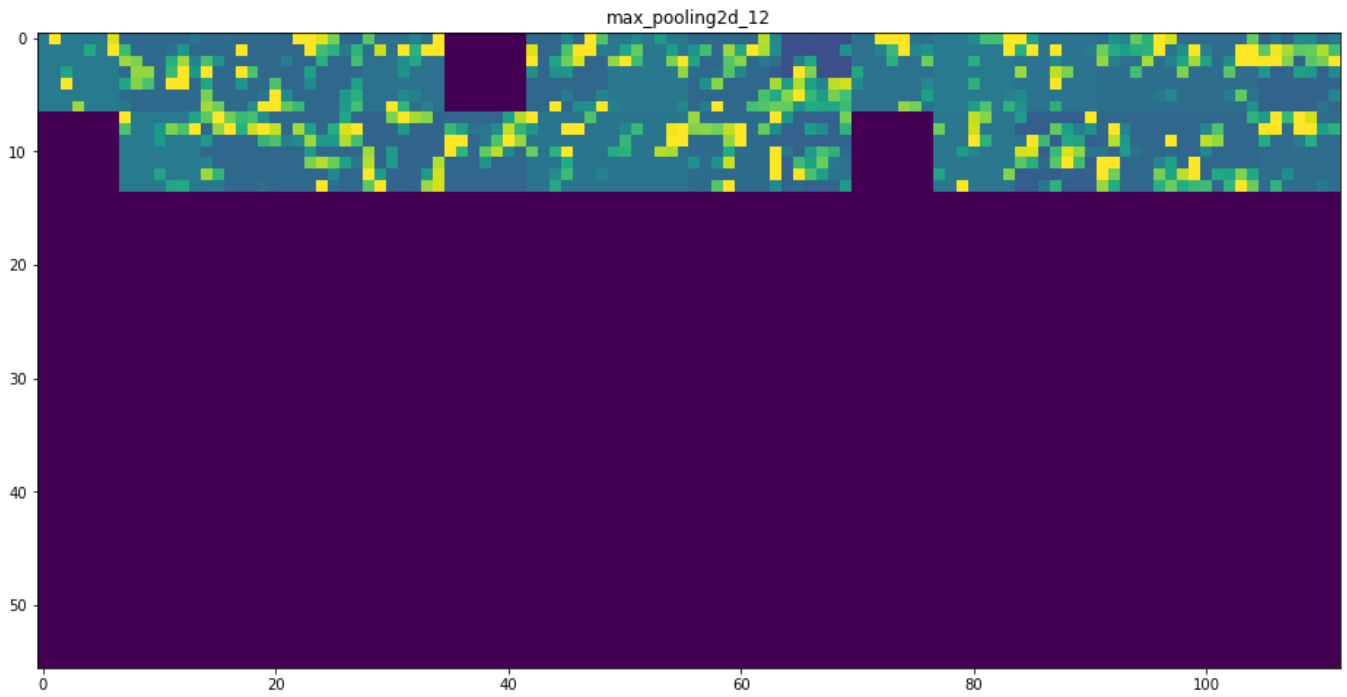

conv2d_12
0
20
40
60
80
100
0
50
100
150
200

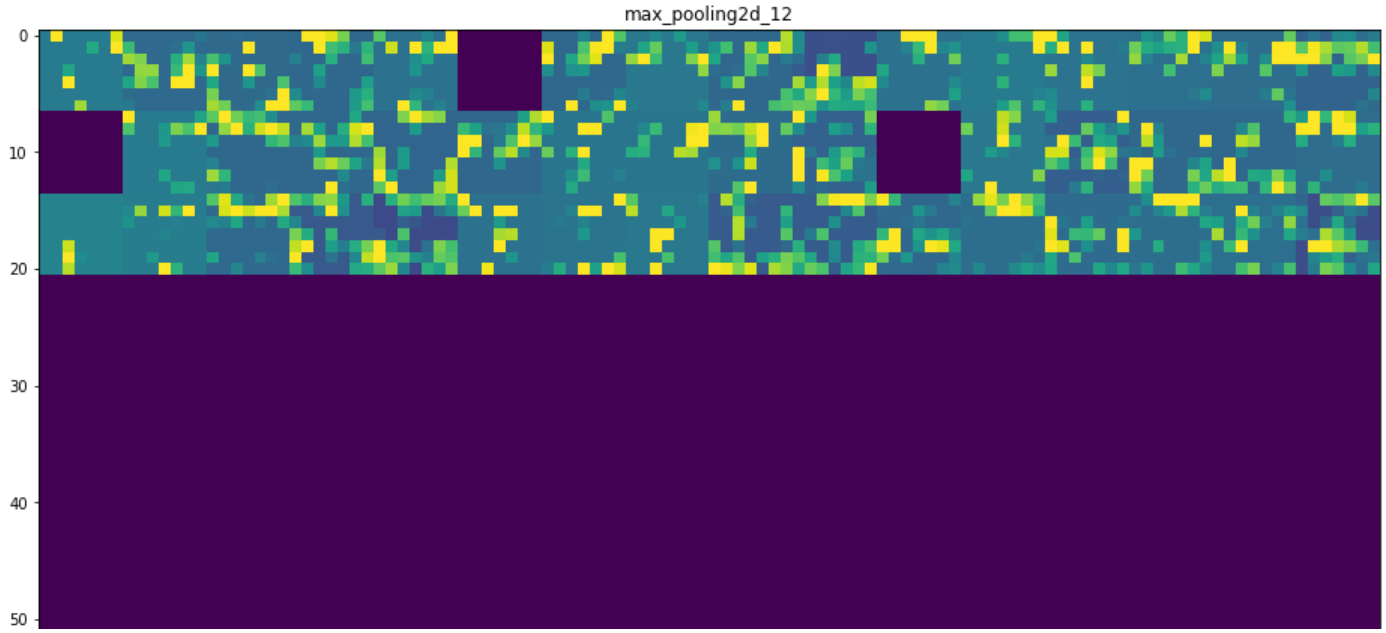

conv2d_12
0
20

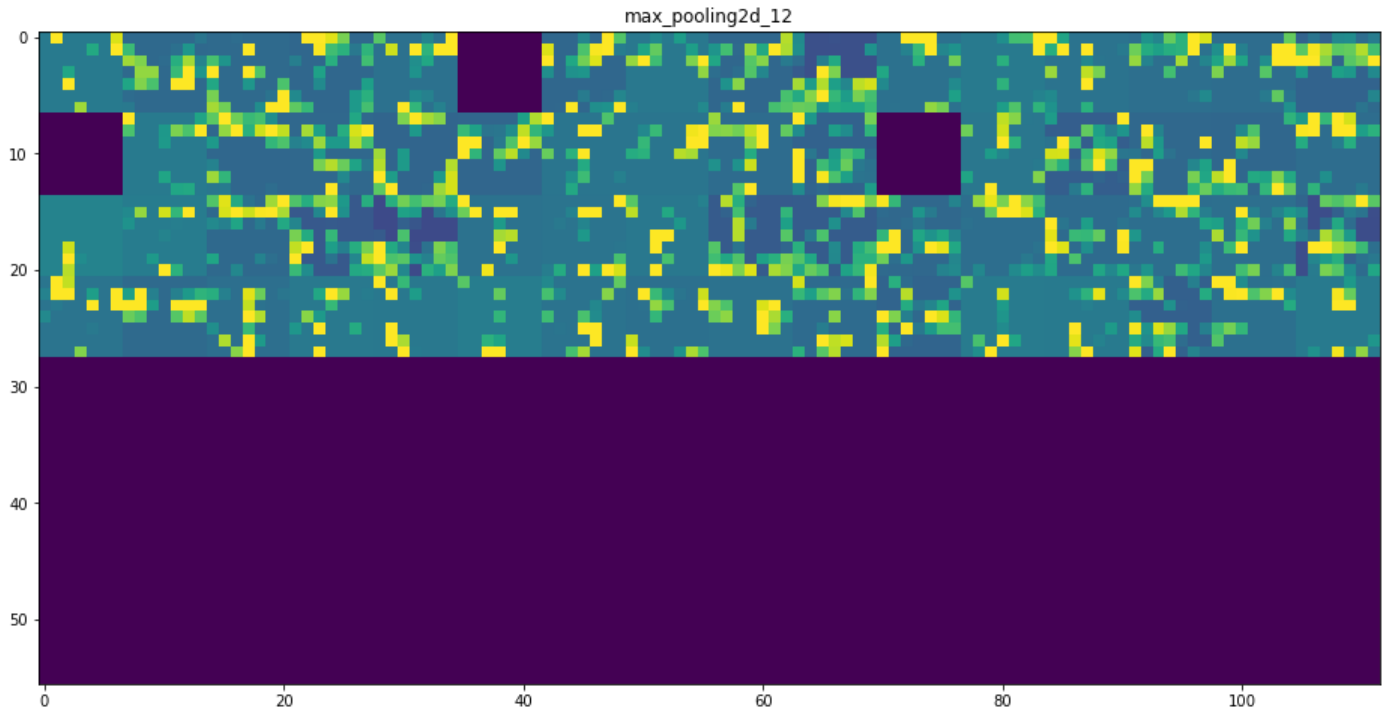

conv2d_12
0
20
40
60
80
100
0
50
100
150
200

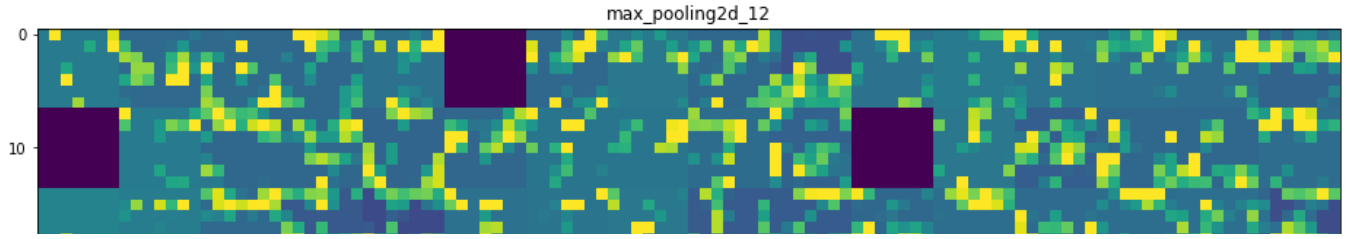

conv2d_12
0

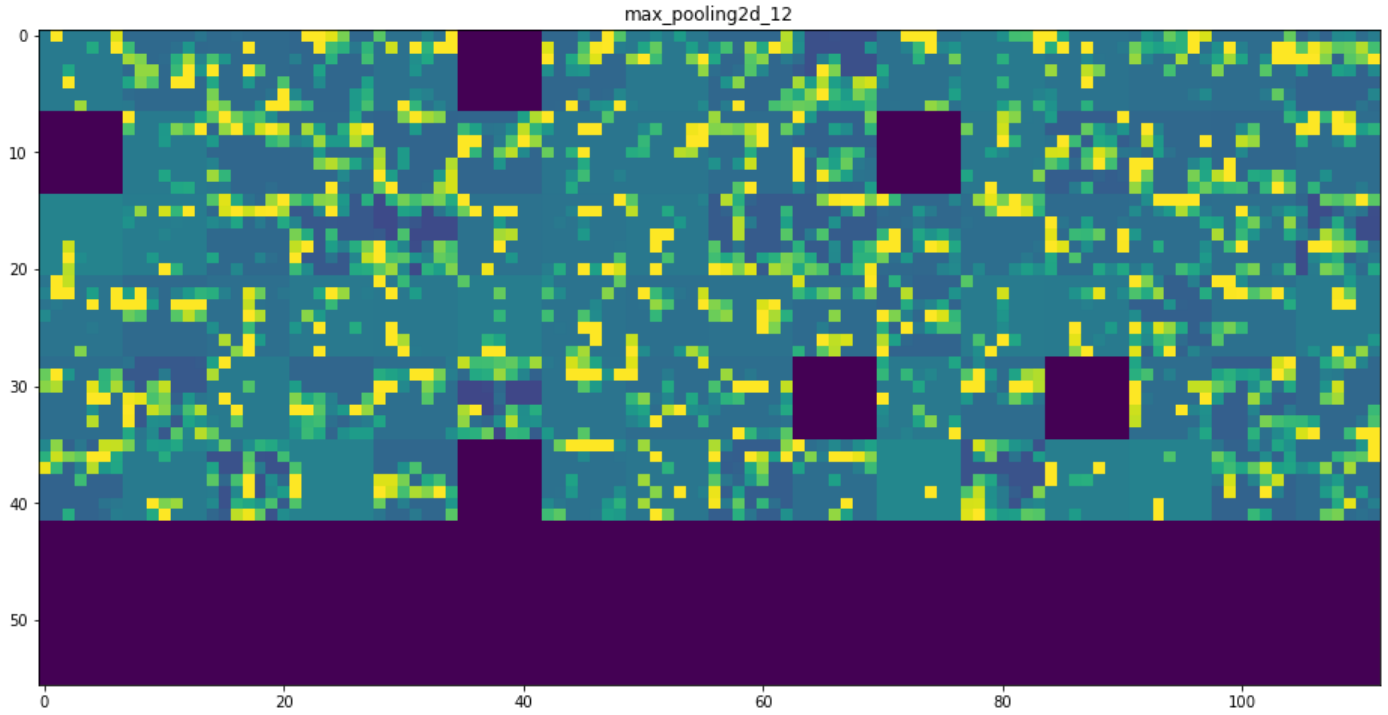

max_pooling2d_12
0
10
20
30
40
50
0
20
40
60
80
100

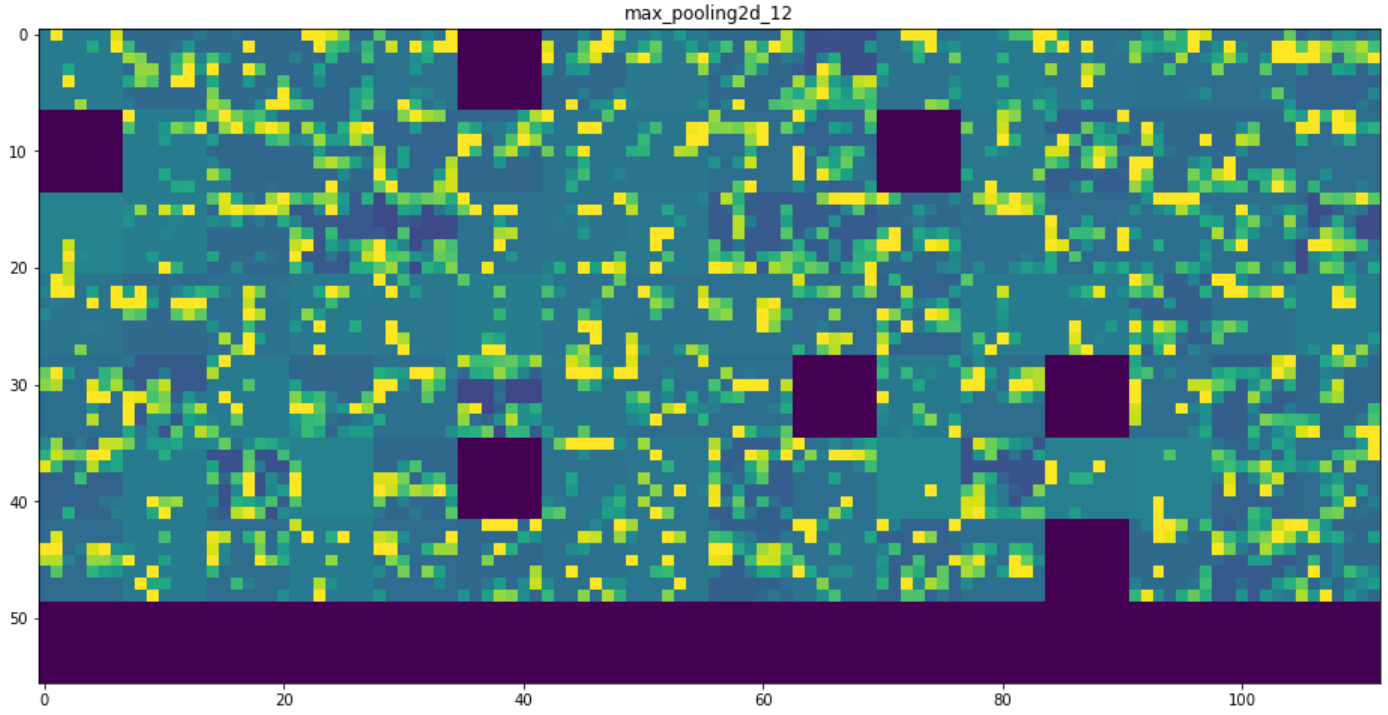

max_pooling2d_12
0
10
20
30
40
50
0
20
40
60
80
100

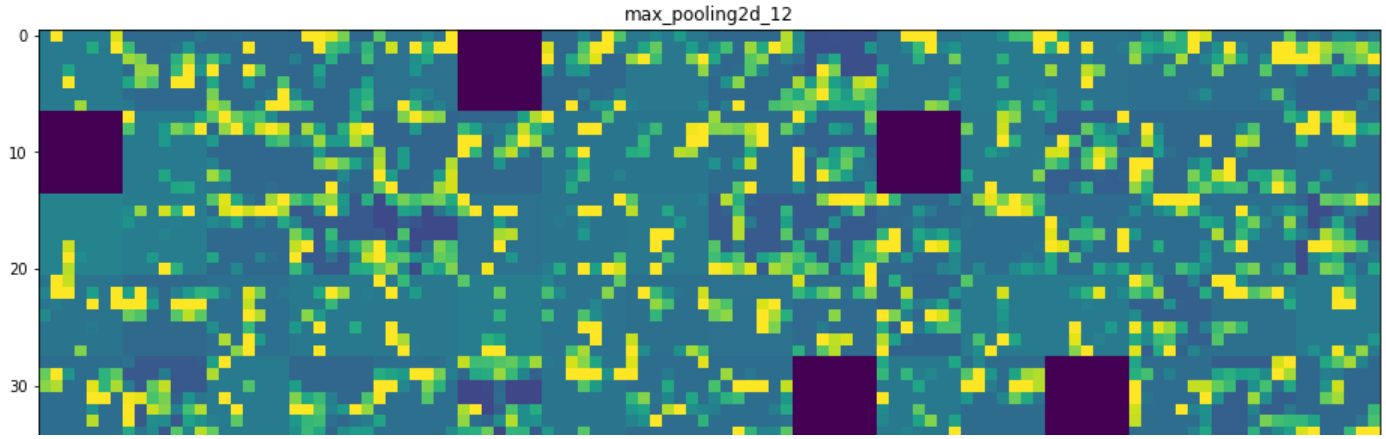

max_pooling2d_12
0
10
20
30
40
50


max_pooling2d_12
0
10
20
30
40
50
0
20
40
60
80
100


max_pooling2d_12
0
10


max_pooling2d_12
0
10
20
30
40
50
0
20
40
60
80
100


max_pooling2d_12
0
10
20
30
40
50
0
20
40
60
80
100



In [104]:
```
# ================================================================================
# F                                                                              |
# Visualize convolutional filters: get the gradient of the loss with regard     |
# to the input, apply stochastic gradient descent, include a code for filter    |
# visualization and generate a grid of all filter response patterns in a layer|
# ================================================================================
```

In [103]:
```
# ------------------------------------------------
# DEFINING THE LOSS TENSOR FOR FILTER VISUALATIZATION |
# ------------------------------------------------

model = VGG16(weights='imagenet', include_top=False)
layer_name = 'block3_conv1'
filter_index = 0

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

In [105]:
```
# ------------------------------------------------------------
# OBTAINING THE GRADIENT OF THE LOSS WITH REGARD TO THE INPUT |
# ------------------------------------------------------------

grads = K.gradients(loss, model.input)[0]
# the call to gradients returns a list of tensors (of size 1 in this case)
# keeps only the 1st elements which is a tensor
```

In [106]:
```
# ----------------------------
# GRADIENT-NORMALIZATION TRICK |
# ----------------------------

grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)
# add 1e-5 before dividing to avoid accidentally dividing by 0
```

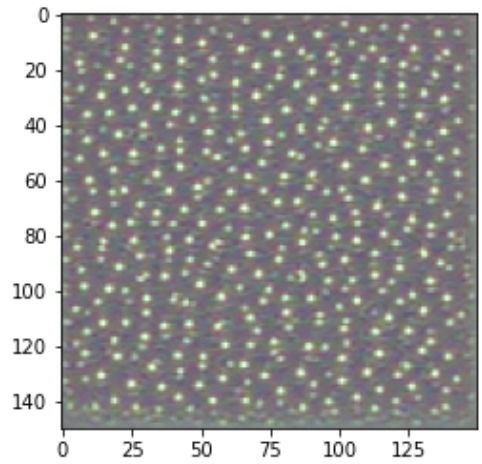
In [107]:
```
# ------------------------------------------------
# FETCHING NUMPY OUTPUT VALUES GIVEN NUMPY INPUT VALUES |
# ------------------------------------------------

iterate = K.function([model.input], [loss, grads])

loss_value, grads_value = iterate([np.zeros((1, 150, 150, 3))])
```

In [109]:
```
# ---------------------------------------------------
# LOSS MAXIMIZATION VIA STOCHASTIC GRADIENT DESCENT |
# ---------------------------------------------------

input_img_data = np.random.random((1, 150, 150, 3)) * 20 + 128.
# starts from a grey image with some noise

step = 1.  # magnitude of each gradient update
for i in range(40):  # runs gradient ascent for 40 steps
    loss_value, grads_value = iterate([input_img_data])
    # computes the loss value and gradient value

    input_img_data += grads_value * step
    # adjusts the input image in the direction that maximizes the loss
```

In [110]:
```
# ---------------------------------------------------------
# UTILITY FUNCTION TO CONVERT A TENSOR INTO A VALID IMAGE |
# ---------------------------------------------------------

def deprocess_image(x):
    # Normalizes the tensorL centers on 0, ensure that std is 0.1
    x -= x.mean()
    x /= (x.std() + 1e-5)
    x *= 0.1

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

    # Converts to an RBG array
    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

In [112]:
```
# ----------------------------------------
# FUNCTION TO GENERATE FILTER VISUALIZATIONS |
# ----------------------------------------

def generate_pattern(layer_name, filter_index, size=150):
    # Builds a loss function that maximizes that activation of the nth
    # filter of the layer under consideration
    layer_output = model.get_layer(layer_name).output
    loss = K.mean(layer_output[:, :, :, filter_index])

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

    # Normalization trickL normalizes the gradient
    grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)

    # Returns the loss and grads given the input picture
    iterate = K.function([model.input], [loss, grads])

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

    # Runs gradient ascent for 40 steps
    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)

plt.imshow(generate_pattern('block3_conv1', 0))
```

Out[112]: <matplotlib.image.AxesImage at 0x264d06e9bc8>

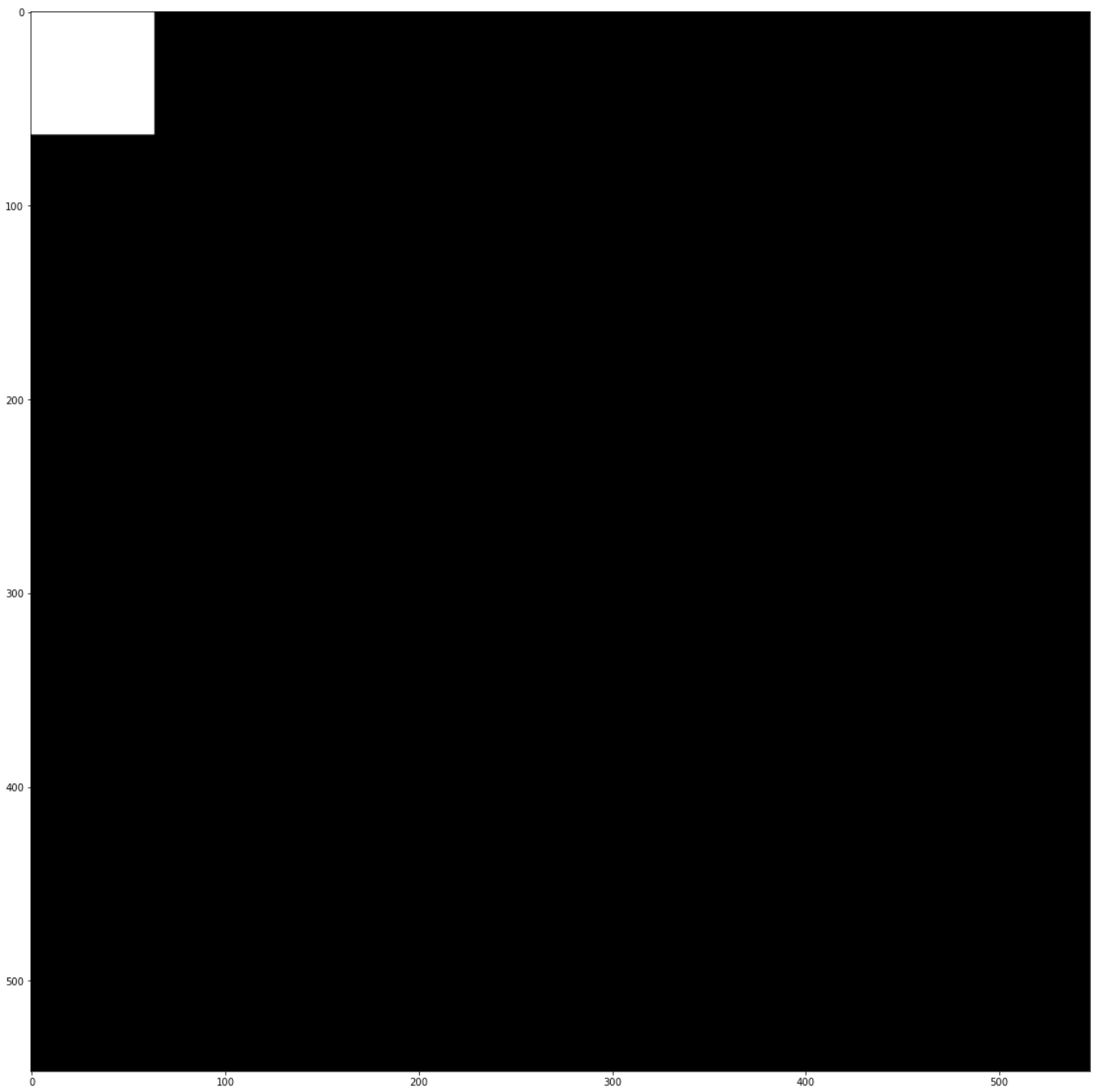


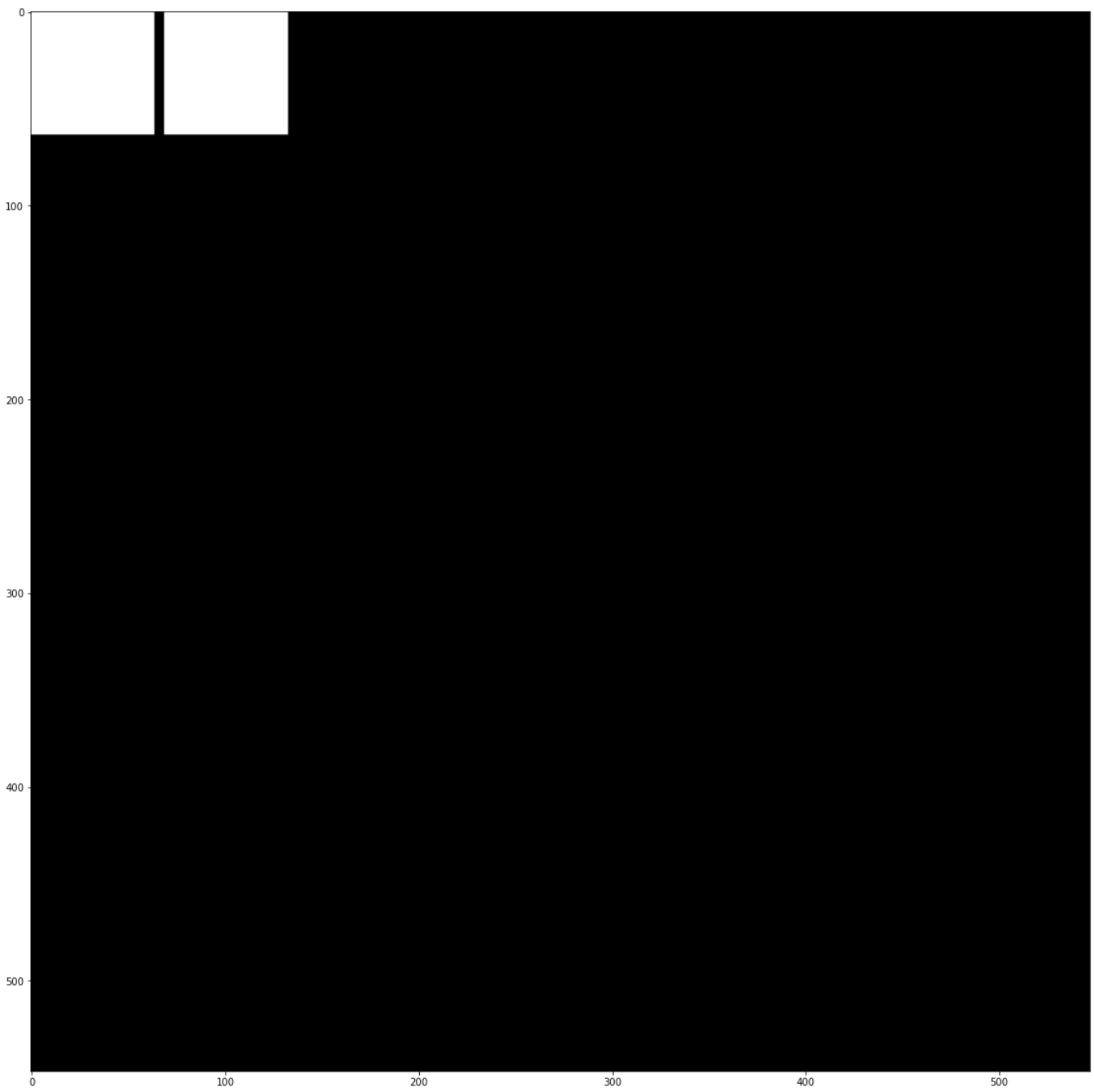
In [114]:
```python
# -----------------------------------------------------------
# GENERATING A GRID OF ALL FILTER RESPONSE PATTERNS IN A LAYER |
# -----------------------------------------------------------

layer_name = 'block1_conv1'
size = 64
margin = 5

# Empty (black) image to store results
results = np.zeros((8 * size + 7 * margin, 8 * size + 7 * margin, 3))

for i in range(8):  # Iterate over the rows of the results grid
    for j in range(8):  # iterates over the columns of the results grid
        # Generates the pattern for filter i+(j*8) in layer_name
        filter_img = generate_pattern(layer_name, i + (j * 8), size=size)

        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, :] = filter_img

        # Displays the results grid
        plt.figure(figsize=(20, 20))
        plt.imshow(results)
```

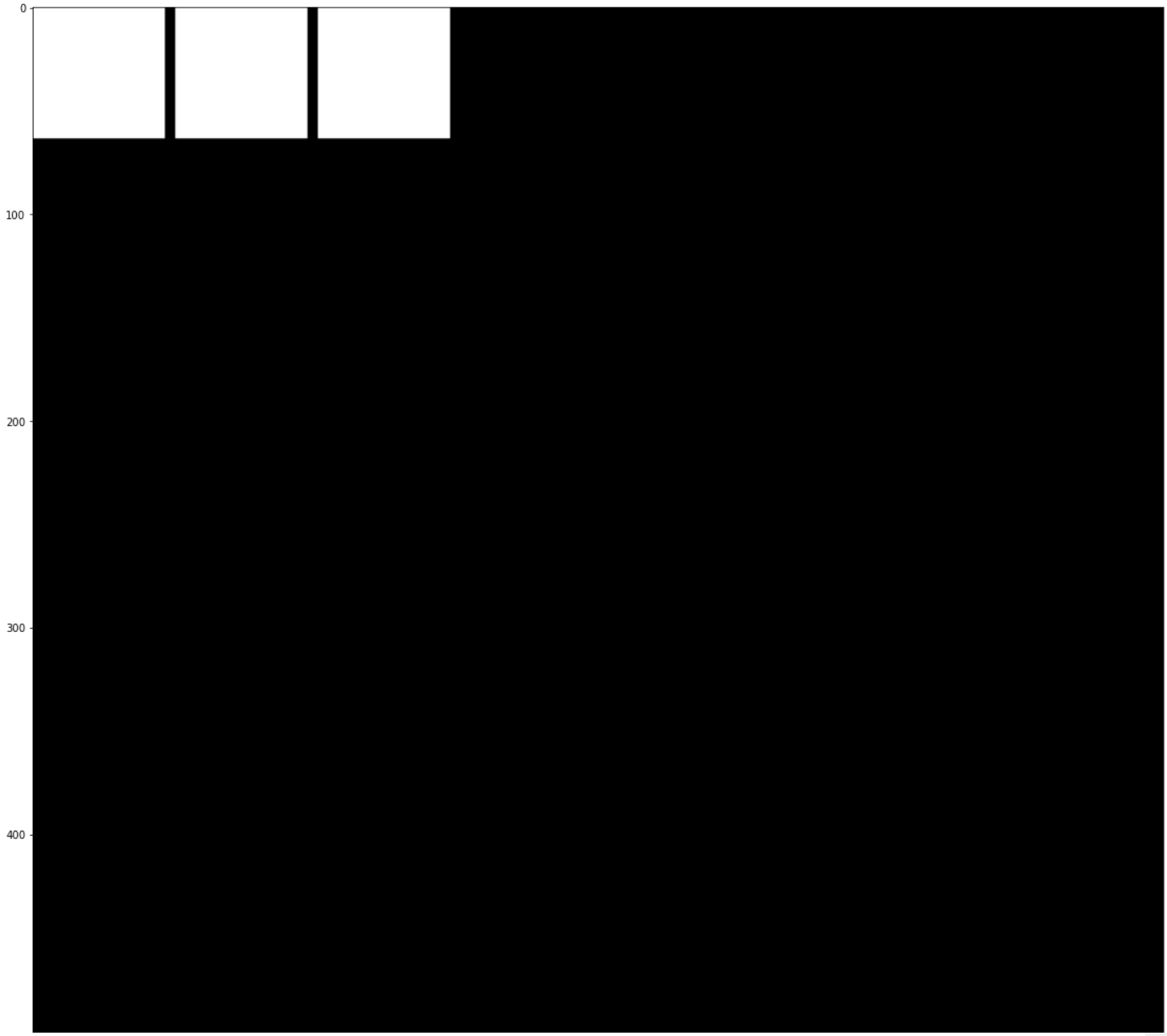
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
```

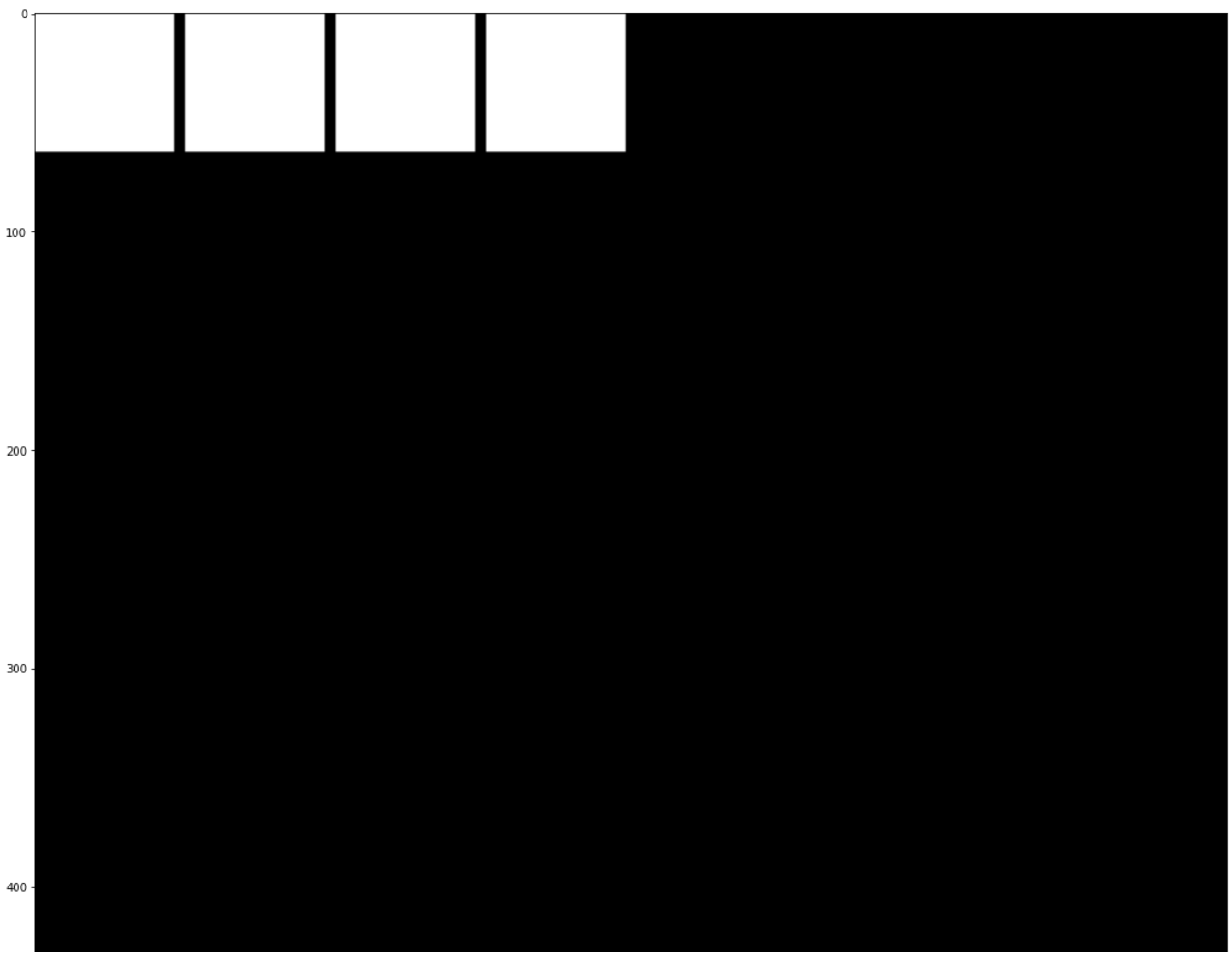
```
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
C:\Users\User\Anaconda3\lib\site-packages\ipykernel_launcher.py:24: RuntimeWarning: More tha
n 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyp
lot.figure`) are retained until explicitly closed and may consume too much memory. (To contr
ol this warning, see the rcParam `figure.max_open_warning`).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
```

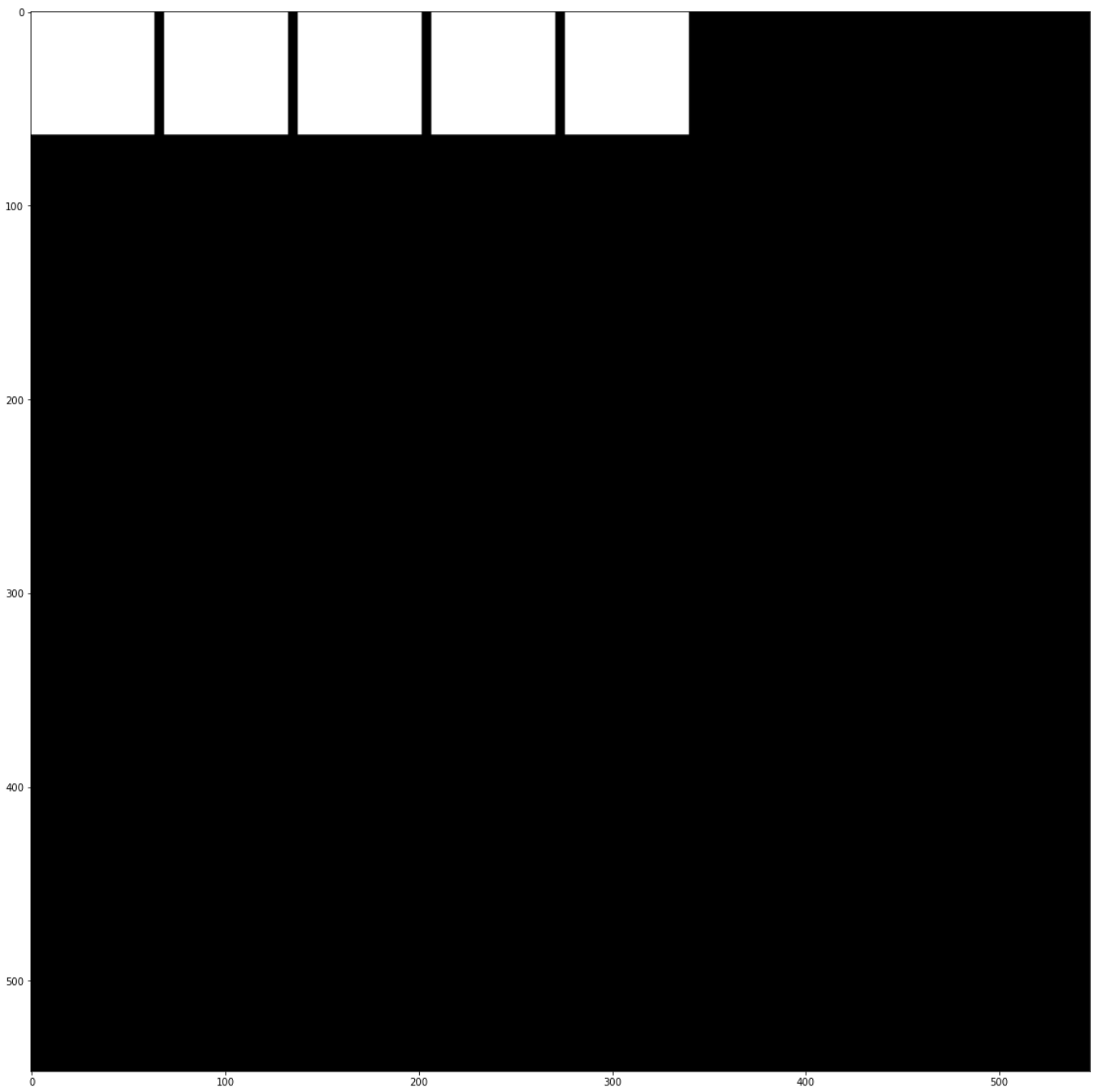
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
5] for integers).
```

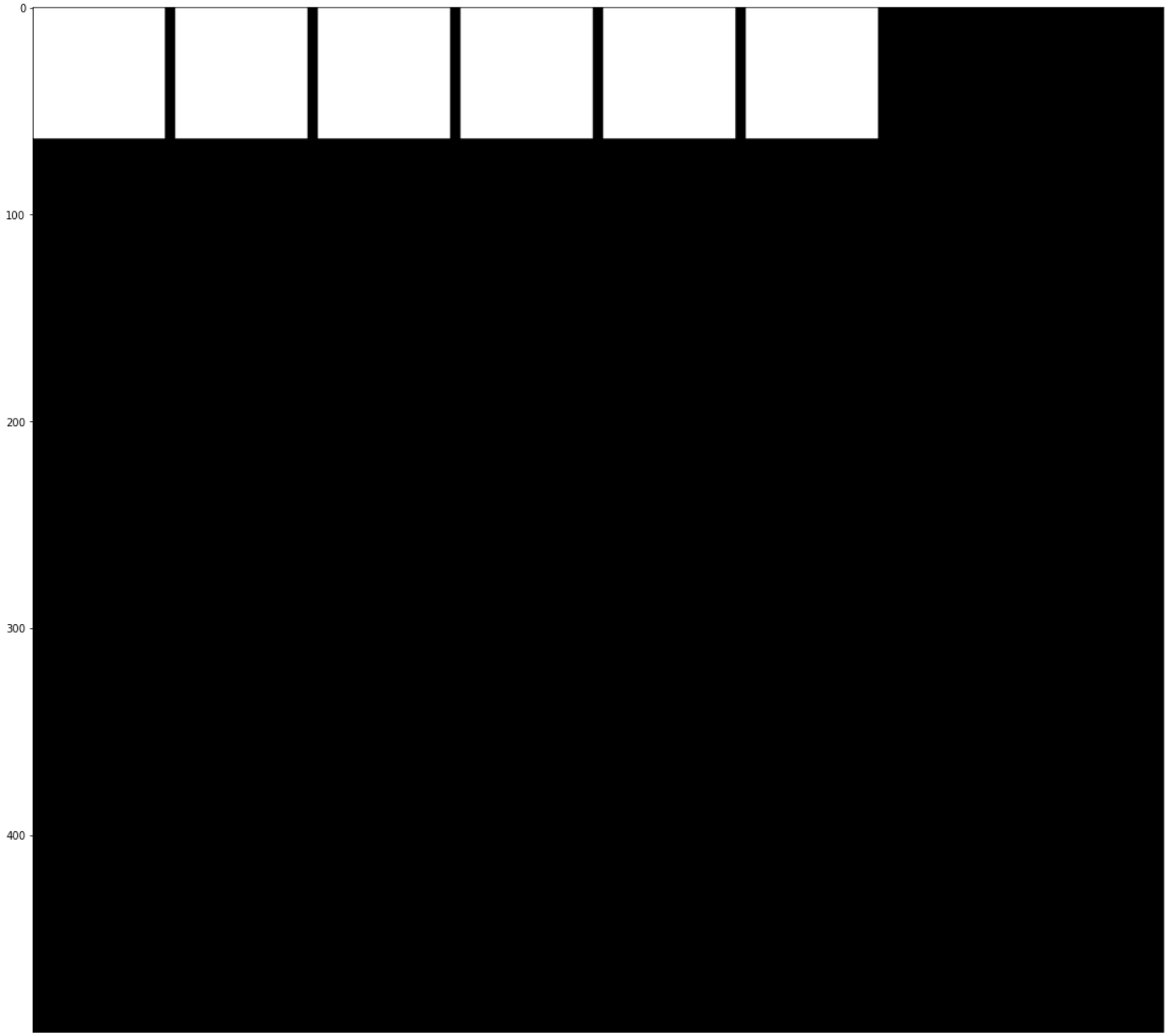

0
100
200
300
400
500
0
100
200
300
400
500

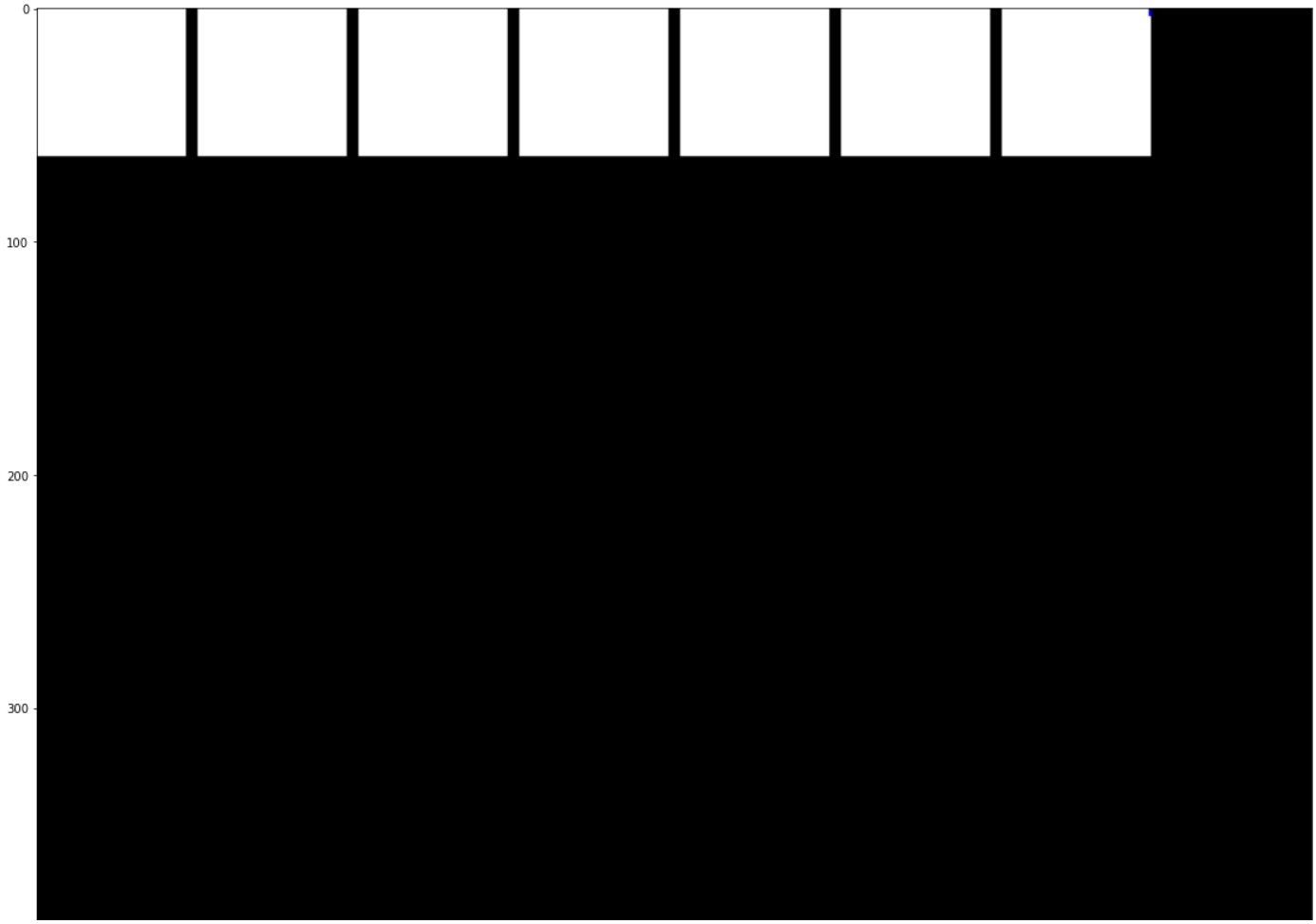

0
100
200
300
400
500
0
100
200
300
400
500

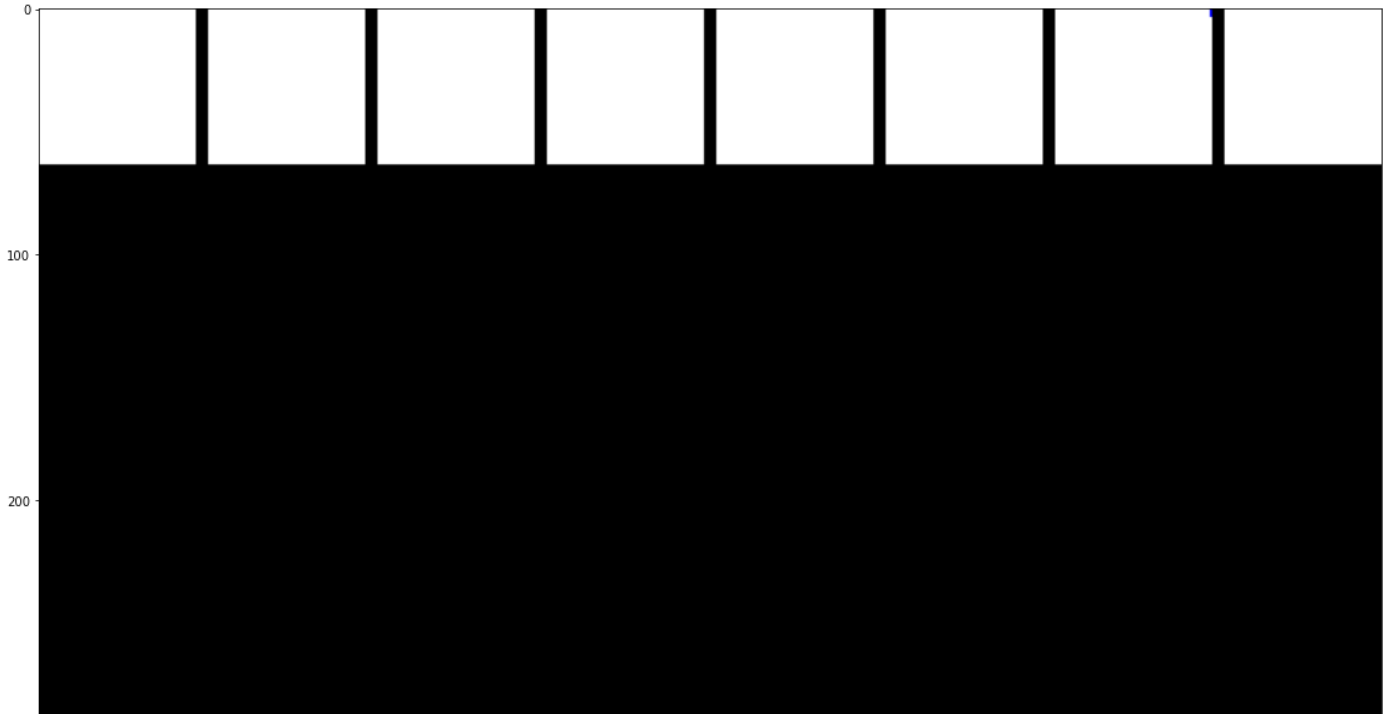

0
100
200
300
400

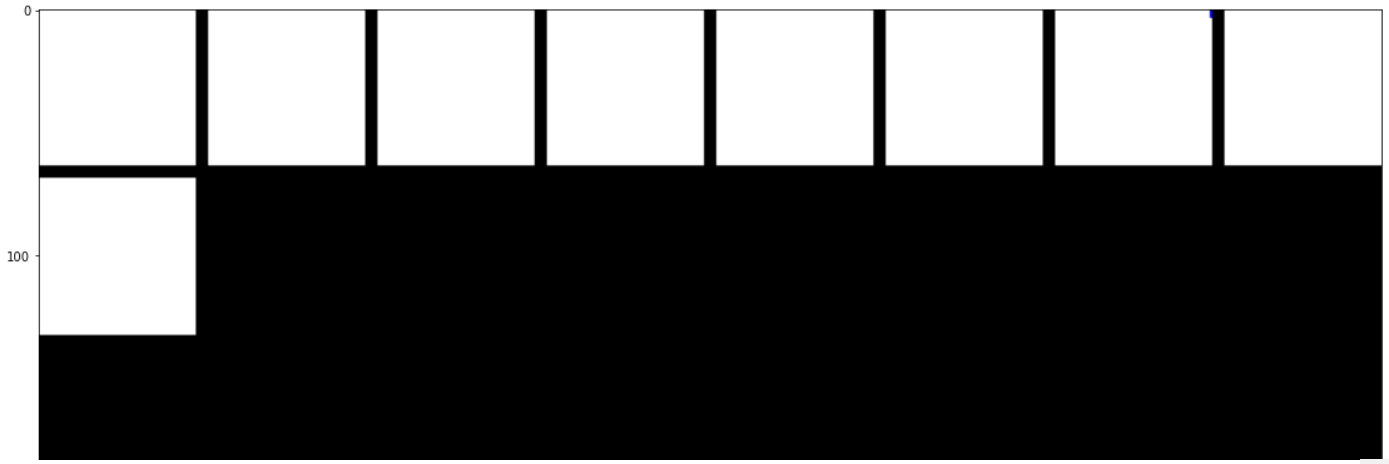

0
100
200
300
400

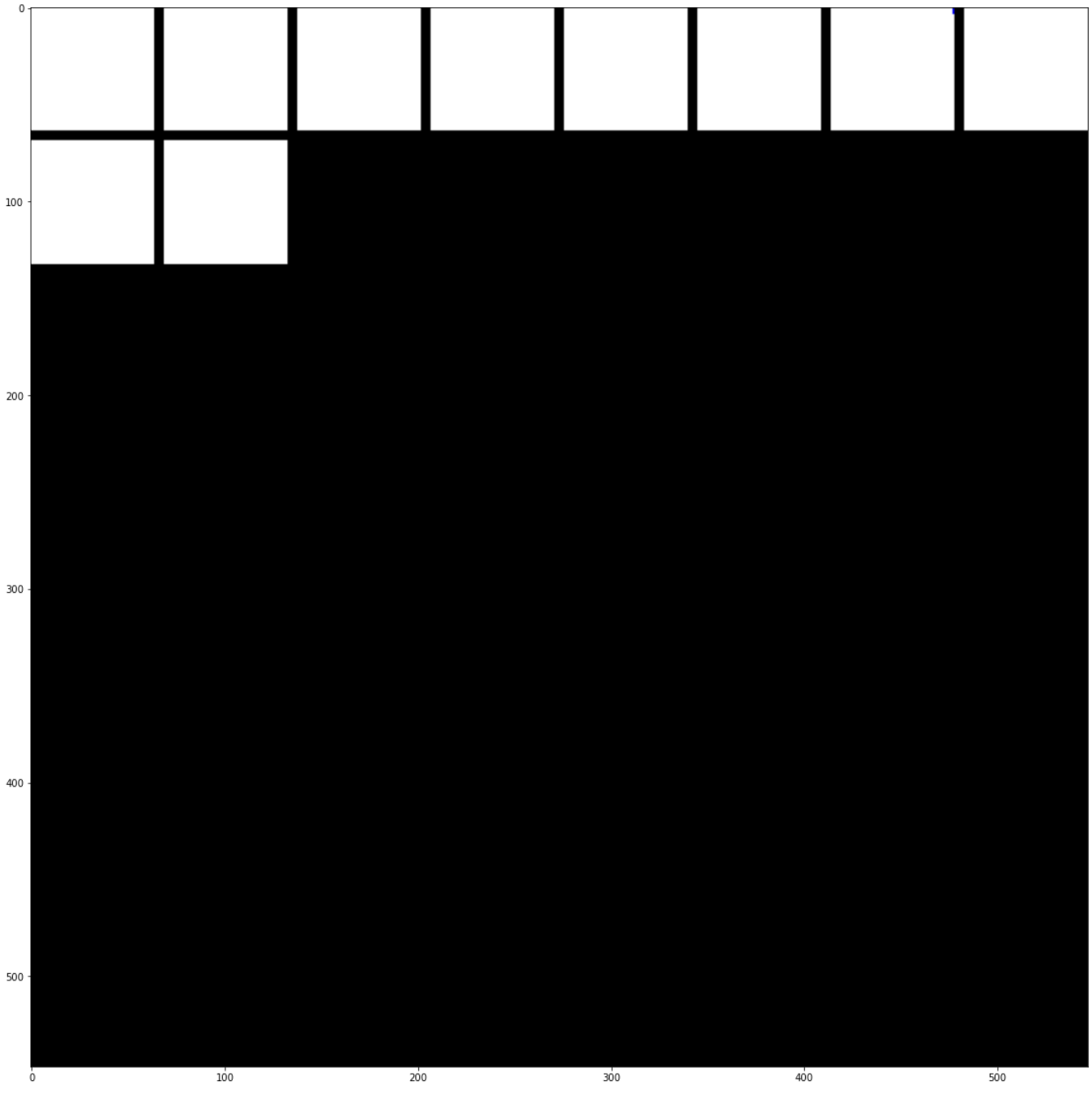

0
100
200
300
400
500
0
100
200
300
400
500

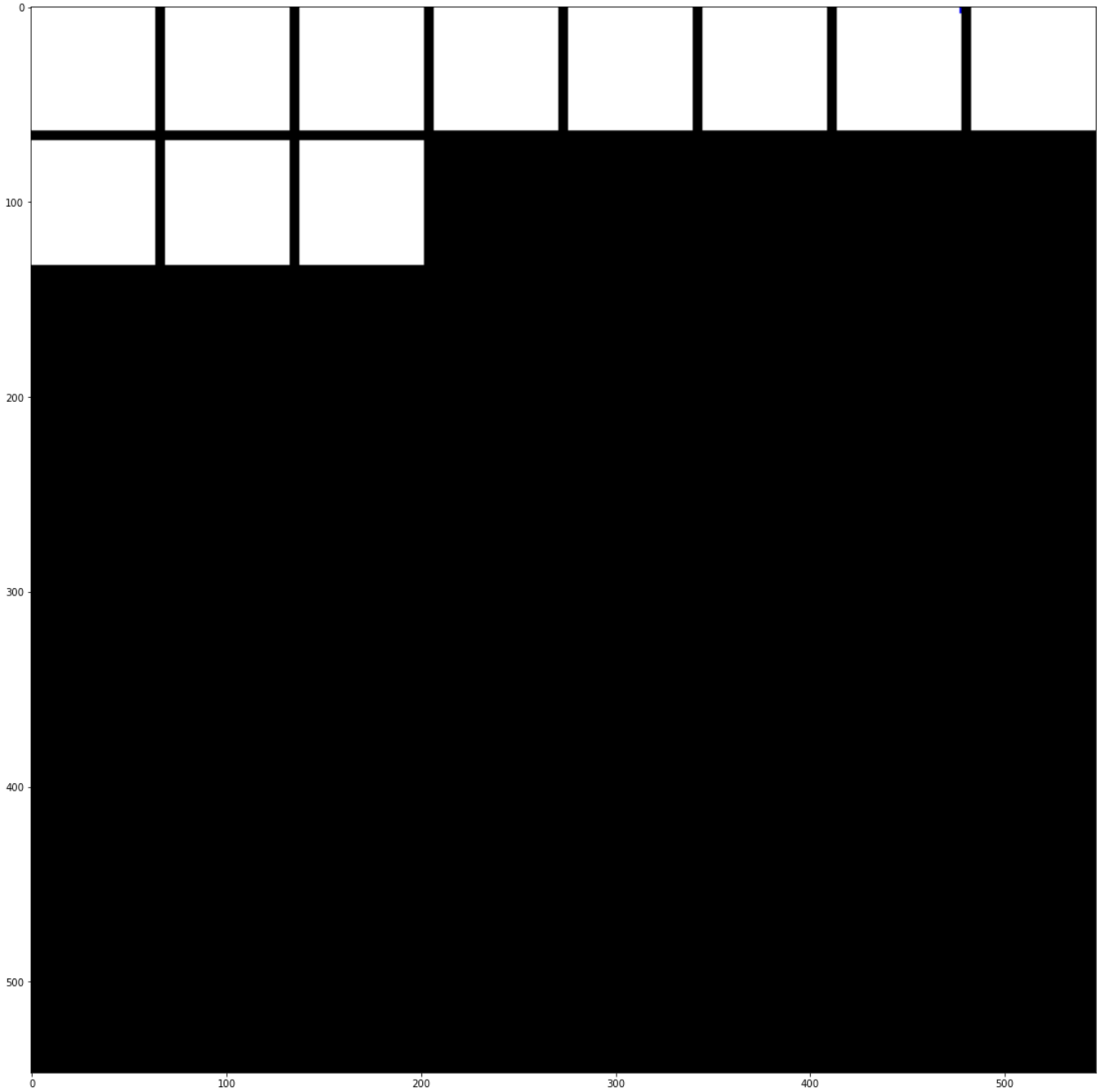

0
100
200
300
400

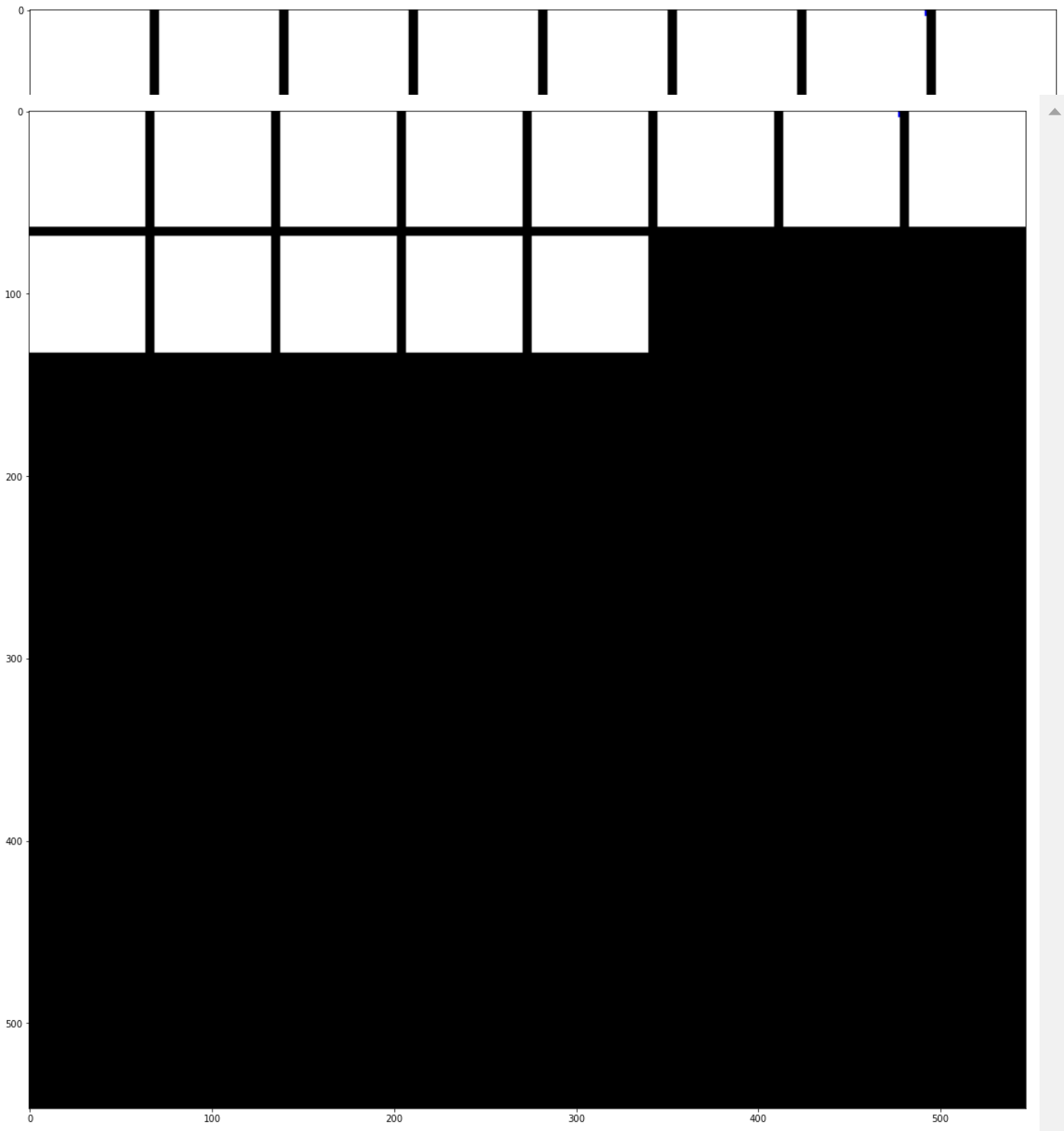

0
100
200
300

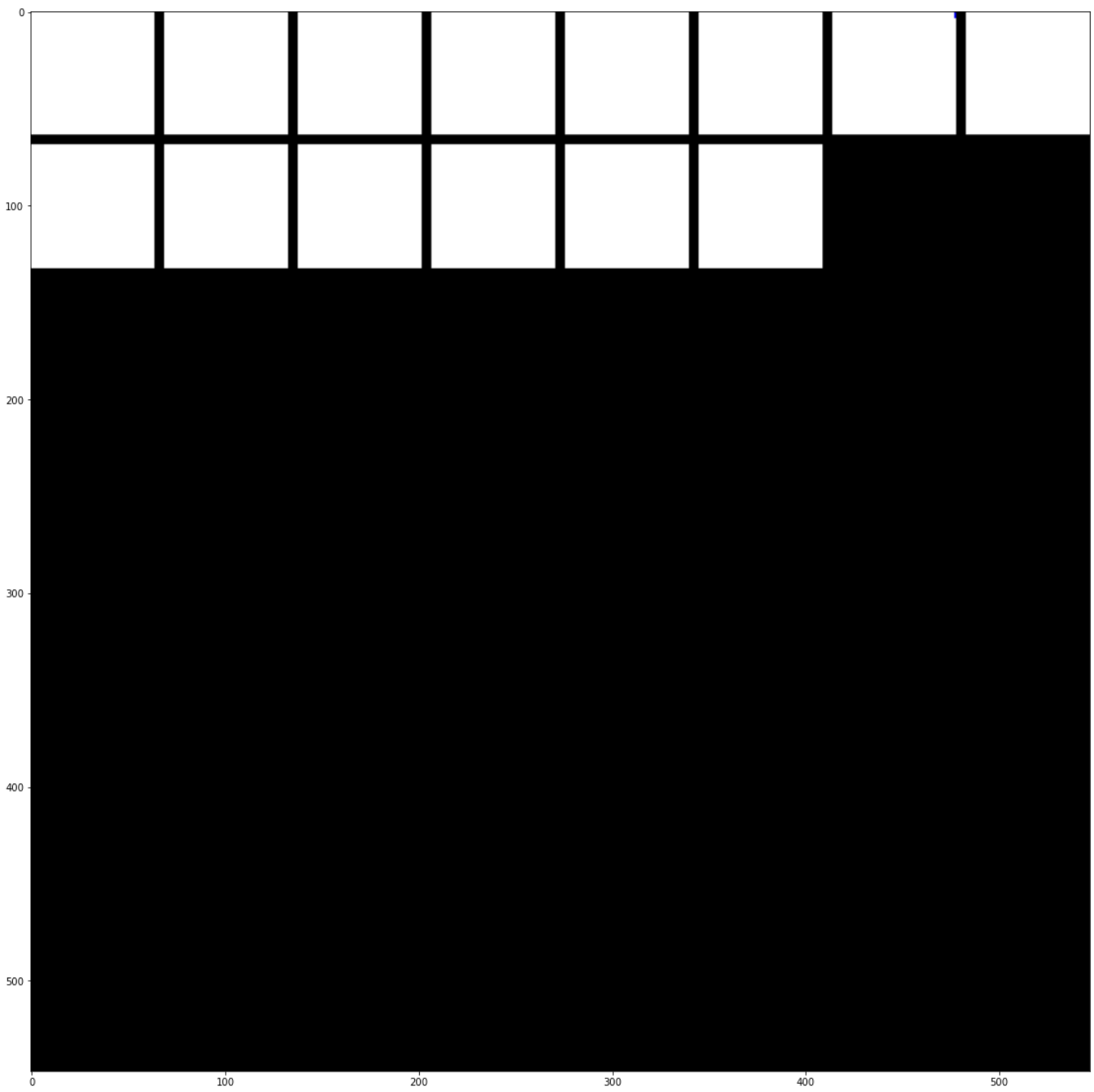

0
100
200

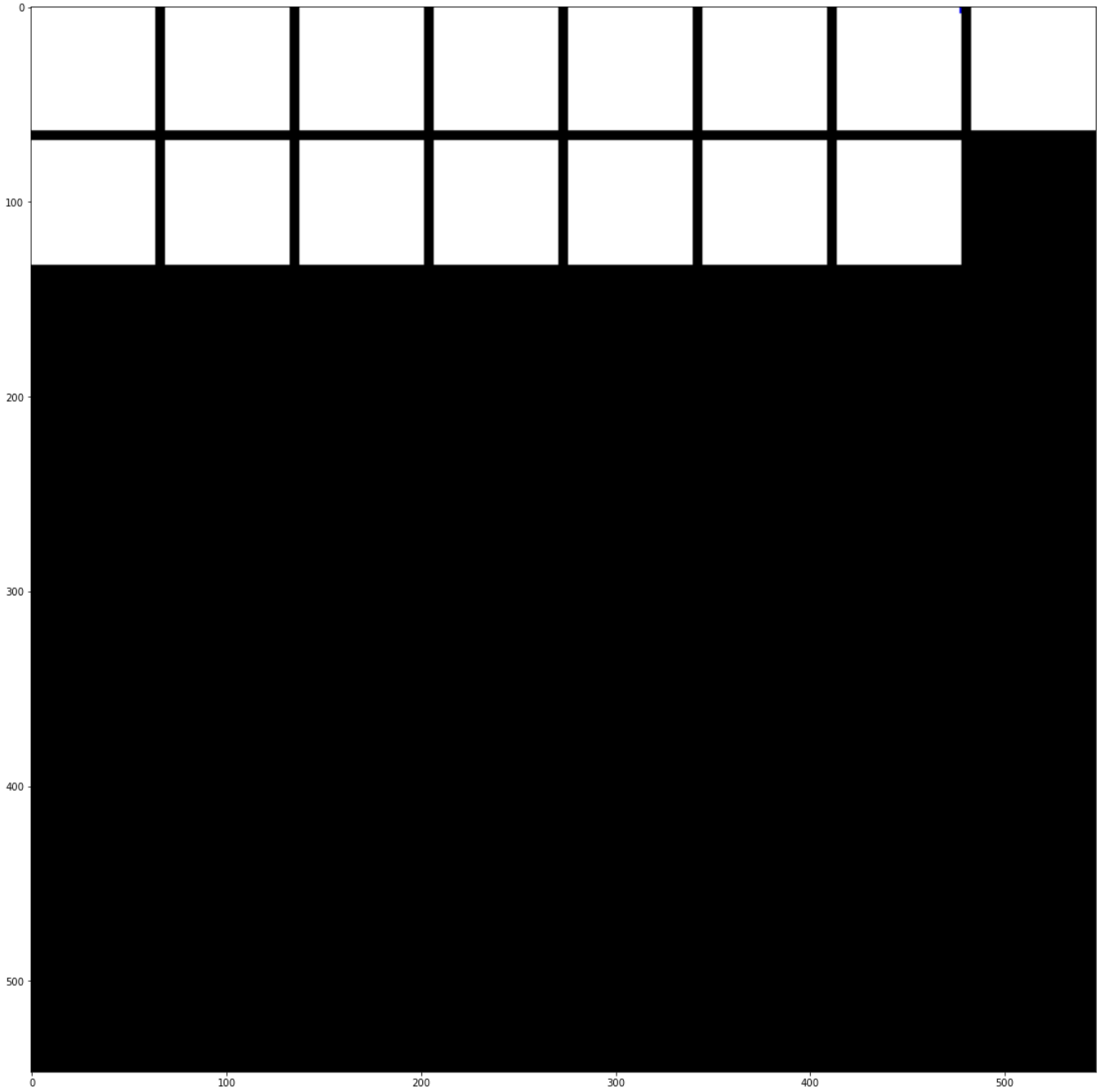

0
100

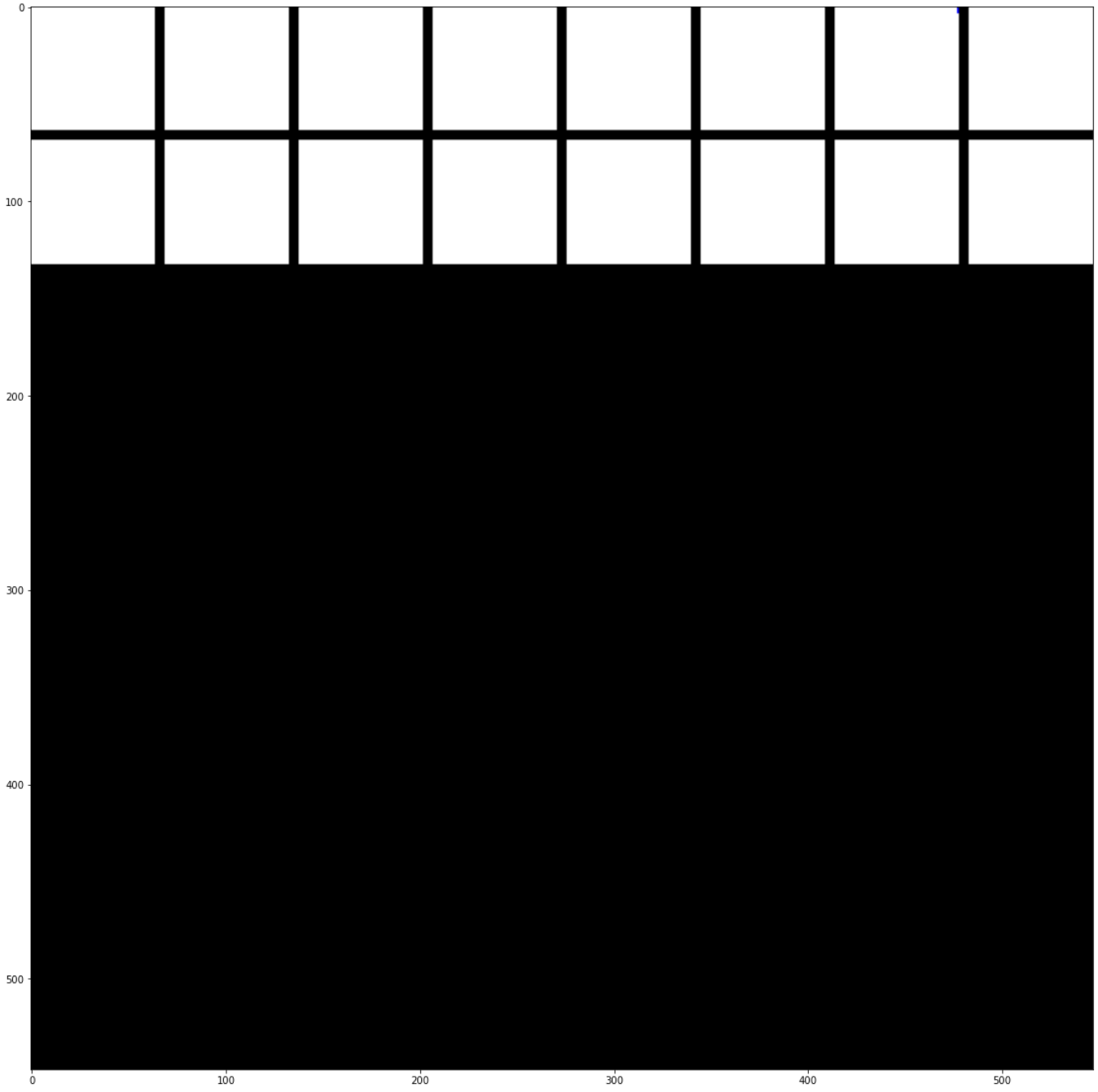

0
100
200
300
400
500
0
100
200
300
400
500

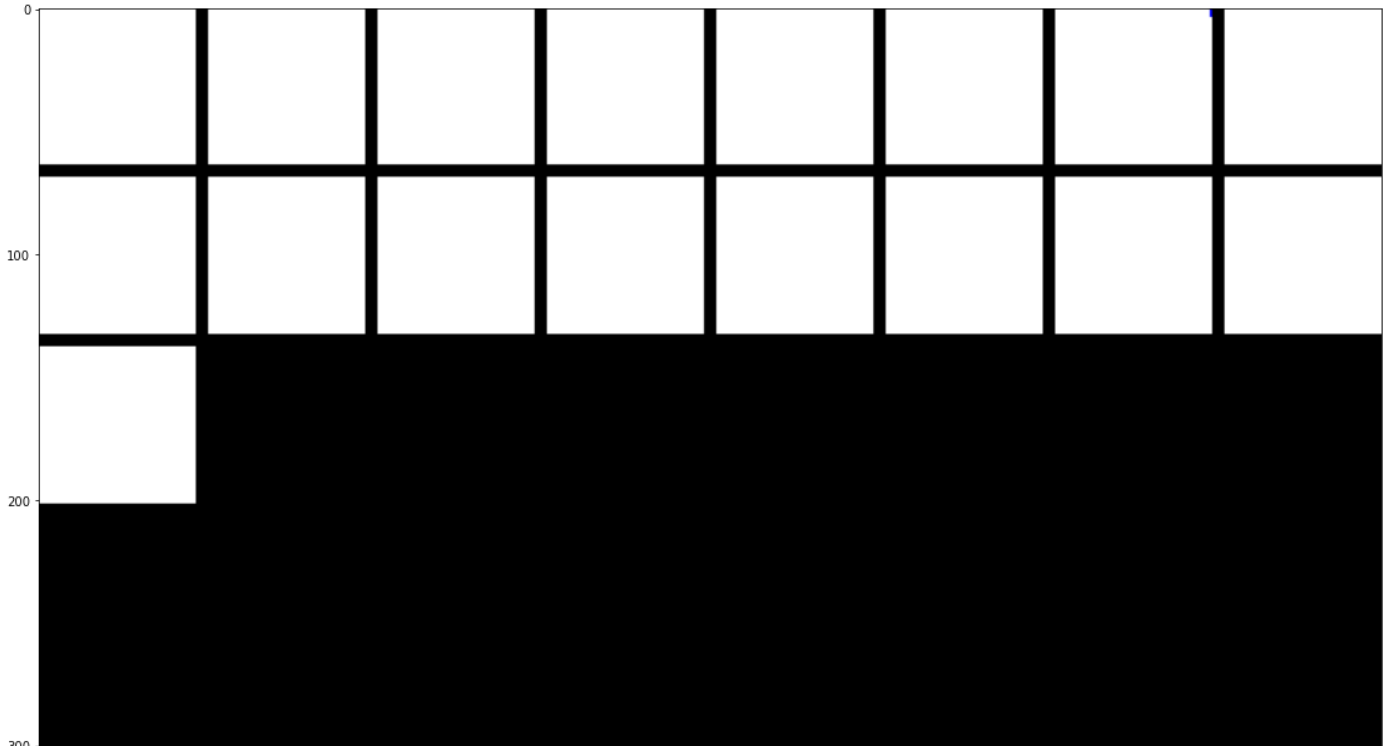

0
100
200
300
400
500
0
100
200
300
400
500

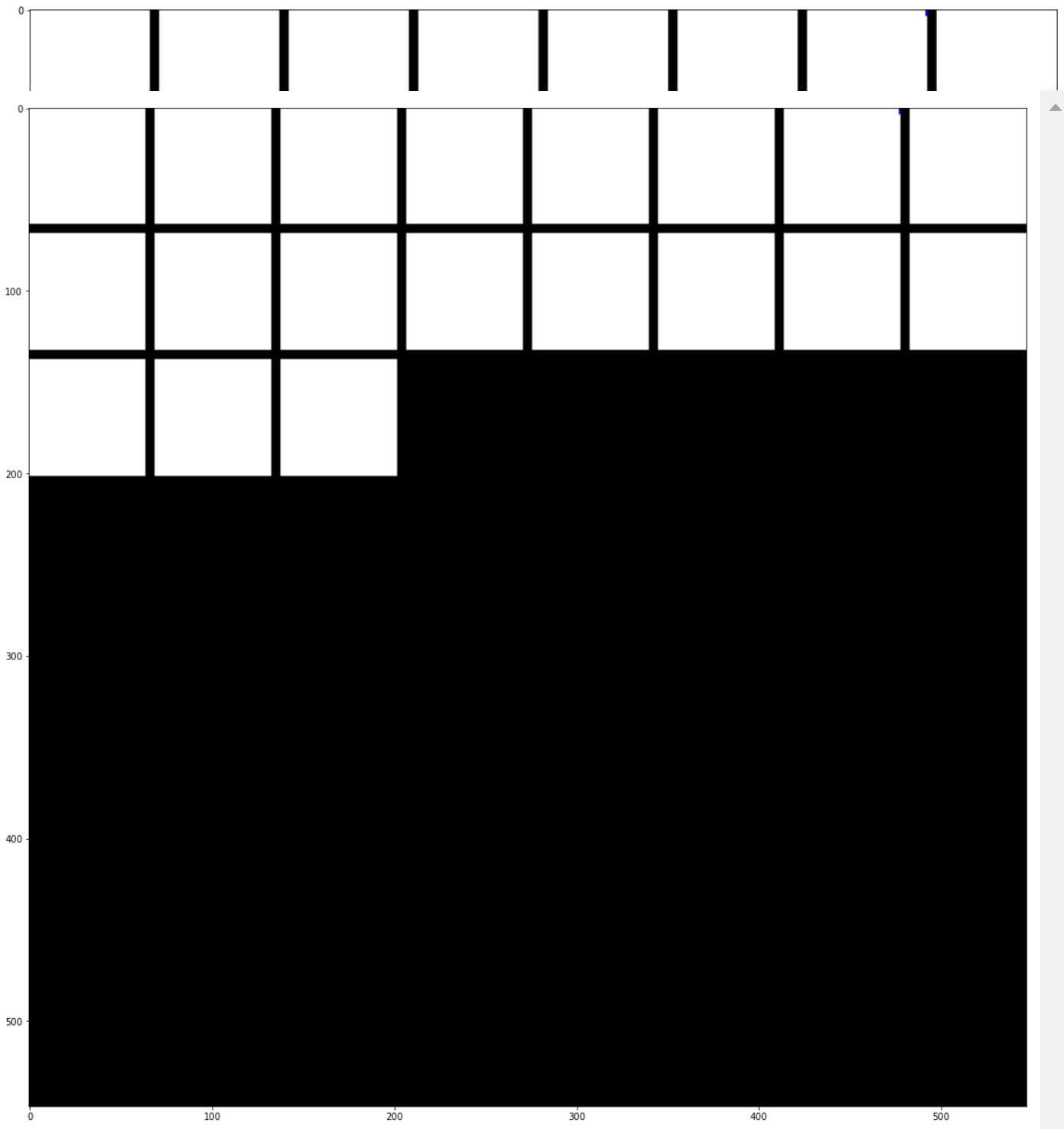

0
0
100
200
300
400
500
0
100
200
300
400
500

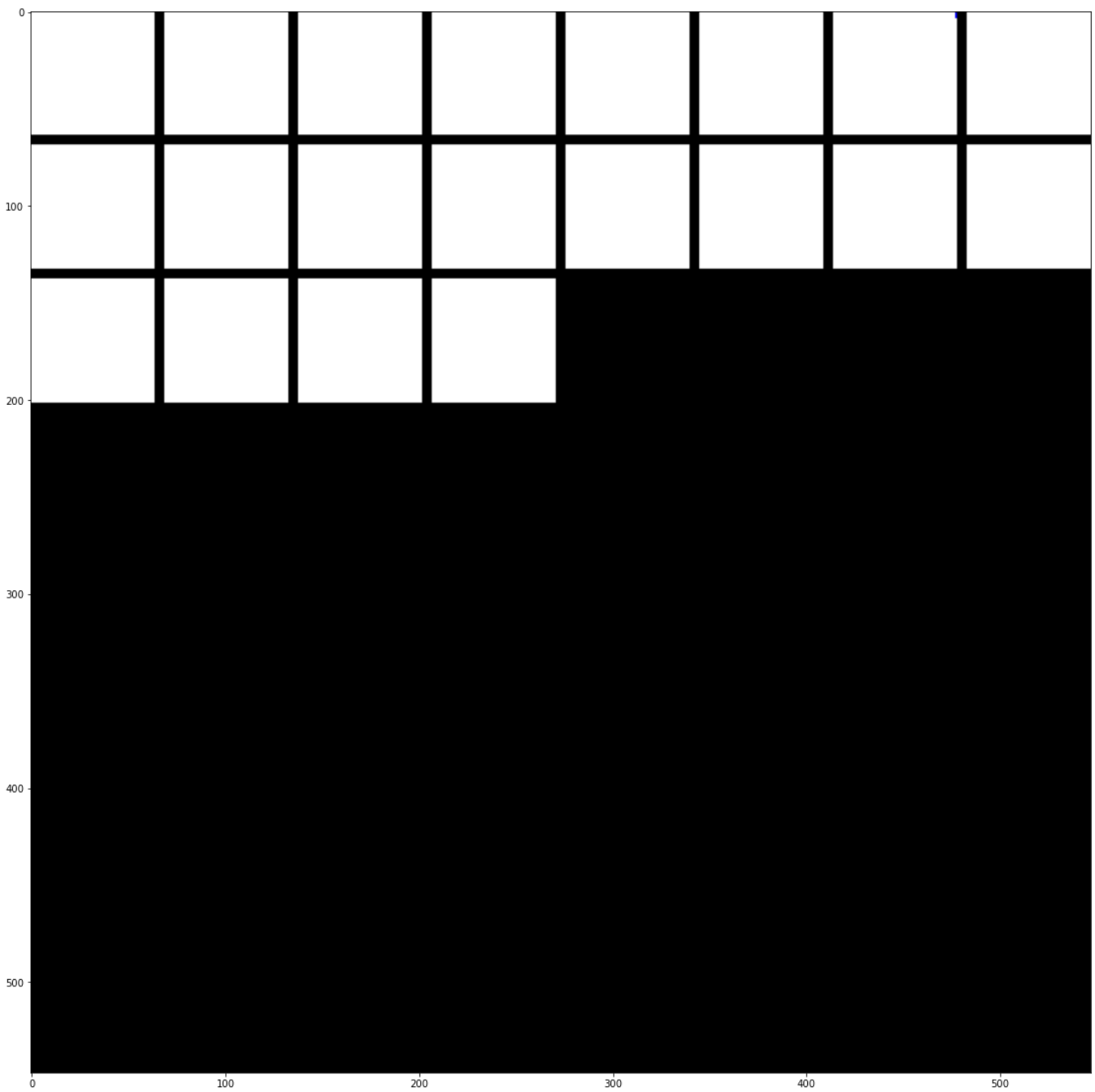

0
100
200
300
400
500
0
100
200
300
400
500

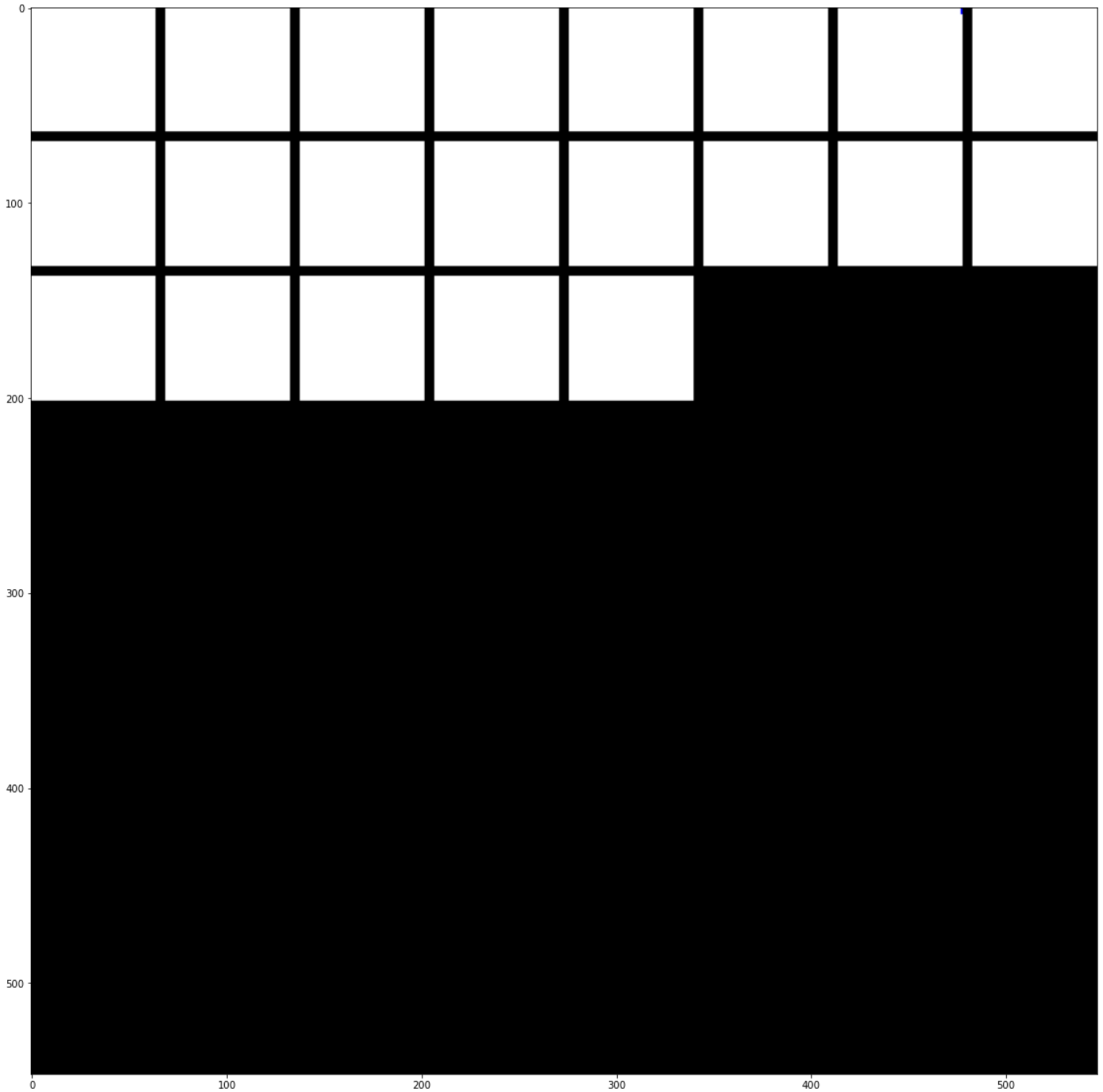

0
100
200
300
400
500
0
100
200
300
400
500

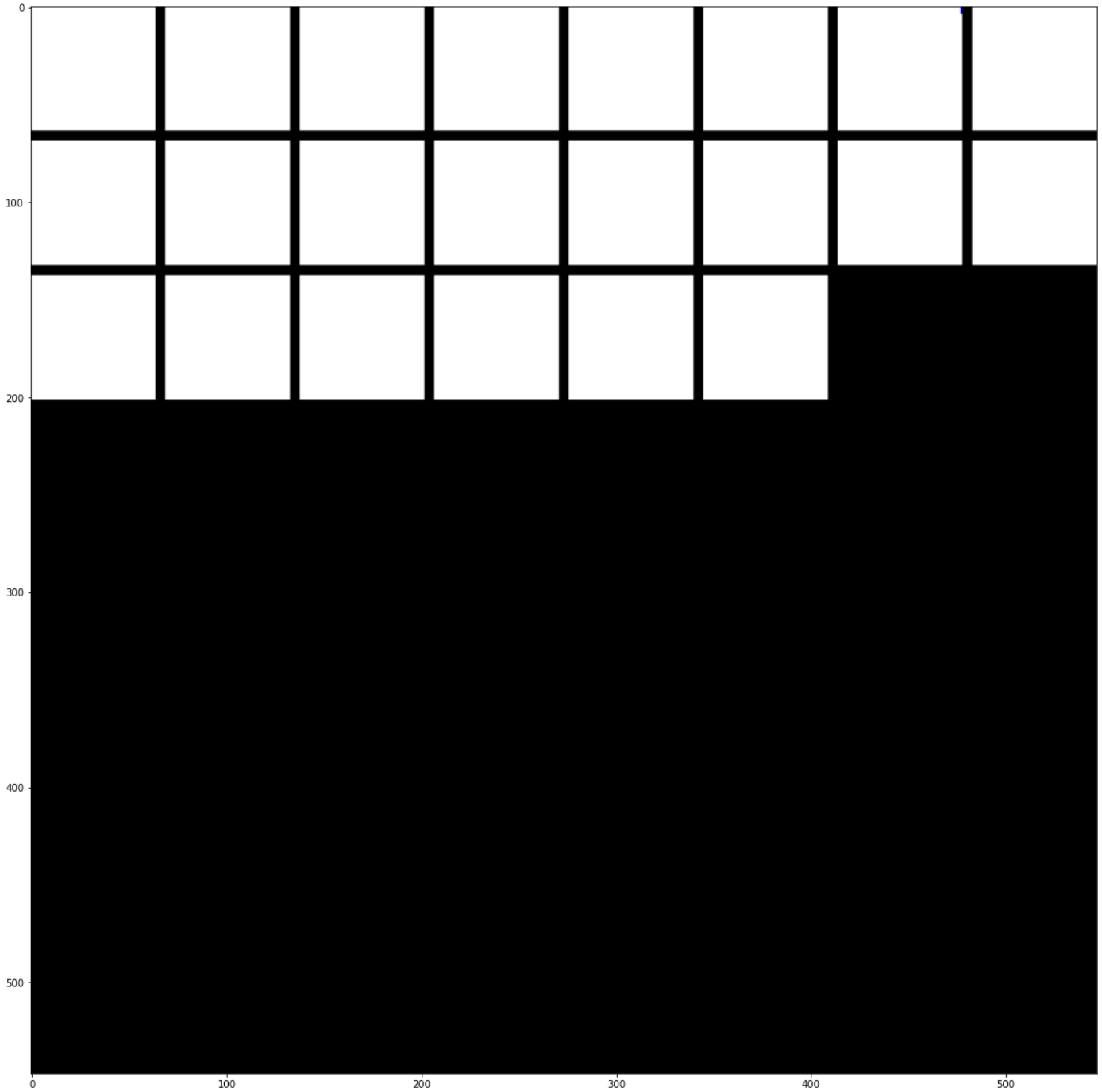

0
100
200
300
400
500
0
100
200
300
400
500

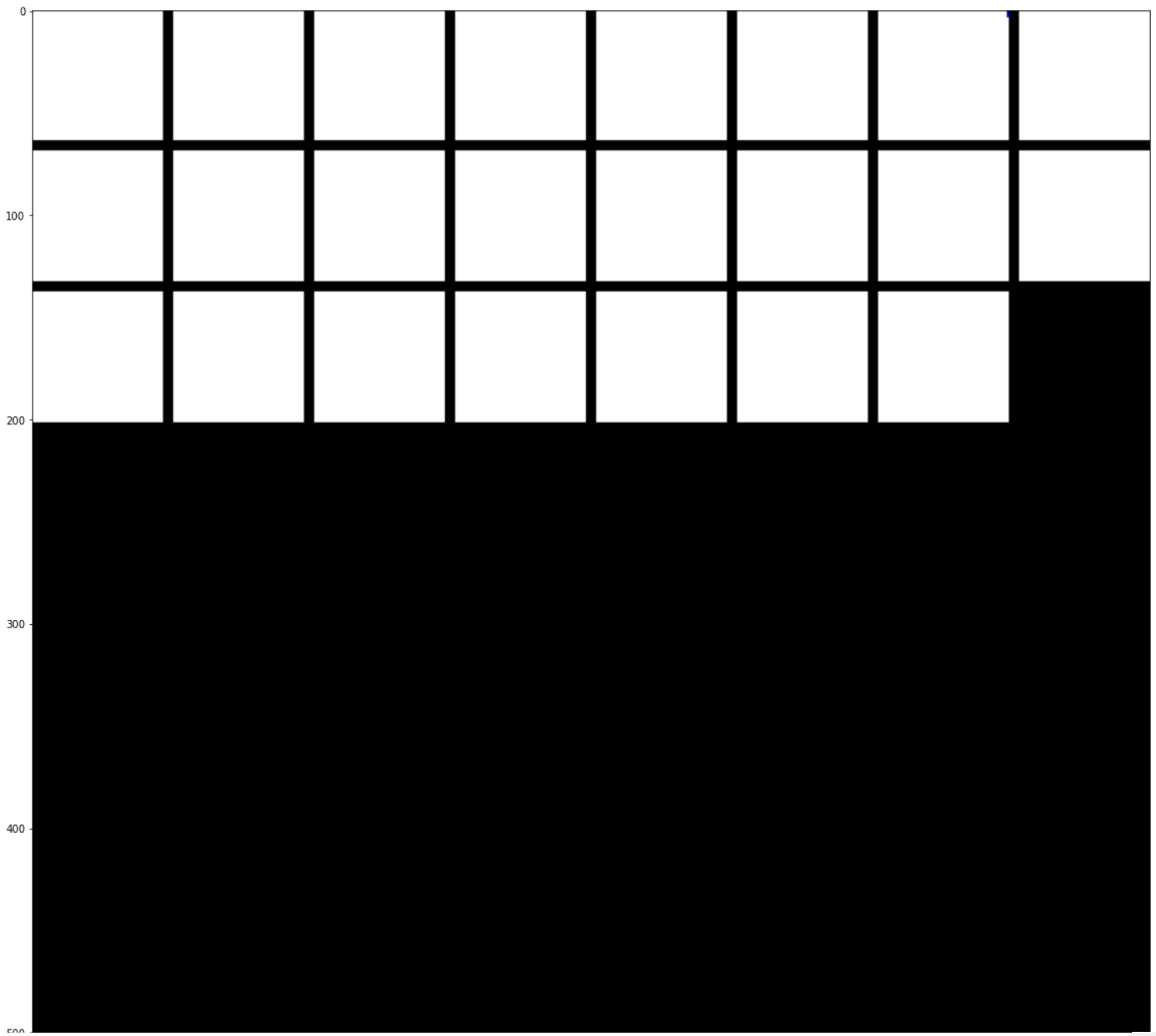

0
100
200

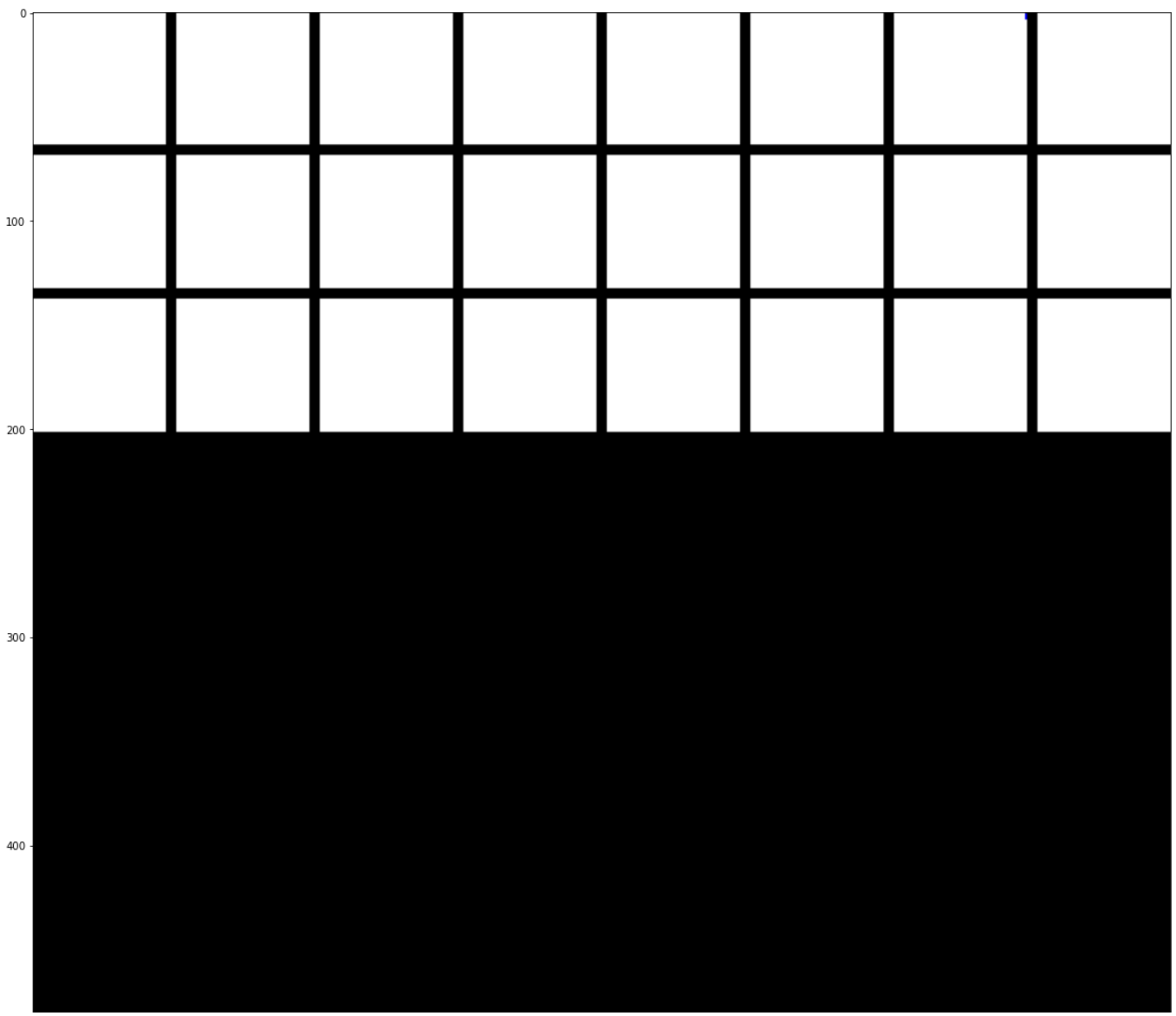

0
0
100
200
300
400
500
0
100
200
300
400
500

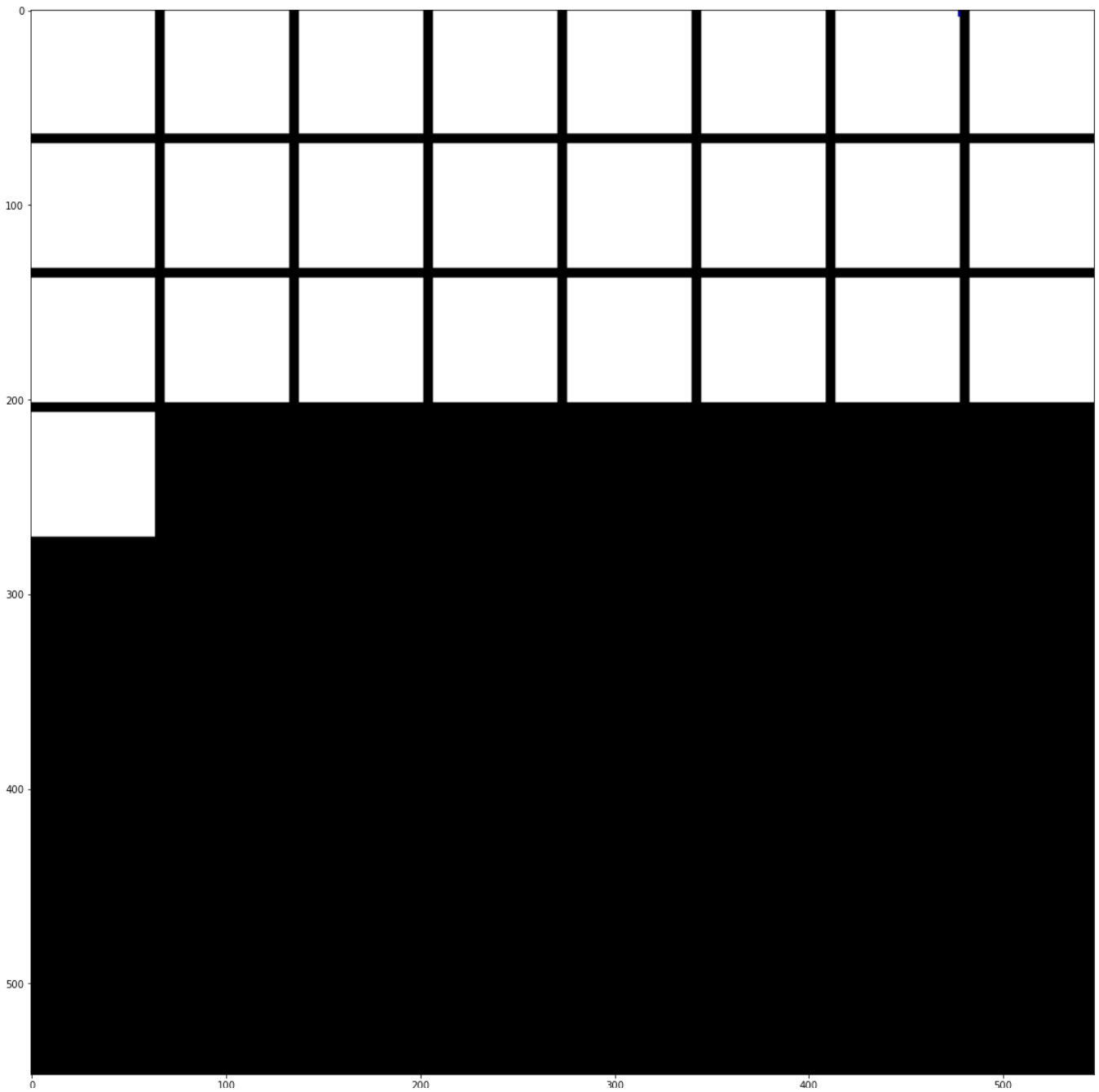

0
100
200
300
400
500
0
100
200
300
400
500

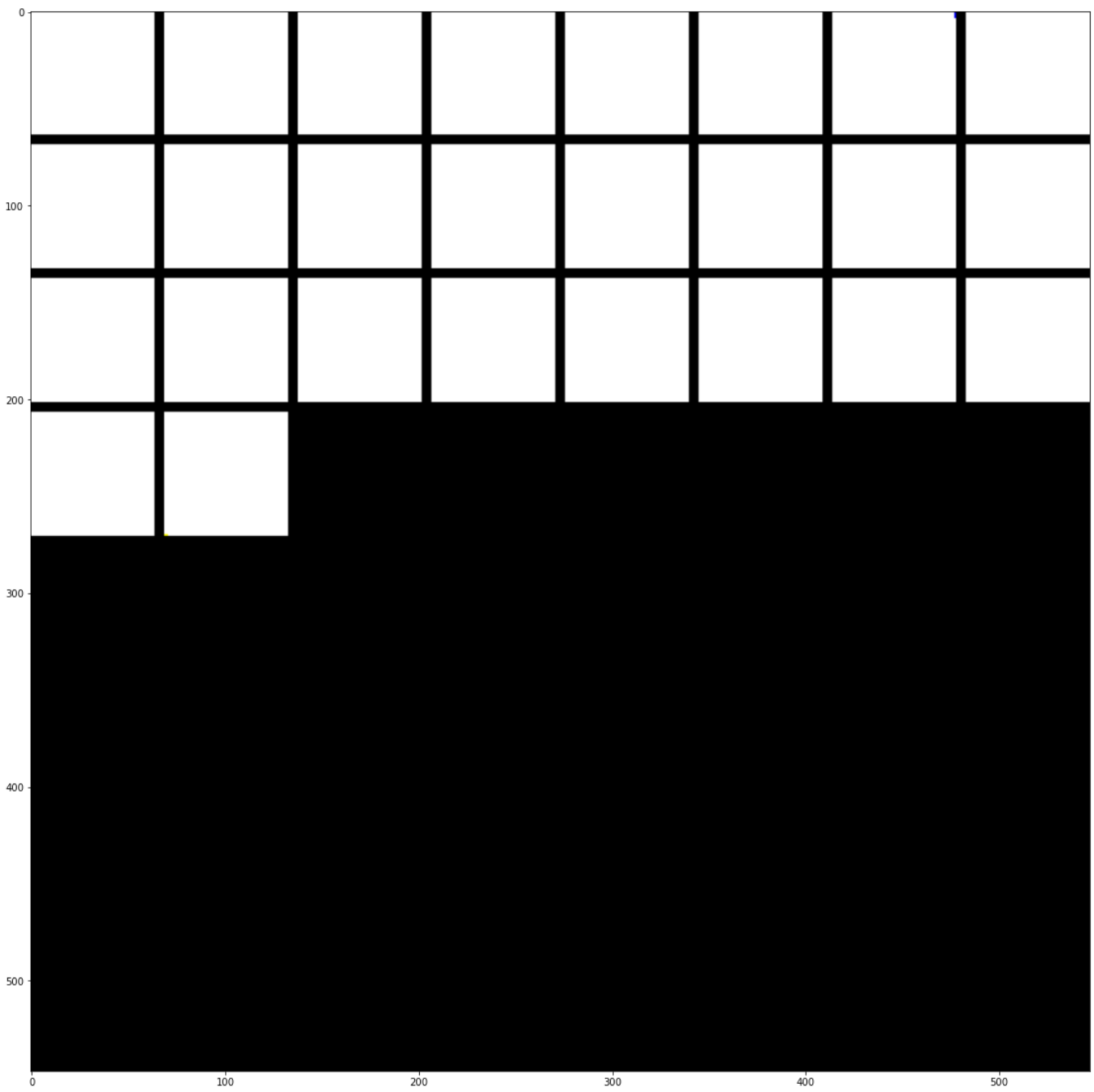

0
100
200
300
400
500
0
100
200
300
400
500

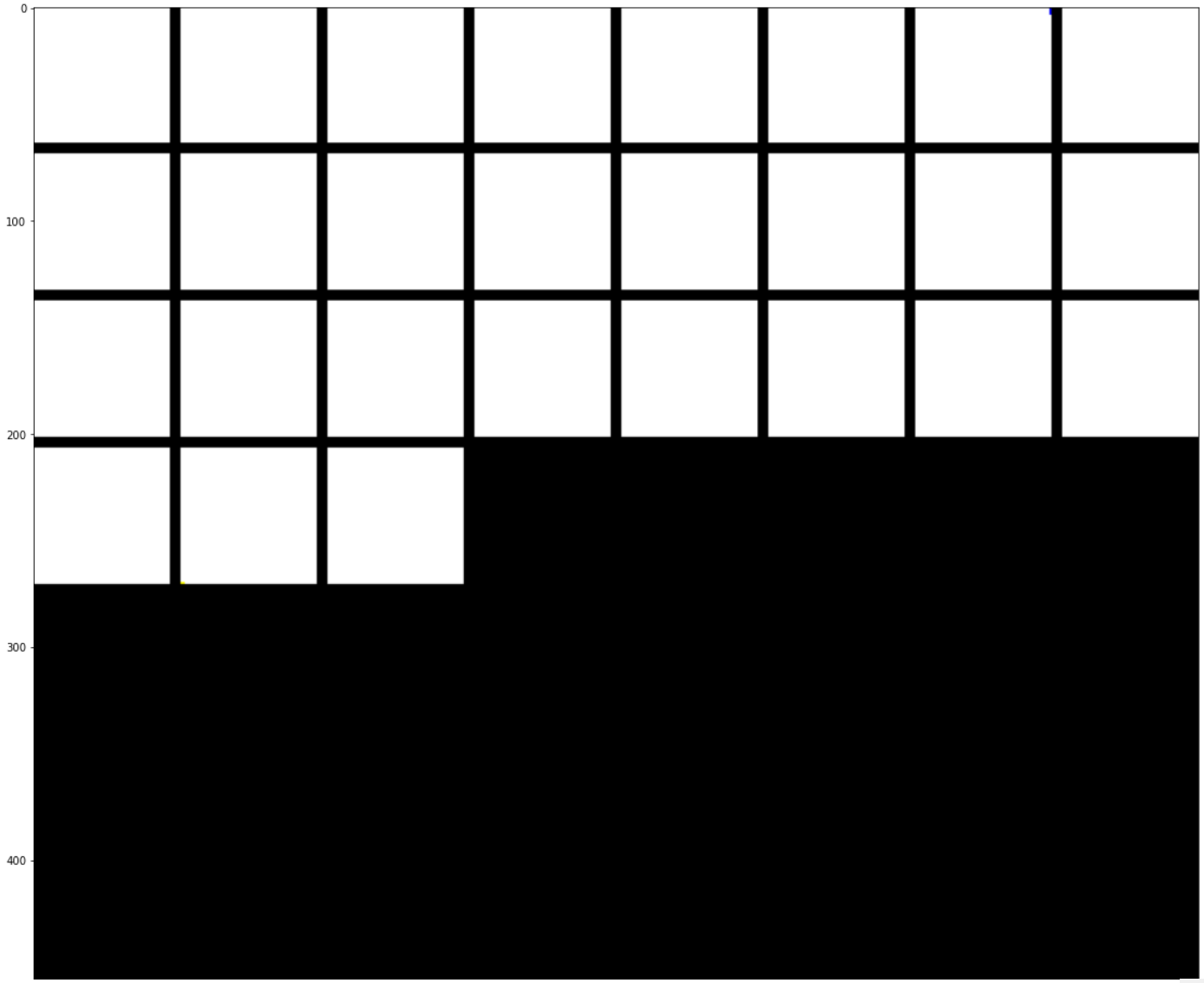

0
100
200
300
400
500
0
100
200
300
400
500

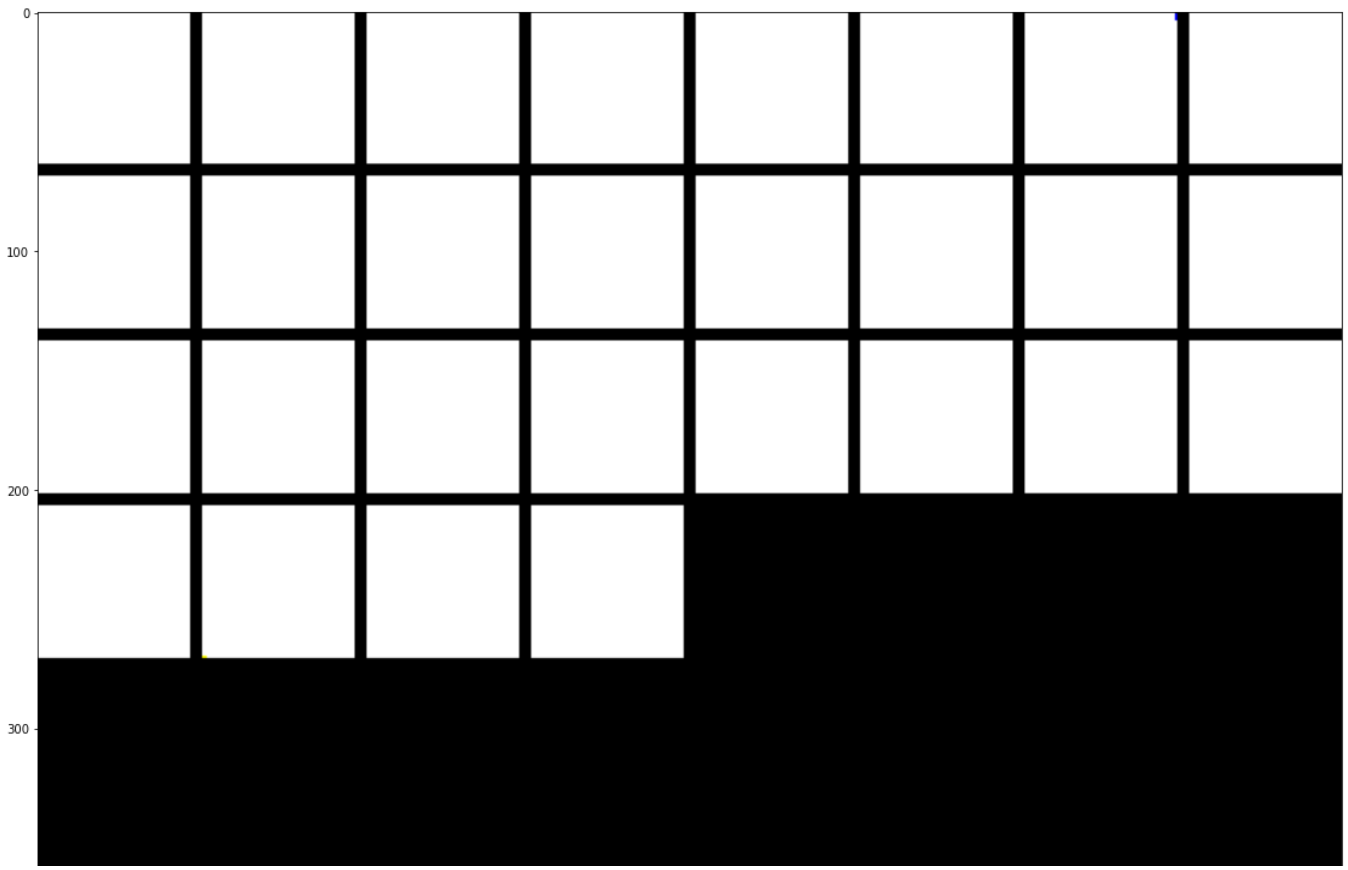

0
100
200
300
400
500

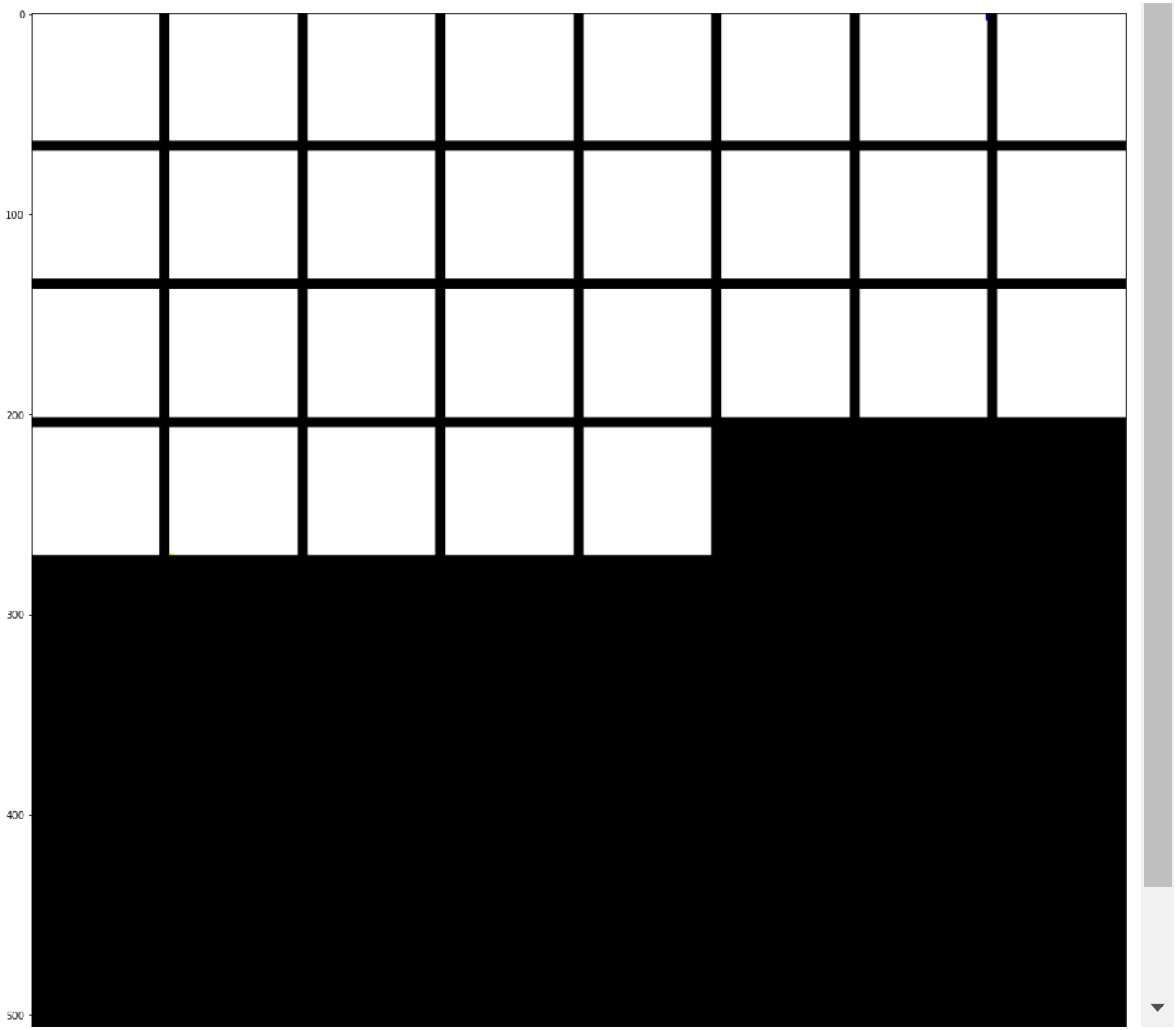

0
100
200
300
400

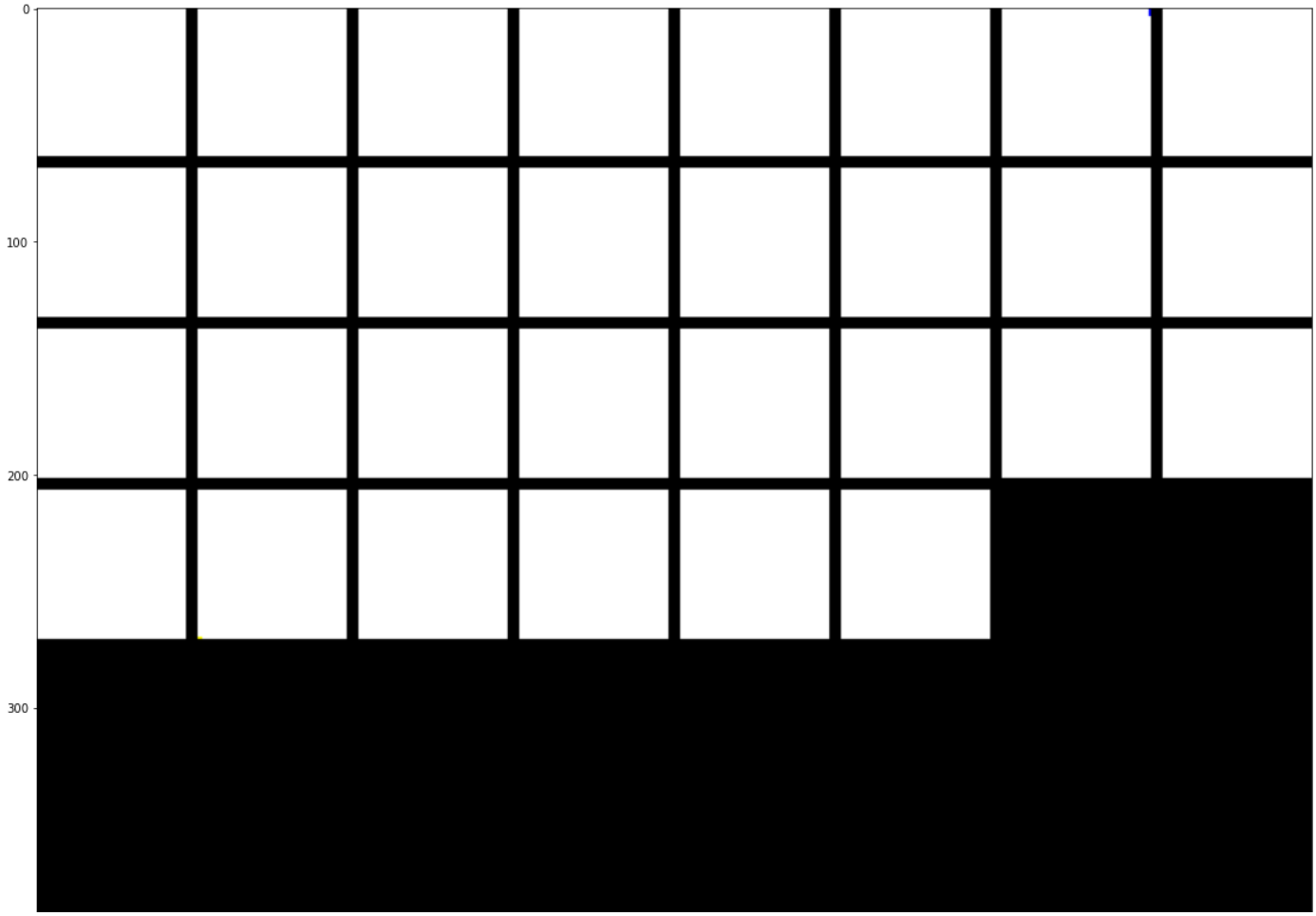

0
100
200
300
400
500
0
100
200
300
400
500

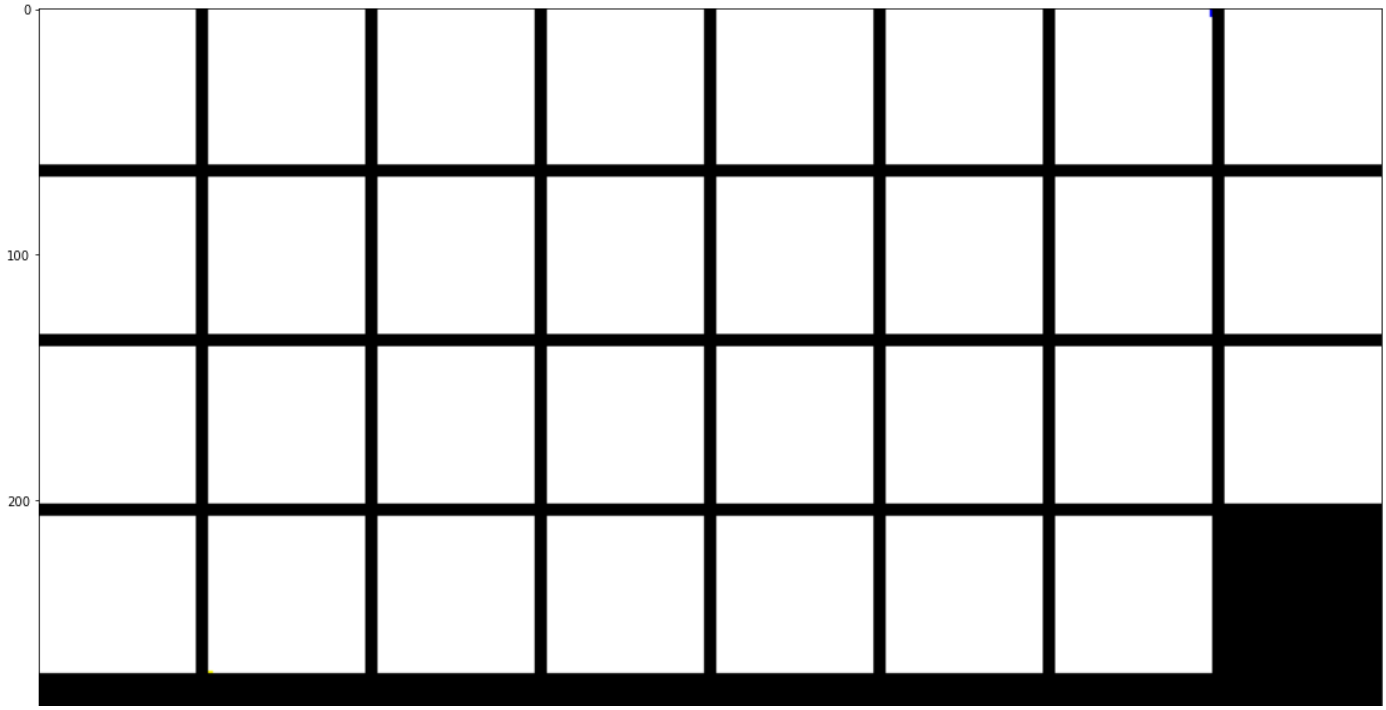

0
100
200
300
400
500
0
100
200
300
400
500

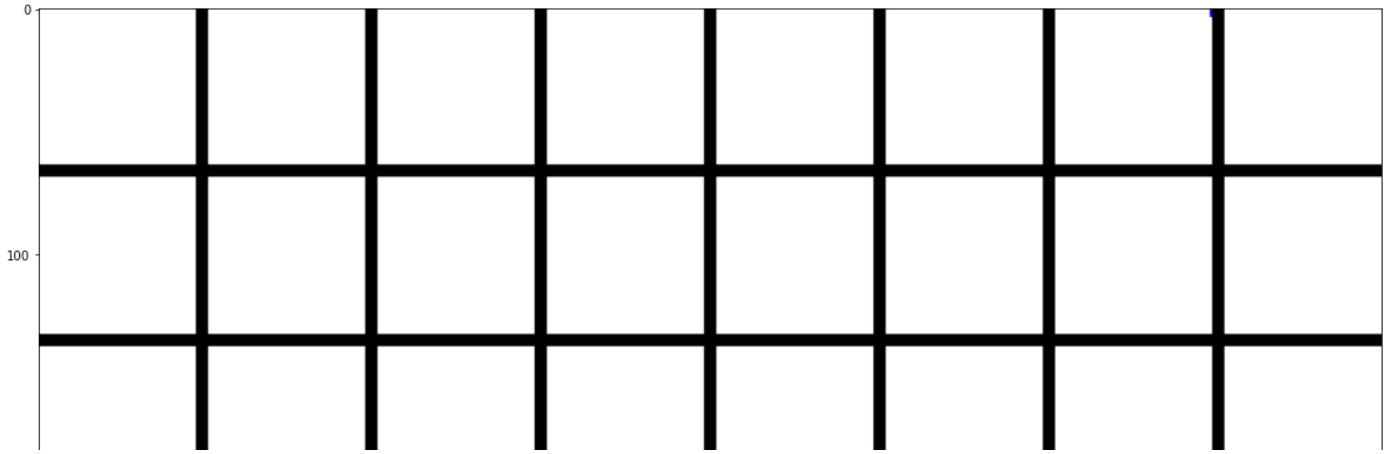

0
100
200
300
400

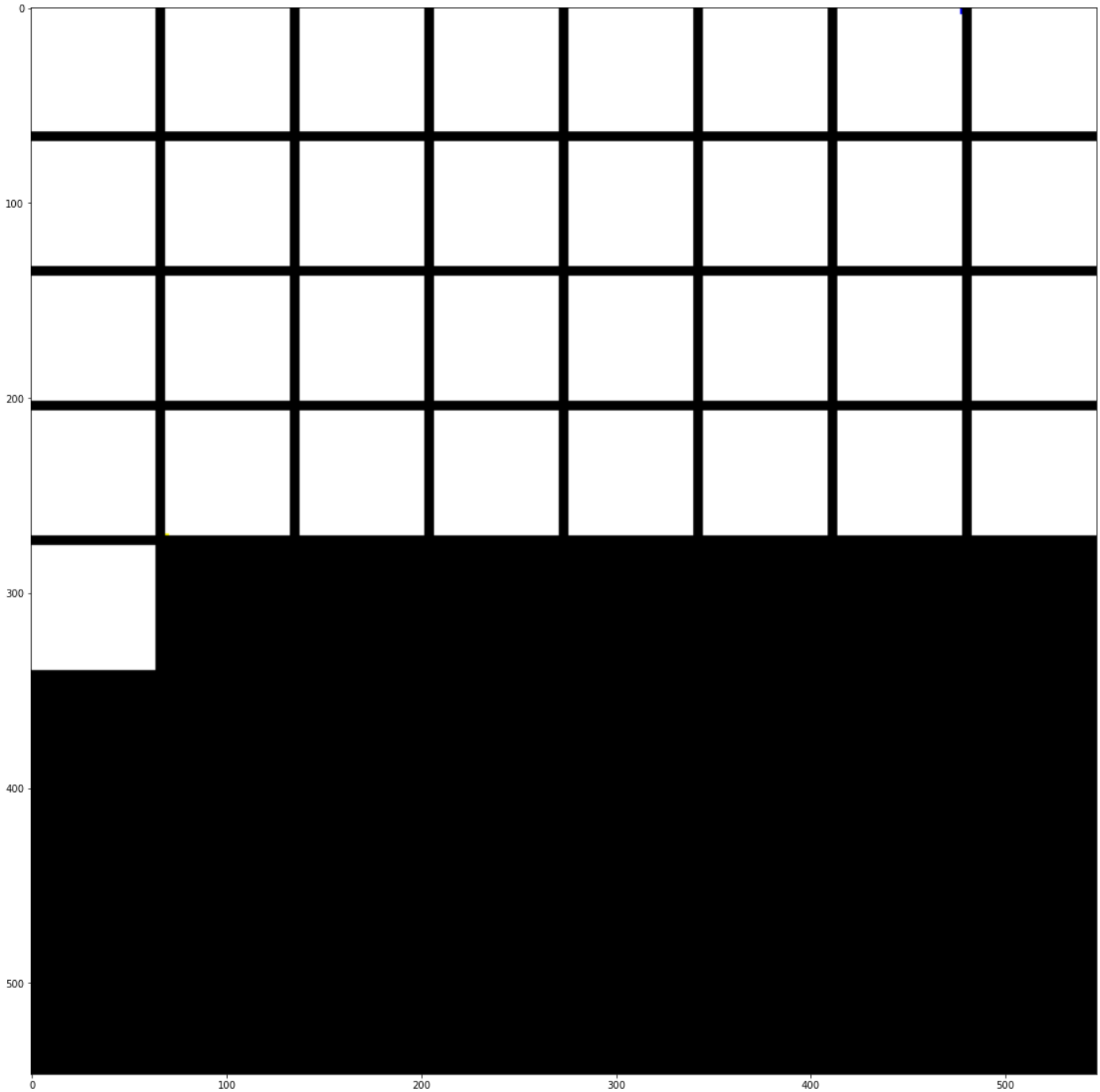

0
100
200
300

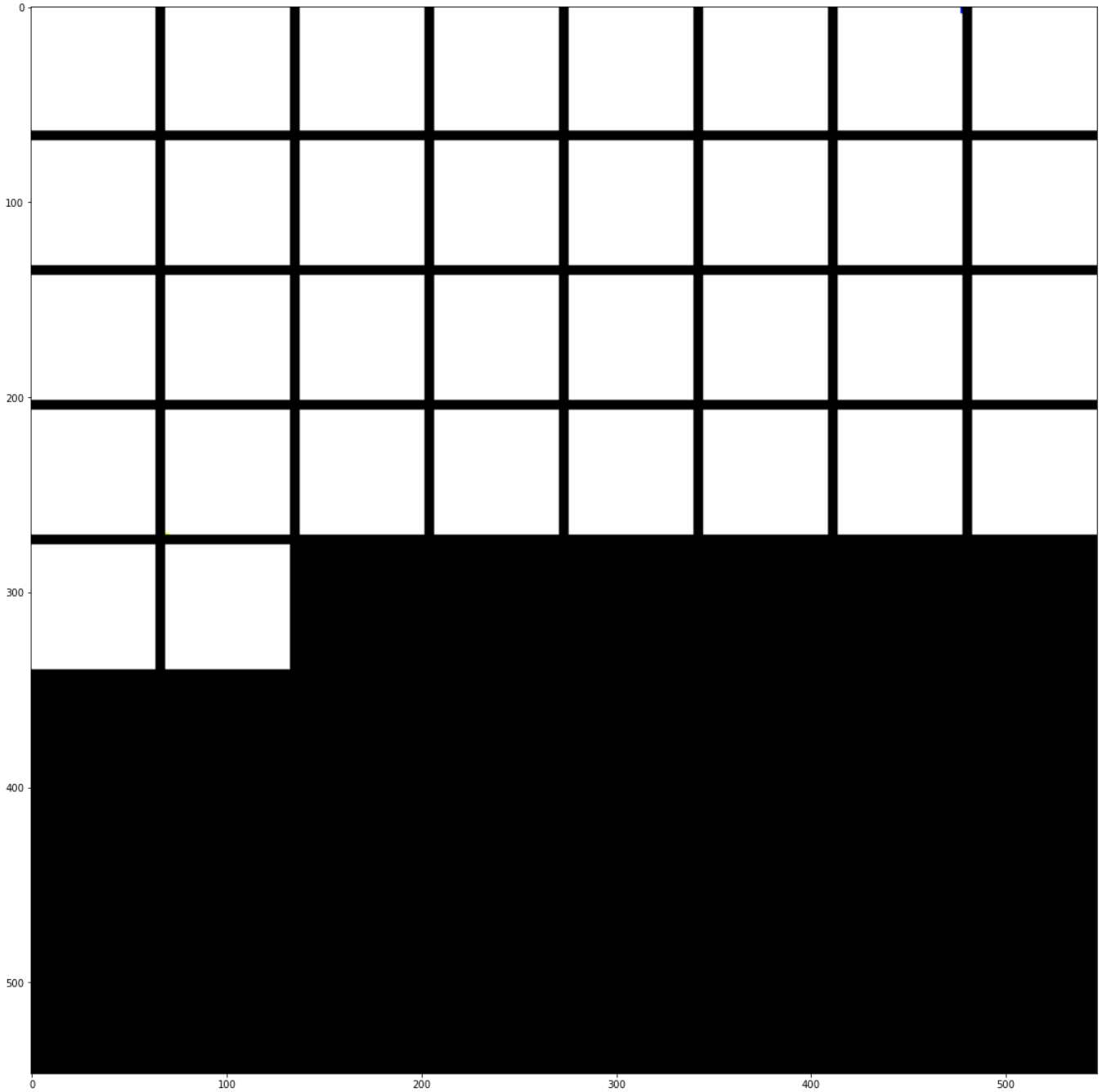

0
100
200
300
400
500

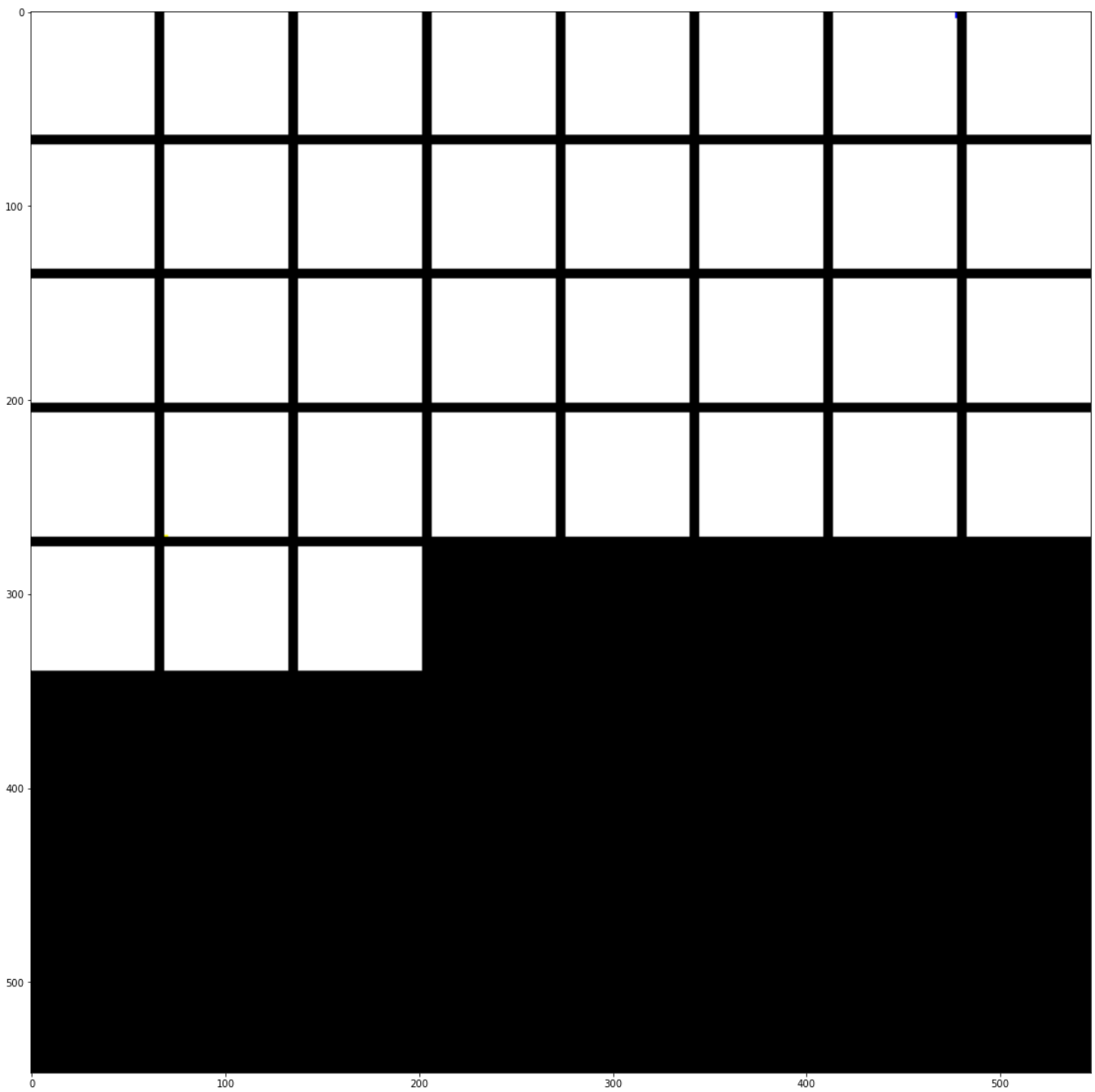

0
100
200
300

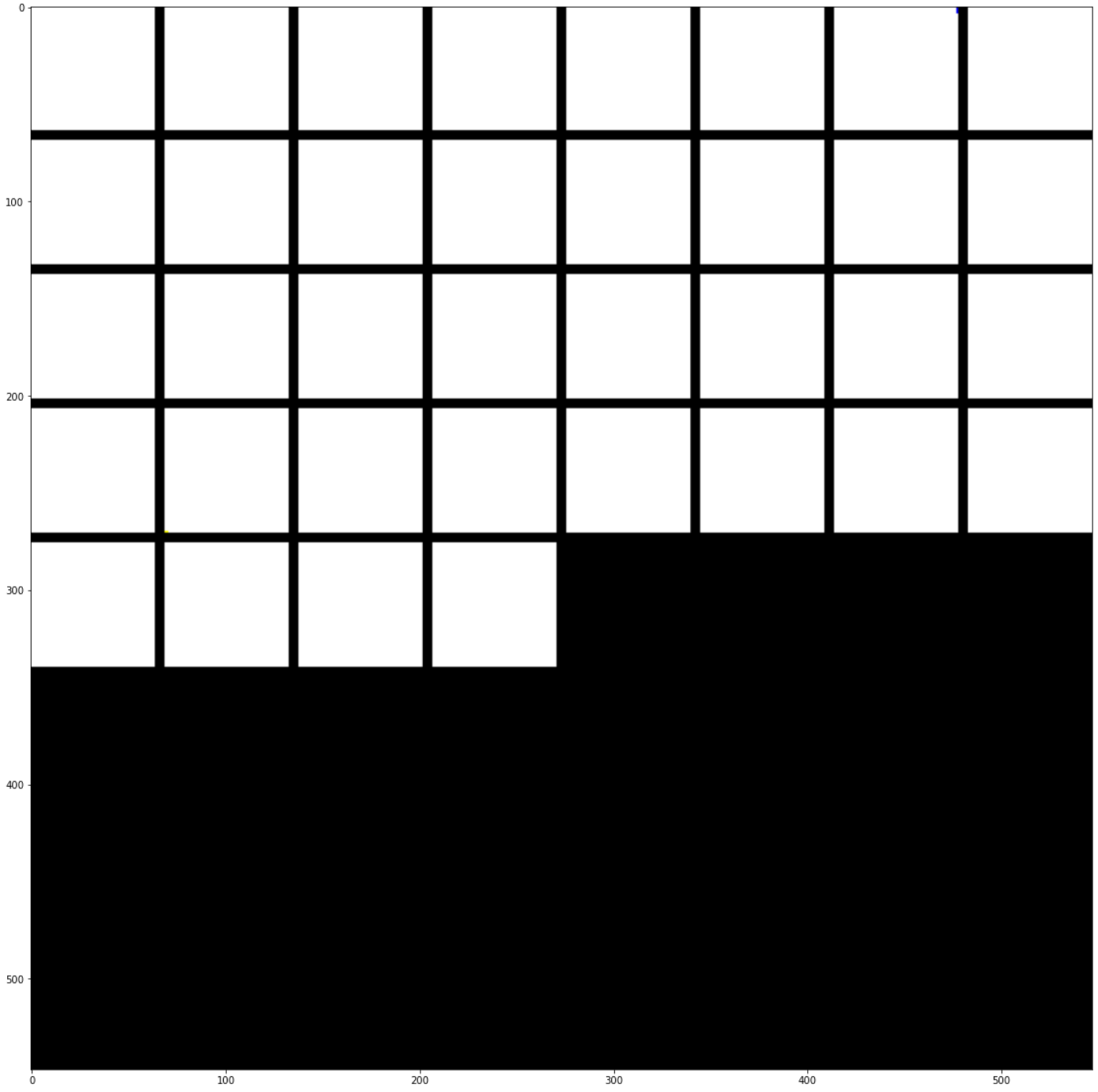

0
100
200

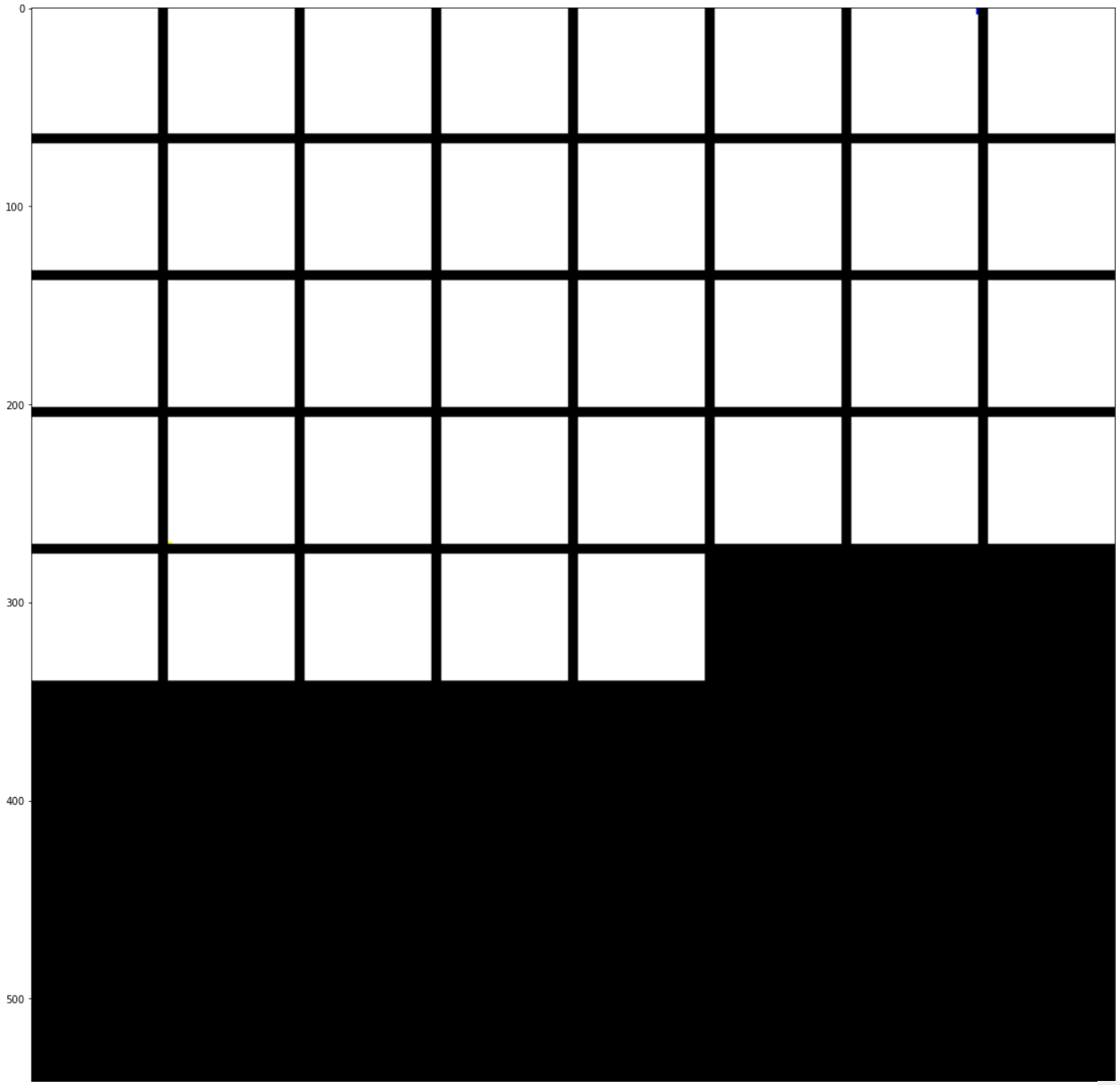

0
100

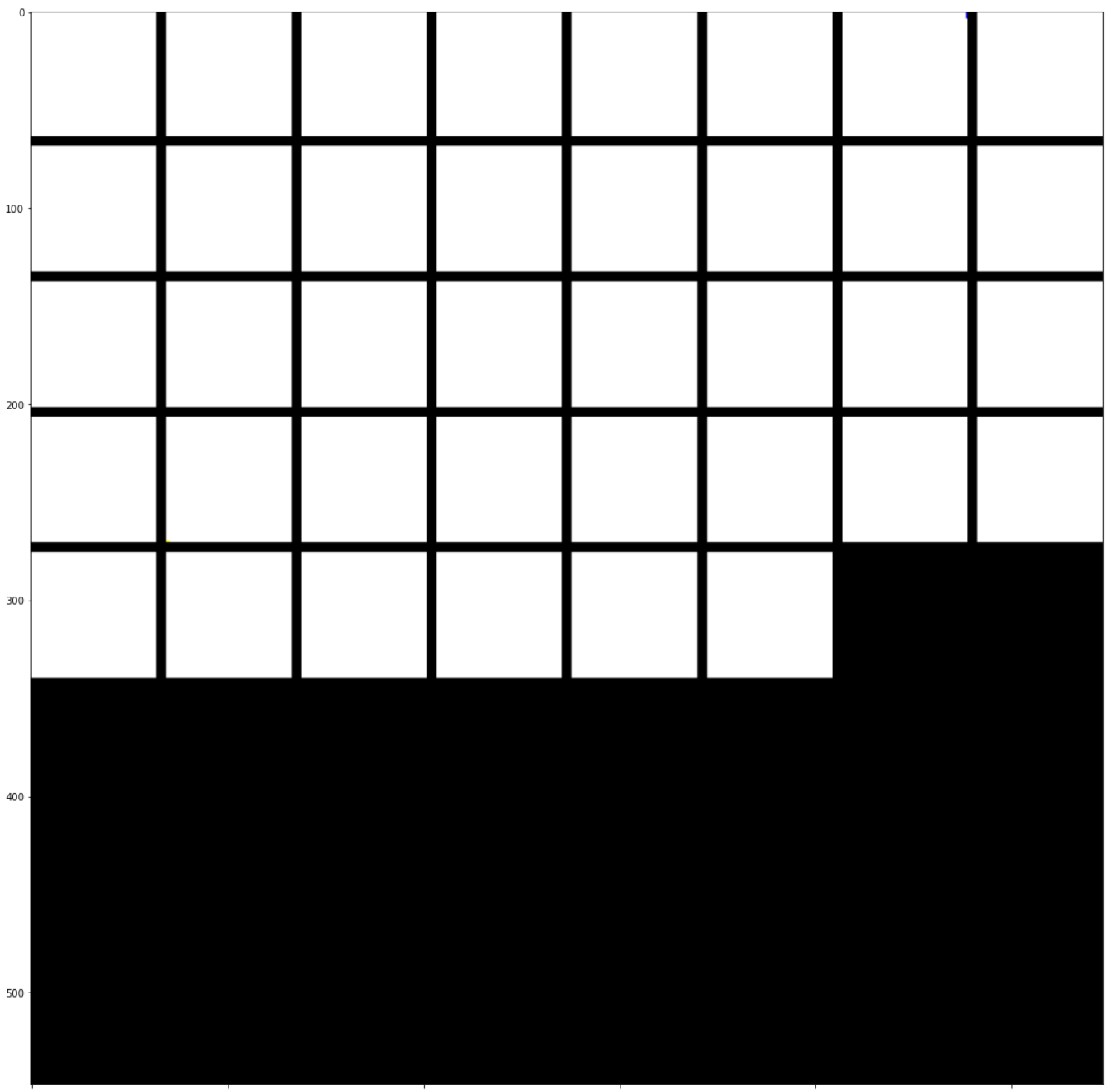

0
100
200
300
400
500
0
100
200
300
400
500

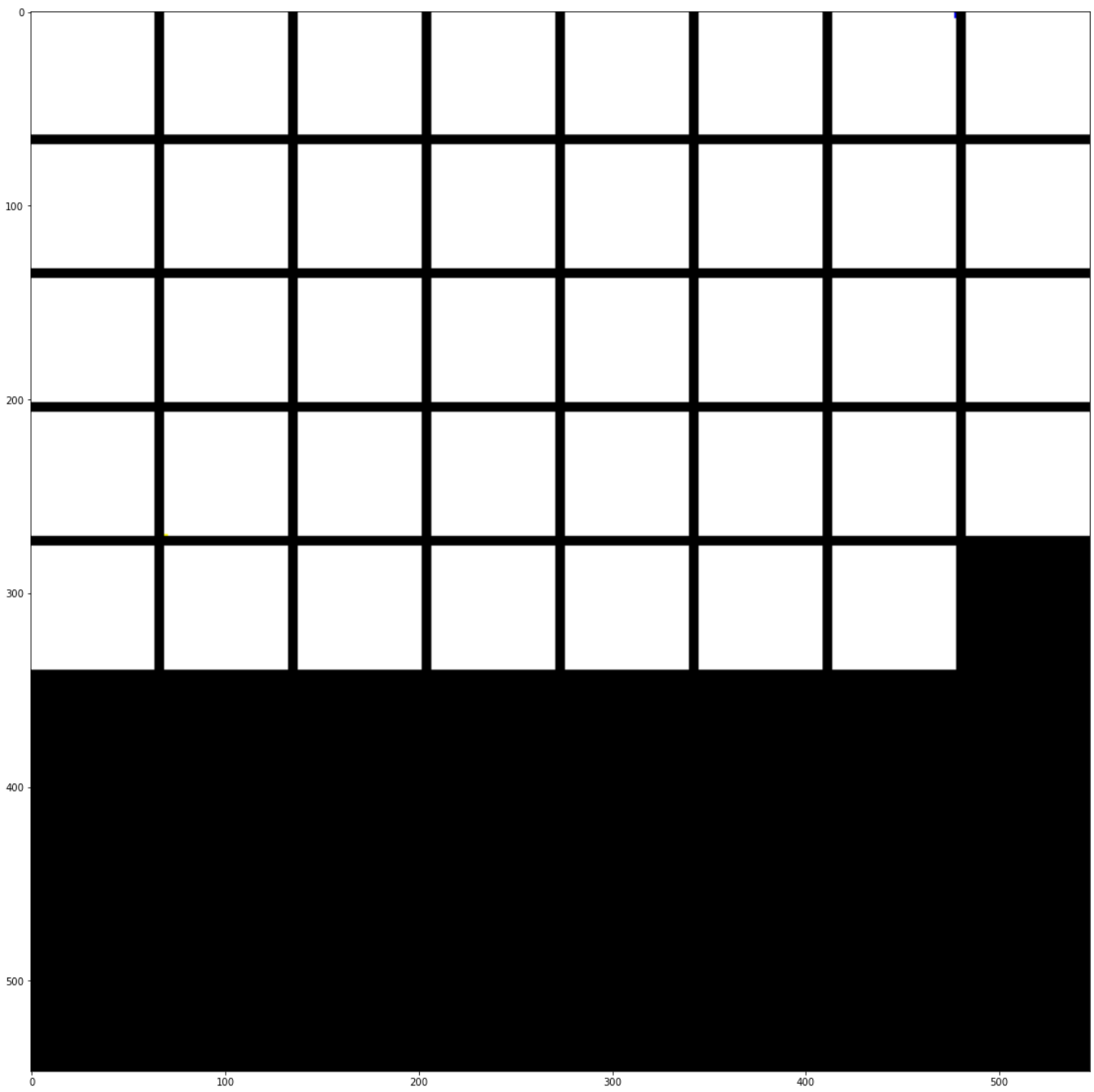

0
100
200
300
400
500
0
100
200
300
400
500

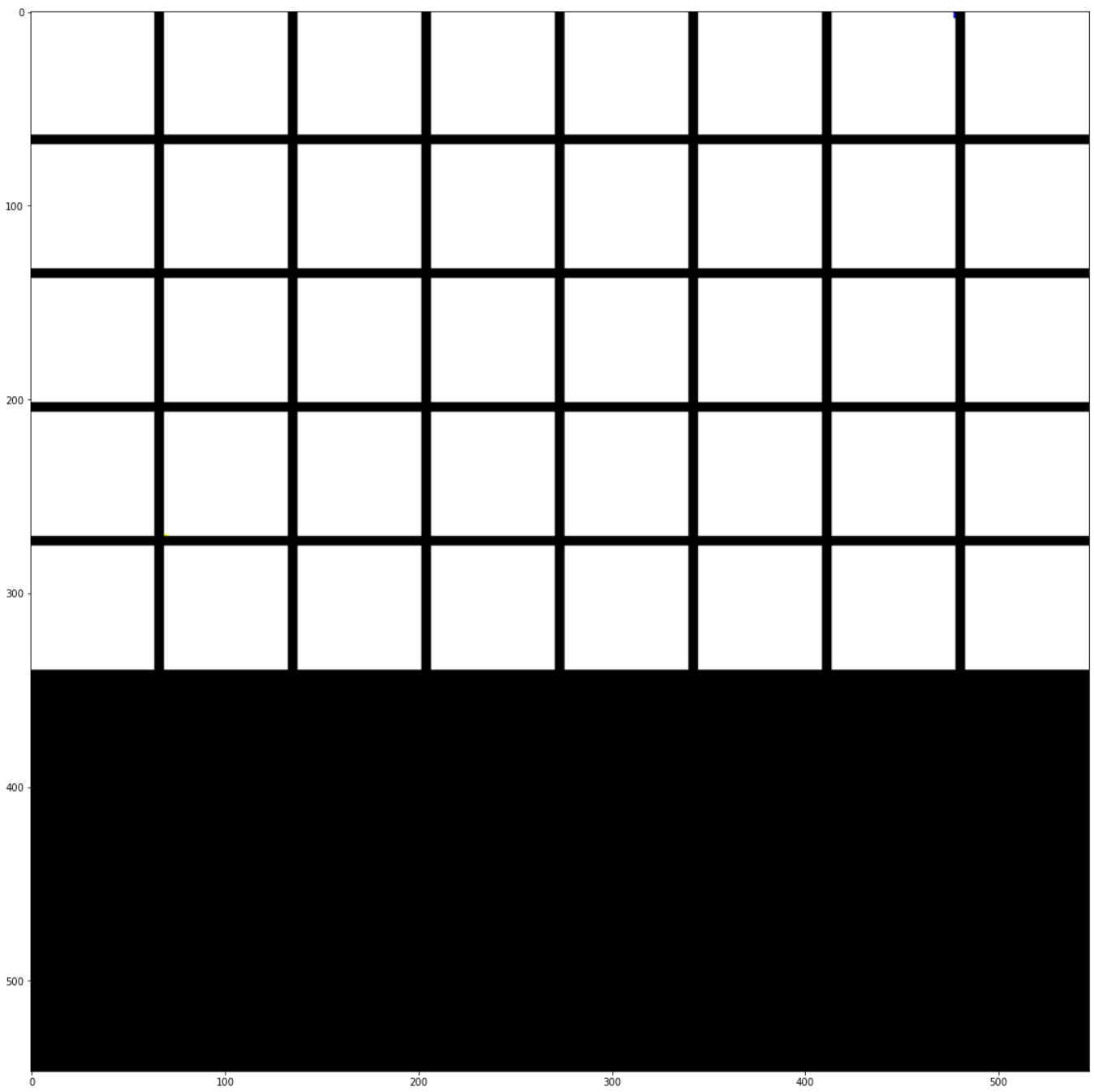

0
100
200
300
400
500
0
100
200
300
400
500

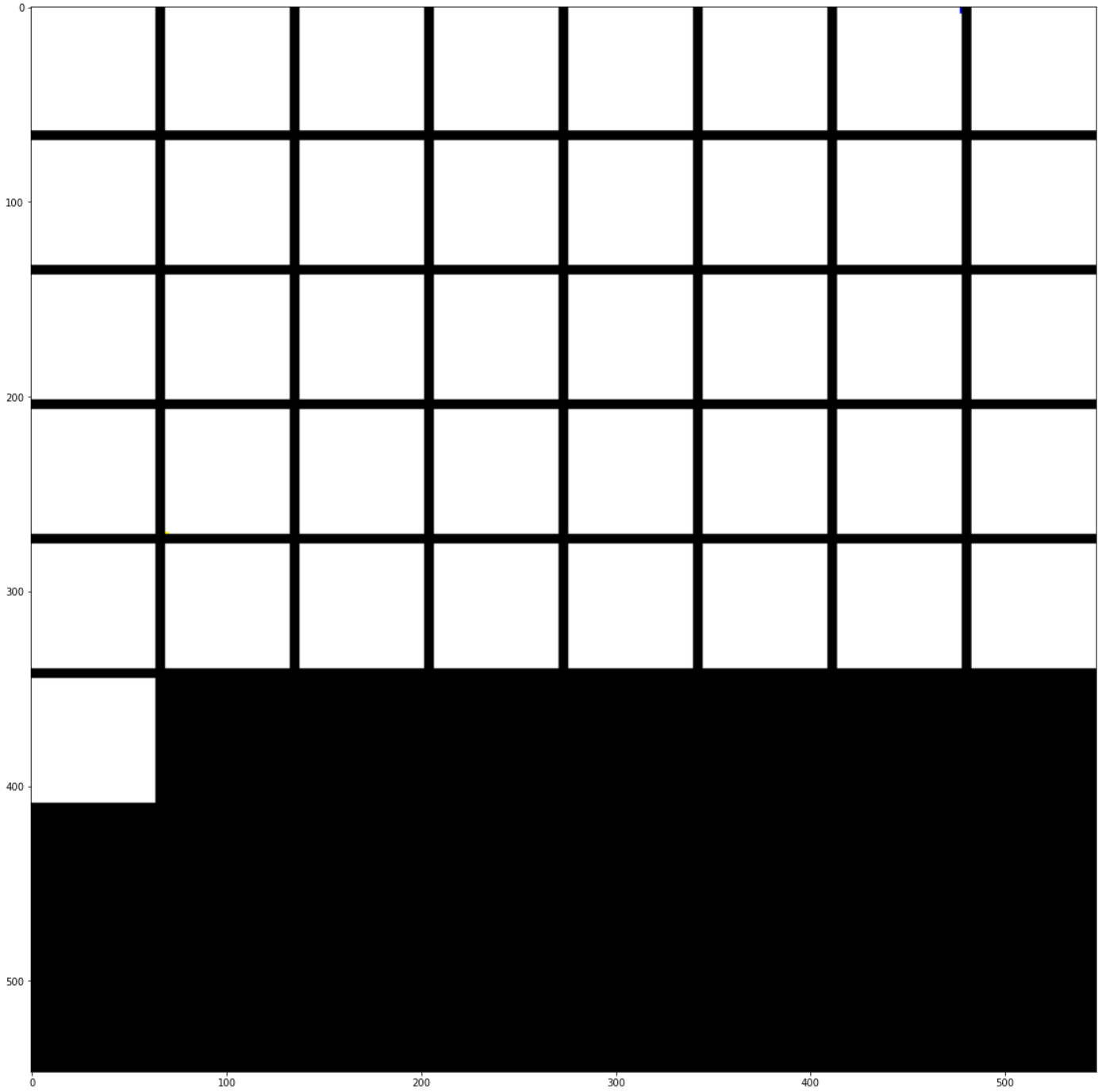

0
100
200
300
400
500
0
100
200
300
400
500

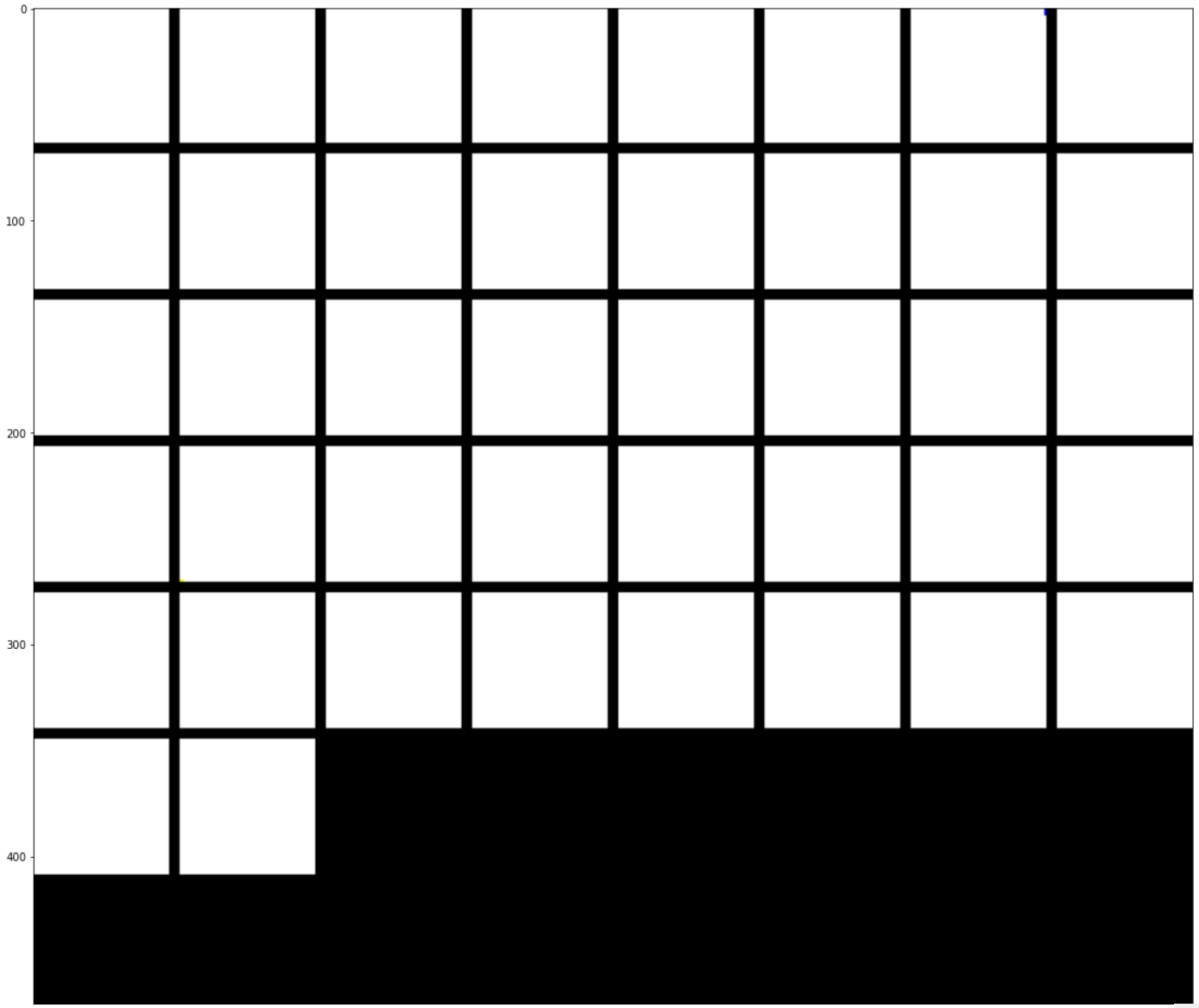

0
100
200
300
400
500

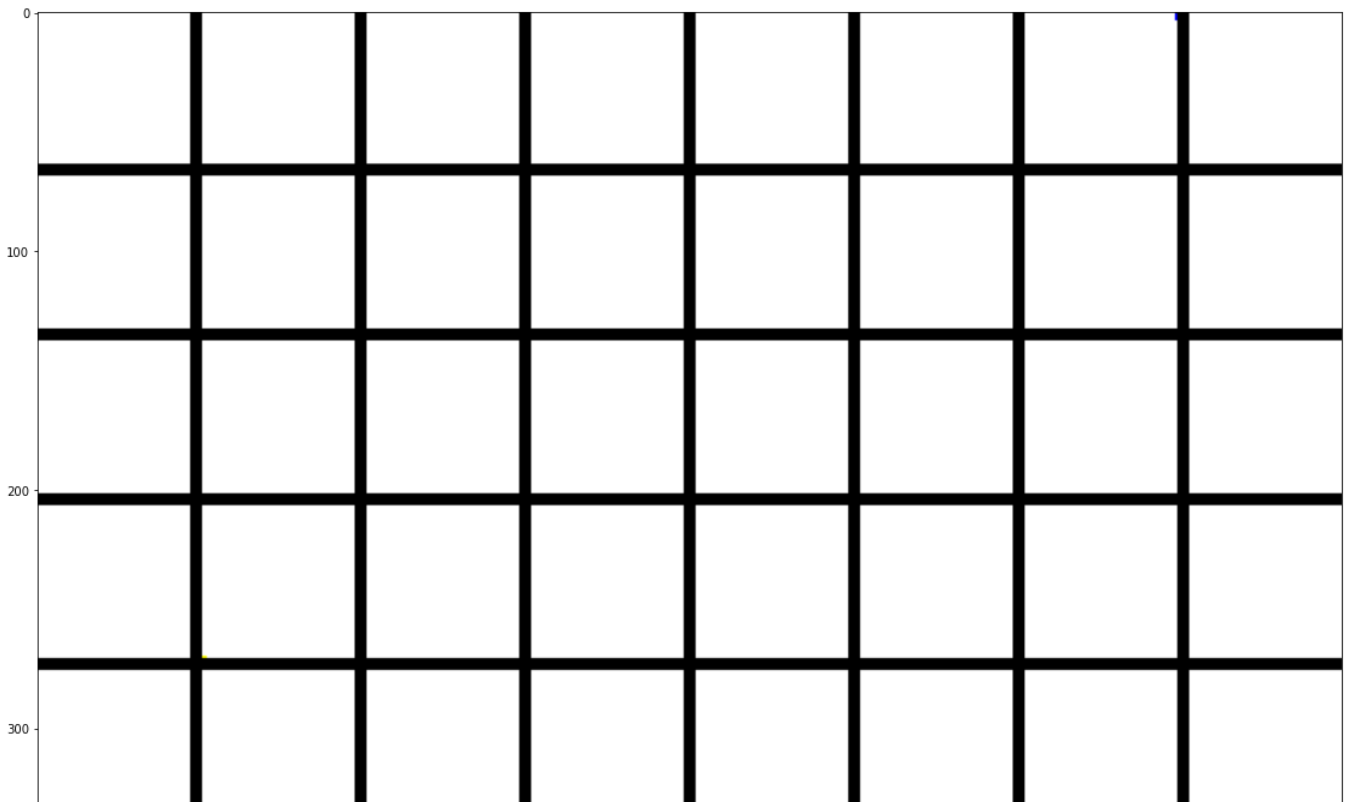

0
100
200
300
400
500

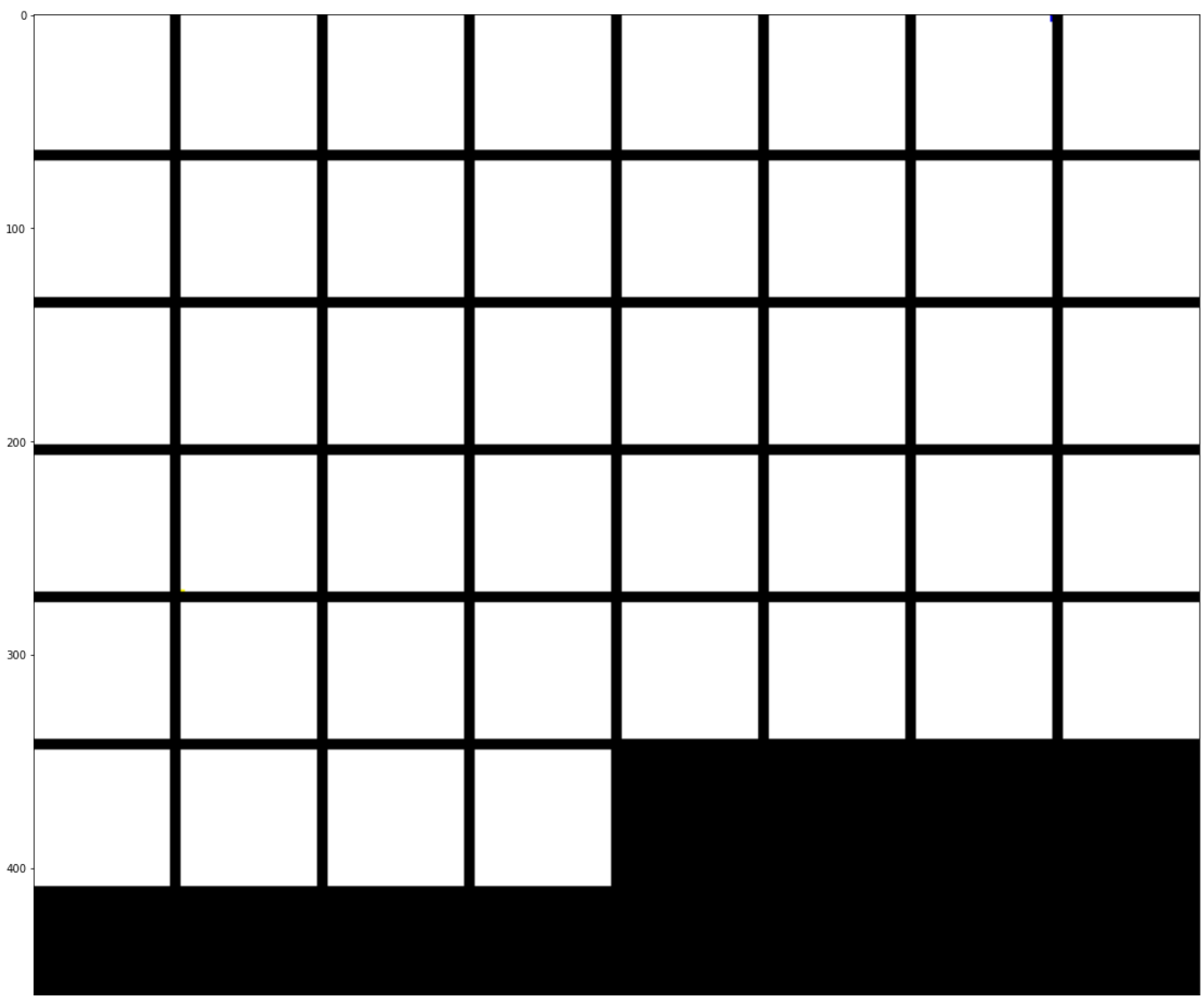

0
100
200
300
400
500
0
100
200
300
400
500

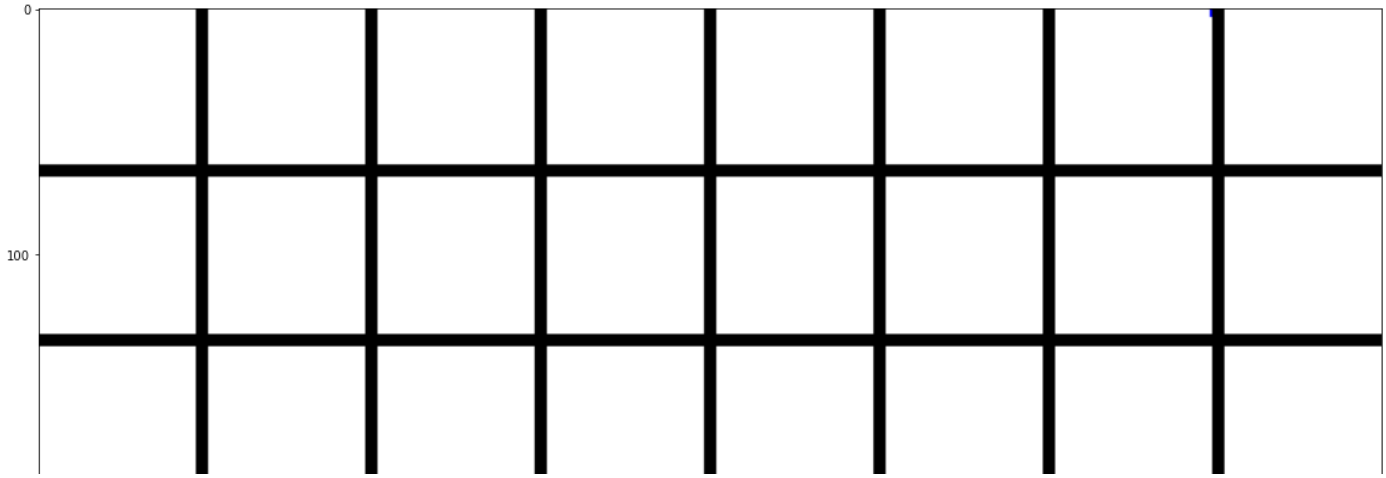

0
100
200
300
400
500
0
100
200
300
400
500

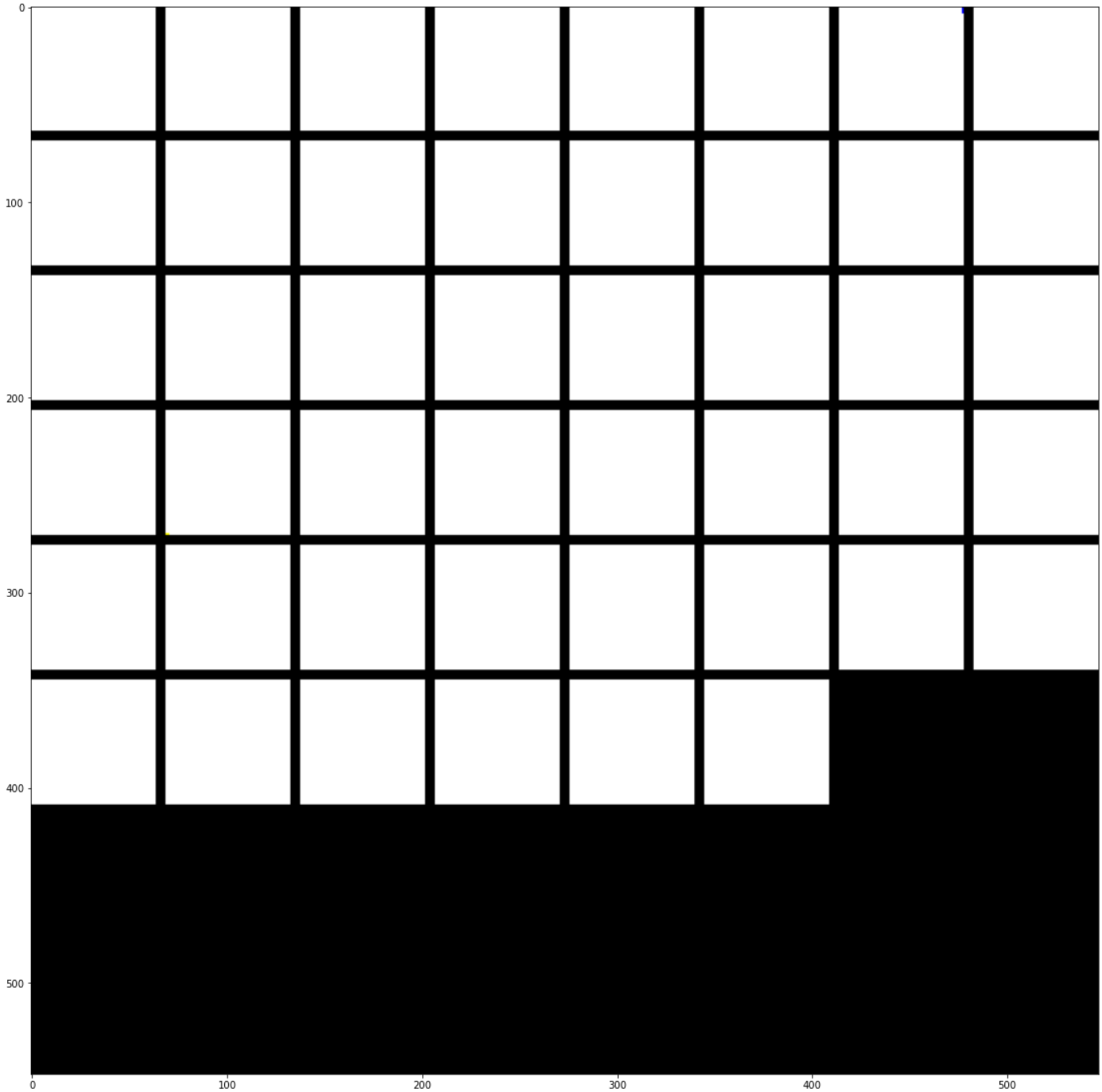

0
100
200
300
400
500
0
100
200
300
400
500

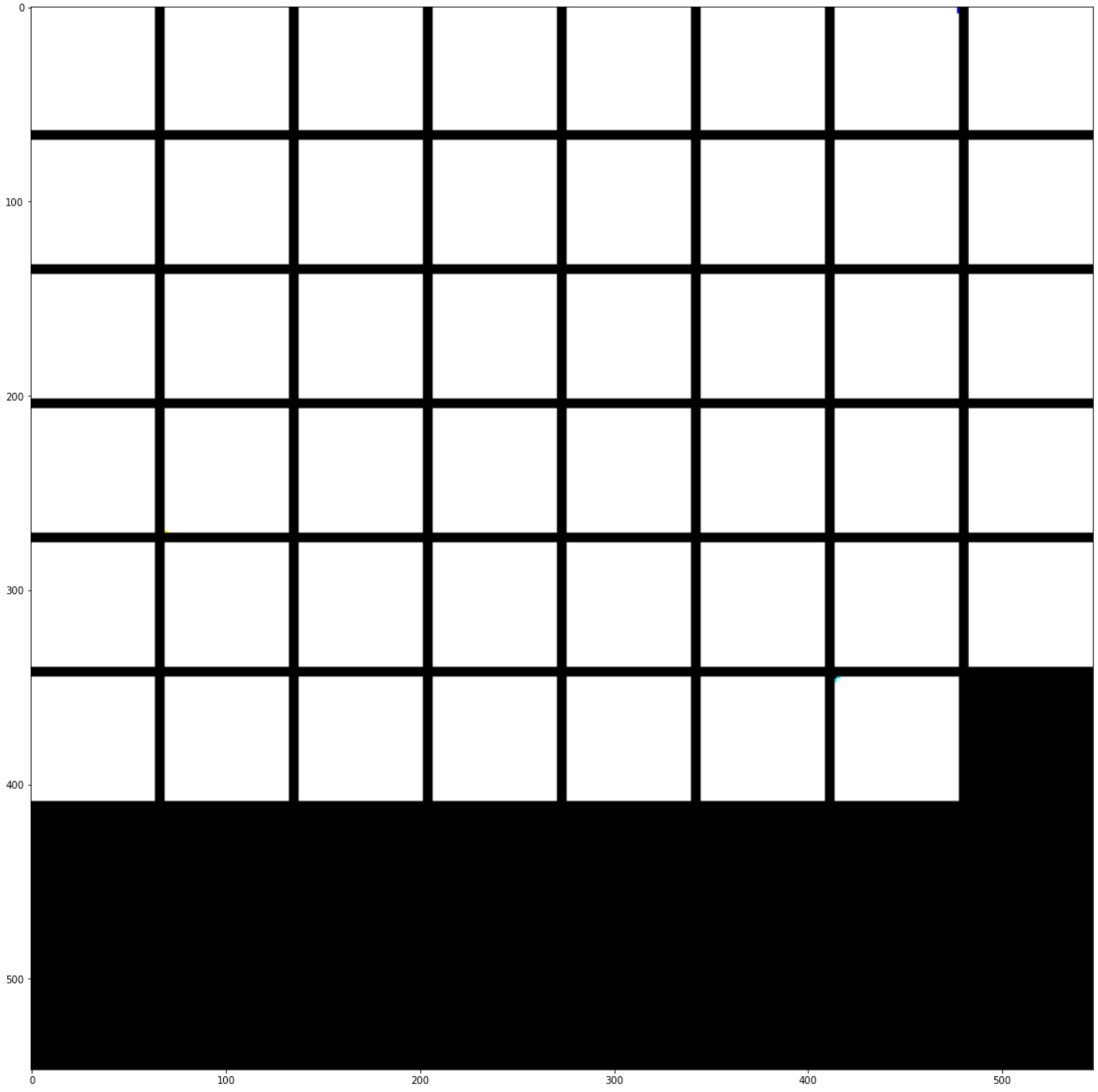

0
100
200
300
400

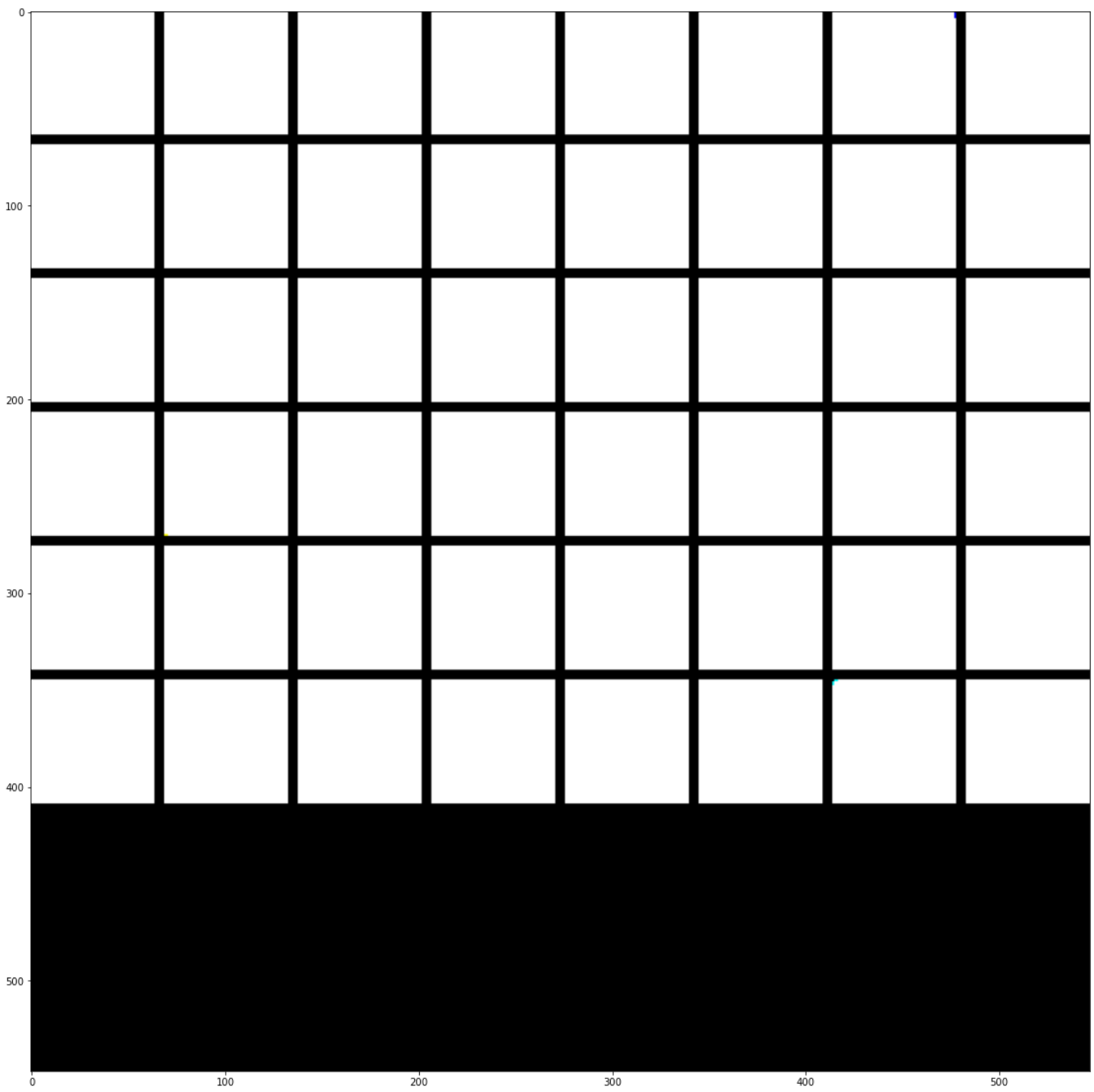

0
100
200
300

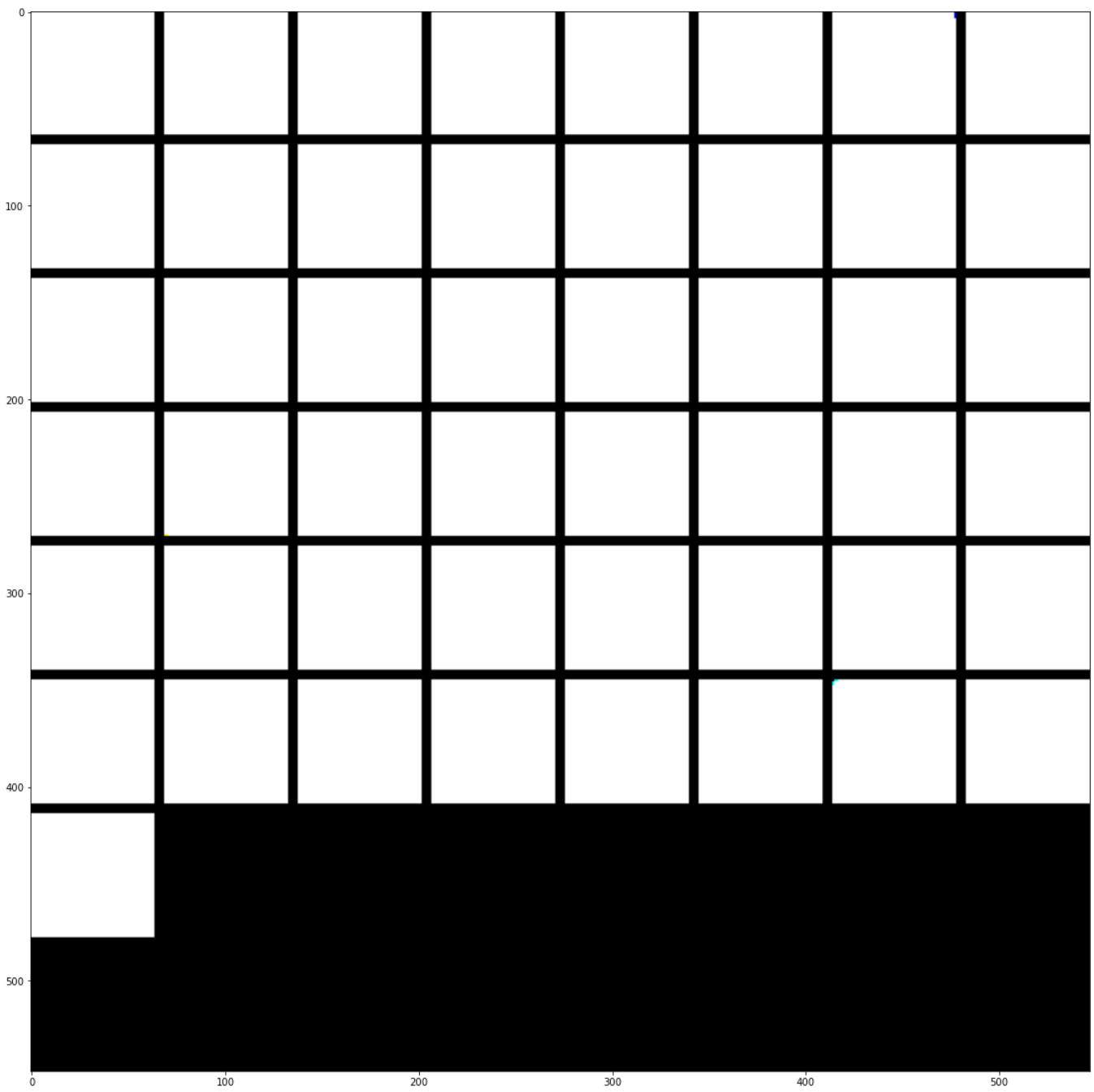

0
100
200
300
400

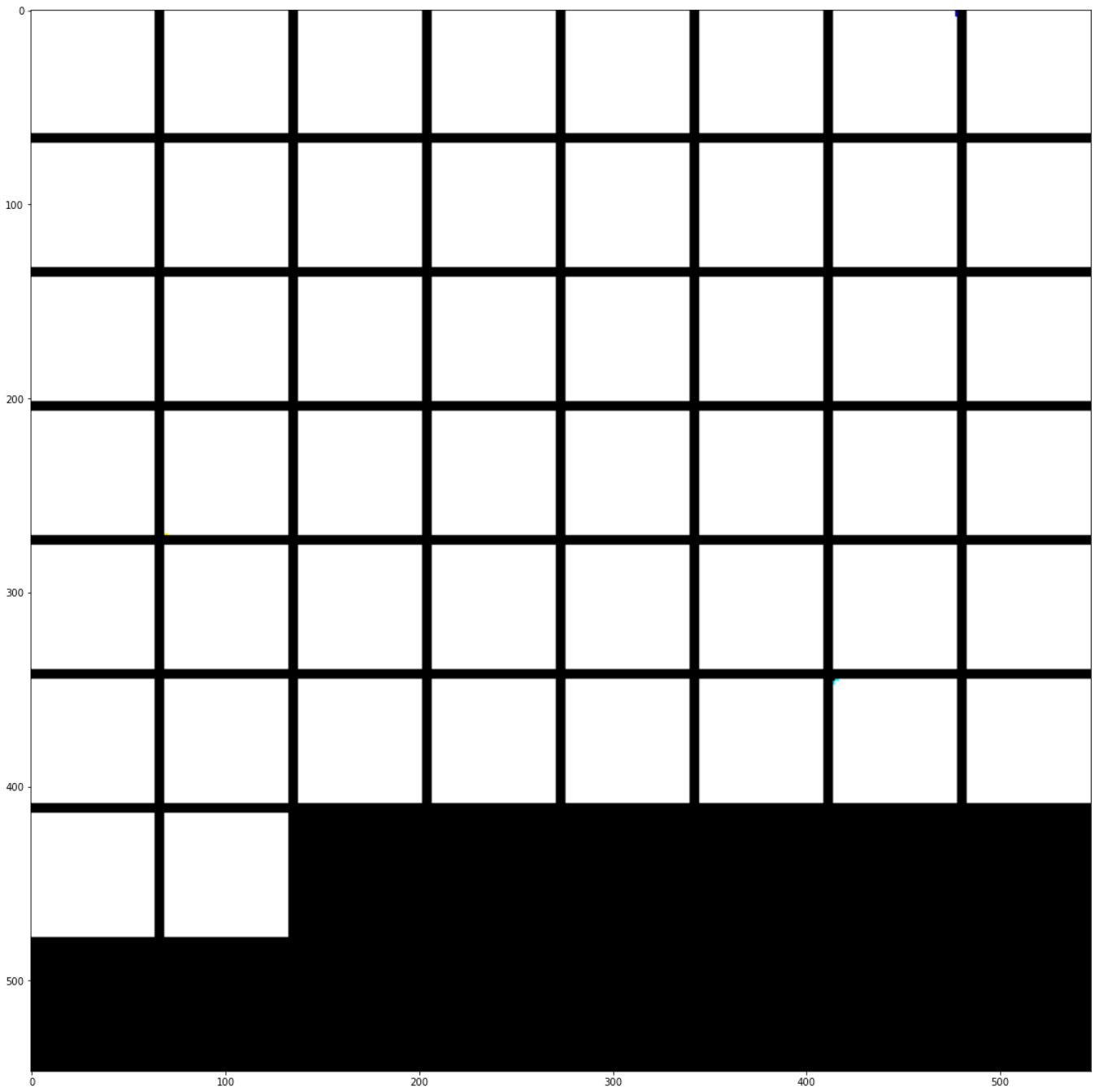

0
100

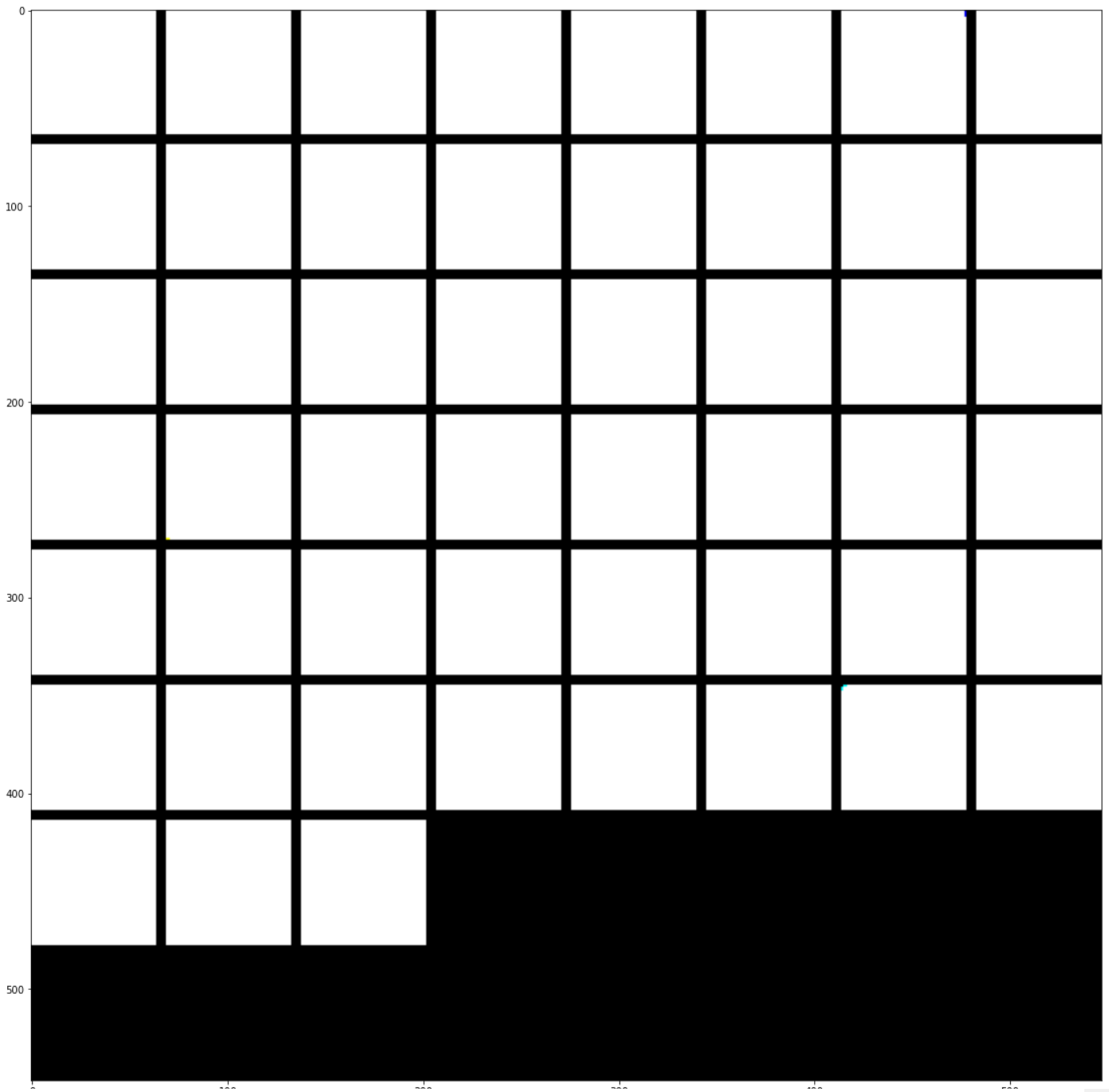

0
100
200
300
400
500
0
100
200
300
400
500

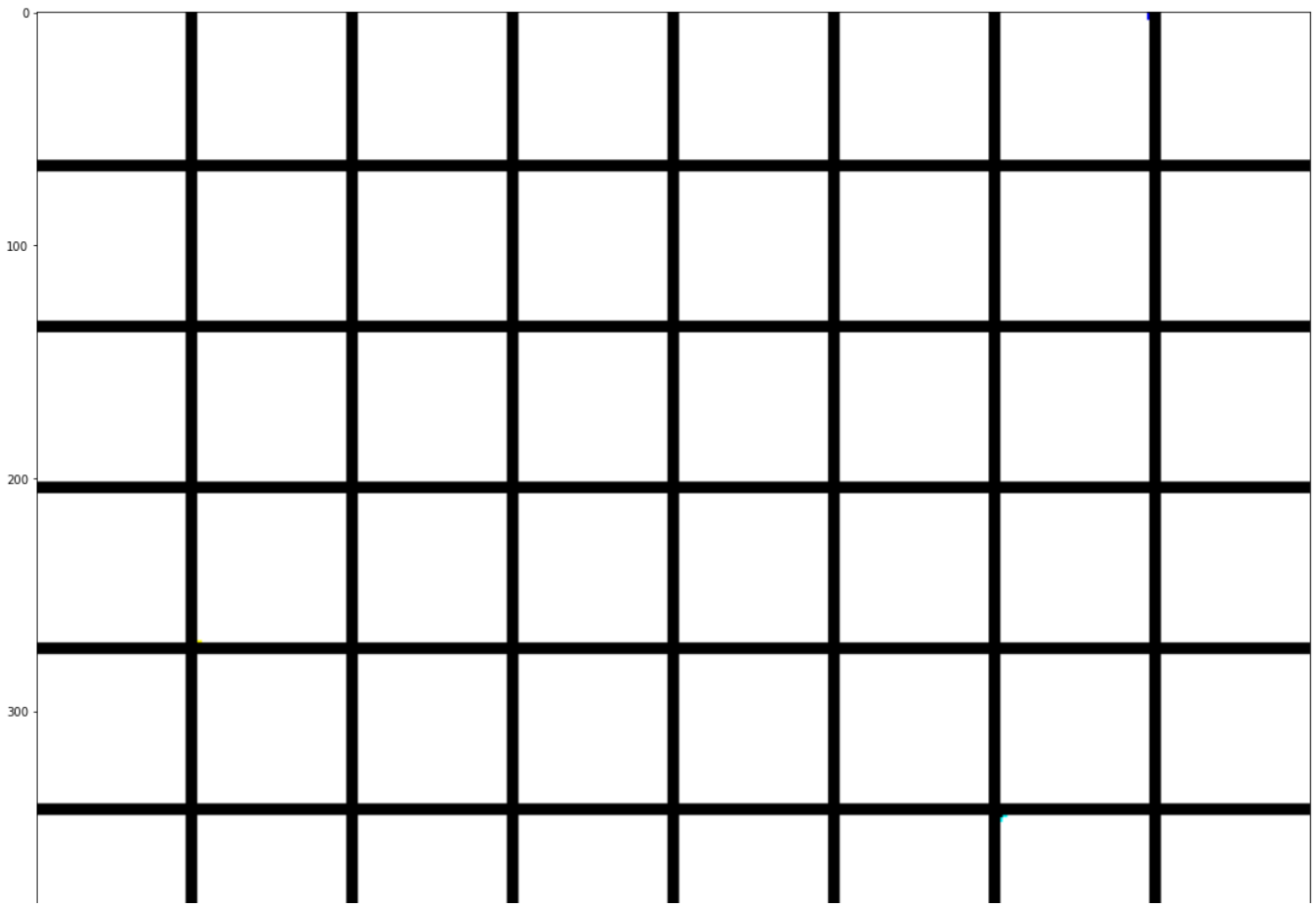

0
100
200
300
400
500
0
100
200
300
400
500

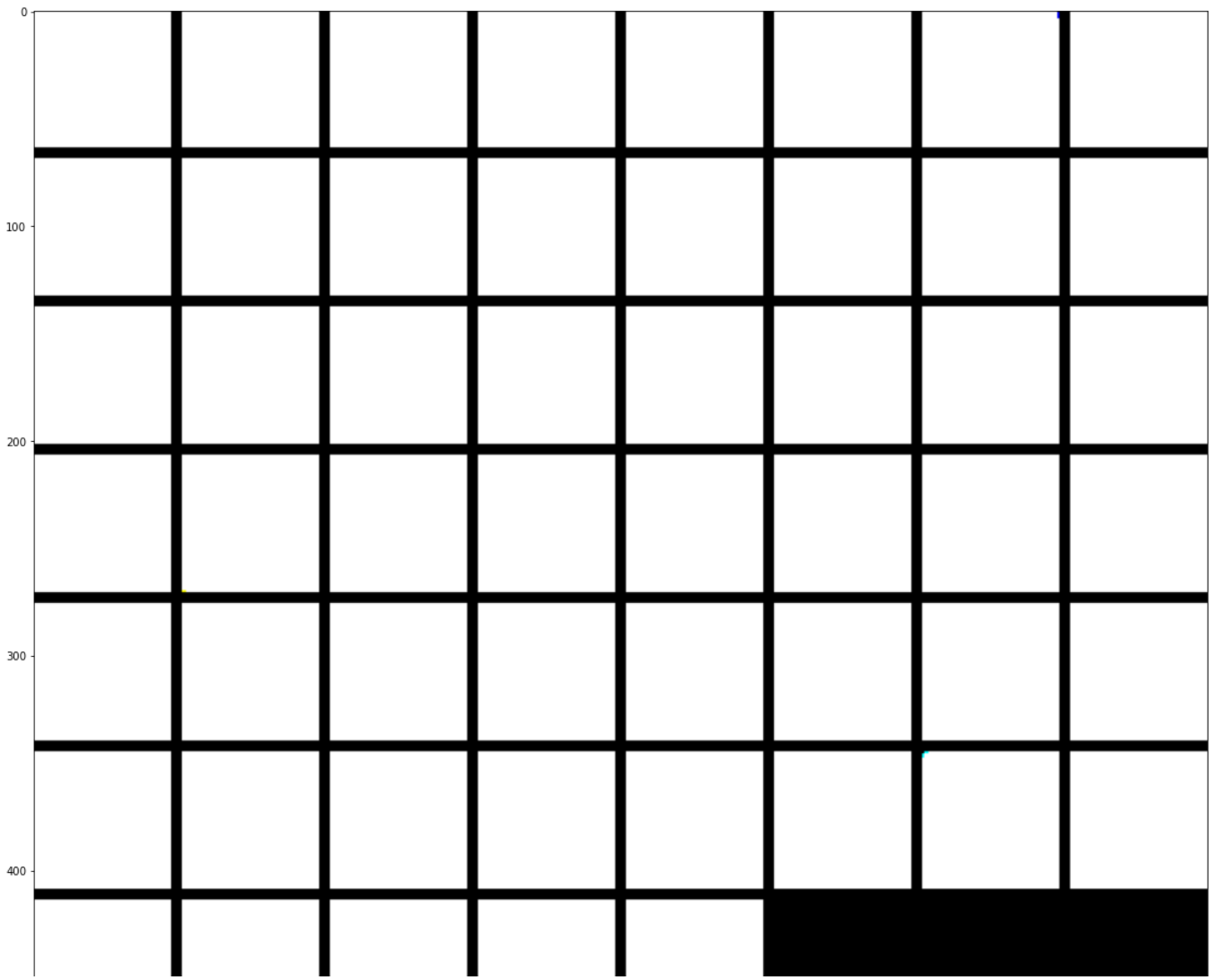

0
100
200
300
400
500
0
100
200
300
400
500

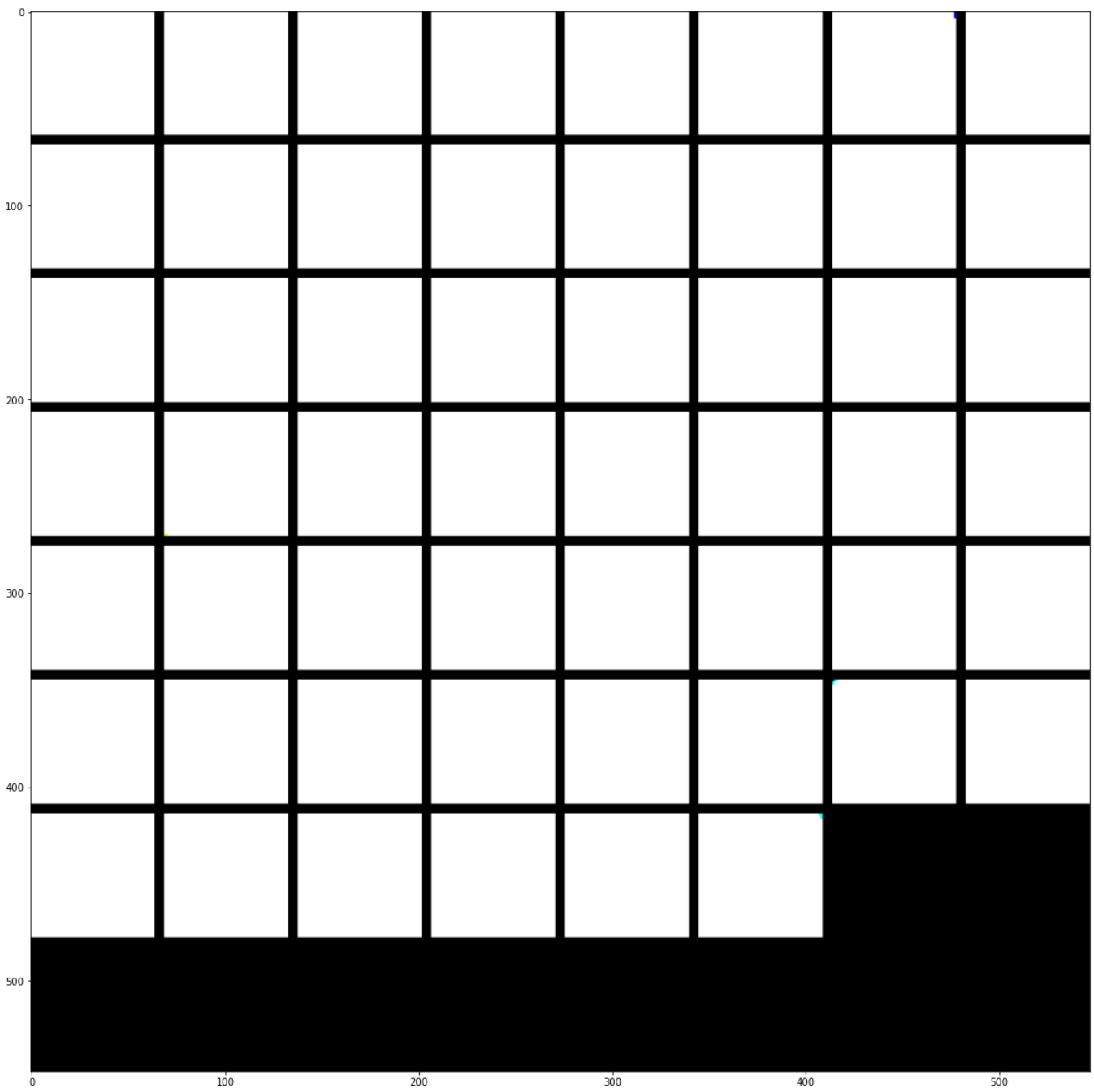

0
100
200
300
400
500
0
100
200
300
400
500

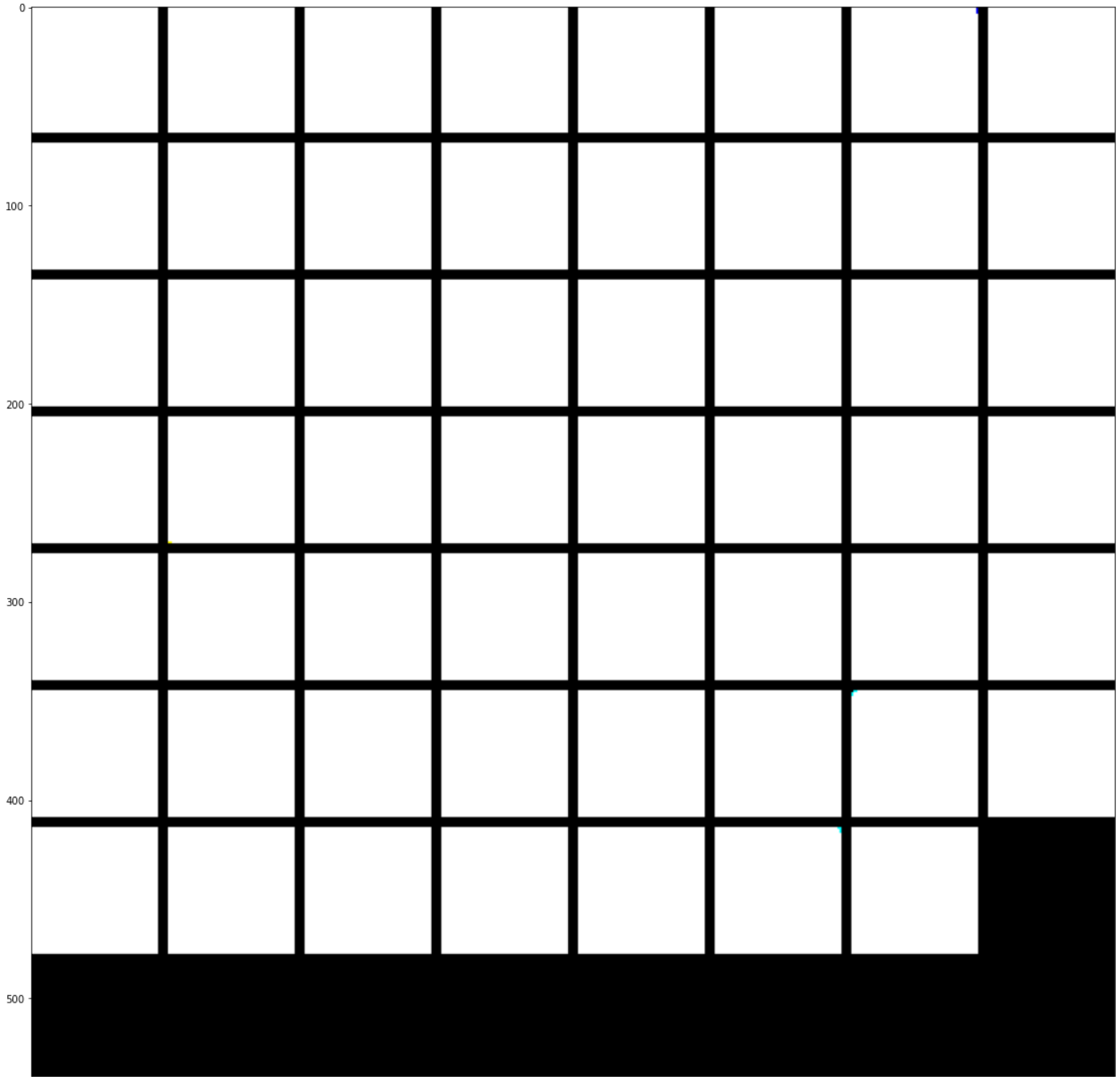

0
100
200
300
400
500
0
100
200
300
400
500

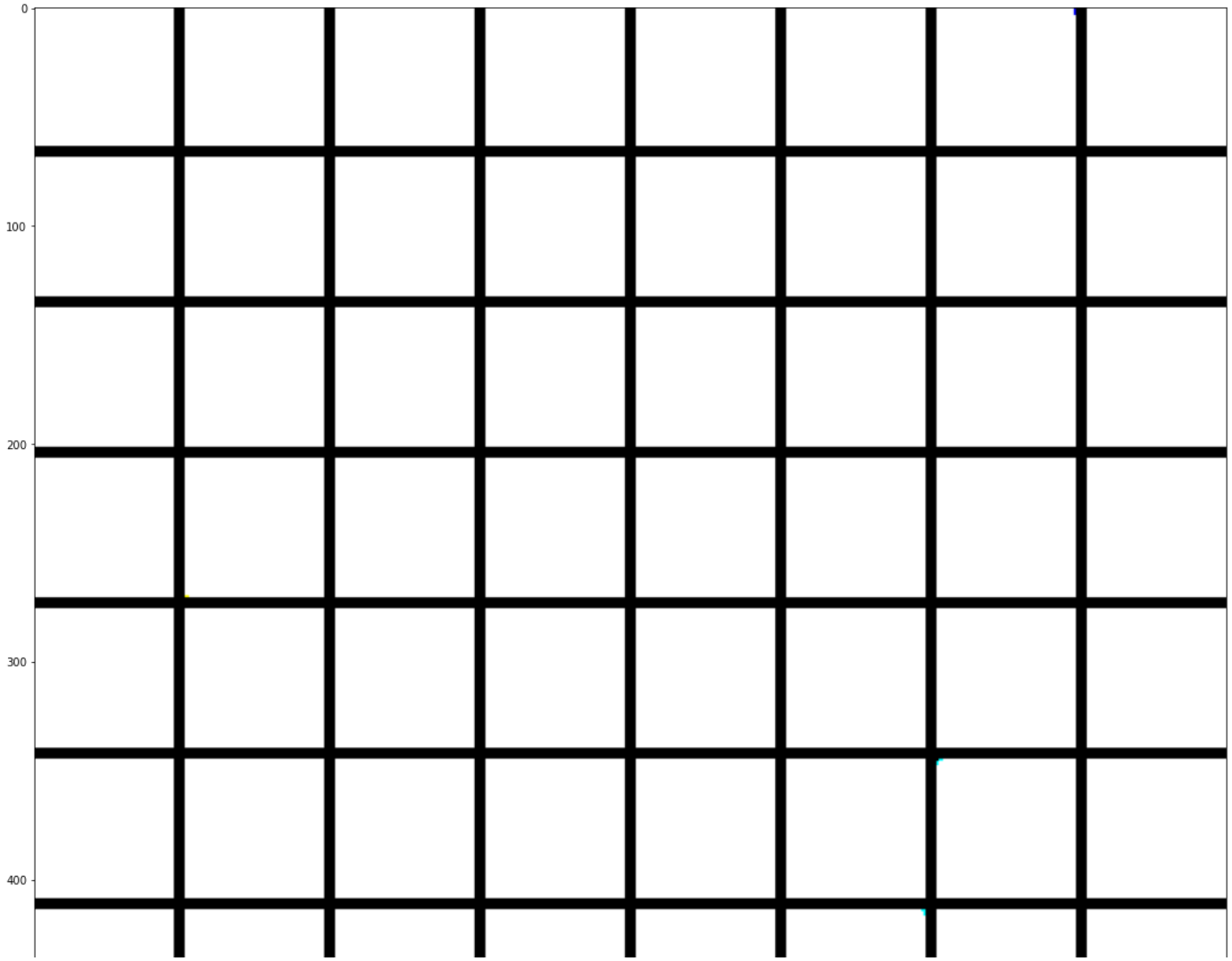

0
100
200
300
400
500

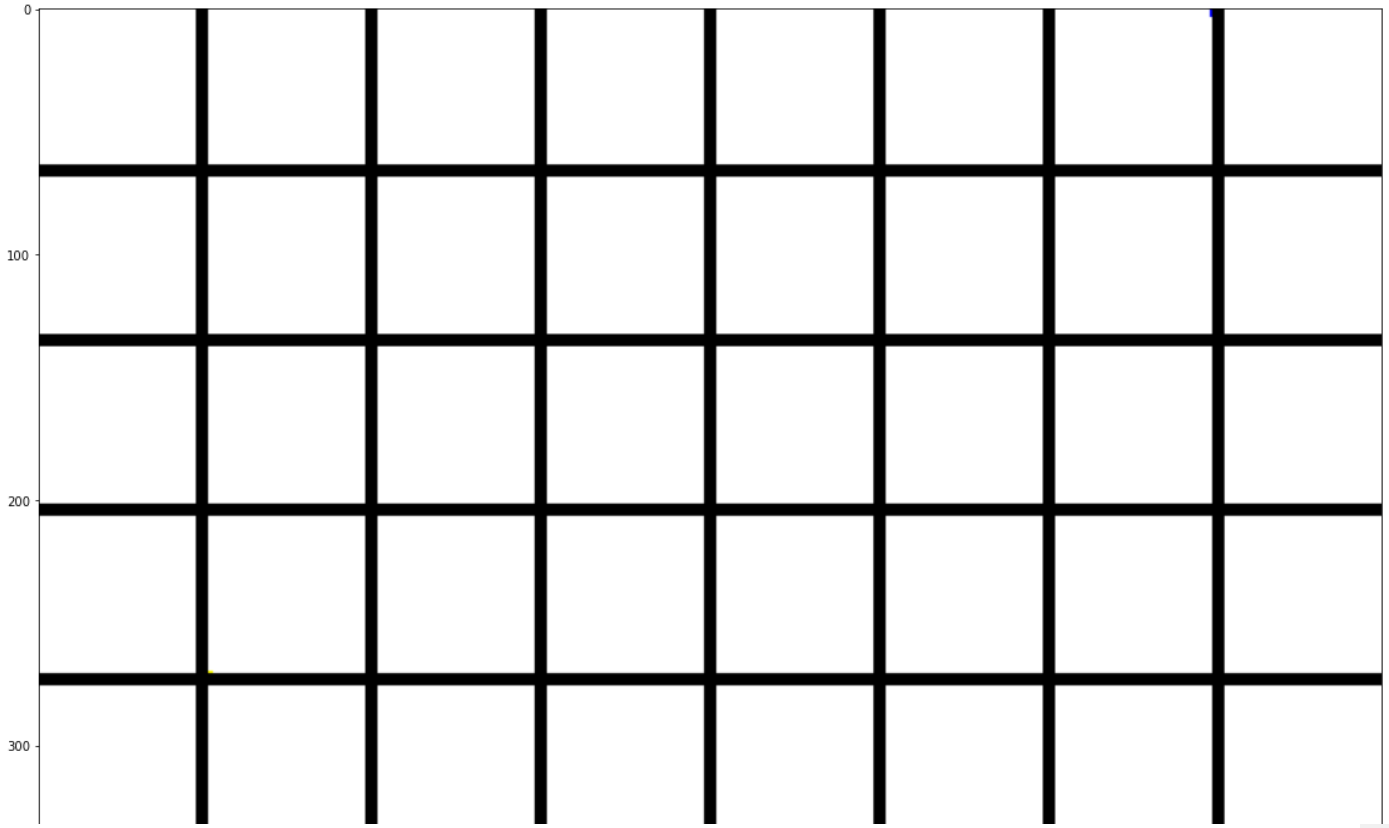

0
100
200
300

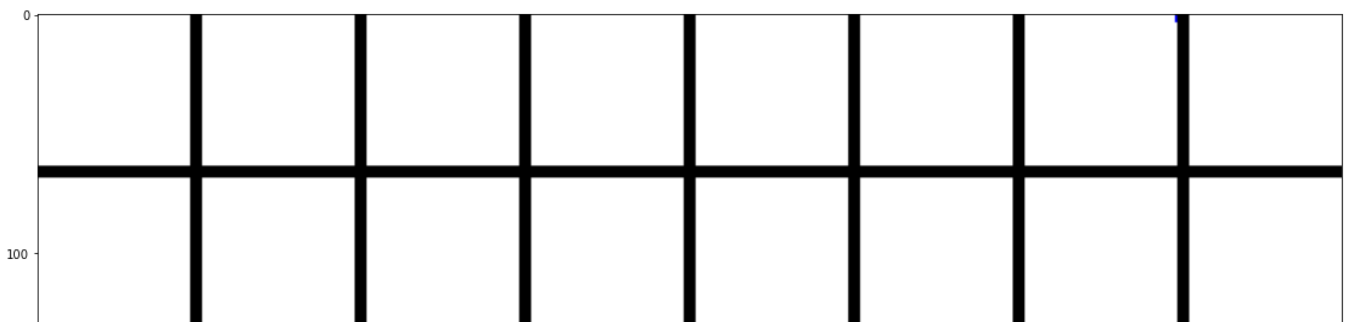

0
100
200
300
400

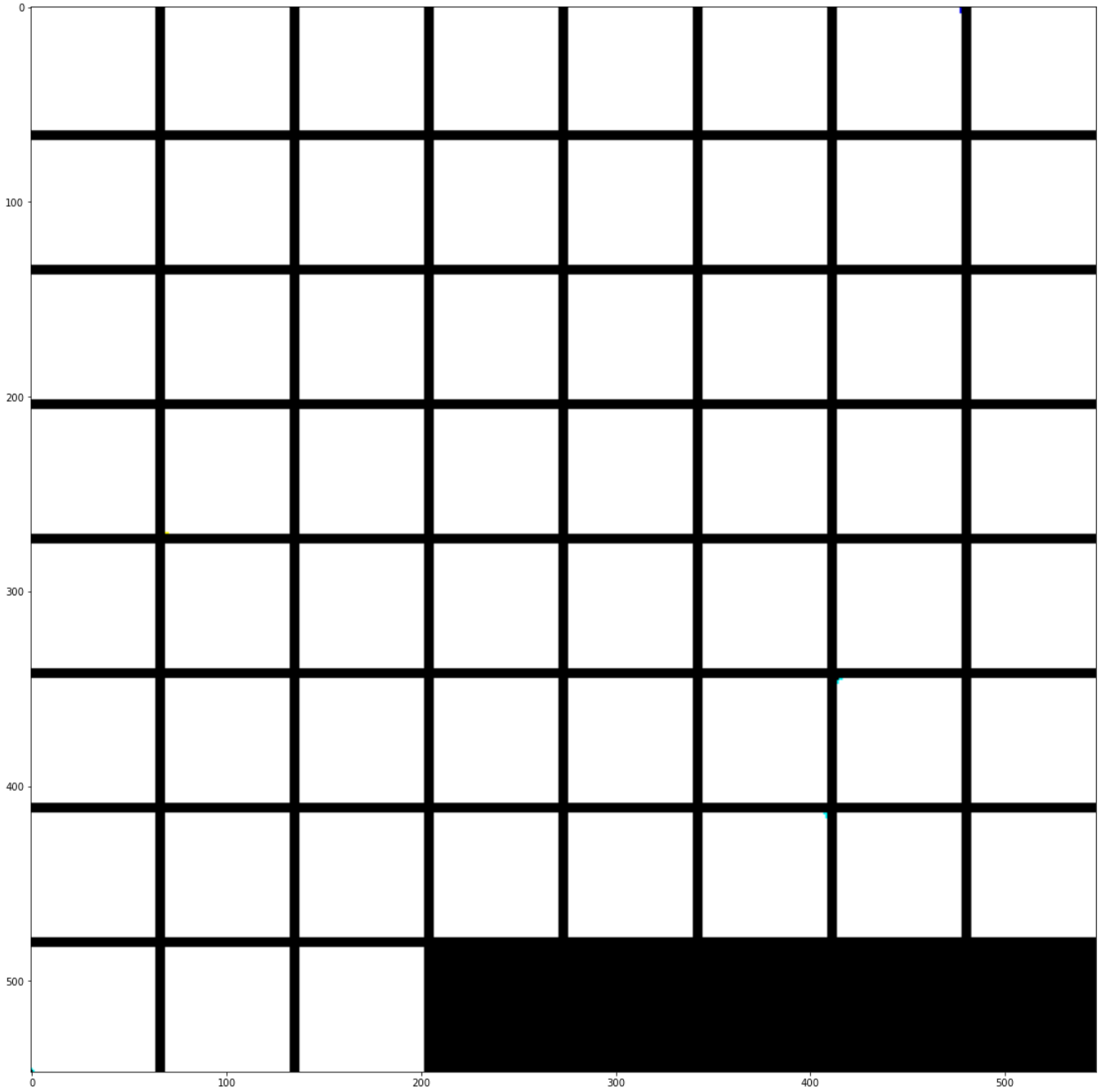

0
100
200
300
400
500
0
100
200
300
400
500

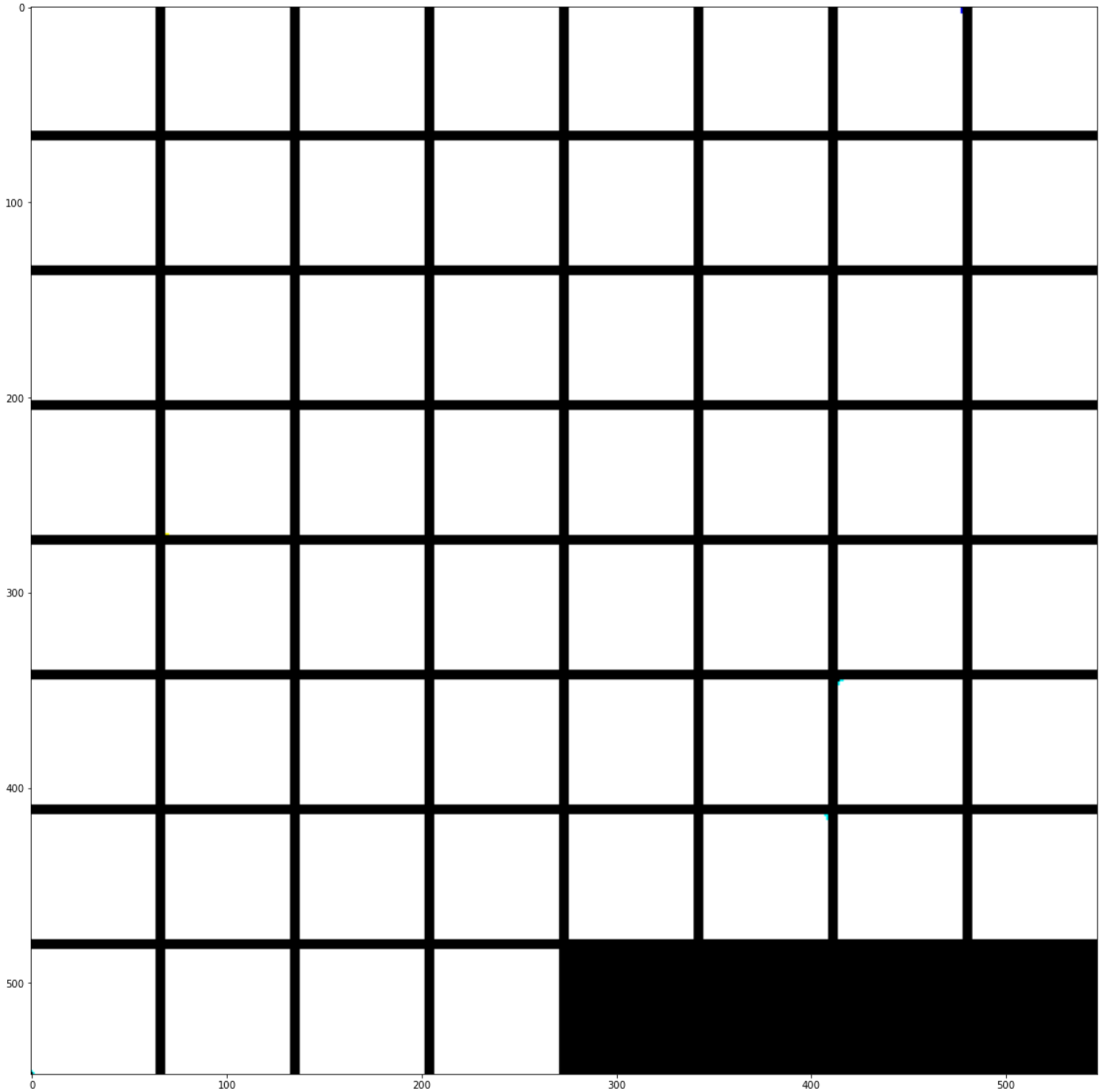

0
100
200
300
400
500

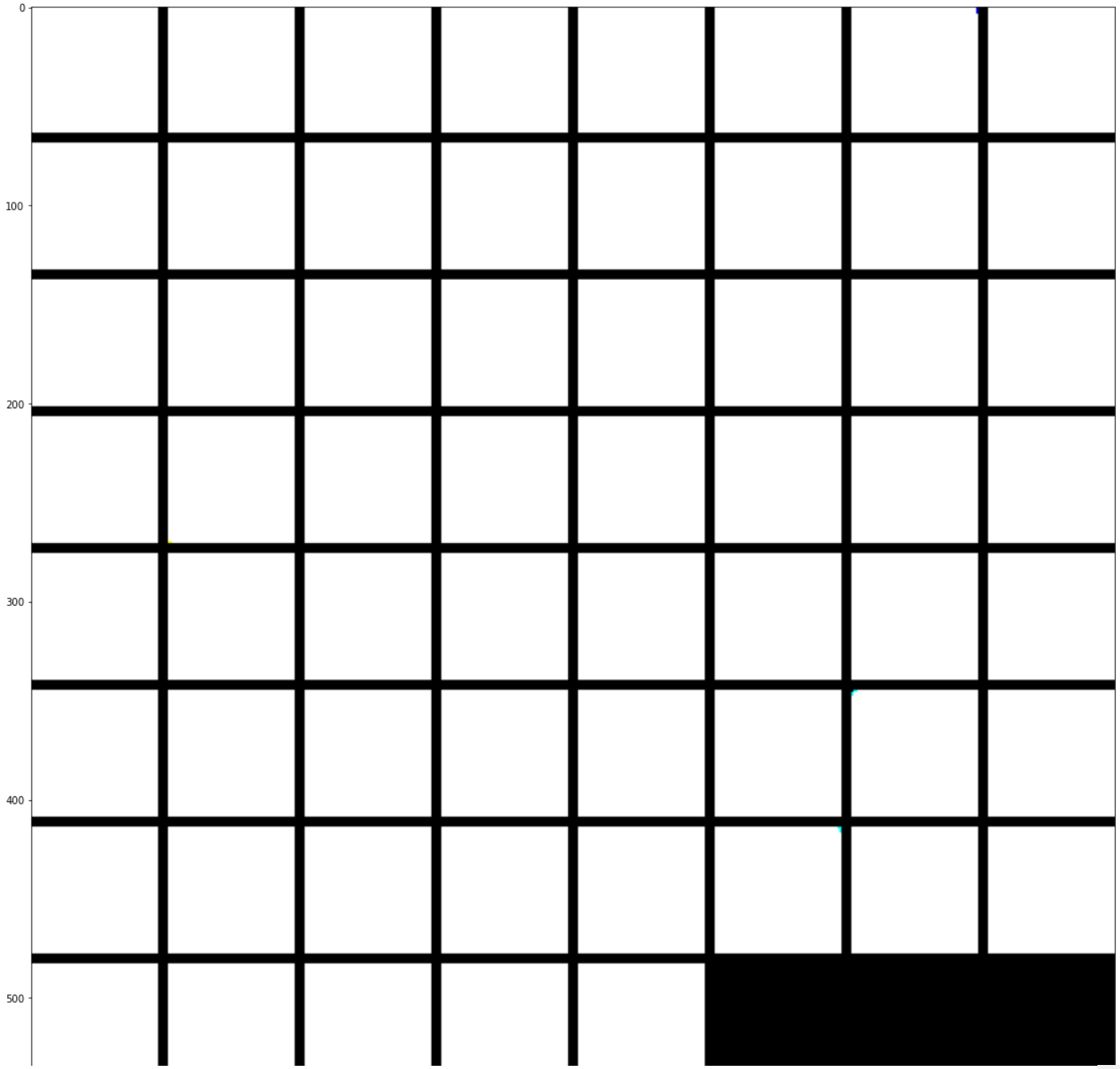

0
100
200
300
400

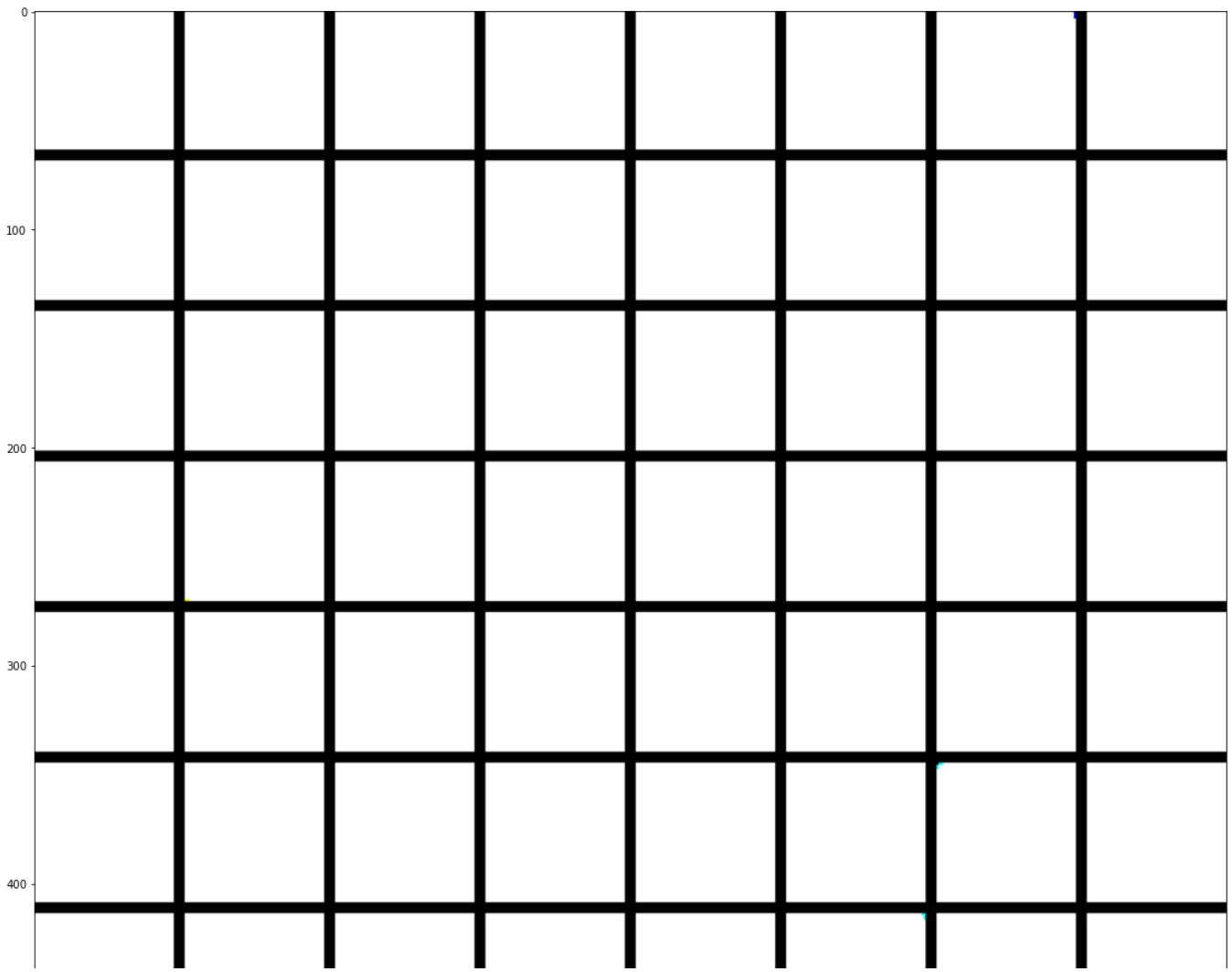

0
100
200
300

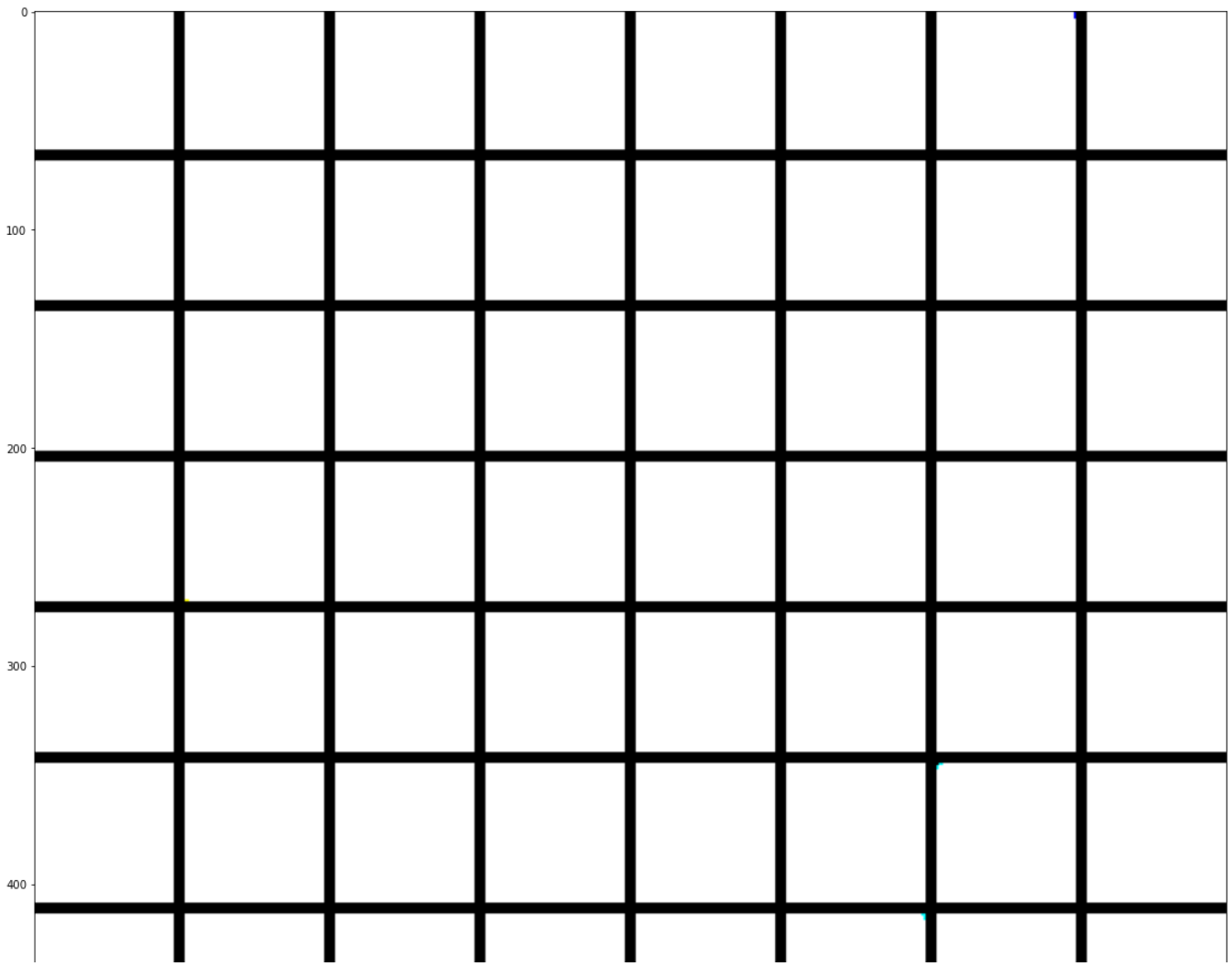

0
100

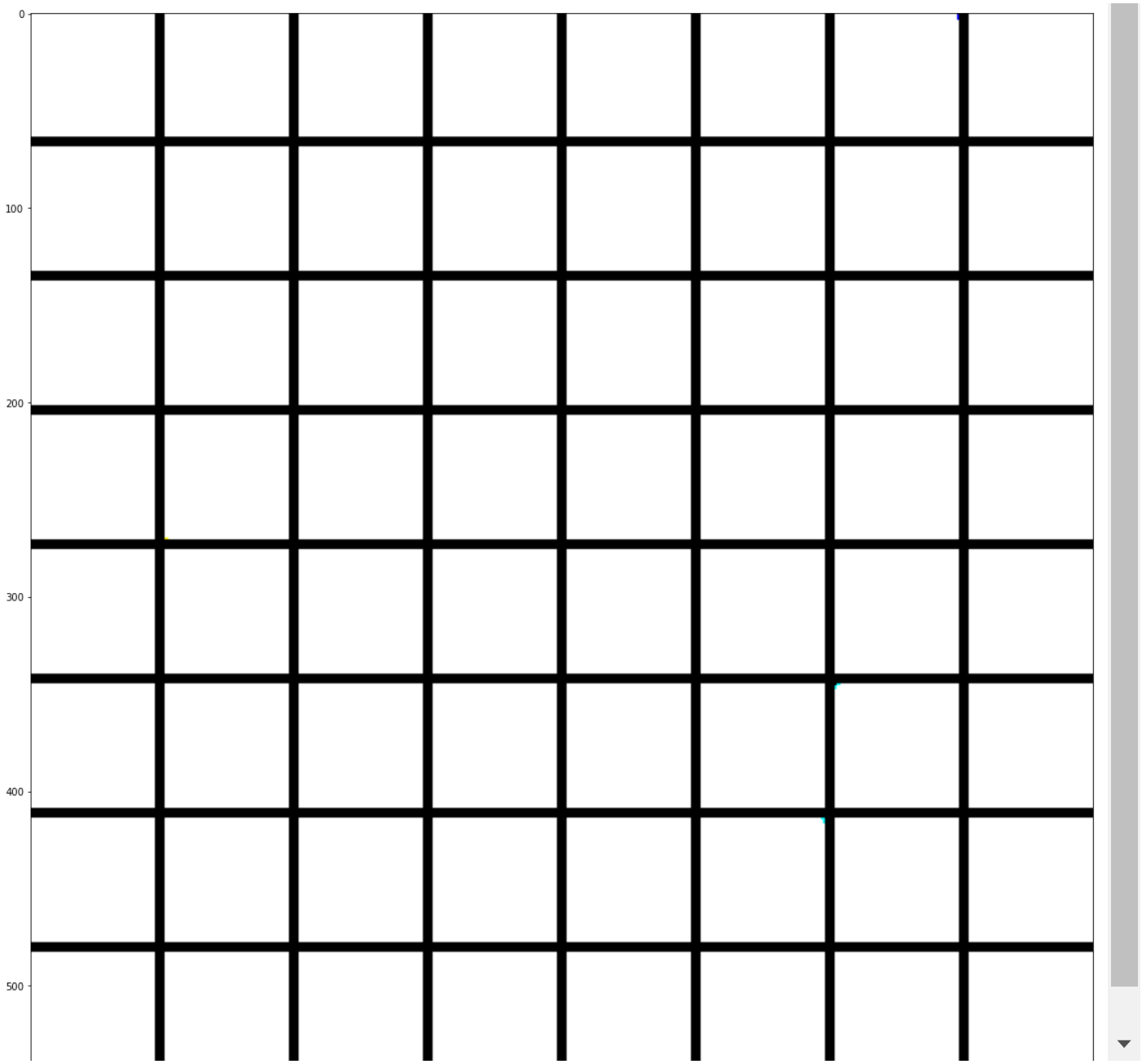

0
100
200
300
400
500
0
100
200
300
400
500


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


0
100
200
300
400
500


0
100
200
300
400


0
100
200
300
400


0
100
200
300
400
500

/