yes-no-classification-sliding-window

March 26, 2021

1 Yes/No Classification with sliding Window

In this example a simple one class classification model is trained to classify wether an image contains a 'brick' or not. It is trained on a small set of images of bricks and random images (not bricks) [2]. In a later stage a sliding window is used to classify probabilities and location of brick occurences in a given image.

1.1 Prepare the Data

The data is prepared as one 'brick' class and one of random images (not bricks). Those will be stored in:

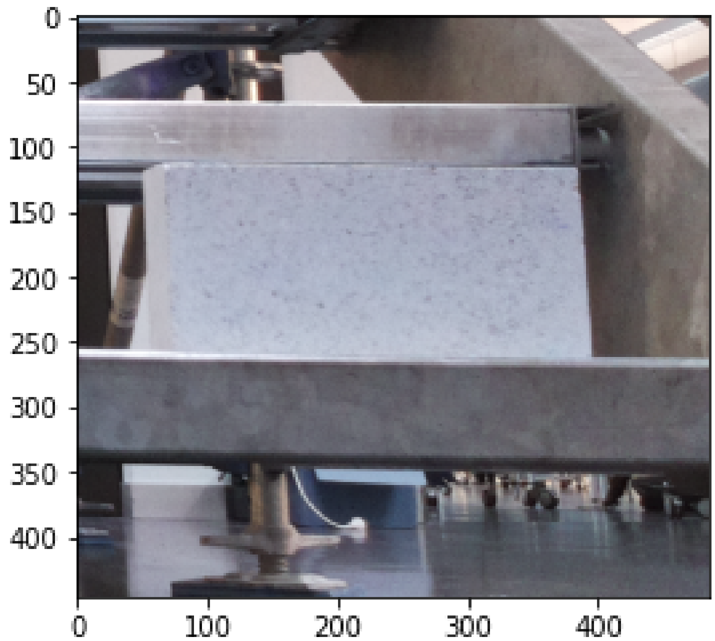
- `data/brick`
- `data/not_brick`

Here are some examples of raw data:

[1]:
```
import cv2
import matplotlib.pyplot as plt
from pathlib import Path

RAW_DATA = Path("raw_data/")
DATA = Path("data/")
```

[2]:
```
plt.imshow(
    cv2.imread(str(RAW_DATA / "brick" / "2021-03-05_10-17.png"))
)
```

[2]:
```
<matplotlib.image.AxesImage at 0x7f7881ea9610>
```



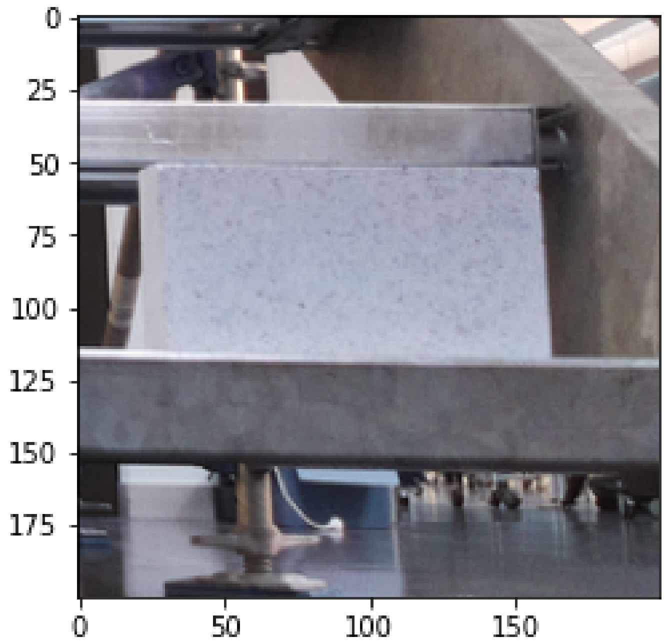
The images need to be the same size to fit into our neural network, this will be a tensor of (200, 200, 1). For this the images need to be converted to black and white and cropped/resized to 200/200

[3]:
```
# resize all brick images to 200x200 [3] png's
!ls -1 raw_data/brick/* | xargs -n 1 bash -c 'convert -resize 200x200! "$0"␣
↪"data/brick/$(basename ${0%.*}.png)"'
```

Data after processing:

[4]:
```
plt.imshow(
    cv2.imread(str(DATA / "brick" / "2021-03-05_10-17.png"))
)
```

[4]: <matplotlib.image.AxesImage at 0x7f78c157a790>



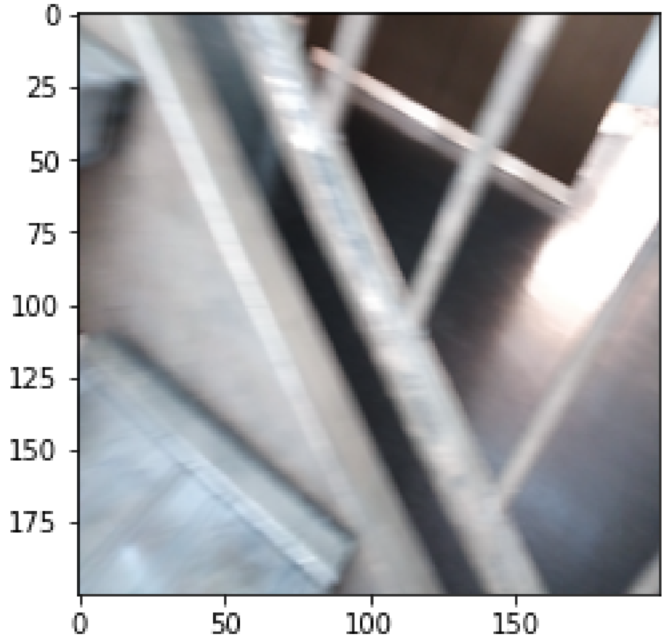
For the ‘no_brick’ data class some random images will be downloaded [2] in addition to the predefined ‘no_brick’ images.

[22]:
```
# download 200 additional random 'no_brick' images
import requests

for i in range(200):
    response = requests.get("https://picsum.photos/200/200/?random")
    if response.ok:
        (RAW_DATA / "not_brick" / f"not_brick_{i}.jpg").write_bytes(response.
 ↪content)
```

[5]:
```
# resize all not brick images to 200x200 [3] png's
!ls -1 raw_data/not_brick/* | xargs -n 1 bash -c 'convert -resize 200x200! "$0"␣
 ↪"data/not_brick/$(basename ${0%.*}.png)"'
```

[6]:
```
# example 'not_brick' image
plt.imshow(
    cv2.imread(str(DATA / "not_brick" / "IMG_20210304_103351_01.png"))
)
```

[6]: `<matplotlib.image.AxesImage at 0x7f78c14dd460>`



1.2 Training a CNN Model to classify the images

The training of the images classification is based on the 'cats vs docs' example from keras [1] and a question on stackoverflow for yes/no classification [2]. As a first step the data will be split into training and validation data to evaluate the model later on. To reduce the amount of overfitting data augmentation is used, this can easily be done with keras `ImageDataGenerator` [4].

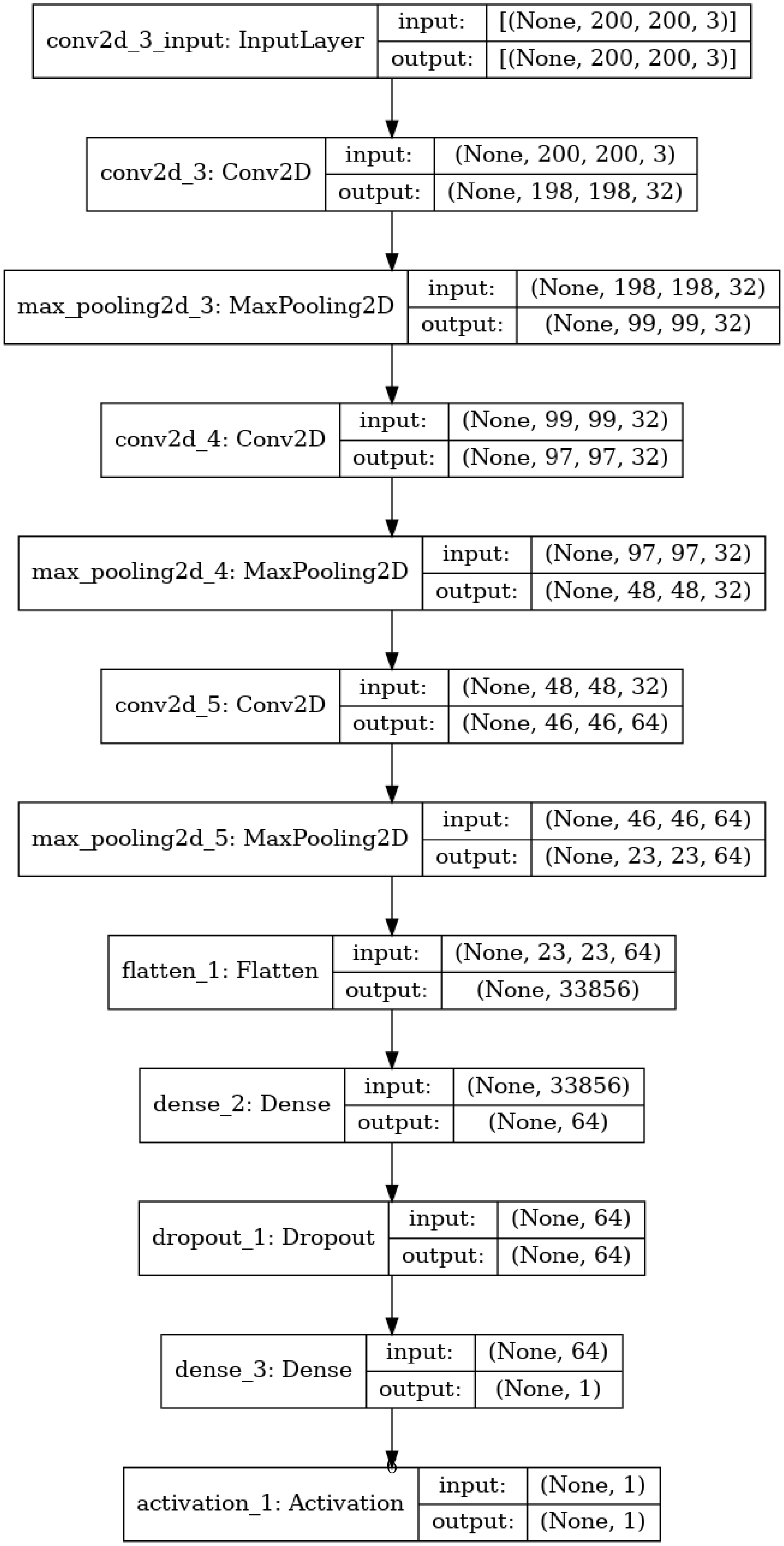
[7]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

[24]:
```
image_size = (200, 200)  # 200x200 png images

from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Activation

model = Sequential()

# use a (200, 200, 3) tensor as input for 200x200 rgb color images
```

```
model.add(Conv2D(32, (3, 3), input_shape = (*image_size, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(32, (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(units = 64, activation = 'relu'))

model.add(Dropout(0.5))

# output layer
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(
    optimizer = 'adam',
    loss = 'binary_crossentropy',
    metrics = ['binary_accuracy']
)
```

[25]:
```
keras.utils.plot_model(model, show_shapes=True)
```

[25]:


conv2d_3_input: InputLayer
input:
[(None, 200, 200, 3)]
output:
[(None, 200, 200, 3)]
conv2d_3: Conv2D
input:
(None, 200, 200, 3)
output:
(None, 198, 198, 32)
max_pooling2d_3: MaxPooling2D
input:
(None, 198, 198, 32)
output:
(None, 99, 99, 32)
conv2d_4: Conv2D
input:
(None, 99, 99, 32)
output:
(None, 97, 97, 32)
max_pooling2d_4: MaxPooling2D
input:
(None, 97, 97, 32)
output:
(None, 48, 48, 32)
conv2d_5: Conv2D
input:
(None, 48, 48, 32)
output:
(None, 46, 46, 64)
max_pooling2d_5: MaxPooling2D
input:
(None, 46, 46, 64)
output:
(None, 23, 23, 64)
flatten_1: Flatten
input:
(None, 23, 23, 64)
output:
(None, 33856)
dense_2: Dense
input:
(None, 33856)
output:
(None, 64)
dropout_1: Dropout
input:
(None, 64)
output:
(None, 64)
dense_3: Dense
input:
(None, 64)
output:
(None, 1)
activation_1: Activation
input:
(None, 1)
output:
(None, 1)

[26]:

```
from keras.preprocessing.image import ImageDataGenerator

batch_size = 32
validation_split = 0.2

# create train/validation split with augmented data [4], [5]
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=validation_split
)

train_generator = train_datagen.flow_from_directory(
    str(DATA),
    target_size=image_size,
    class_mode='binary',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    str(DATA),
    target_size=image_size,
    class_mode='binary',
    subset='validation'
)
```

```
Found 287 images belonging to 2 classes.
Found 71 images belonging to 2 classes.
```

[27]:

```
import numpy as np

epochs = 20

TRAIN_STEPS_PER_EPOCH = np.ceil(
    (train_generator.samples * (1 - validation_split) / batch_size) - 1
)
VALIDATIOON_STEPS_PER_EPOCH = np.ceil(
    (validation_generator.samples * validation_split / batch_size) - 1
)

callbacks = [
    keras.callbacks.ModelCheckpoint("save_at_{epoch}.h5"),
]
```

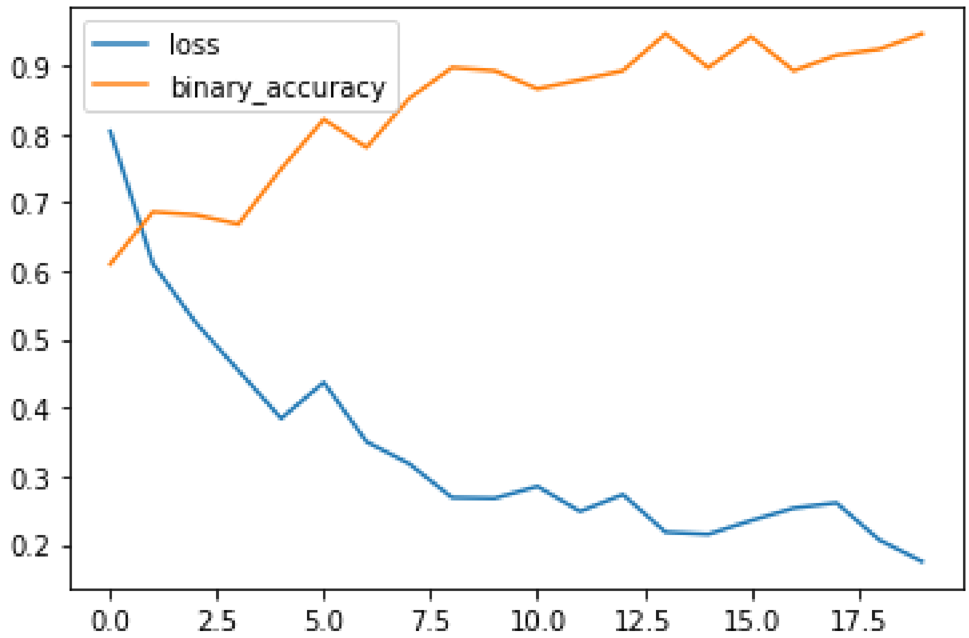
```
history = model.fit(
    train_generator,
    epochs=epochs,
    steps_per_epoch = TRAIN_STEPS_PER_EPOCH,
    validation_data = validation_generator,
    validation_steps = VALIDATIOON_STEPS_PER_EPOCH,
    callbacks=callbacks
)
```

```
Epoch 1/20
7/7 [==============================] - 2s 290ms/step - loss: 0.8001 -
binary_accuracy: 0.6611
Epoch 2/20
7/7 [==============================] - 2s 241ms/step - loss: 0.6224 -
binary_accuracy: 0.6911
Epoch 3/20
7/7 [==============================] - 2s 239ms/step - loss: 0.5191 -
binary_accuracy: 0.7141
Epoch 4/20
7/7 [==============================] - 2s 236ms/step - loss: 0.4624 -
binary_accuracy: 0.6315
Epoch 5/20
7/7 [==============================] - 2s 236ms/step - loss: 0.3847 -
binary_accuracy: 0.6980
Epoch 6/20
7/7 [==============================] - 2s 237ms/step - loss: 0.4297 -
binary_accuracy: 0.8257
Epoch 7/20
7/7 [==============================] - 2s 235ms/step - loss: 0.3959 -
binary_accuracy: 0.7441
Epoch 8/20
7/7 [==============================] - 2s 244ms/step - loss: 0.3453 -
binary_accuracy: 0.8394
Epoch 9/20
7/7 [==============================] - 2s 237ms/step - loss: 0.2607 -
binary_accuracy: 0.8964
Epoch 10/20
7/7 [==============================] - 2s 242ms/step - loss: 0.2797 -
binary_accuracy: 0.8653
Epoch 11/20
7/7 [==============================] - 2s 236ms/step - loss: 0.2598 -
binary_accuracy: 0.8615
Epoch 12/20
7/7 [==============================] - 2s 236ms/step - loss: 0.2304 -
binary_accuracy: 0.8984
Epoch 13/20
```

```
7/7 [==============================] - 2s 235ms/step - loss: 0.2912 -
binary_accuracy: 0.9027
Epoch 14/20
7/7 [==============================] - 2s 231ms/step - loss: 0.2384 -
binary_accuracy: 0.9460
Epoch 15/20
7/7 [==============================] - 2s 237ms/step - loss: 0.2219 -
binary_accuracy: 0.8876
Epoch 16/20
7/7 [==============================] - 2s 237ms/step - loss: 0.2261 -
binary_accuracy: 0.9362
Epoch 17/20
7/7 [==============================] - 2s 237ms/step - loss: 0.2128 -
binary_accuracy: 0.9205
Epoch 18/20
7/7 [==============================] - 2s 232ms/step - loss: 0.2457 -
binary_accuracy: 0.9178
Epoch 19/20
7/7 [==============================] - 2s 233ms/step - loss: 0.2500 -
binary_accuracy: 0.9091
Epoch 20/20
7/7 [==============================] - 2s 234ms/step - loss: 0.1811 -
binary_accuracy: 0.9512
```

[28]:
```python
import pandas as pd

pd.DataFrame(history.history).plot()
```

[28]:
```
<AxesSubplot:>
```



[30]:
```
# let's print some examples
image = keras.preprocessing.image.load_img(
    str(DATA / "brick" / "2021-03-05_10-17.png")
)
img_array = keras.preprocessing.image.img_to_array(image)
img_array = tf.expand_dims(img_array, 0)  # Create batch axis

predictions = model.predict(img_array)
predictions
```

[30]: array([[0.]], dtype=float32)

[34]:
```
# let's print some examples
image = keras.preprocessing.image.load_img(
    str(DATA / "not_brick" / "IMG_20210304_103351_01.png")
)
img_array = keras.preprocessing.image.img_to_array(image)
img_array = tf.expand_dims(img_array, 0)  # Create batch axis

predictions = model.predict(img_array)
predictions
```

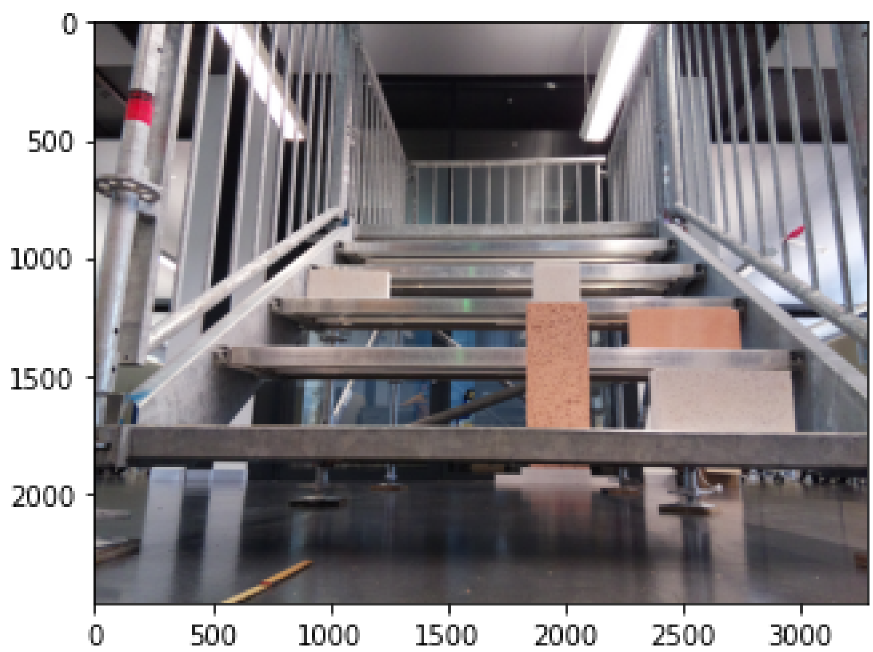
[34]: array([[1.]], dtype=float32)

1.3 Sliding window heatmap generation

This simple 'brick' detection model can now be used to generate a heatmap based on a sliding window over an image containing bricks.

[61]:
```
image = cv2.cvtColor(
    cv2.imread(str(RAW_DATA / "stair" / "validation" /␣
 ↪"162702ac-7fe2-4192-b018-3c8473a828a3.jpg")),
    cv2.COLOR_BGR2RGB
)
plt.imshow(image)
```

[61]: <matplotlib.image.AxesImage at 0x7f7824207850>



[62]:
```
scale_percent = 25
width = int(image.shape[1] * scale_percent / 100)
height = int(image.shape[0] * scale_percent / 100)

# [6] resize image to match the brick size
image = cv2.resize(image, (width, height), interpolation = cv2.INTER_AREA)
```

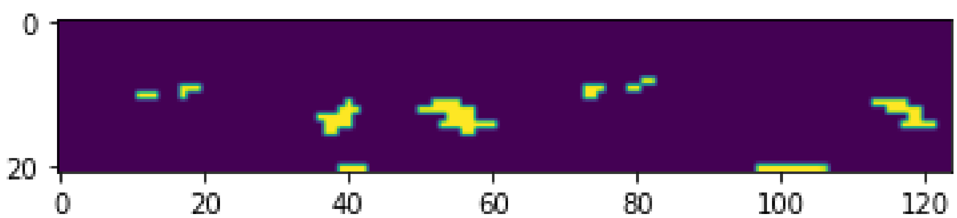
[63]:
```
# slide a window across the image [7]

def sliding_window(image, window_size, step=1):
    for y in range(0, image.shape[0] - window_size[1], step):
        for x in range(0, image.shape[1] - window_size[0], step):
```

```
            yield (x, y, image[y:y + window_size[1], x:x + window_size[0]])
```

[83]:
```
detection_image = image.copy()
heatmap = []

for (x, y, region_of_interest) in sliding_window(image, image_size, step=10):
    img_array = keras.preprocessing.image.img_to_array(region_of_interest)
    img_array = tf.expand_dims(img_array, 0)  # Create batch axis

    predictions = model.predict(img_array)
    if predictions[0] < 0.5:
        # well, it's a brick
        cv2.rectangle(detection_image, (x, y), (x+200, y+200), (255, 0, 0), 2)

    y_index = int(y / 20)
    if len(heatmap) <= y_index:
        heatmap.append([])

    heatmap[y_index].append(int(255 * (1 - predictions[0])))
```

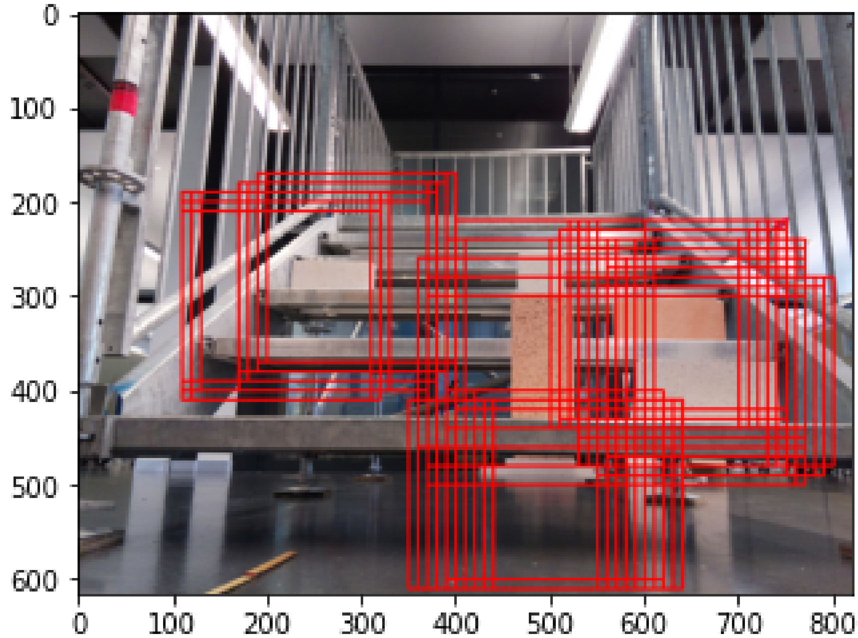
[84]:
```
plt.imshow(heatmap)
```

[84]: <matplotlib.image.AxesImage at 0x7f77a06ce7c0>



[85]:
```
plt.imshow(detection_image)
```

[85]: <matplotlib.image.AxesImage at 0x7f782412b430>



1.4 Conclusion

Even dough it is simpler to train a classification network and turn it into an object detection framework [7], there are a few downsides. Mainly performance is really bad with the sliding window method. In addition to bad performance the network does not scale. In order to scale the network image pyramids could be used together with non maxima supression [7]. However those problems have already been solved with Regression

1.4.1 References

- [1] https://keras.io/examples/vision/image_classification_from_scratch/
- [2] https://stackoverflow.com/questions/57309958/one-class-classification-using-keras-and-python
- [3] https://stackoverflow.com/a/20439152
- [4] https://keras.io/api/preprocessing/image/#imagedatagenerator-class
- [5] https://stackoverflow.com/questions/42443936/keras-split-train-test-set-when-using-imagedatagenerator
- [6] https://www.tutorialkart.com/opencv/python/opencv-python-resize-image/
- [7] https://www.pyimagesearch.com/2020/06/22/turning-any-cnn-image-classifier-into-an-object-detector-with-keras-tensorflow-and-opencv/
- [8] https://towardsdatascience.com/building-the-hotdog-not-hotdog-classifier-from-hbos-silicon-valley-c0cb2317711f
- [9] https://github.com/J-Yash/Hotdog-Not-Hotdog/blob/master/Hotdog_classifier_transfer_learning.ipynb