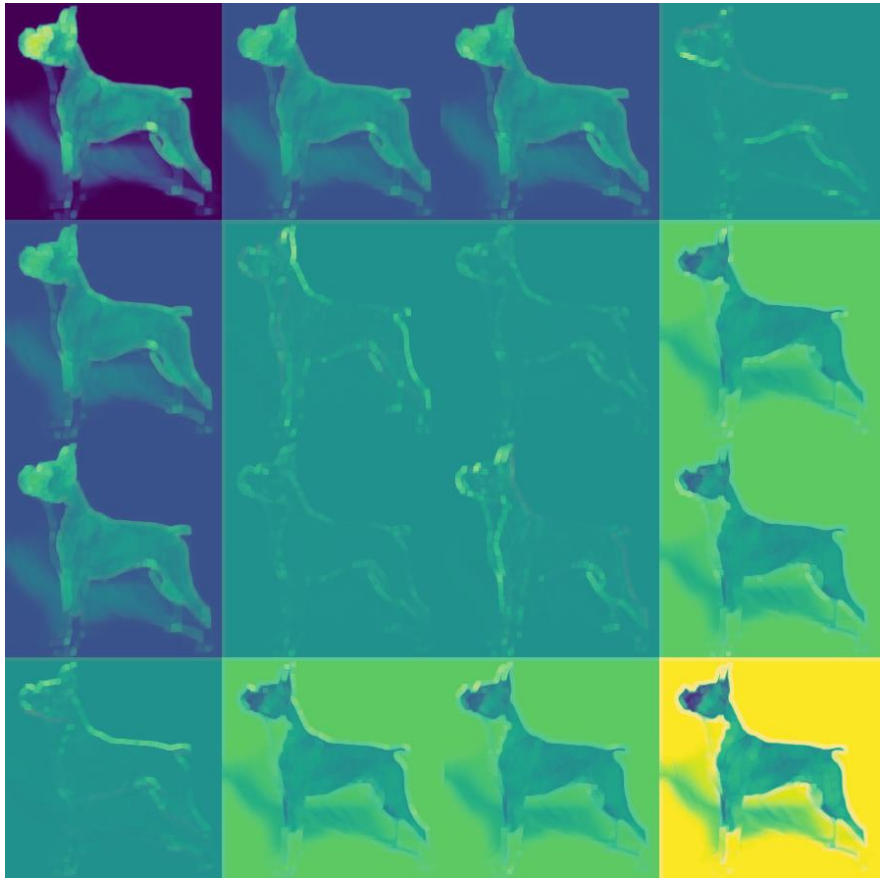


Basics of Computer Vision

Dmitry Ryabokon, github.com/dryabokon

Content

1. CNNs
1.1 CNNs out of box
1.2 Basic operations
1.3 Feature extraction
1.4 Transfer learning
1.5 Auto-encoders
1.6. GANs
1.7 Fine Tuning
1.8 Benchmarking the CNNs

2. Projective Geometry
2.1 KeyPoints detection
2.2 Matching the KeyPoints
2.3 Homography
2.4 Blending
2.5 Camera calibration
2.6 Stereopair rectification
2.7 Pose estimation

3. OpenGL
3.1 Aruco
3.2 Augmented reality

4. Object detection
4.1 YOLO
4.2 SSD
4.3 Mask RCNN
4.4 CAM
4.5 Captioning

5. Computer Vision Applications
5.1 Stereo Vision
5.2 Face Swap
5.3 Face Filter
5.4 Style Transfer

Part 1: CNNs

CNNs out of box
Basic operations
Feature extraction
Transfer learning
Auto-encoders
GANs
Fine Tuning
Benchmarking the CNNs

CNNs out of box

CNNs out of box

The popular networks

Classification

- LeNet Model
- AlexNet Model
- VGG Model
- ResNet Paper
- YOLO9000 Paper
- DenseNet Paper

Segmentation

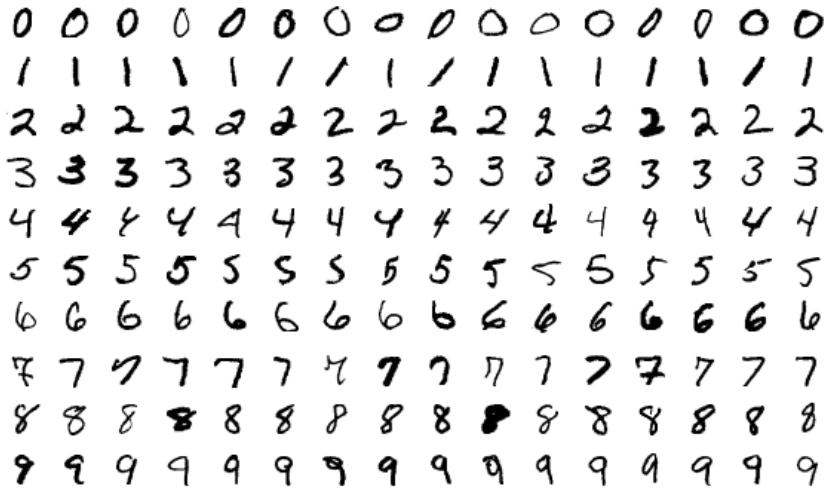
- FCN8 Paper
- SegNet Paper
- U-Net Paper
- E-Net Paper
- ResNetFCN Paper
- PSPNet Paper
- Mask RCNN Paper

Detection

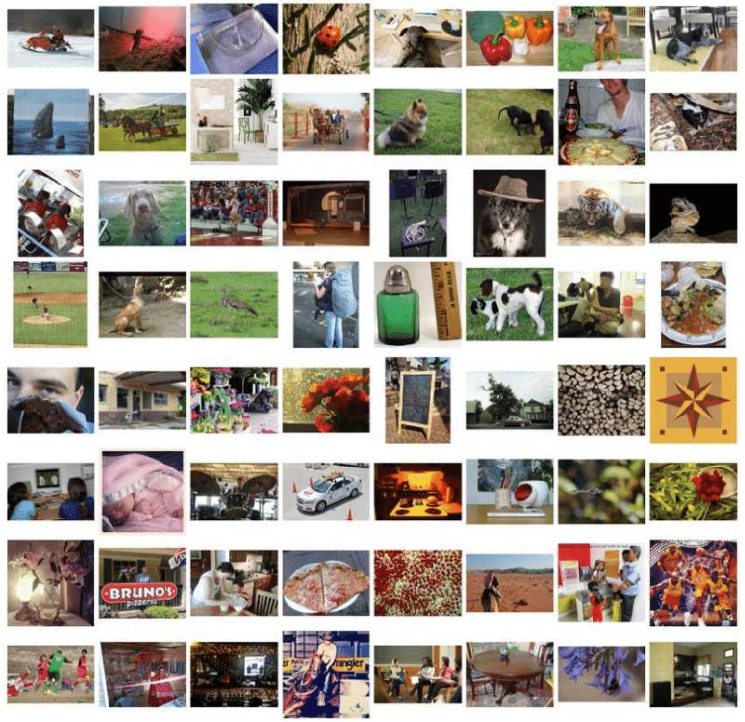
- Faster RCNN Paper
- SSD Paper
- YOLOv2 Paper
- R-FCN Paper

CNNs out of box

Some datasets available for research



MNIST: 10 classes, ~7000 ex. per class

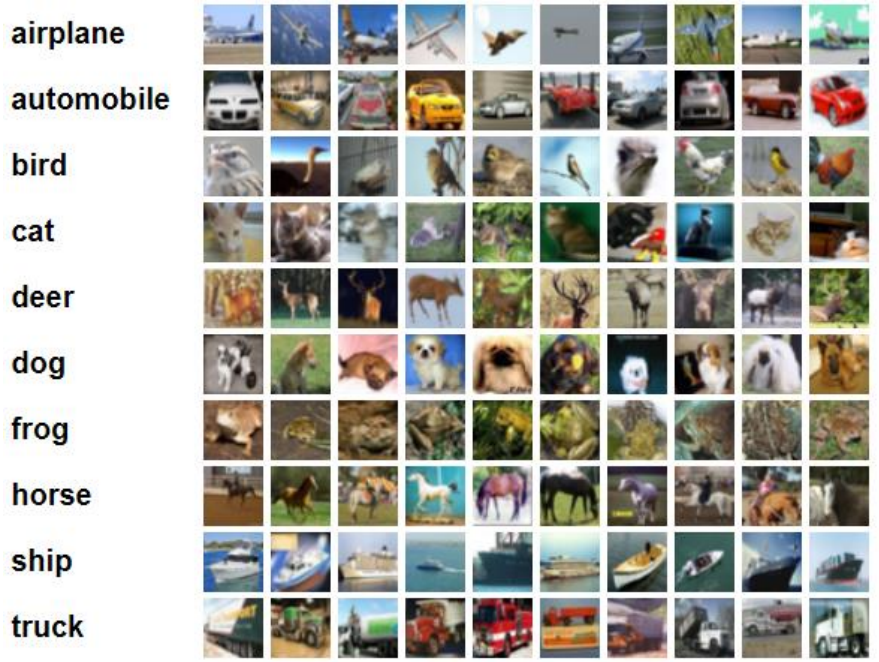


ImageNet: 1000 classes, ~100 ex per class

CNNs out of box

Some datasets available for research

The Street View House Numbers
10 classes, ~2000 ex. per class

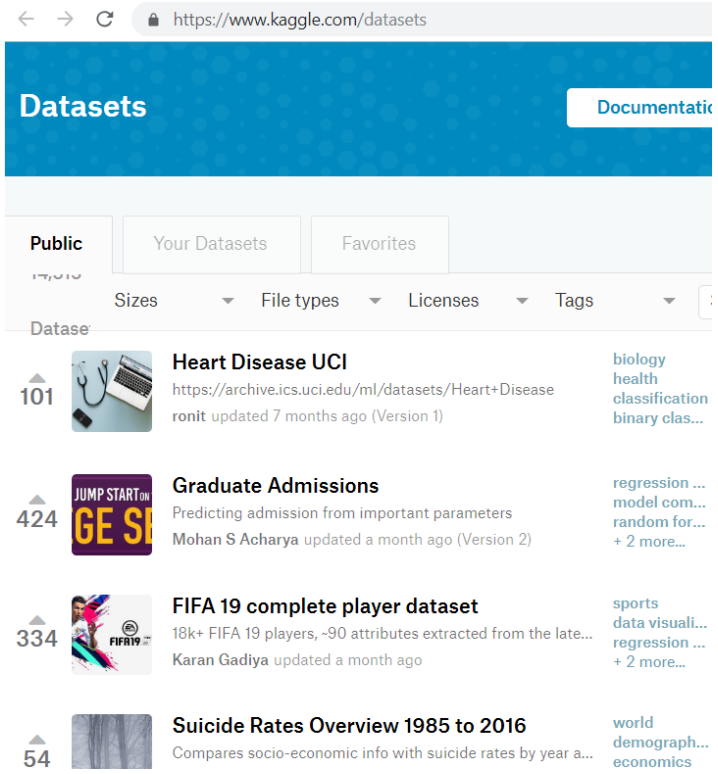


CIFAR: 10 classes, 6000 ex. per class
100 classes, 600 ex per class

CNNs out of box

Some datasets available for research

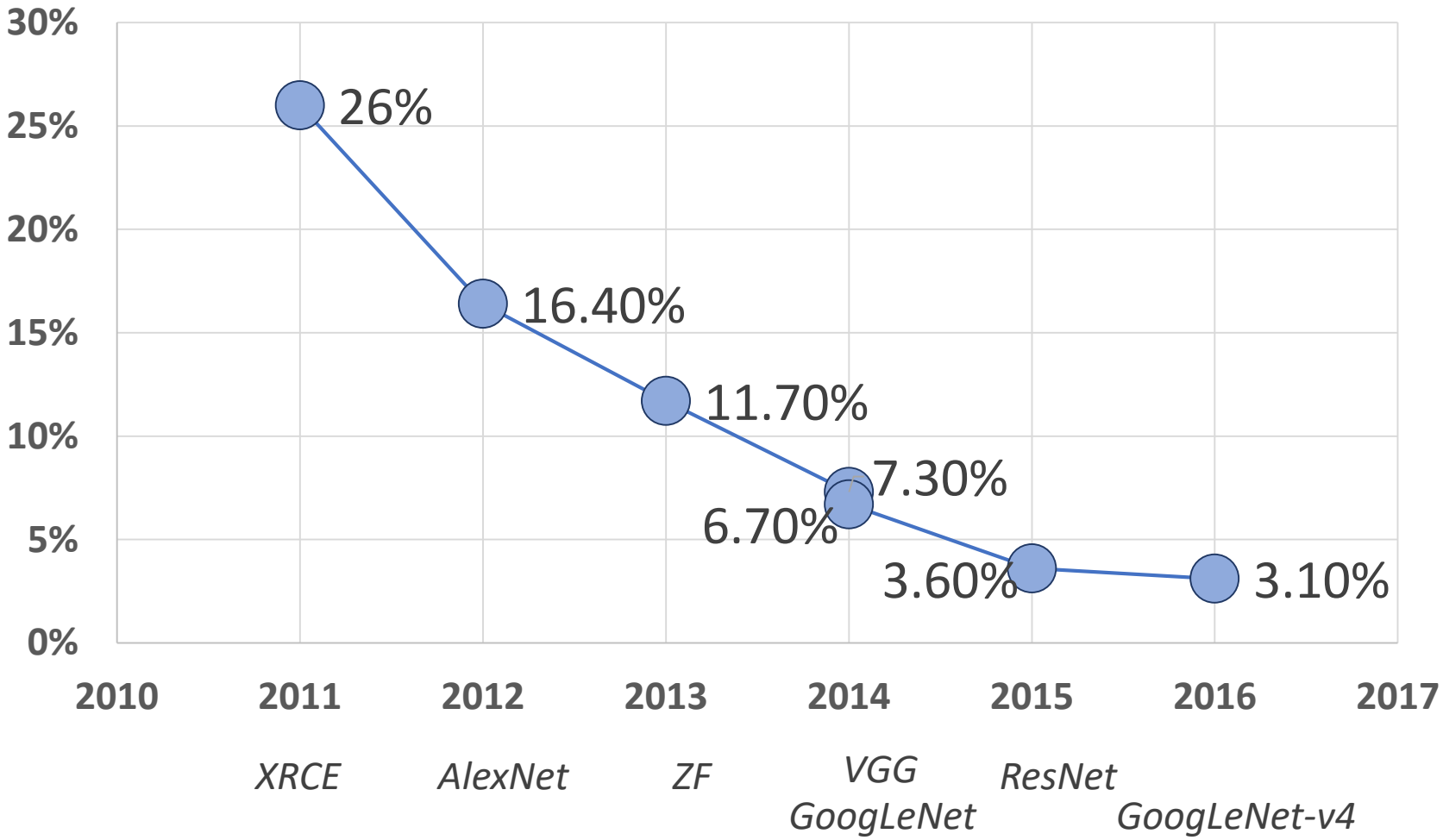
Olivetti database: 40 classes



.. and much more @ kaggle

CNNs out of box

ImageNet Classification error



CNNs out of box

Usage of the neural networks

identify the name of a street (in France) from an image

compressing and decompressing images

semantic image segmentation

real-world image text extraction.

image classification

image-to-text

unsupervised learning

3D object reconstruction

image matching and retrieval

automatic speech recognition

localizing and identifying multiple objects in a single image

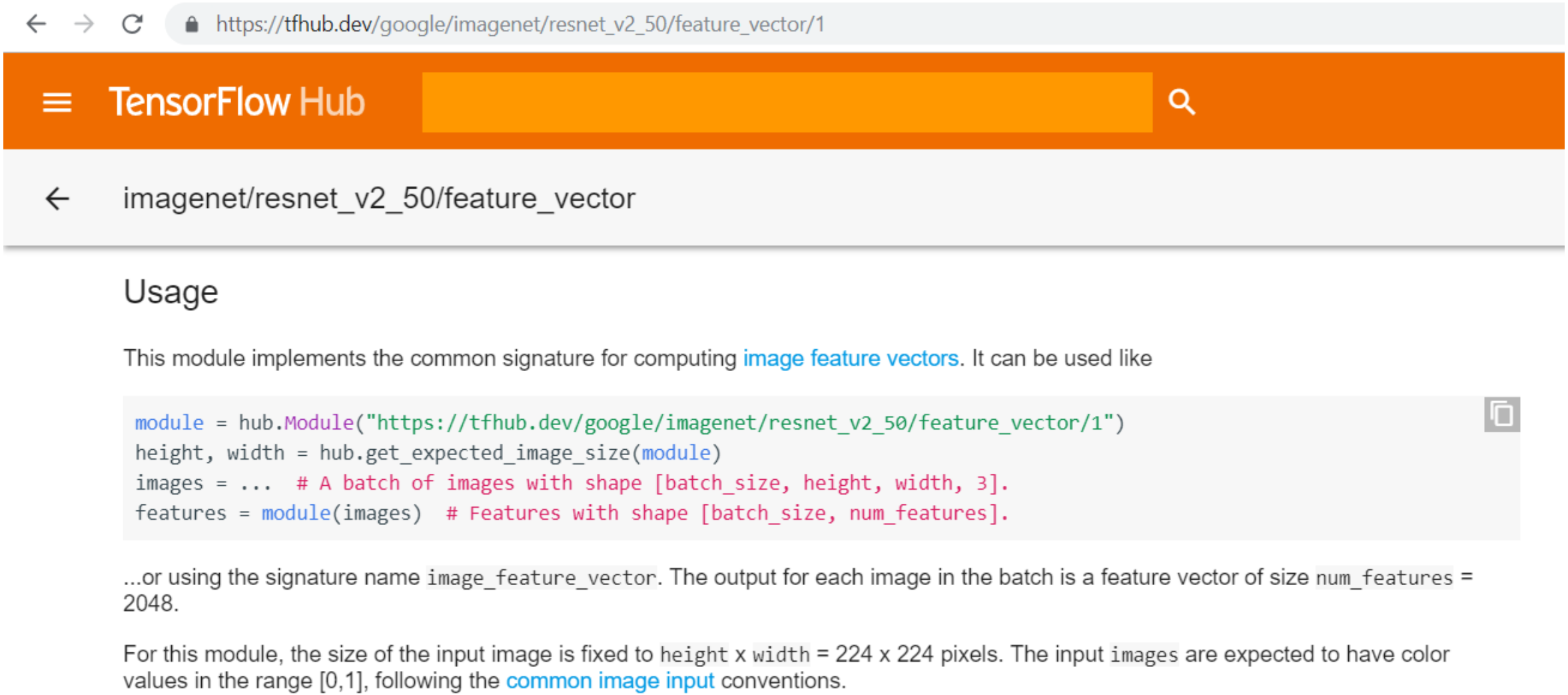
computer vision

discovery of latent 3D keypoints

predicting future video frames

https://github.com/keras-team/keras/tree/master/examples
https://github.com/tensorflow/models/tree/master/research

CNNs out of box



CNNs out of box

```
import tensorflow_hub as hub
import urllib.request
import cv2
import numpy
import json
import tensorflow as tf
# ----------------------------------------------------------------------------------------------------------
URL = 'https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json'
data = json.loads(urllib.request.urlopen(URL).read().decode())
class_names = [data['%d'%i][1] for i in range(0,999)]
# ----------------------------------------------------------------------------------------------------------
def example_predict():

    module = hub.Module("https://tfhub.dev/google/imagenet/resnet_v2_50/classification/1")
    img = cv2.imread('data/ex-natural/dog/dog_0000.jpg')
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (224, 224)).astype(numpy.float32)
    img = numpy.array([img]).astype(numpy.float32) / 255.0
    sess = tf.Session()
    sess.run(tf.global_variables_initializer())
    sess.run(tf.tables_initializer())

    outputs = module(dict(images=img), signature="image_classification", as_dict=True)
    prob = outputs["default"].eval(session=sess)[0]
    idx = numpy.argsort(-prob)[0]

    print(class_names[idx], prob[idx])
    sess.close()

    return
# ----------------------------------------------------------------------------------------------------------
if __name__ == '__main__':
    example_predict()
```

CNNs out of box

https://keras.io/getting-started/faq/#how-can-i-use-pre-trained-models-in-keras

Keras Documentation

Search docs

Home

Why use Keras

GETTING STARTED

Guide to the Sequential model

Guide to the Functional API

FAQ

How should I cite Keras?

How can I run Keras on GPU?

How can I run a Keras model on multiple GPUs?

What does "sample", "batch", "epoch" mean?

How can I save a Keras model?

Why is the training loss much higher than the testing loss?

How can I obtain the output of an intermediate layer?

How can I use Keras with datasets that don't fit in memory?

How can I interrupt training when the validation loss isn't decreasing

How can I use pre-trained models in Keras?

Code and pre-trained weights are available for the following image classification models:

- Xception
- VGG16
- VGG19
- ResNet50
- Inception v3
- Inception-ResNet v2
- MobileNet v1
- DenseNet
- NASNet
- MobileNet v2

They can be imported from the module `keras.applications`:

```
from keras.applications.xception import Xception
from keras.applications.vgg16 import VGG16
from keras.applications.vgg19 import VGG19
from keras.applications.resnet50 import ResNet50
from keras.applications.inception_v3 import InceptionV3
from keras.applications.inception_resnet_v2 import InceptionResNetV2
from keras.applications.mobilenet import MobileNet
from keras.applications.densenet import DenseNet121
from keras.applications.densenet import DenseNet169
from keras.applications.densenet import DenseNet201
from keras.applications.nasnet import NASNetLarge
from keras.applications.nasnet import NASNetMobile
from keras.applications.mobilenet_v2 import MobileNetV2

model = VGG16(weights='imagenet', include_top=True)
```

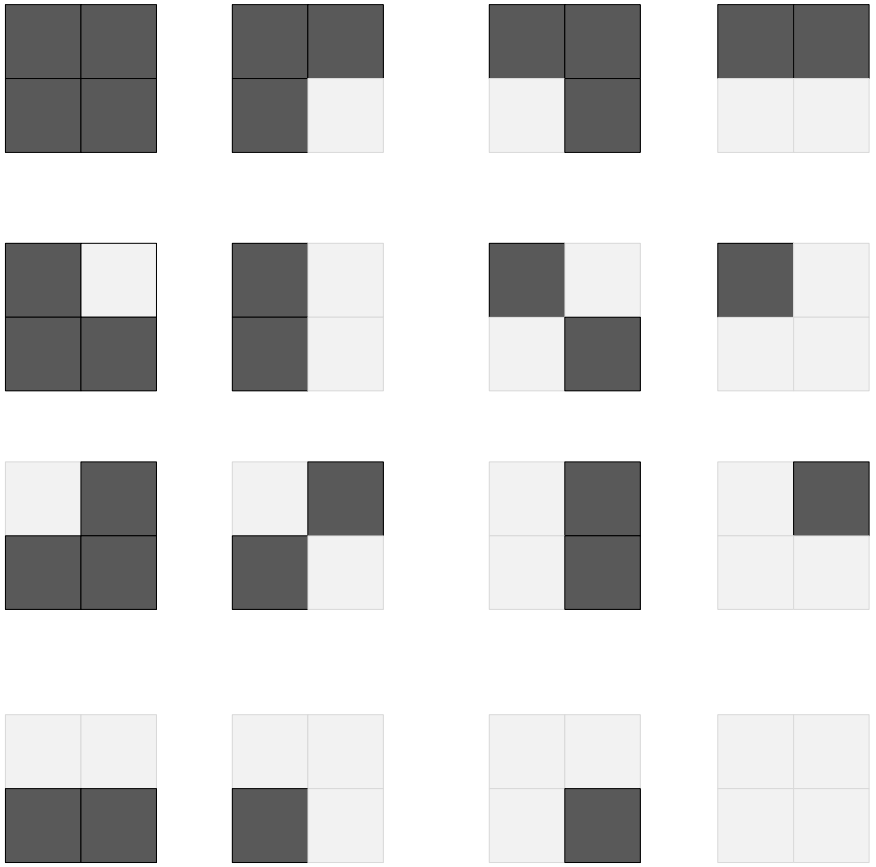
CNNs out of box

```
from keras.applications import MobileNet
from keras.applications.xception import Xception
from keras.applications.mobilenet import preprocess_input
import urllib.request
import cv2
import numpy
import json
from keras import backend as K
K.set_image_dim_ordering('tf')
# ---------------------------------------------------------------------------------------------------------------------
URL = 'https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json'
data = json.loads(urllib.request.urlopen(URL).read().decode())
class_names = [data['%d'%i][1] for i in range(0,999)]
# ---------------------------------------------------------------------------------------------------------------------
def example_predict():

    CNN = MobileNet()
    #CNN = Xception()

    img = cv2.imread('data/ex-natural/dog/dog_0000.jpg')
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (224, 224)).astype(numpy.float32)

    prob = CNN.predict(preprocess_input(numpy.array([img])))
    idx = numpy.argsort(-prob[0])[0]
    print(class_names[idx], prob[0, idx])

    return
# ---------------------------------------------------------------------------------------------------------------------
if __name__ == '__main__':
    example_predict()
```

Basic operations

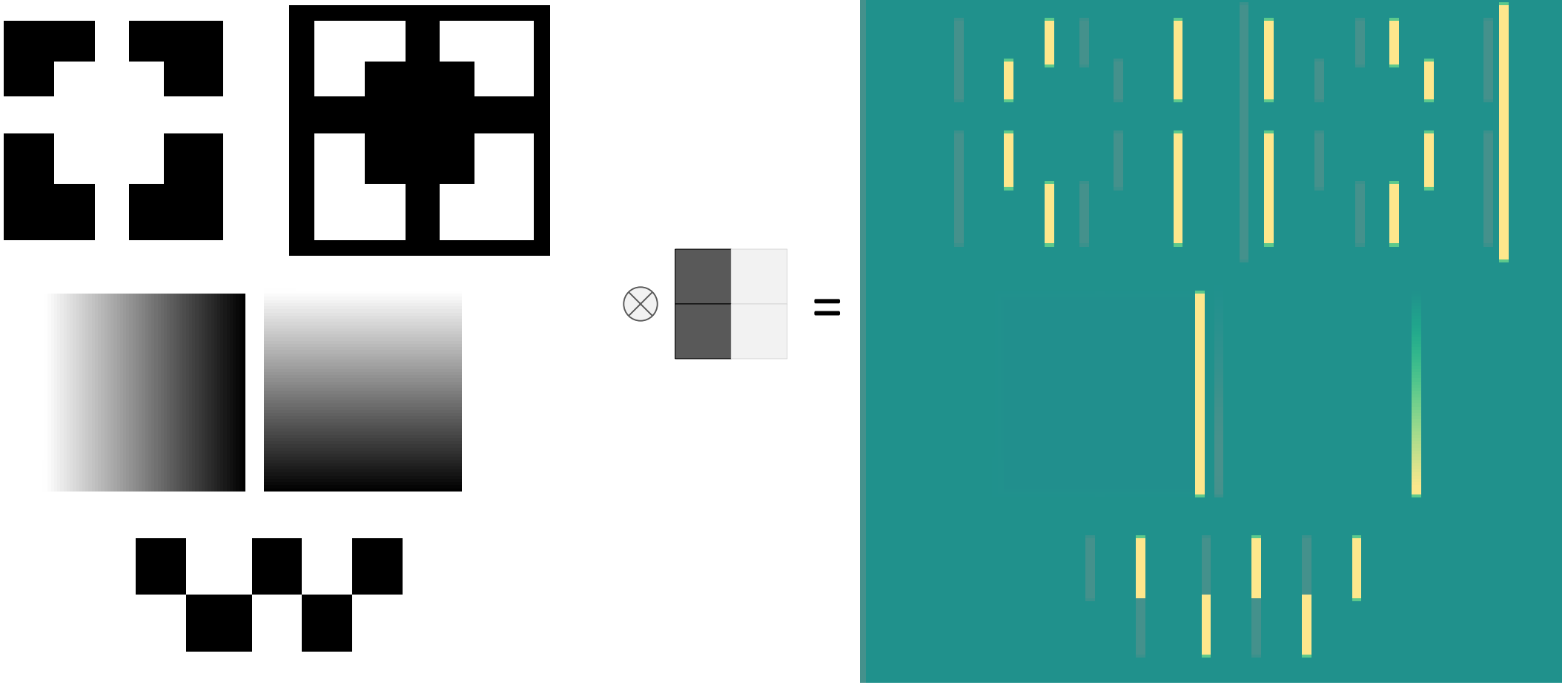
Basic operations

Convolution



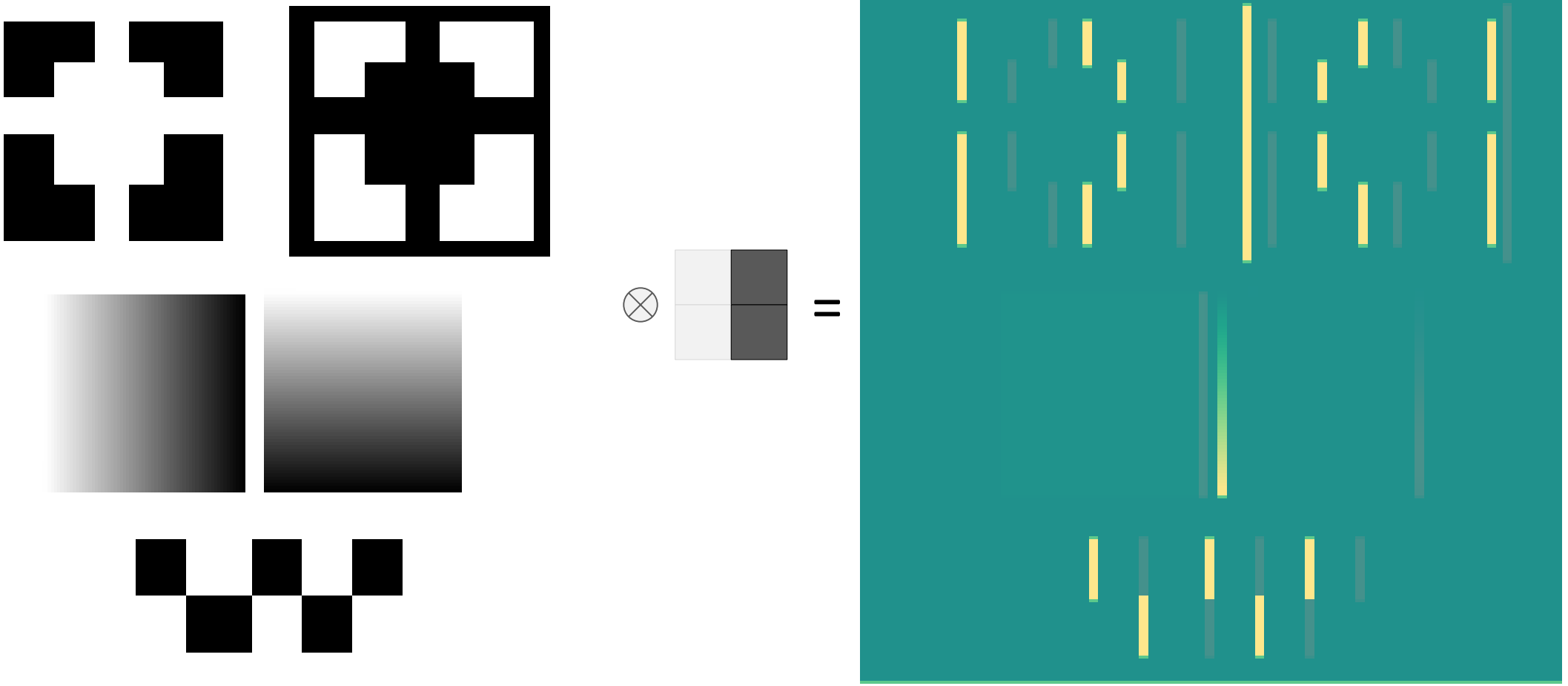
Basic operations

Convolution



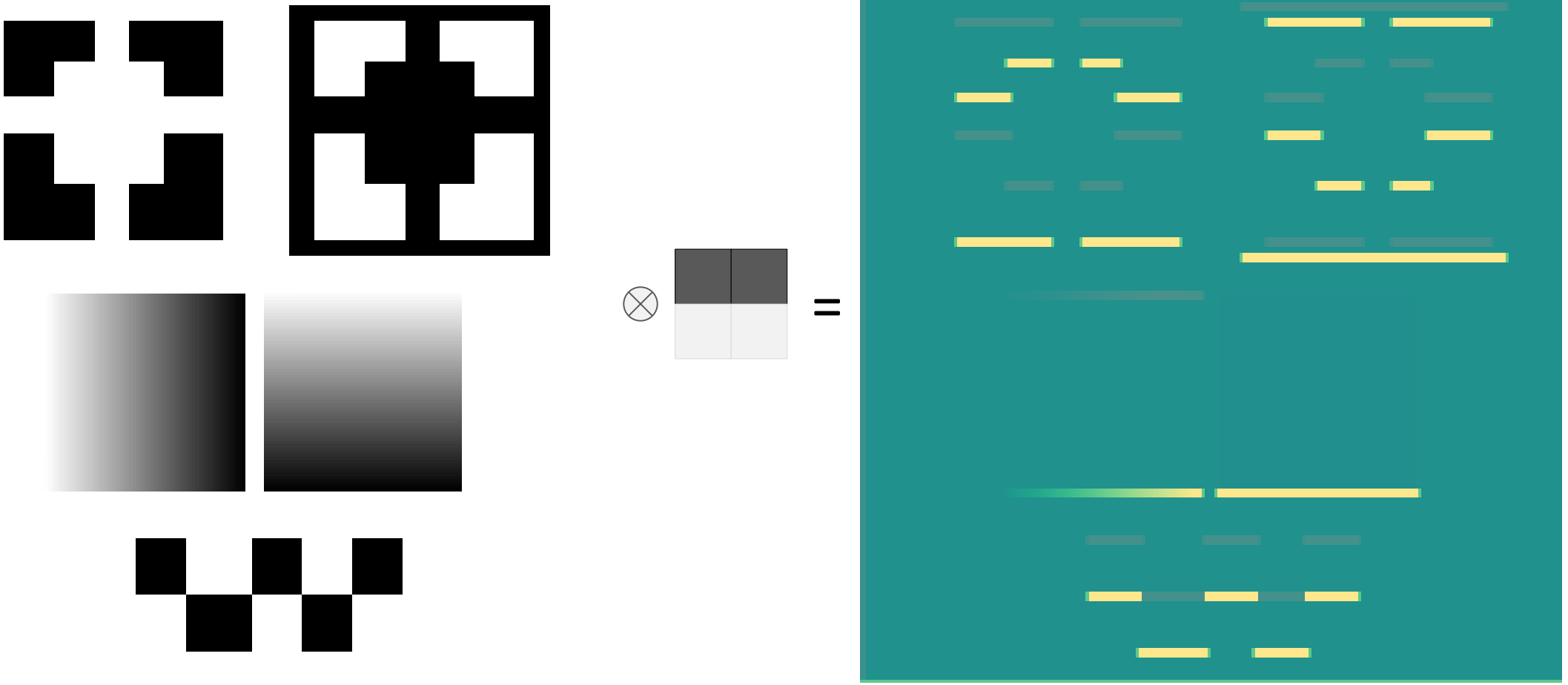
Basic operations

Convolution



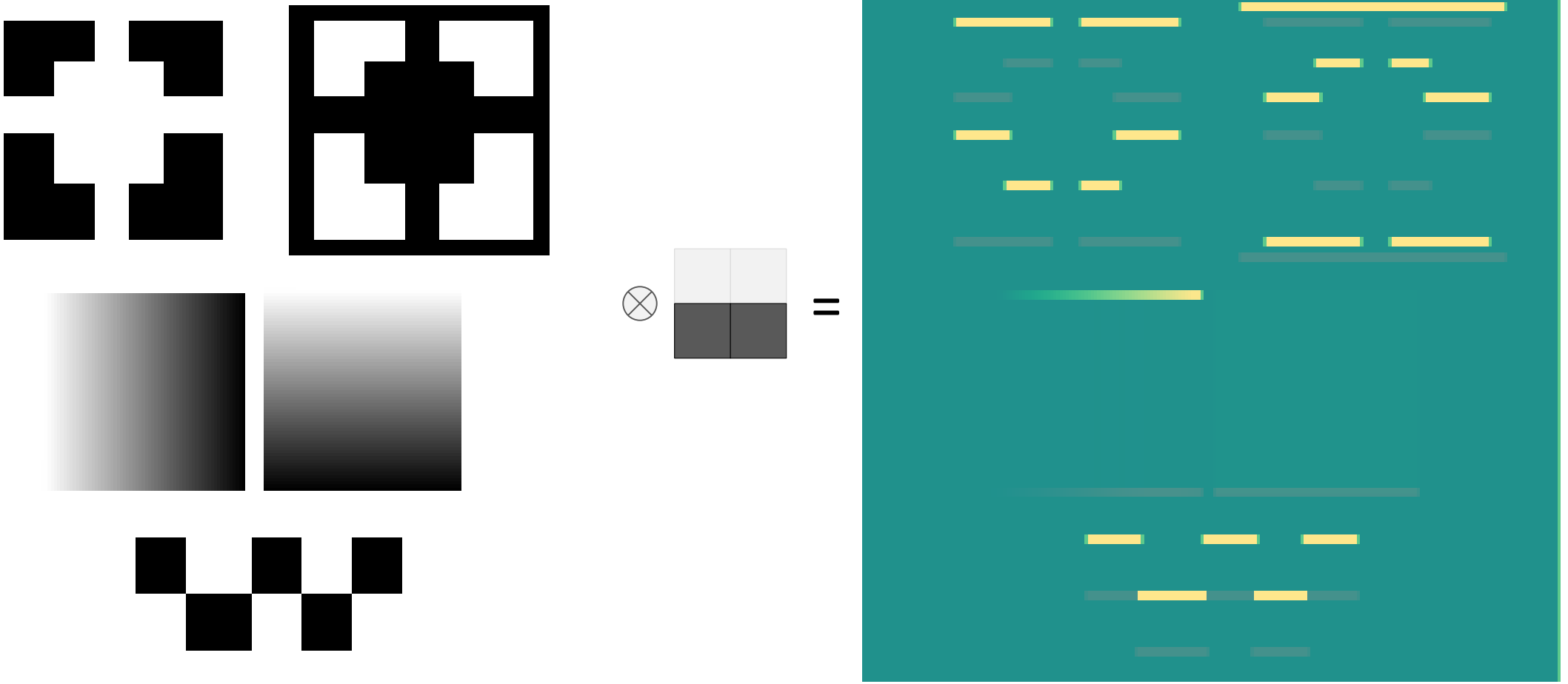
Basic operations

Convolution



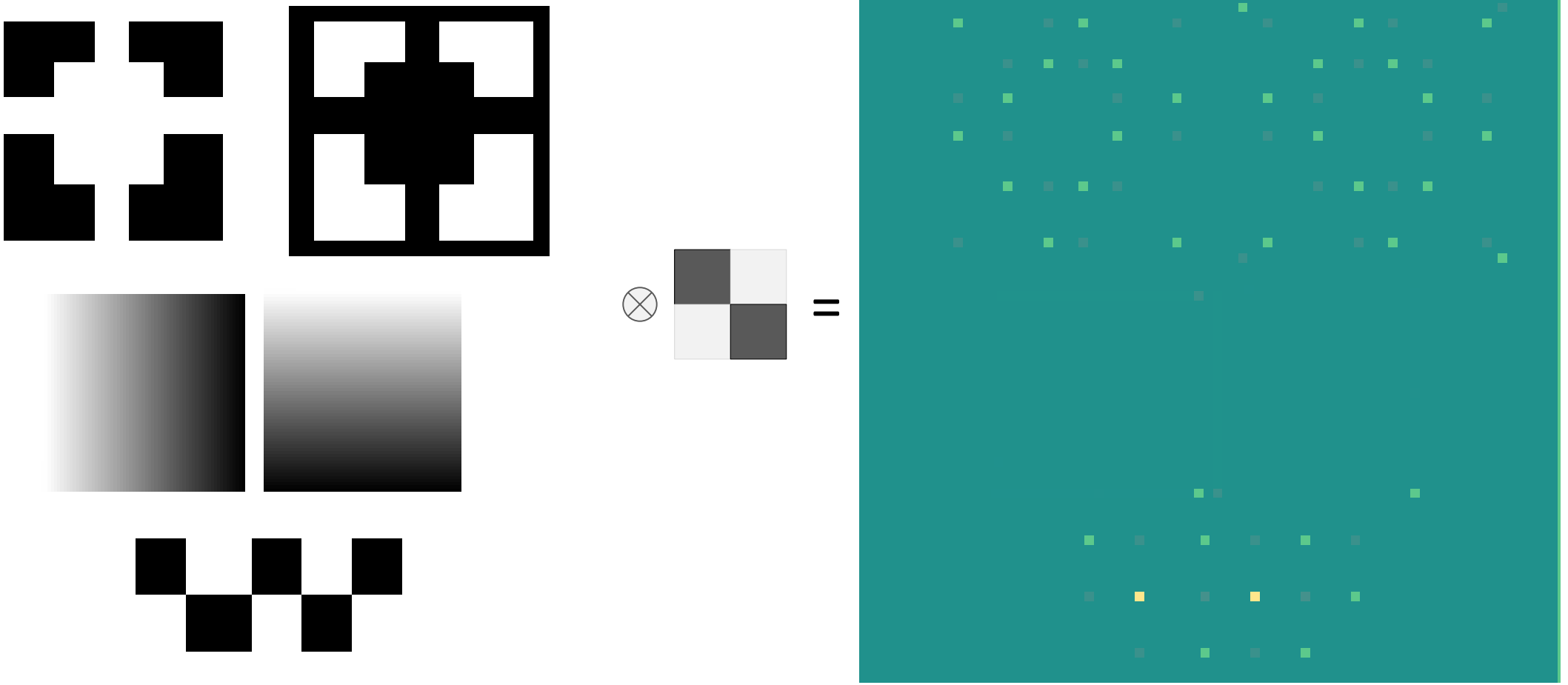
Basic operations

Convolution



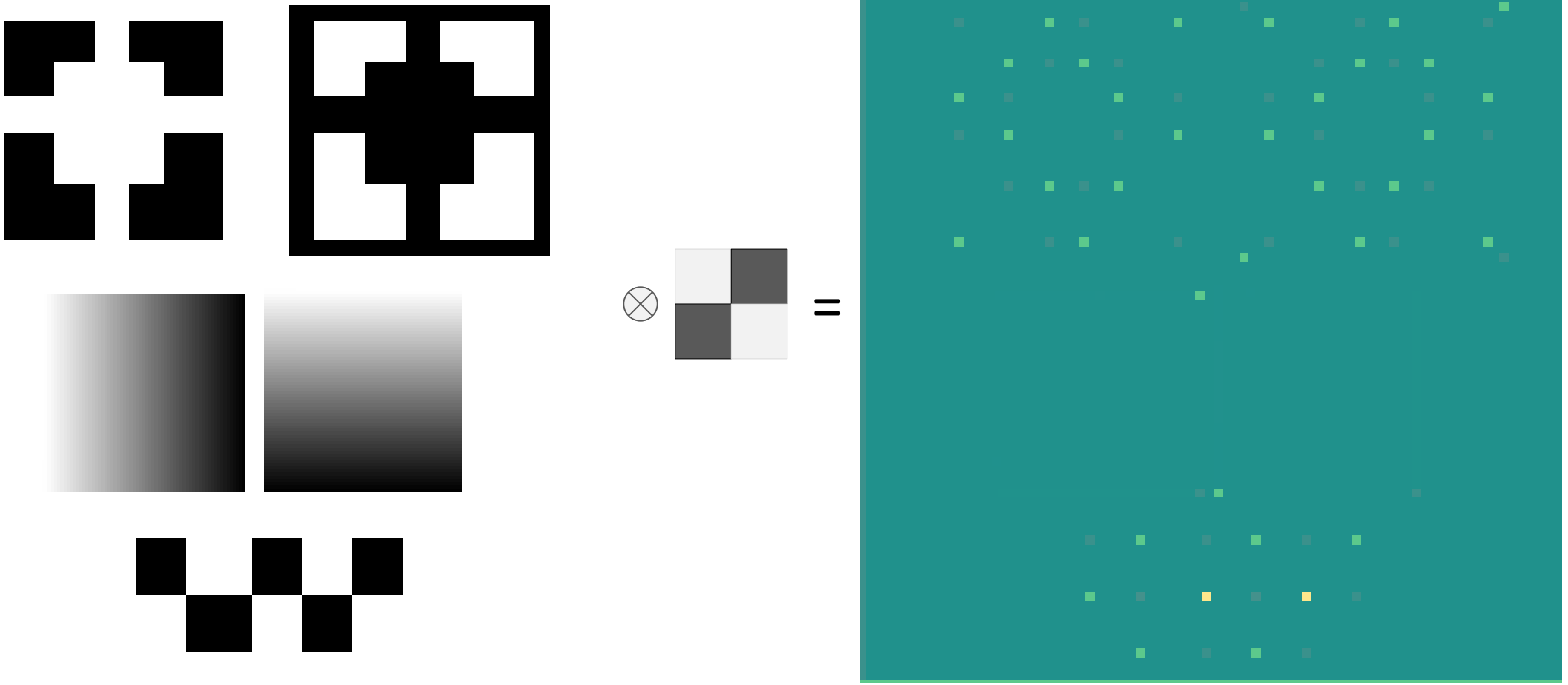
Basic operations

Convolution



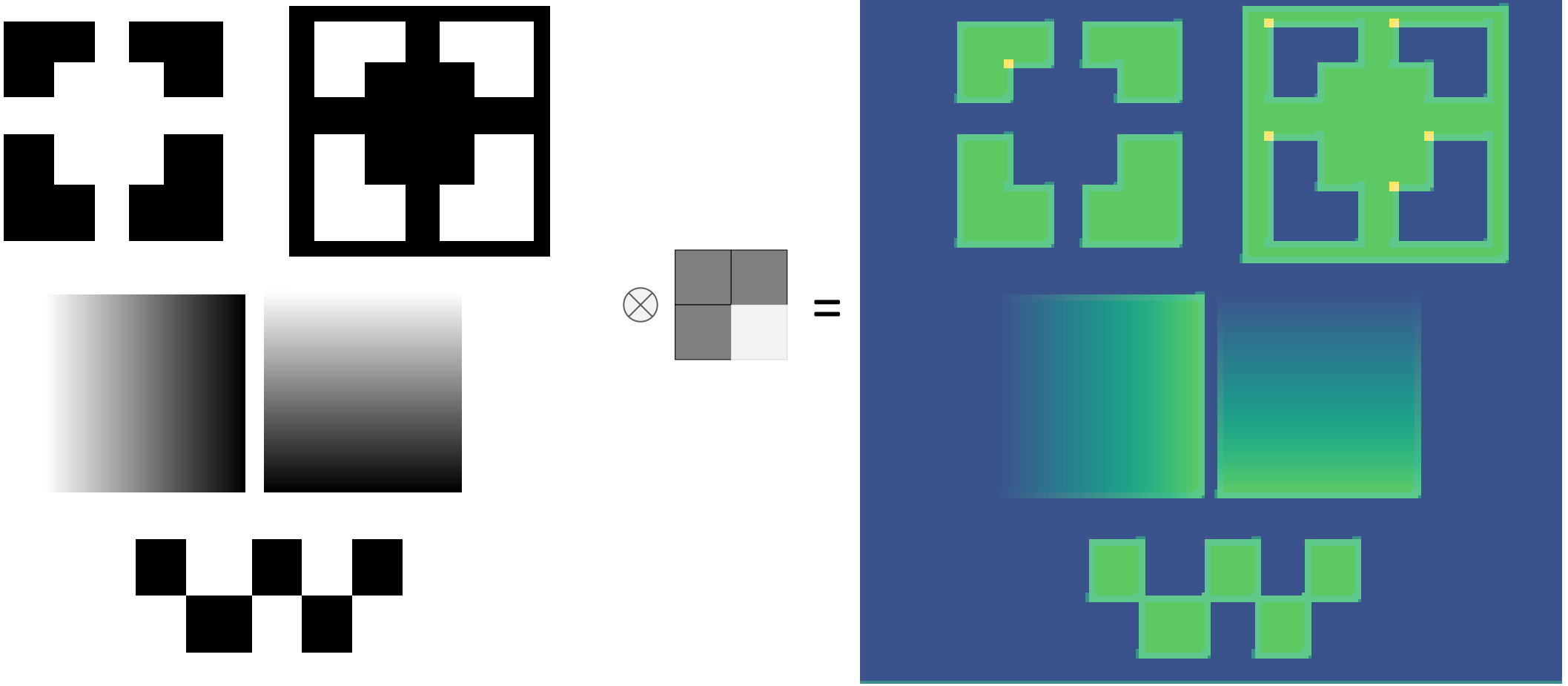
Basic operations

Convolution



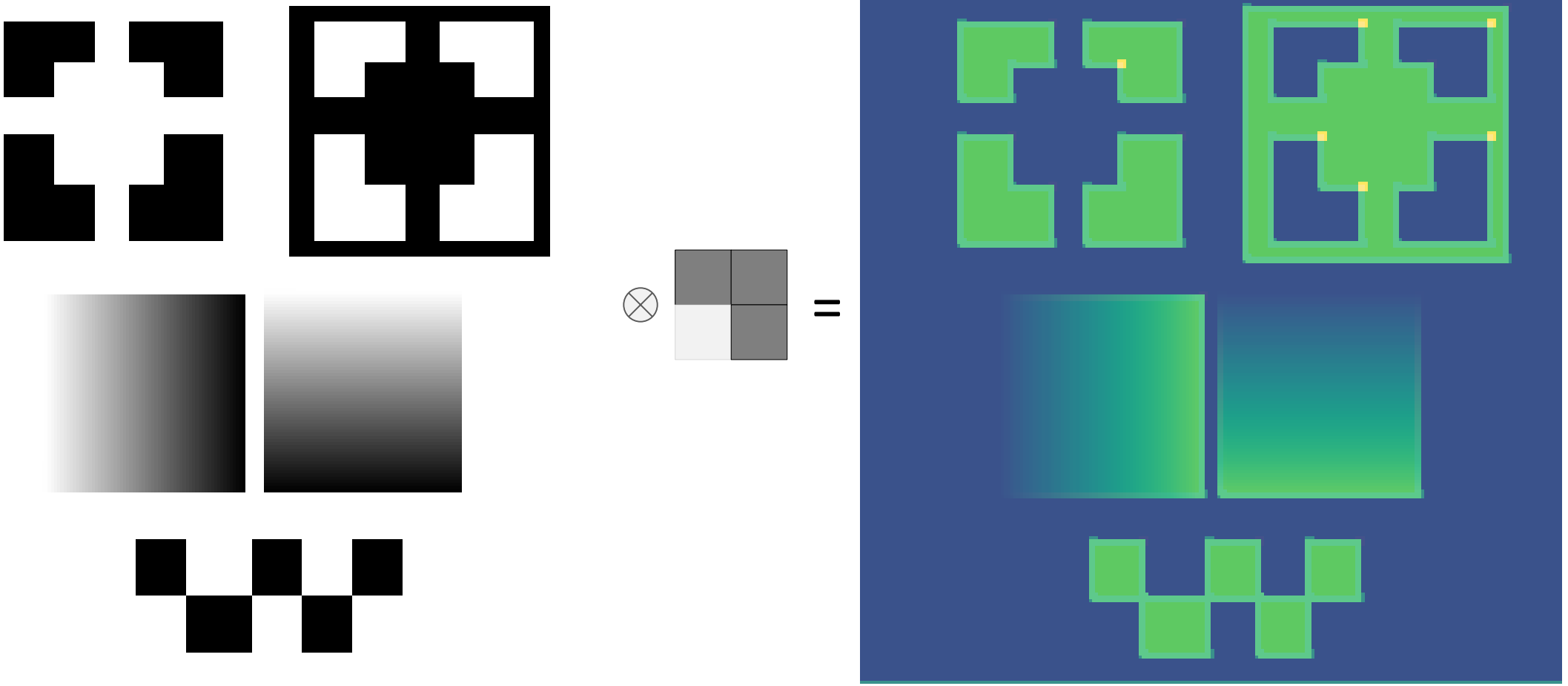
Basic operations

Convolution



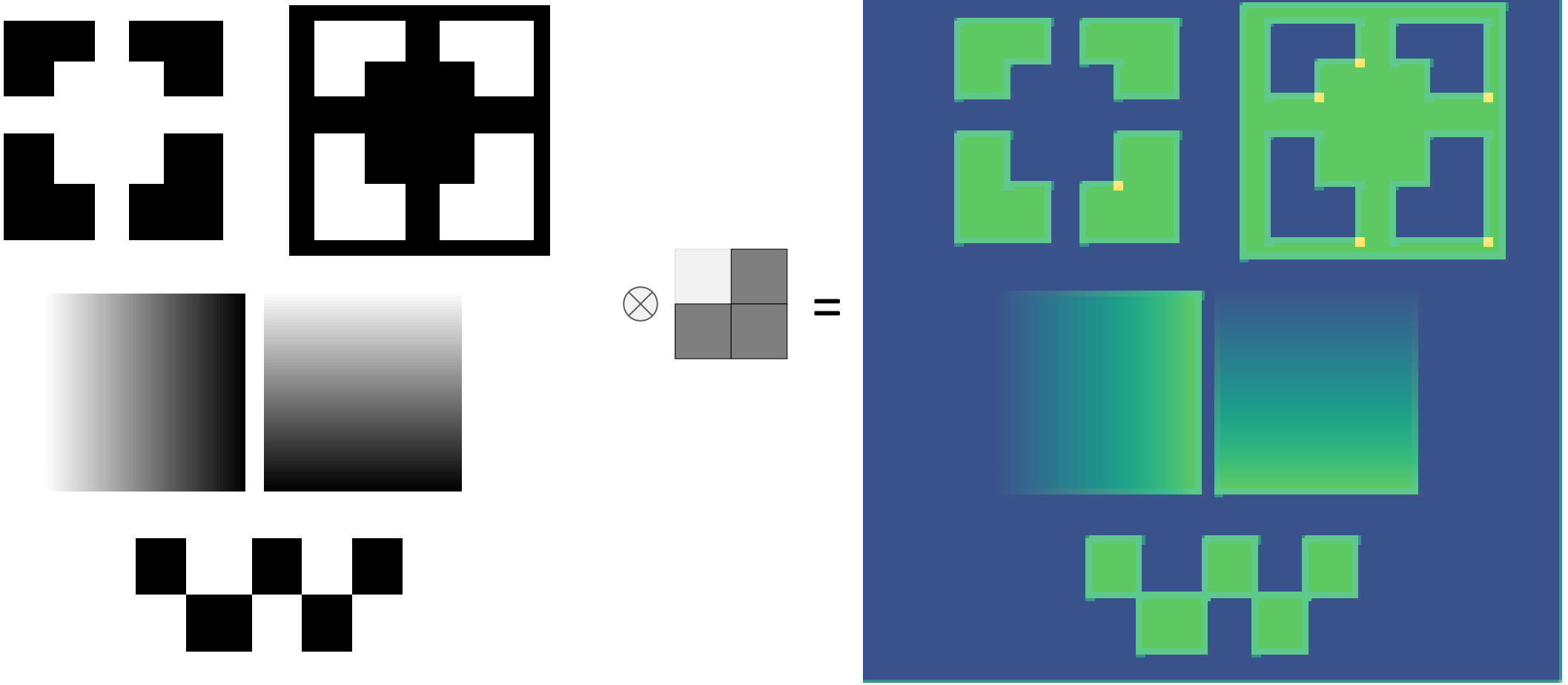
Basic operations

Convolution



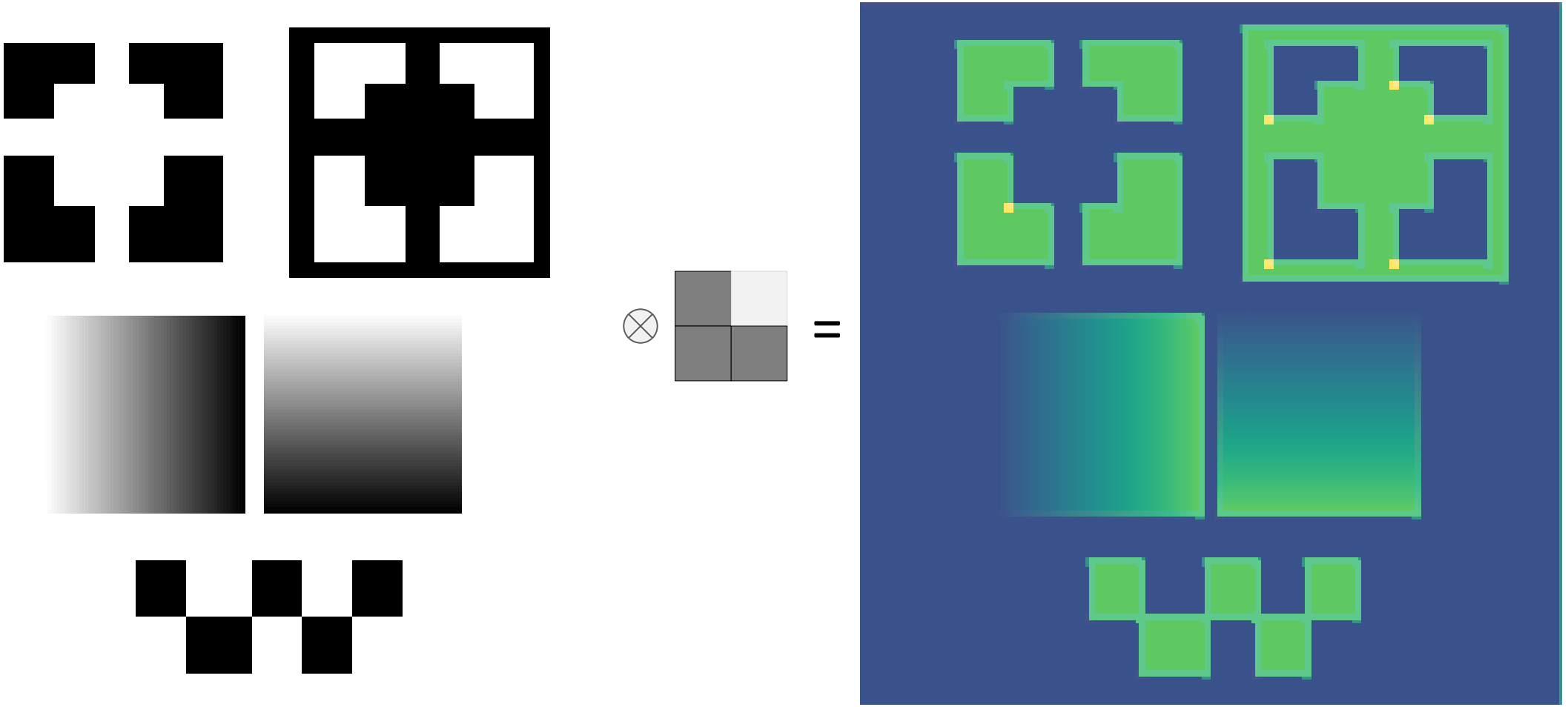
Basic operations

Convolution



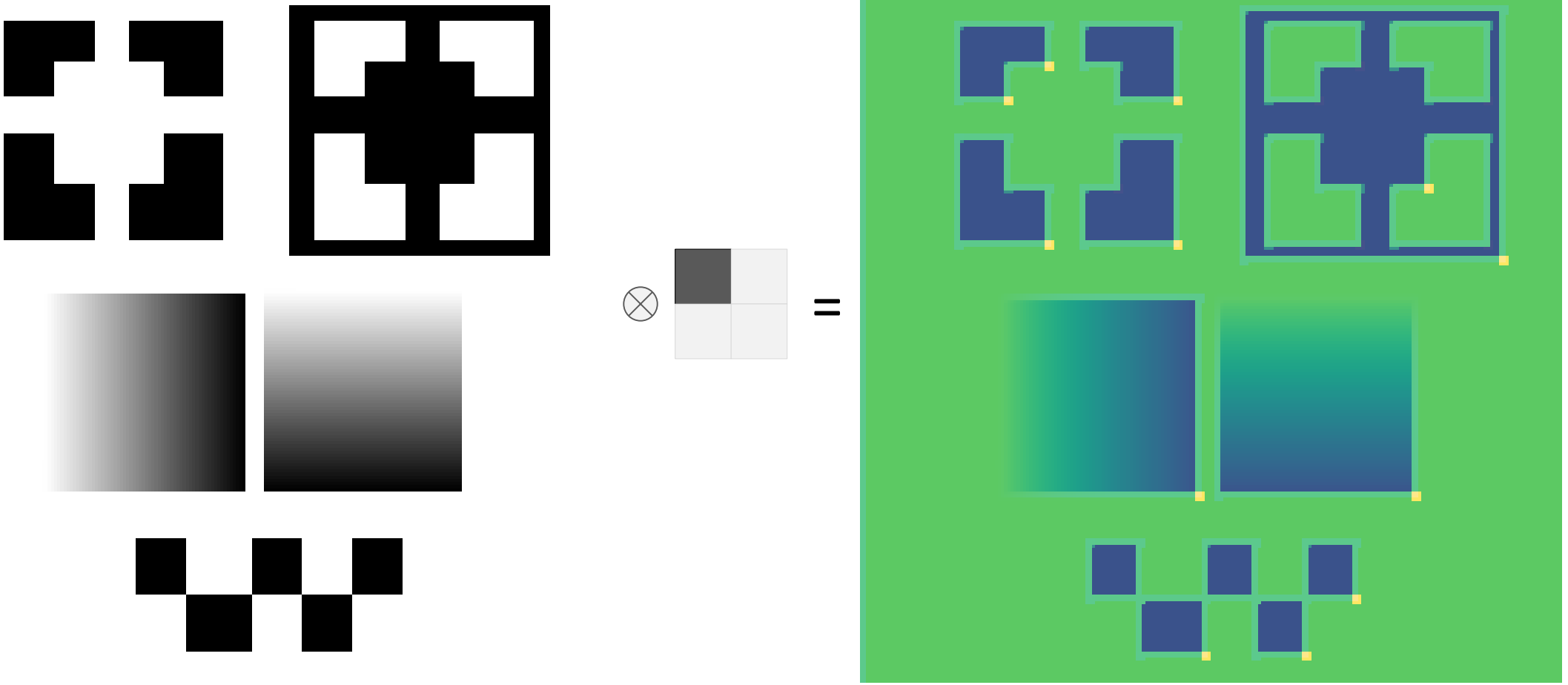
Basic operations

Convolution



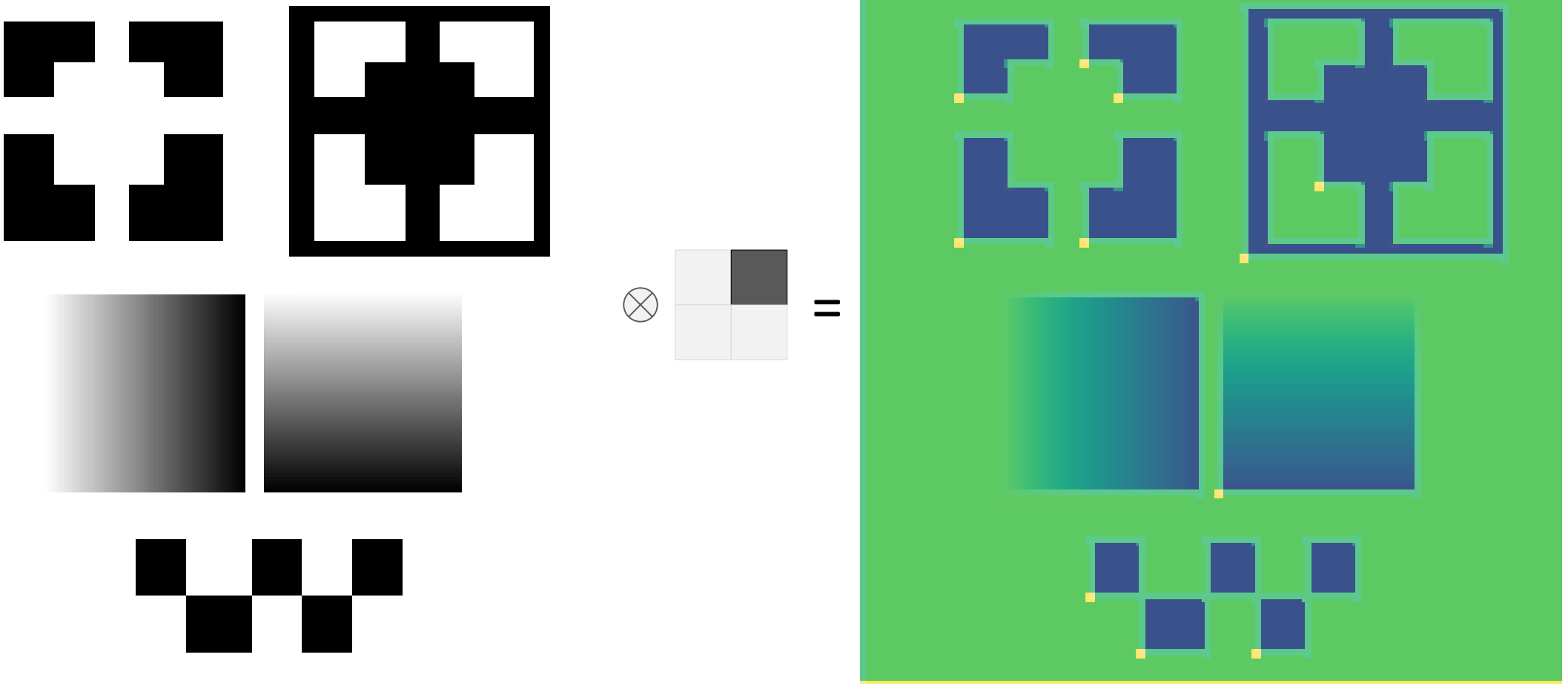
Basic operations

Convolution



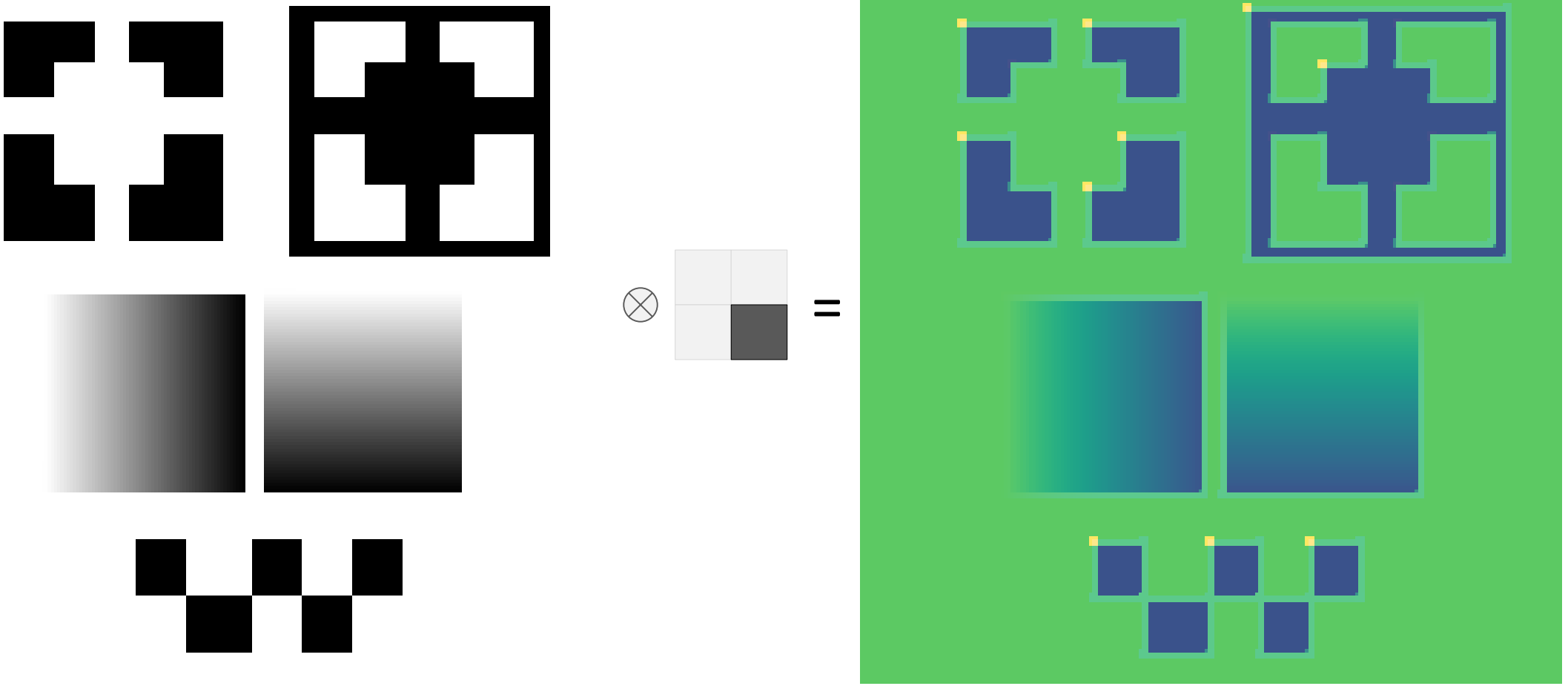
Basic operations

Convolution



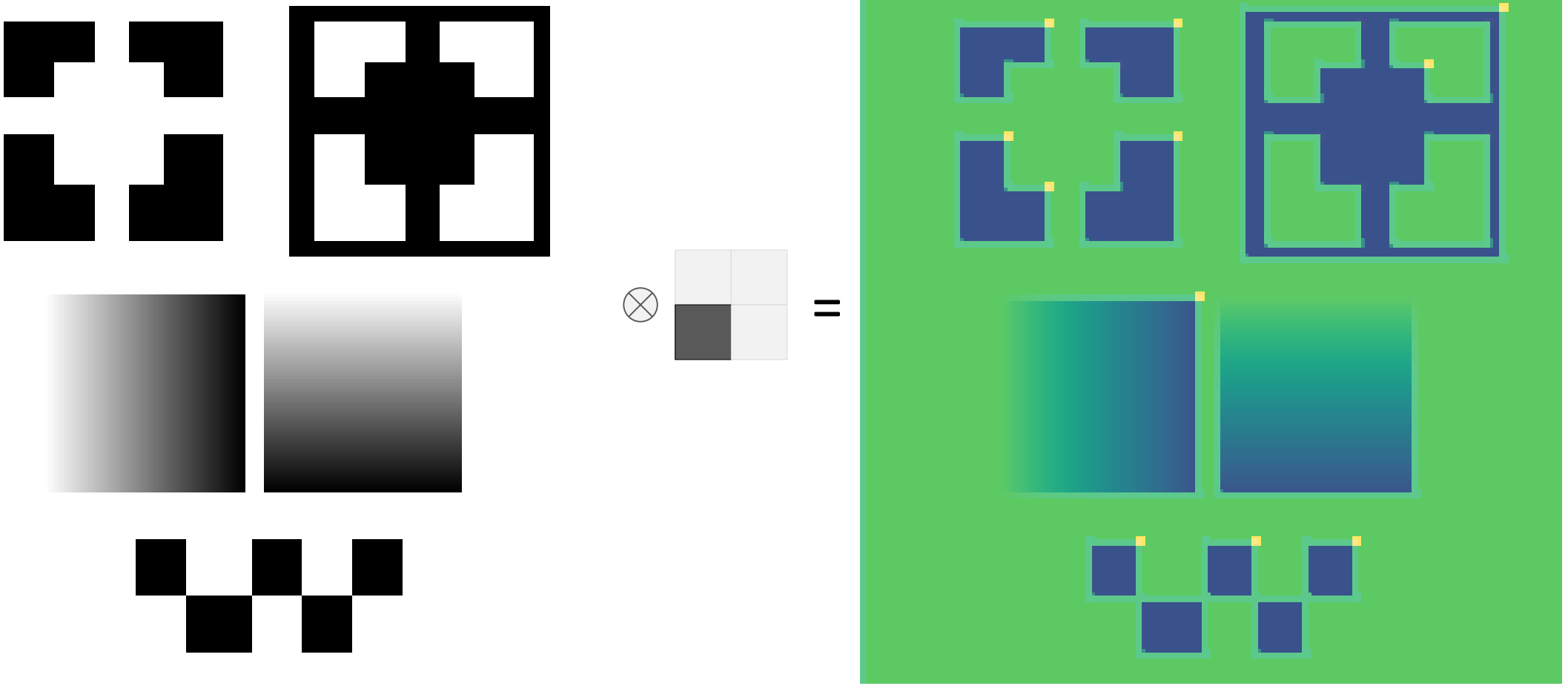
Basic operations

Convolution



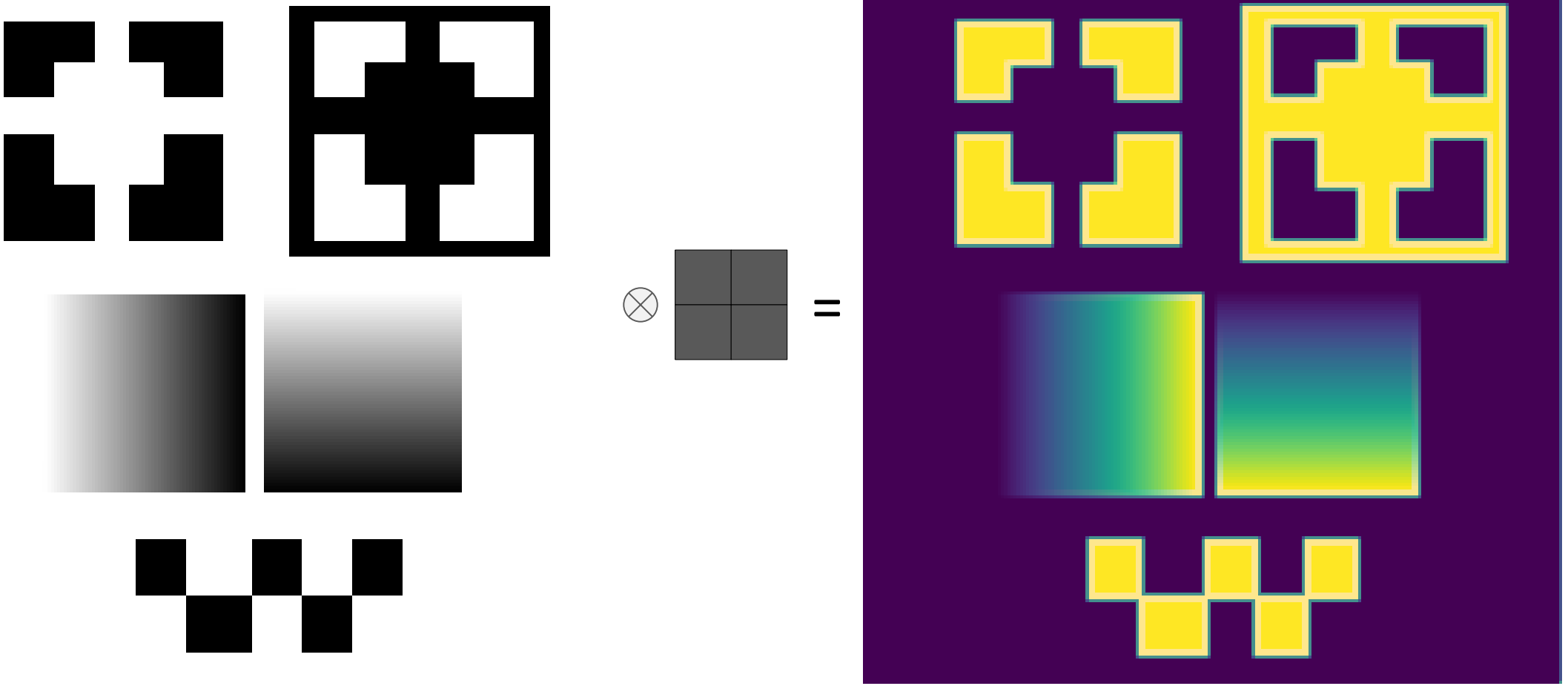
Basic operations

Convolution



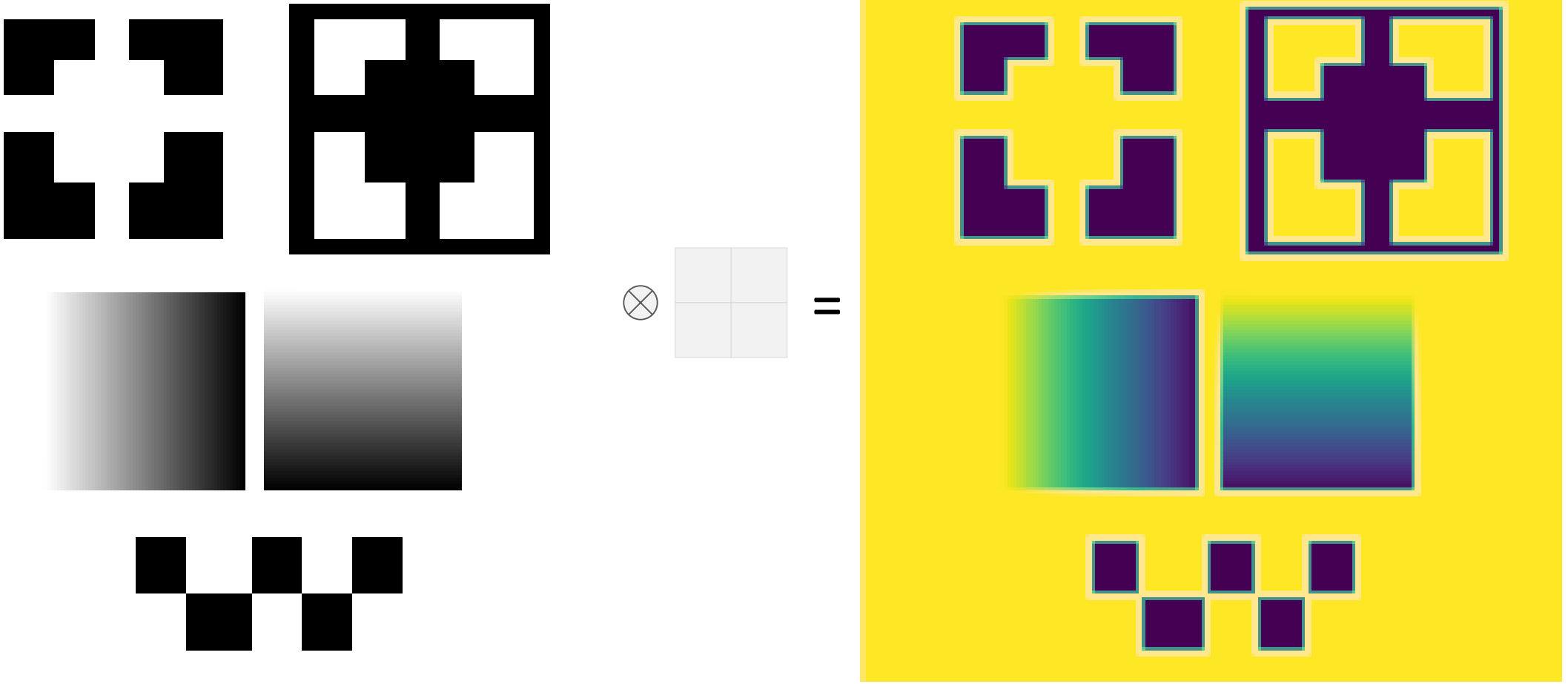
Basic operations

Convolution



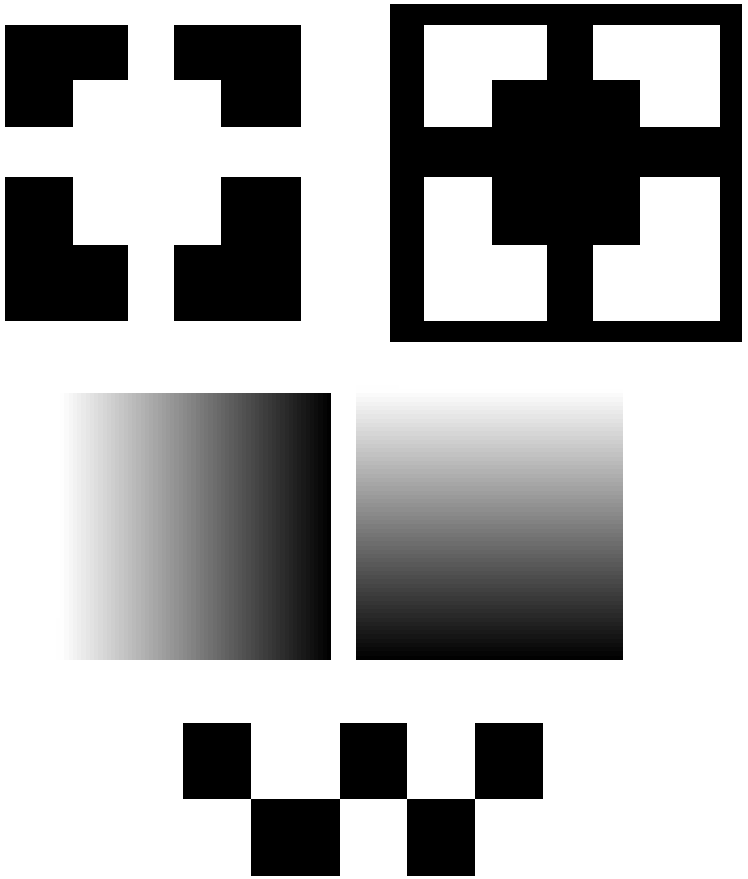
Basic operations

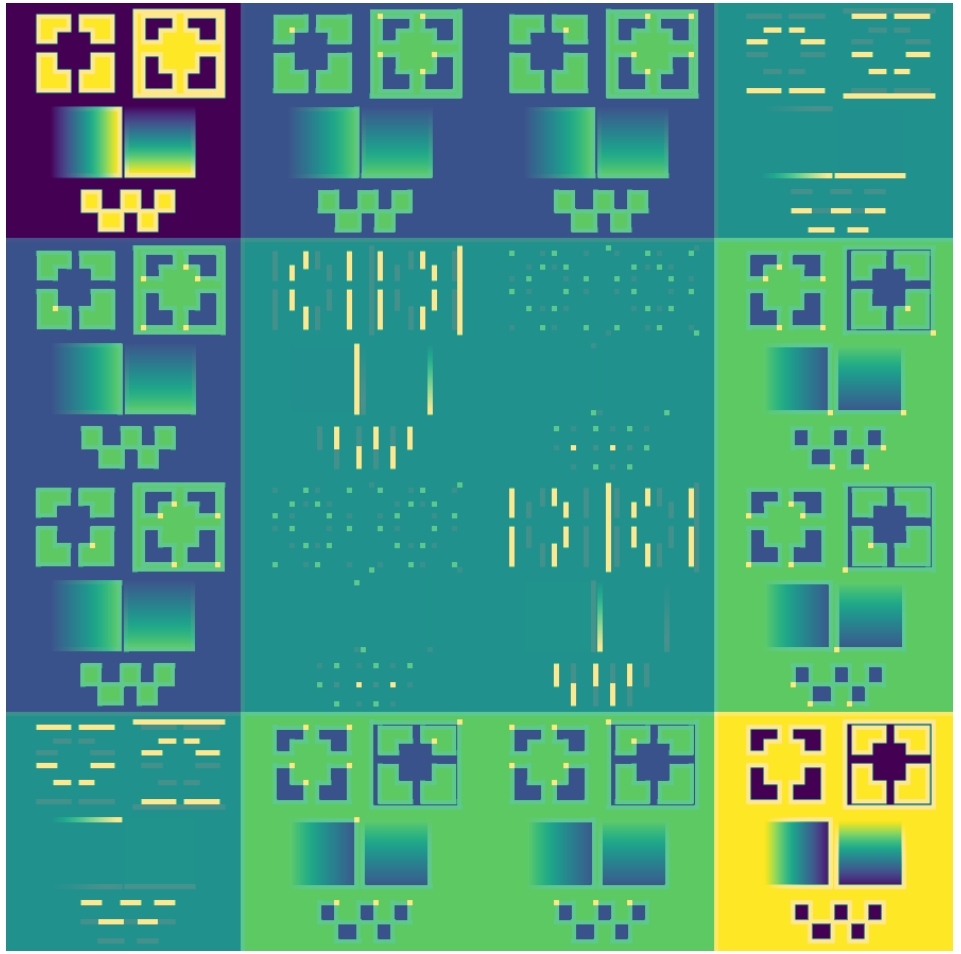
Convolution



Basic operations

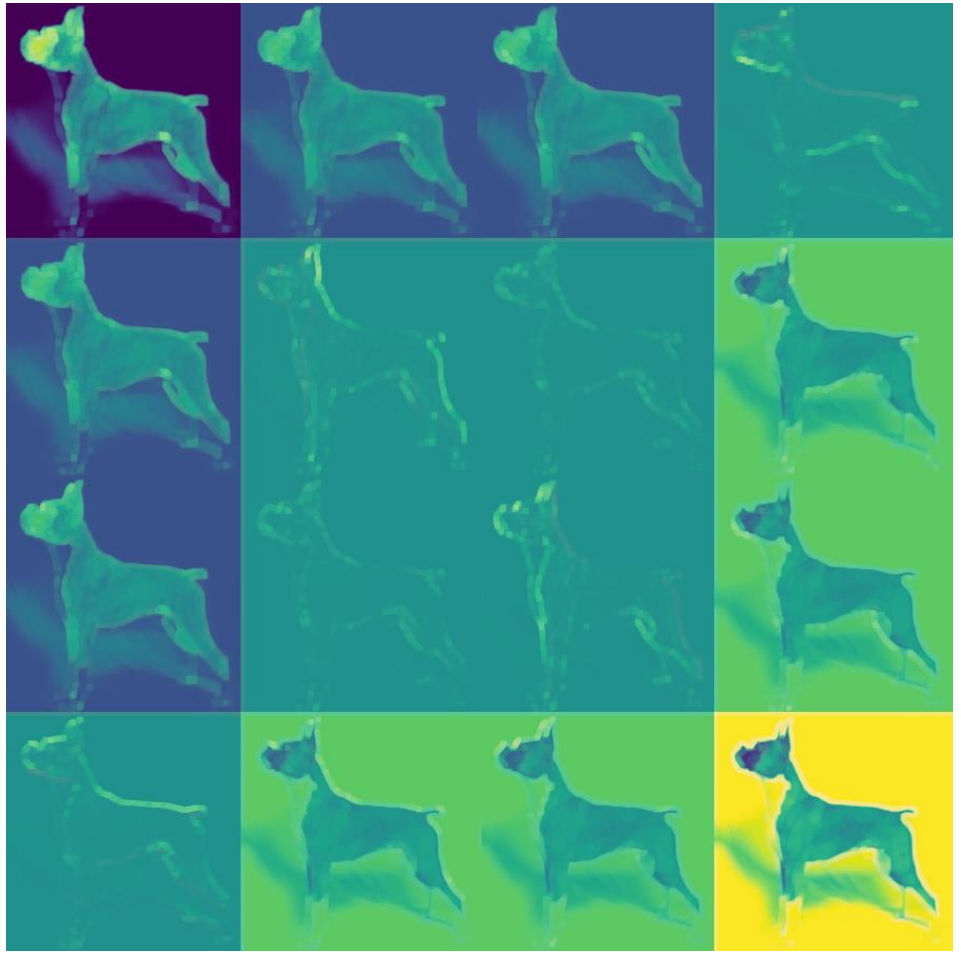
Convolution





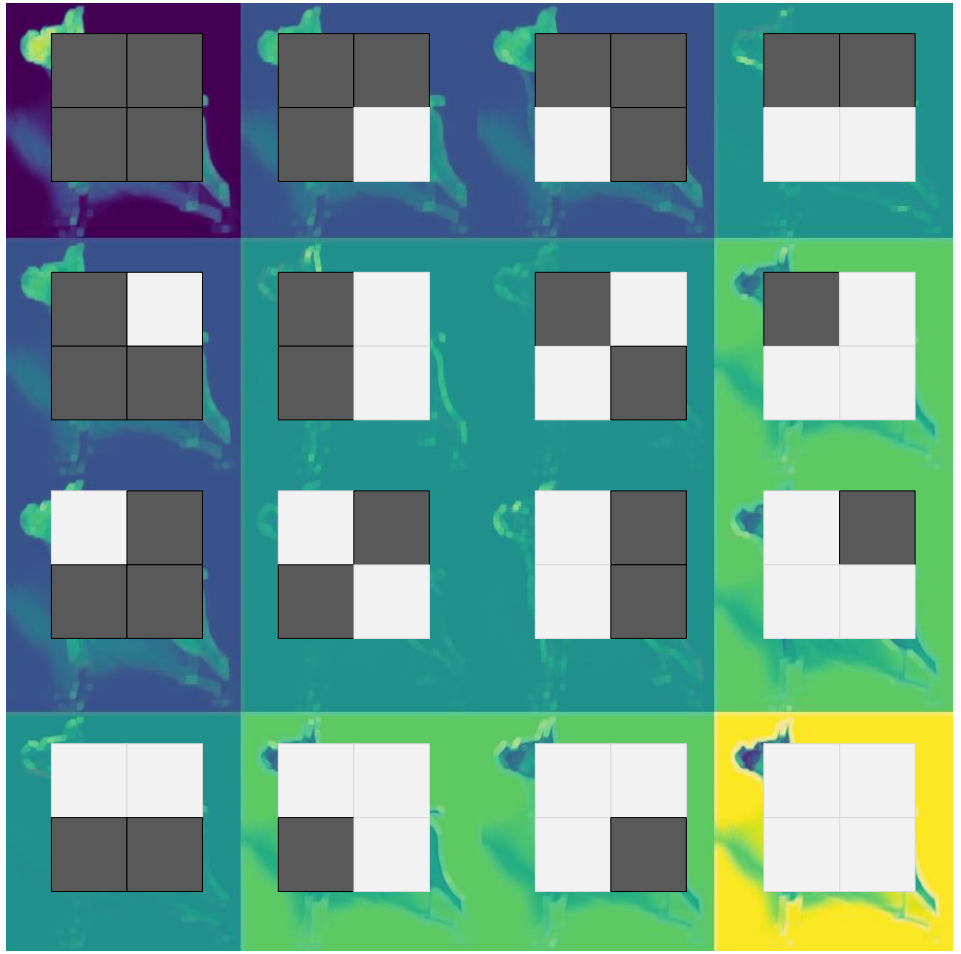
Basic operations

Convolution



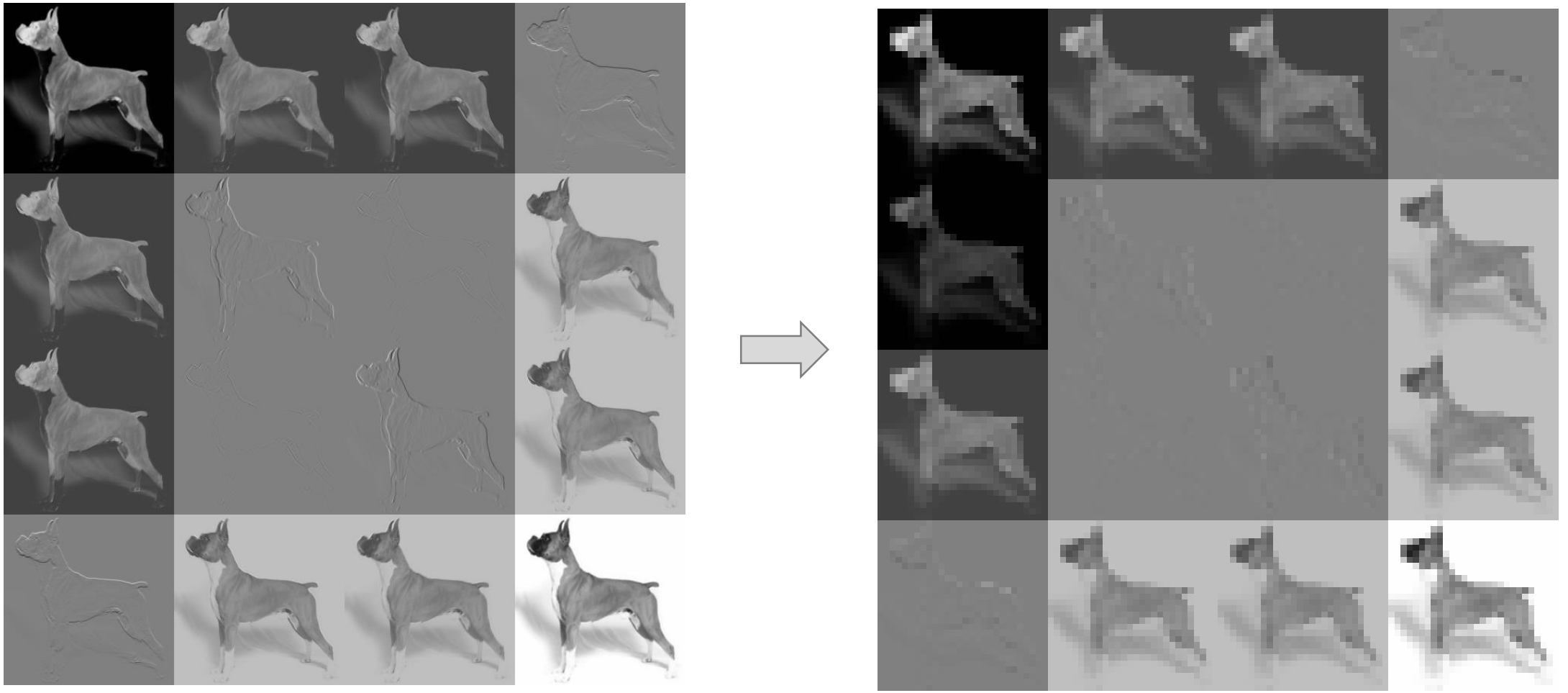
Basic operations

De-Convolution



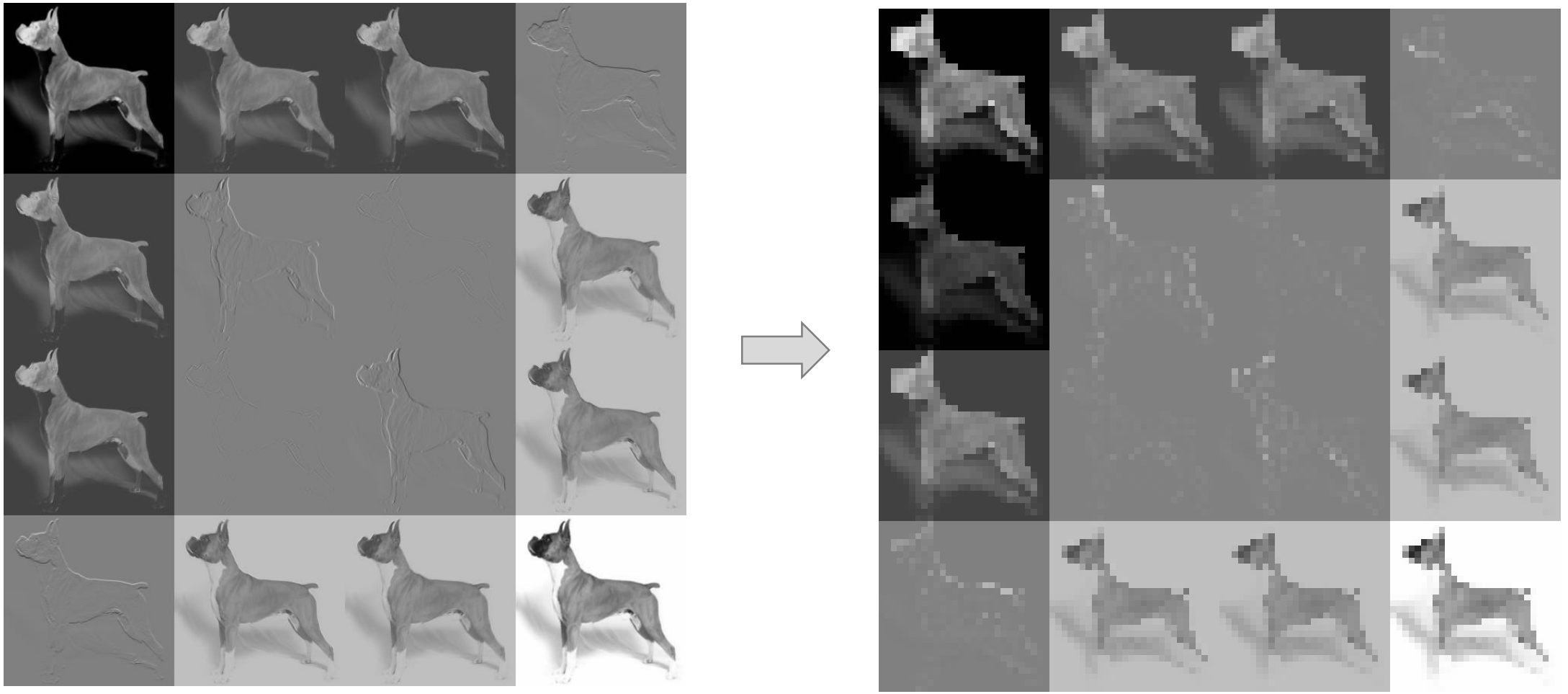
Basic operations

Pool: avg



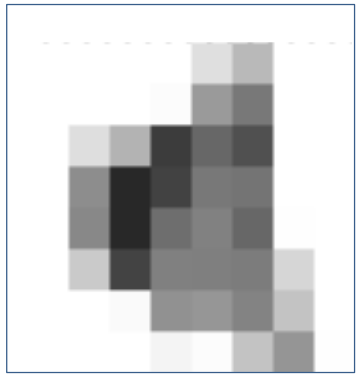
Basic operations

Pool: max



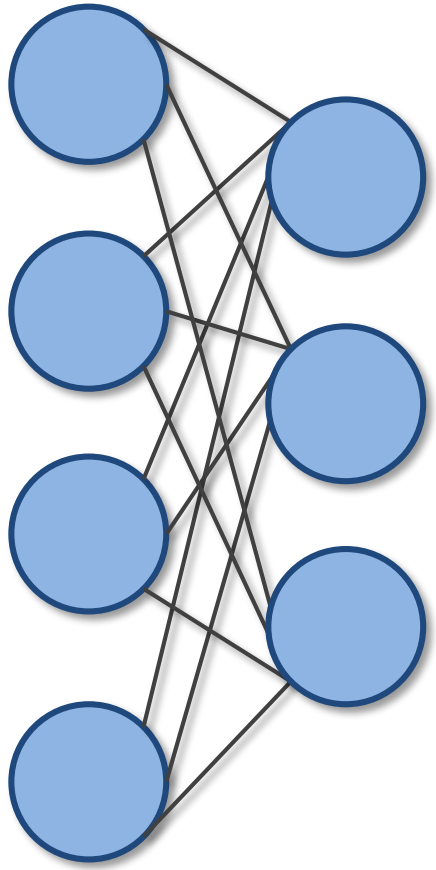
Basic operations

Flatten



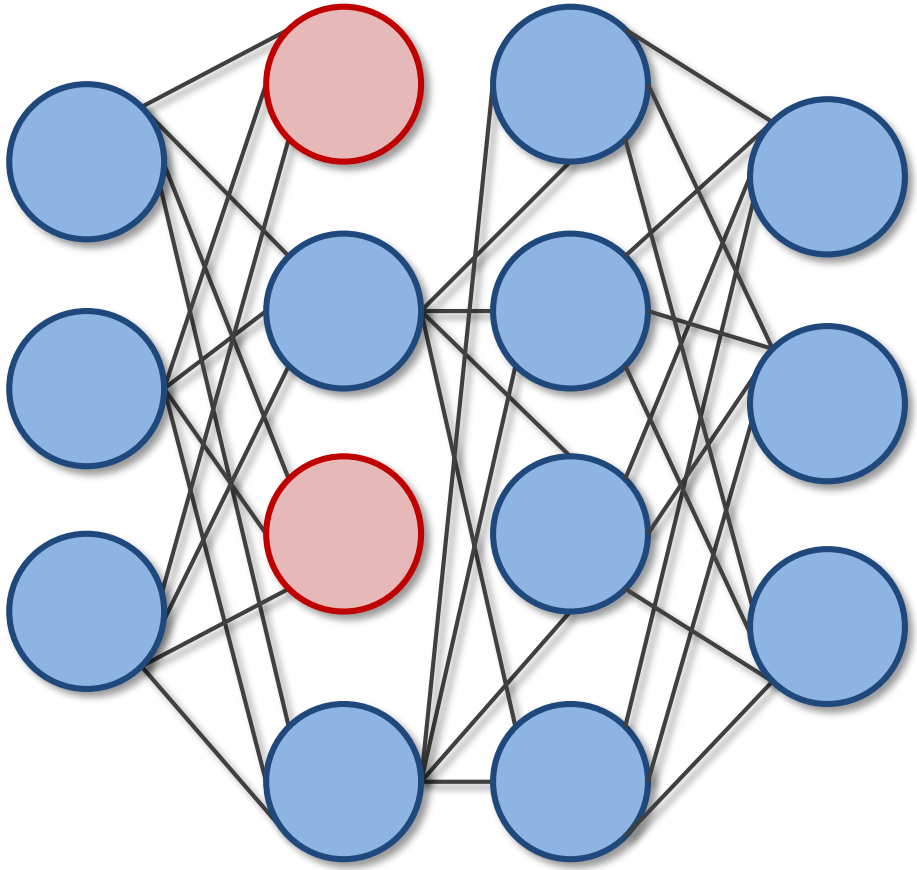
Basic operations

Fully Connected



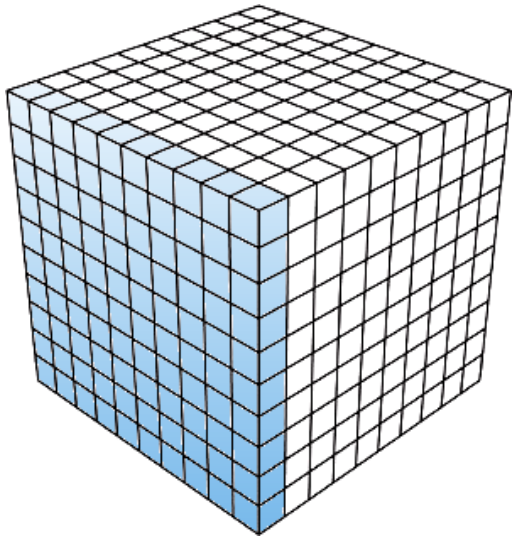
Basic operations

Dropout

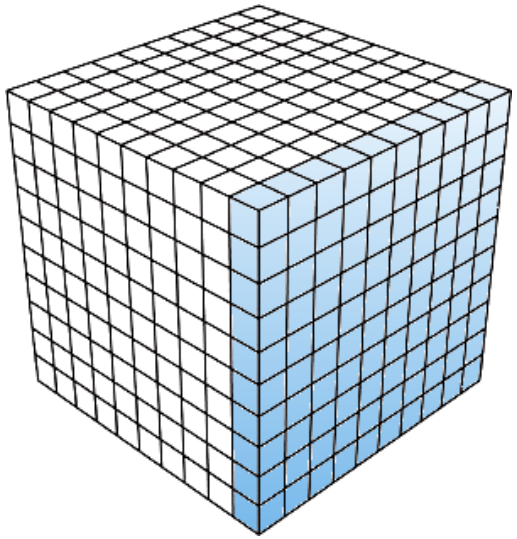


Basic operations

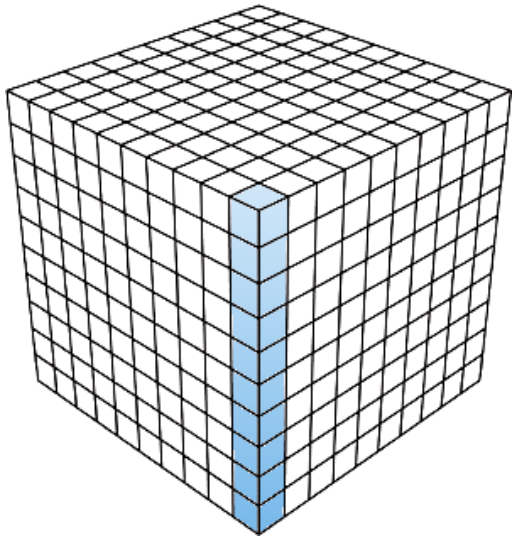
Normalization



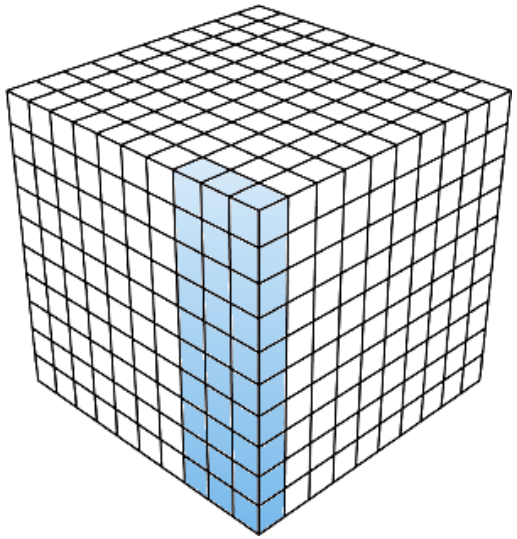
layer



batch



instance



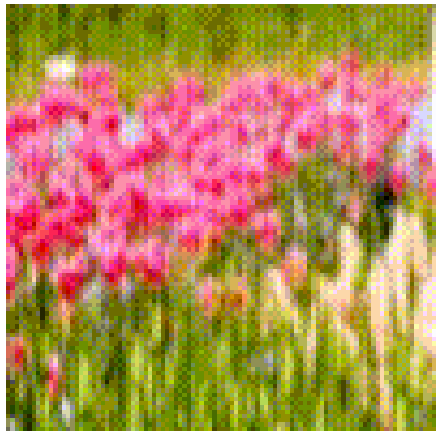
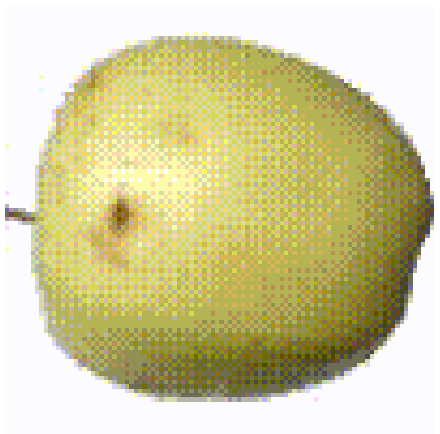
group

Feature extraction

Feature extraction

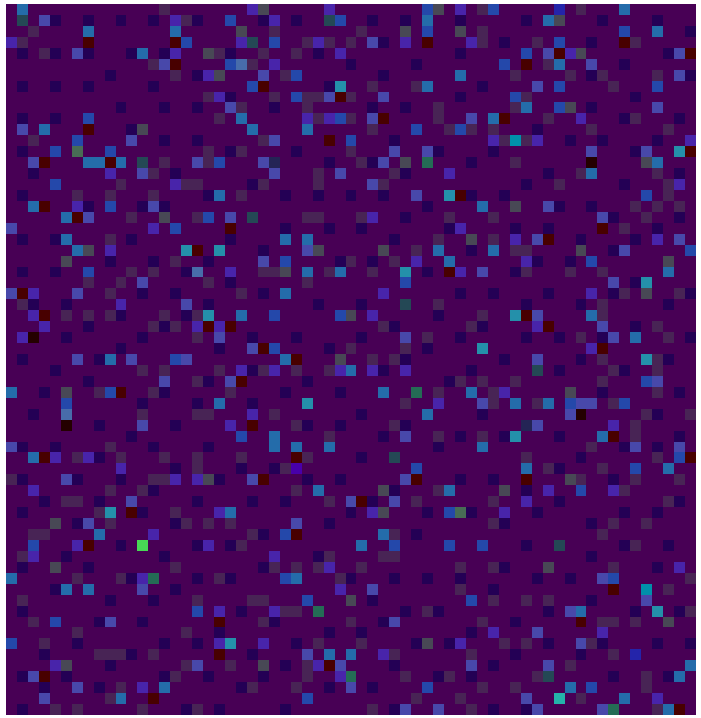




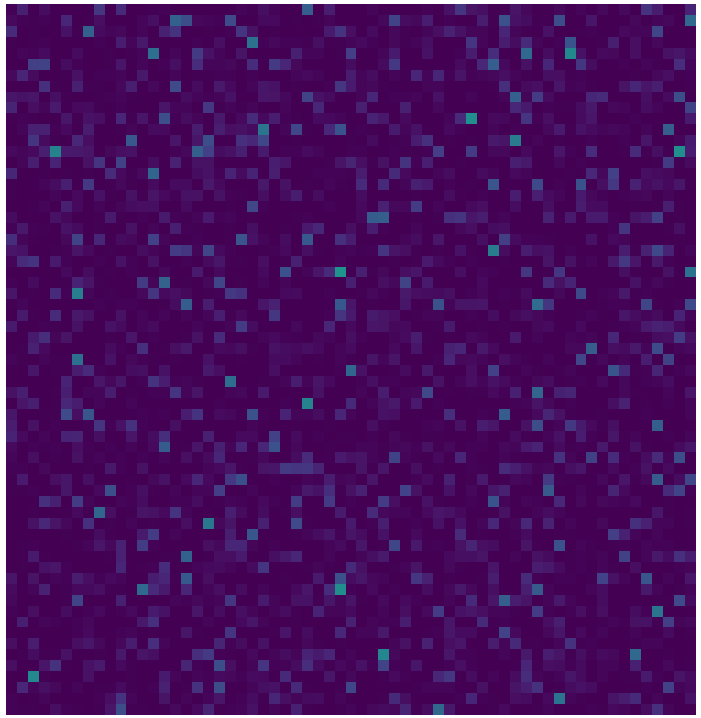



Feature extraction

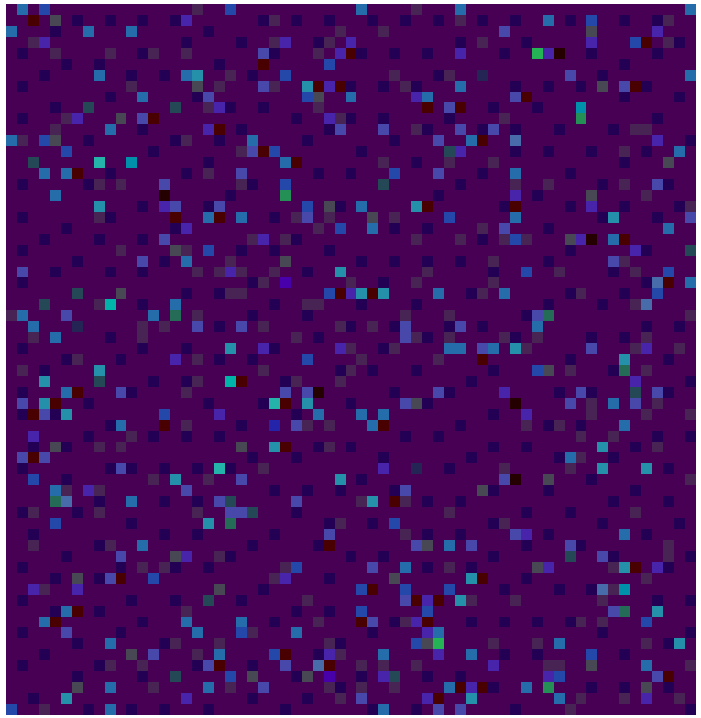
Extract the features



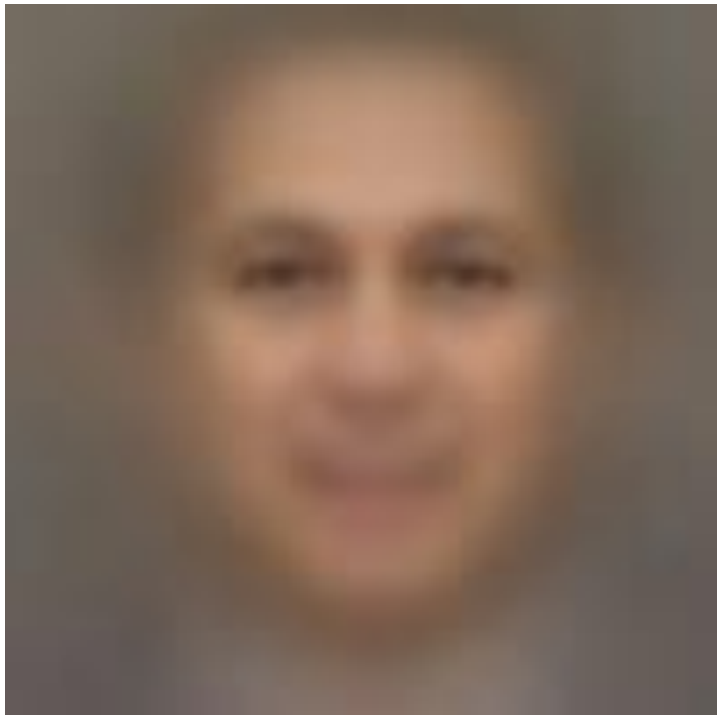
Feature extraction

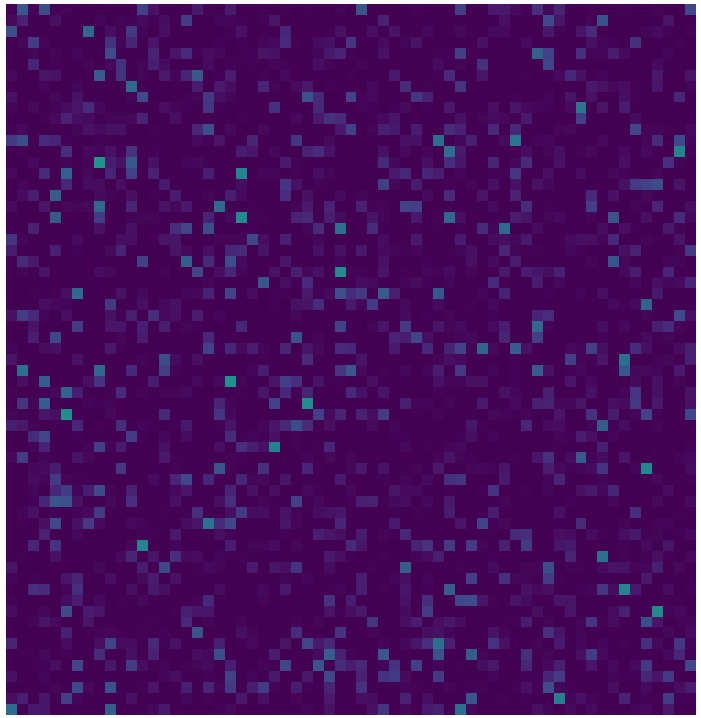


Feature extraction

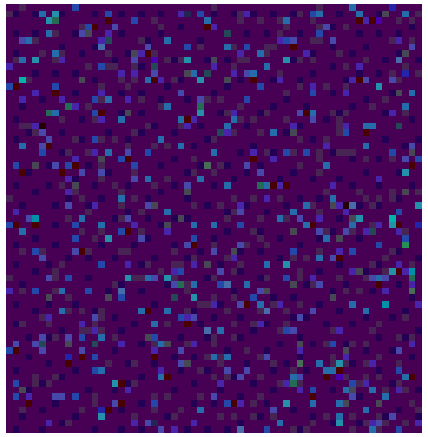


Feature extraction

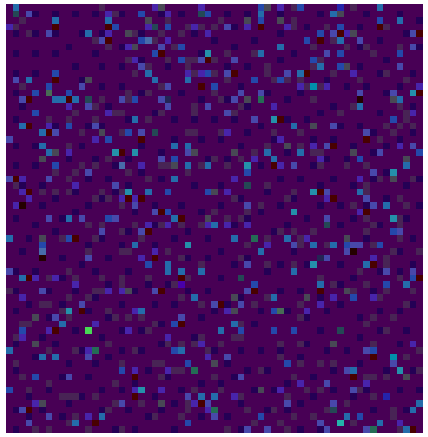




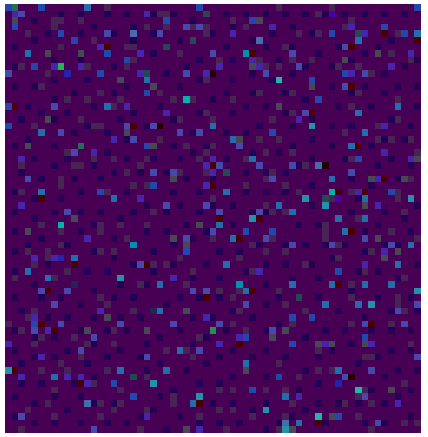
Feature extraction



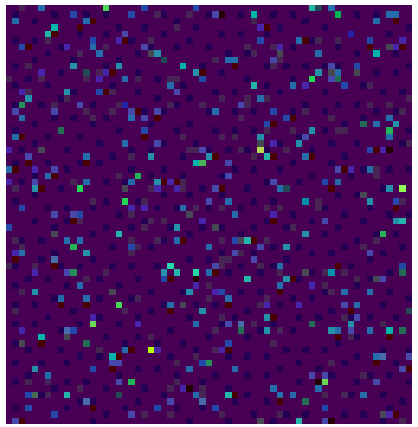
airplane



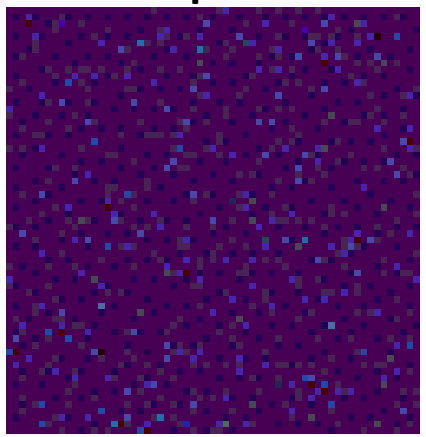
car



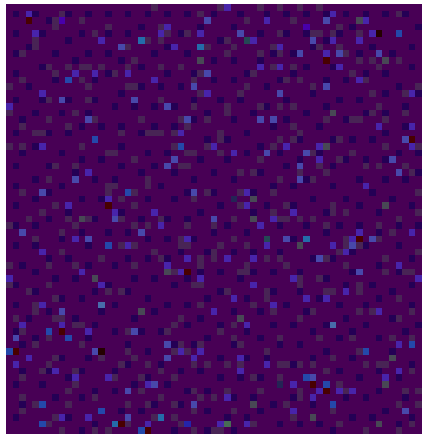
cat



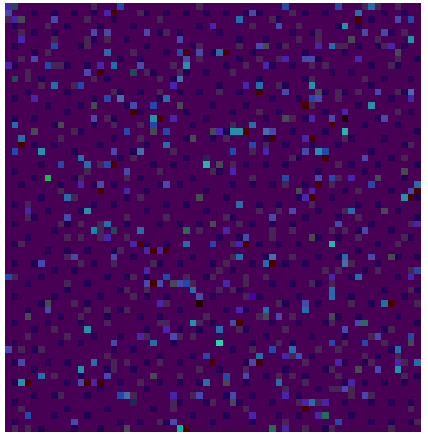
dog



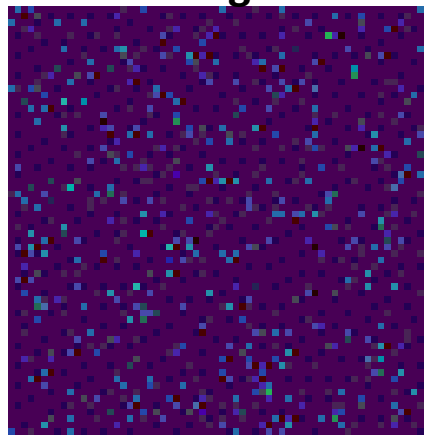
flower



fruit

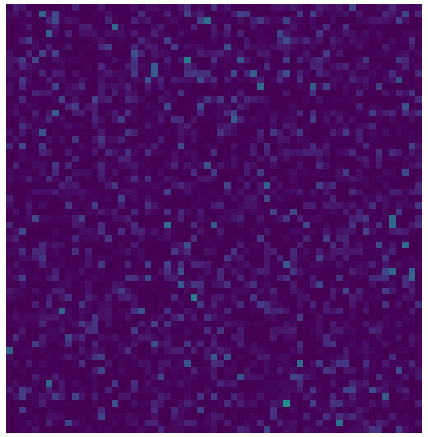


motorbike

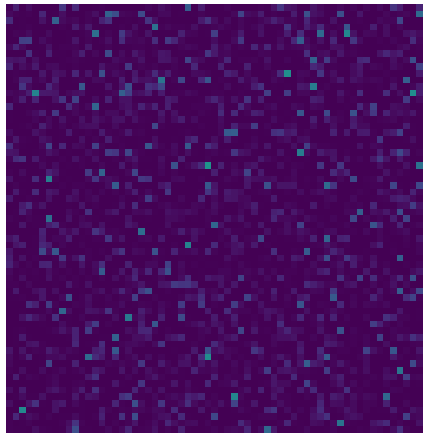


person

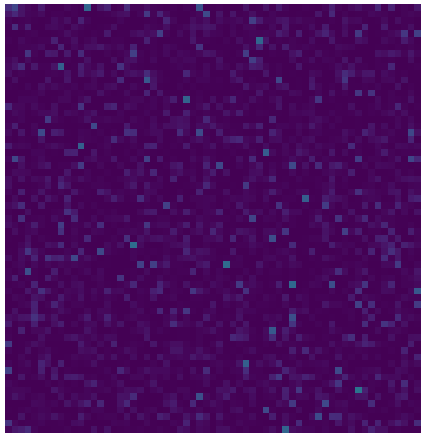
Feature extraction



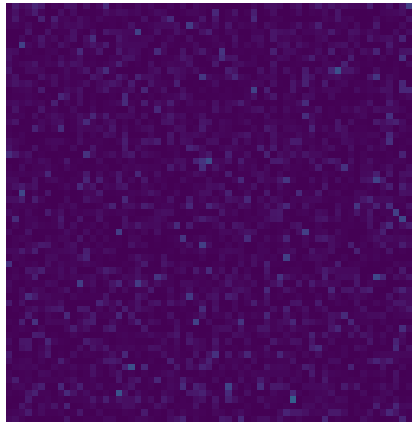
airplane



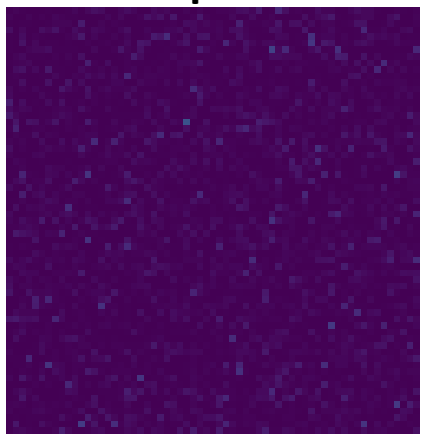
car



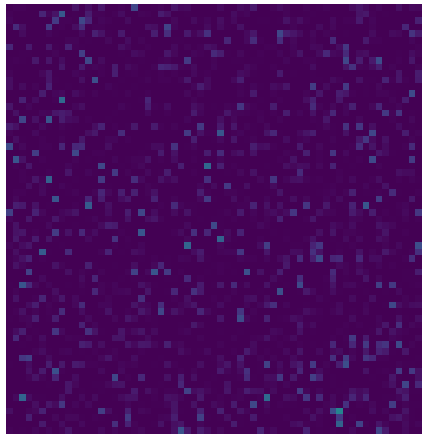
cat



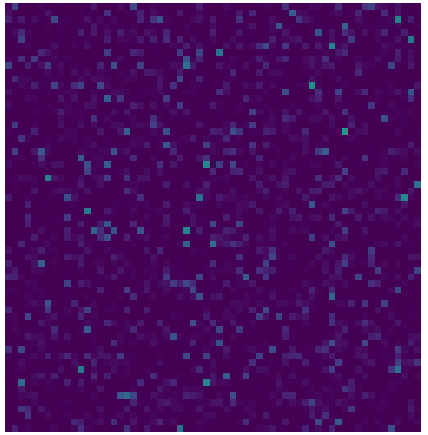
dog



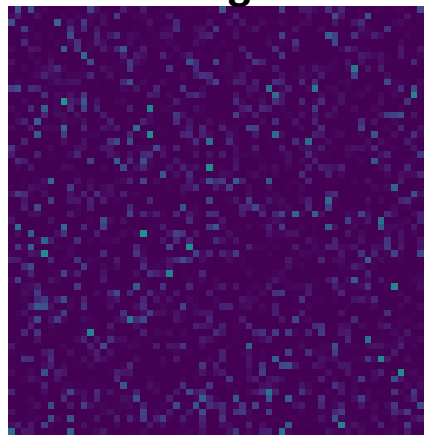
flower



fruit



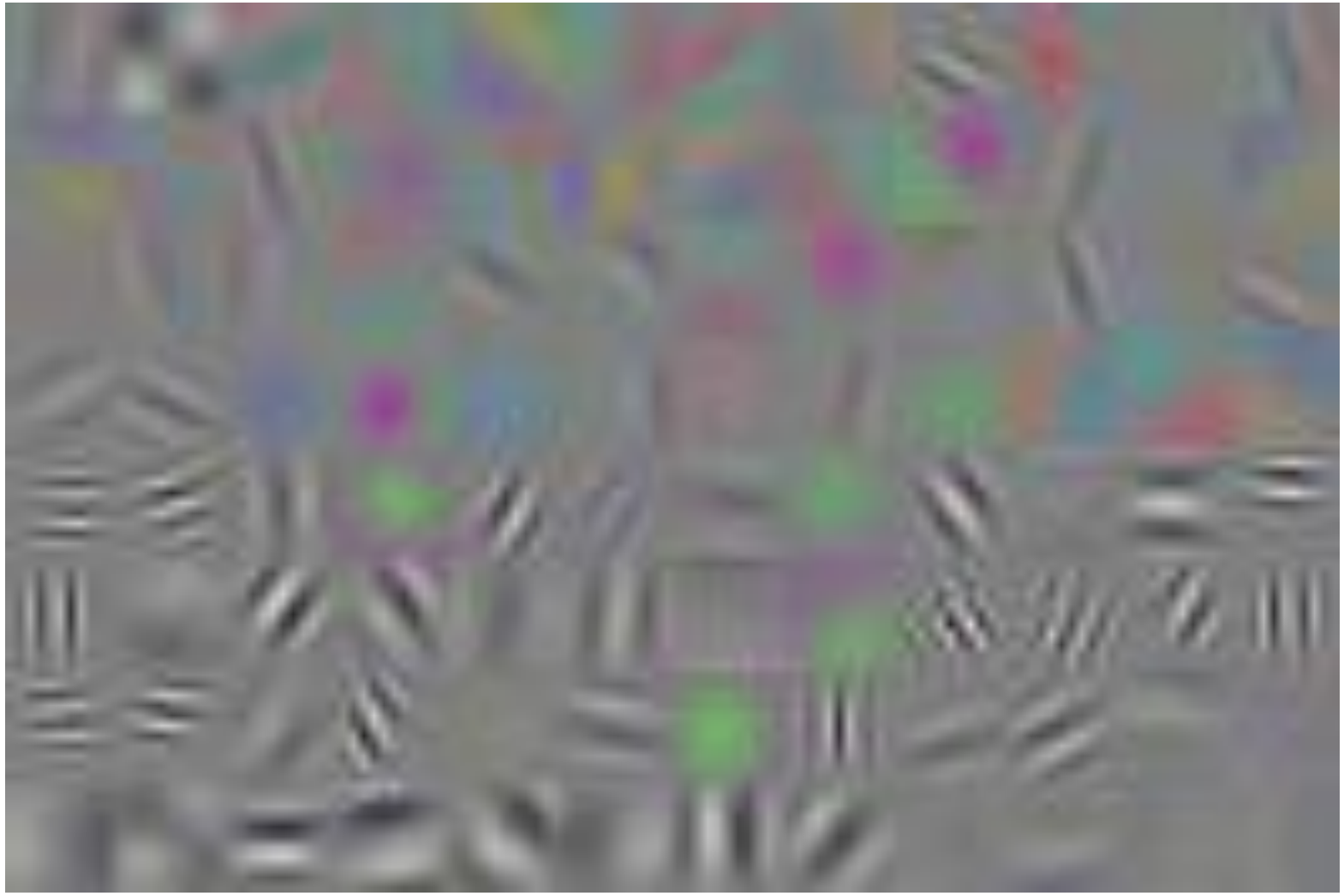
motorbike



person

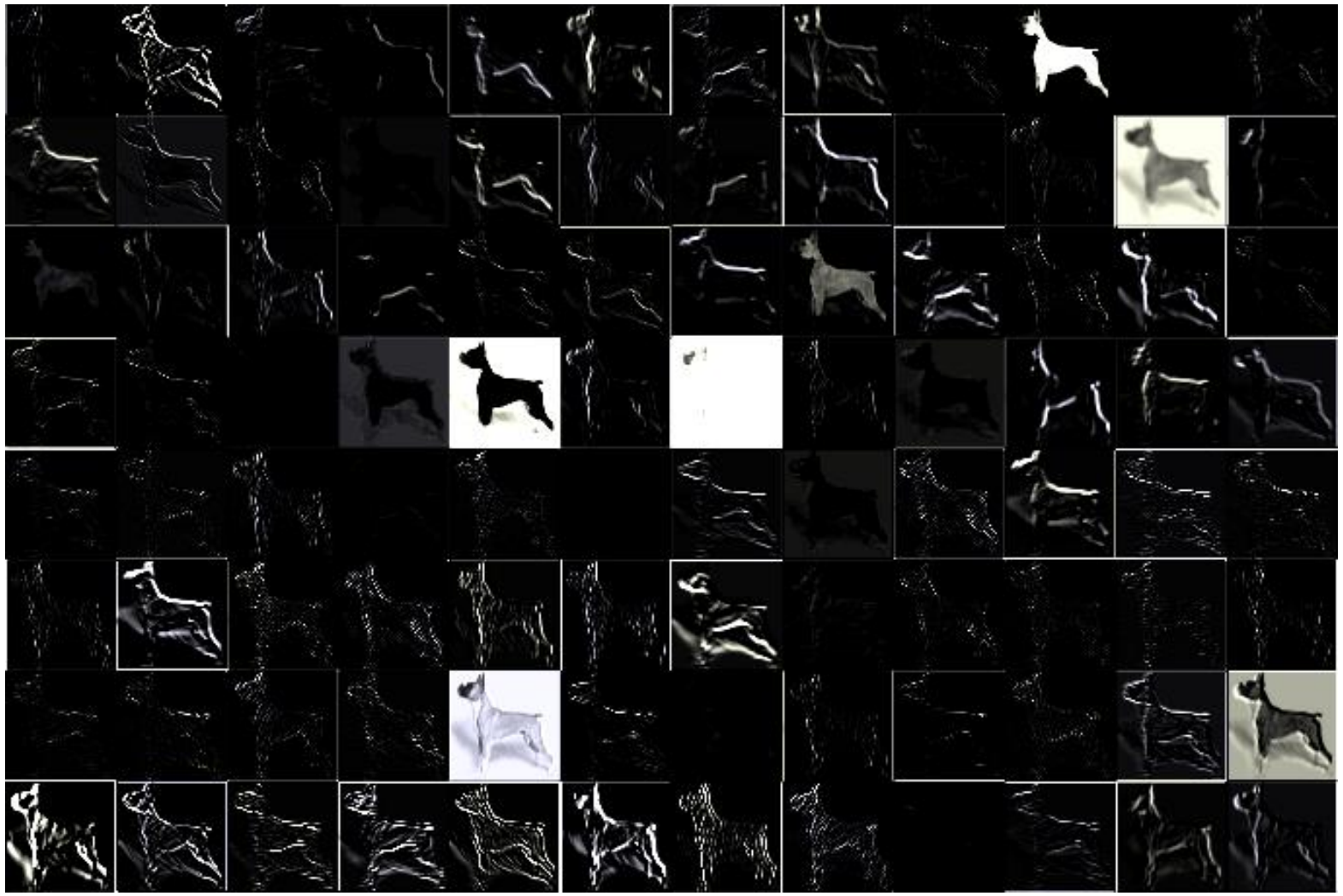
Feature extraction

AlexNet



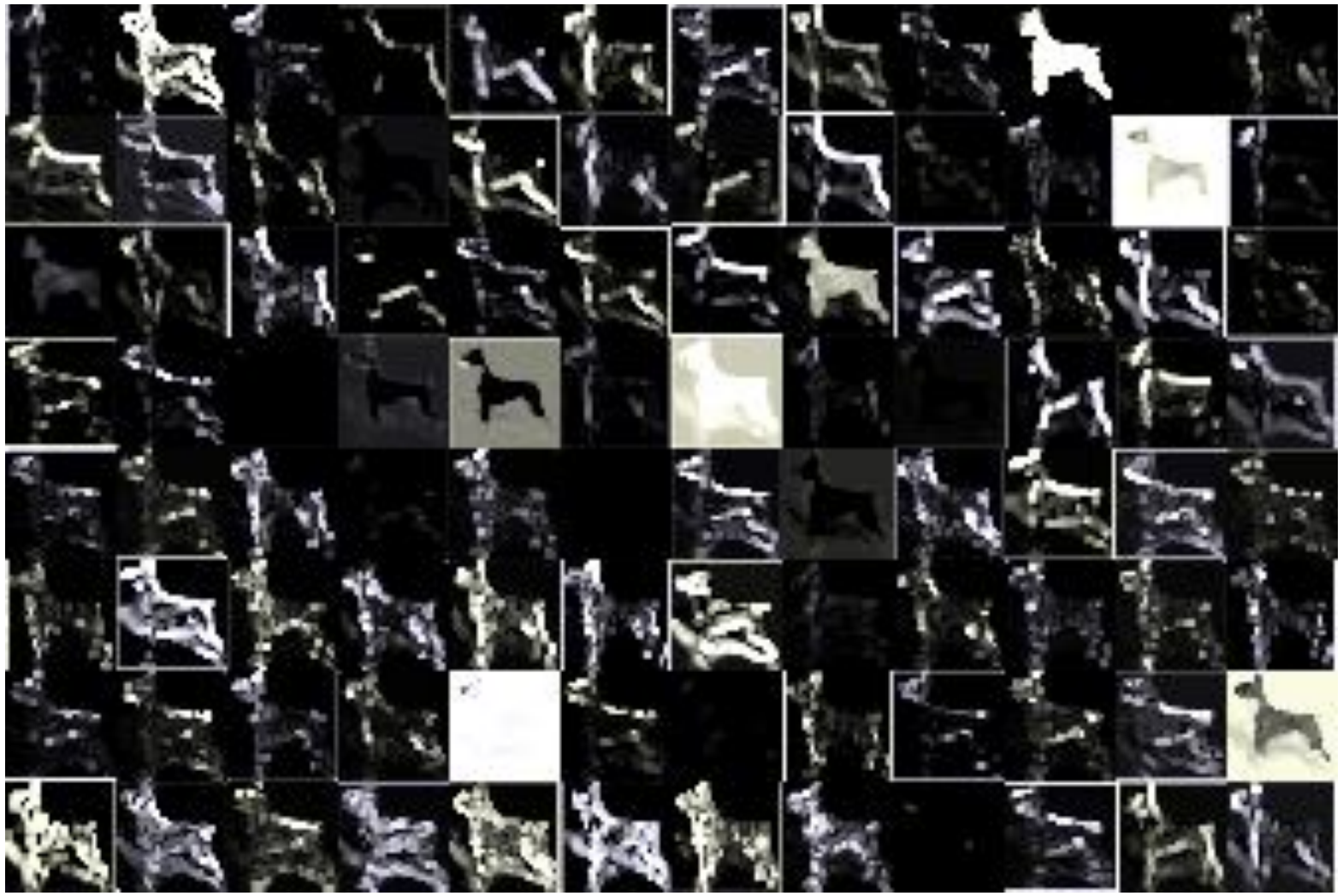
Feature extraction

AlexNet



Feature extraction

AlexNet

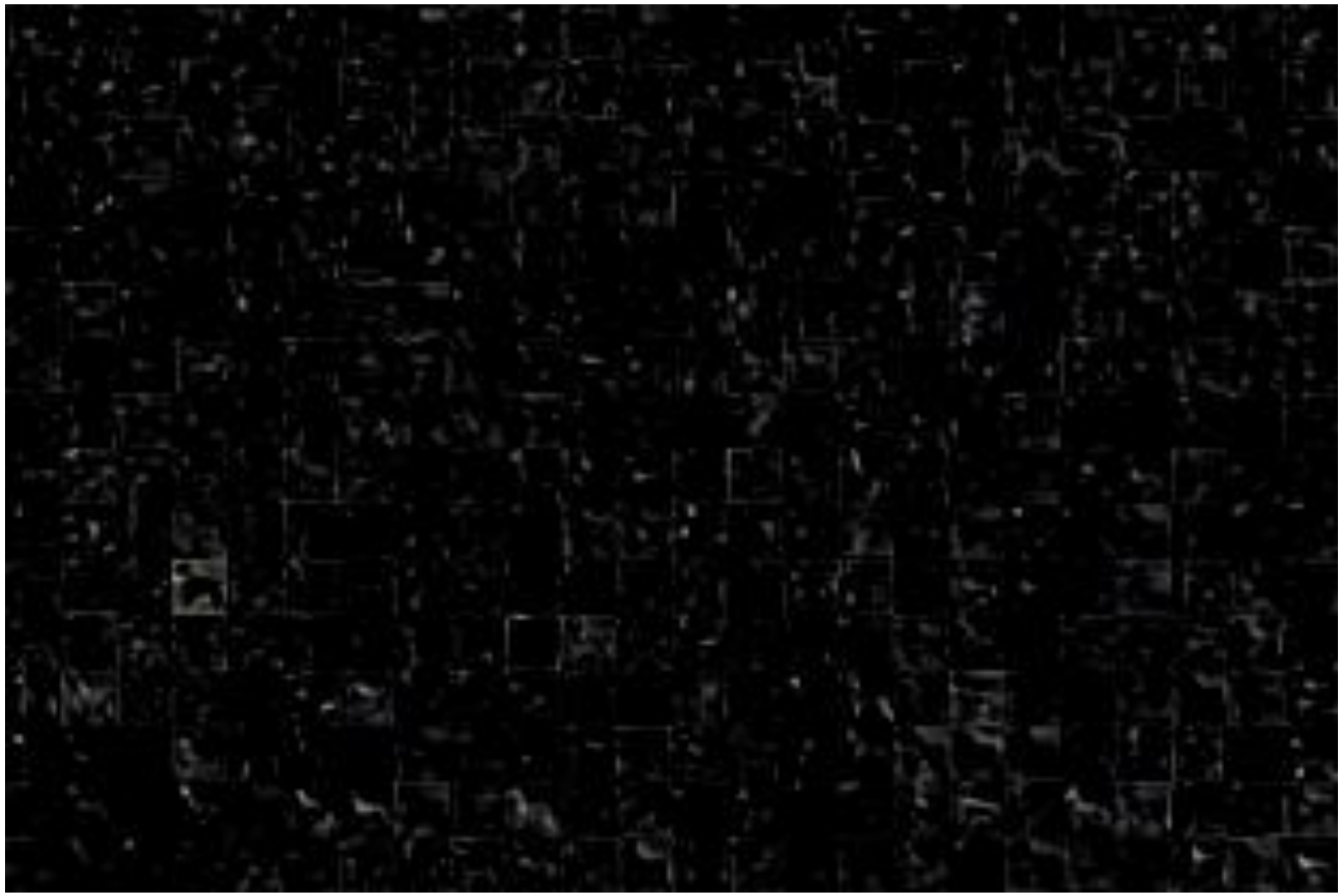


Feature extraction

AlexNet

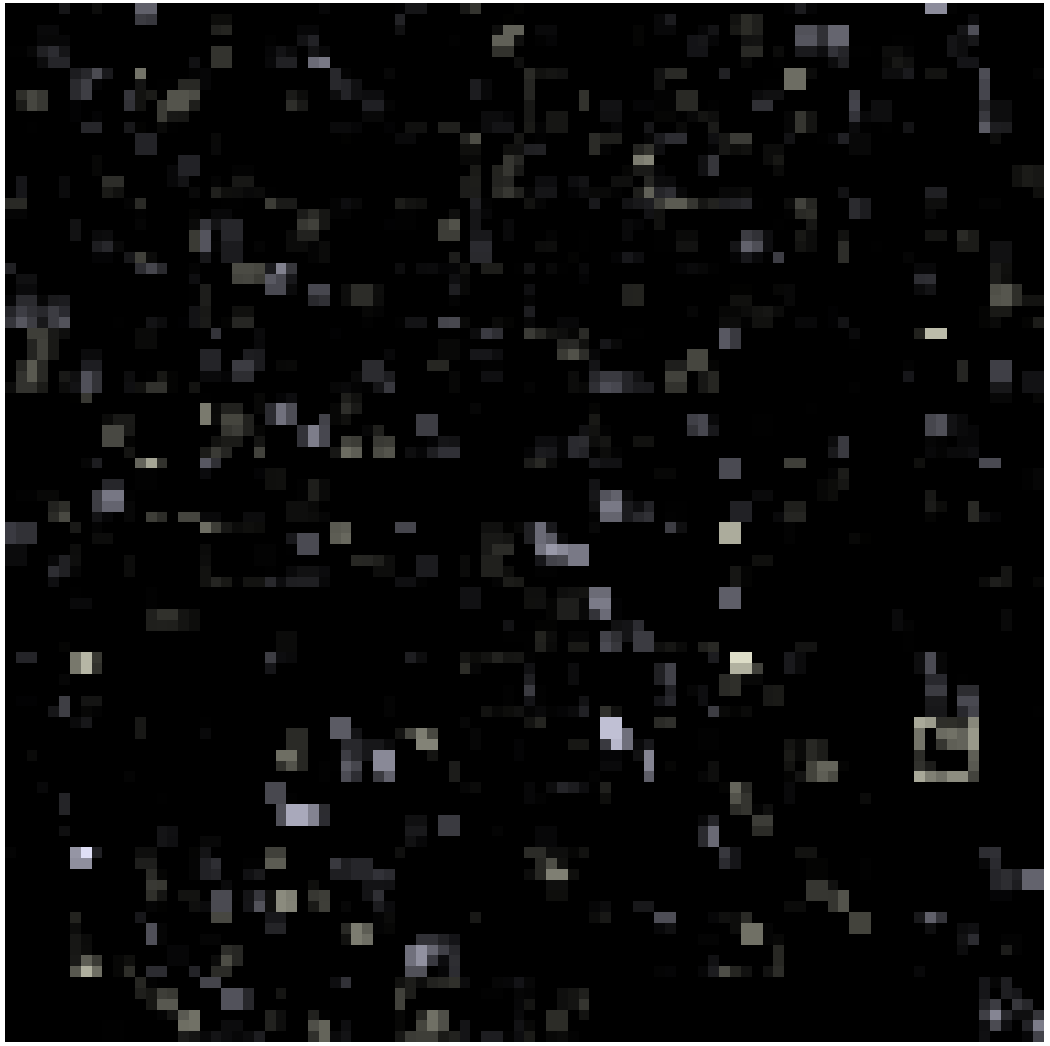
Feature extraction

AlexNet



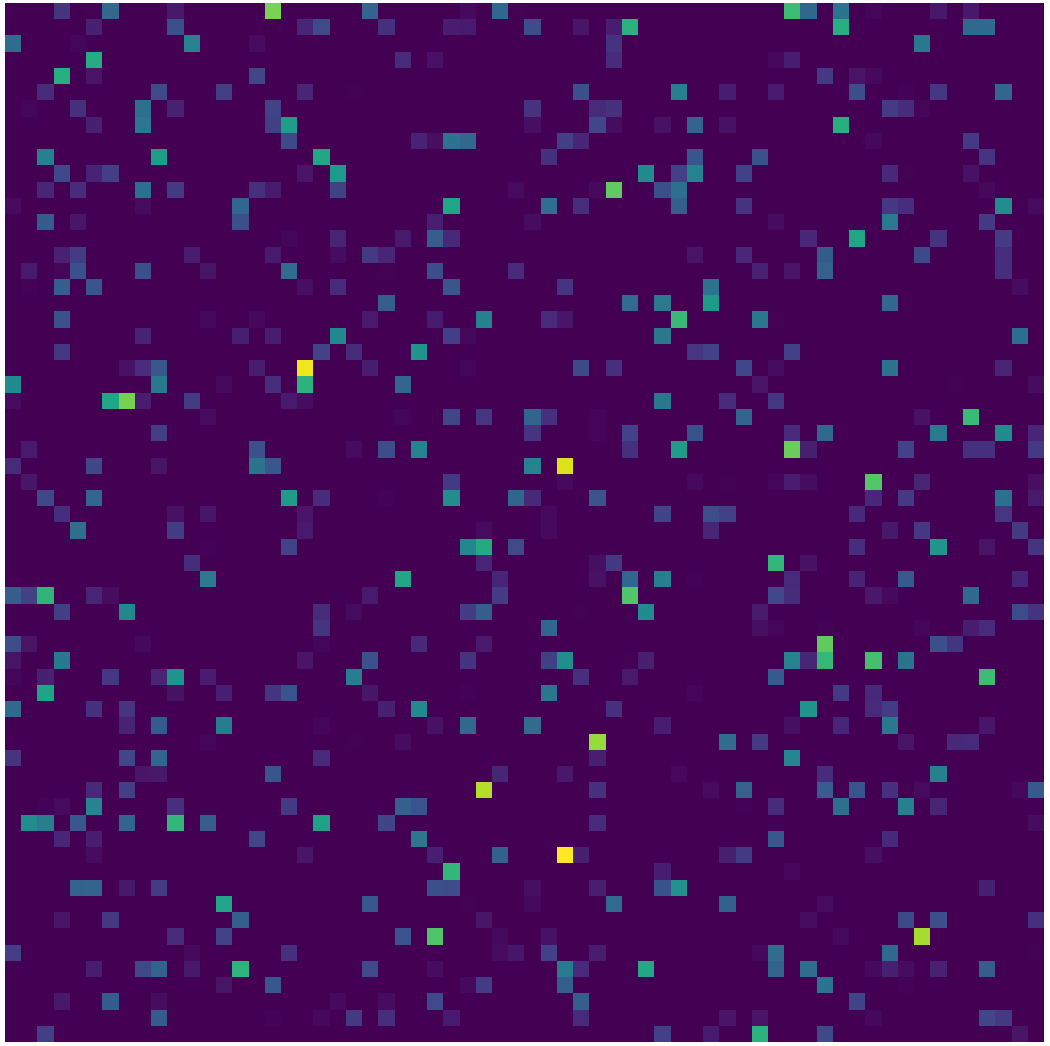
Feature extraction

AlexNet



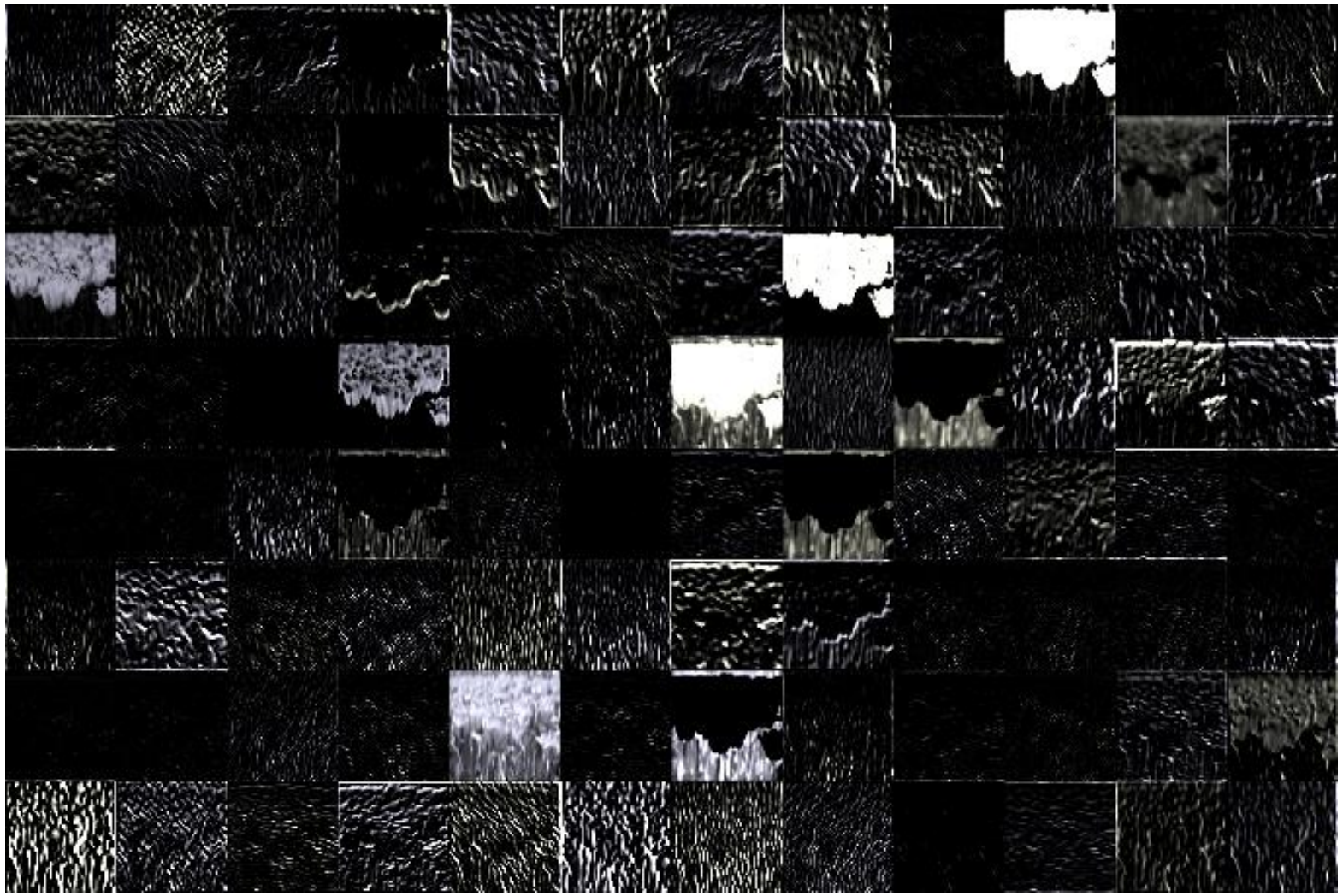
Feature extraction

AlexNet



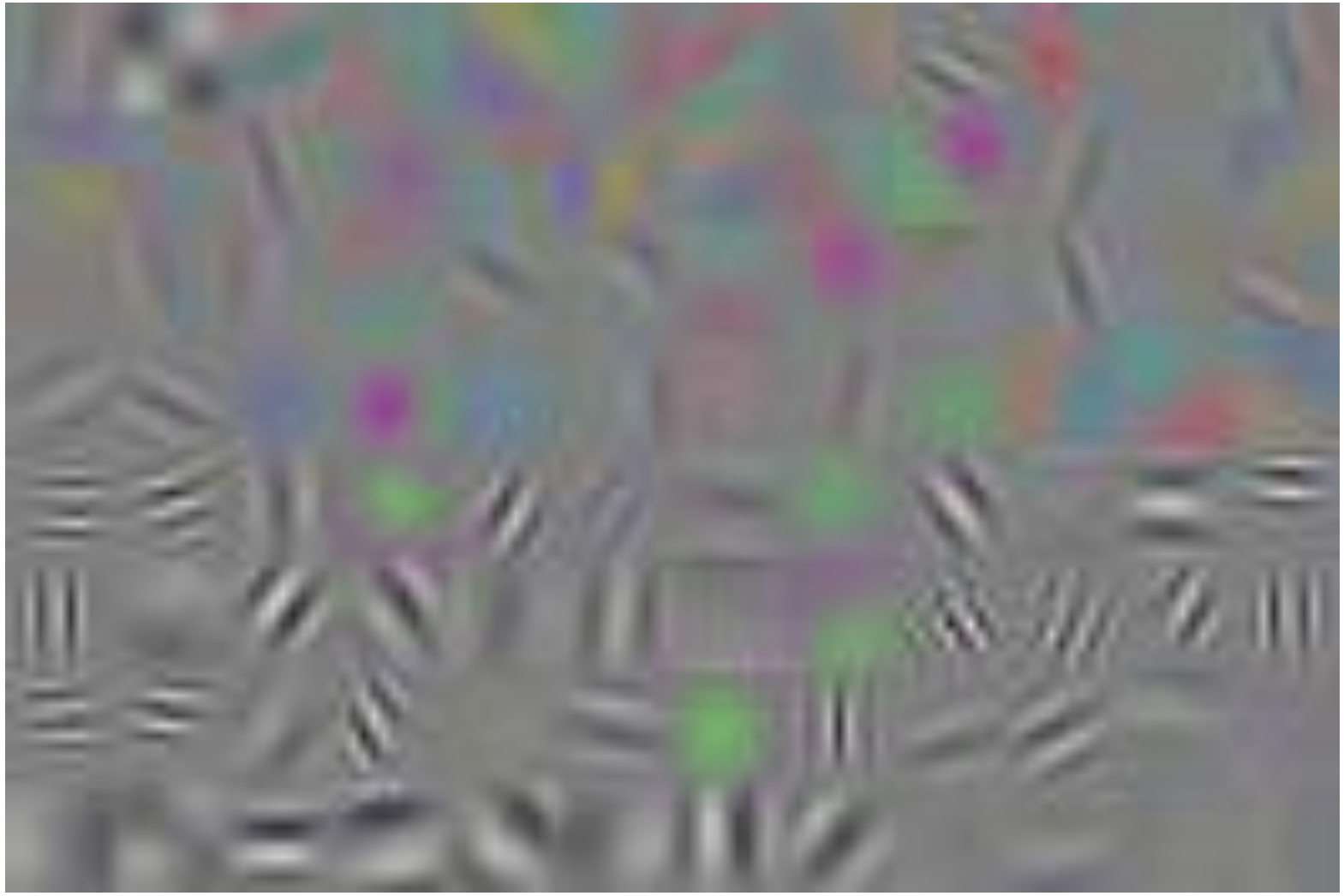
Feature extraction

AlexNet



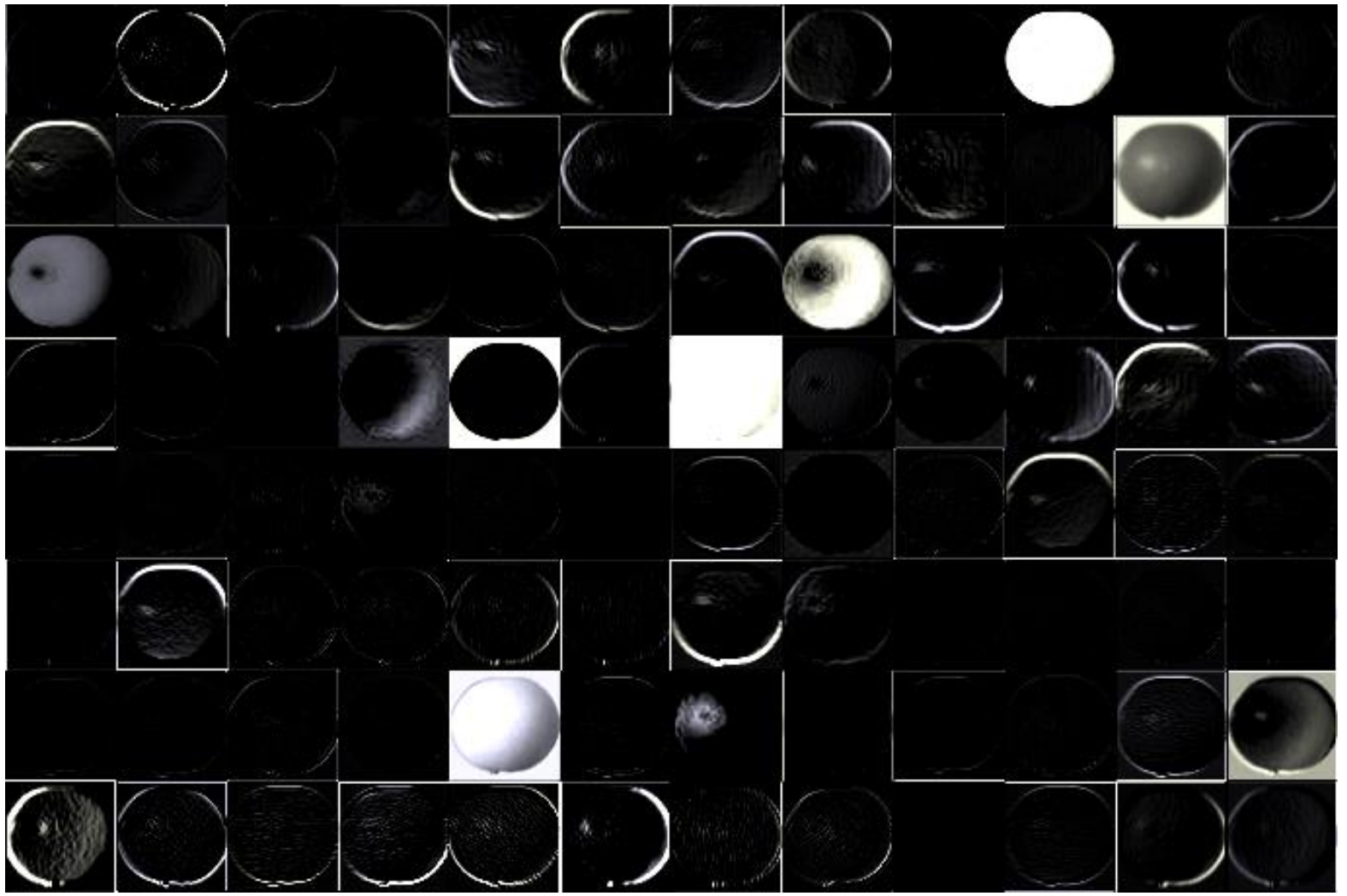
Feature extraction

AlexNet



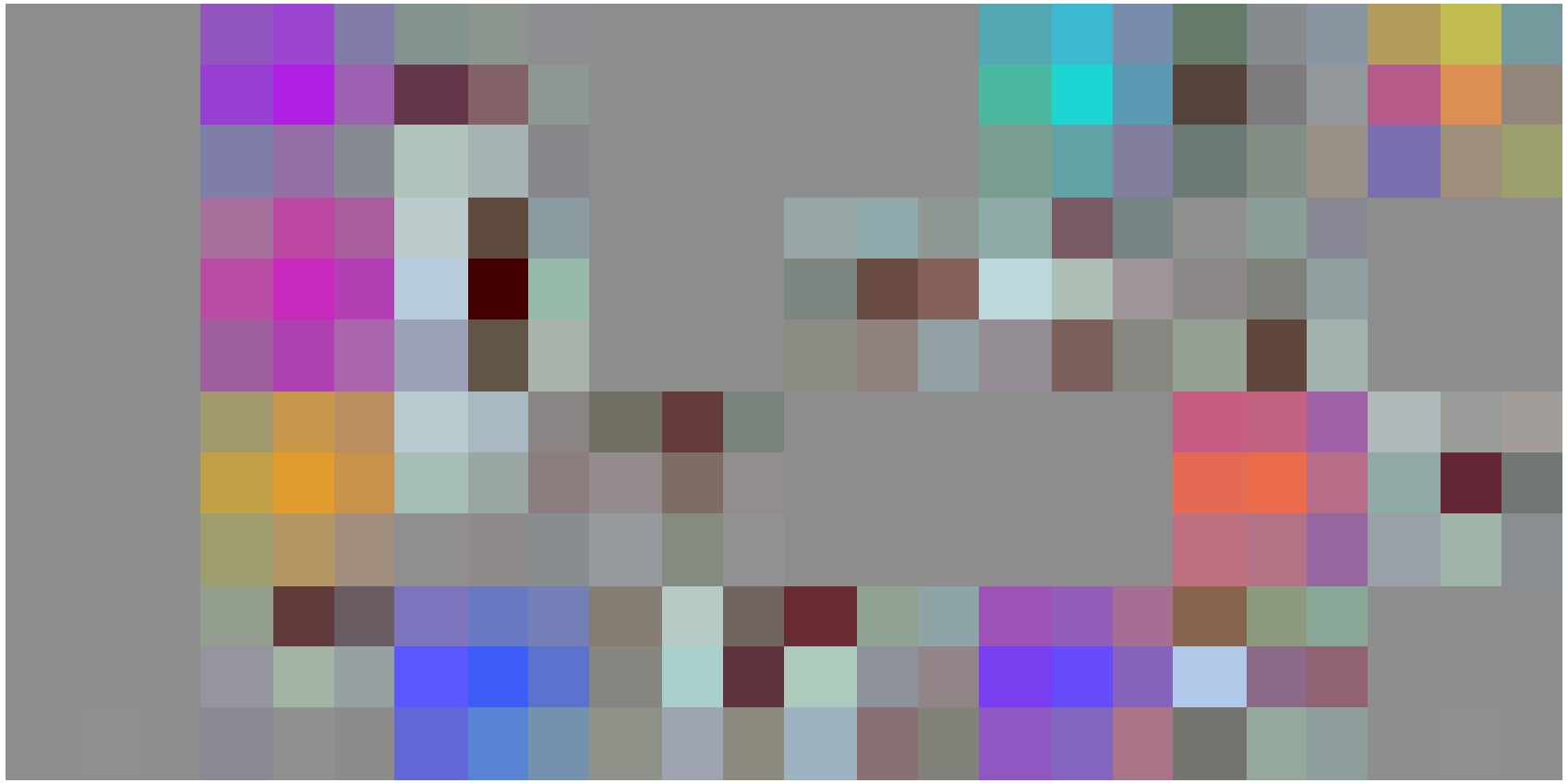
Feature extraction

AlexNet



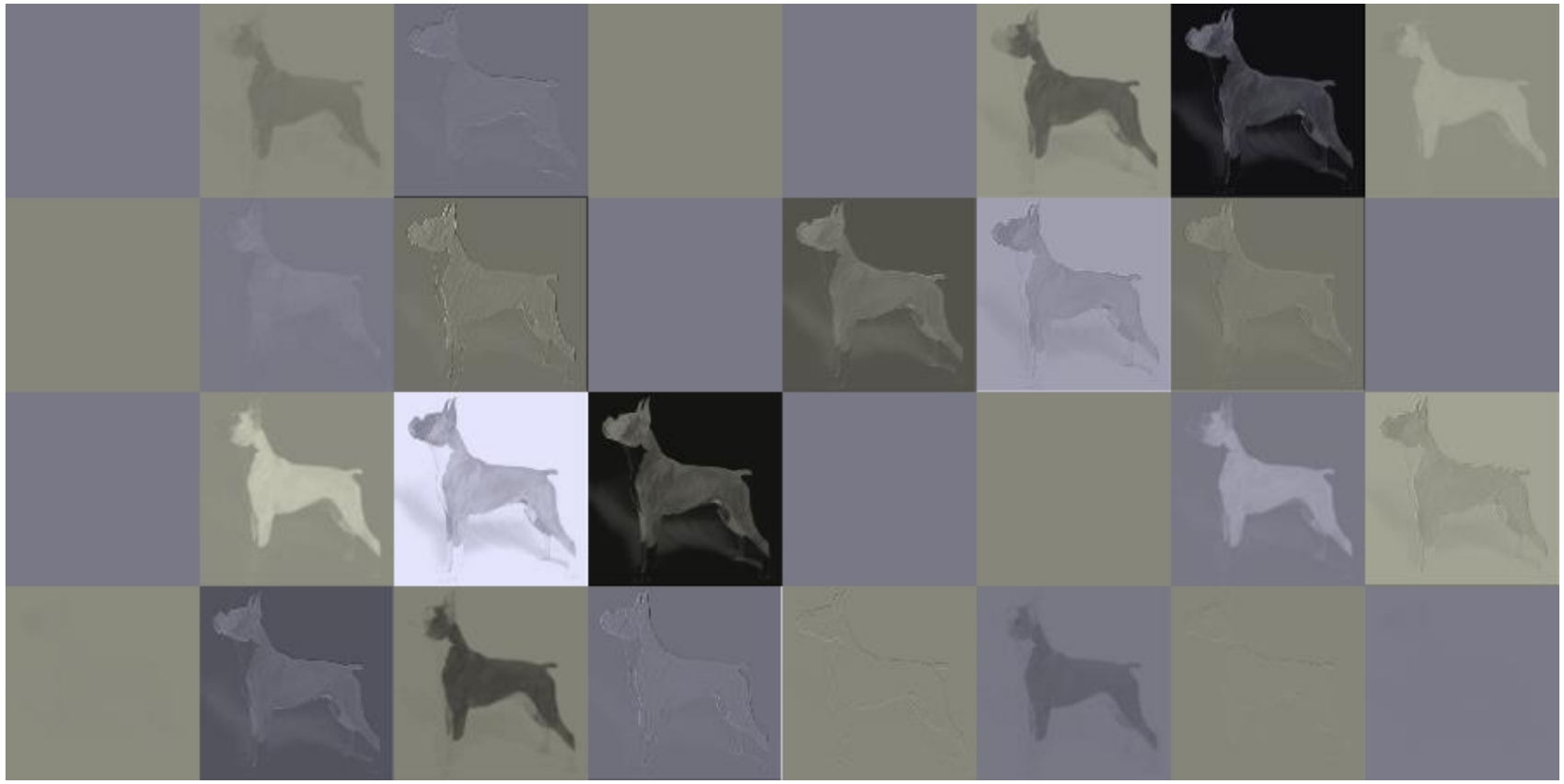
Feature extraction

MobileNet



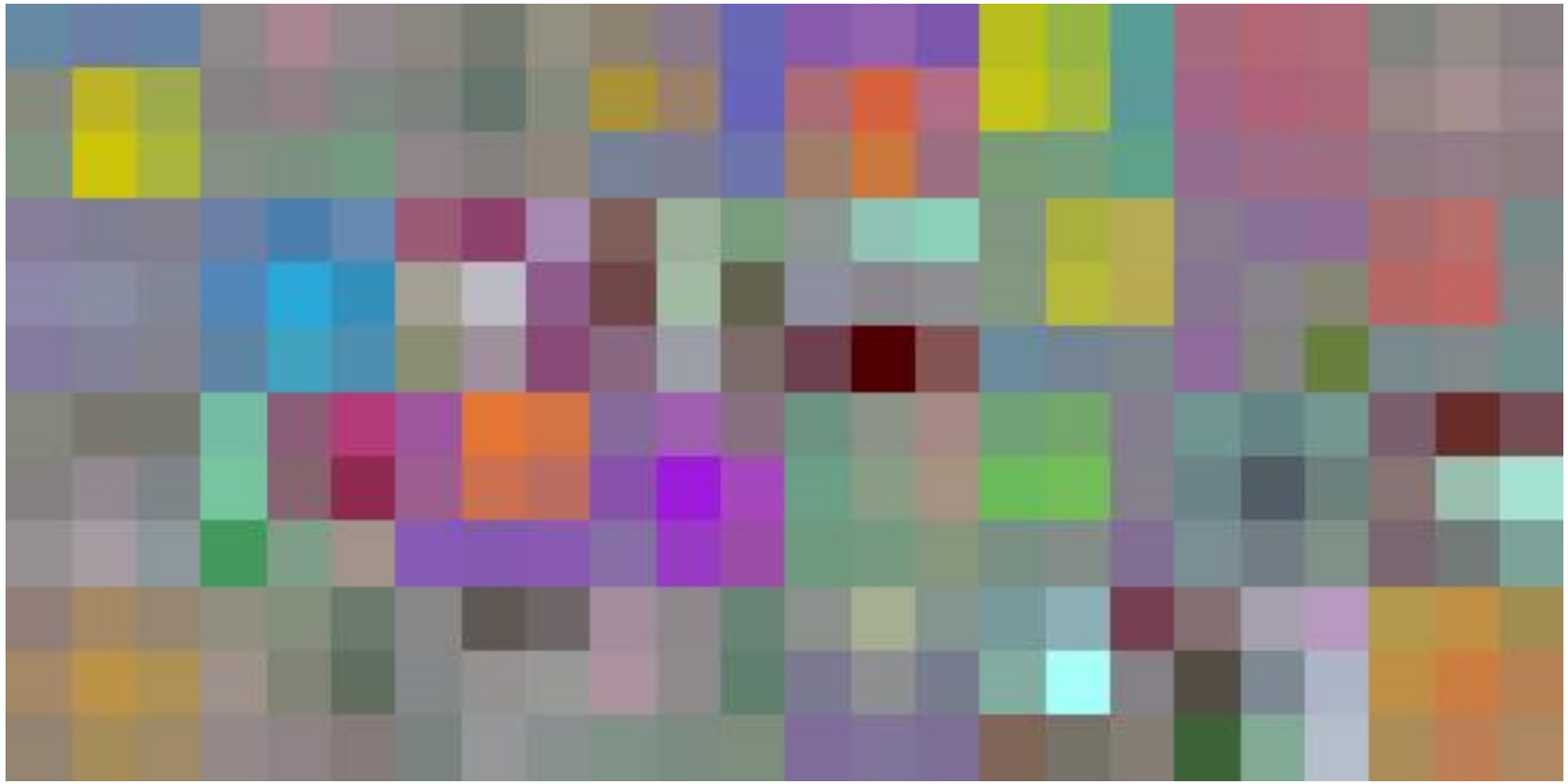
Feature extraction

MobileNet



Feature extraction

Xception



Feature extraction

MobileNet

Transfer learning

Transfer learning

Fine tuning

Finetuning

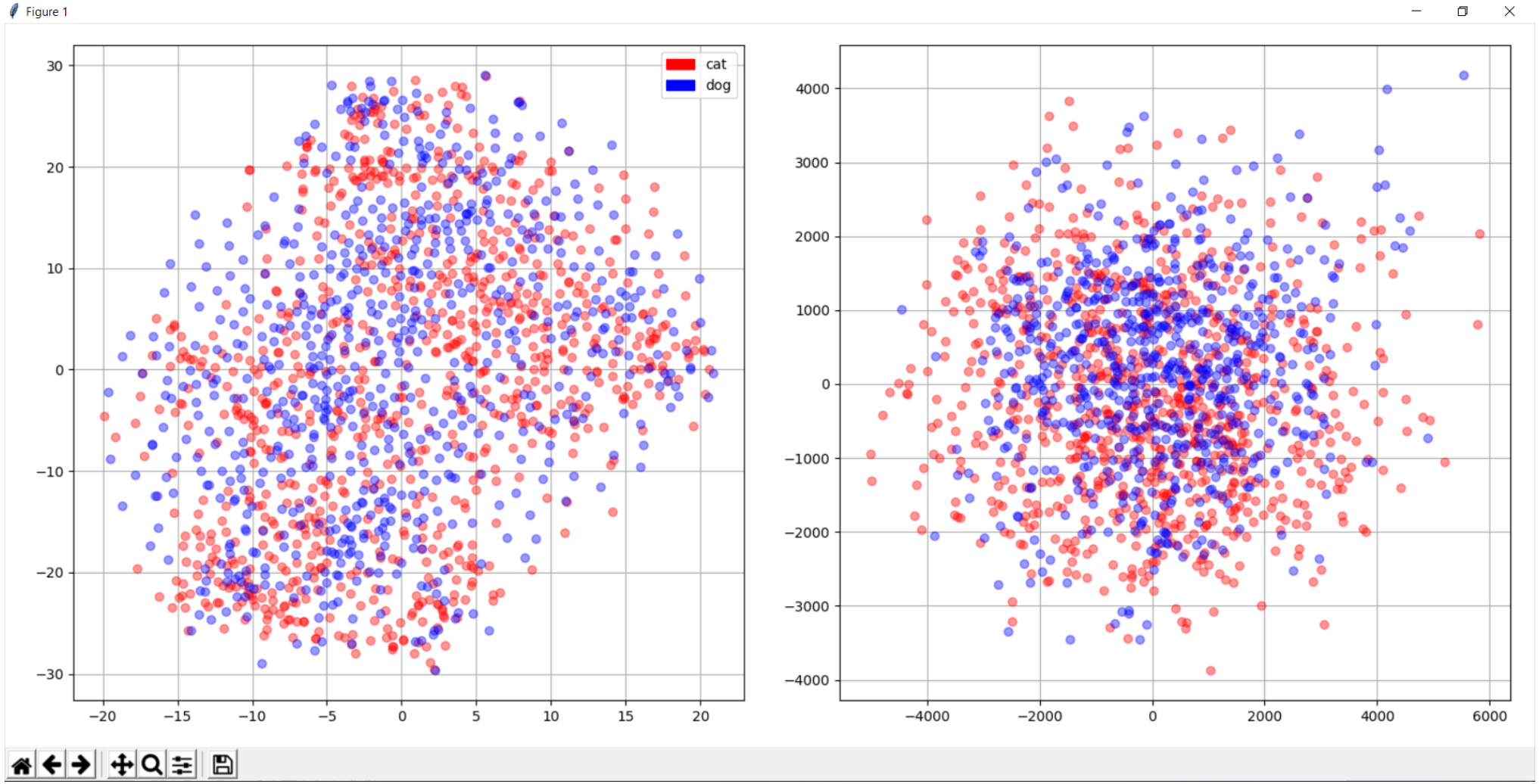


Figure 1
cat
dog
30
20
10
0
−10
−20
−30
−20
−15
−10
−5
0
5
10
15
20
4000
3000
2000
1000
0
−1000
−2000
−3000
−4000
−4000
−2000
0
2000
4000
6000

Finetuning

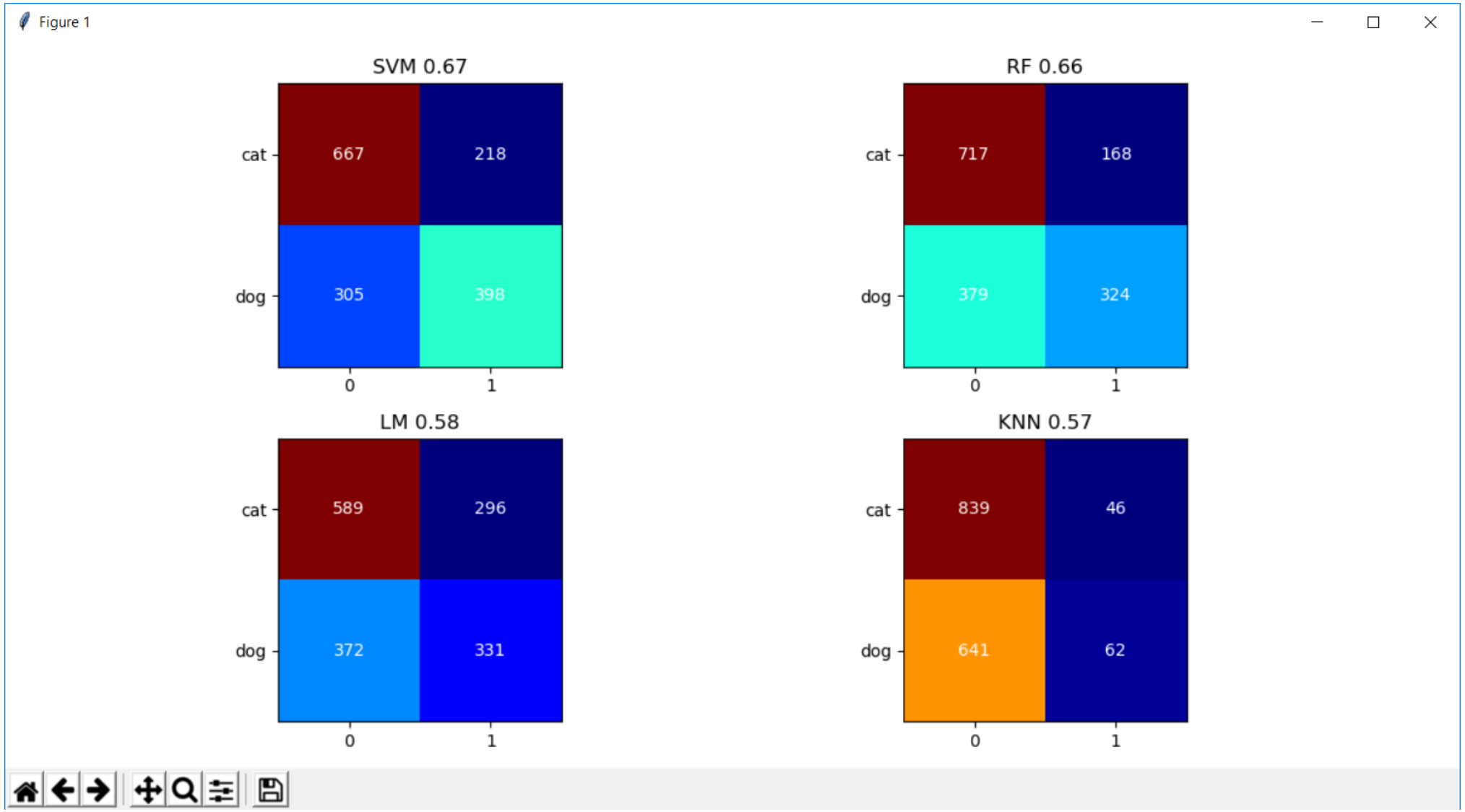


Figure 1
SVM 0.67
cat
dog
667
218
305
398
0
1
RF 0.66
cat
dog
717
168
379
324
0
1
LM 0.58
cat
dog
589
296
372
331
0
1
KNN 0.57
cat
dog
839
46
641
62
0
1

Finetuning

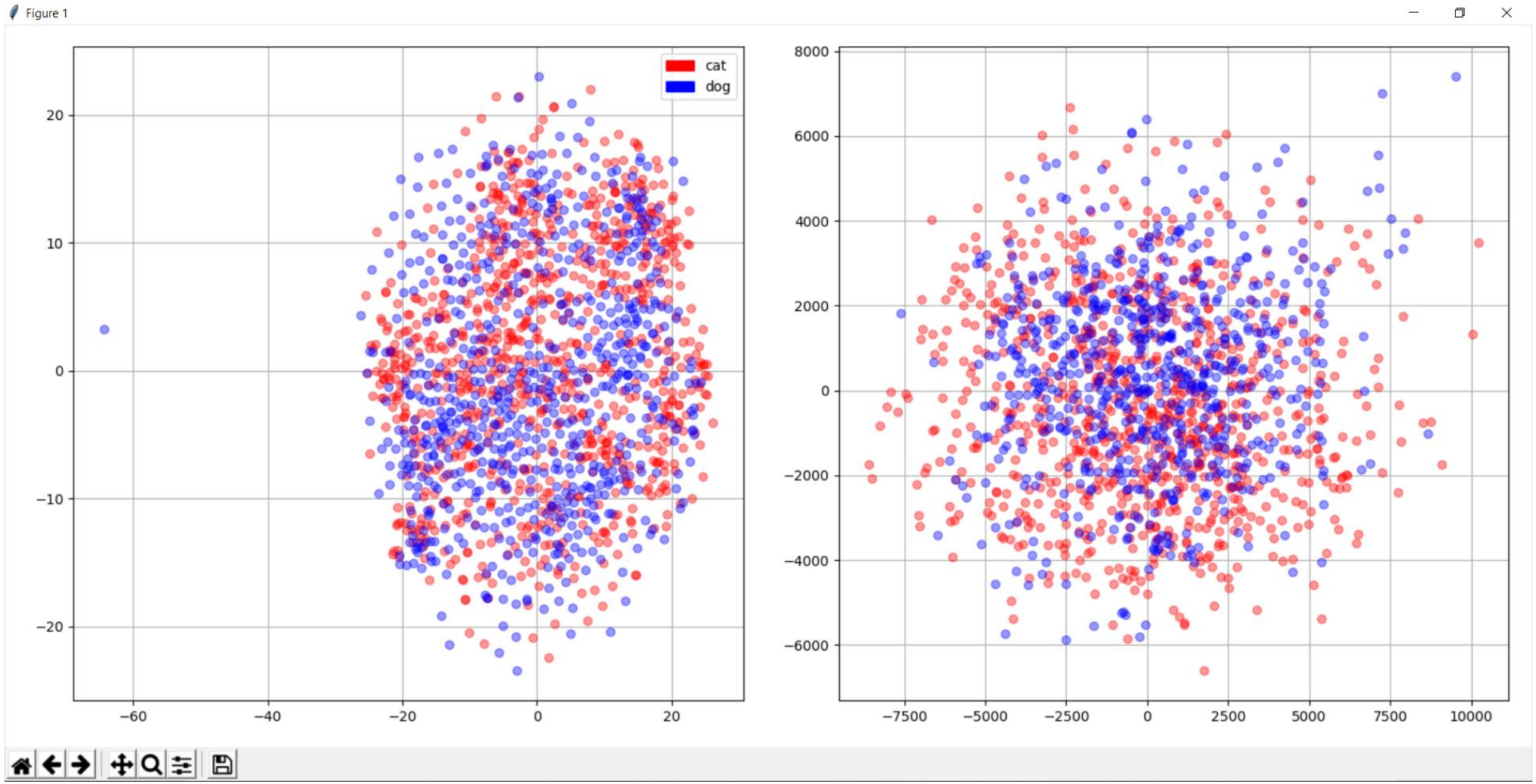


Figure 1
cat
dog
20
10
0
−10
−20
−60
−40
−20
0
20
8000
6000
4000
2000
0
−2000
−4000
−6000
−7500
−5000
−2500
0
2500
5000
7500
10000

Finetuning

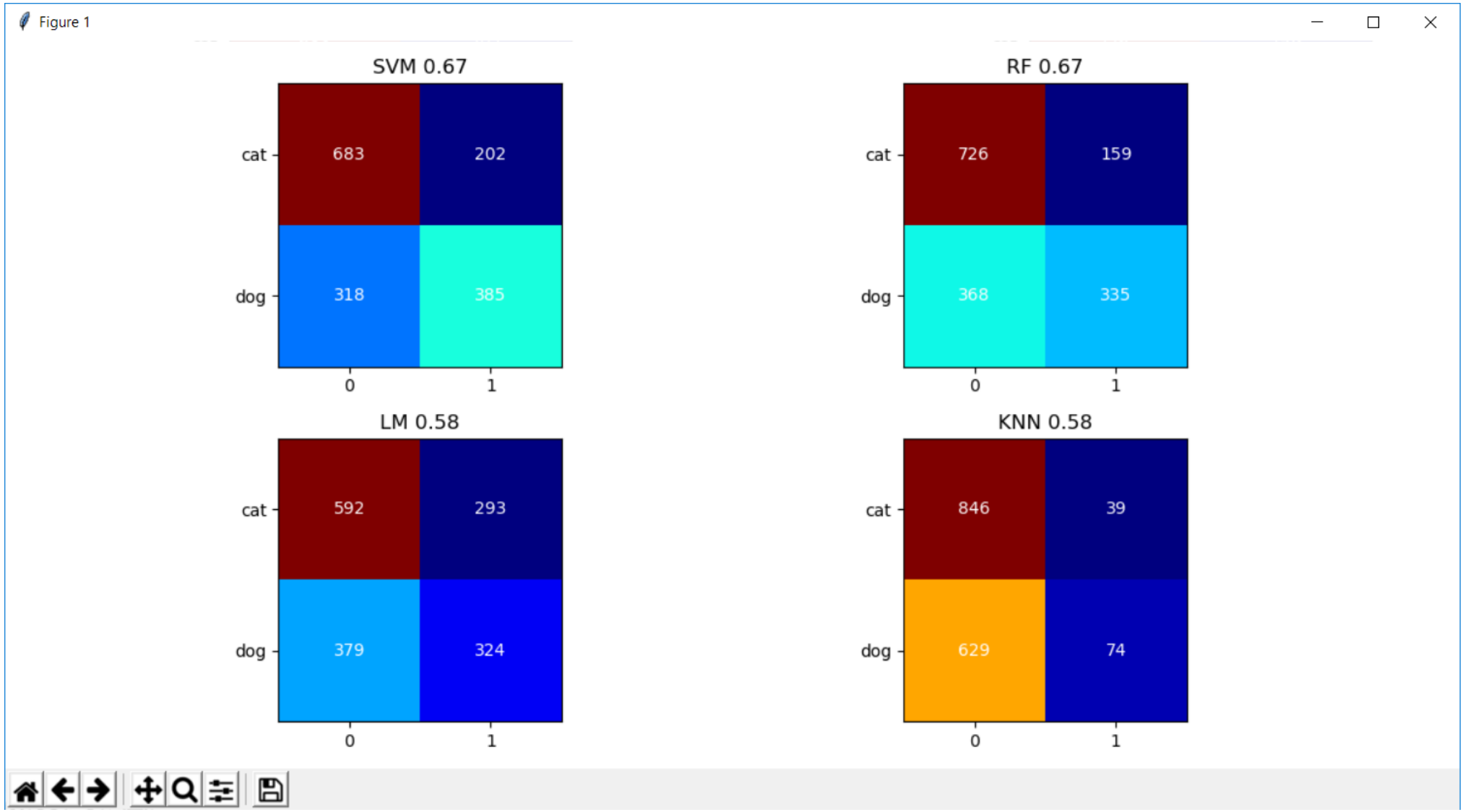
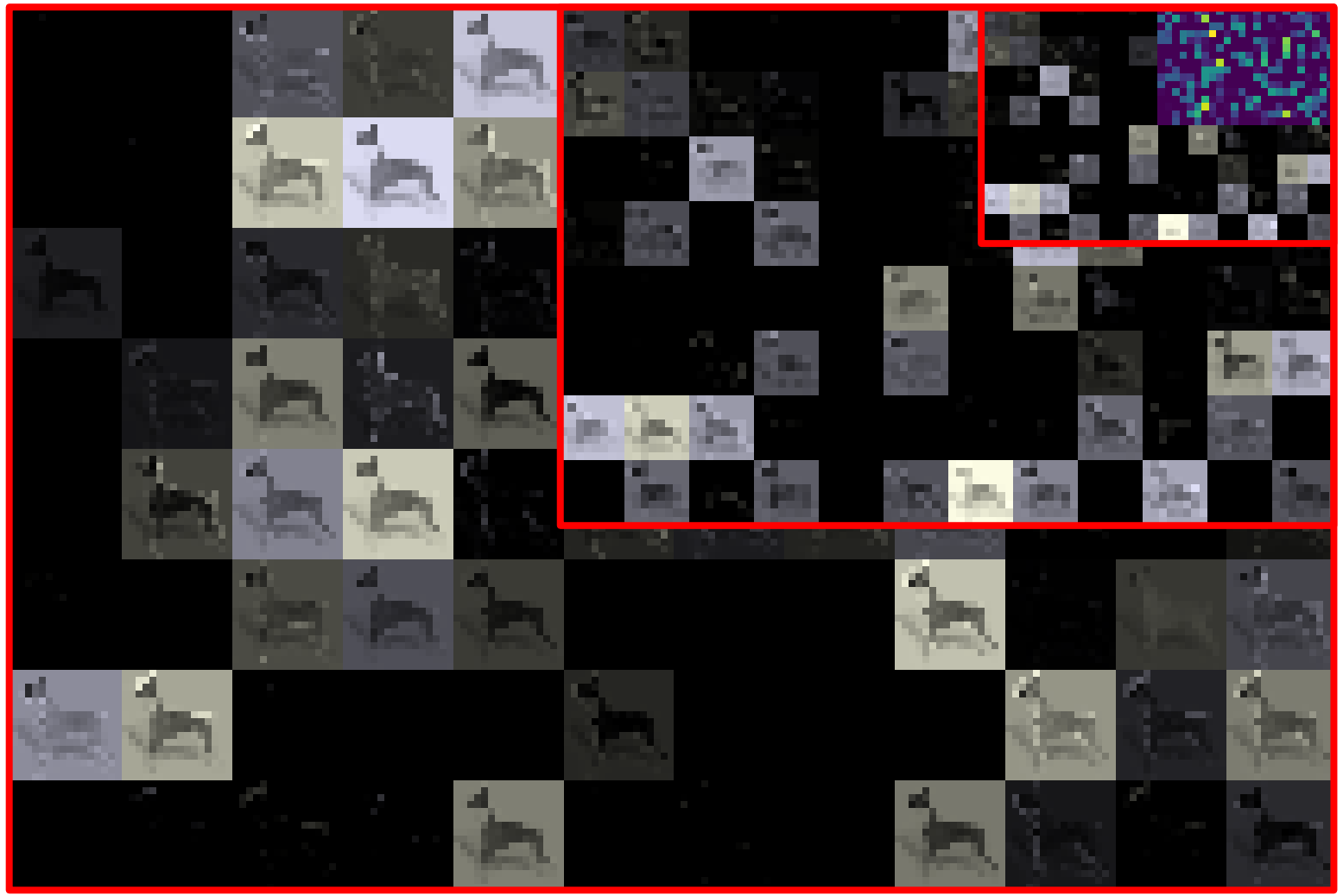


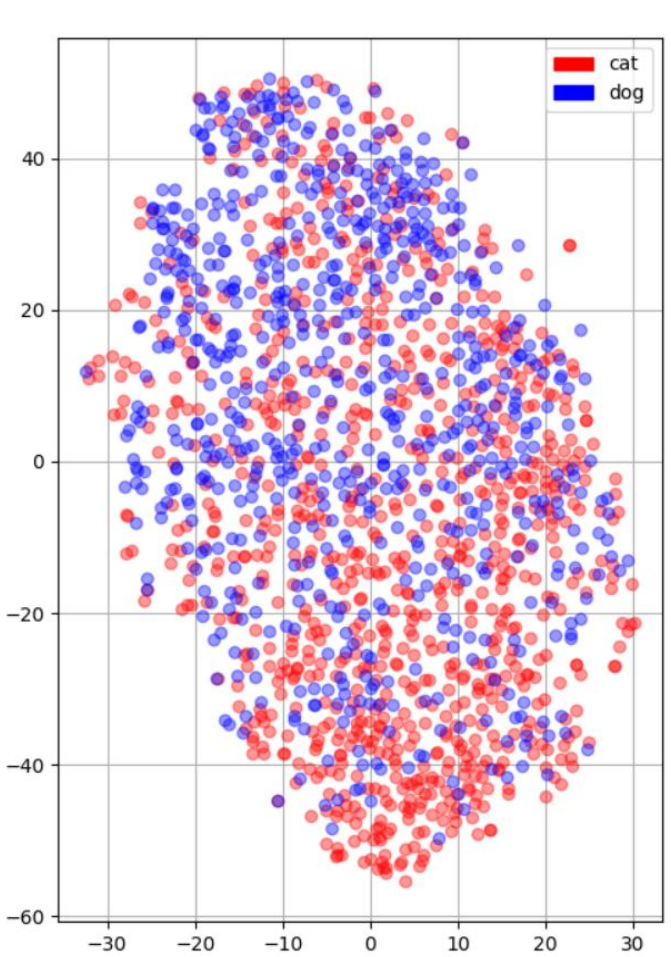
Figure 1
SVM 0.67
cat
dog
683
202
318
385
0
1
RF 0.67
cat
dog
726
159
368
335
0
1
LM 0.58
cat
dog
592
293
379
324
0
1
KNN 0.58
cat
dog
846
39
629
74
0
1

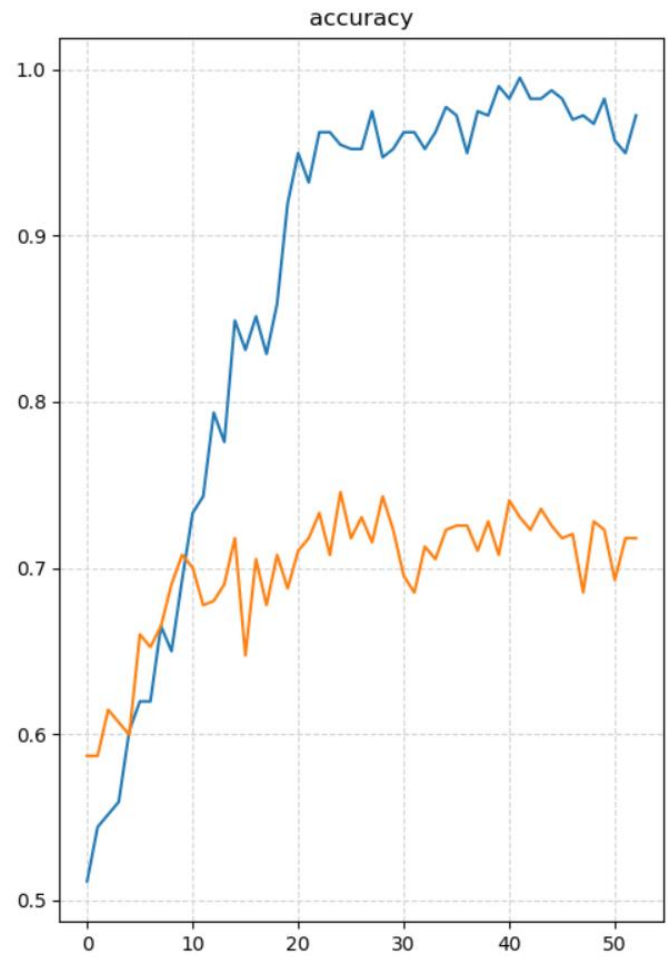
Finetuning

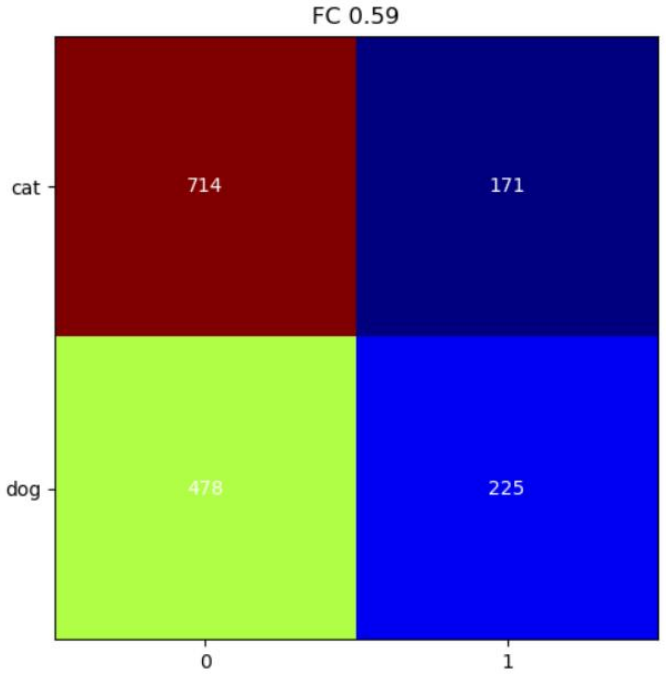


Finetuning

Figure 1







x=0.219177 y= [714]

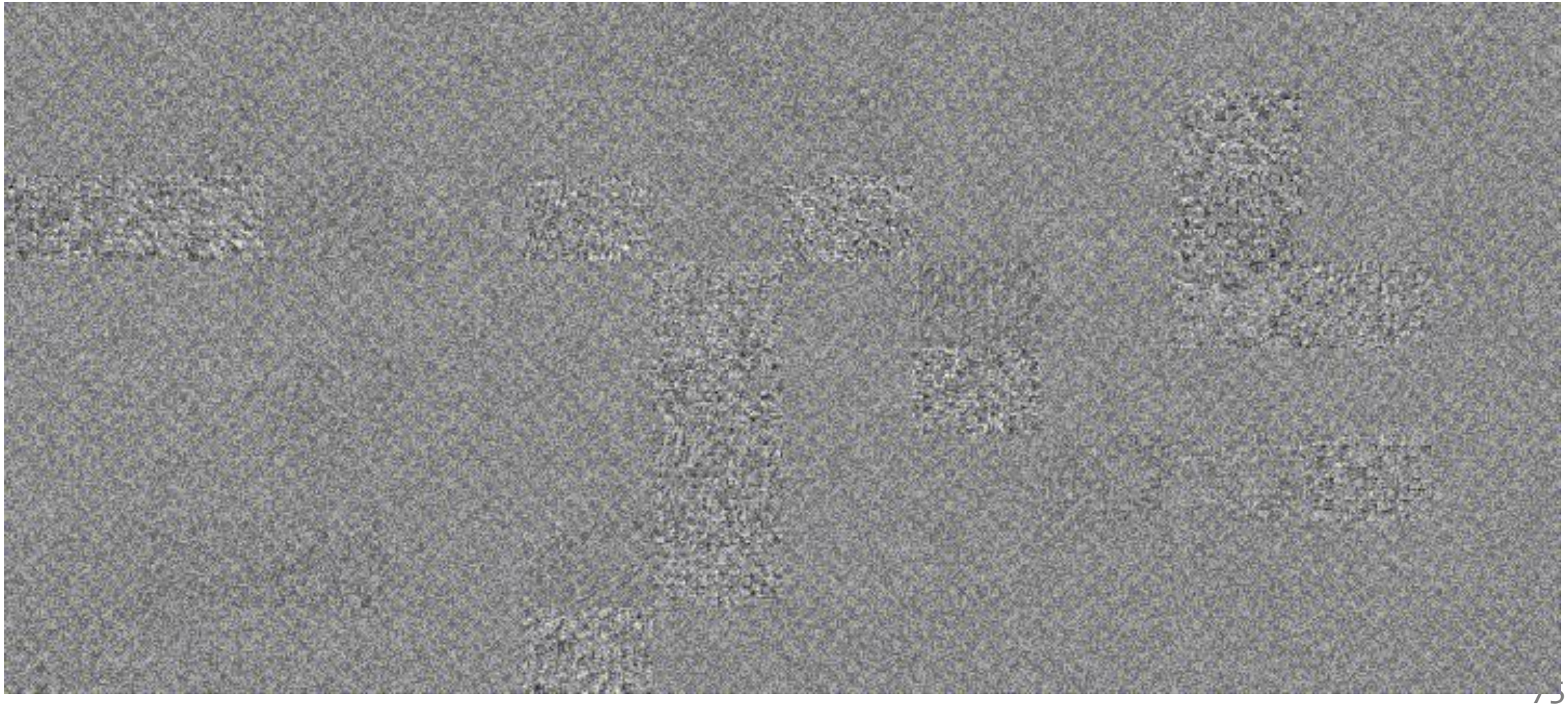
Finetuning

Random before learning

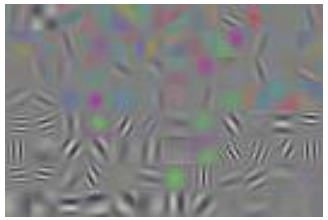
After learning

Finetuning

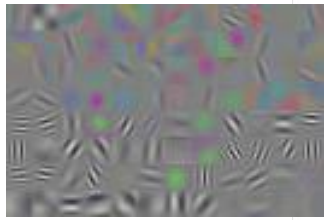
Filters after learning

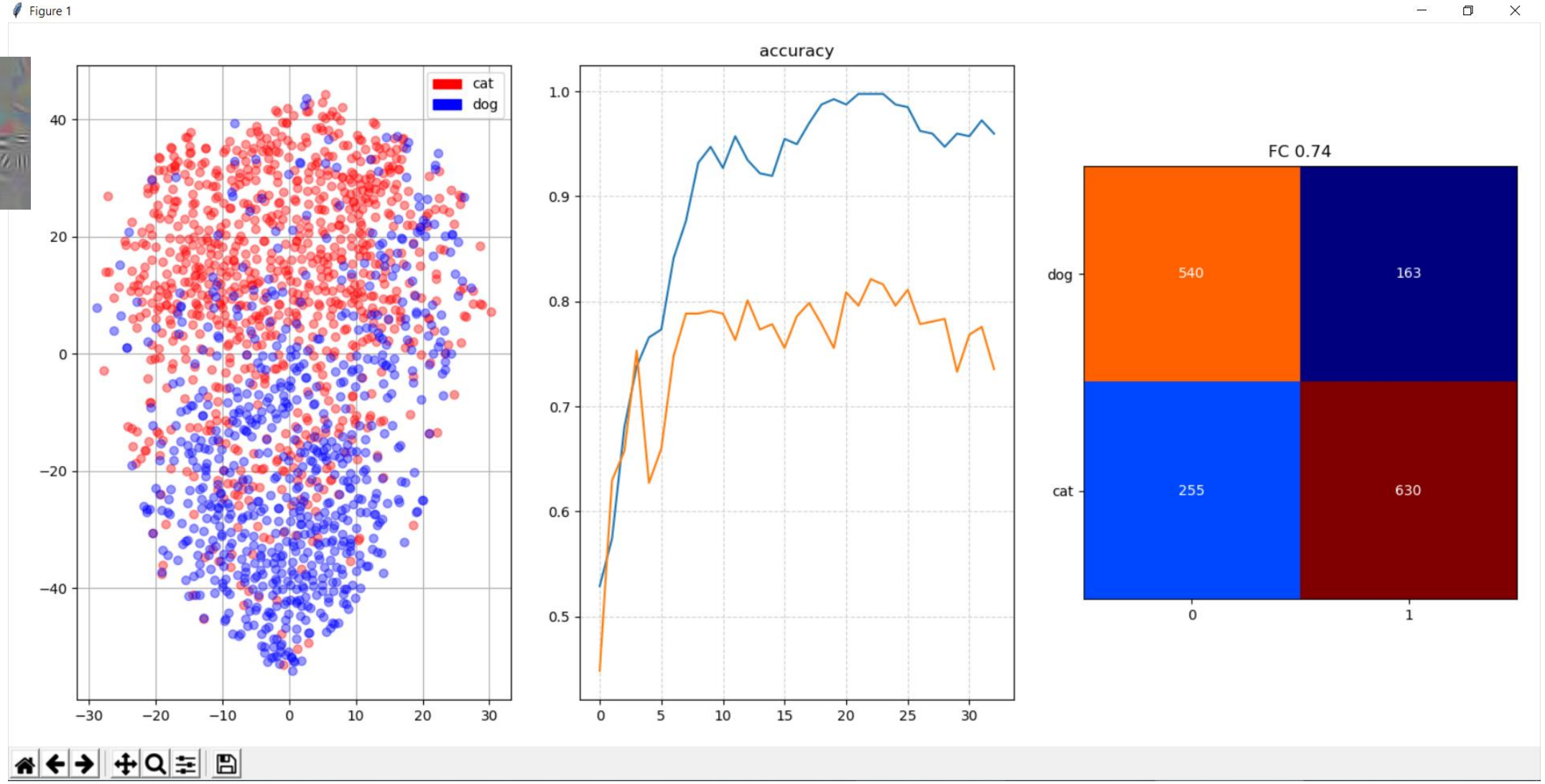


Finetuning



Finetuning

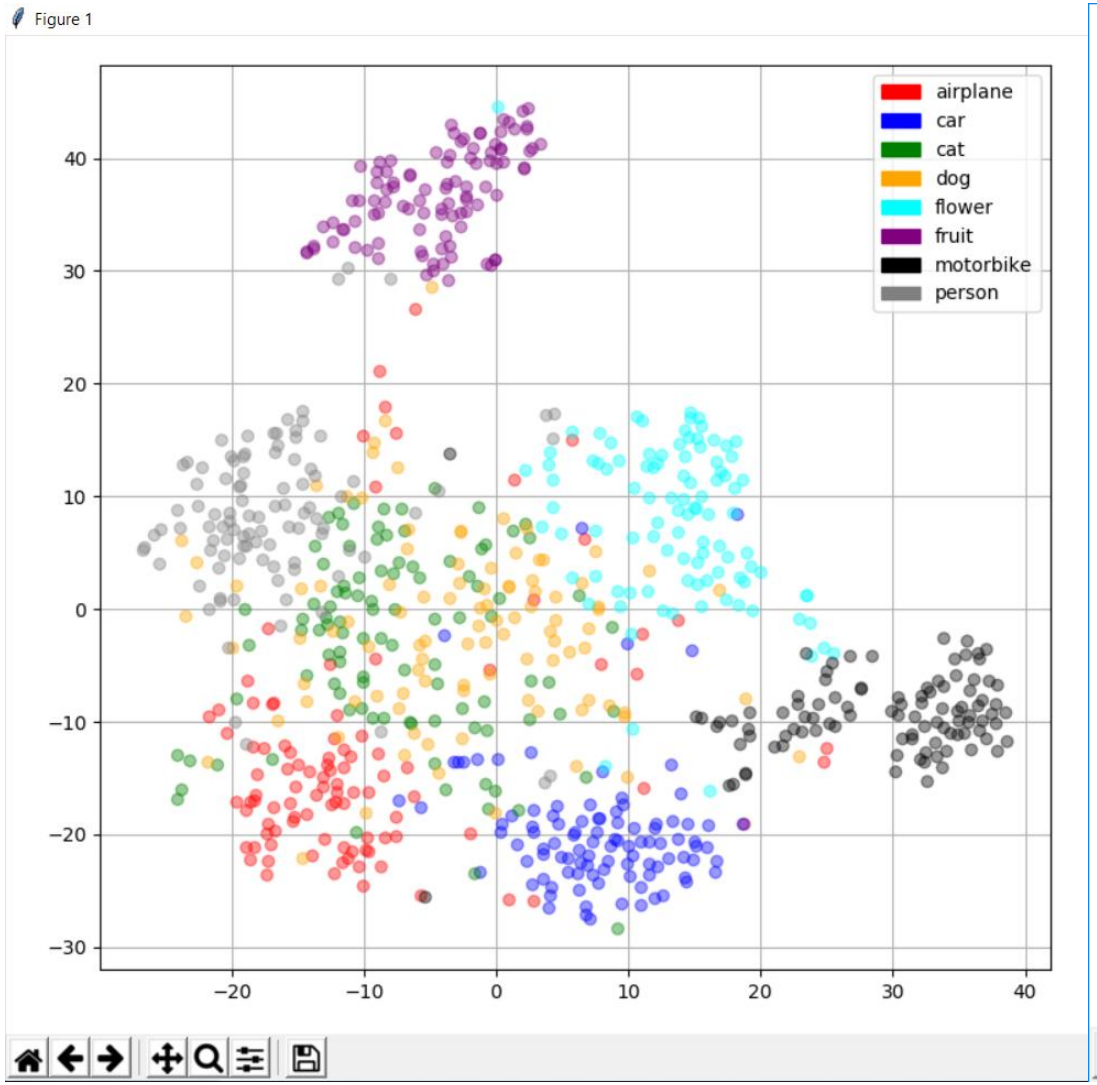


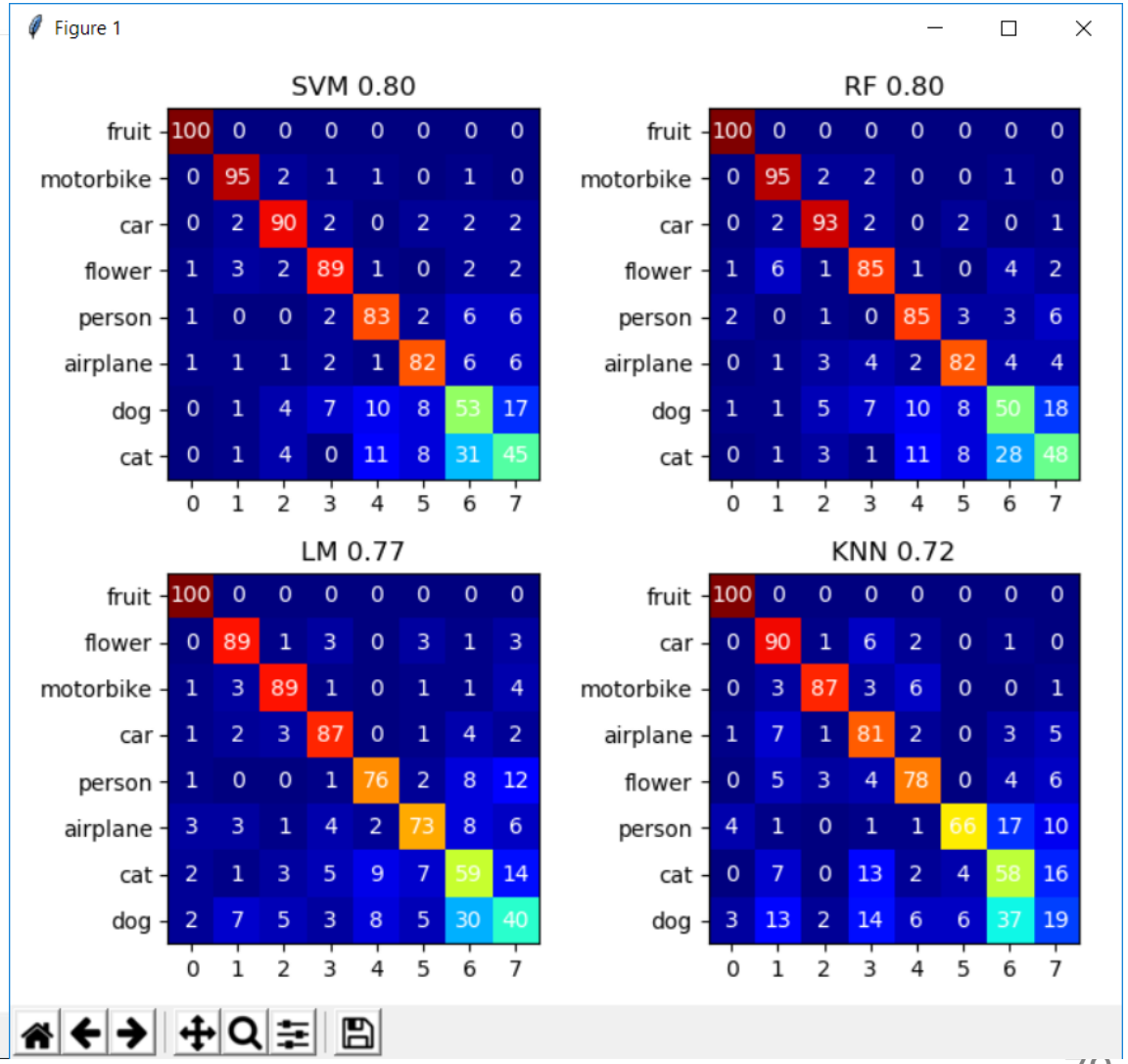


Benchmarking the CNNs

Benchmarking the CNNs

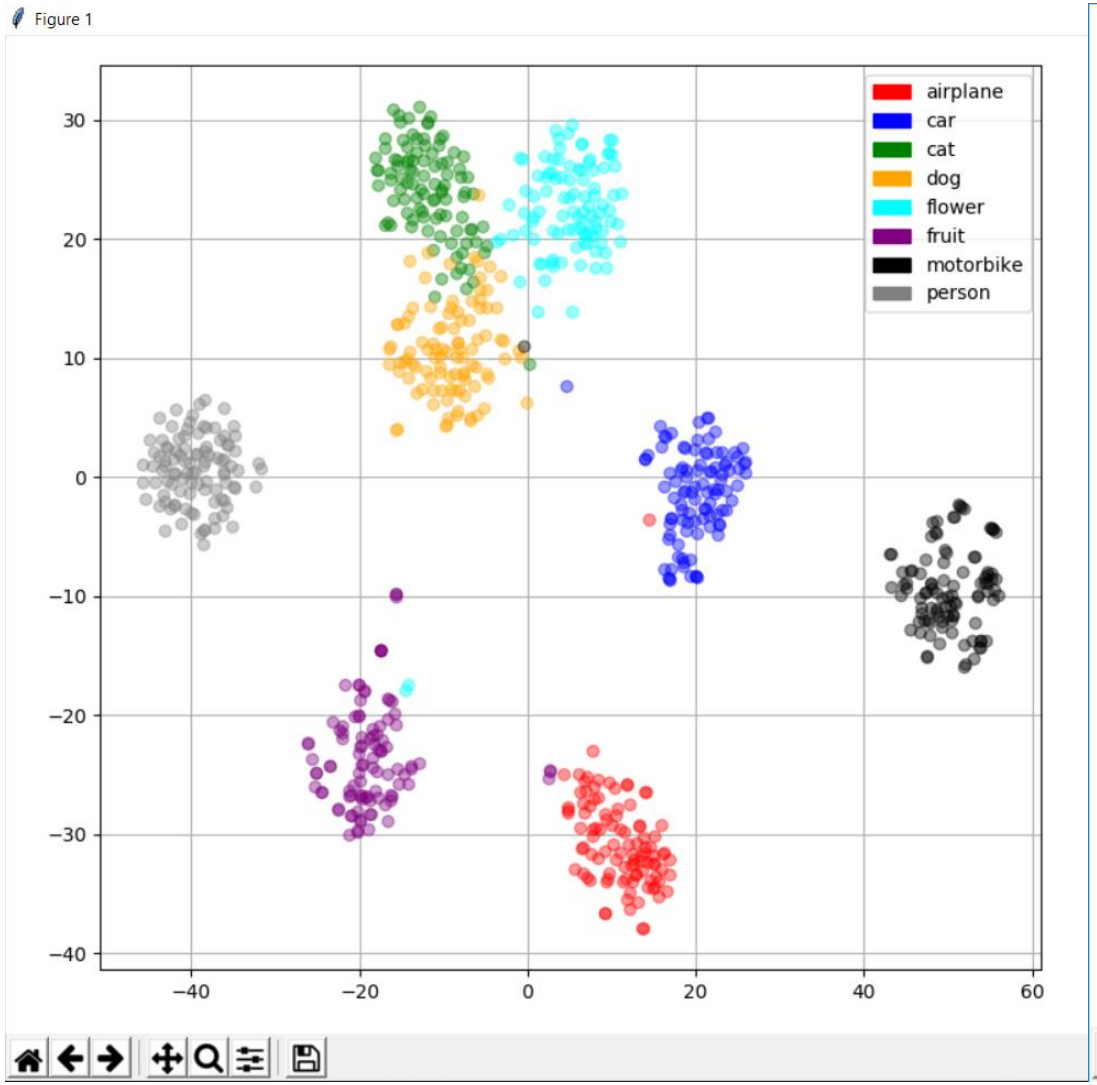
2 Conv layers

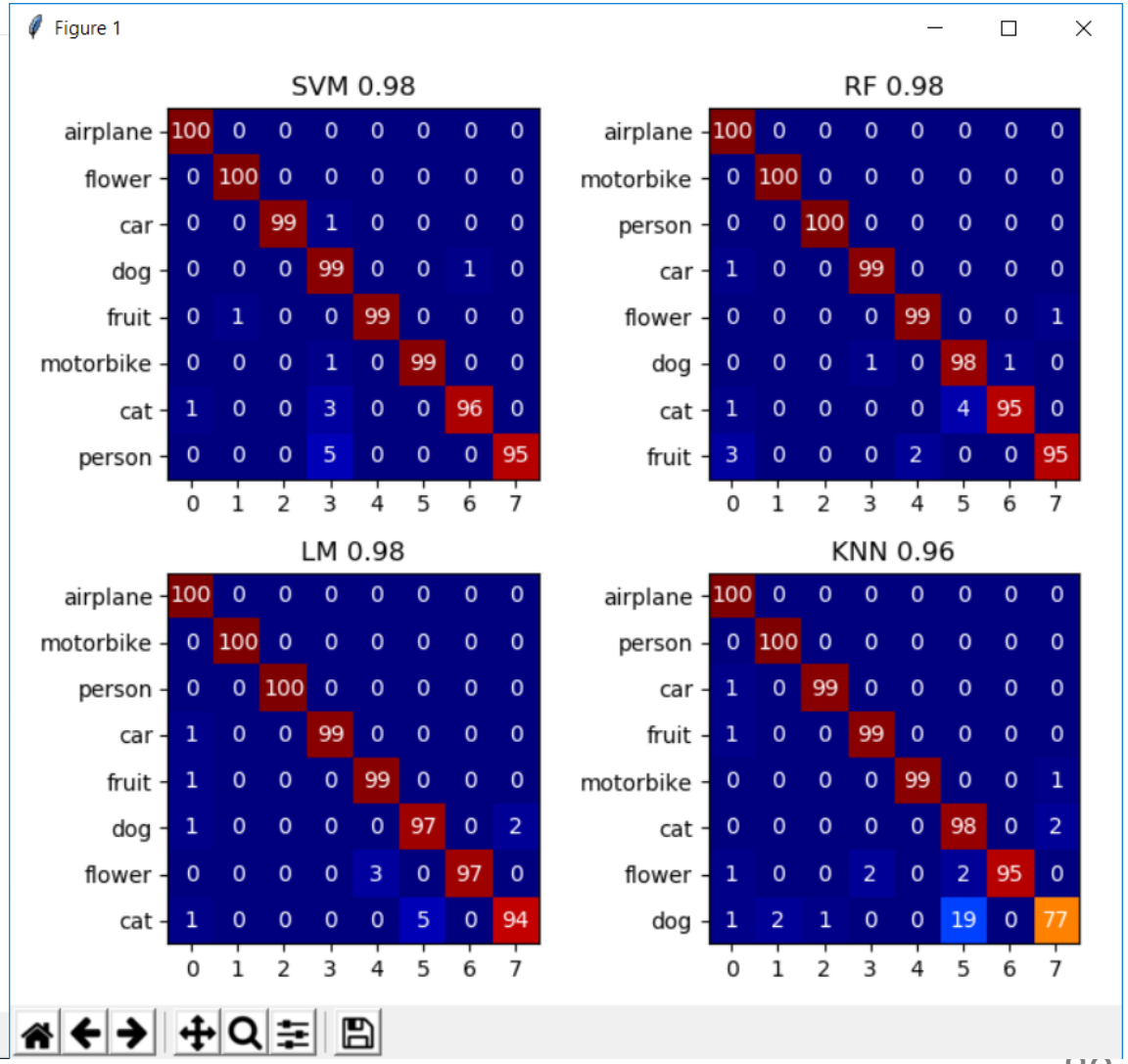




Benchmarking the CNNs

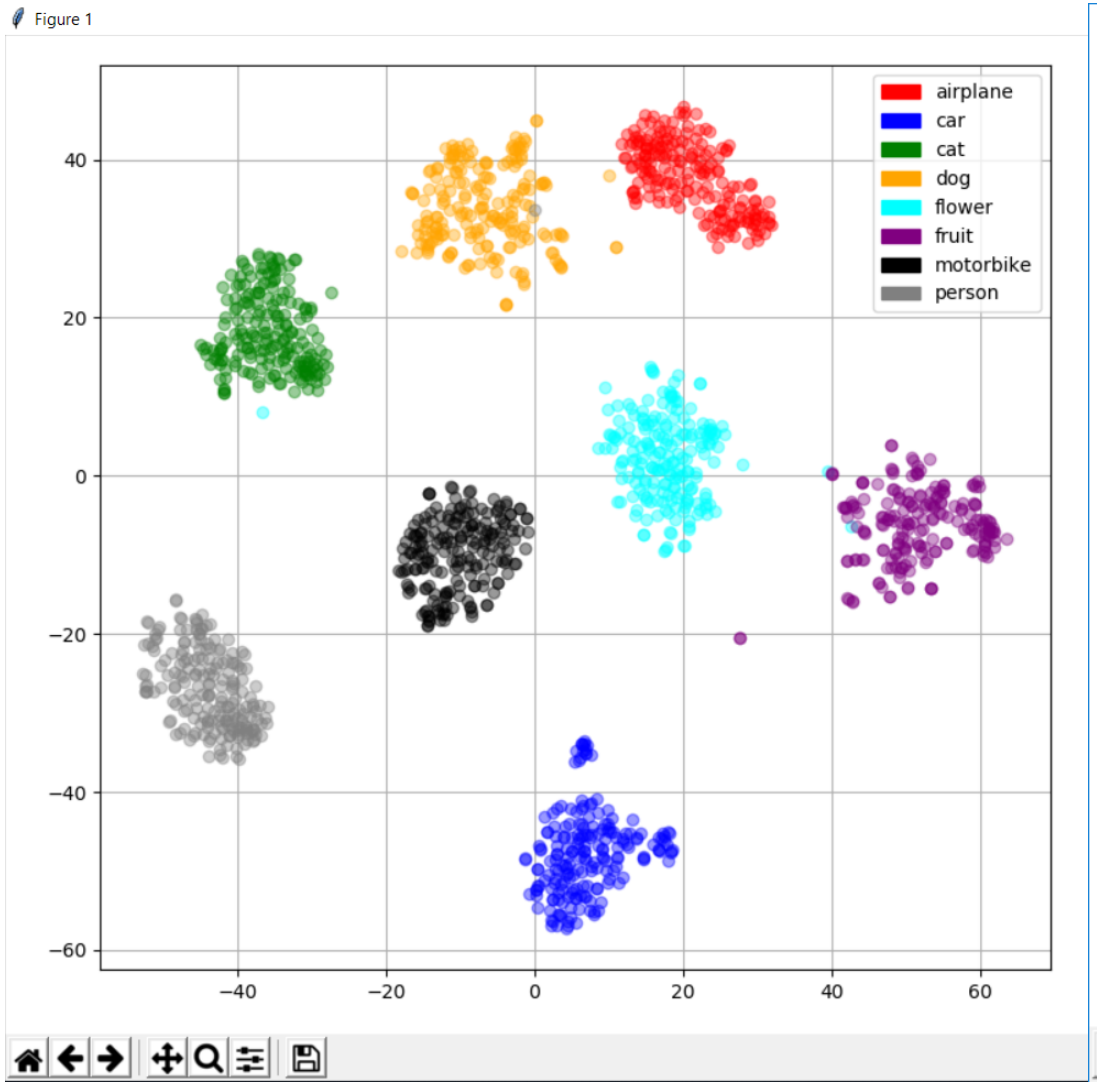
AlexNet

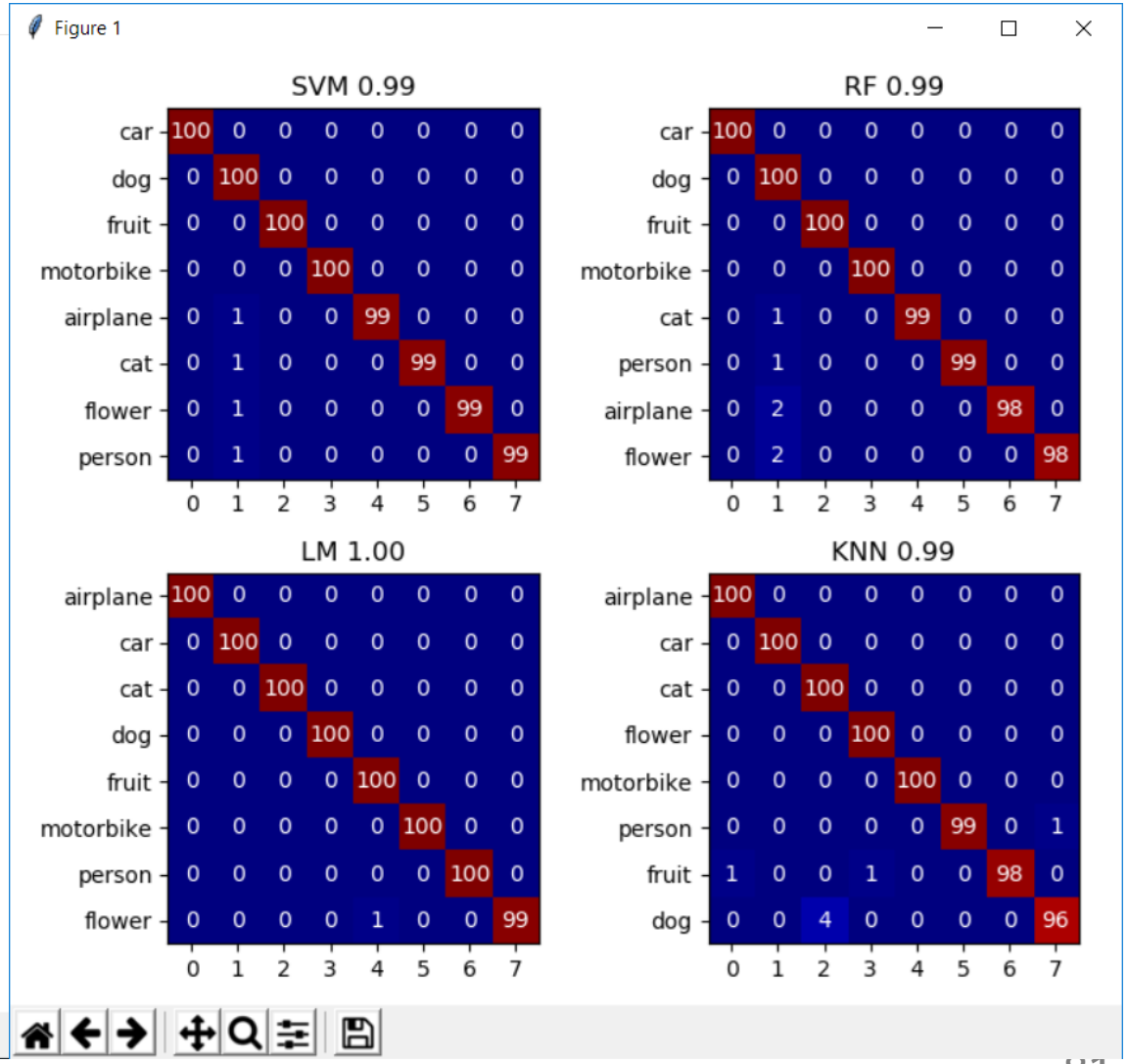




Benchmarking the CNNs

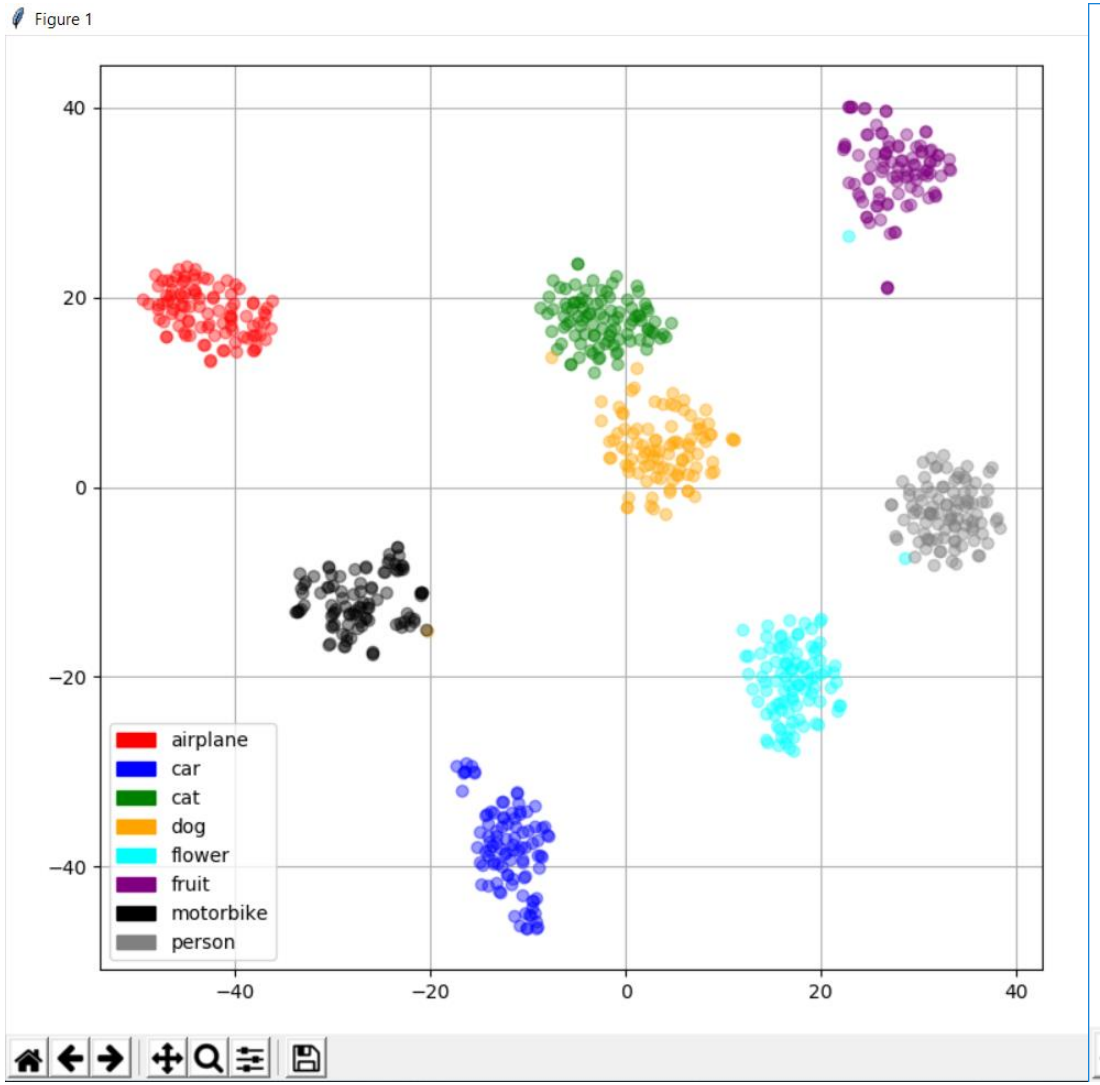
Inception

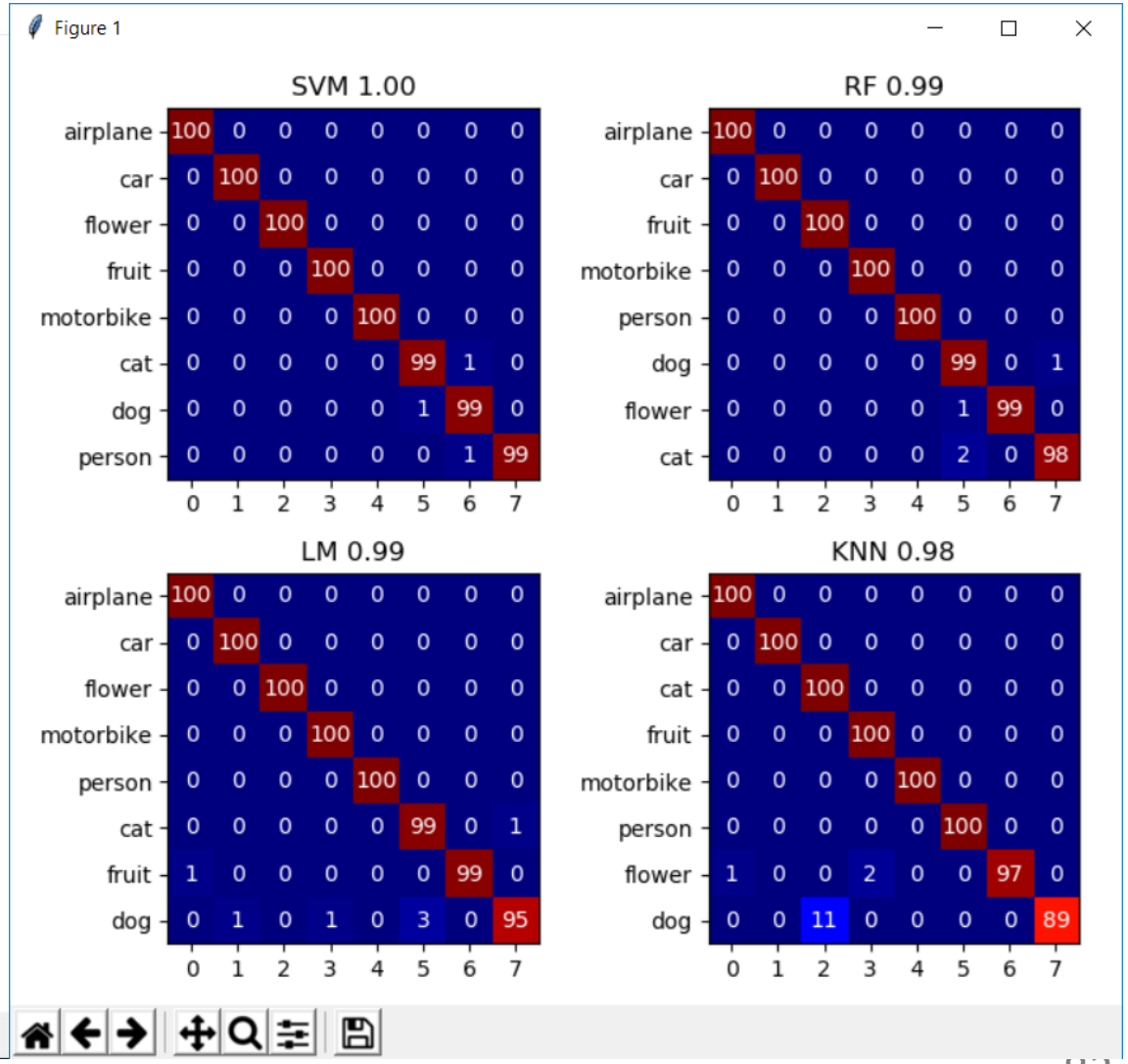




Benchmarking the CNNs

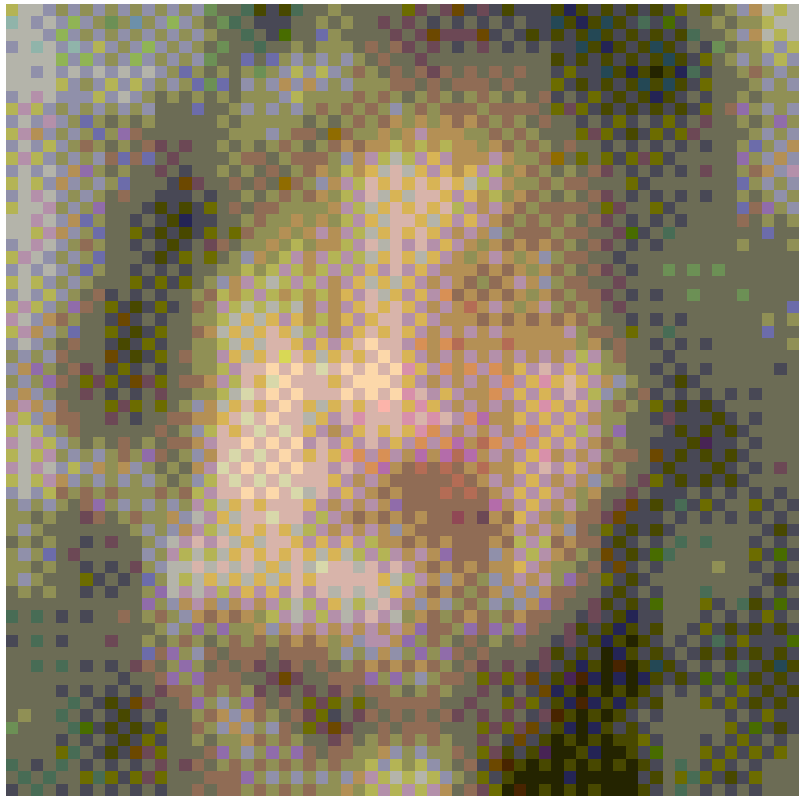
MobileNet



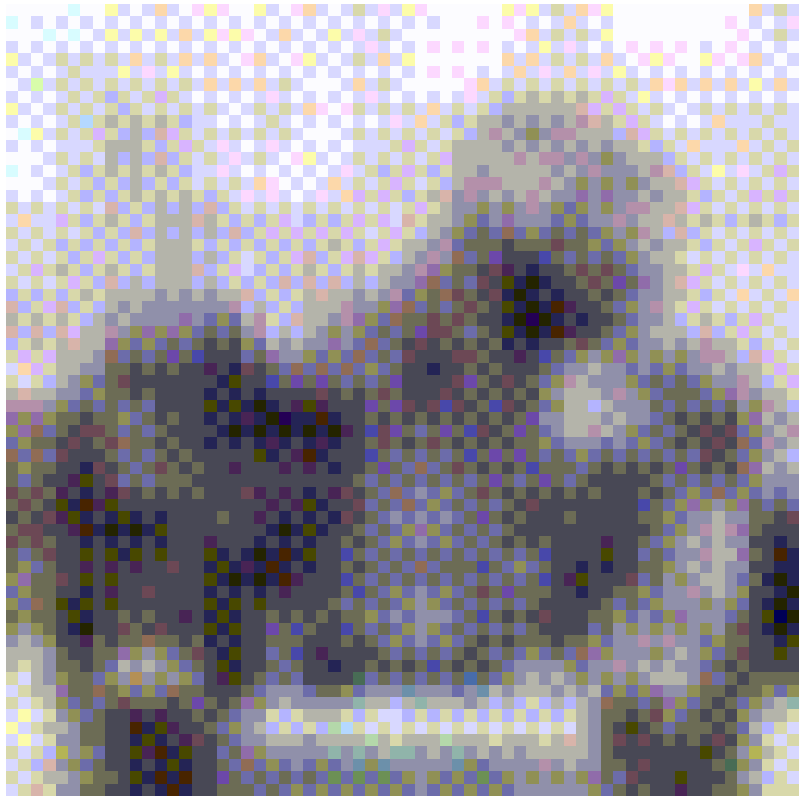


Auto-encoders

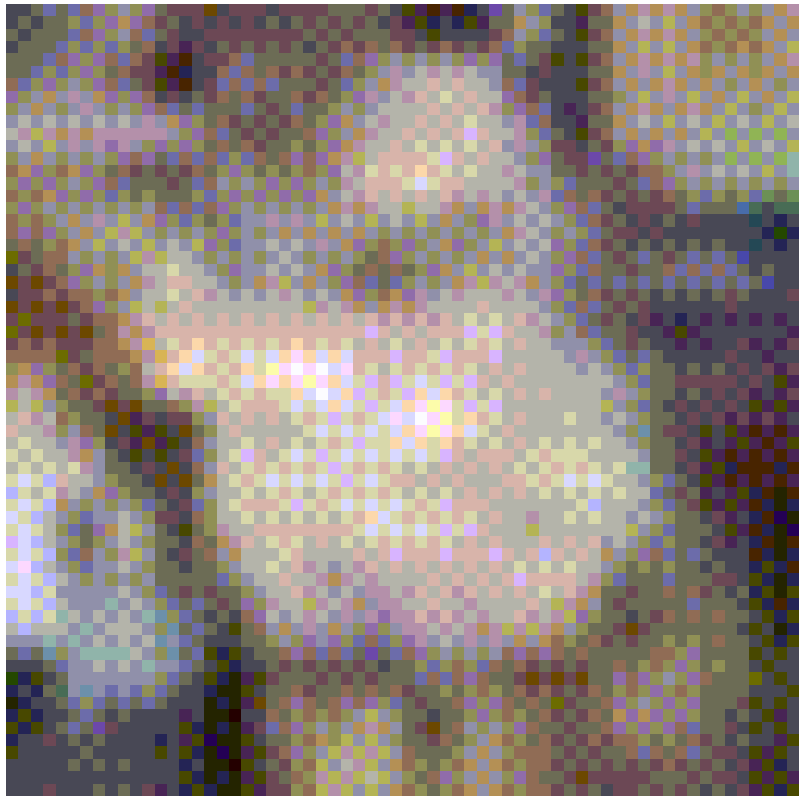
Reconstruct features back into images



Reconstruct features back into images

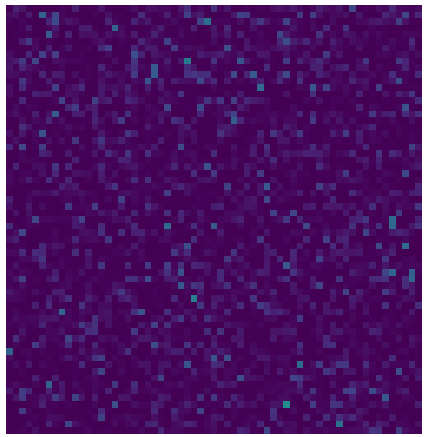


Reconstruct features back into images

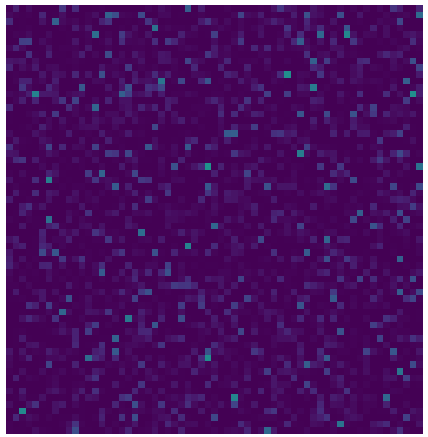


Reconstruct features back into images

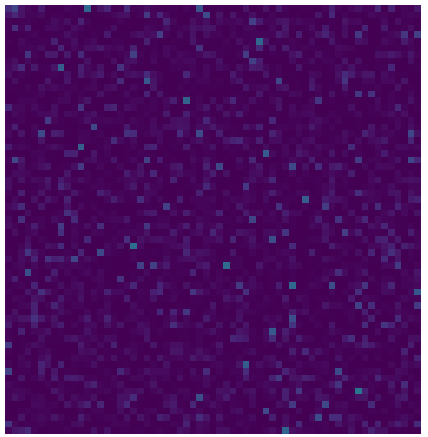
Features



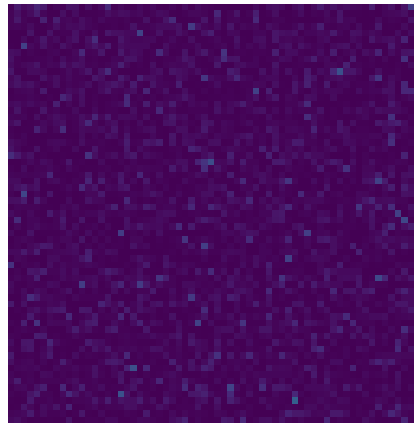
airplane



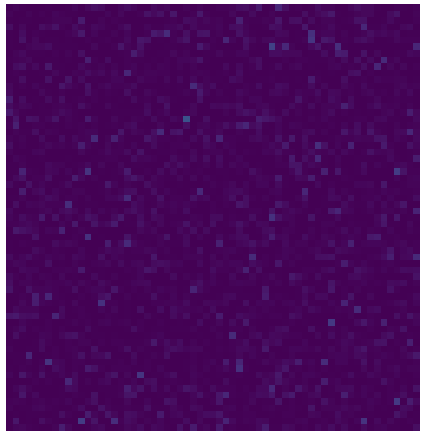
car



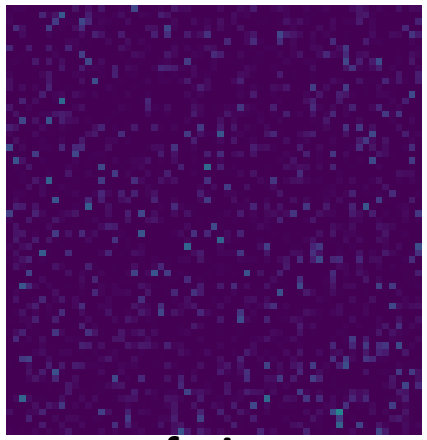
cat



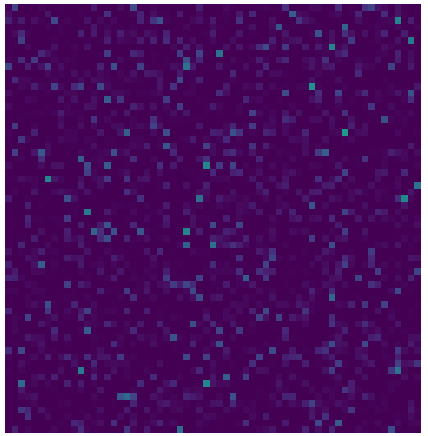
dog



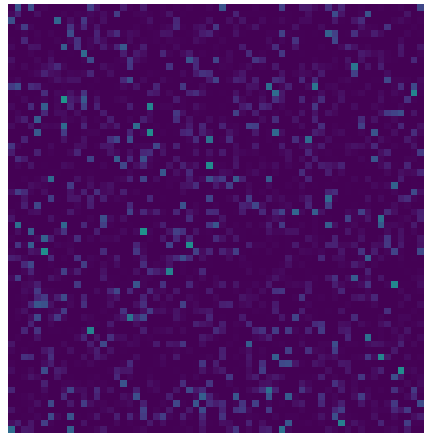
flower



fruit



motorbike



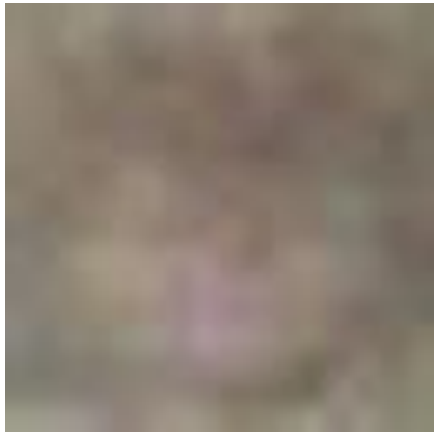
person

Reconstruct features back into images

Reconstructed

airplane

car



cat

dog

flower

fruit

motorbike

person

Reconstruct features back into images

Original

airplane

car

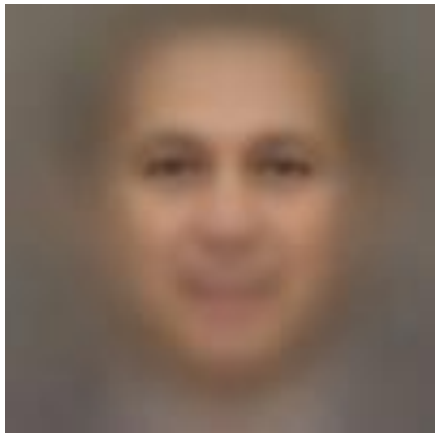
cat

dog

flower

fruit

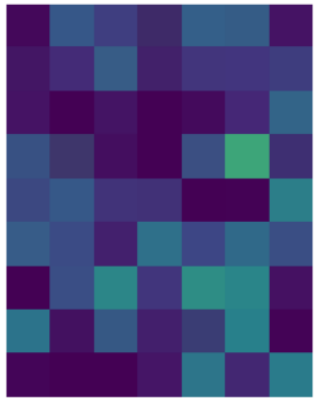
motorbike



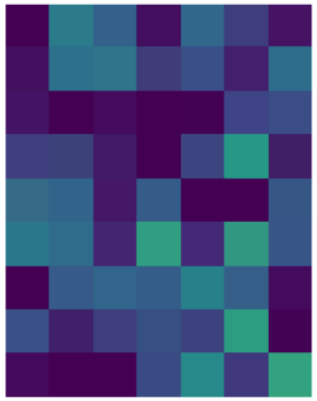
person

Reconstruct features back into images

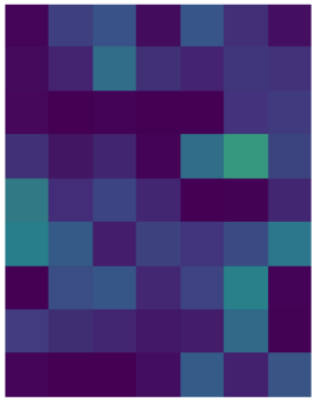
Features



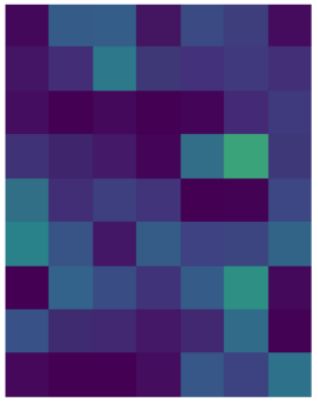
airplane



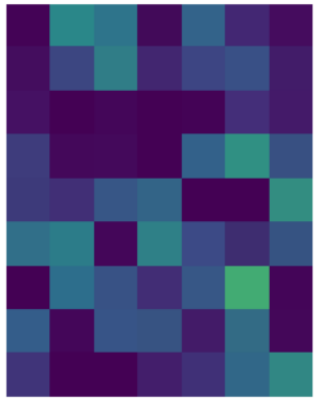
car



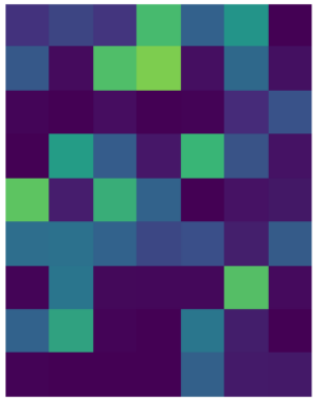
cat



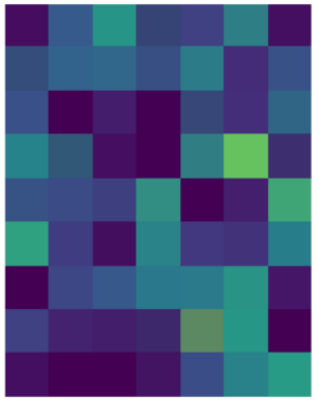
dog



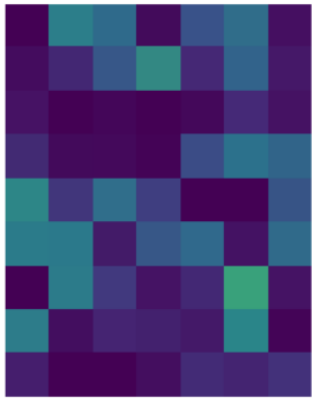
flower



fruit



motorbike



person

Reconstruct features back into images

Reconstructed


airplane


car


cat


dog

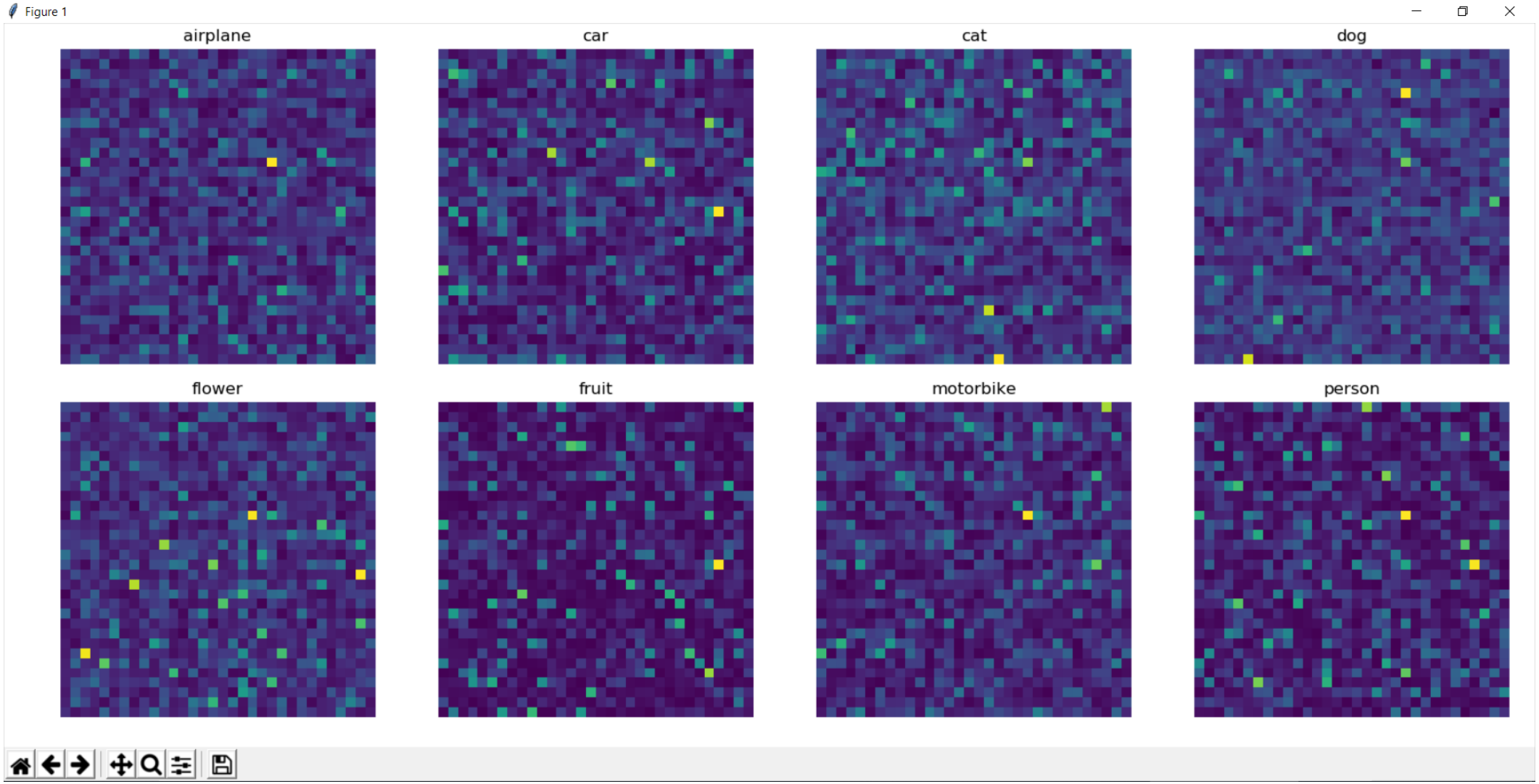

flower


fruit

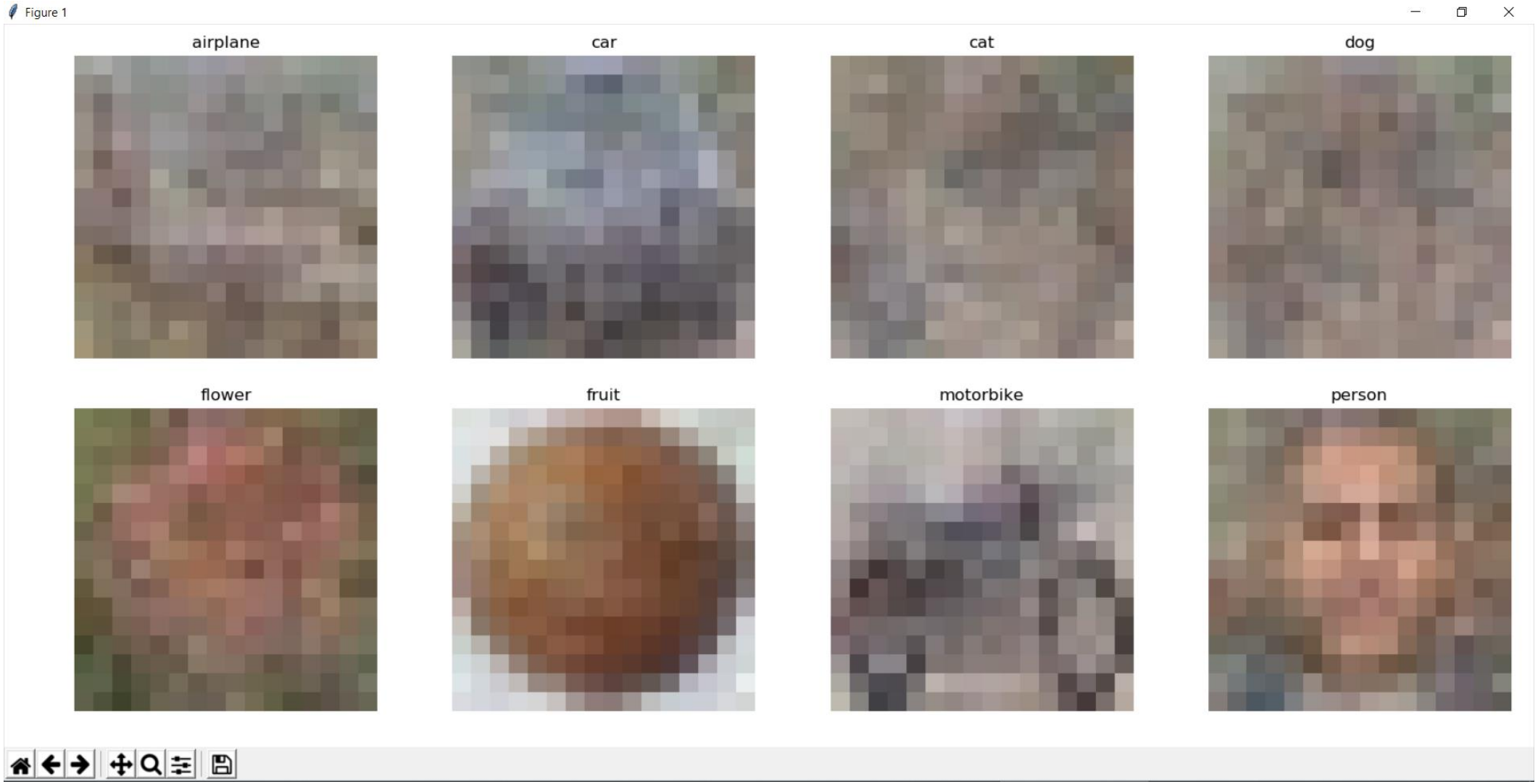

motorbike


person

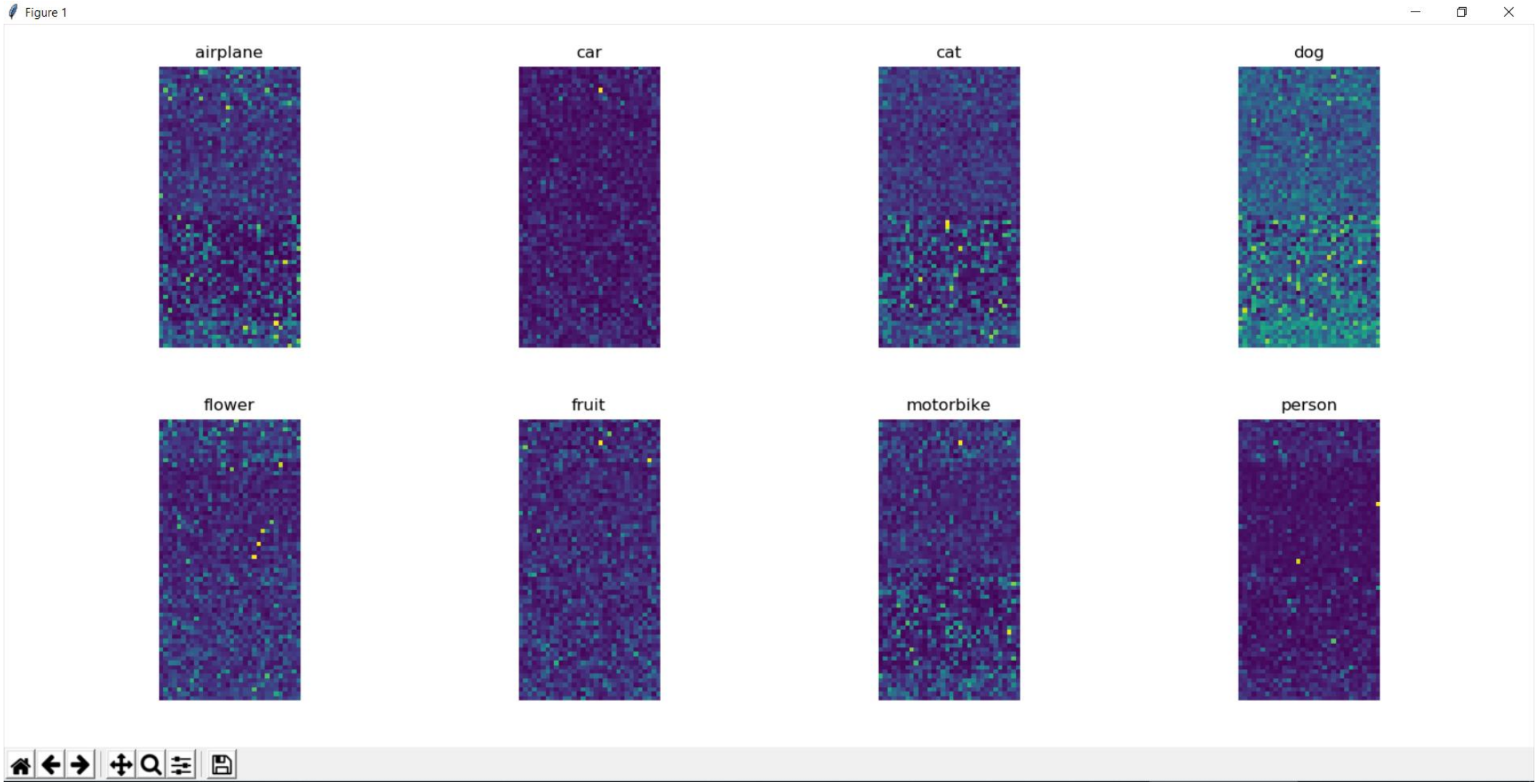
Reconstruct features back into images



Reconstruct features back into images



Reconstruct features back into images



Reconstruct features back into images

Object Detection

by Cascades

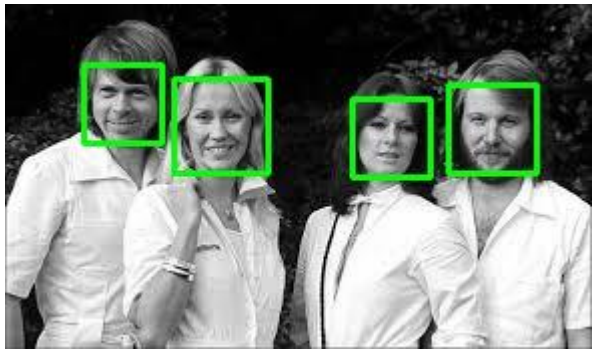
https://github.com/opencv/opencv/wiki/TensorFlow-Object-Detection-API

Object detection by Cascades

```
def demo_cascade():

    if USE_CAMERA:
        cap = cv2.VideoCapture(0)
    else:
        cap = []
        frame = cv2.imread(filename_in)

    while (True):
        if USE_CAMERA:
            ret, frame = cap.read()

        gray_rgb = tools_image.desaturate(frame)
        faces = faceCascade.detectMultiScale(frame,scaleFactor=1.1,minNeighbors=5,minSize=(30, 30))

        for (x, y, w, h) in faces:
            cv2.rectangle(gray_rgb, (x, y), (x + w, y + h), (0, 255, 0), 2)

        cv2.imshow('frame', gray_rgb)
        key = cv2.waitKey(1)
        if key & 0xFF == 27:
            break

        if (key & 0xFF == 13) or (key & 0xFF == 32):
            cv2.imwrite(filename_out, gray_rgb)

    if USE_CAMERA:
        cap.release()

    cv2.destroyAllWindows()
    if not USE_CAMERA:
        cv2.imwrite(filename_out, gray_rgb)

    return
```

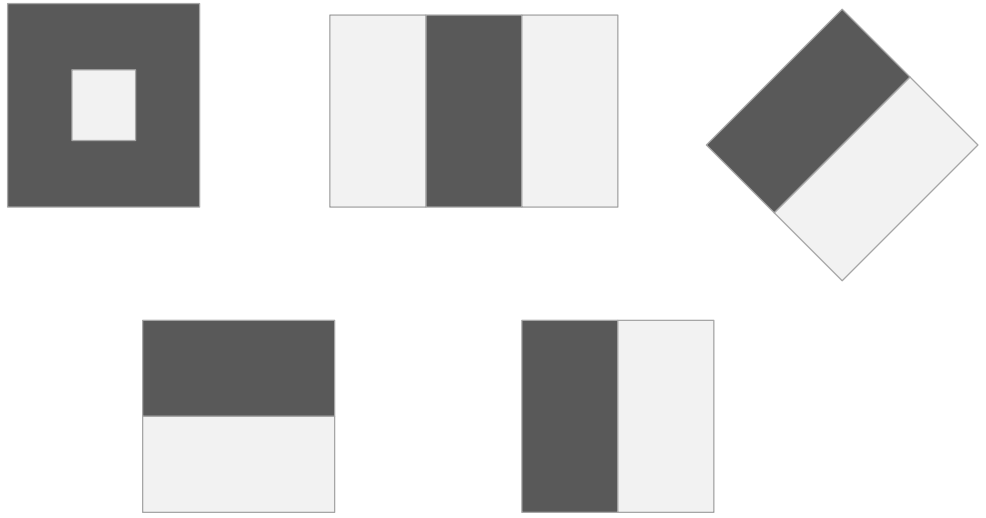
Object detection by Cascades

haarcascade_eye.xml
haarcascade_eye_tree_eyeglasses.xml
haarcascade_frontalcatface.xml
haarcascade_frontalcatface_extended.xml
haarcascade_frontalface_alt.xml
haarcascade_frontalface_alt2.xml
haarcascade_frontalface_alt_tree.xml
haarcascade_frontalface_default.xml
haarcascade_fullbody.xml
haarcascade_lefteye_2splits.xml
haarcascade_licence_plate_rus_16stages.xml
haarcascade_lowerbody.xml
haarcascade_profileface.xml
haarcascade_righteye_2splits.xml
haarcascade_russian_plate_number.xml
haarcascade_smile.xml
haarcascade_upperbody.xml



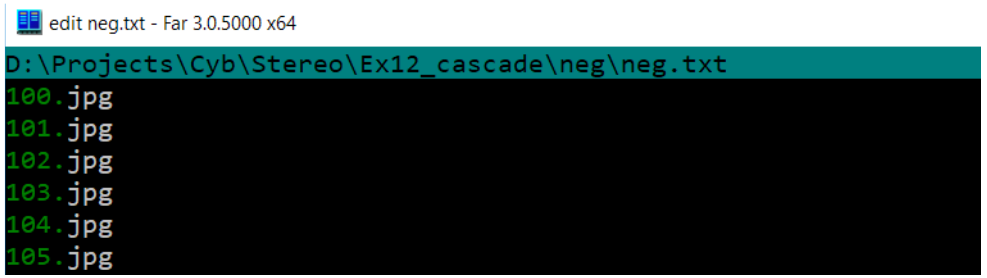
Object detection: Haar cascades

1. Create samples
2. Verify samples
3. Train the cascade



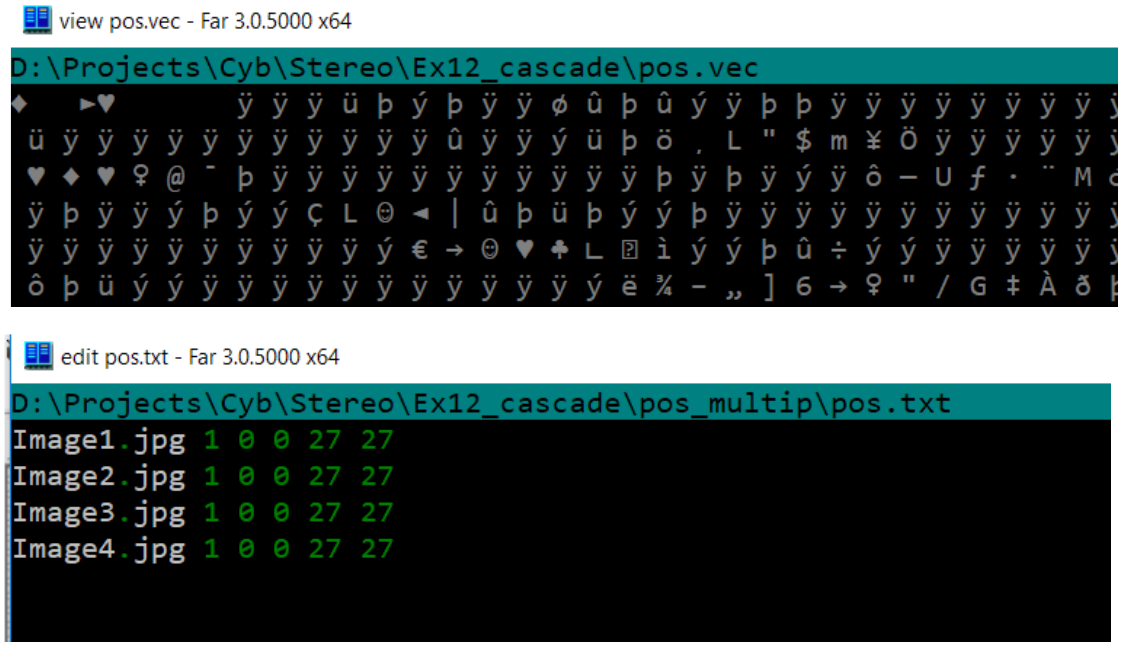
Object detection: create samples from single positive

```
opencv_createsamples   -vec pos.vec                    result vec file
                       -img pos_single\object.jpg      input positive sample file
                       -bg neg\neg.txt                 list of negative samples
                       -num 100
                       -bgcolor 0
                       -bgthresh 0
                       -w 28 -h 28
                       -maxxangle 1.1
                       -maxyangle 1.1
                       -maxzangle 0.5
                       -maxidev 40
```



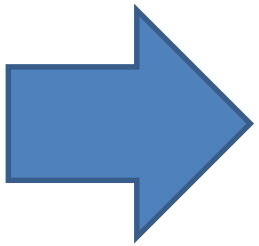
Object detection: create samples from many positive

```
opencv_createsamples   -vec pos.vec                    result file
                       -info pos_multip\pos.txt        list of pos samples
                       -num 4
                       -w 28 -h 28
```



Object detection: verify vec file

```
opencv_createsamples   -vec pos.vec
                       -show
```



Object detection: training

```
opencv_traincascade    -data output_folder
                       -vec ../positive/positives.vex
                       -bg ../negative/bg_train.txt
                       -numPos 800
                       -numNeg 400
                       -numStages 10
                       -w 20
                       -h 20
```

existing folder
made by opencv_createsamples
list of negative samples

edit bg_train.txt - Far 3.0.5000 x64

```
D:\Projects\Cyb\Stereo\Ex11_cascade\negative\bg_train.txt
../negative/bg/62.jpg
../negative/bg/22.jpg
../negative/bg/372.jpg
../negative/bg/712.jpg
../negative/bg/156.jpg
../negative/bg/416.jpg
../negative/bg/275.jpg
../negative/bg/811.jpg
../negative/bg/152.jpg
../negative/bg/451.jpg
../negative/bg/606.jpg
../negative/bg/136.jpg
../negative/bg/286.jpg
../negative/bg/116.jpg
../negative/bg/398.jpg
../negative/bg/707.jpg
```

edit cascade.xml - Far 3.0.5000 x64

```
C:\Users\dryabokon\source\Digits\ML\data\ex12_single\output_s
<?xml version="1.0"?>
<opencv_storage>
<cascade>
  <stageType>BOOST</stageType>
  <featureType>HAAR</featureType>
  <height>40</height>
  <width>40</width>
  <stageParams>
    <boostType>GAB</boostType>
    <minHitRate>9.9500000476837158e-01</minHitRate>
    <maxFalseAlarm>5.0000000000000000e-01</maxFalseAlarm>
    <weightTrimRate>9.4999999999999996e-01</weightTrimRate>
    <maxDepth>1</maxDepth>
    <maxWeakCount>100</maxWeakCount></stageParams>
  <featureParams>
    <maxCatCount>0</maxCatCount>
    <featSize>1</featSize>
    <mode>BASIC</mode></featureParams>
```

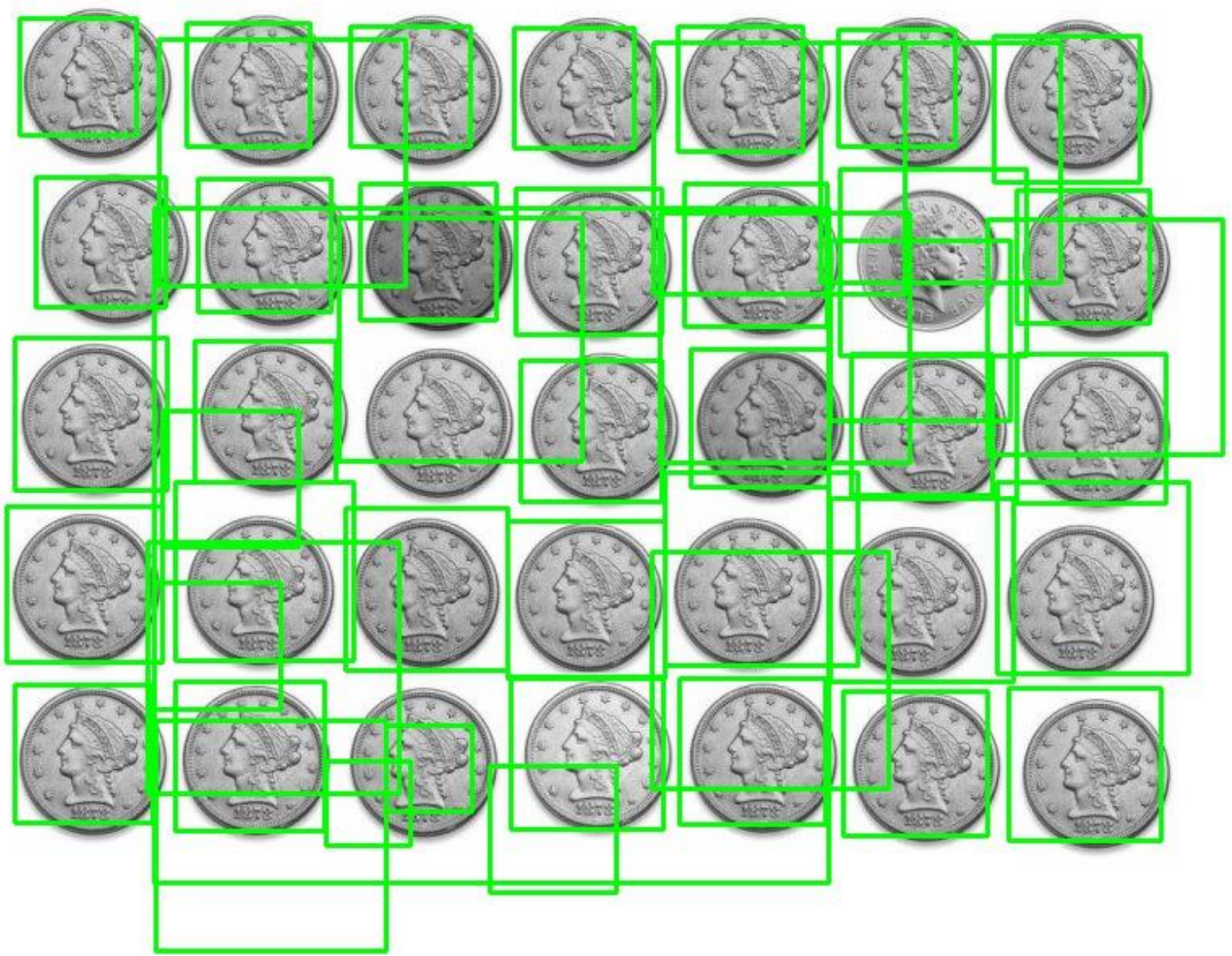
view positives.vex - Far 3.0.5000 x64

```
D:\Projects\Cyb\Stereo\Ex11_cascade\positive\positives.vex
0000000000: 33 03 00 00 90 01 00 00 | 00 00 00 00 00 98 00 9D  3♥  ▯☺        ˜ ▯
0000000010: 00 93 00 7B 00 84 00 93 | 00 C0 00 B7 00 AF 00 9B   “ { „ “ À · ¯ ›
0000000020: 00 91 00 8D 00 79 00 75 | 00 80 00 9E 00 DA 00 D7   ‘ ▯ y u € ž Ú ×
0000000030: 00 75 00 72 00 9A 00 95 | 00 8F 00 8B 00 9E 00 98   u r š • ▯ ‹ ž ˜
0000000040: 00 A6 00 A2 00 A3 00 99 | 00 A3 00 91 00 8F 00 90   ¦ ¢ £ ™ £ ‘ ▯ ▯
0000000050: 00 9C 00 AE 00 CB 00 BC | 00 66 00 7E 00 60 00 68   œ ® Ë ¼ f ~ ` h
0000000060: 00 97 00 CF 00 9B 00 80 | 00 68 00 69 00 66 00 77   — Ï › € h i f w
0000000070: 00 8B 00 78 00 5E 00 72 | 00 87 00 8B 00 B0 00 A9   ‹ x ^ r ‡ ‹ ° ©
0000000080: 00 72 00 81 00 62 00 56 | 00 A8 00 A3 00 95 00 9D   r ▯ b V ¨ £ • ▯
0000000090: 00 A3 00 88 00 8C 00 B7 | 00 B2 00 9F 00 A7 00 A0   £ ˆ Œ · ² Ÿ §
00000000A0: 00 A2 00 BF 00 D7 00 F2 | 00 ED 00 72 00 8B 00 71   ¢ ¿ × ò í r ‹ q
00000000B0: 00 CF 00 87 00 87 00 A4 | 00 C8 00 C9 00 C7 00 C5   Ï ‡ ‡ ¤ È É Ç Å
00000000C0: 00 C4 00 C3 00 C1 00 C6 | 00 C5 00 C6 00 C5 00 C6   Ä Ã Á Æ Å Æ Å Æ
00000000D0: 00 DE 00 8C 00 9B 00 84 | 00 CC 00 A9 00 95 00 82   Þ Œ › „ Ì © • ‚
00000000E0: 00 C2 00 BB 00 C6 00 B1 | 00 A9 00 97 00 A1 00 A6   Â » Æ ± © — ¡ ¦
00000000F0: 00 A9 00 AE 00 C0 00 CE | 00 DD 00 99 00 83 00 93   © ® À Î Ý ™ ƒ “
```

Train the Cascades

```
import cv2
import os
import tools_image
# ----------------------------------------------------------------------------------------------------------------
path = 'data/ex12_single/'
filename_in  = path + 'pos_single/object_to_detect.jpg'
filename_out = 'data/output/detect_result.jpg'
object_detector= cv2.CascadeClassifier(path + 'output_single/cascade.xml')

# ----------------------------------------------------------------------------------------------------------------
def train_cascade():
    os.chdir(path)
    os.system('1-create_vec_from_single.bat')#os.system('2-verify_vec_single.bat')
    os.system('3-train_from_single.bat')
    return
# ----------------------------------------------------------------------------------------------------------------


if __name__ == '__main__':

    #train_cascade()

    image = tools_image.desaturate(cv2.imread(filename_in))
    objects, rejectLevels, levelWeights = \
        object_detector.detectMultiScale3(image, scaleFactor=1.05, minSize=(20, 20), outputRejectLevels=True)
    for (x, y, w, h) in objects:cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
    cv2.imwrite(filename_out,image)
```

Object detection: example of training

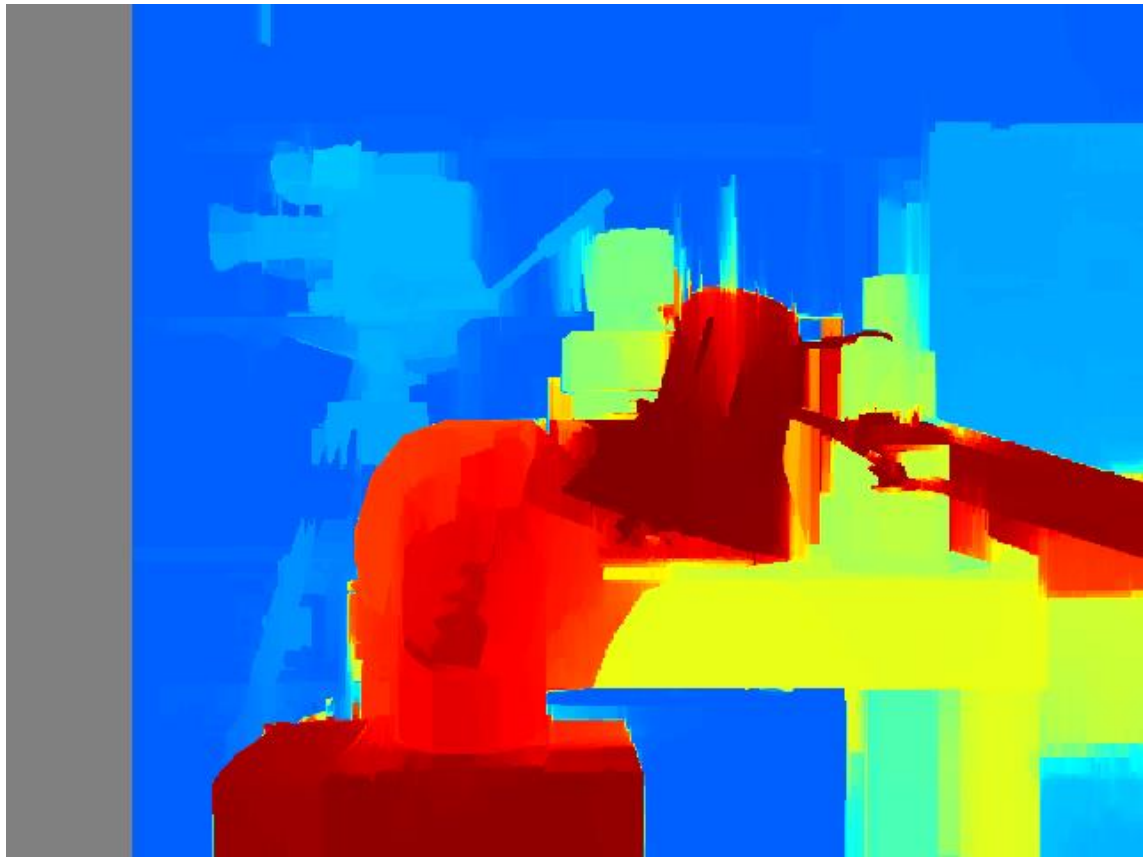


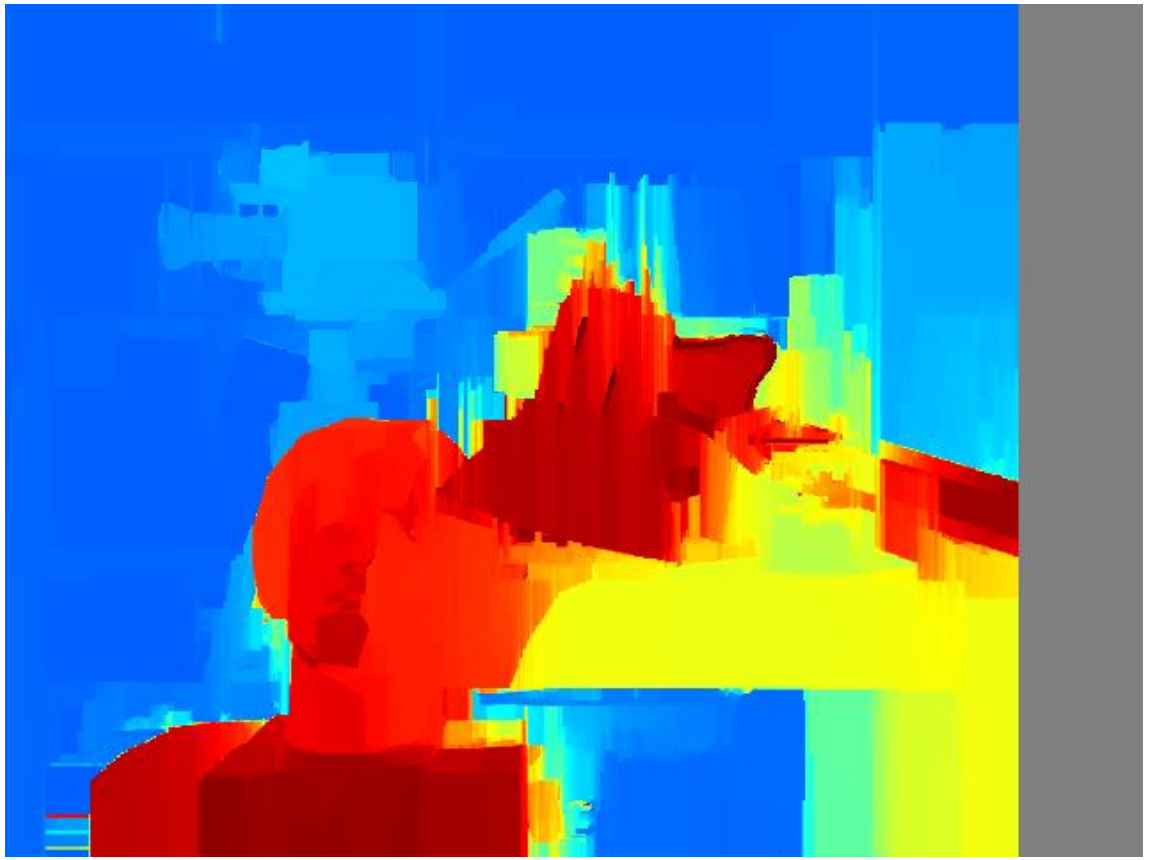
Stereo Vision

Stereo vision

Stereo vision

Stereo vision

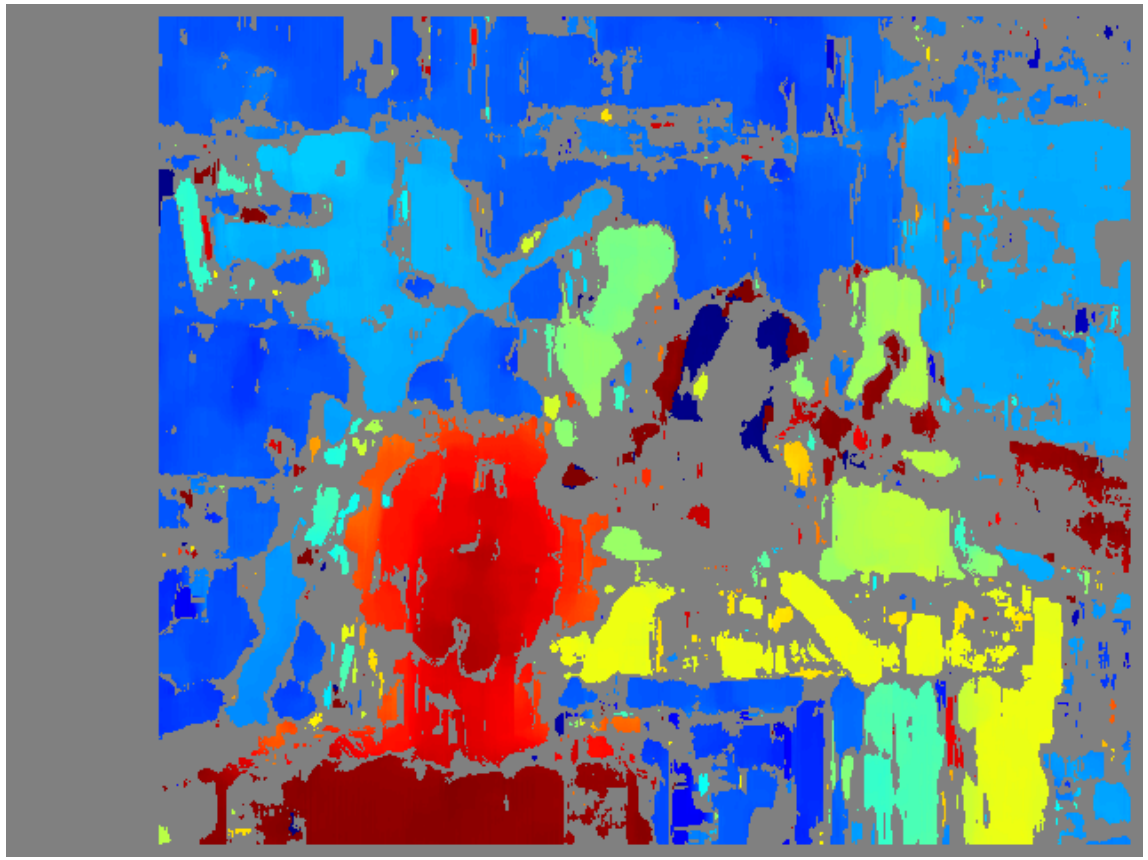


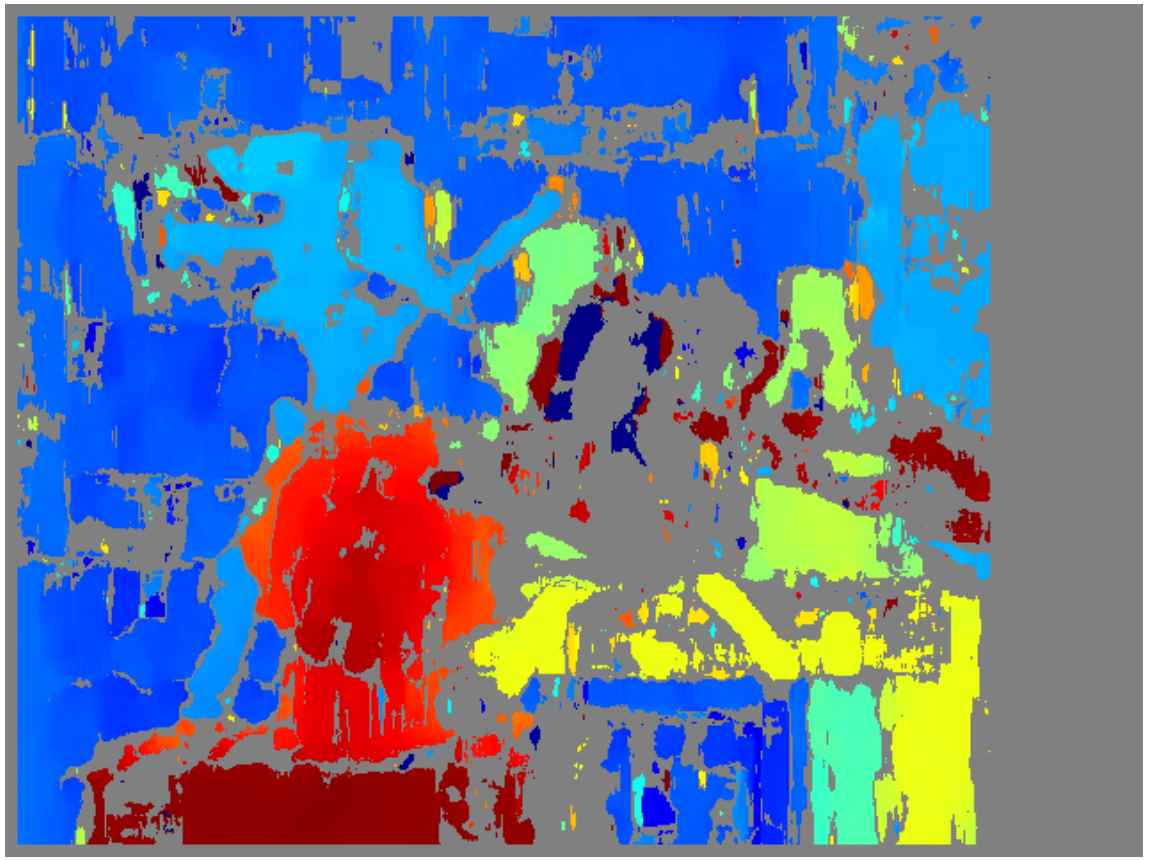


Stereo vision

```
def get_disparity_v_01(imgL, imgR, disp_v1, disp_v2, disp_h1, disp_h2):
    window_size = 7
    left_matcher = cv2.StereoSGBM_create(
        minDisparity=disp_h1,
        numDisparities=int(0 + (disp_h2 - disp_h1) / 16) * 16,
        blockSize=5,
        P1=8 * 3 * window_size ** 2,
        P2=32 * 3 * window_size ** 2,
        disp12MaxDiff=disp_h2,
        uniquenessRatio=15,
        speckleWindowSize=0,
        speckleRange=2,
        preFilterCap=63,
        mode=cv2.STEREO_SGBM_MODE_SGBM_3WAY
    )

    right_matcher = cv2.ximgproc.createRightMatcher(left_matcher)
    dispr = right_matcher.compute(imgR, imgL)
    displ = left_matcher.compute(imgL, imgR)

    wls_filter = cv2.ximgproc.createDisparityWLSFilter(matcher_left=left_matcher)
    wls_filter.setLambda(80000)
    wls_filter.setSigmaColor(1.2)

    filteredImg_L = wls_filter.filter(displ, imgL, None, dispr)
    wls_filter = cv2.ximgproc.createDisparityWLSFilter(matcher_left=right_matcher)
    filteredImg_R = wls_filter.filter(dispr, imgR, None, displ)
```

Stereo vision

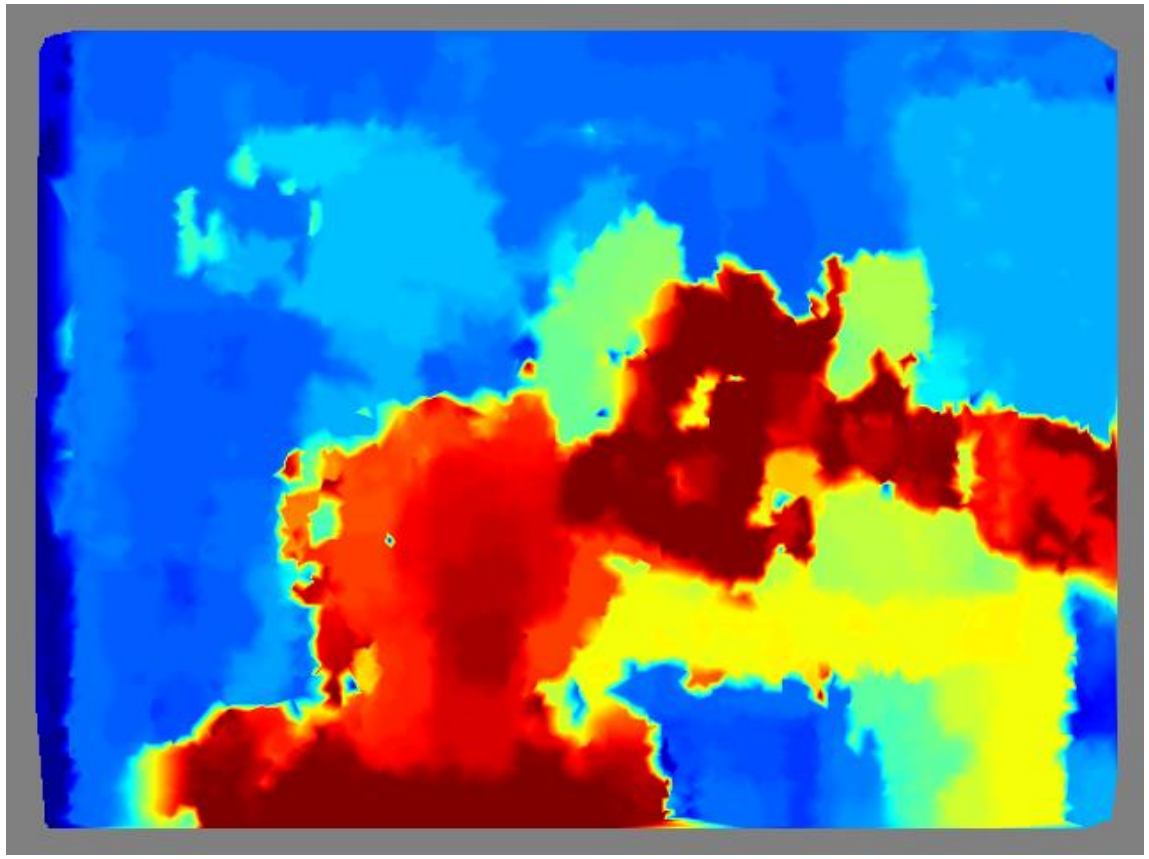


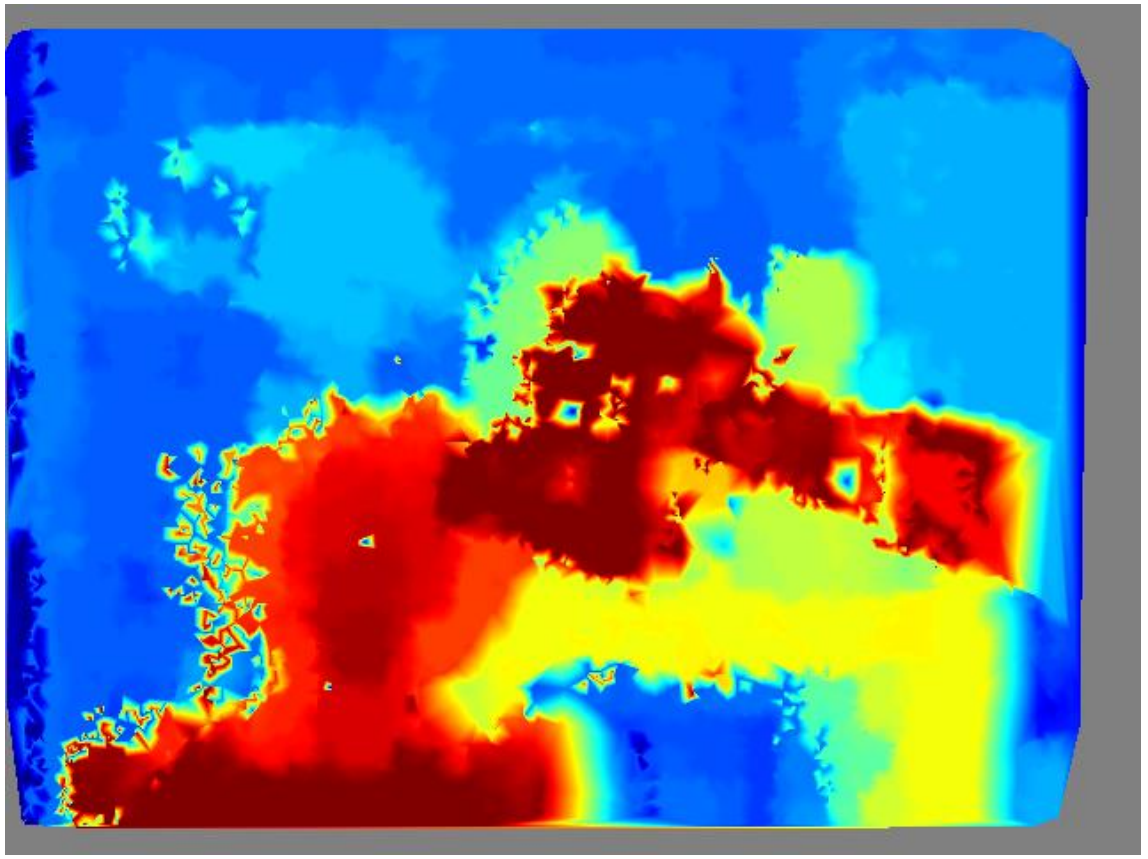


Stereo vision

```
def get_disparity_v_02(imgL, imgR, disp_v1, disp_v2, disp_h1, disp_h2):
    max = numpy.maximum(math.fabs(disp_h1), math.fabs(disp_h2))
    levels = int(1 + (max) / 16) * 16
    stereo = cv2.StereoBM_create(numDisparities=levels, blockSize=15)
    displ = stereo.compute(imgL, imgR)

    dispr = numpy.flip(stereo.compute(numpy.flip(imgR, axis=1), numpy.flip(imgL, axis=1)), axis=1)
    return -displ / 16, -dispr / 16
```

Stereo vision





Stereo vision

```
def get_best_matches(image1,image2,disp_v1, disp_v2, disp_h1, disp_h2, window_size=15,step=10):

    N = int(image1.shape[0] * image1.shape[1] / step / step)
    rand_points = numpy.random.rand(N,2)
    rand_points[:, 0] = window_size + (rand_points[:, 0]*(image1.shape[0]-2*window_size))
    rand_points[:, 1] = window_size + (rand_points[:, 1]*(image1.shape[1]-2*window_size))

    coord1,coord2,quality = [],[],[]

    for each in rand_points:
        row,col = int(each[0]),int(each[1])

        template = tools_image.crop_image(image1,row-window_size,col-window_size,row+window_size,col+window_size)
        top,left,bottom,right = row - window_size + disp_v1, col - window_size + disp_h1,row + window_size + disp_v2,col + window_size + disp_h2
        field = tools_image.crop_image(image2,top,left,bottom,right)

        q = cv2.matchTemplate(field, template, method=cv2.TM_CCOEFF_NORMED)
        q = q[1:, 1:]
        q = (q + 1) * 128

        idx = numpy.argmax(q.flatten())
        q_best = numpy.max(q.flatten())
        qq=q[int(idx/q.shape[1]), idx % q.shape[1]]

        if q_best>0:
            dr = int(idx/q.shape[1])+disp_v1
            dc = idx % q.shape[1]+disp_h1

            if col + dc>=0 and col + dc<image1.shape[1] and row + dr>=0 and row + dr<image1.shape[0]:
                coord1.append([col    ,row    ])
                coord2.append([col+ dc, row+dr])
                quality.append(q_best)

    return numpy.array(coord1), numpy.array(coord2),numpy.array(quality)
```

Part 2: Projective Geometry

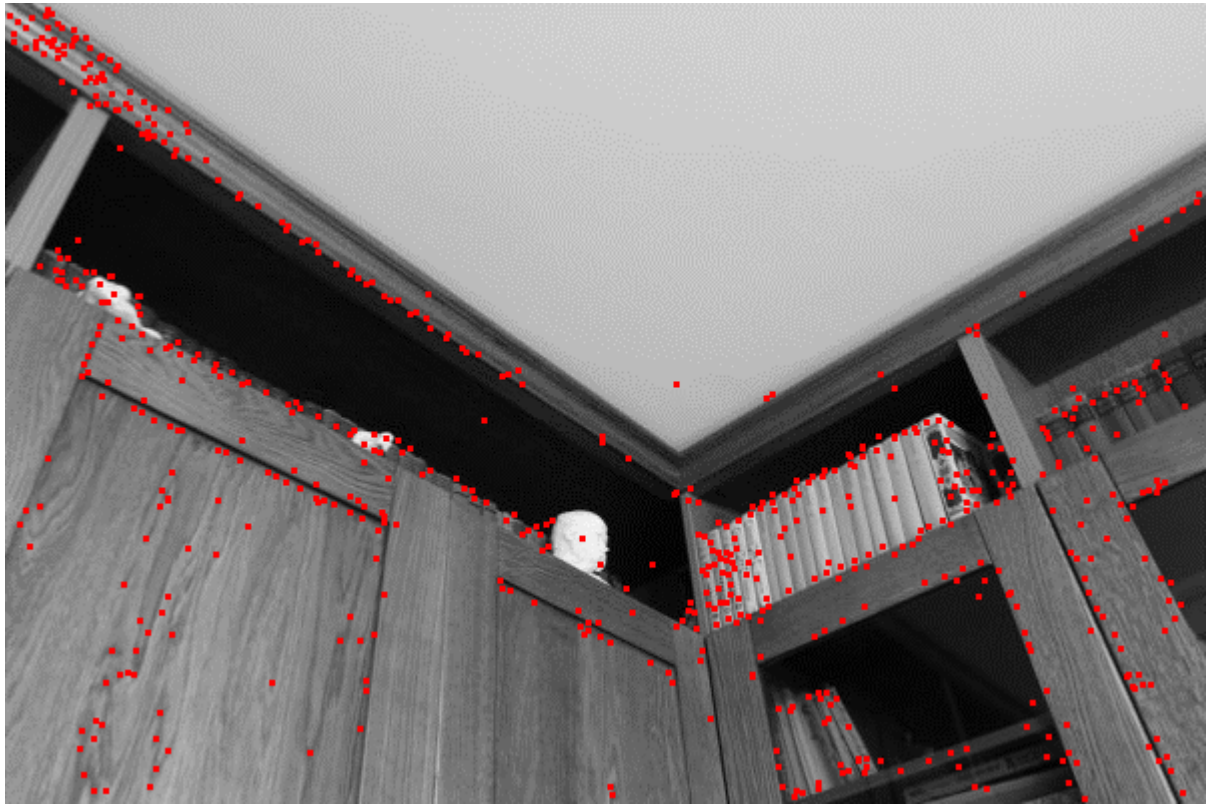
KeyPoints detection
Matching the KeyPoints
Homography
Blending
Camera calibration
Stereopair rectification
Pose estimation

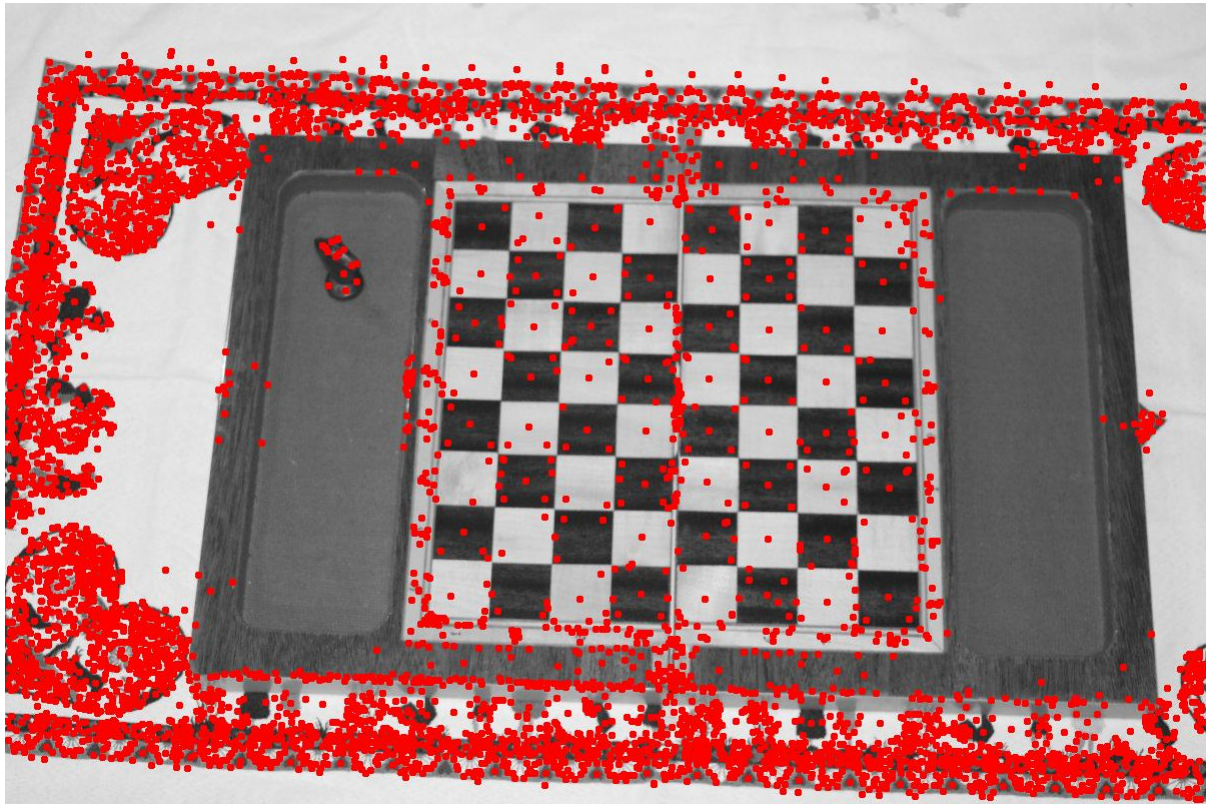
KeyPoints detection

https://github.com/dryabokon/geometry/blob/master/ex01_keypoints_detect.py

KeyPoints detection

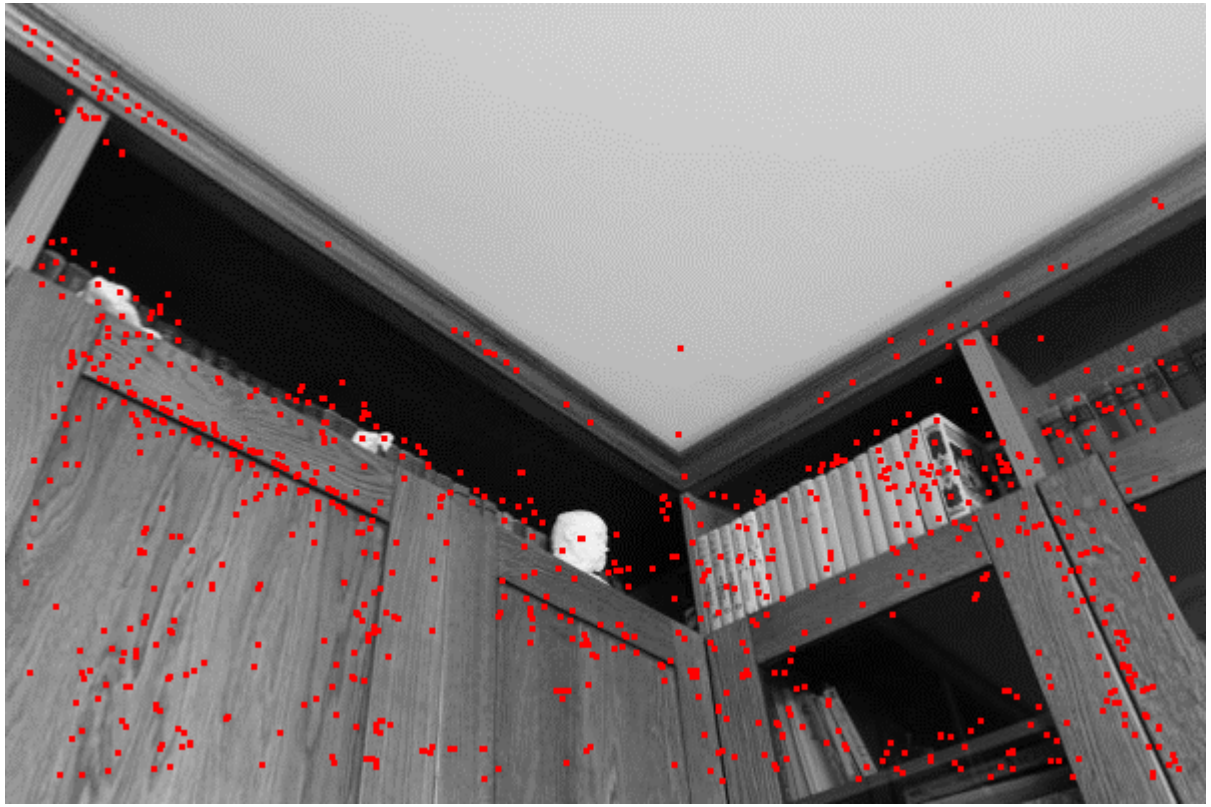
SIFT: Scale-Invariant Feature Transform

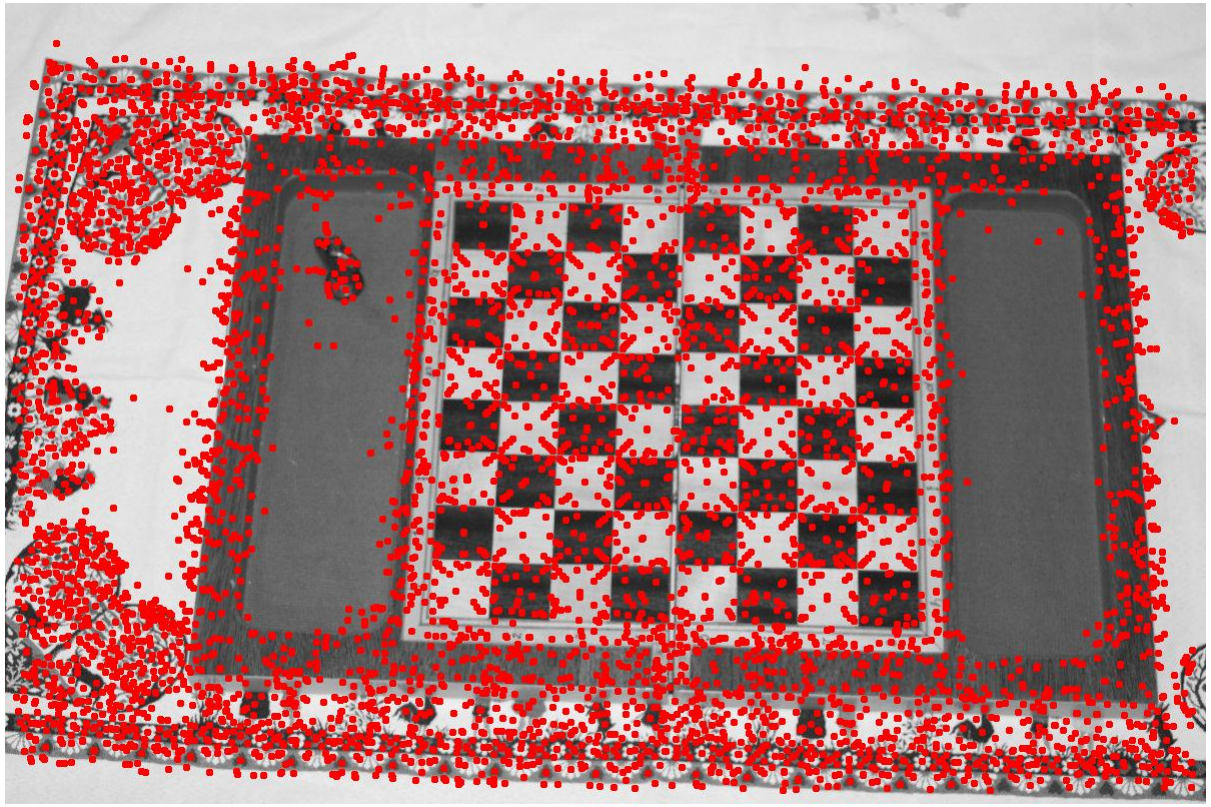




KeyPoints detection

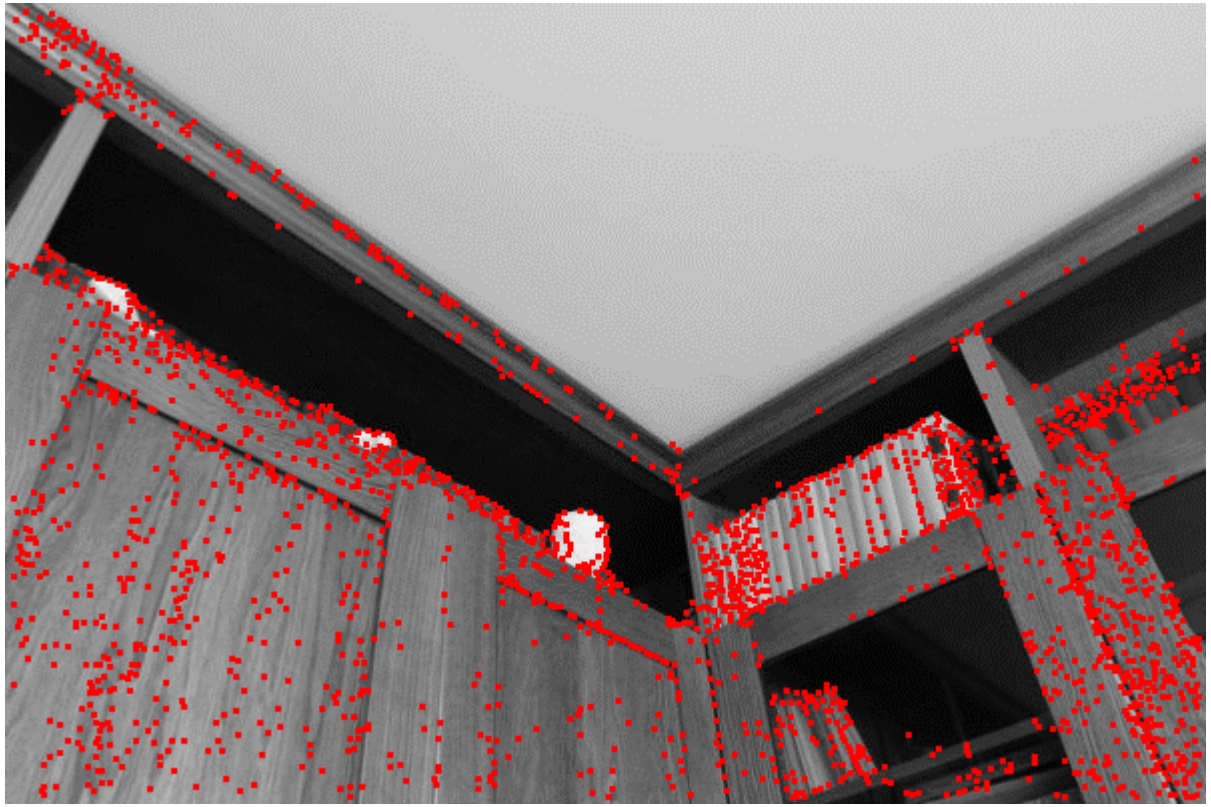
SURF: Speeded-Up Robust Features

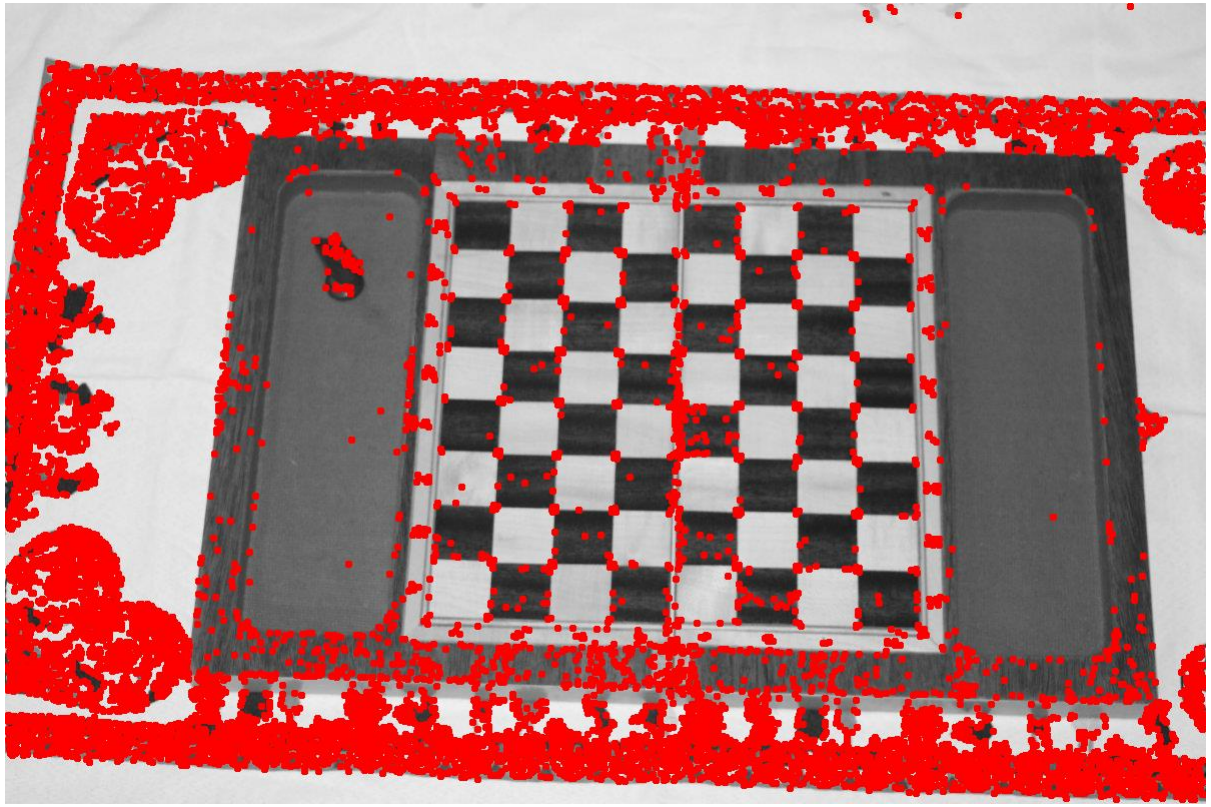




KeyPoints detection

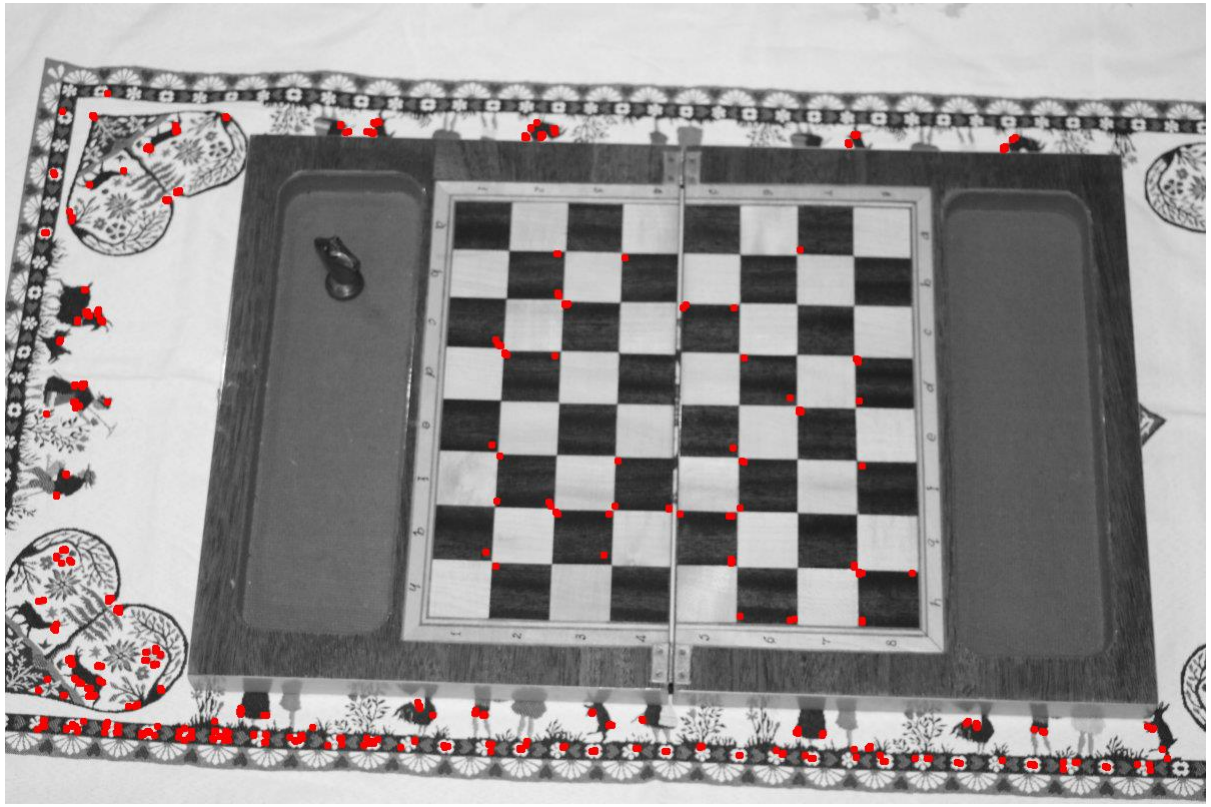
FAST: Features from Accelerated Segment Test





KeyPoints detection

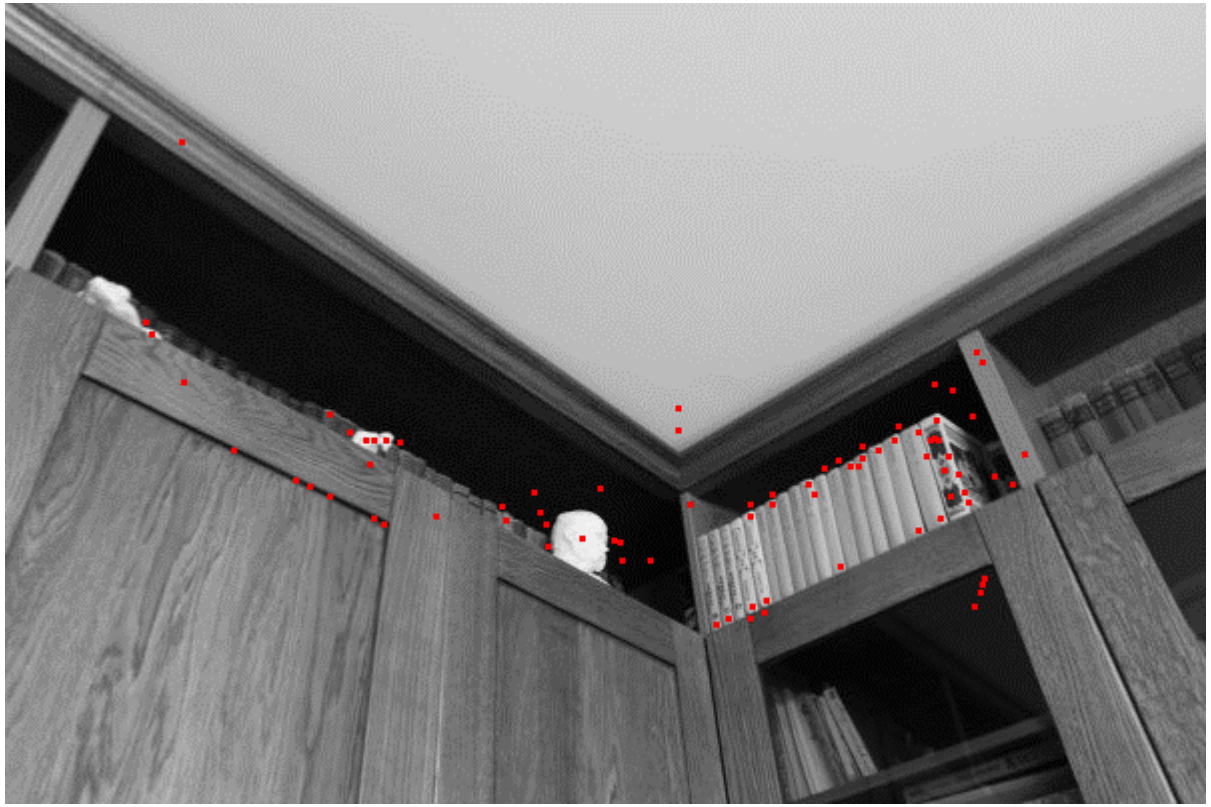
ORB: Oriented FAST and Rotated BRIEF

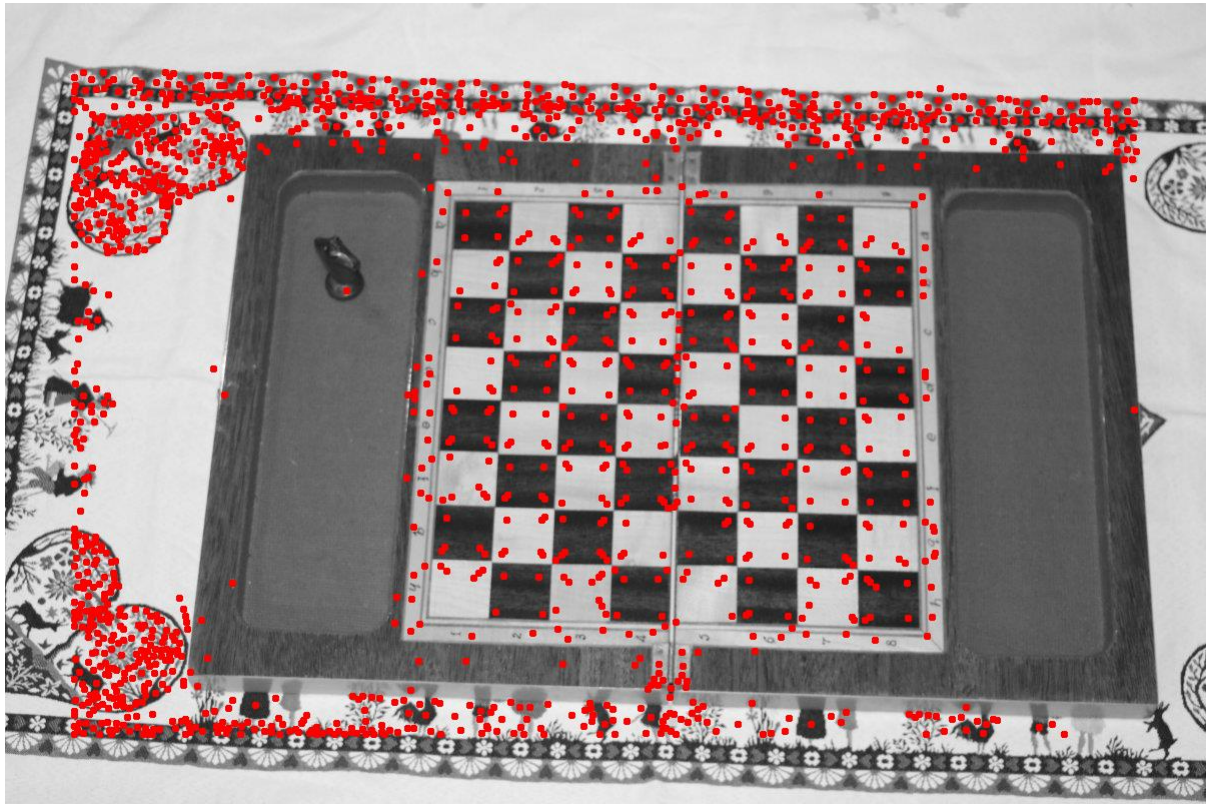


KeyPoints detection

STAR: scale-invariant **cen**ter-**surr**ound detector(CENSURE)

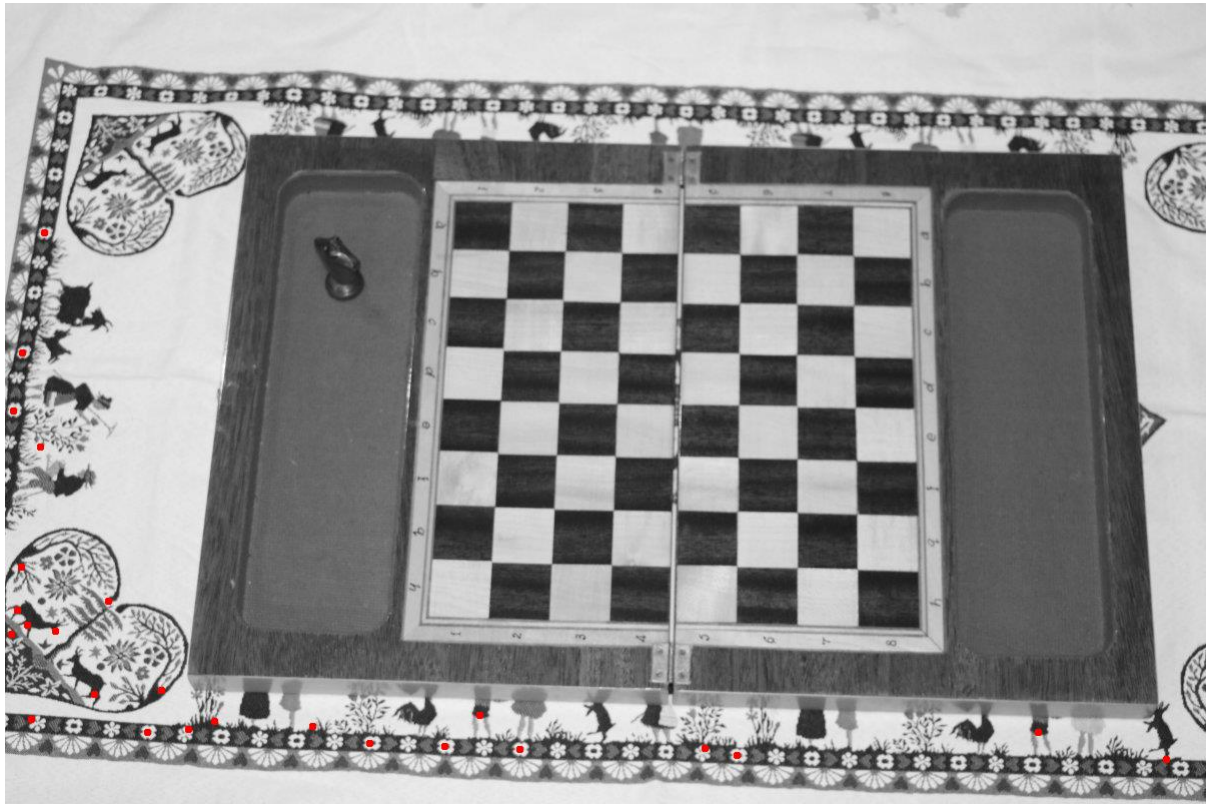
called STAR detector in OpenCV





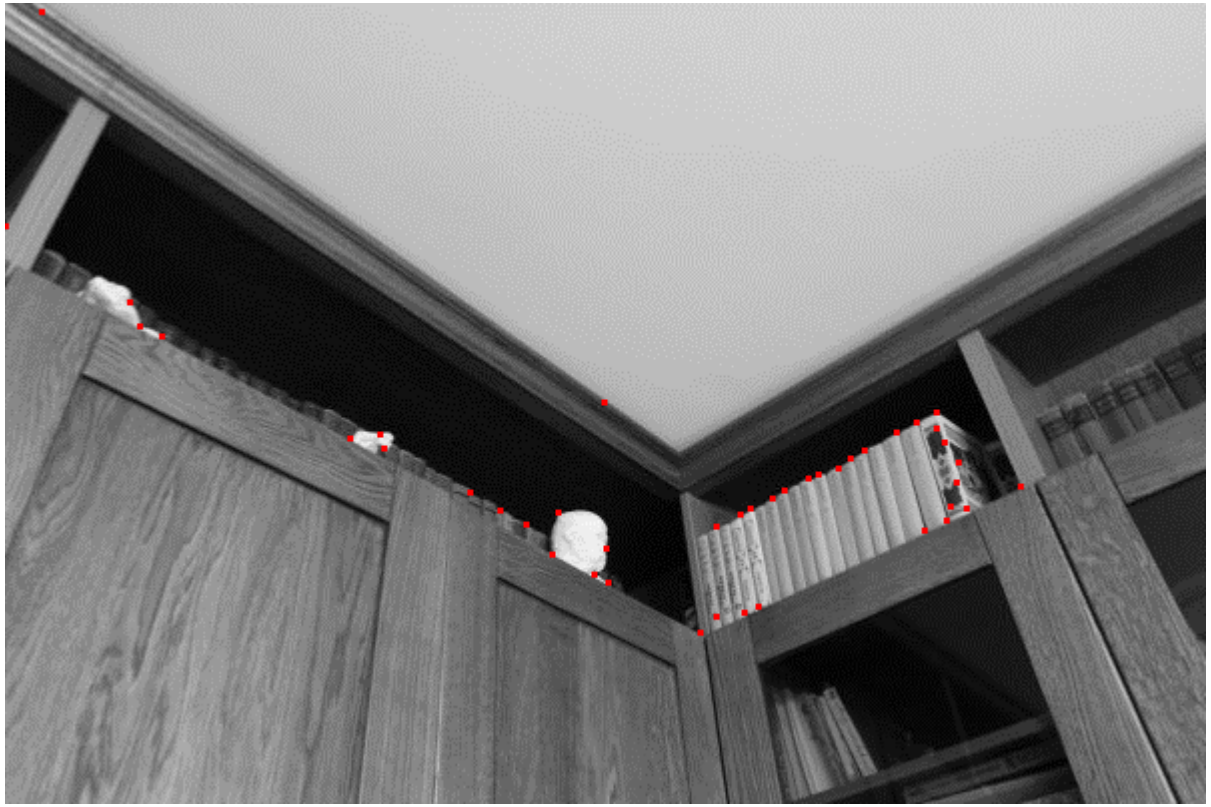
KeyPoints detection

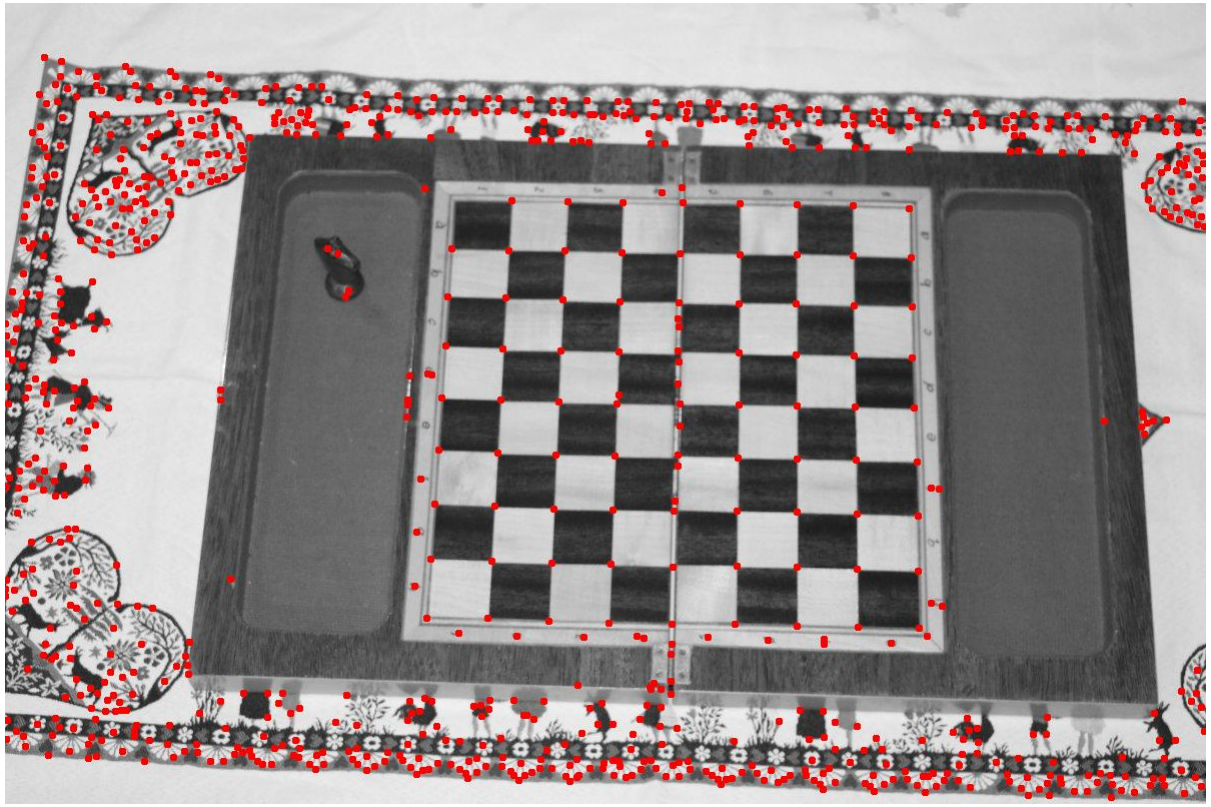
Shi-Tomasi



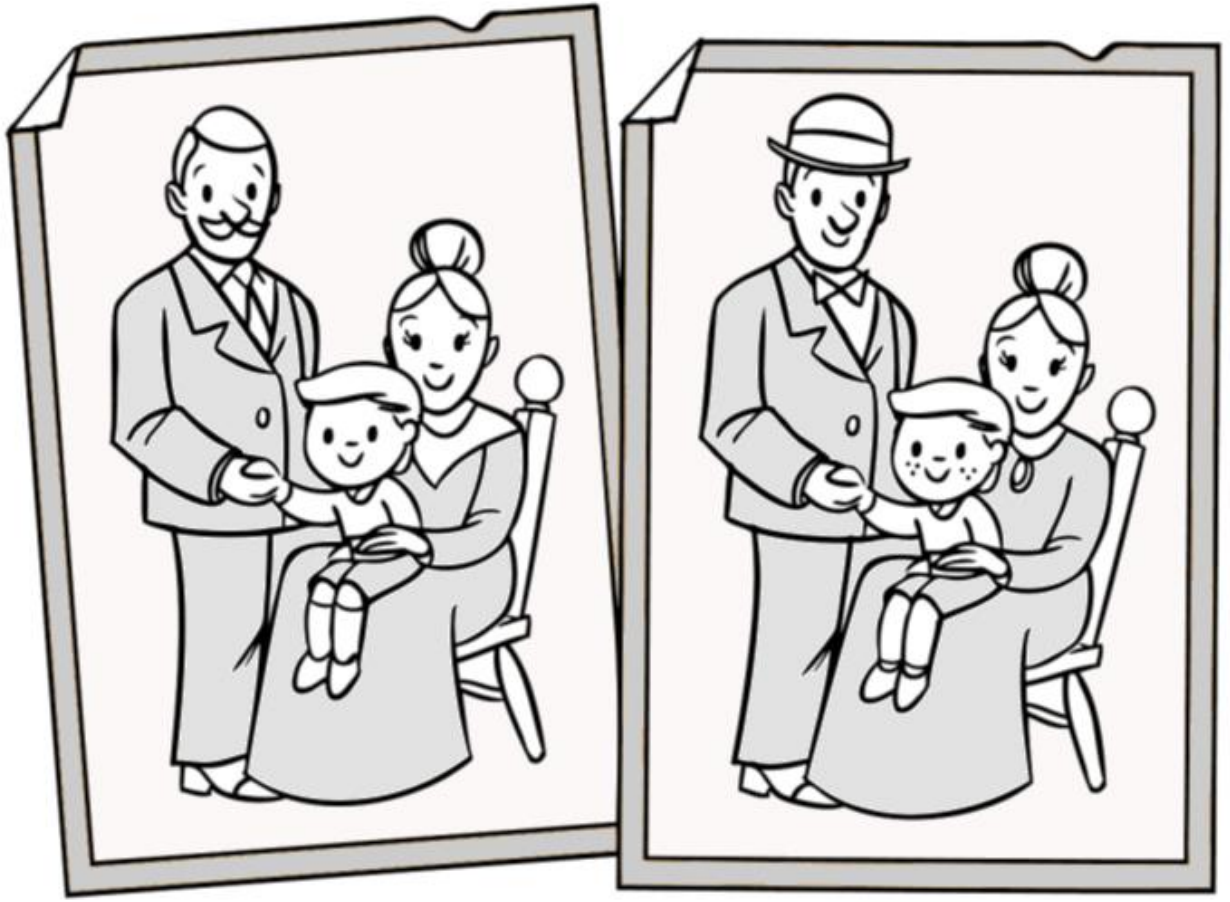
KeyPoints detection

Harris



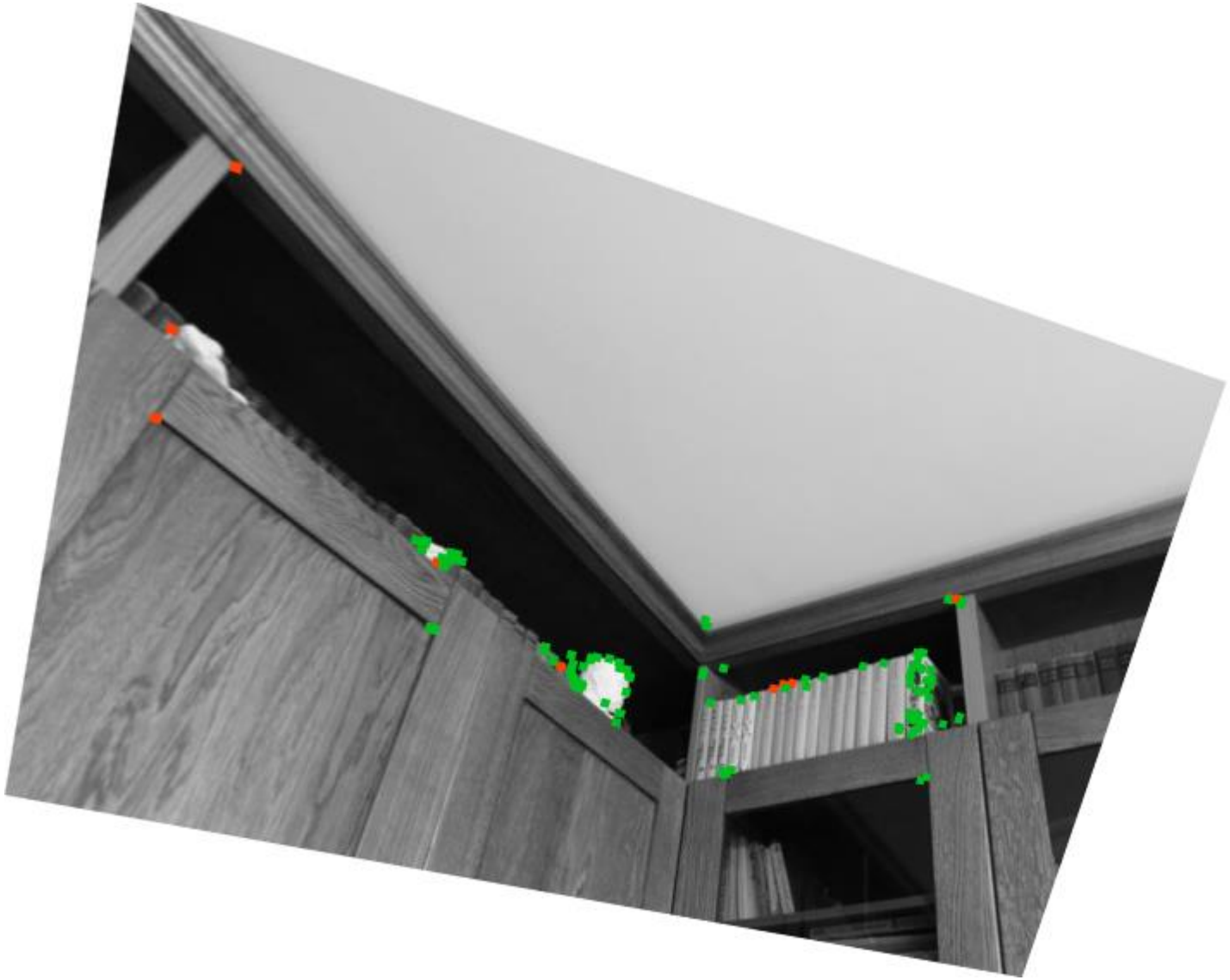


Matching the KeyPoints



https://github.com/dryabokon/geometry/blob/master/ex02_keypoints_match.py

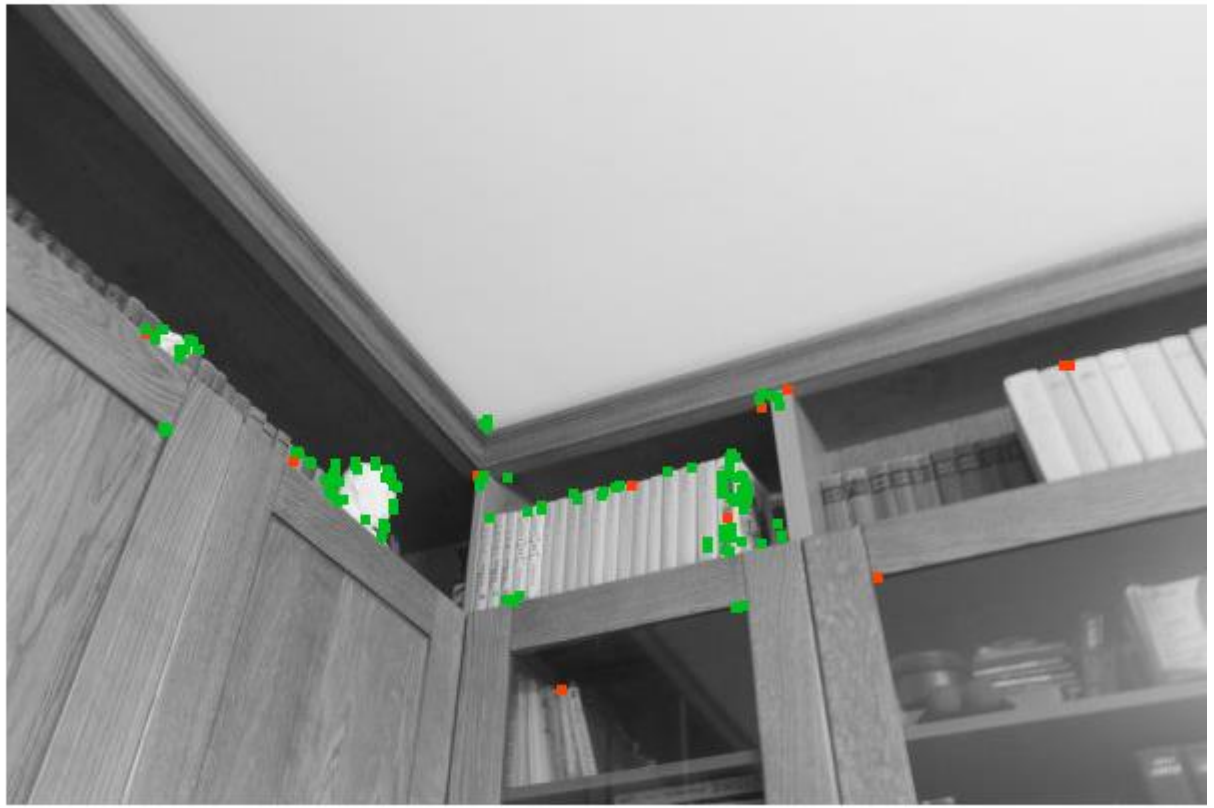
Matching the KeyPoints



Detector: ORB
Matcher: FLANN

True Positive
False Positive

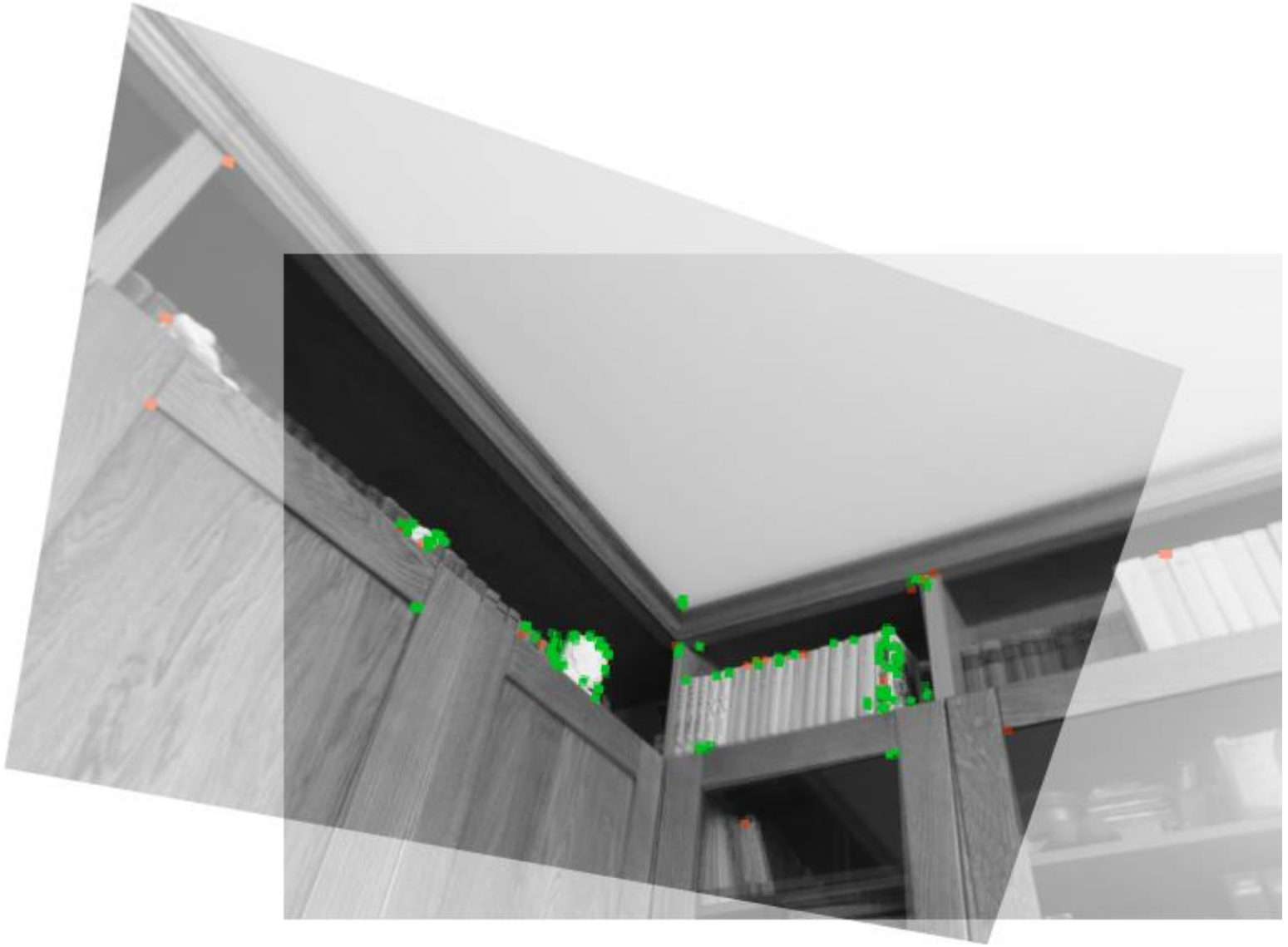
Matching the KeyPoints



Detector: ORB
Matcher: FLANN

True Positive
False Positive

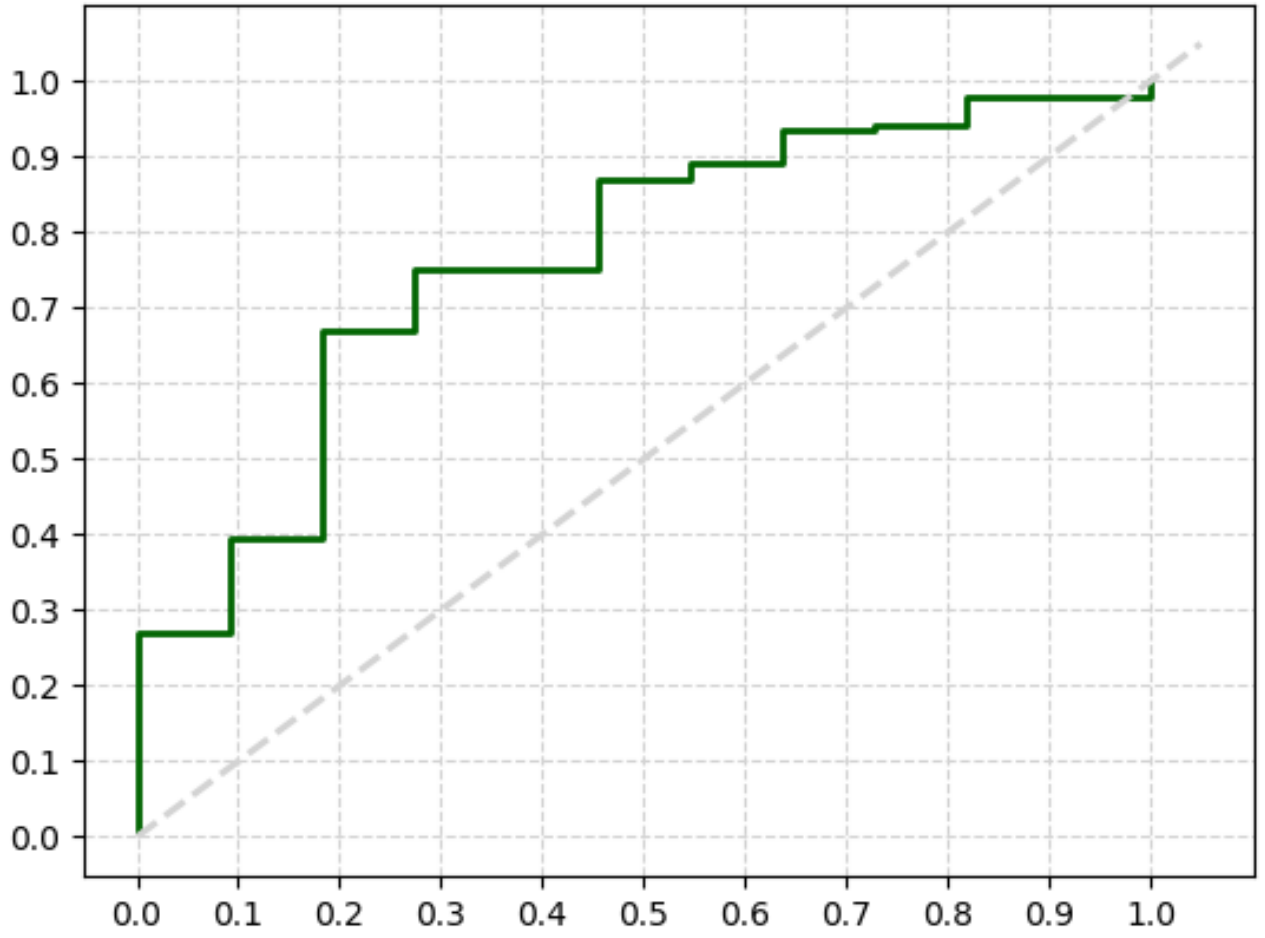
Matching the KeyPoints



Detector: ORB
Matcher: FLANN

True Positive
False Positive

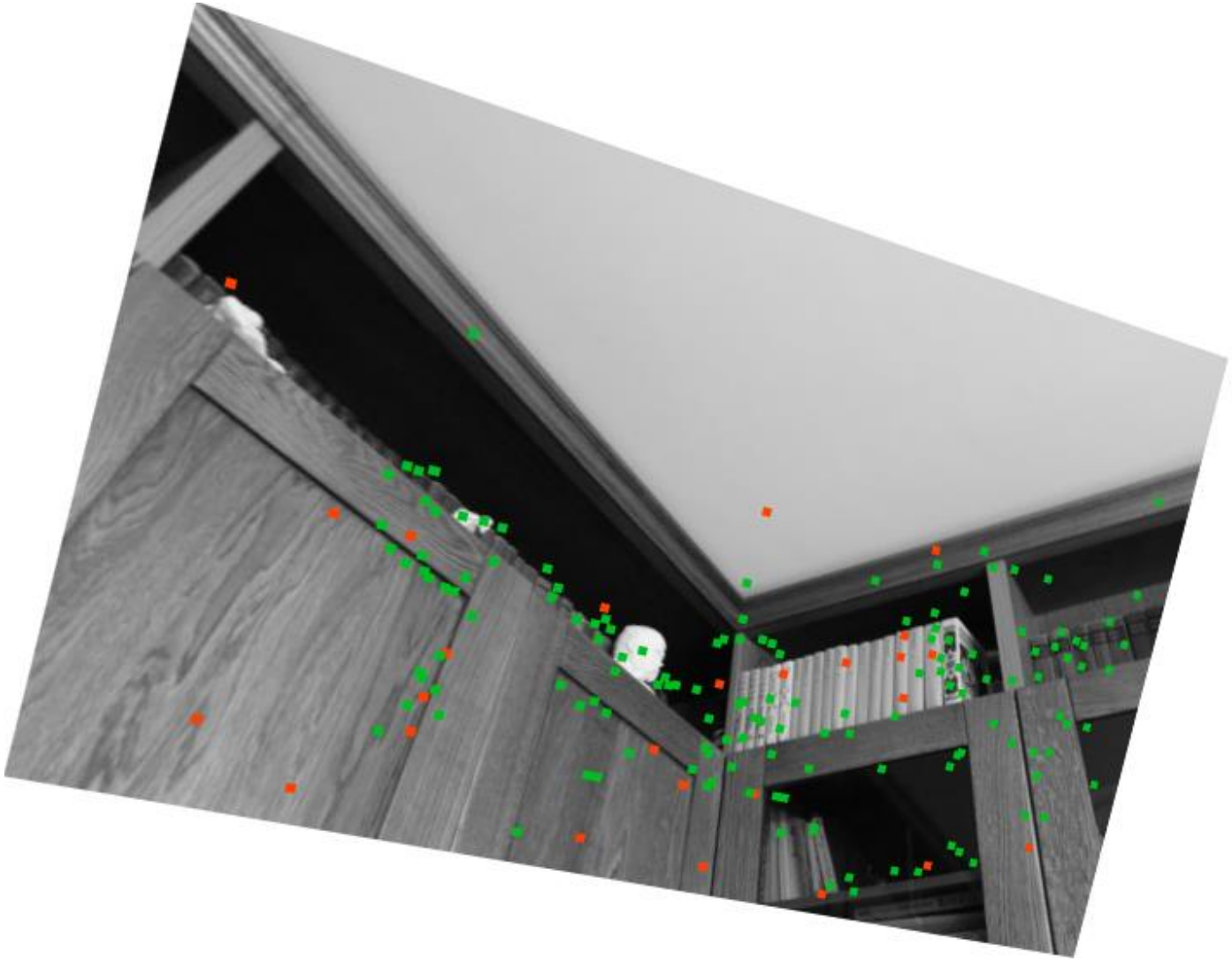
Matching the KeyPoints



Detector: ORB
Matcher: FLANN

AUC = 76%

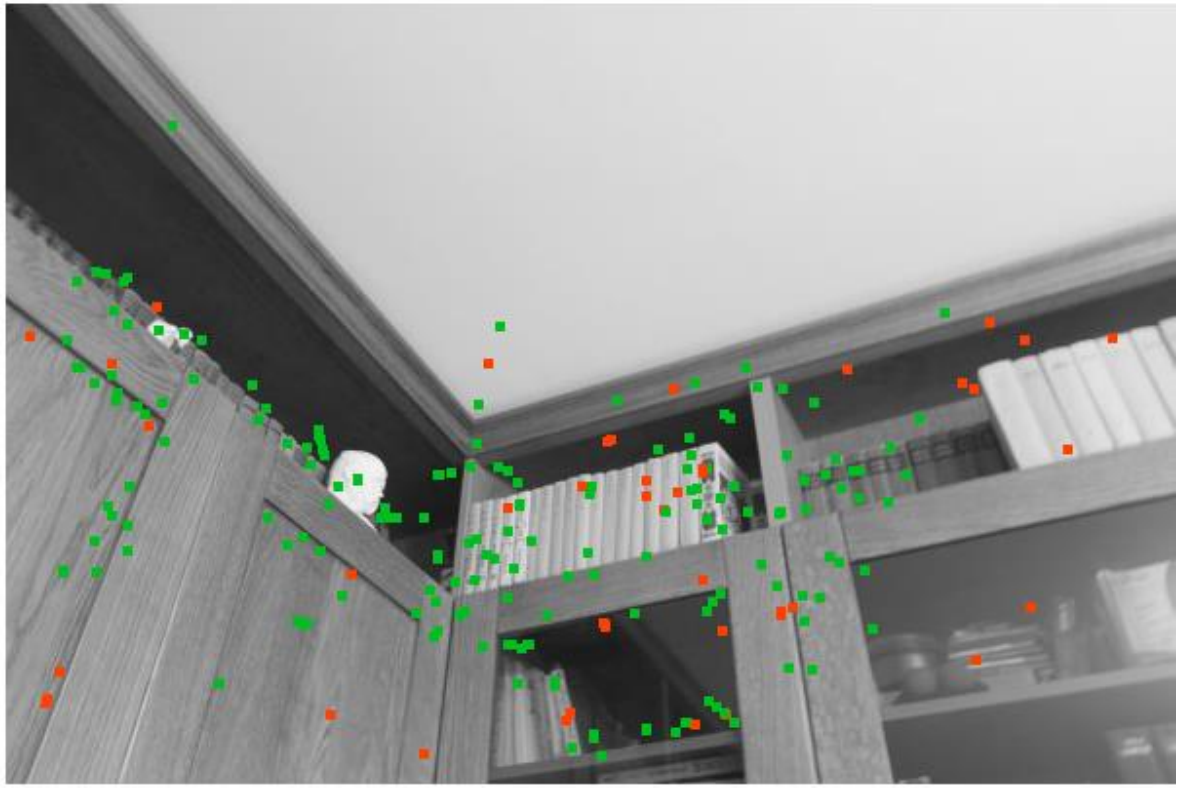
Matching the KeyPoints



Detector: SURF
Matcher: KNN

True Positive
False Positive

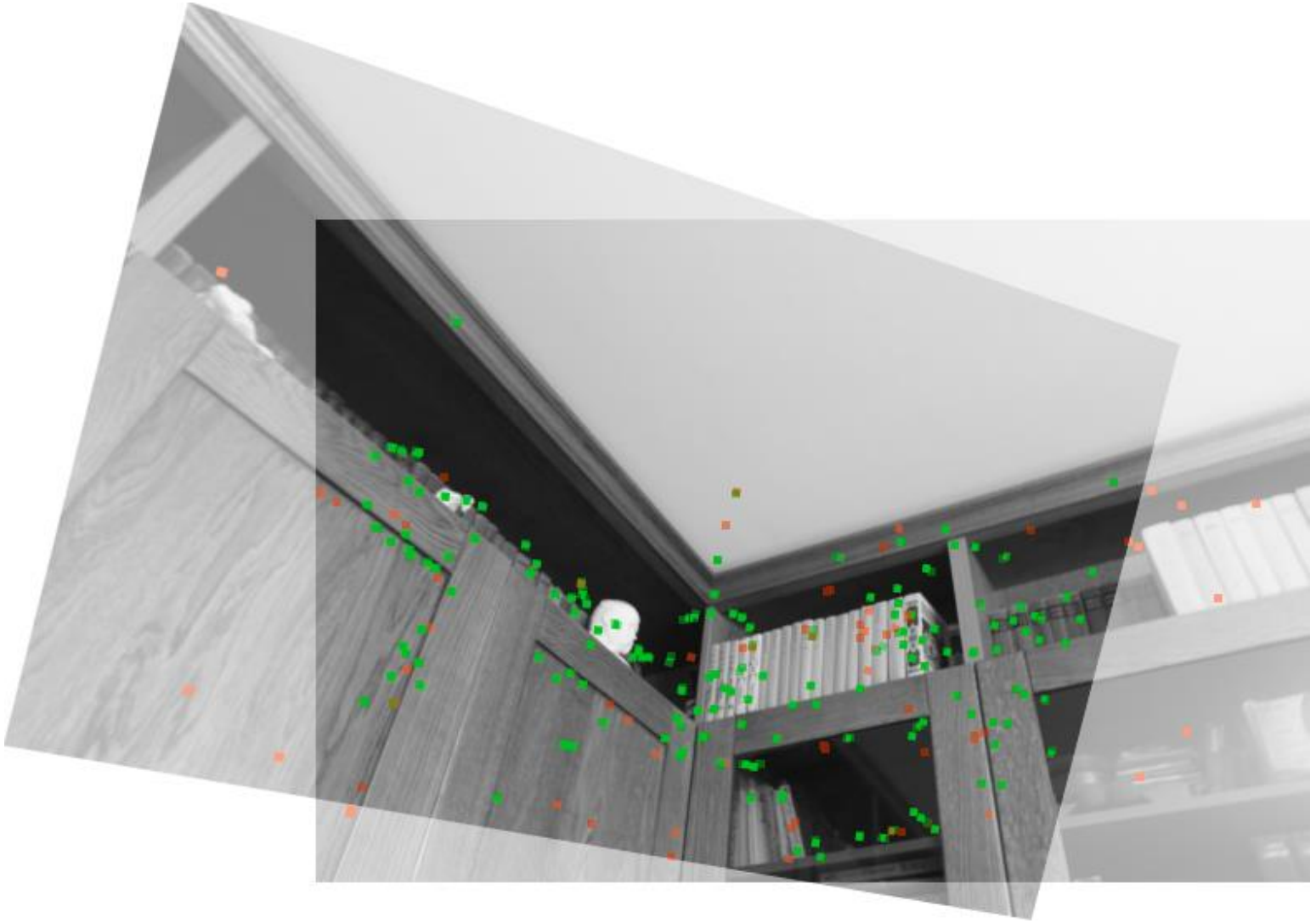
Matching the KeyPoints



Detector: SURF
Matcher: KNN

True Positive
False Positive

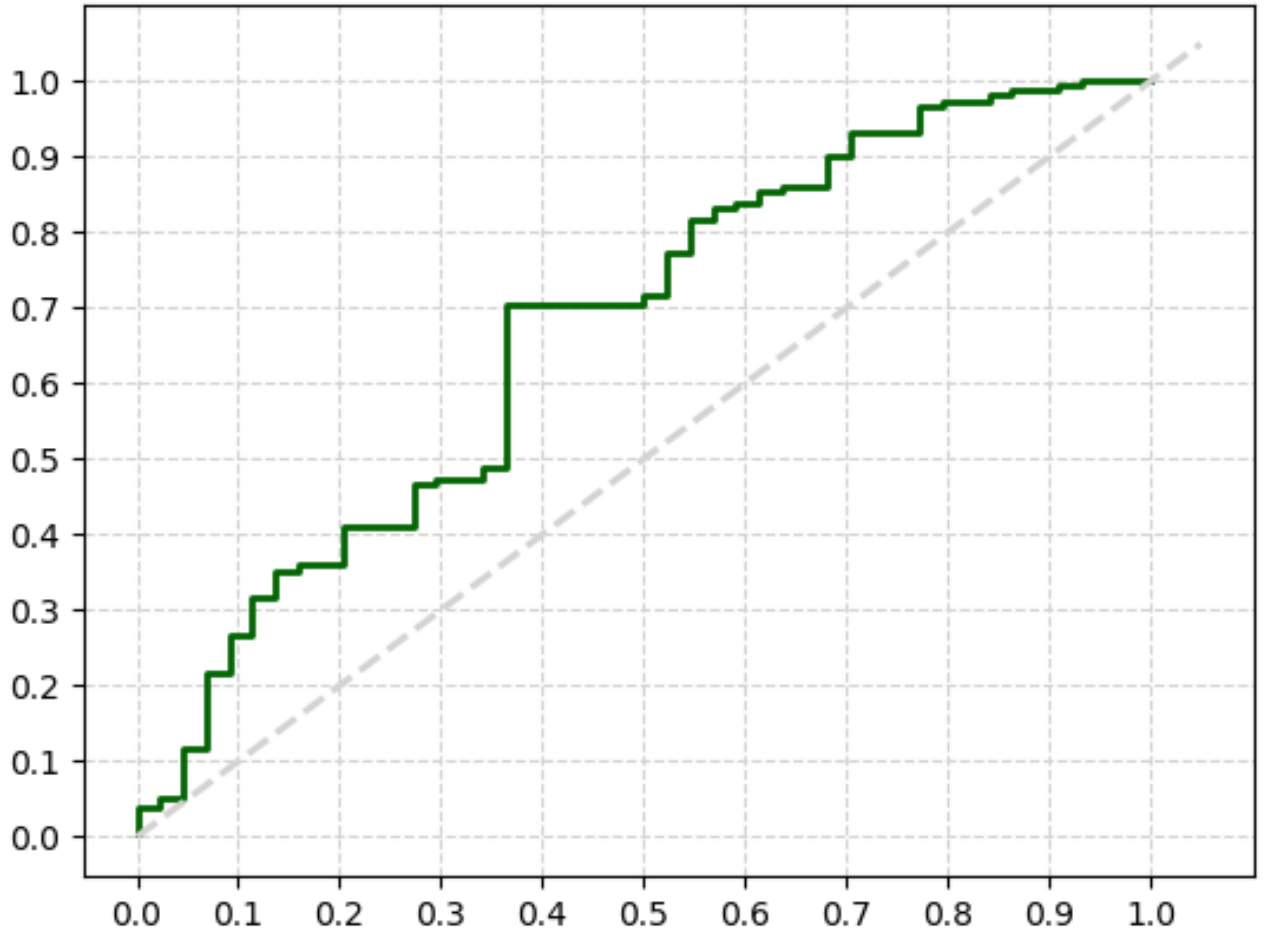
Matching the KeyPoints



Detector: SURF
Matcher: KNN

True Positive
False Positive

Matching the KeyPoints



Detector: SURF
Matcher: KNN

AUC = 67%

Matching the KeyPoints

	BFMatcher.**match**	BFMatcher.**knnMatch**	**Flann**BasedMatcher.knnMatch
ORB	91%	75%	76%
SIFT	63%	69%	70%
SURF	-	67%	69%

3. Homography

https://github.com/dryabokon/geometry/blob/master/ex03_homography.py

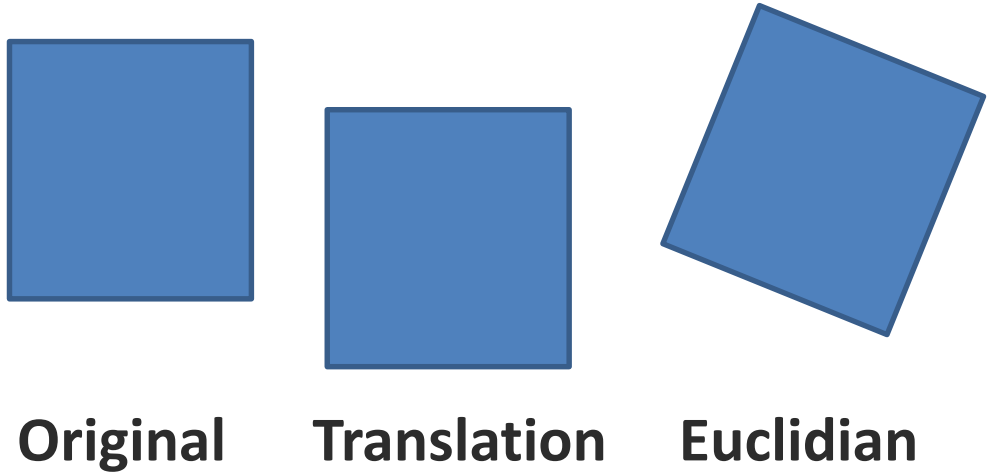
Homography: transformations

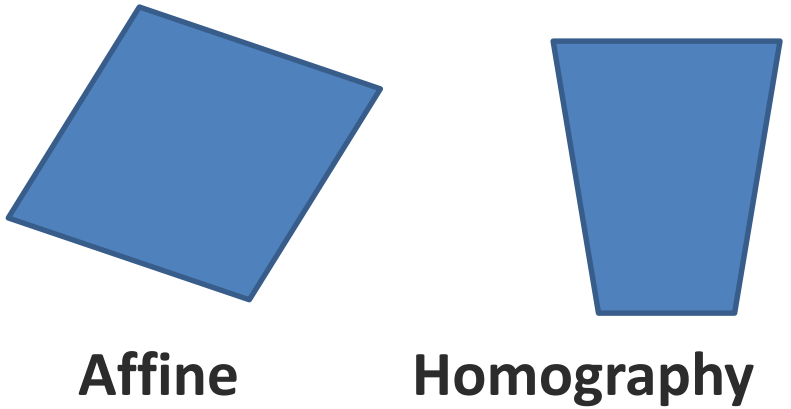
- *findTransformECC*
- *findHomography*

- *estimateAffine2D,*
- *estimateAffinePartial2D*

- *transform*
- *perspectiveTransform*
- *getPerspectiveTransform*

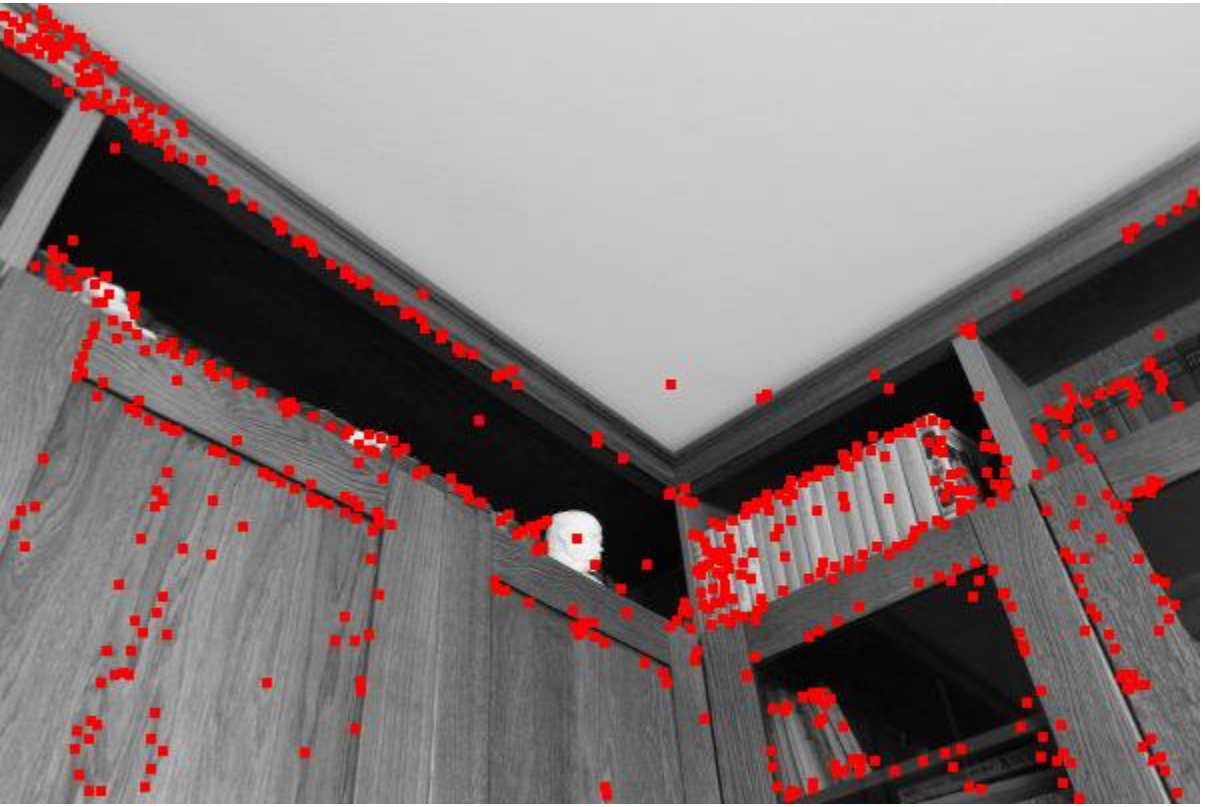
- *warpAffine*
- *warpPerspective*

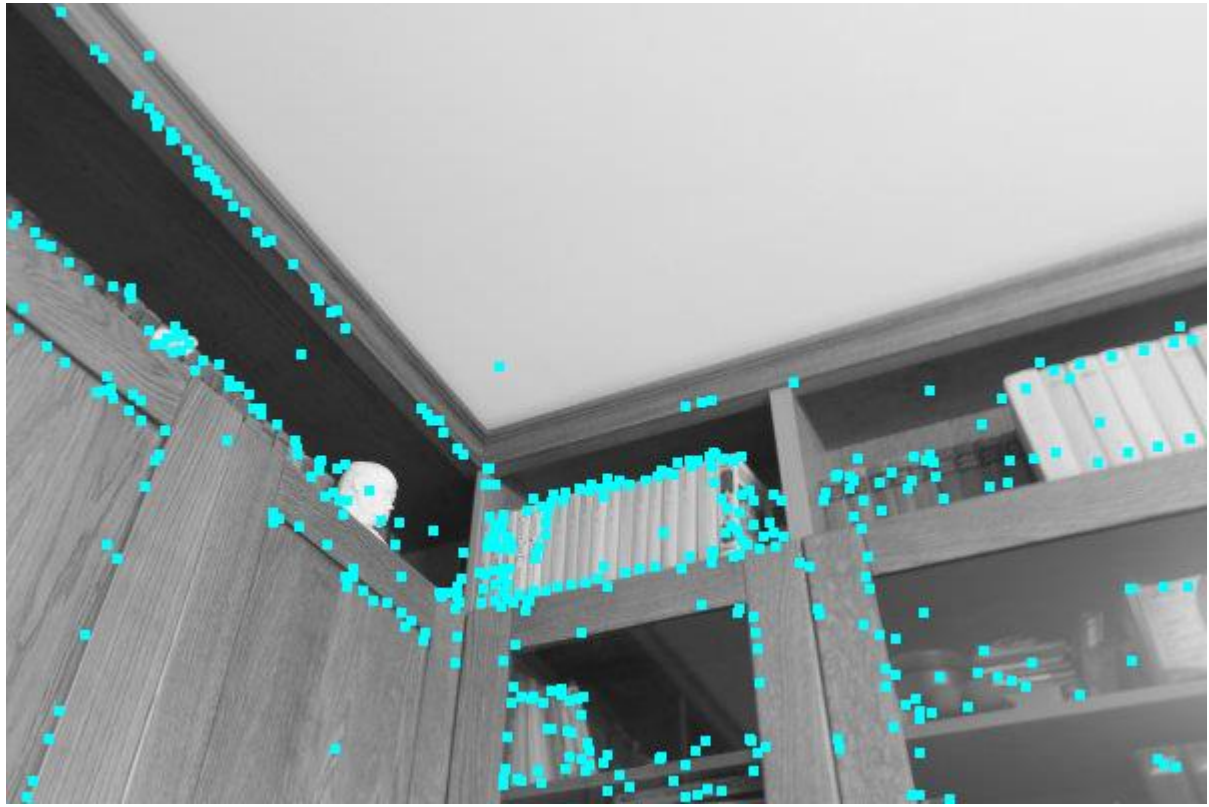




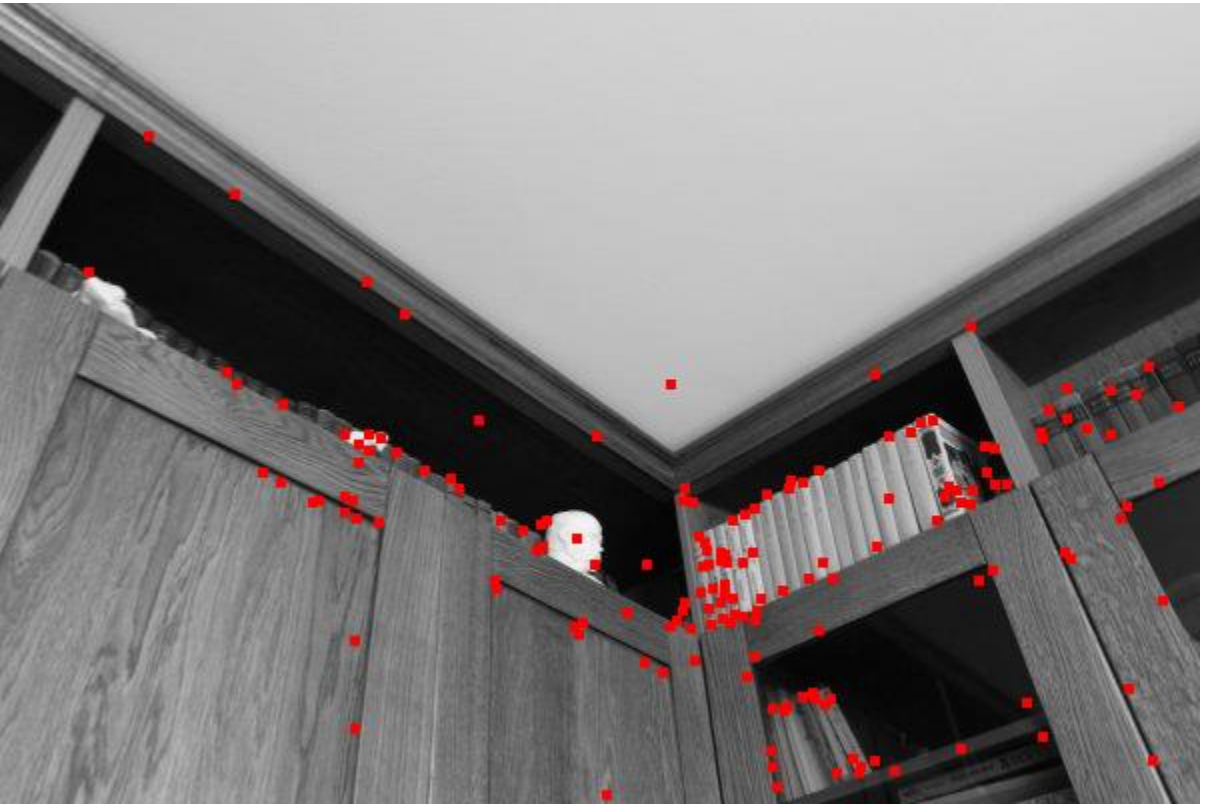
Homography: manual match

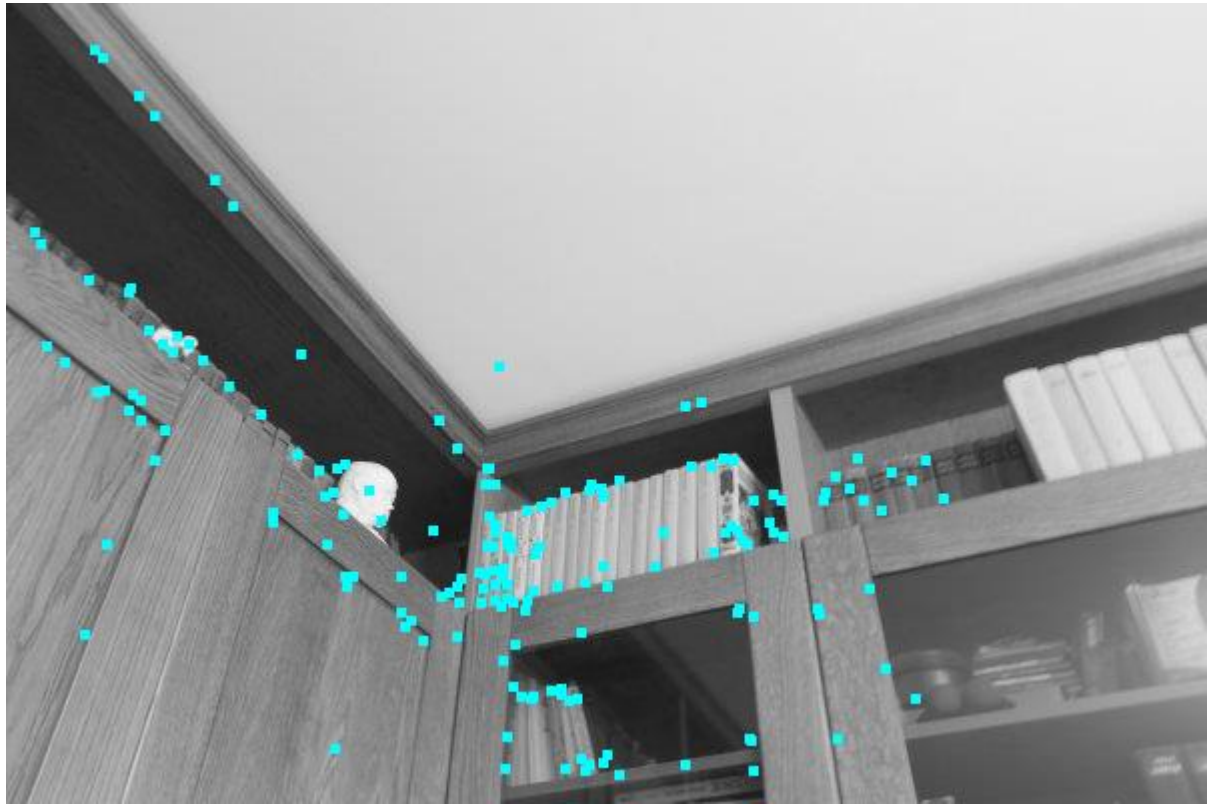
Homography: auto-match



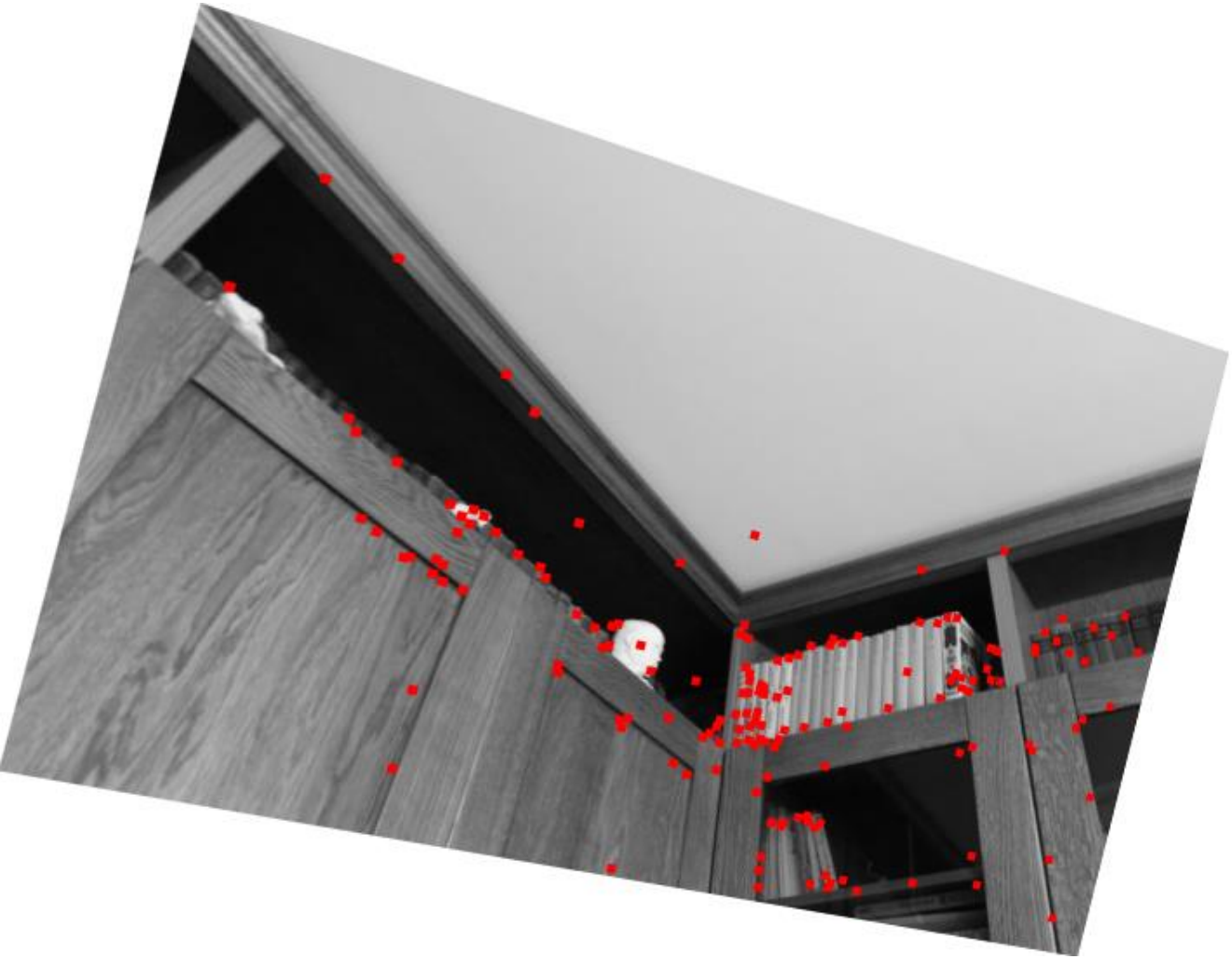


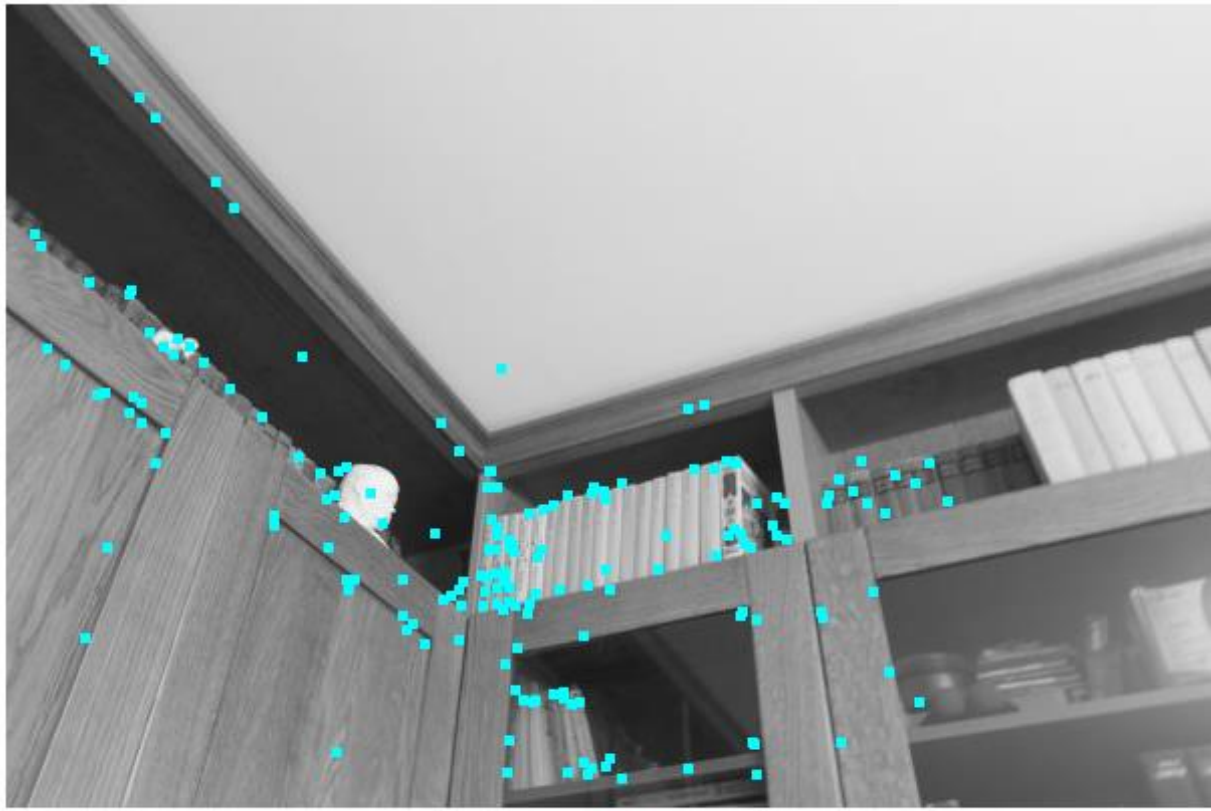
Homography: auto-match



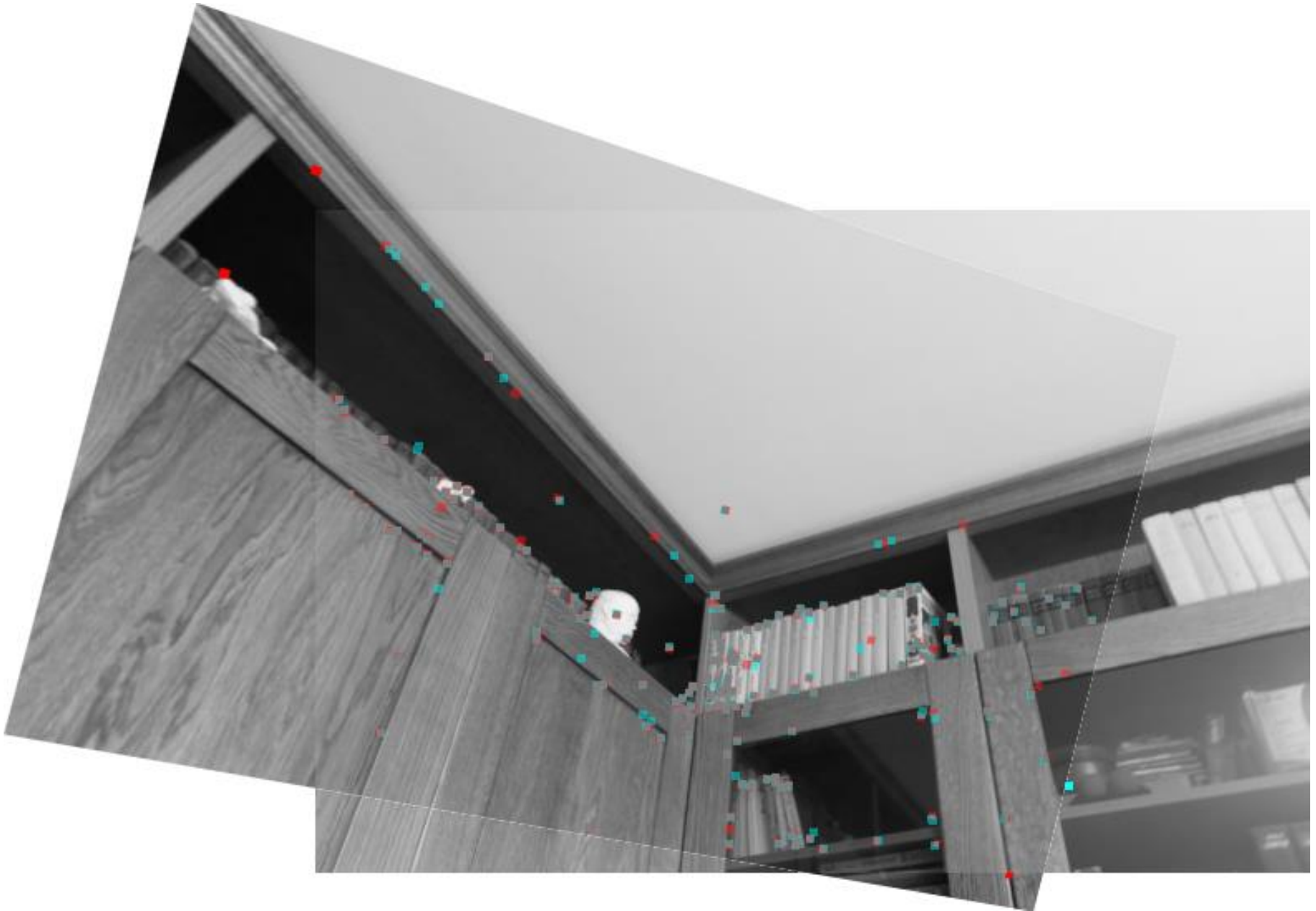


Homography: auto-match

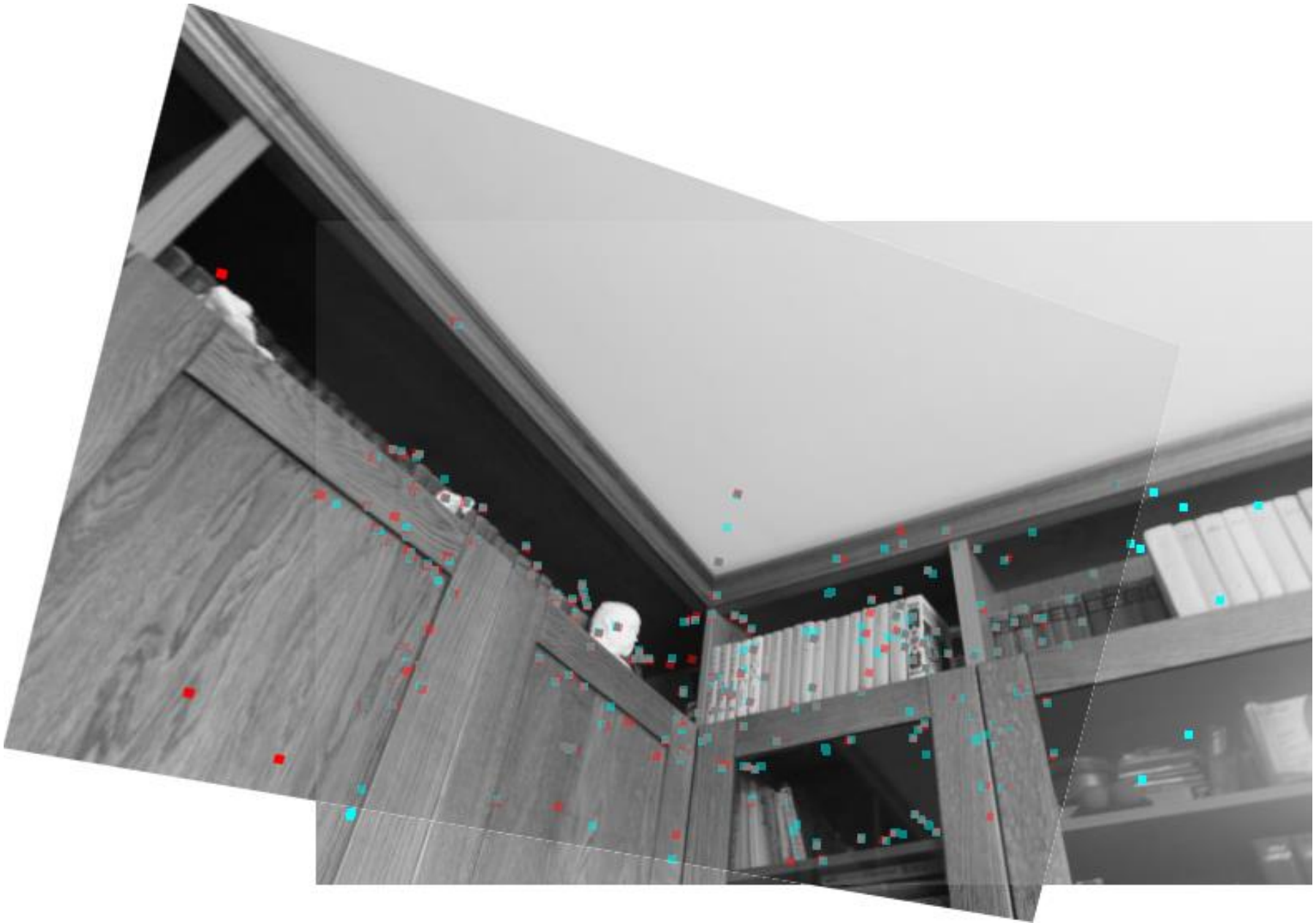




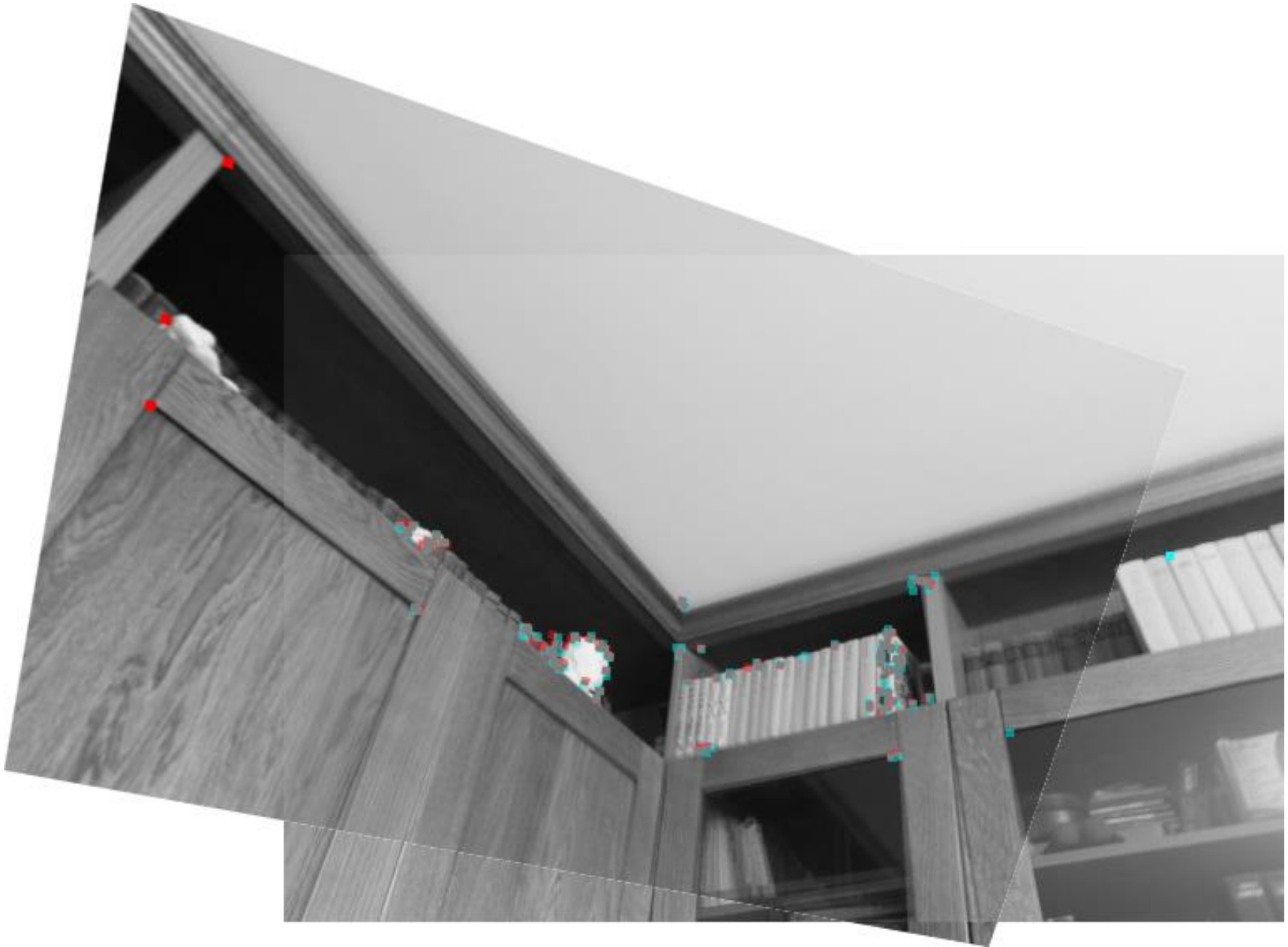
Homography: auto-match SIFT



Homography: auto-match SURF

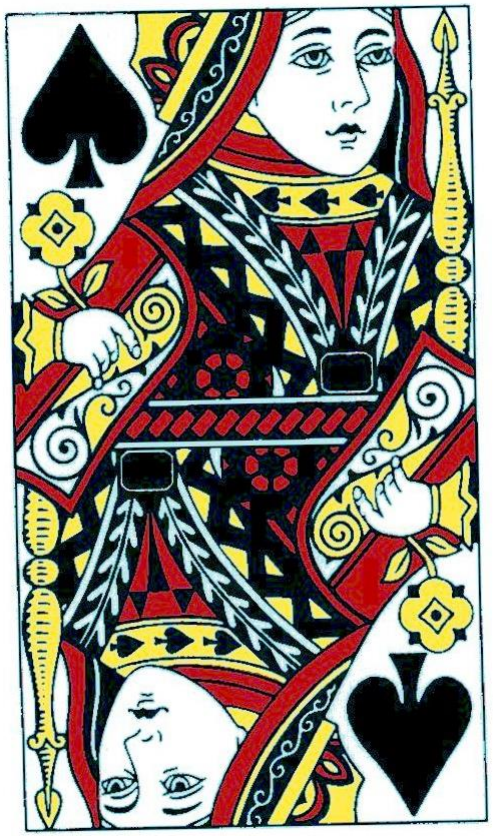


Homography: auto-match ORB



Homography: bitmap transformation

first



second

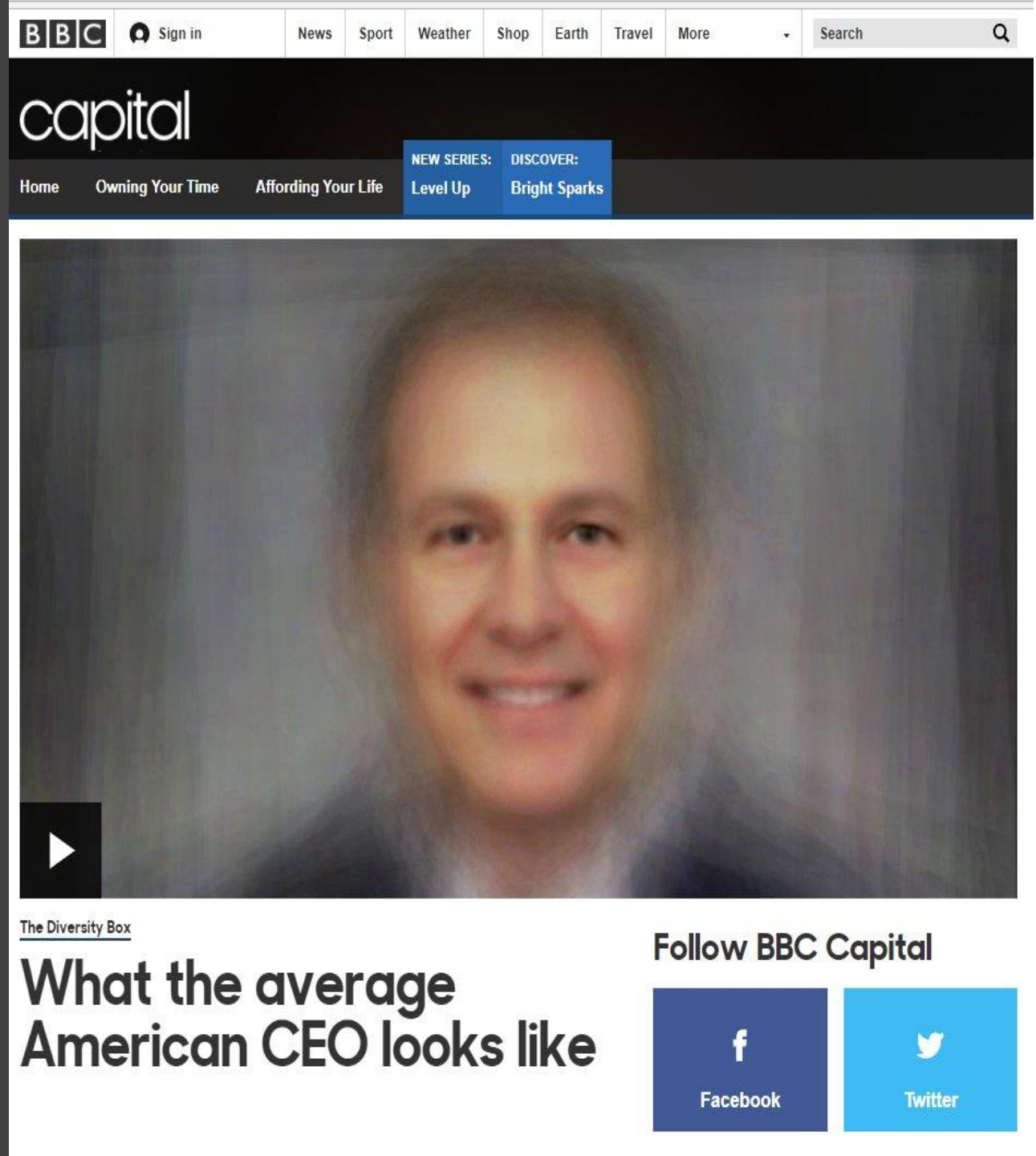
Homography: bitmap transformation

*transformed **second***

*transformed **first***

4. Blending

https://github.com/dryabokon/geometry/blob/master/ex04_blending.py



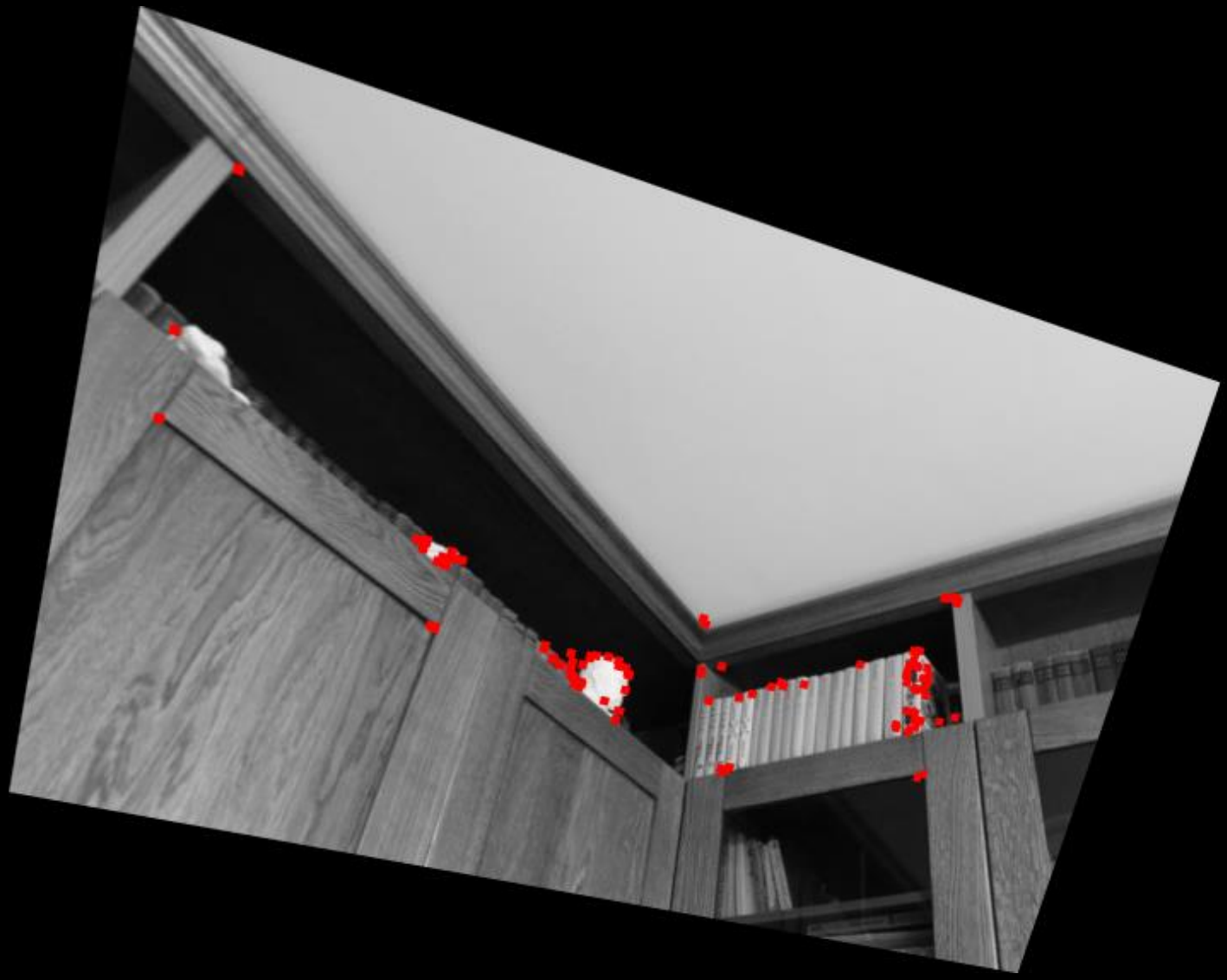
Blending: average

Blending: average

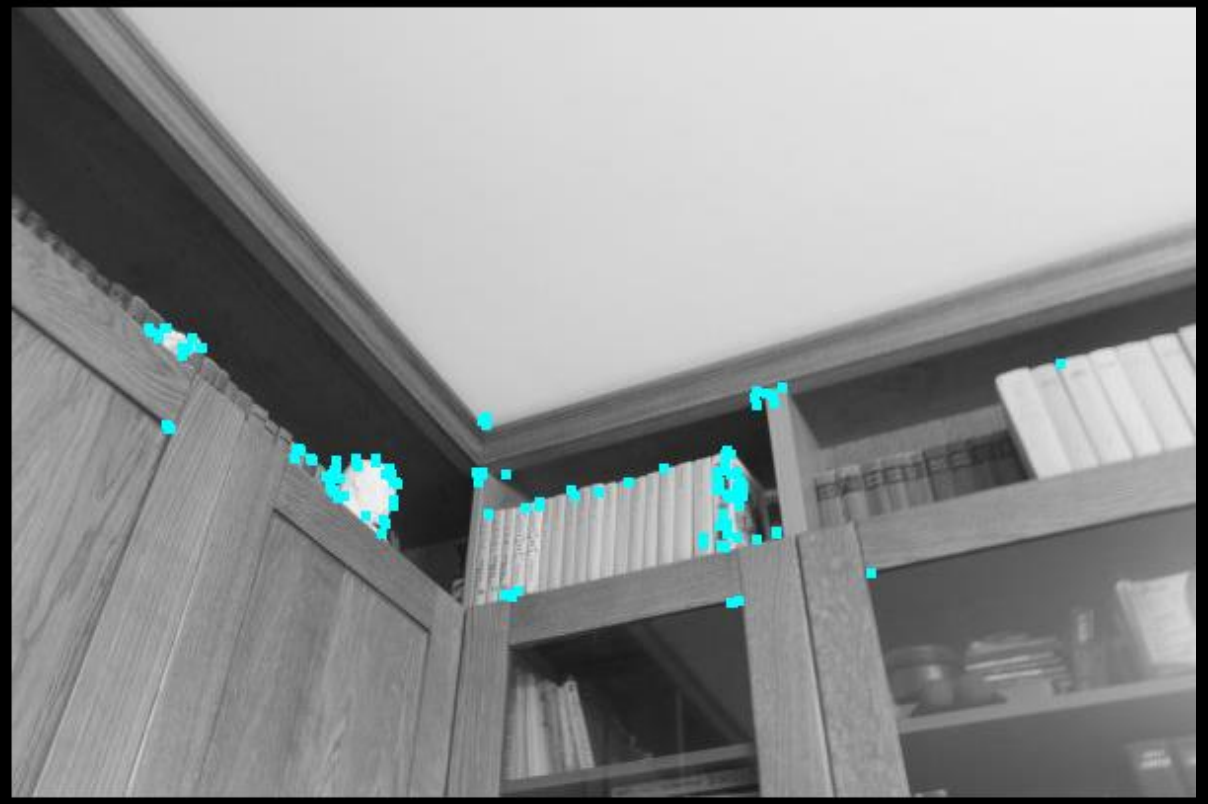
Blending: average

Blending: average

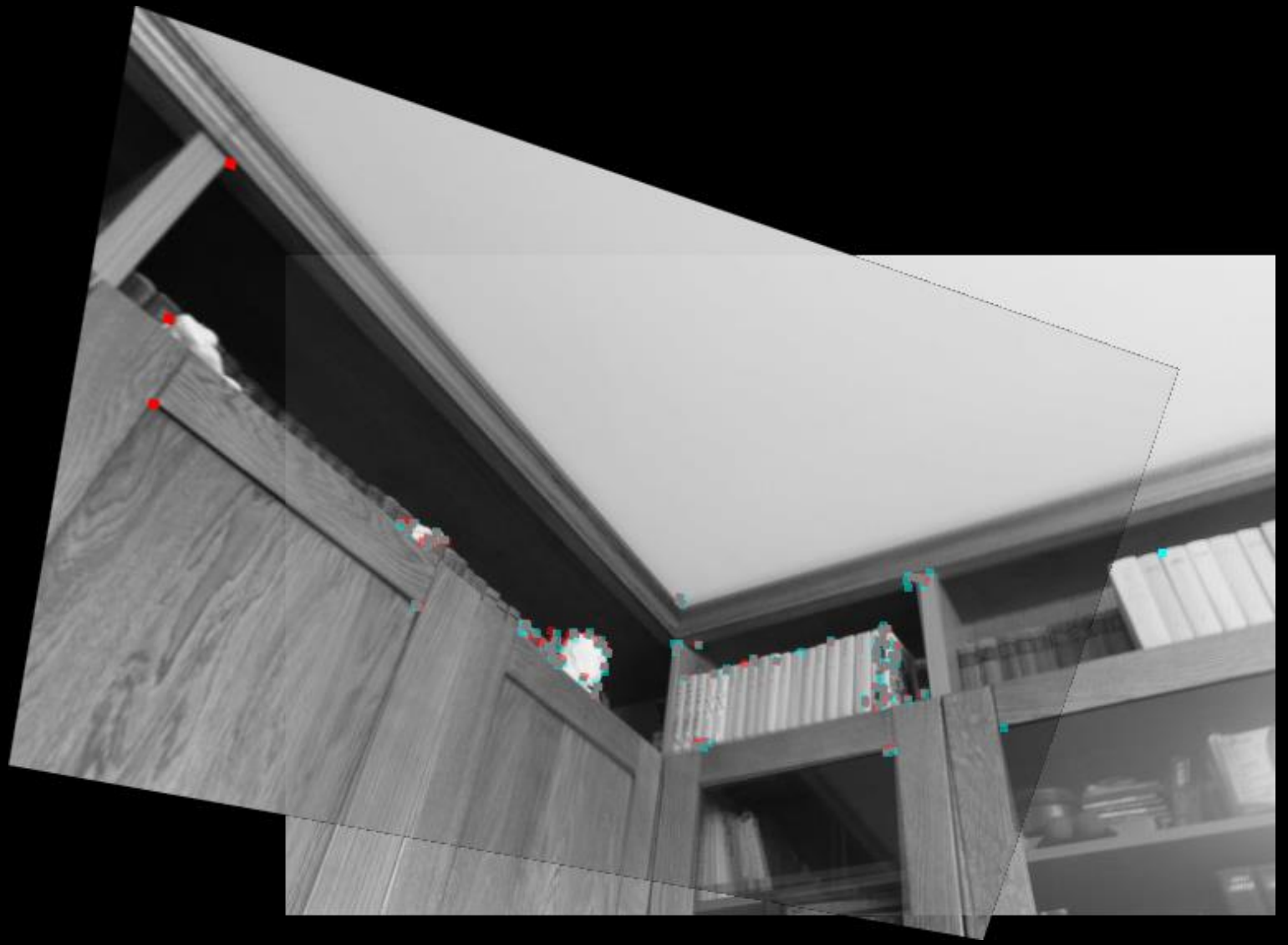
Blending: average



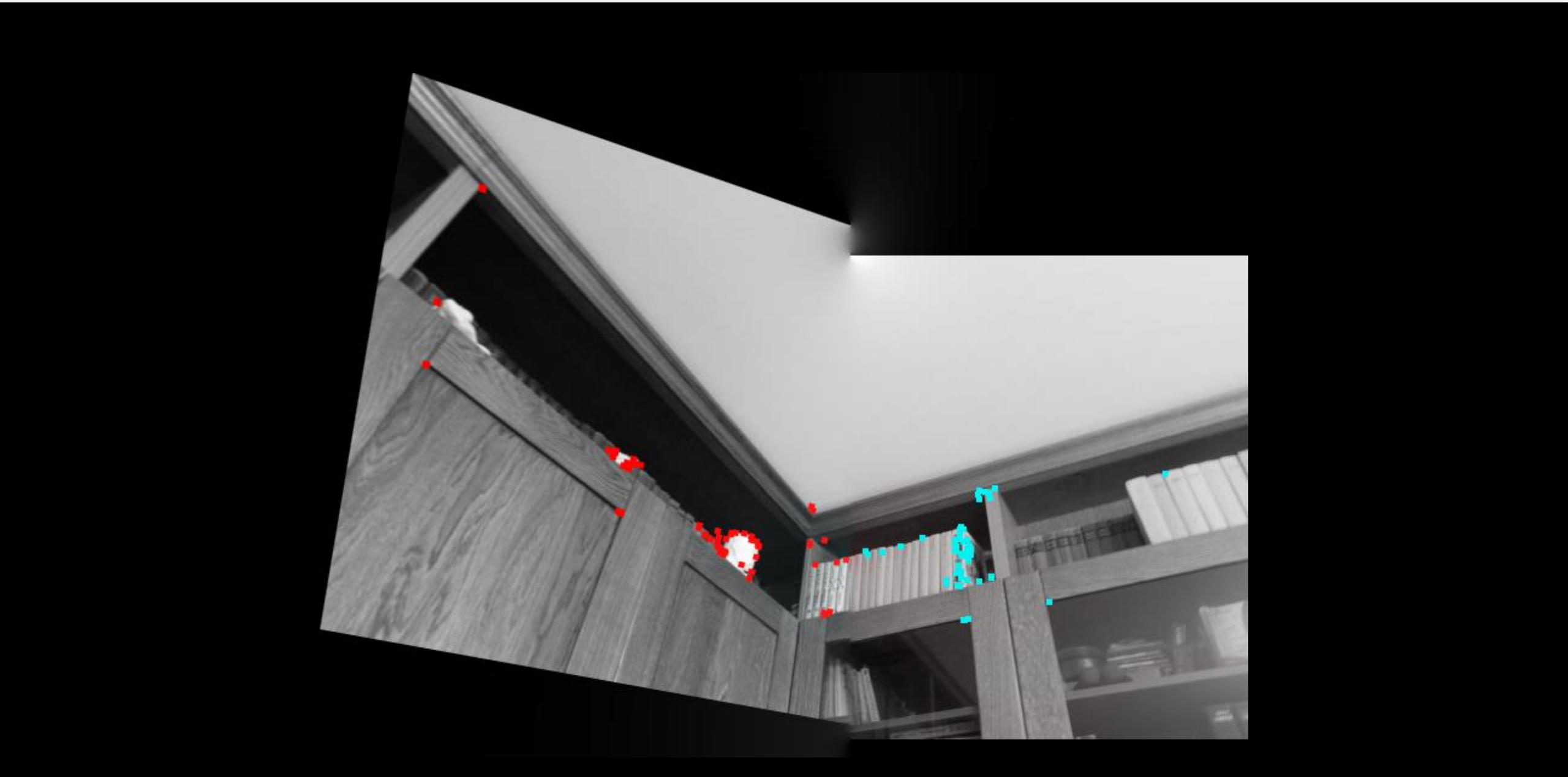
Blending: average



Blending: average



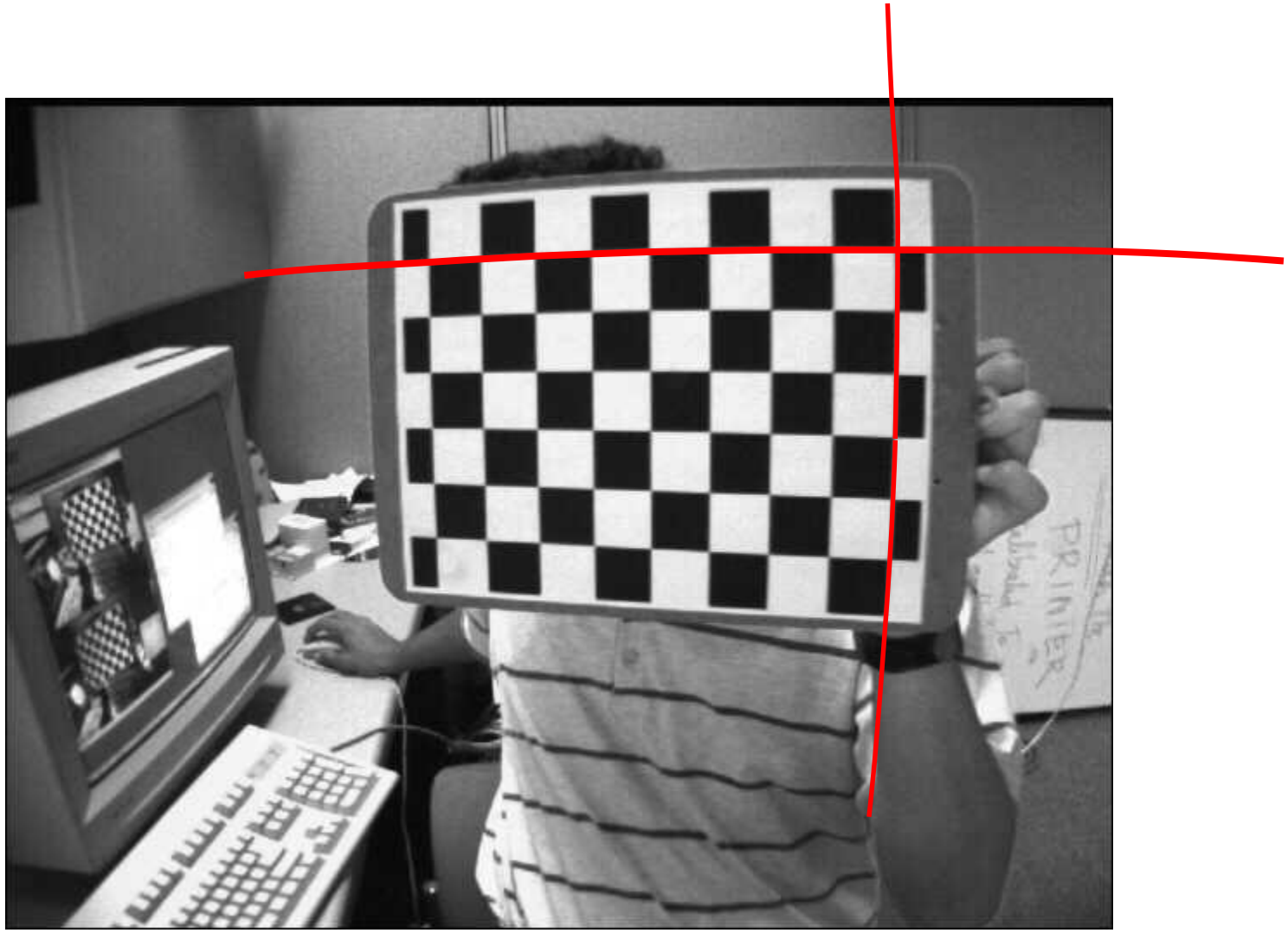
Blending: average



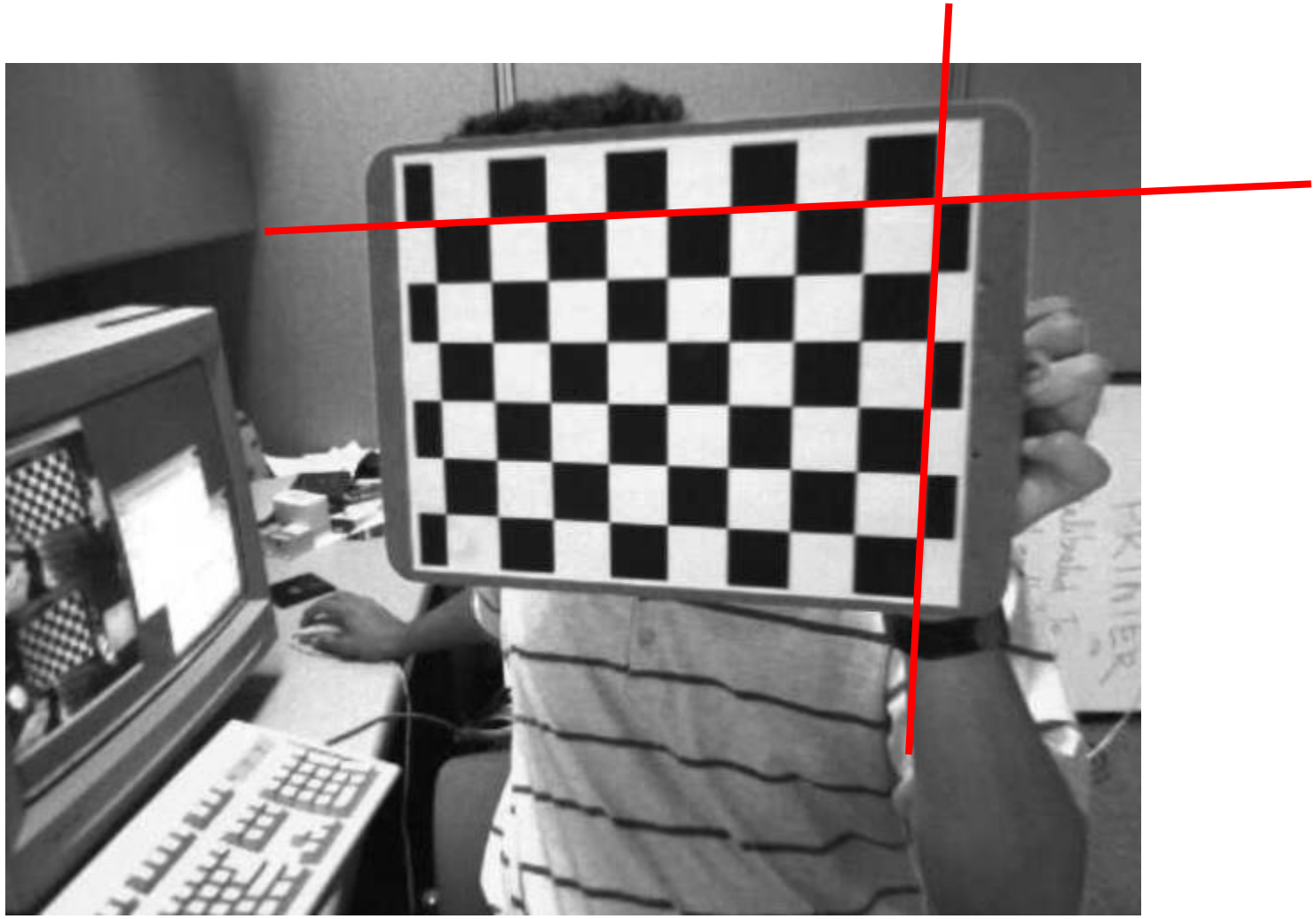
Camera calibration

https://github.com/dryabokon/geometry/blob/master/ex06_camera_calibration.py

Camera calibration



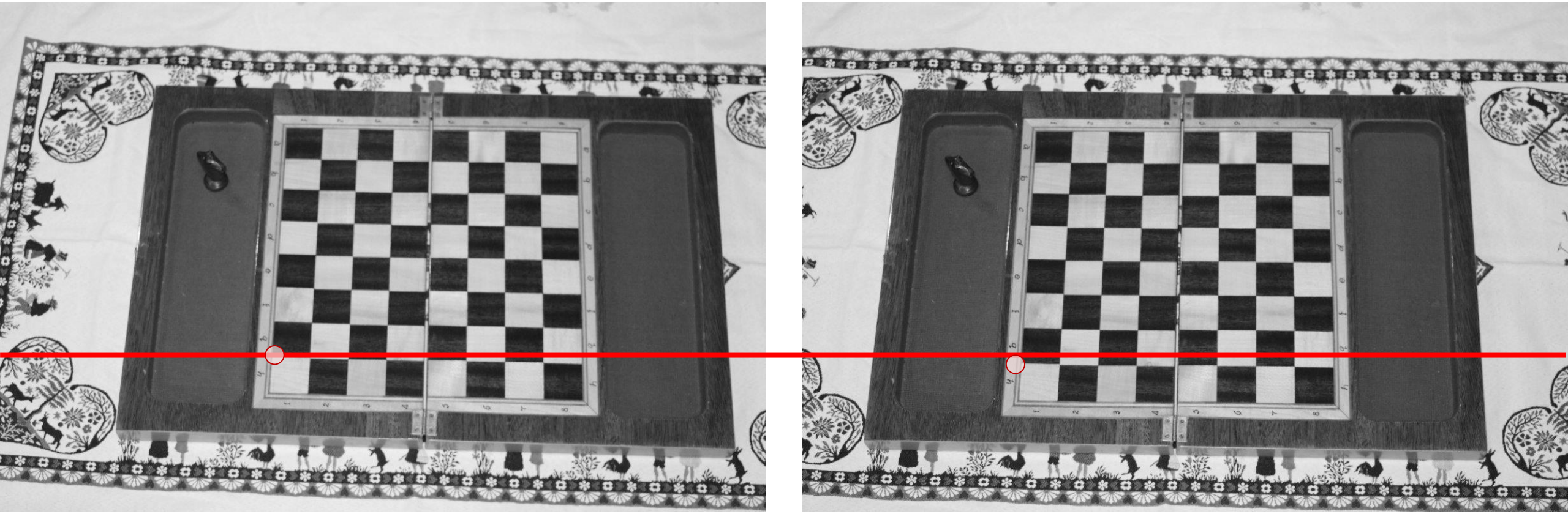
Camera calibration



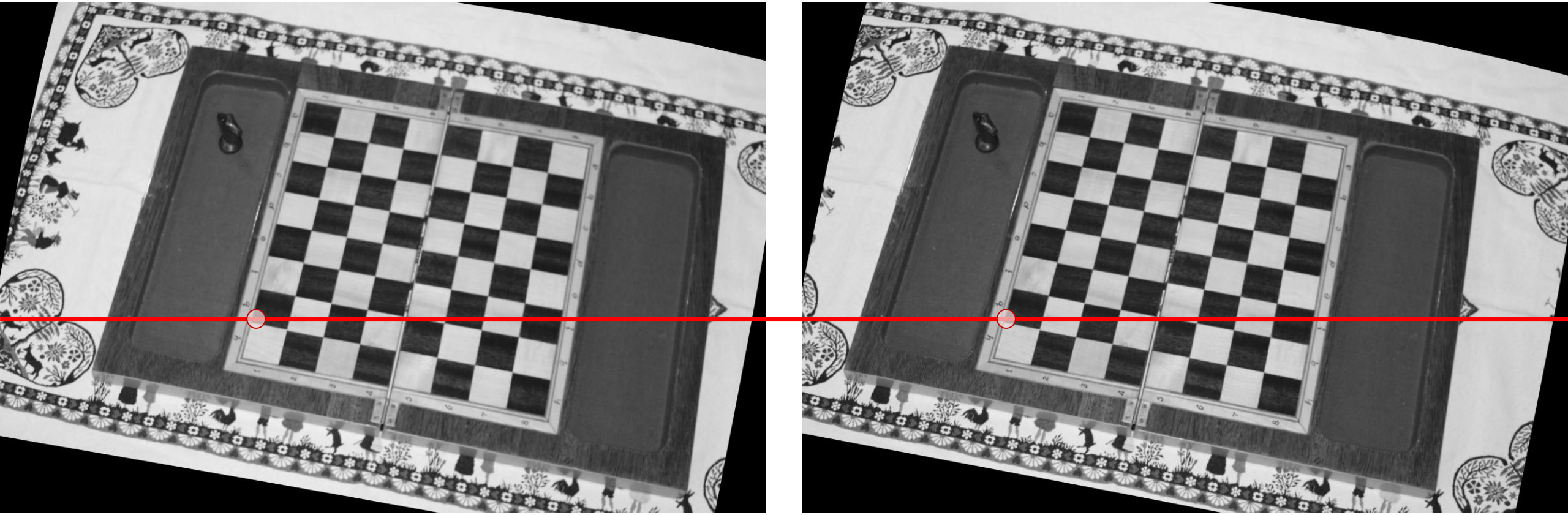
Stereopair rectification

https://github.com/dryabokon/geometry/blob/master/ex07_rectifiy_stereopair.py

Stereopair rectification

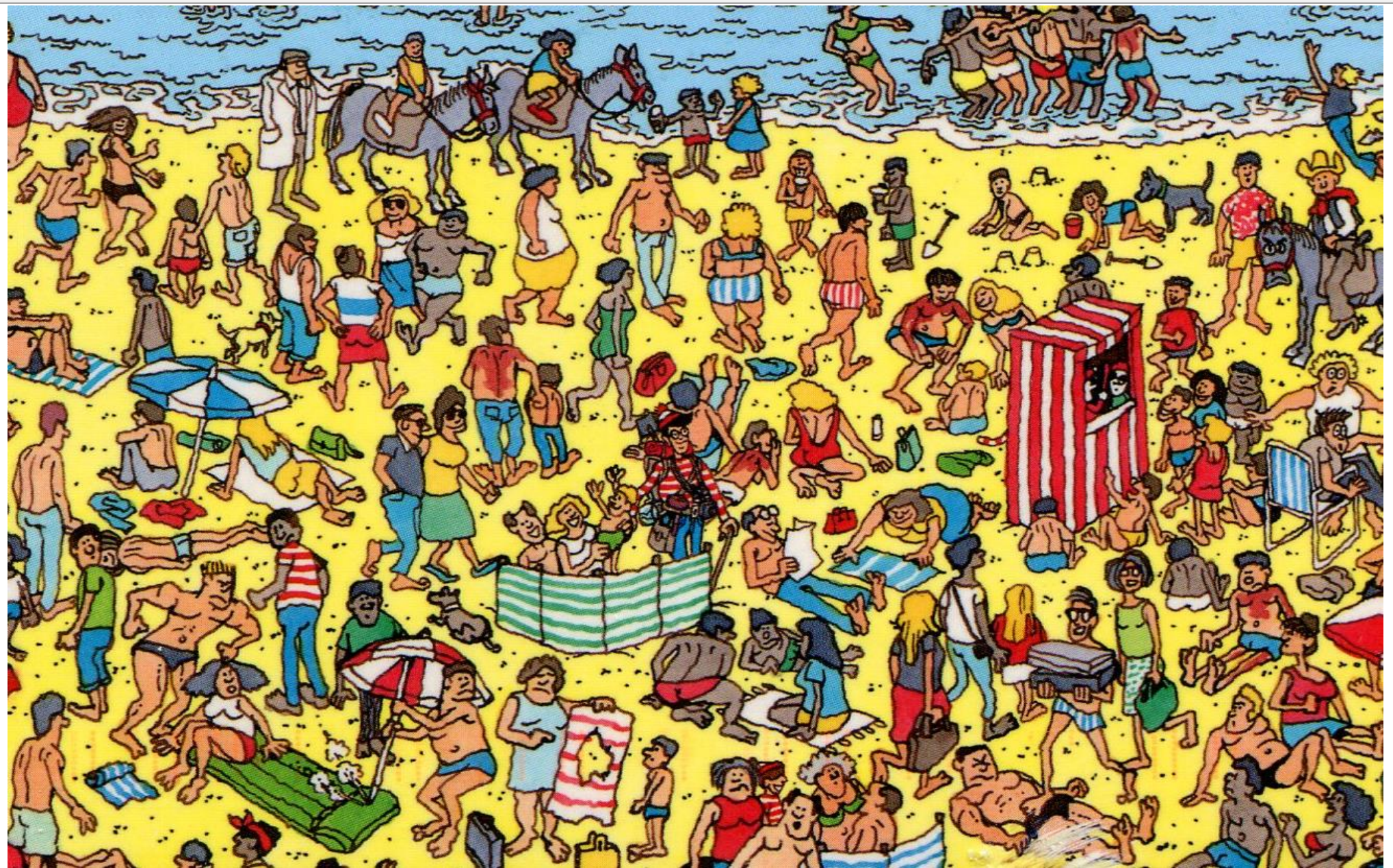


Stereopair rectification



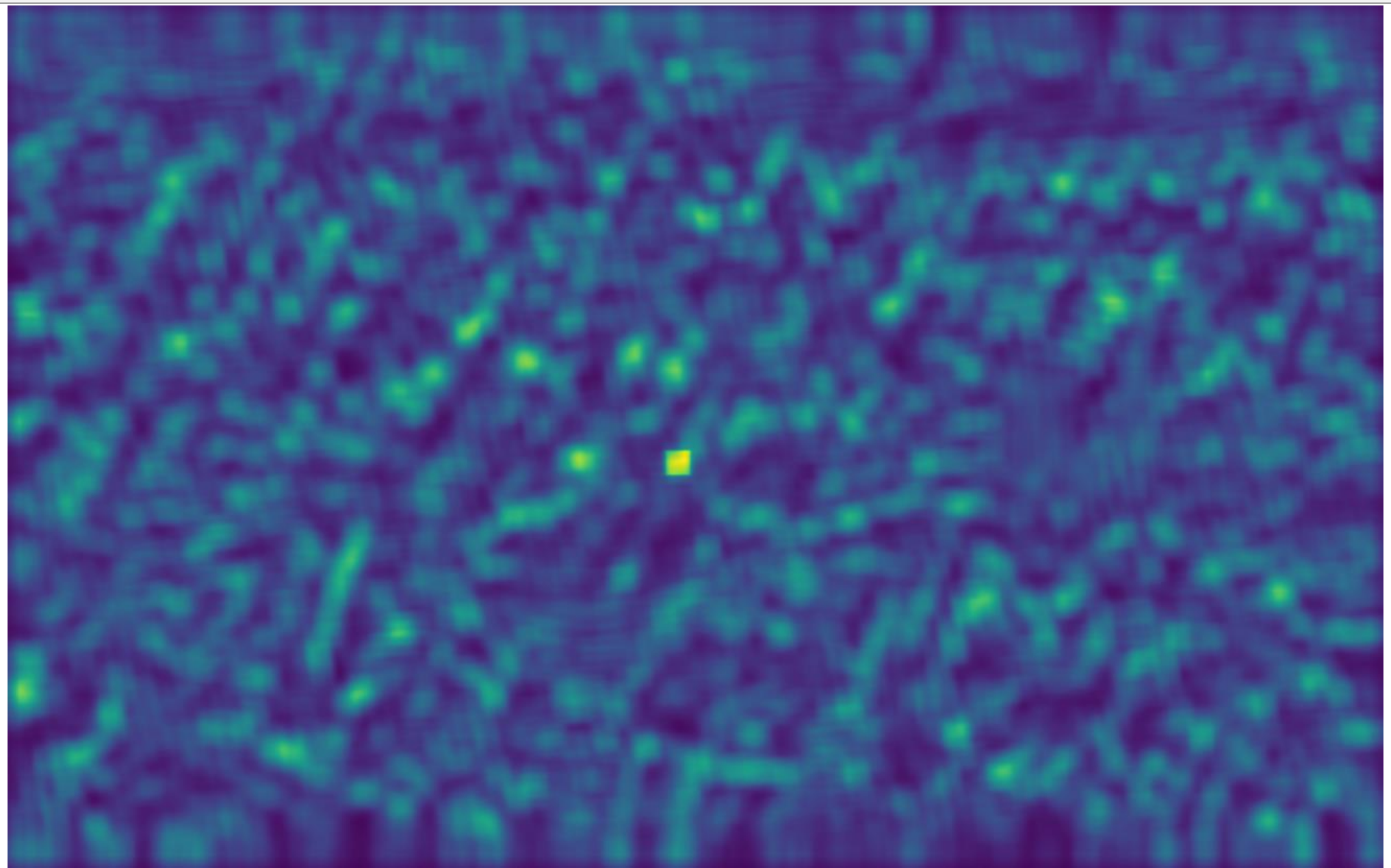
Template matching

Template Matching

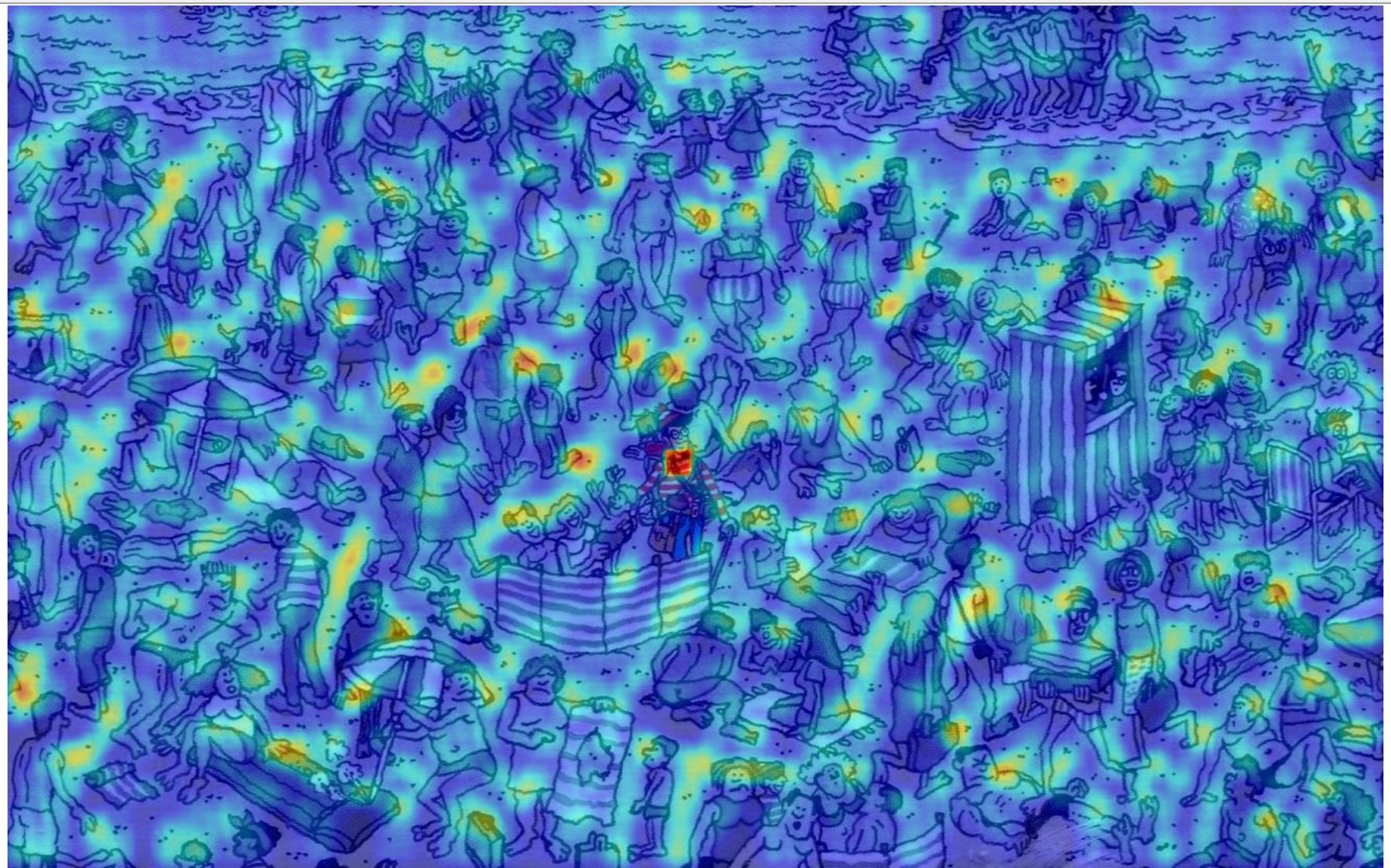


Template Matching

Template Matching



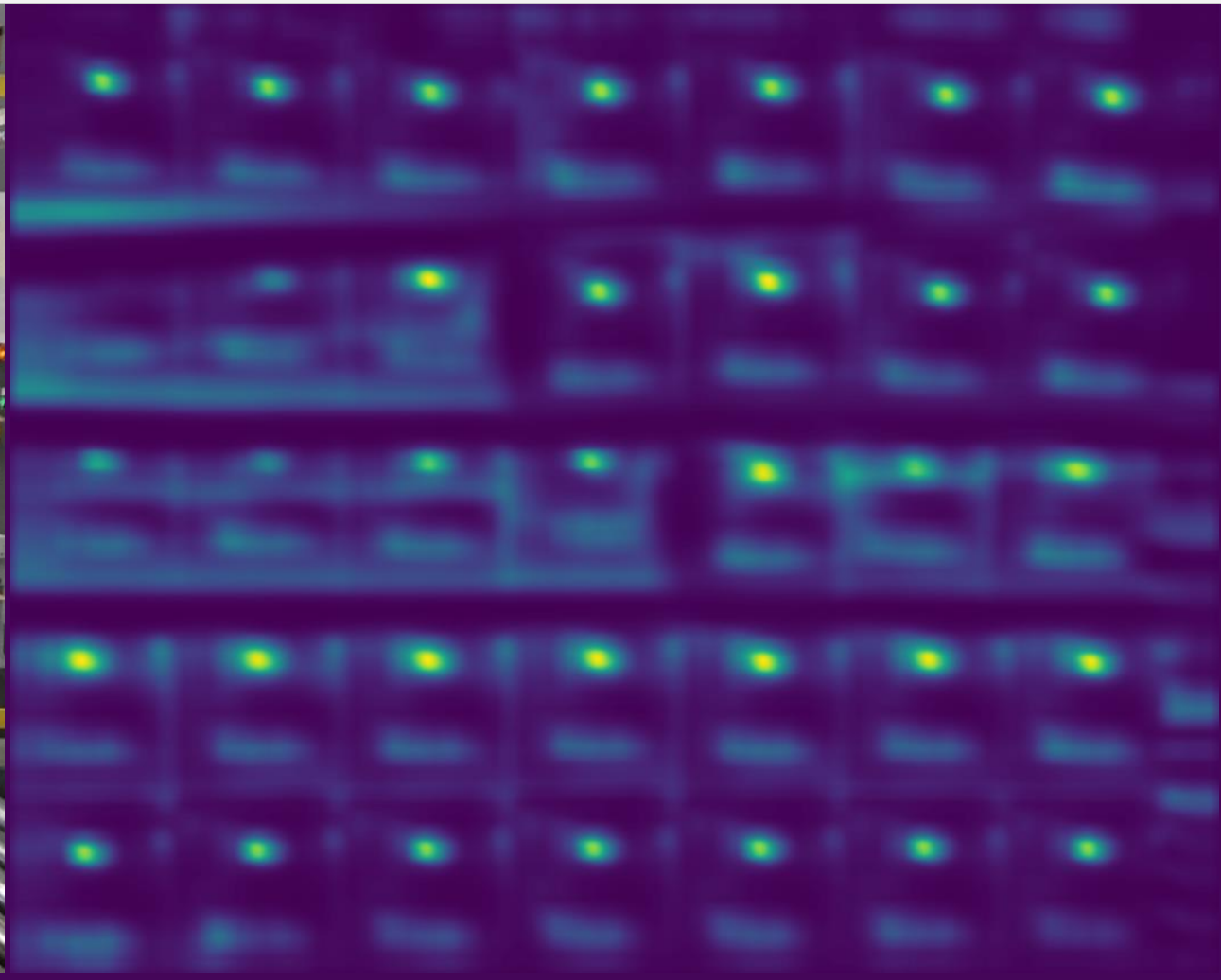
Template Matching



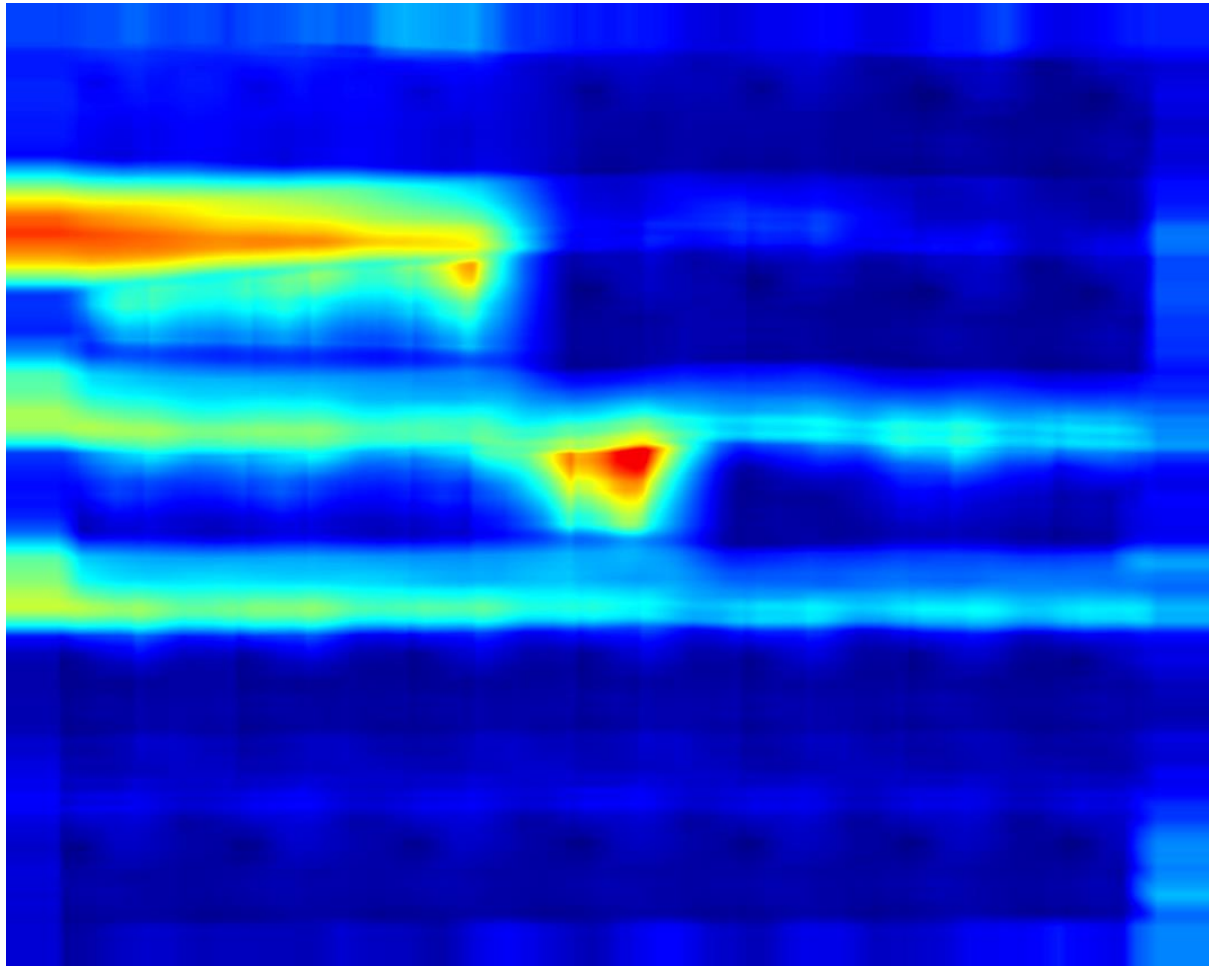
Template Matching

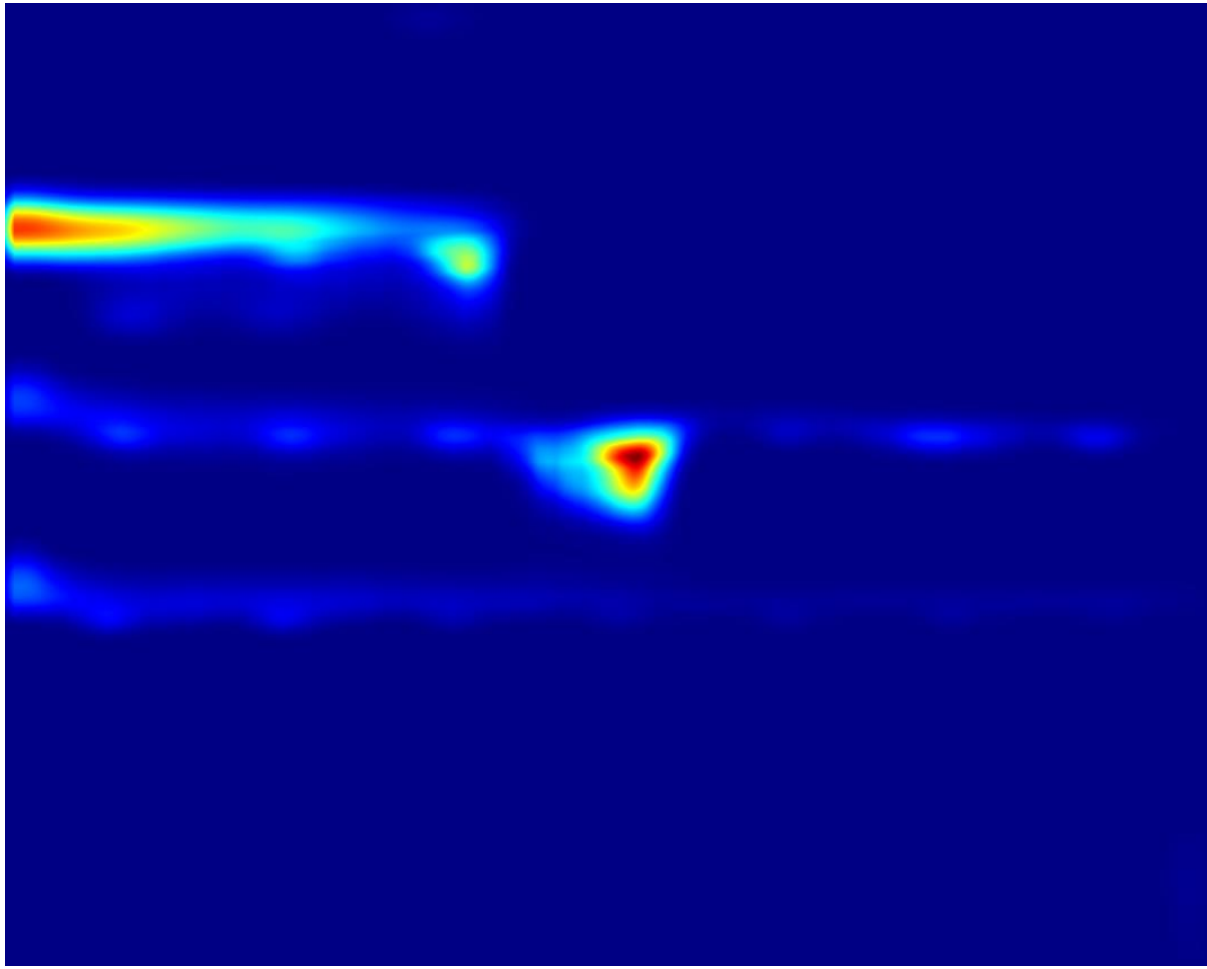
Template Matching

Template Matching

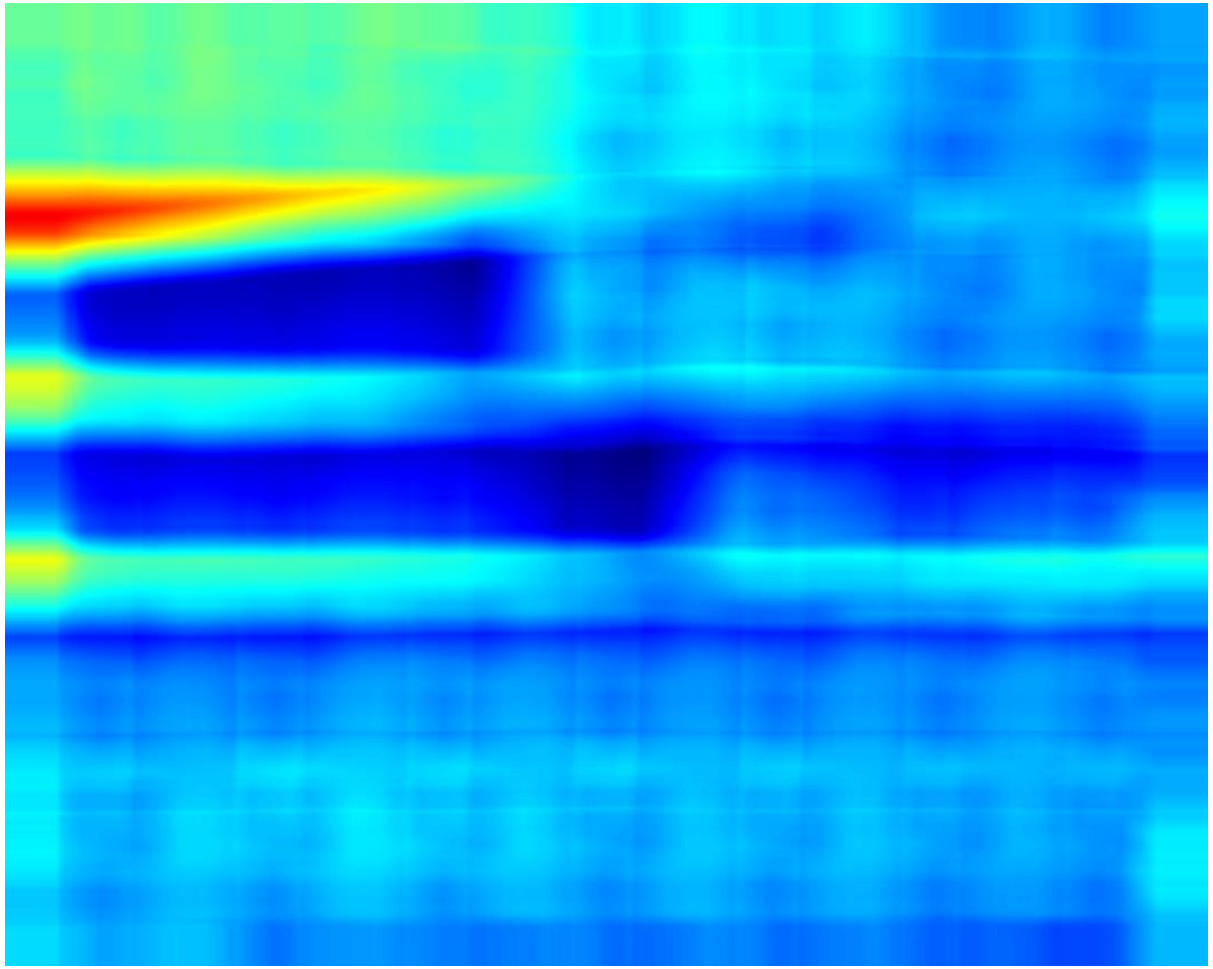


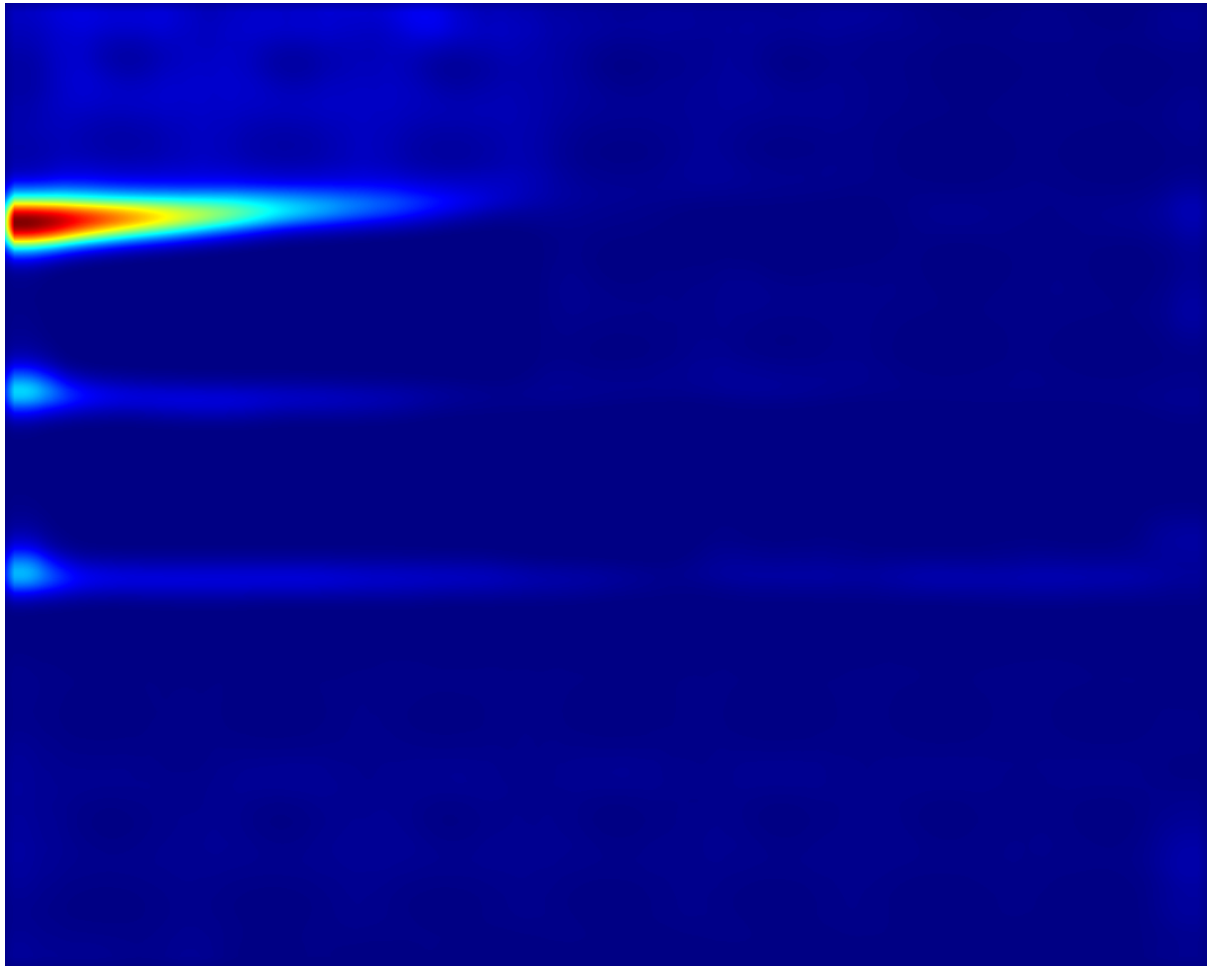
Template Matching: TM_SQDIFF_NORMED



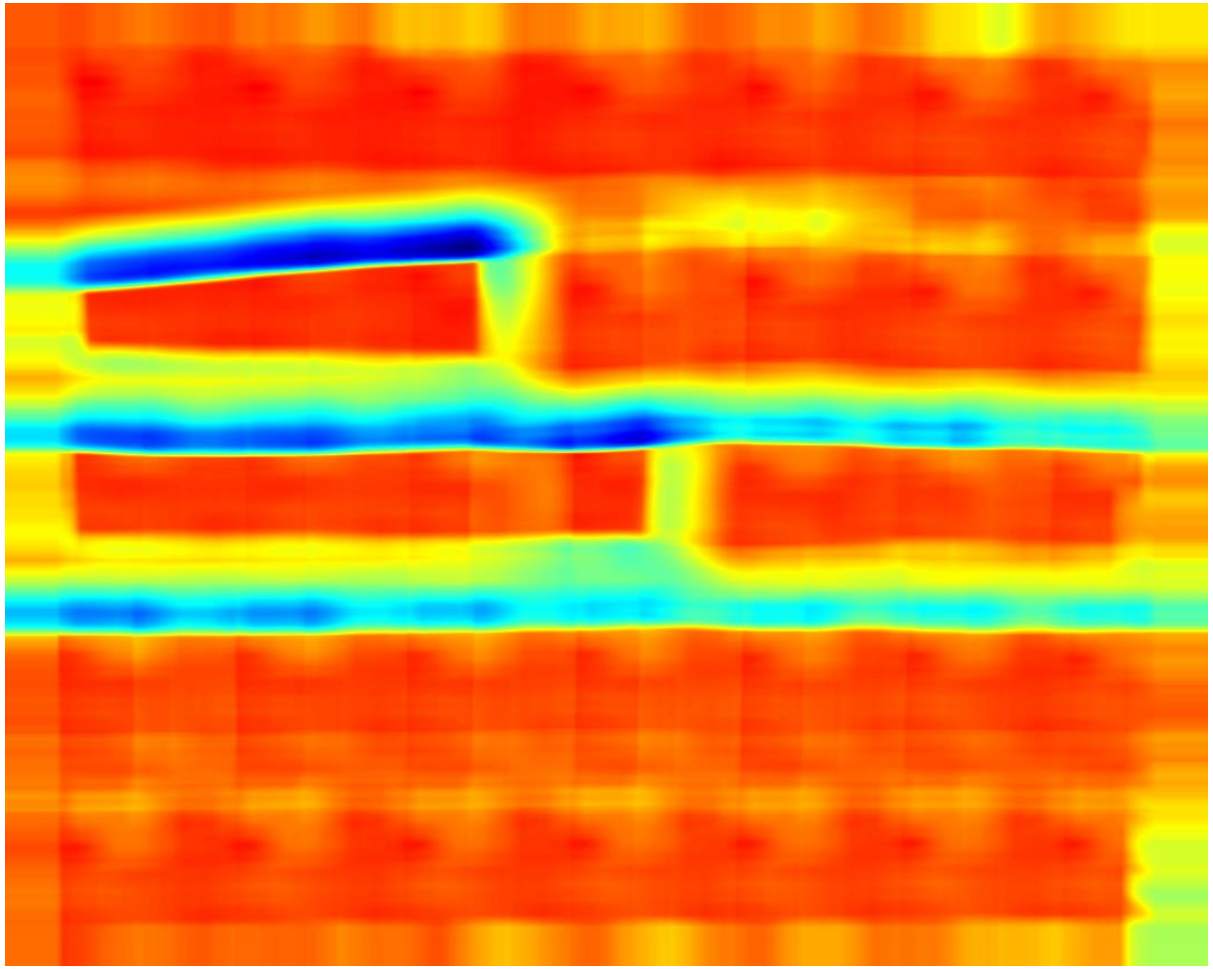


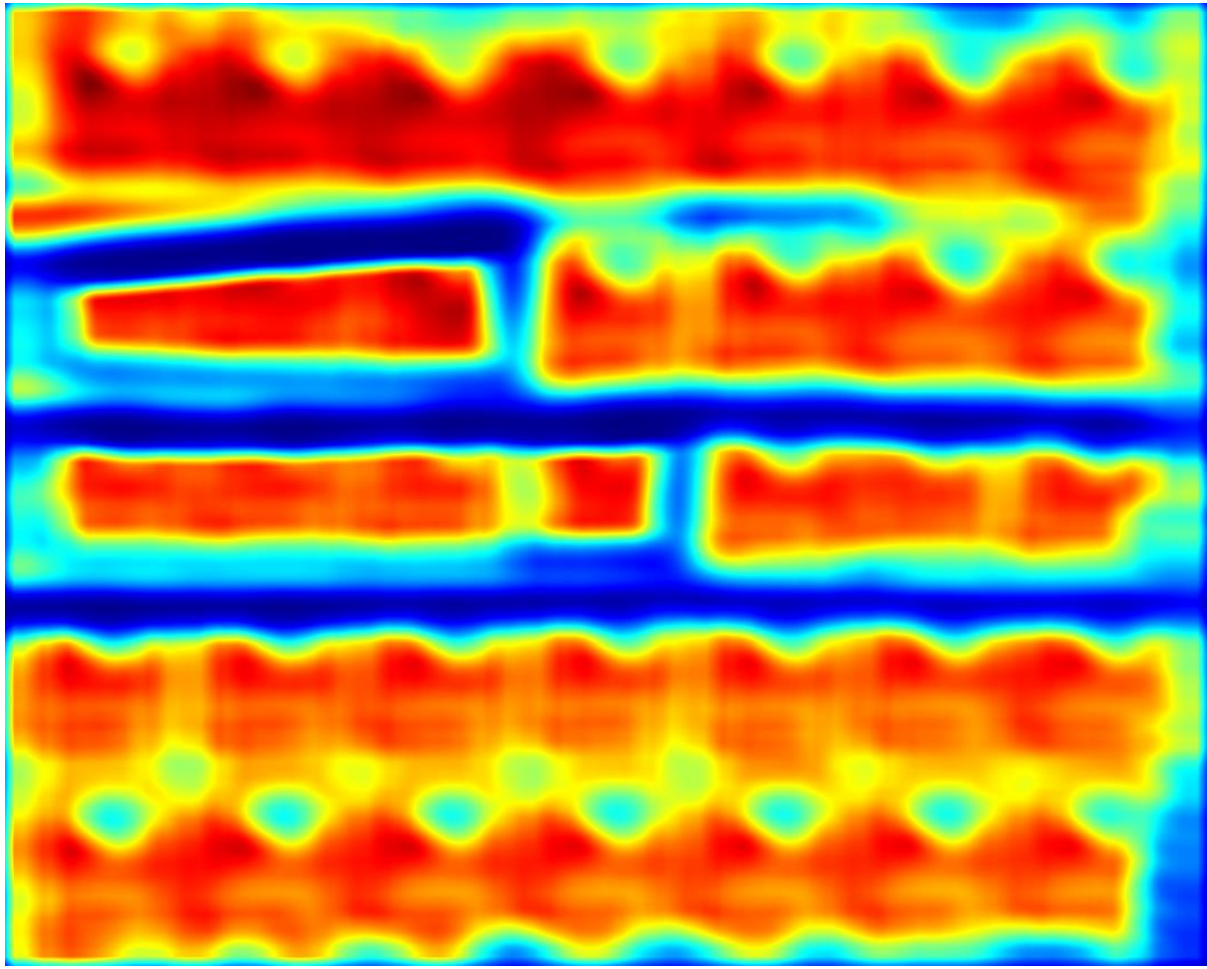
Template Matching: TM_CCORR



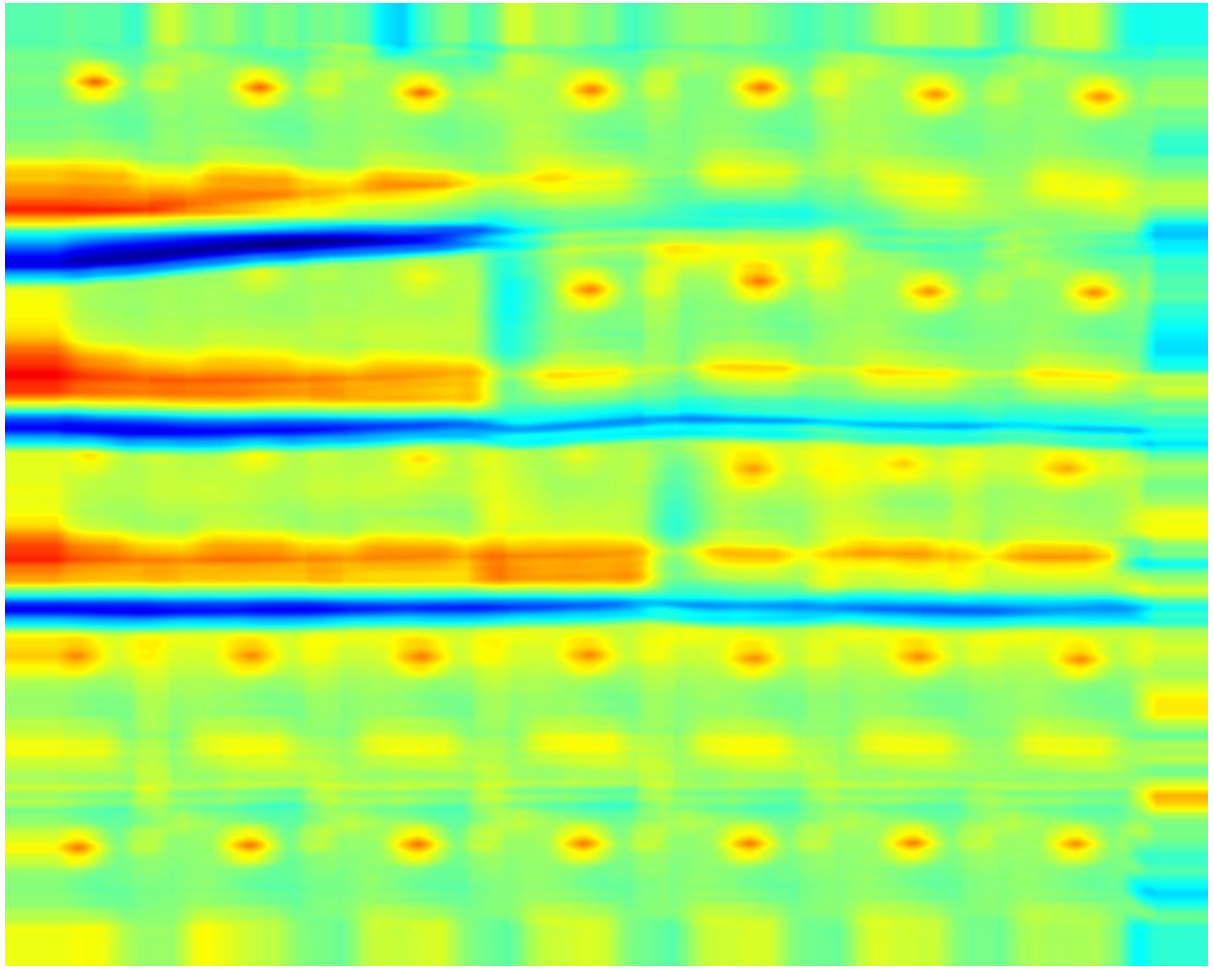


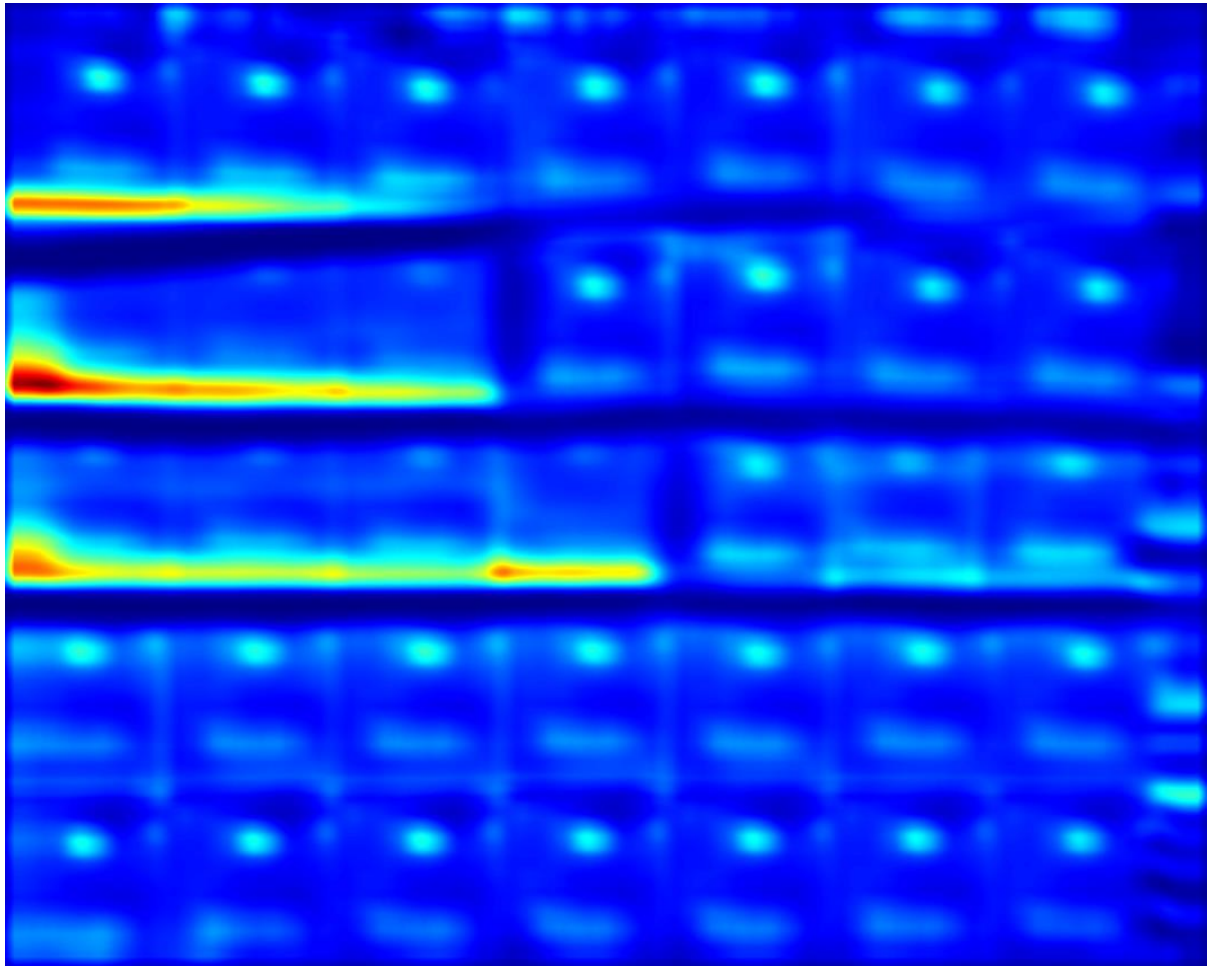
Template Matching: TM_CCORR_NORMED



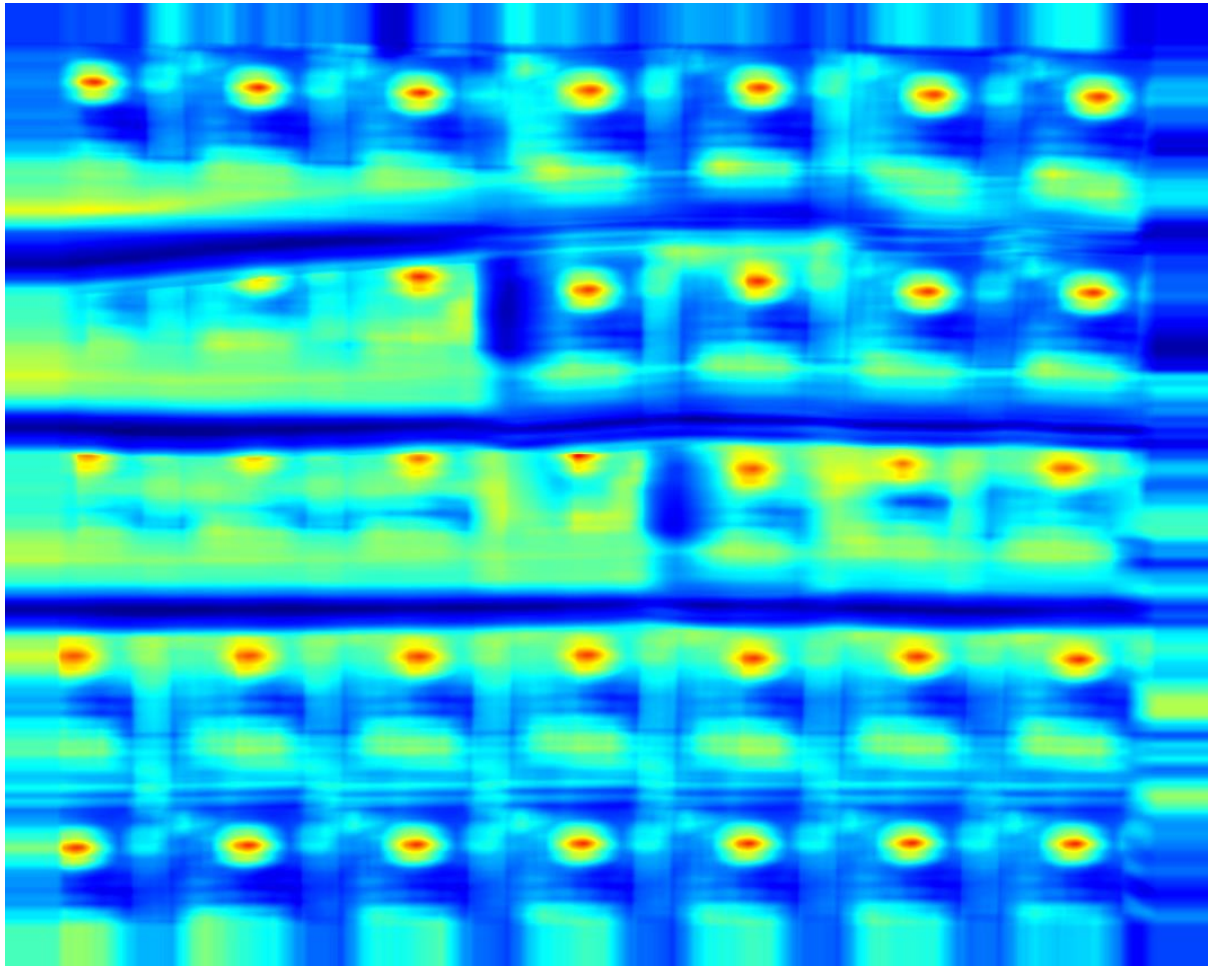


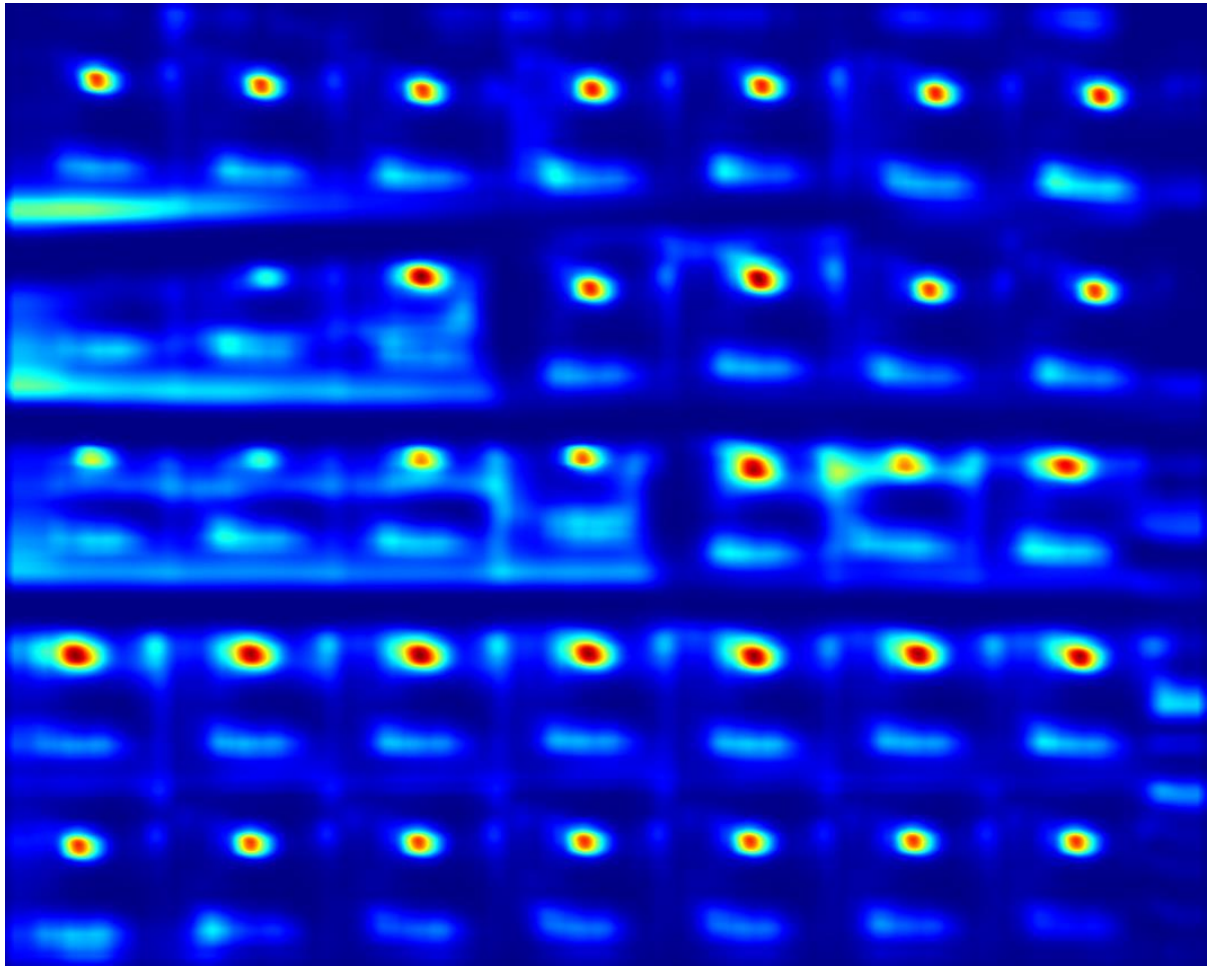
Template Matching: TM_CCOEFF





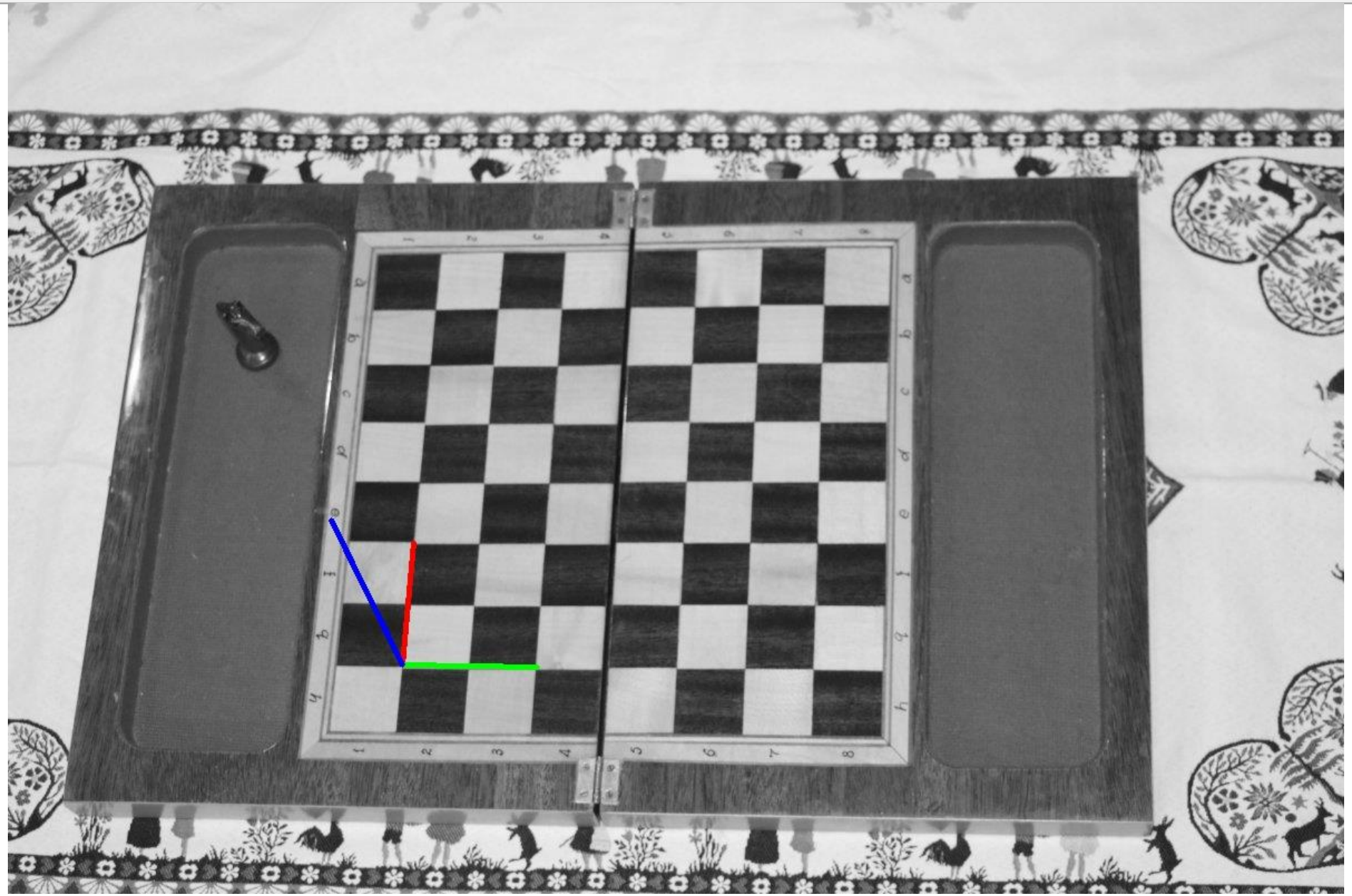
Template Matching: TM_CCOEFF_NORMED





Pose estimation

Pose estimation



Part 5: Computer Vision Applications

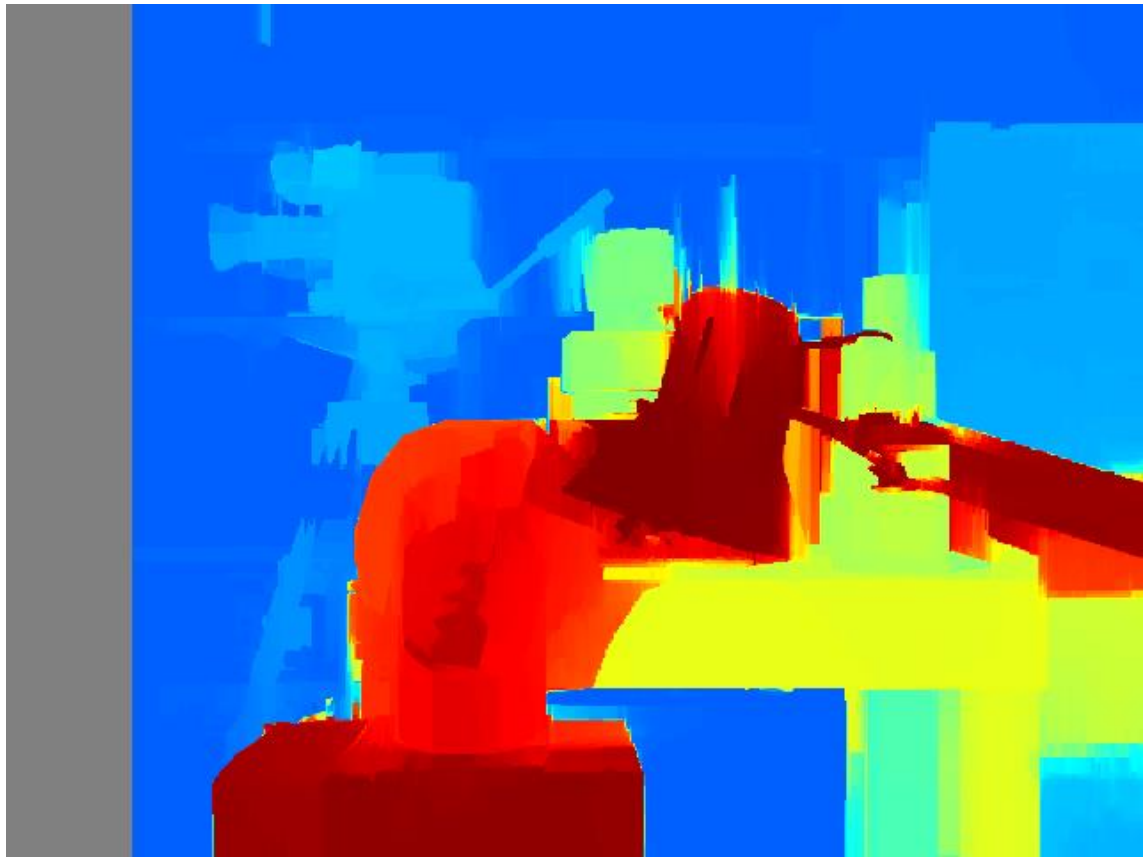
Stereo Vision
Face Swap
Face Filter
Style Transfer

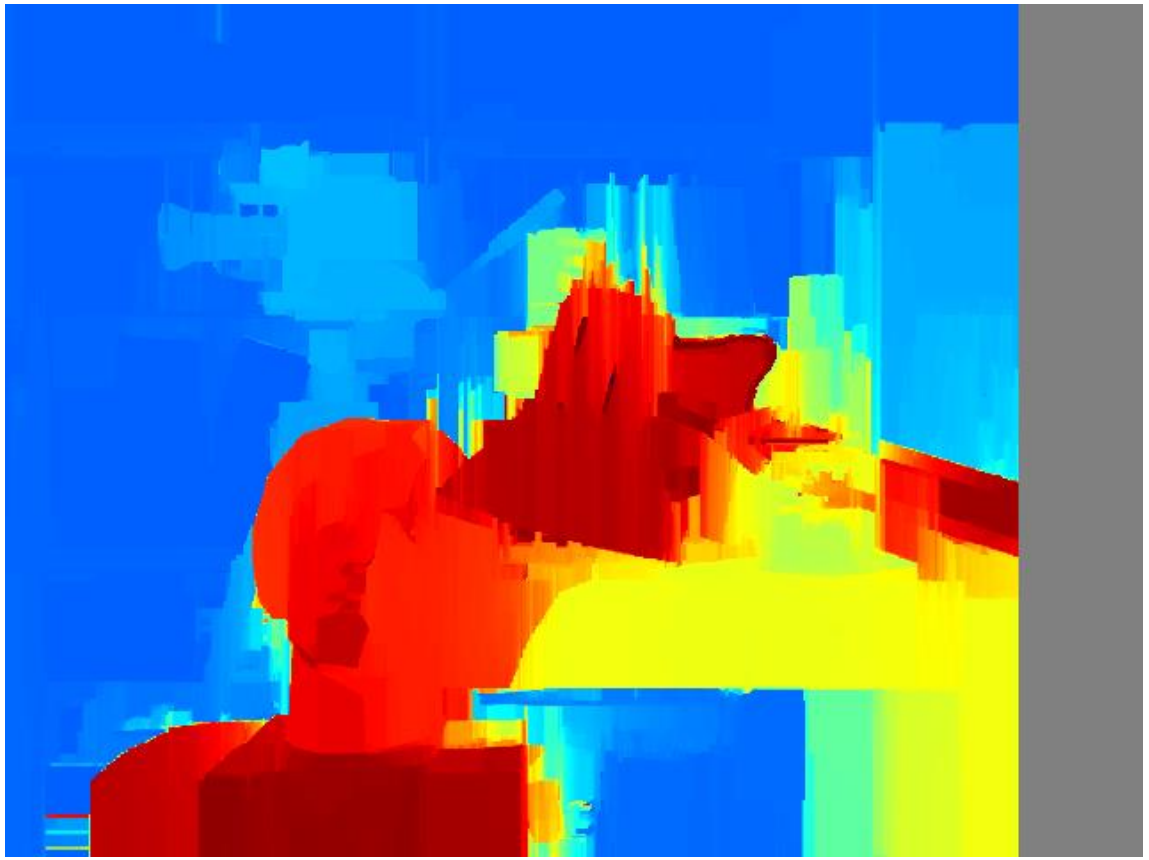
Stereo Vision

Stereo vision

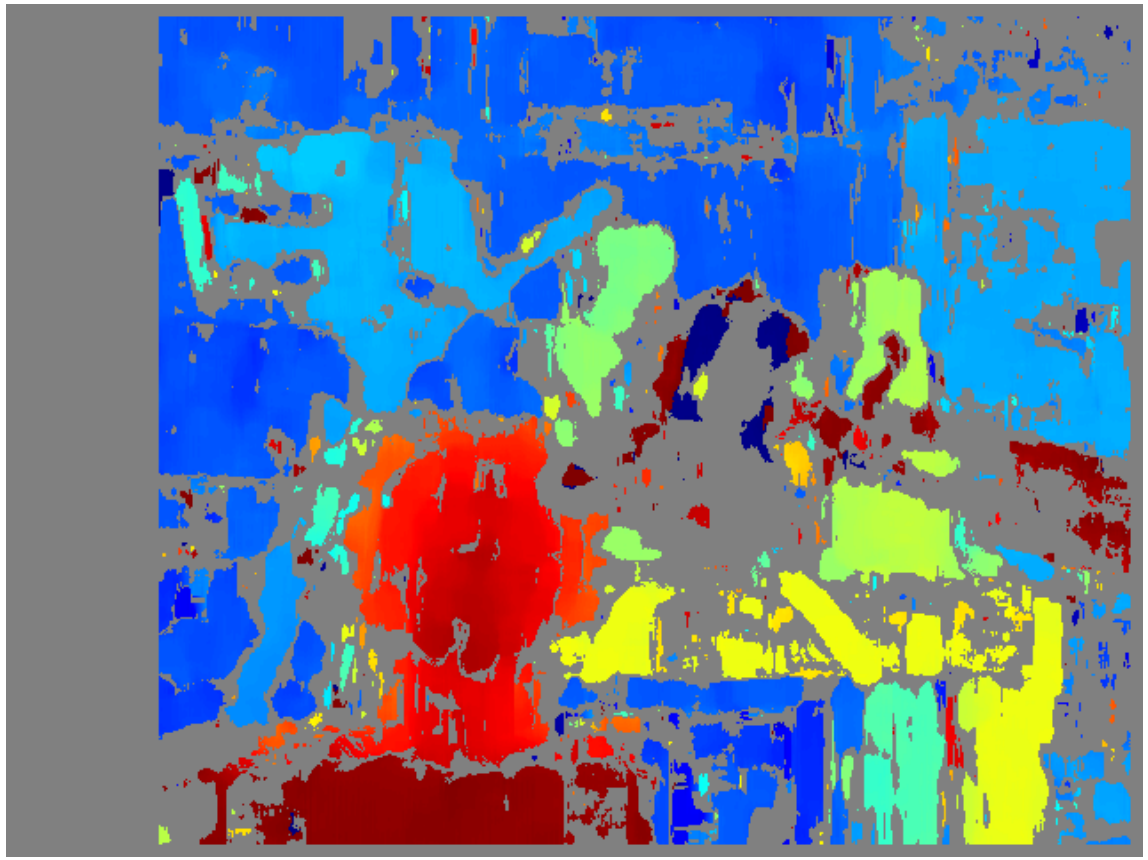
Stereo vision

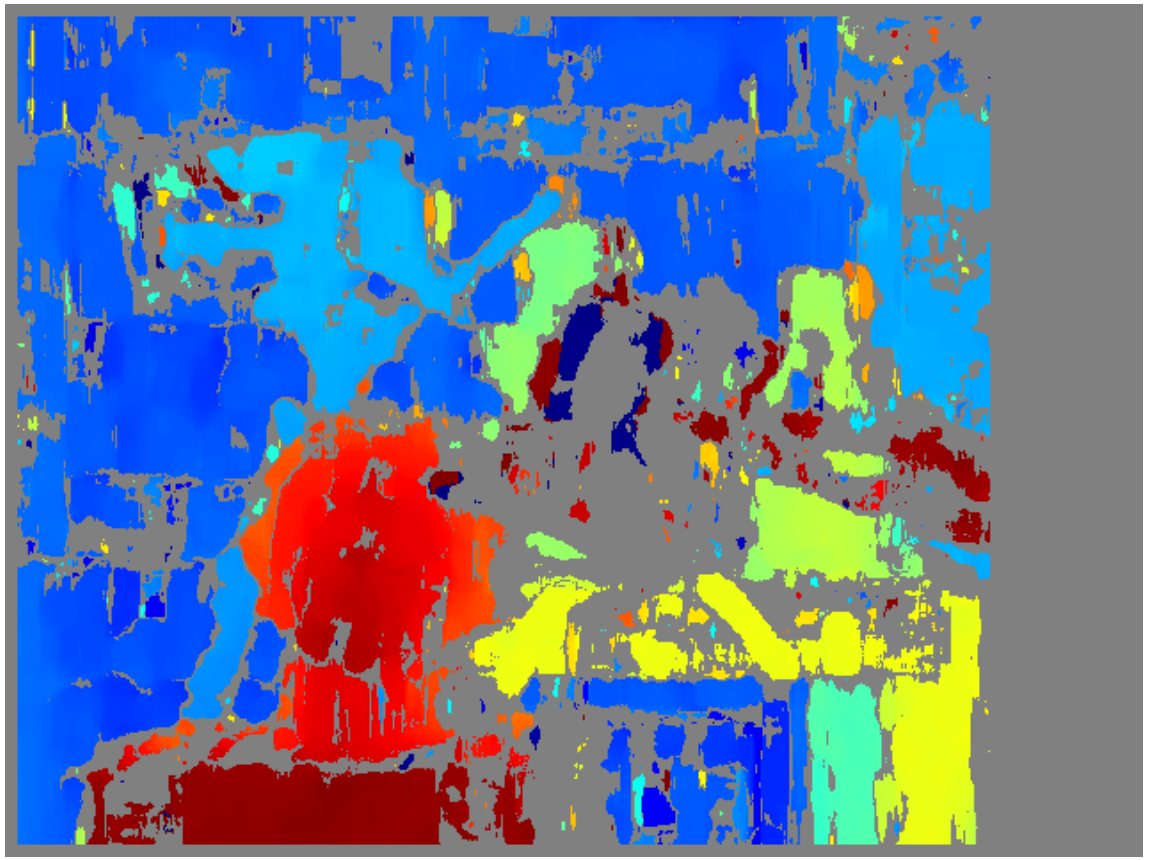
Stereo vision





Stereo vision





Stereo vision

```
def get_disparity_v_01(imgL, imgR, disp_v1, disp_v2, disp_h1, disp_h2):
    window_size = 7
    left_matcher = cv2.StereoSGBM_create(
        minDisparity=disp_h1,
        numDisparities=int(0 + (disp_h2 - disp_h1) / 16) * 16,
        blockSize=5,
        P1=8 * 3 * window_size ** 2,
        P2=32 * 3 * window_size ** 2,
        disp12MaxDiff=disp_h2,
        uniquenessRatio=15,
        speckleWindowSize=0,
        speckleRange=2,
        preFilterCap=63,
        mode=cv2.STEREO_SGBM_MODE_SGBM_3WAY
    )

    right_matcher = cv2.ximgproc.createRightMatcher(left_matcher)
    dispr = right_matcher.compute(imgR, imgL)
    displ = left_matcher.compute(imgL, imgR)

    wls_filter = cv2.ximgproc.createDisparityWLSFilter(matcher_left=left_matcher)
    wls_filter.setLambda(80000)
    wls_filter.setSigmaColor(1.2)

    filteredImg_L = wls_filter.filter(displ, imgL, None, dispr)
    wls_filter = cv2.ximgproc.createDisparityWLSFilter(matcher_left=right_matcher)
    filteredImg_R = wls_filter.filter(dispr, imgR, None, displ)
```

Stereo vision

```
def get_disparity_v_02(imgL, imgR, disp_v1, disp_v2, disp_h1, disp_h2):
    max = numpy.maximum(math.fabs(disp_h1), math.fabs(disp_h2))
    levels = int(1 + (max) / 16) * 16
    stereo = cv2.StereoBM_create(numDisparities=levels, blockSize=15)
    displ = stereo.compute(imgL, imgR)

    dispr = numpy.flip(stereo.compute(numpy.flip(imgR, axis=1), numpy.flip(imgL, axis=1)), axis=1)
    return -displ / 16, -dispr / 16
```