BME646 and ECE60146: Homework 8

Spring 2025

Completed by

Talha Ibn Mahmud

tibnmahm@purdue.edu

34722789

Due Date: Sunday, March 25, 2025, 11:59pm

Section 3 and 4: Programming Tasks

For this section I introduced the ASPP (Atrous Spatial Pyramid Pooling) layer at the bottleneck of the mUNet and also introduced the Dice Loss to the model for better convergence.

While running the script, I found out an issue quite like this piazza post. However, the solution mentioned there did not work for me, and I had to run every model for the first time by deleting the checkpoint file. This is one of the main reasons my simulation took this much time.

Executing the Script with Modified the Modified Model ASPP at the Bottleneck

For the ASPP design I mainly followed the design of this piazza post. I concatenated the output of 4 different convolutions: 1x1 kernel without dilation, 3x3 kernel with dilation=2, 3x3 kernel with dilation=4, and 3x3 kernel with dilation=6. After concatenating, I passed the output to a final convolutional layer. This is the code snippet from my customUnet (which is basically the same mUNet with the ASPP module).

```
  # I created a custom mUNet here to introduce the ASPP module at the bottleneck
  class CustomUnet(DLStudio.SemanticSegmentation.mUNet):
      def __init__(self, skip_connections=True, depth=16):
        super(CustomUnet, self).__init__(skip_connections=True, depth=depth)
        self.depth = depth // 2
        self.conv_in = nn.Conv2d(3, 64, 3, padding=1)
        ##  For the DN arm of the U:
        self.bn1DN  = nn.BatchNorm2d(64)
        self.bn2DN  = nn.BatchNorm2d(128)
        self.skip64DN_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip64DN_arr.append(SkipBlockDN(64, 64,
skip_connections=skip_connections))
        self.skip64dsDN = SkipBlockDN(64, 64,   downsample=True,
skip_connections=skip_connections)
        self.skip64to128DN = SkipBlockDN(64, 128, skip_connections=skip_connections )
        self.skip128DN_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip128DN_arr.append(SkipBlockDN(128, 128,
skip_connections=skip_connections))
        self.skip128dsDN = SkipBlockDN(128,128, downsample=True,
skip_connections=skip_connections)
        '''
        Start of the ASPP Module. I got the main instruction from this piazza post:
        https://piazza.com/class/m5qxf8zm2ds2gf/post/275
        In short, concatenated the output of 4 different convolutions:
        1x1 kernel without dilation
        3x3 kernel with dilation=2
        3x3 kernel with dilation=4
        3x3 kernel with dilation=6
        Finally passed through the concatenated output to another convolutional layer
        to regain the expected output shape.
        '''
        self.aspp_conv1 = nn.Conv2d(128, 128, 1, padding=0)
        self.aspp_conv2 = nn.Conv2d(128, 128, 3, padding=2, dilation=2)
        self.aspp_conv3 = nn.Conv2d(128, 128, 3, padding=4, dilation=4)
        self.aspp_conv4 = nn.Conv2d(128, 128, 3, padding=6, dilation=6)
        self.aspp_final_conv = nn.Conv2d(512, 128, 1)
```

```
        ##  For the UP arm of the U:
        self.bn1UP  = nn.BatchNorm2d(128)
        self.bn2UP  = nn.BatchNorm2d(64)
        self.skip64UP_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip64UP_arr.append(SkipBlockUP(64, 64,
skip_connections=skip_connections))
        self.skip64usUP = SkipBlockUP(64, 64, upsample=True,
skip_connections=skip_connections)
        self.skip128to64UP = SkipBlockUP(128, 64, skip_connections=skip_connections )
        self.skip128UP_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip128UP_arr.append(SkipBlockUP(128, 128,
skip_connections=skip_connections))
        self.skip128usUP = SkipBlockUP(128,128, upsample=True,
skip_connections=skip_connections)
        self.conv_out = nn.ConvTranspose2d(64, 5, 3,
stride=2,dilation=2,output_padding=1,padding=2)

      def forward(self, x):
        ##  Going down to the bottom of the U:
        x = nn.MaxPool2d(2,2)(nn.functional.relu(self.conv_in(x)))
        for i,skip64 in enumerate(self.skip64DN_arr[:self.depth//4]):
            x = skip64(x)

        num_channels_to_save1 = x.shape[1] // 2
        save_for_upside_1 = x[:,:num_channels_to_save1,:,:].clone()
        x = self.skip64dsDN(x)
        for i,skip64 in enumerate(self.skip64DN_arr[self.depth//4:]):
            x = skip64(x)
        x = self.bn1DN(x)
        num_channels_to_save2 = x.shape[1] // 2
        save_for_upside_2 = x[:,:num_channels_to_save2,:,:].clone()
        x = self.skip64to128DN(x)
        for i,skip128 in enumerate(self.skip128DN_arr[:self.depth//4]):
            x = skip128(x)

        x = self.bn2DN(x)
        num_channels_to_save3 = x.shape[1] // 2
        save_for_upside_3 = x[:,:num_channels_to_save3,:,:].clone()
        for i,skip128 in enumerate(self.skip128DN_arr[self.depth//4:]):
            x = skip128(x)
        x = self.skip128dsDN(x)

        '''
        Designing the ASPP:
        aspp1,2,3,4 means no dilation, dilation=2,4,6 respectively.
        aspp_concat concatenates these four outputs which then passes
```

```
        through the final conv layer.
        '''
        aspp1 = self.aspp_conv1(x)
        aspp2 = self.aspp_conv2(x)
        aspp3 = self.aspp_conv3(x)
        aspp4 = self.aspp_conv4(x)
        aspp_concat = torch.cat((aspp1, aspp2, aspp3, aspp4), dim=1)
        x = self.aspp_final_conv(aspp_concat)

        ## Coming up from the bottom of U on the other side:
        x = self.skip128usUP(x)
        for i,skip128 in enumerate(self.skip128UP_arr[:self.depth//4]):
            x = skip128(x)
        x[:,:num_channels_to_save3,:,:] =  save_for_upside_3
        x = self.bn1UP(x)
        for i,skip128 in enumerate(self.skip128UP_arr[:self.depth//4]):
            x = skip128(x)
        x = self.skip128to64UP(x)
        for i,skip64 in enumerate(self.skip64UP_arr[self.depth//4:]):
            x = skip64(x)
        x[:,:num_channels_to_save2,:,:] =  save_for_upside_2
        x = self.bn2UP(x)
        x = self.skip64usUP(x)
        for i,skip64 in enumerate(self.skip64UP_arr[:self.depth//4]):
            x = skip64(x)
        x[:,:num_channels_to_save1,:,:] =  save_for_upside_1
        x = self.conv_out(x)
        return x

model = CustomUnet(skip_connections=True, depth=16)
```

After completing the model architecture, I ran a set of performance analysis by varying the loss function and these hyperparameters: learning rate, batch size, and epoch. The values that I used for the hyperparameters are:

Learning rate: 1e-4 and 1e-5

Batch size: 4, 8, 16

Epoch: 6, 12, 20, 30

In short this is the list of analysis that I did for the PurdueShapes5MultiObjectDataset:

learning_rate	epoch	batch_size
1e-5	6	4
1e-5	6	16
1e-5	12	4
1e-4	6	4
1e-5	12	8

This is the list of analysis that I did for the MSCOCO:

learning_rate	epoch	batch_size
1e-5	20	4
1e-5	30	4
1e-4	20	4
1e-4	30	4

For both datasets, I repeated the same set for MSE only, Dice Only, MSE+ αDice where α was toggled between 1 and 100, the reason of this will be explained later on this report.

This is the complete code for analyzing the performance of the mUNet model with ASPP:

```
# Completed on March 25
# 34722789

'''
This code is heavily borrowed from the semantic_segmentation.py,
and the DLStudio SemanticSegmentation Class, especially mUNet.
'''

import random
import os, sys
import torch.nn as nn
import copy
import torch.optim as optim
import sys,os,os.path,glob
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as tvt
import torch.optim as optim
import numpy as np
from PIL import ImageFilter
import numbers
import re
import math
import random
import copy
import matplotlib.pyplot as plt
import gzip
import pickle
import pymsgbox
import time
import logging
from DLStudio import *
from multiprocessing import freeze_support


'''
I will be changing the learning rate, batch size, and epoch to
find the apparently best and worst combination for optimum performance

learning_rate      epoch       batch_size
-------------      -----       ----------
1e-5                6              4
1e-5                6              16
1e-5                12             4
```

```
1e-4                6               4
1e-5                12              8
'''


if __name__ == "__main__":
  freeze_support()

  dls = DLStudio(
  #                  dataroot = "/home/kak/ImageDatasets/PurdueShapes5MultiObject/",
                     dataroot = "./data/",
                     image_size = [64,64],
                     path_saved_model = "./saved_model",
                     momentum = 0.9,
                     learning_rate = 1e-4,
                     epochs = 12,
                     batch_size = 8,
                     classes = ('rectangle','triangle','disk','oval','star'),
                     use_gpu = True,
                )

  segmenter = DLStudio.SemanticSegmentation(
                     dl_studio = dls,
                     max_num_objects = 5,
                )

  dataserver_train = DLStudio.SemanticSegmentation.PurdueShapes5MultiObjectDataset(
                             train_or_test = 'train',
                             dl_studio = dls,
                             segmenter = segmenter,
                             dataset_file = "PurdueShapes5MultiObject-10000-train.gz",
                            )
  dataserver_test = DLStudio.SemanticSegmentation.PurdueShapes5MultiObjectDataset(
                             train_or_test = 'test',
                             dl_studio = dls,
                             segmenter = segmenter,
                             dataset_file = "PurdueShapes5MultiObject-1000-test.gz"
                            )
  segmenter.dataserver_train = dataserver_train
  segmenter.dataserver_test = dataserver_test

  segmenter.load_PurdueShapes5MultiObject_dataset(dataserver_train, dataserver_test)

  class SkipBlockDN(nn.Module):
      """
      This class for the skip connections in the downward leg of the "U"

      Class Path:   DLStudio  ->  SemanticSegmentation  ->  SkipBlockDN
```

```
        """
        def __init__(self, in_ch, out_ch, downsample=False, skip_connections=True):
            super(SkipBlockDN, self).__init__()
            self.downsample = downsample
            self.skip_connections = skip_connections
            self.in_ch = in_ch
            self.out_ch = out_ch
            self.convo1 = nn.Conv2d(in_ch, out_ch, 3, stride=1, padding=1)
            self.convo2 = nn.Conv2d(in_ch, out_ch, 3, stride=1, padding=1)
            self.bn1 = nn.BatchNorm2d(out_ch)
            self.bn2 = nn.BatchNorm2d(out_ch)
            if downsample:
                self.downsampler = nn.Conv2d(in_ch, out_ch, 1, stride=2)
        def forward(self, x):
            identity = x
            out = self.convo1(x)
            out = self.bn1(out)
            out = nn.functional.relu(out)
            if self.in_ch == self.out_ch:
                out = self.convo2(out)
                out = self.bn2(out)
                out = nn.functional.relu(out)
            if self.downsample:
                out = self.downsampler(out)
                identity = self.downsampler(identity)
            if self.skip_connections:
                if self.in_ch == self.out_ch:
                    out = out + identity
                else:
                    out = out + torch.cat((identity, identity), dim=1)
            return out

    class SkipBlockUP(nn.Module):
        """
        This class is for the skip connections in the upward leg of the "U"

        Class Path:   DLStudio  ->  SemanticSegmentation  ->  SkipBlockUP
        """
        def __init__(self, in_ch, out_ch, upsample=False, skip_connections=True):
            super(SkipBlockUP, self).__init__()
            self.upsample = upsample
            self.skip_connections = skip_connections
            self.in_ch = in_ch
            self.out_ch = out_ch
            self.convoT1 = nn.ConvTranspose2d(in_ch, out_ch, 3, padding=1)
            self.convoT2 = nn.ConvTranspose2d(in_ch, out_ch, 3, padding=1)
            self.bn1 = nn.BatchNorm2d(out_ch)
            self.bn2 = nn.BatchNorm2d(out_ch)
```

```
            if upsample:
                self.upsampler = nn.ConvTranspose2d(in_ch, out_ch, 1, stride=2,
dilation=2, output_padding=1, padding=0)
        def forward(self, x):
            identity = x
            out = self.convoT1(x)
            out = self.bn1(out)
            out = nn.functional.relu(out)
            out  =  nn.ReLU(inplace=False)(out)
            if self.in_ch == self.out_ch:
                out = self.convoT2(out)
                out = self.bn2(out)
                out = nn.functional.relu(out)
            if self.upsample:
                out = self.upsampler(out)
                identity = self.upsampler(identity)
            if self.skip_connections:
                if self.in_ch == self.out_ch:
                    out = out + identity
                else:
                    out = out + identity[:,self.out_ch:,:,:]
            return out
    # I created a custom mUNet here to introduce the ASPP module at the bottleneck
    class CustomUnet(DLStudio.SemanticSegmentation.mUNet):
        def __init__(self, skip_connections=True, depth=16):
            super(CustomUnet, self).__init__(skip_connections=True, depth=depth)
            self.depth = depth // 2
            self.conv_in = nn.Conv2d(3, 64, 3, padding=1)
            ##  For the DN arm of the U:
            self.bn1DN  = nn.BatchNorm2d(64)
            self.bn2DN  = nn.BatchNorm2d(128)
            self.skip64DN_arr = nn.ModuleList()
            for i in range(self.depth):
                self.skip64DN_arr.append(SkipBlockDN(64, 64,
skip_connections=skip_connections))
            self.skip64dsDN = SkipBlockDN(64, 64,   downsample=True,
skip_connections=skip_connections)
            self.skip64to128DN = SkipBlockDN(64, 128, skip_connections=skip_connections )
            self.skip128DN_arr = nn.ModuleList()
            for i in range(self.depth):
                self.skip128DN_arr.append(SkipBlockDN(128, 128,
skip_connections=skip_connections))
            self.skip128dsDN = SkipBlockDN(128,128, downsample=True,
skip_connections=skip_connections)
            '''
            Start of the ASPP Module. I got the main instruction from this piazza post:
            https://piazza.com/class/m5qxf8zm2ds2gf/post/275
            In short, concatenated the output of 4 different convolutions:
```

```
        1x1 kernel without dilation
        3x3 kernel with dilation=2
        3x3 kernel with dilation=4
        3x3 kernel with dilation=6
        Finally passed through the concatenated output to another convolutional layer
        to regain the expected output shape.
        '''
        self.aspp_conv1 = nn.Conv2d(128, 128, 1, padding=0)
        self.aspp_conv2 = nn.Conv2d(128, 128, 3, padding=2, dilation=2)
        self.aspp_conv3 = nn.Conv2d(128, 128, 3, padding=4, dilation=4)
        self.aspp_conv4 = nn.Conv2d(128, 128, 3, padding=6, dilation=6)
        self.aspp_final_conv = nn.Conv2d(512, 128, 1)

        ##  For the UP arm of the U:
        self.bn1UP  = nn.BatchNorm2d(128)
        self.bn2UP  = nn.BatchNorm2d(64)
        self.skip64UP_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip64UP_arr.append(SkipBlockUP(64, 64,
skip_connections=skip_connections))
        self.skip64usUP = SkipBlockUP(64, 64, upsample=True,
skip_connections=skip_connections)
        self.skip128to64UP = SkipBlockUP(128, 64, skip_connections=skip_connections )
        self.skip128UP_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip128UP_arr.append(SkipBlockUP(128, 128,
skip_connections=skip_connections))
        self.skip128usUP = SkipBlockUP(128,128, upsample=True,
skip_connections=skip_connections)
        self.conv_out = nn.ConvTranspose2d(64, 5, 3,
stride=2,dilation=2,output_padding=1,padding=2)

      def forward(self, x):
        ##  Going down to the bottom of the U:
        x = nn.MaxPool2d(2,2)(nn.functional.relu(self.conv_in(x)))
        for i,skip64 in enumerate(self.skip64DN_arr[:self.depth//4]):
            x = skip64(x)

        num_channels_to_save1 = x.shape[1] // 2
        save_for_upside_1 = x[:,:num_channels_to_save1,:,:].clone()
        x = self.skip64dsDN(x)
        for i,skip64 in enumerate(self.skip64DN_arr[self.depth//4:]):
            x = skip64(x)
        x = self.bn1DN(x)
        num_channels_to_save2 = x.shape[1] // 2
        save_for_upside_2 = x[:,:num_channels_to_save2,:,:].clone()
        x = self.skip64to128DN(x)
        for i,skip128 in enumerate(self.skip128DN_arr[:self.depth//4]):
```

```
            x = skip128(x)

        x = self.bn2DN(x)
        num_channels_to_save3 = x.shape[1] // 2
        save_for_upside_3 = x[:,:num_channels_to_save3,:,:].clone()
        for i,skip128 in enumerate(self.skip128DN_arr[self.depth//4:]):
            x = skip128(x)
        x = self.skip128dsDN(x)

        '''
        Designing the ASPP:
        aspp1,2,3,4 means no dilation, dilation=2,4,6 respectively.
        aspp_concat concatenates these four outputs which then passes
        through the final conv layer.
        '''
        aspp1 = self.aspp_conv1(x)
        aspp2 = self.aspp_conv2(x)
        aspp3 = self.aspp_conv3(x)
        aspp4 = self.aspp_conv4(x)
        aspp_concat = torch.cat((aspp1, aspp2, aspp3, aspp4), dim=1)
        x = self.aspp_final_conv(aspp_concat)

        ## Coming up from the bottom of U on the other side:
        x = self.skip128usUP(x)
        for i,skip128 in enumerate(self.skip128UP_arr[:self.depth//4]):
            x = skip128(x)
        x[:,:num_channels_to_save3,:,:] =  save_for_upside_3
        x = self.bn1UP(x)
        for i,skip128 in enumerate(self.skip128UP_arr[:self.depth//4]):
            x = skip128(x)
        x = self.skip128to64UP(x)
        for i,skip64 in enumerate(self.skip64UP_arr[self.depth//4:]):
            x = skip64(x)
        x[:,:num_channels_to_save2,:,:] =  save_for_upside_2
        x = self.bn2UP(x)
        x = self.skip64usUP(x)
        for i,skip64 in enumerate(self.skip64UP_arr[:self.depth//4]):
            x = skip64(x)
        x[:,:num_channels_to_save1,:,:] =  save_for_upside_1
        x = self.conv_out(x)
        return x

  model = CustomUnet(skip_connections=True, depth=16)
  number_of_learnable_params = sum(p.numel() for p in model.parameters() if
p.requires_grad)
  print("\n\nThe number of learnable parameters in the model: %d\n" %
number_of_learnable_params)
```

```
  def run_code_for_testing_semantic_segmentation(net):
    net.load_state_dict(torch.load(dls.path_saved_model))
    batch_size = dls.batch_size
    image_size = dls.image_size
    max_num_objects = segmenter.max_num_objects
    with torch.no_grad():
        for i, data in enumerate(segmenter.test_dataloader):
            im_tensor, mask_tensor, bbox_tensor = data['image'], data['mask_tensor'],
data['bbox_tensor']
            if i % 50 == 0:
                print("\n\n\n\nShowing output for test batch %d: " % (i+1))
                count = i
                outputs = net(im_tensor)

                output_bw_tensor = torch.zeros(batch_size, 1, image_size[0],
image_size[1], dtype=torch.float32)
                for image_idx in range(batch_size):
                    for layer_idx in range(max_num_objects):
                        for m in range(image_size[0]):
                            for n in range(image_size[1]):
                                output_bw_tensor[image_idx, 0, m, n] =
torch.max(outputs[image_idx, :, m, n])
                display_tensor = torch.zeros(7 * batch_size, 3, image_size[0],
image_size[1], dtype=torch.float32)
                for idx in range(batch_size):
                    for bbox_idx in range(max_num_objects):
                        bb_tensor = bbox_tensor[idx, bbox_idx]
                        for k in range(max_num_objects):
                            i1 = int(bb_tensor[k][1])
                            i2 = int(bb_tensor[k][3])
                            j1 = int(bb_tensor[k][0])
                            j2 = int(bb_tensor[k][2])
                            output_bw_tensor[idx, 0, i1:i2, j1] = 255
                            output_bw_tensor[idx, 0, i1:i2, j2] = 255
                            output_bw_tensor[idx, 0, i1, j1:j2] = 255
                            output_bw_tensor[idx, 0, i2, j1:j2] = 255
                            im_tensor[idx, 0, i1:i2, j1] = 255
                            im_tensor[idx, 0, i1:i2, j2] = 255
                            im_tensor[idx, 0, i1, j1:j2] = 255
                            im_tensor[idx, 0, i2, j1:j2] = 255
                display_tensor[:batch_size, :, :, :] = output_bw_tensor
                display_tensor[batch_size:2*batch_size, :, :, :] = im_tensor

                os.makedirs("./mixed_results_lr4_e12_b8_s1/masks", exist_ok=True)
                for batch_im_idx in range(batch_size):
                    for mask_layer_idx in range(max_num_objects):
                        for i in range(image_size[0]):
                            for j in range(image_size[1]):
```

```
                                if mask_layer_idx == 0:
                                    if 25 < outputs[batch_im_idx, mask_layer_idx, i,
j] < 85:
                                        outputs[batch_im_idx, mask_layer_idx, i, j] =
255
                                    else:
                                        outputs[batch_im_idx, mask_layer_idx, i, j] =
50
                                elif mask_layer_idx == 1:
                                    if 65 < outputs[batch_im_idx, mask_layer_idx, i,
j] < 135:
                                        outputs[batch_im_idx, mask_layer_idx, i, j] =
255
                                    else:
                                        outputs[batch_im_idx, mask_layer_idx, i, j] =
50
                                elif mask_layer_idx == 2:
                                    if 115 < outputs[batch_im_idx, mask_layer_idx, i,
j] < 185:
                                        outputs[batch_im_idx, mask_layer_idx, i, j] =
255
                                    else:
                                        outputs[batch_im_idx, mask_layer_idx, i, j] =
50
                                elif mask_layer_idx == 3:
                                    if 165 < outputs[batch_im_idx, mask_layer_idx, i,
j] < 230:
                                        outputs[batch_im_idx, mask_layer_idx, i, j] =
255
                                    else:
                                        outputs[batch_im_idx, mask_layer_idx, i, j] =
50
                                elif mask_layer_idx == 4:
                                    if outputs[batch_im_idx, mask_layer_idx, i, j] >
210:
                                        outputs[batch_im_idx, mask_layer_idx, i, j] =
255
                                    else:
                                        outputs[batch_im_idx, mask_layer_idx, i, j] =
50

                        display_tensor[2*batch_size + batch_size*mask_layer_idx +
batch_im_idx, :, :, :] = outputs[batch_im_idx, mask_layer_idx, :, :]
                        # I added the following lines to save the masks
                        mask_save_path =
f"./mixed_results_lr4_e12_b8_s1/masks/batch_{count+1}_image_{batch_im_idx+1}_class_{ma
sk_layer_idx+1}.png"
```

```
                        mask_tensor = outputs[batch_im_idx, mask_layer_idx, :,
:].unsqueeze(0).unsqueeze(0)
                        torchvision.utils.save_image(mask_tensor, mask_save_path,
normalize=True)
                        print(f"Saved mask for batch {count+1}, image
{batch_im_idx+1}, class {mask_layer_idx+1} to {mask_save_path}")

                save_path =
f"./mixed_results_lr4_e12_b8_s1/output_batch_{count+1}.png"
                # Slightly changed the following line and added some to save the grid
as image instead of displaying it.
                # I ran all these code on a server, did not access to display.
                grid = torchvision.utils.make_grid(display_tensor, nrow=batch_size,
normalize=True, padding=2, pad_value=100)
                grid_np = grid.permute(1, 2, 0).cpu().numpy()
                plt.figure(figsize=(10, 10))
                plt.imshow(grid_np)
                plt.axis('off')
                plt.tight_layout()
                plt.savefig(save_path, dpi=300, bbox_inches='tight')
                plt.close()
                print(f"Saved visualization to {save_path}")

  def save_model(model, save_path):
    torch.save(model.state_dict(), save_path)
    print(f"Model saved to {save_path}")

  '''
  This is the required dice_function of which the skeleton was provided
  '''
  def dice_loss(preds: torch.Tensor, ground_truth: torch.Tensor, epsilon=1e-6):
    # Flattened the prediction and ground truth vector first
    # Got this idea from the first solution of Spring 2024 Page 4
    #
https://engineering.purdue.edu/DeepLearn/2_best_solutions/2024/Homeworks/HW7/2BestSolu
tions/1.pdf
    preds_flat = preds.view(preds.size(0), preds.size(1), -1)
    ground_truth_flat = ground_truth.view(ground_truth.size(0), ground_truth.size(1),
-1)
    # Step 1: Compute Dice Coefficient
    numerator = torch.sum(preds_flat * ground_truth_flat, dim=-1)
    denominator = torch.sum(preds_flat ** 2, dim=-1) + torch.sum(ground_truth_flat **
2, dim=-1)
    # Step 2: dice_coefficient = 2*numerator / (denominator + epsilon)
    dice_coefficient = (2.0 * numerator) / (denominator + epsilon)  # Shape:
[batch_size, num_classes]
    # Step 3: Compute dice_loss  = 1 - dice_coefficient
    dice_loss = 1.0 - torch.mean(dice_coefficient)
```

```
    return dice_loss

  def run_code_for_training_for_semantic_segmentation(net):
    filename_for_out1 = "performance_numbers_" + str(dls.epochs) + ".txt"
    FILE1 = open(filename_for_out1, 'w')
    net = copy.deepcopy(net)
    net = net.to(dls.device)
    criterion1 = nn.MSELoss()
    optimizer = optim.SGD(net.parameters(),
                          lr=dls.learning_rate, momentum=dls.momentum)
    start_time = time.perf_counter()
    # Used these to generate loss curve
    losses = []
    iterations = []
    for epoch in range(dls.epochs):
        print("")
        running_loss_segmentation = 0.0
        for i, data in enumerate(segmenter.train_dataloader):
            im_tensor, mask_tensor, bbox_tensor = data['image'], data['mask_tensor'],
data['bbox_tensor']
            im_tensor = im_tensor.to(dls.device)
            mask_tensor = mask_tensor.type(torch.FloatTensor)
            mask_tensor = mask_tensor.to(dls.device)
            bbox_tensor = bbox_tensor.to(dls.device)
            optimizer.zero_grad()
            output = net(im_tensor)
            # The following three lines are for MSE loss, Dice Loss, and MSE+Dice loss
respectively.
            # segmentation_loss = criterion1(output, mask_tensor)
            # segmentation_loss = dice_loss(output, mask_tensor)
            segmentation_loss = criterion1(output, mask_tensor) + 1*dice_loss(output,
mask_tensor) # toggled between 1 and 100
            segmentation_loss.backward()
            optimizer.step()
            running_loss_segmentation += segmentation_loss.item()
            if i % 500 == 499:
                current_time = time.perf_counter()
                elapsed_time = current_time - start_time
                avg_loss_segmentation = running_loss_segmentation / float(500)
                print("[epoch=%d/%d, iter=%4d  elapsed_time=%3d secs]   MSE loss:
%.3f" %
                      (epoch+1, dls.epochs, i+1, elapsed_time, avg_loss_segmentation))
                FILE1.write("%.3f\n" % avg_loss_segmentation)
                FILE1.flush()
                # Extra lines that I added to plot the loss curve
                losses.append(avg_loss_segmentation)
                iterations.append(len(losses))
```

```
                running_loss_segmentation = 0.0
    FILE1.close()
    print("\nFinished Training\n")
    # Plot loss vs. iteration
    plt.figure(figsize=(10, 6))
    plt.plot(iterations, losses, label="MSE + Dice Loss", color="blue")
    plt.title("Mixed_Result_lr4_e12_b8_s1", fontsize=16)
    plt.xlabel("Iteration", fontsize=14)
    plt.ylabel("MSE + Dice Loss", fontsize=14)
    plt.grid(True)
    plt.legend(fontsize=12)
    plt.tight_layout()
    save_path = "./loss_vs_iteration_with_aspp_with_lr4_e12_b8_s1_mixed.png"
    plt.savefig(save_path, dpi=1200)
    plt.close()
    print(f"Saved loss vs. iteration plot to {save_path}")
    save_model(net, dls.path_saved_model)

run_code_for_training_for_semantic_segmentation(model)
run_code_for_testing_semantic_segmentation(model)
```

Dataset Creation for MSCOCO

This is the segment I had to do most of the brainstorming for this homework. As I could not find an explicit guideline how to create a similar dataset to PurdueShapes5MultiObjectDataset, I borrowed some ideas from previous years solution 2. Also posted in the piazza. As per the suggestion as well as previous solution and largely by looking closely at the structure of the PurdueShapes5MultiObjectDataset class line 4660 to line 4789, I modified my own idea to generate a dataset. I posted the idea in this piazza post which the TA approved. Here is my workflow:

- First, I filtered through the COCO dataset to choose only those RGB images that fall into either pizza/cat/bus. But now, additional criteria where the mask dimension should be a minimum of 200x200.
- For each of the images, I generated a 3D NumPy mask array (with a default 0-array mask for all the categories) for all the images to make sure each image always has 3 masks irrespective of the actual number of categories present in the image.
- I also created a dictionary to list the bounding box coordinates of the 3 categories. Then I saved all this info as a npz file for individual images.
- Finally, in the model training and testing code, I loaded those npz files by creating a custom dataset class and then used them for training and validation after slight preprocessing.

Here is the code that I used to generate the npz files from the COCO dataset:

```
# Completed on March 25
# 34722789

import os
import cv2
import numpy as np
from PIL import Image
from pycocotools.coco import COCO
import json

# ann_file = './HW4/data/annotations/instances_val2014.json'
# image_dir = './HW4/data/val2014'
# output_dir = './dataset_with_test_masks'

ann_file = './HW4/data/annotations/instances_train2014.json'
image_dir = './HW4/data/train2014'
output_dir = './dataset_with_train_masks'

os.makedirs(output_dir, exist_ok=True)

# Kept the first three labels of the PurdueShapes5MultiObjectDataset
label_map = {'pizza': 50, 'cat': 100, 'bus': 150}
```

```
coco = COCO(ann_file)
pizza_id = coco.getCatIds(catNms='pizza')[0]
cat_id = coco.getCatIds(catNms='cat')[0]
bus_id = coco.getCatIds(catNms='bus')[0]
target_cat_ids = [pizza_id, cat_id, bus_id]
pizza_imgs = set(coco.getImgIds(catIds=pizza_id))
cat_imgs = set(coco.getImgIds(catIds=cat_id))
bus_imgs = set(coco.getImgIds(catIds=bus_id))
all_img_ids = list(pizza_imgs.union(cat_imgs).union(bus_imgs))

for img_id in all_img_ids:
    img_info = coco.loadImgs(img_id)[0]
    ann_ids = coco.getAnnIds(imgIds=img_id, iscrowd=False)
    anns = coco.loadAnns(ann_ids)
    # Applying the additional criterion here: height and width each has to be >= 200
    valid_anns = [
        ann for ann in anns
        if ann['category_id'] in target_cat_ids and
            ann['bbox'][2] >= 200 and ann['bbox'][3] >= 200
    ]
    if not valid_anns:
        continue
    img_path = os.path.join(image_dir, img_info['file_name'])
    image = cv2.imread(img_path)

    # Ran into an issue before, probably one of the images in the dataset was
grayscale instead of RGB
    # Putting this RGB checking condition here to solve that issue
    if image is None or image.shape[2] != 3:
        continue
    height, width = image.shape[:2]
    resized_image = cv2.resize(image, (256, 256))
    R = resized_image[:, :, 0].flatten()
    G = resized_image[:, :, 1].flatten()
    B = resized_image[:, :, 2].flatten()
    # Initialize mask array and bounding box map
    mask_array = np.zeros((3, 256, 256), dtype=np.uint8)
    mask_val_to_bbox_map = {50: [], 100: [], 150: []}

    # Process each valid annotation
    for ann in valid_anns:
        cat_id = ann['category_id']
        cat_name = coco.loadCats(cat_id)[0]['name']
        mask_value = label_map[cat_name]
        mask = coco.annToMask(ann)
        resized_mask = cv2.resize(mask.astype(np.uint8), (256, 256),
interpolation=cv2.INTER_NEAREST)
        mask_layer_index = list(label_map.values()).index(mask_value)
```

```
        mask_array[mask_layer_index][resized_mask == 1] = mask_value
        x, y, w, h = map(int, ann['bbox'])
        original_width, original_height = img_info['width'], img_info['height']
        resized_width, resized_height = 256, 256
        x_resized = int(x * resized_width / original_width)
        y_resized = int(y * resized_height / original_height)
        w_resized = int(w * resized_width / original_width)
        h_resized = int(h * resized_height / original_height)
        # Assigned upper-left corner and lower-right corner similar to
PurdueShapes5MultiObjectDataset
        bbox = [x_resized, y_resized, x_resized + w_resized, y_resized + h_resized]
        mask_val_to_bbox_map[mask_value].append(bbox)

    output_filename = os.path.splitext(img_info['file_name'])[0] + '.npz'
    output_path = os.path.join(output_dir, output_filename)
    # Saving the data for individual image as npz file
    np.savez_compressed(
        output_path,
        R=R,
        G=G,
        B=B,
        mask_array=mask_array,
        mask_val_to_bbox_map=mask_val_to_bbox_map
    )
```

After creating the npz files, here is the complete code that reads the npz files and use them to train the model

```
# Completed on March 25
# 34722789

'''
This code is heavily borrowed from the semantic_segmentation.py,
and the DLStudio SemanticSegmentation Class, especially mUNet.
Additionally, idea of customizing the dataset was borrowed heavily
from the PurdueShapes5MultiObjectDataset and the solution2 of spring 2024
(https://engineering.purdue.edu/DeepLearn/2_best_solutions/2024/Homeworks/HW7/2BestSol
utions/2.pdf)
Just like the previous code,
I will be changing the learning rate, batch size, and epoch to
find the apparently best combination for optimum performance
learning_rate      epoch       batch_size
-------------      -----       ----------
1e-5                20             4
1e-5                30             4
1e-4                20             4
1e-4                30             4

after changing batch size some error occured, avoided increasing batch size.
'''

import random
import numpy
import torch
import os, sys
import torch.nn as nn
import copy
import torch.optim as optim
import sys,os,os.path,glob
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as tvt
import torch.optim as optim
import numpy as np
from PIL import ImageFilter
import numbers
import re
import math
import random
import copy
```

```
import matplotlib.pyplot as plt
import gzip
import pickle
import pymsgbox
import time
import logging
from torch.utils.data import DataLoader
from torch.utils.data import Dataset
from DLStudio import *
from multiprocessing import freeze_support

if __name__ == "__main__":
  freeze_support()

  class MyNPZDataset(Dataset):
    def __init__(self, root_dir):
        # The idea here is to first list the files in the root directory that have npz
extension
        # After getting the list I can extract different information from individual
npz files
        self.root_dir = root_dir
        self.file_list = [f for f in os.listdir(root_dir) if f.endswith('.npz')]
    def __len__(self):
        return len(self.file_list)
    def __getitem__(self, idx):
        file_name = self.file_list[idx]
        file_path = os.path.join(self.root_dir, self.file_list[idx])
        data = np.load(file_path, allow_pickle=True) # the code did not run without
allow_pickle=True
        R = data['R']
        G = data['G']
        B = data['B']
        mask_array = data['mask_array']
        mask_val_to_bbox_map = data['mask_val_to_bbox_map'].item()
        H, W = 256, 256
        R = R.reshape(H, W)
        G = G.reshape(H, W)
        B = B.reshape(H, W)
        image = np.stack([R, G, B], axis=-1)
        image = image.astype(np.float32) / 255.0
        image = torch.from_numpy(image).permute(2, 0, 1)
        mask_tensor = torch.from_numpy(mask_array).float()

        bbox_tensor = torch.zeros((3, 4), dtype=torch.float32)
        for i, mask_value in enumerate([50, 100, 150]):
            bboxes = mask_val_to_bbox_map.get(mask_value, [])
            if bboxes:
                # For better performance, I think this should be modified.
```

```
                # Instead of taking only the first coordinate, I should take all of
them if multiple exists
                bbox_tensor[i] = torch.tensor(bboxes[0])

        return {
            # Needed the file_name later for the bonus part, wanted to find the image
demonstrated in the test batches
            'image': image,
            'mask_tensor': mask_tensor,
            'bbox_tensor': bbox_tensor,
            'file_name': file_name
        }


  dls = DLStudio(
  #                  dataroot = "/home/kak/ImageDatasets/PurdueShapes5MultiObject/",
                   dataroot = "./data/",
                   image_size = [256,256],
                   path_saved_model = "./saved_model",
                   momentum = 0.9,
                   learning_rate = 1e-4,
                   epochs = 30,
                   batch_size = 4,
                   classes = ('rectangle','triangle','disk','oval','star'),
                   use_gpu = True,
                )

  segmenter = DLStudio.SemanticSegmentation(
                   dl_studio = dls,
                   max_num_objects = 3,
                )

  train_dir = './dataset_with_train_masks'
  test_dir = './dataset_with_test_masks'
  train_dataset = MyNPZDataset(root_dir=train_dir)
  test_dataset = MyNPZDataset(root_dir=test_dir)
  train_dataloader = DataLoader(train_dataset, batch_size=dls.batch_size,
shuffle=True)
  test_dataloader = DataLoader(test_dataset, batch_size=dls.batch_size, shuffle=False)
  segmenter.train_dataloader = train_dataloader
  segmenter.test_dataloader = test_dataloader

  class SkipBlockDN(nn.Module):
      """
      This class for the skip connections in the downward leg of the "U"

      Class Path:   DLStudio  ->  SemanticSegmentation  ->  SkipBlockDN
      """
```

```
    def __init__(self, in_ch, out_ch, downsample=False, skip_connections=True):
        super(SkipBlockDN, self).__init__()
        self.downsample = downsample
        self.skip_connections = skip_connections
        self.in_ch = in_ch
        self.out_ch = out_ch
        self.convo1 = nn.Conv2d(in_ch, out_ch, 3, stride=1, padding=1)
        self.convo2 = nn.Conv2d(in_ch, out_ch, 3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(out_ch)
        self.bn2 = nn.BatchNorm2d(out_ch)
        if downsample:
            self.downsampler = nn.Conv2d(in_ch, out_ch, 1, stride=2)
    def forward(self, x):
        identity = x
        out = self.convo1(x)
        out = self.bn1(out)
        out = nn.functional.relu(out)
        if self.in_ch == self.out_ch:
            out = self.convo2(out)
            out = self.bn2(out)
            out = nn.functional.relu(out)
        if self.downsample:
            out = self.downsampler(out)
            identity = self.downsampler(identity)
        if self.skip_connections:
            if self.in_ch == self.out_ch:
                out = out + identity
            else:
                out = out + torch.cat((identity, identity), dim=1)
        return out

class SkipBlockUP(nn.Module):
    """
    This class is for the skip connections in the upward leg of the "U"

    Class Path:   DLStudio  ->  SemanticSegmentation  ->  SkipBlockUP
    """
    def __init__(self, in_ch, out_ch, upsample=False, skip_connections=True):
        super(SkipBlockUP, self).__init__()
        self.upsample = upsample
        self.skip_connections = skip_connections
        self.in_ch = in_ch
        self.out_ch = out_ch
        self.convoT1 = nn.ConvTranspose2d(in_ch, out_ch, 3, padding=1)
        self.convoT2 = nn.ConvTranspose2d(in_ch, out_ch, 3, padding=1)
        self.bn1 = nn.BatchNorm2d(out_ch)
        self.bn2 = nn.BatchNorm2d(out_ch)
        if upsample:
```

```
            self.upsampler = nn.ConvTranspose2d(in_ch, out_ch, 1, stride=2,
dilation=2, output_padding=1, padding=0)
      def forward(self, x):
          identity = x
          out = self.convoT1(x)
          out = self.bn1(out)
          out = nn.functional.relu(out)
          out  =  nn.ReLU(inplace=False)(out)
          if self.in_ch == self.out_ch:
              out = self.convoT2(out)
              out = self.bn2(out)
              out = nn.functional.relu(out)
          if self.upsample:
              out = self.upsampler(out)
              identity = self.upsampler(identity)
          if self.skip_connections:
              if self.in_ch == self.out_ch:
                  out = out + identity
              else:
                  out = out + identity[:,self.out_ch:,:,:]
          return out
  # I created a custom mUNet here to introduce the ASPP module at the bottleneck
  class CustomUnet(DLStudio.SemanticSegmentation.mUNet):
      def __init__(self, skip_connections=True, depth=16):
        super(CustomUnet, self).__init__(skip_connections=True, depth=depth)
        self.depth = depth // 2
        self.conv_in = nn.Conv2d(3, 64, 3, padding=1)
        ##  For the DN arm of the U:
        self.bn1DN  = nn.BatchNorm2d(64)
        self.bn2DN  = nn.BatchNorm2d(128)
        self.skip64DN_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip64DN_arr.append(SkipBlockDN(64, 64,
skip_connections=skip_connections))
        self.skip64dsDN = SkipBlockDN(64, 64,   downsample=True,
skip_connections=skip_connections)
        self.skip64to128DN = SkipBlockDN(64, 128, skip_connections=skip_connections )
        self.skip128DN_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip128DN_arr.append(SkipBlockDN(128, 128,
skip_connections=skip_connections))
        self.skip128dsDN = SkipBlockDN(128,128, downsample=True,
skip_connections=skip_connections)

        '''
        Start of the ASPP Module. I got the main instruction from this piazza post:
        https://piazza.com/class/m5qxf8zm2ds2gf/post/275
        In short, concatenated the output of 4 different convolutions:
```

```
        1x1 kernel without dilation
        3x3 kernel with dilation=2
        3x3 kernel with dilation=4
        3x3 kernel with dilation=6
        Finally passed through the concatenated output to another convolutional layer
        to regain the expeted output shape.
        '''
        self.aspp_conv1 = nn.Conv2d(128, 128, 1, padding=0)
        self.aspp_conv2 = nn.Conv2d(128, 128, 3, padding=2, dilation=2)
        self.aspp_conv3 = nn.Conv2d(128, 128, 3, padding=4, dilation=4)
        self.aspp_conv4 = nn.Conv2d(128, 128, 3, padding=6, dilation=6)
        self.aspp_final_conv = nn.Conv2d(512, 128, 1)

        ##  For the UP arm of the U:
        self.bn1UP  = nn.BatchNorm2d(128)
        self.bn2UP  = nn.BatchNorm2d(64)
        self.skip64UP_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip64UP_arr.append(SkipBlockUP(64, 64,
skip_connections=skip_connections))
        self.skip64usUP = SkipBlockUP(64, 64, upsample=True,
skip_connections=skip_connections)
        self.skip128to64UP = SkipBlockUP(128, 64, skip_connections=skip_connections )
        self.skip128UP_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip128UP_arr.append(SkipBlockUP(128, 128,
skip_connections=skip_connections))
        self.skip128usUP = SkipBlockUP(128,128, upsample=True,
skip_connections=skip_connections)
        self.conv_out = nn.ConvTranspose2d(64, 3, 3,
stride=2,dilation=2,output_padding=1,padding=2)

      def forward(self, x):
        ##  Going down to the bottom of the U:
        x = nn.MaxPool2d(2,2)(nn.functional.relu(self.conv_in(x)))
        for i,skip64 in enumerate(self.skip64DN_arr[:self.depth//4]):
            x = skip64(x)

        num_channels_to_save1 = x.shape[1] // 2
        save_for_upside_1 = x[:,:num_channels_to_save1,:,:].clone()
        x = self.skip64dsDN(x)
        for i,skip64 in enumerate(self.skip64DN_arr[self.depth//4:]):
            x = skip64(x)
        x = self.bn1DN(x)
        num_channels_to_save2 = x.shape[1] // 2
        save_for_upside_2 = x[:,:num_channels_to_save2,:,:].clone()
        x = self.skip64to128DN(x)
        for i,skip128 in enumerate(self.skip128DN_arr[:self.depth//4]):
```

```
            x = skip128(x)

        x = self.bn2DN(x)
        num_channels_to_save3 = x.shape[1] // 2
        save_for_upside_3 = x[:,:num_channels_to_save3,:,:].clone()
        for i,skip128 in enumerate(self.skip128DN_arr[self.depth//4:]):
            x = skip128(x)
        x = self.skip128dsDN(x)

        '''
        Designing the ASPP:
        aspp1,2,3,4 means no dilation, dilation=2,4,6 respectively.
        aspp_concat concatenates these four outputs which then passes
        through the final conv layer.
        '''
        aspp1 = self.aspp_conv1(x)
        aspp2 = self.aspp_conv2(x)
        aspp3 = self.aspp_conv3(x)
        aspp4 = self.aspp_conv4(x)
        aspp_concat = torch.cat((aspp1, aspp2, aspp3, aspp4), dim=1)
        x = self.aspp_final_conv(aspp_concat)

        ## Coming up from the bottom of U on the other side:
        x = self.skip128usUP(x)
        for i,skip128 in enumerate(self.skip128UP_arr[:self.depth//4]):
            x = skip128(x)
        x[:,:num_channels_to_save3,:,:] =  save_for_upside_3
        x = self.bn1UP(x)
        for i,skip128 in enumerate(self.skip128UP_arr[:self.depth//4]):
            x = skip128(x)
        x = self.skip128to64UP(x)
        for i,skip64 in enumerate(self.skip64UP_arr[self.depth//4:]):
            x = skip64(x)
        x[:,:num_channels_to_save2,:,:] =  save_for_upside_2
        x = self.bn2UP(x)
        x = self.skip64usUP(x)
        for i,skip64 in enumerate(self.skip64UP_arr[:self.depth//4]):
            x = skip64(x)
        x[:,:num_channels_to_save1,:,:] =  save_for_upside_1
        x = self.conv_out(x)
        return x

  model = CustomUnet(skip_connections=True, depth=16)

  number_of_learnable_params = sum(p.numel() for p in model.parameters() if
p.requires_grad)
  print("\n\nThe number of learnable parameters in the model: %d\n" %
number_of_learnable_params)
```

```
  def run_code_for_testing_semantic_segmentation(net):
    net.load_state_dict(torch.load(dls.path_saved_model))
    net = net.to(dls.device)
    net.eval()
    batch_size = dls.batch_size
    image_size = dls.image_size
    max_num_objects = segmenter.max_num_objects
    with torch.no_grad():
        for i, data in enumerate(segmenter.test_dataloader):
            im_tensor = data['image'].to(dls.device)
            mask_tensor = data['mask_tensor'].to(dls.device)
            bbox_tensor = data['bbox_tensor'].to(dls.device)
            file_names = data['file_name']

            if i % 50 == 0:
                print("\nShowing output for test batch %d: " % (i + 1))
                count = i
                # Wanted to know which files are being used to demonstrate the model
performance
                print("File names in this batch:", file_names)
                outputs = net(im_tensor)
                threshold = 0.5
                outputs_binary = (outputs > threshold).float()
                display_tensor = torch.zeros(5 * batch_size, 3, image_size[0],
image_size[1], dtype=torch.float32)
                '''
                I did not really find any meaning of the first row in the original
code, where we can easily see
                the bounding boxes on top of the original images. So, I modified the
first two rows from gray_scaled
                bbox and original bbox to original image and original bbox. While
writing the report, I saw that the
                images were not showing the actual color. I think I may have messed up
the transformation of the images
                or somewhere in the display line. But given the time constraint, I
have decided not to pursue this matter
                as the main objective was to compare the masking performance of the
model, not the colorful display.
                '''
                display_tensor[:batch_size, :, :, :] = im_tensor
                for idx in range(batch_size):
                    for bbox_idx in range(max_num_objects):
                        bb_tensor = bbox_tensor[idx, bbox_idx]
                        if torch.any(bb_tensor != 0):
                            i1, i2 = int(bb_tensor[1]), int(bb_tensor[3])
                            j1, j2 = int(bb_tensor[0]), int(bb_tensor[2])
```

```
                            # Assigning pixel value of 1 for the bounding boxes
                            im_tensor[idx, :, i1:i2, j1] = 1.0
                            im_tensor[idx, :, i1:i2, j2] = 1.0
                            im_tensor[idx, :, i1, j1:j2] = 1.0
                            im_tensor[idx, :, i2, j1:j2] = 1.0
                display_tensor[batch_size:2 * batch_size, :, :, :] = im_tensor
                for batch_im_idx in range(batch_size):
                    for mask_layer_idx in range(max_num_objects):
                        mask = outputs_binary[batch_im_idx, mask_layer_idx, :,
:].unsqueeze(0).unsqueeze(0)
                        mask = mask.repeat(1, 3, 1, 1)

display_tensor[2*batch_size+batch_im_idx*max_num_objects+mask_layer_idx, :, :, :] =
mask

                os.makedirs("./results_coco_lr4_e30_bs4/masks", exist_ok=True)
                for batch_im_idx in range(batch_size):
                    for mask_layer_idx in range(max_num_objects):
                        mask_save_path =
f"./results_coco_lr4_e30_bs4/masks/batch_{count+1}_image_{batch_im_idx+1}_class_{mask_
layer_idx+1}.png"
                        mask_tensor = outputs[batch_im_idx, mask_layer_idx, :,
:].unsqueeze(0).unsqueeze(0)
                        torchvision.utils.save_image(mask_tensor, mask_save_path,
normalize=True)
                        print(f"Saved mask for batch {i+1}, image {batch_im_idx+1},
class {mask_layer_idx+1} to {mask_save_path}")

                save_path = f"./results_coco_lr4_e30_bs4/output_batch_{count+1}.png"
                grid = torchvision.utils.make_grid(display_tensor, nrow=batch_size,
normalize=False, padding=2, pad_value=100)
                grid_np = grid.permute(1, 2, 0).cpu().numpy()
                plt.figure(figsize=(10, 10))
                plt.imshow(grid_np)
                plt.axis('off')
                plt.tight_layout()
                plt.savefig(save_path, dpi=300, bbox_inches='tight')
                plt.close()
                print(f"Saved visualization to {save_path}")

  def save_model(model, save_path):
    torch.save(model.state_dict(), save_path)
    print(f"Model saved to {save_path}")

  '''
  This is the required dice_function of which the skeleton was provided
  '''
  def dice_loss(preds: torch.Tensor, ground_truth: torch.Tensor, epsilon=1e-6):
```

```
    # Flattened the prediction and ground truth vector first
    # Got this idea from the first solution of Spring 2024 Page 4
    #
https://engineering.purdue.edu/DeepLearn/2_best_solutions/2024/Homeworks/HW7/2BestSolu
tions/1.pdf
    preds_flat = preds.view(preds.size(0), preds.size(1), -1)
    ground_truth_flat = ground_truth.view(ground_truth.size(0), ground_truth.size(1),
-1)
    # Step 1: Compute Dice Coefficient
    numerator = torch.sum(preds_flat * ground_truth_flat, dim=-1)
    denominator = torch.sum(preds_flat ** 2, dim=-1) + torch.sum(ground_truth_flat **
2, dim=-1)
    # Step 2: dice_coefficient = 2*numerator / (denominator + epsilon)
    dice_coefficient = (2.0 * numerator) / (denominator + epsilon)  # Shape:
[batch_size, num_classes]
    # Step 3: Compute dice_loss  = 1 - dice_coefficient
    dice_loss = 1.0 - torch.mean(dice_coefficient)
    return dice_loss

  def run_code_for_training_for_semantic_segmentation(net):
    filename_for_out1 = "performance_numbers_" + str(dls.epochs) + ".txt"
    FILE1 = open(filename_for_out1, 'w')
    net = net.to(dls.device)
    criterion1 = nn.MSELoss()
    optimizer = optim.SGD(net.parameters(), lr=dls.learning_rate,
momentum=dls.momentum)

    start_time = time.perf_counter()
    losses = []
    iterations = []

    for epoch in range(dls.epochs):
        print("")
        running_loss_segmentation = 0.0
        for i, data in enumerate(segmenter.train_dataloader):
            im_tensor = data['image'].to(dls.device)
            mask_tensor = data['mask_tensor'].to(dls.device)

            optimizer.zero_grad()
            output = net(im_tensor)
            # segmentation_loss = criterion1(output, mask_tensor)
            segmentation_loss = dice_loss(output, mask_tensor)
            # segmentation_loss = criterion1(output, mask_tensor) + 100 *
dice_loss(output, mask_tensor)
            segmentation_loss.backward()
            optimizer.step()

            running_loss_segmentation += segmentation_loss.item()
```

```
            if i % 500 == 499:
                current_time = time.perf_counter()
                elapsed_time = current_time - start_time
                avg_loss_segmentation = running_loss_segmentation / float(500)
                print("[epoch=%d/%d, iter=%4d  elapsed_time=%3d secs]   MSE loss:
%.3f" %
                      (epoch+1, dls.epochs, i+1, elapsed_time, avg_loss_segmentation))
                FILE1.write("%.3f\n" % avg_loss_segmentation)
                FILE1.flush()
                losses.append(avg_loss_segmentation)
                iterations.append(len(losses))
                running_loss_segmentation = 0.0

    FILE1.close()
    print("\nFinished Training\n")
    plt.figure(figsize=(10, 6))
    plt.plot(iterations, losses, label="MSE Loss", color="blue")
    plt.title("Results_coco_lr4_e30_bs4", fontsize=16)
    plt.xlabel("Iteration", fontsize=14)
    plt.ylabel("MSE Loss", fontsize=14)
    plt.grid(True)
    plt.legend(fontsize=12)
    plt.tight_layout()
    save_path = "./loss_vs_iteration_with_aspp_coco_lr4_e30_bs4.png"
    plt.savefig(save_path, dpi=1200)
    plt.close()
    print(f"Saved loss vs. iteration plot to {save_path}")
    save_model(net, dls.path_saved_model)

  run_code_for_training_for_semantic_segmentation(model)
  run_code_for_testing_semantic_segmentation(model)
```

MSE only Analysis

Training Loss of PurdueShapes5MultiObjectDataset

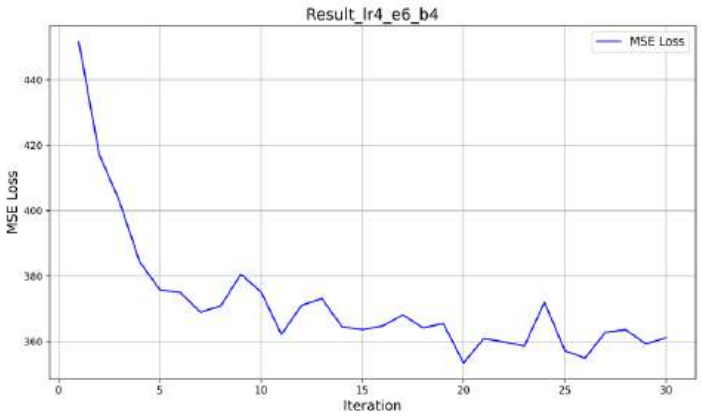


Fig: Loss curve for learning rate 1e-4, epoch 6, batch 4

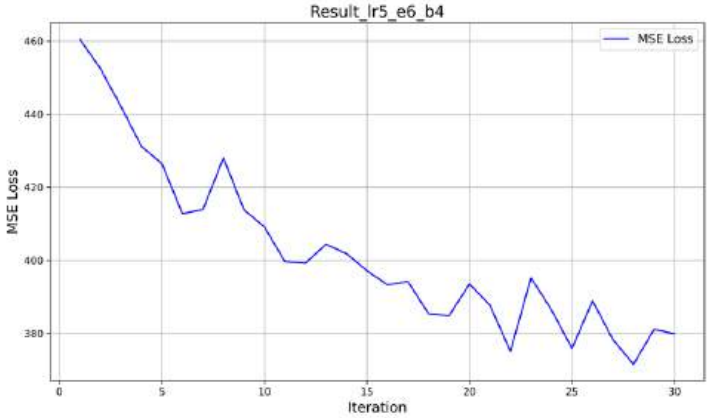


Fig: Loss curve for learning rate 1e-5, epoch 6, batch 4

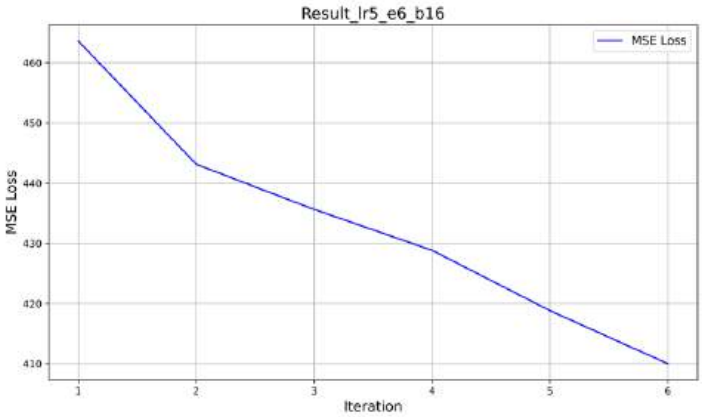


Fig: Loss curve for learning rate 1e-5, epoch 6, batch 16

Fig: Loss curve for learning rate 1e-5, epoch 12, batch 4

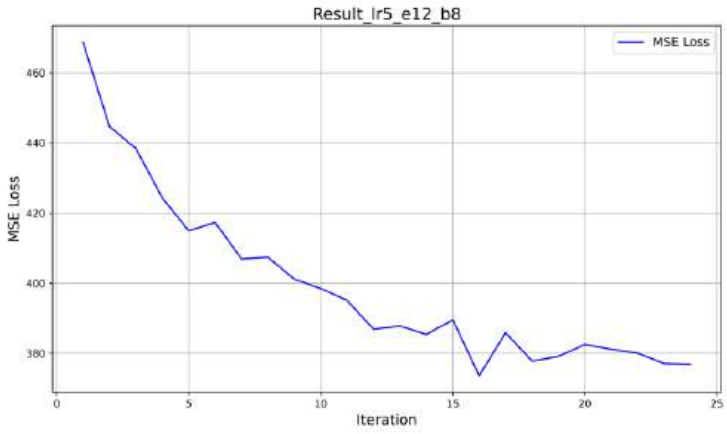


Fig: Loss curve for learning rate 1e-5, epoch 12, batch 8

Training Loss of MSCOCO

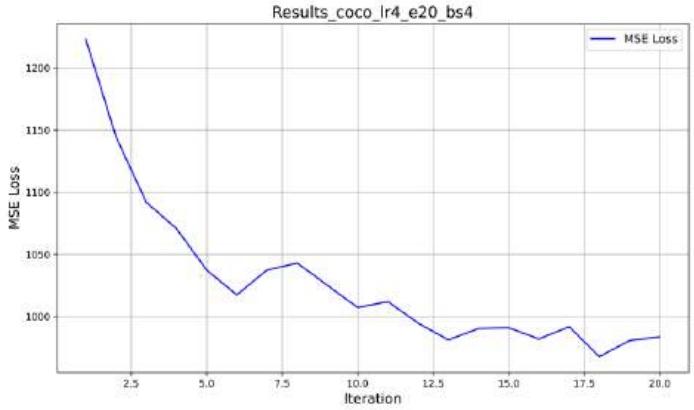


Fig: Loss curve for learning rate 1e-4, epoch 20, batch 4

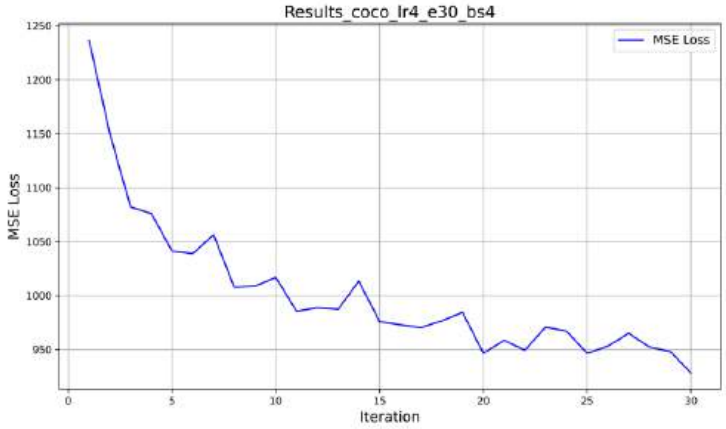


Fig: Loss curve for learning rate 1e-4, epoch 30, batch 4

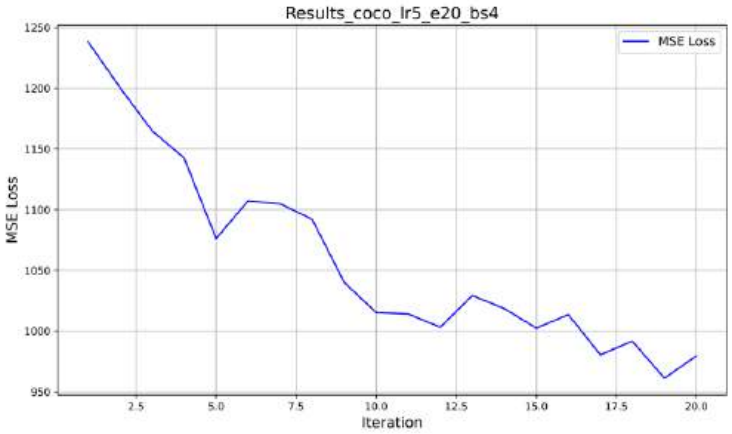


Fig: Loss curve for learning rate 1e-5, epoch 20, batch 4

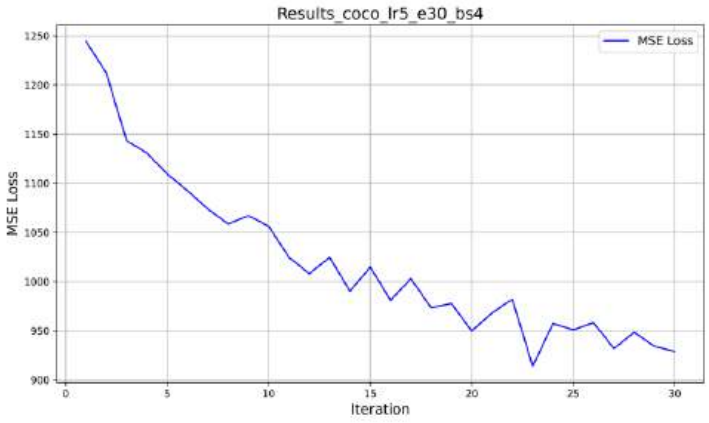
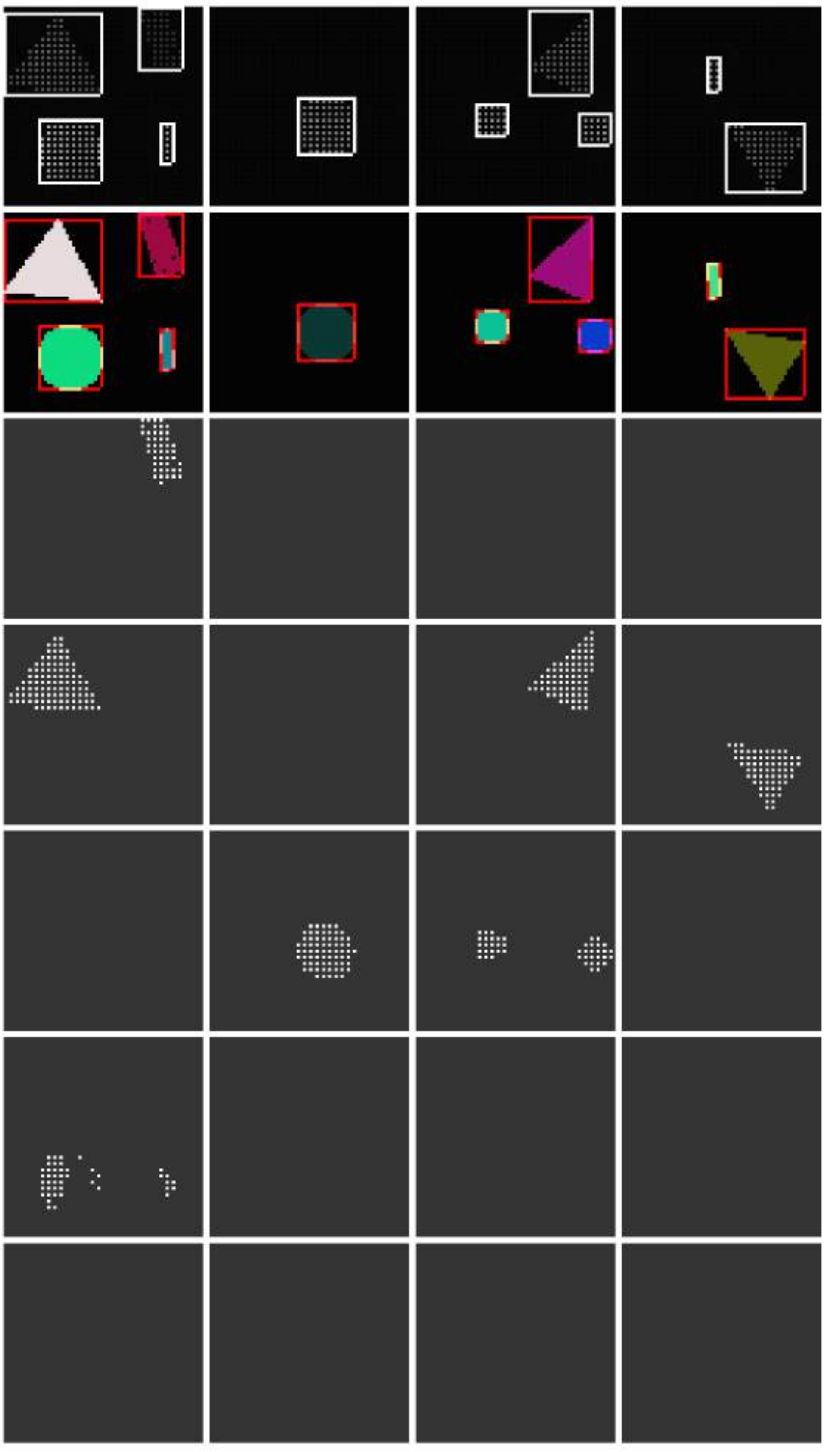


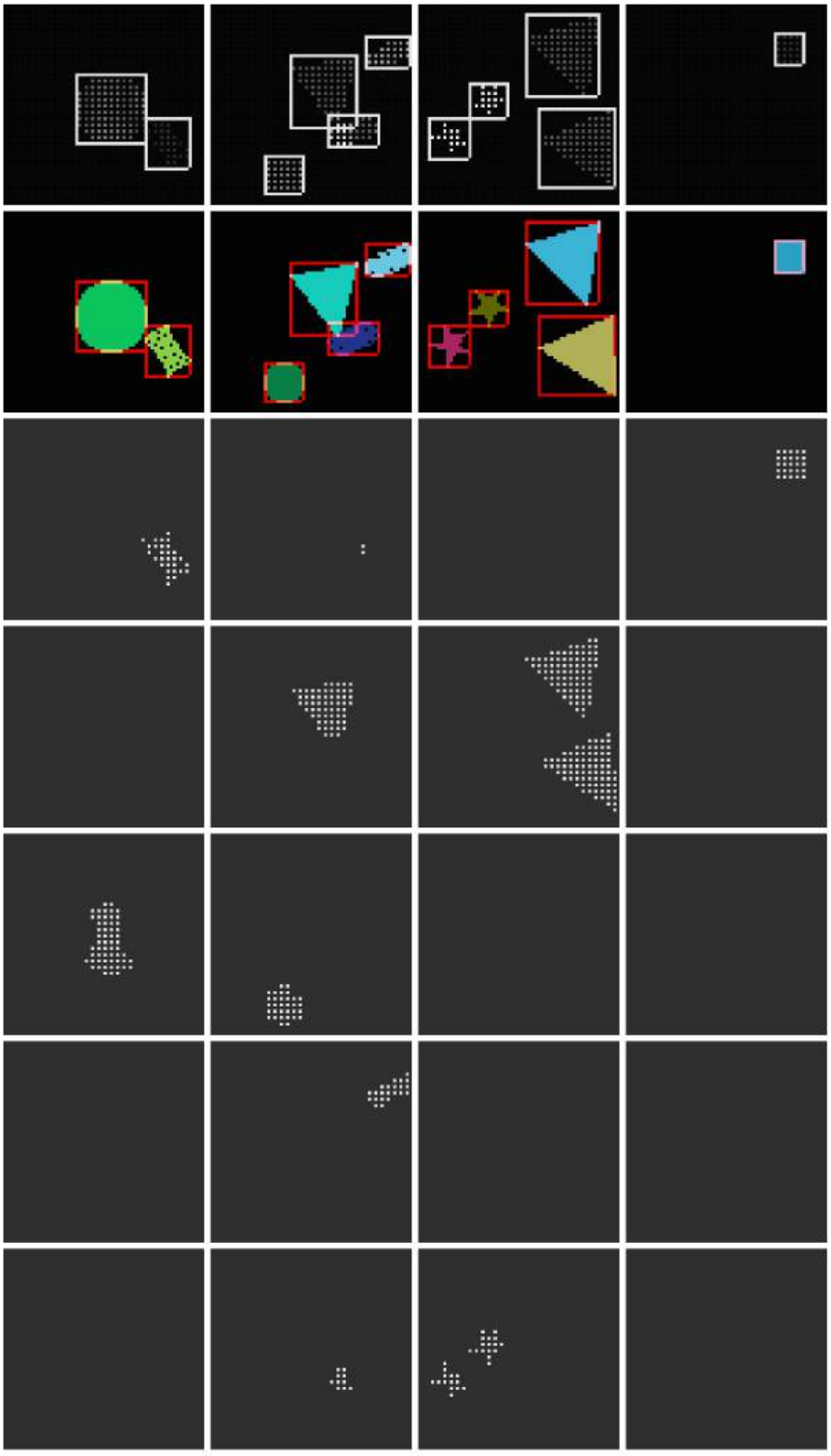
Fig: Loss curve for learning rate 1e-5, epoch 30, batch 4

Test Results of PurdueShapes5MultiObjectDataset

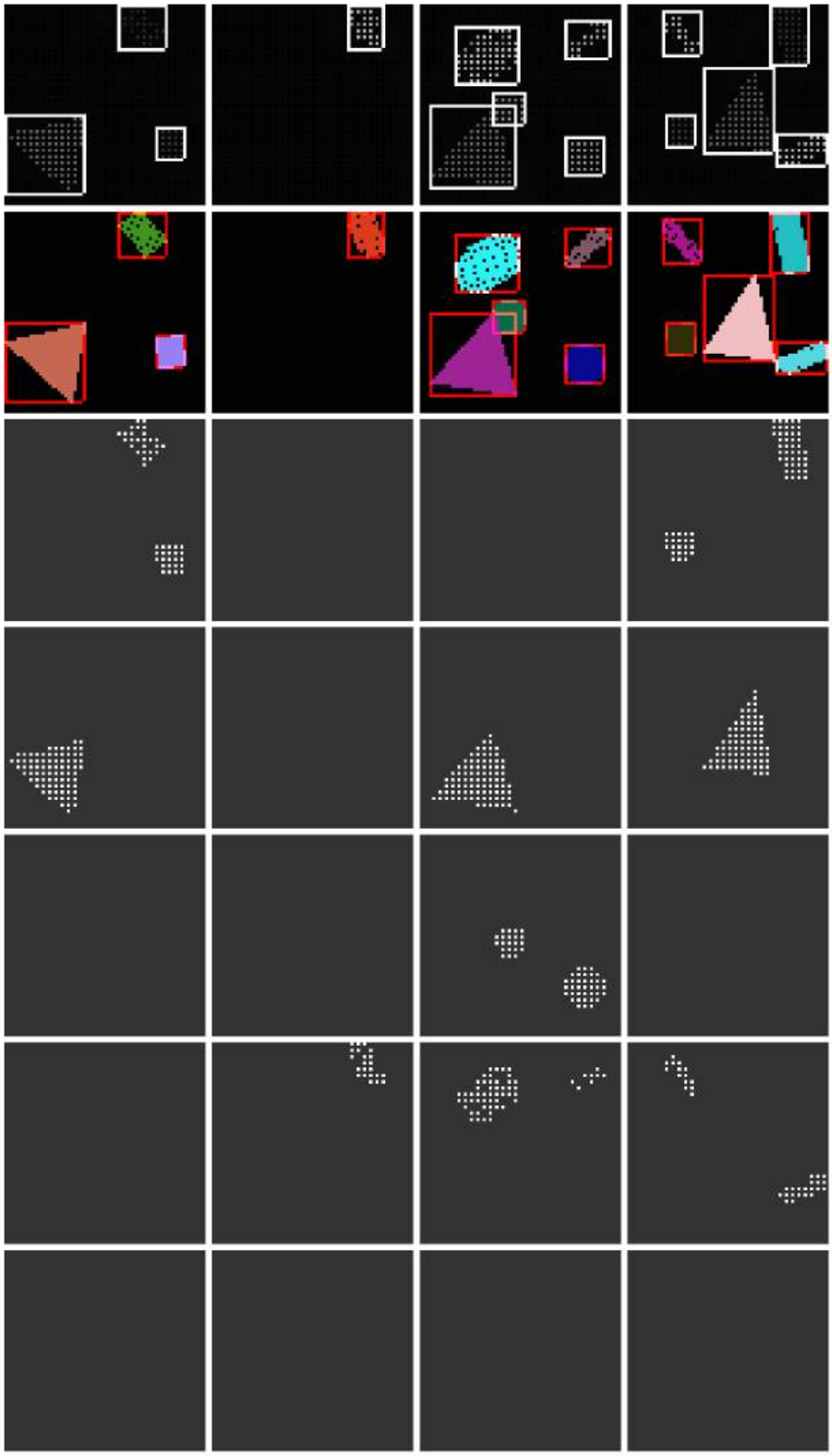
Output of learning rate 1e-4, epoch 6, batch 4



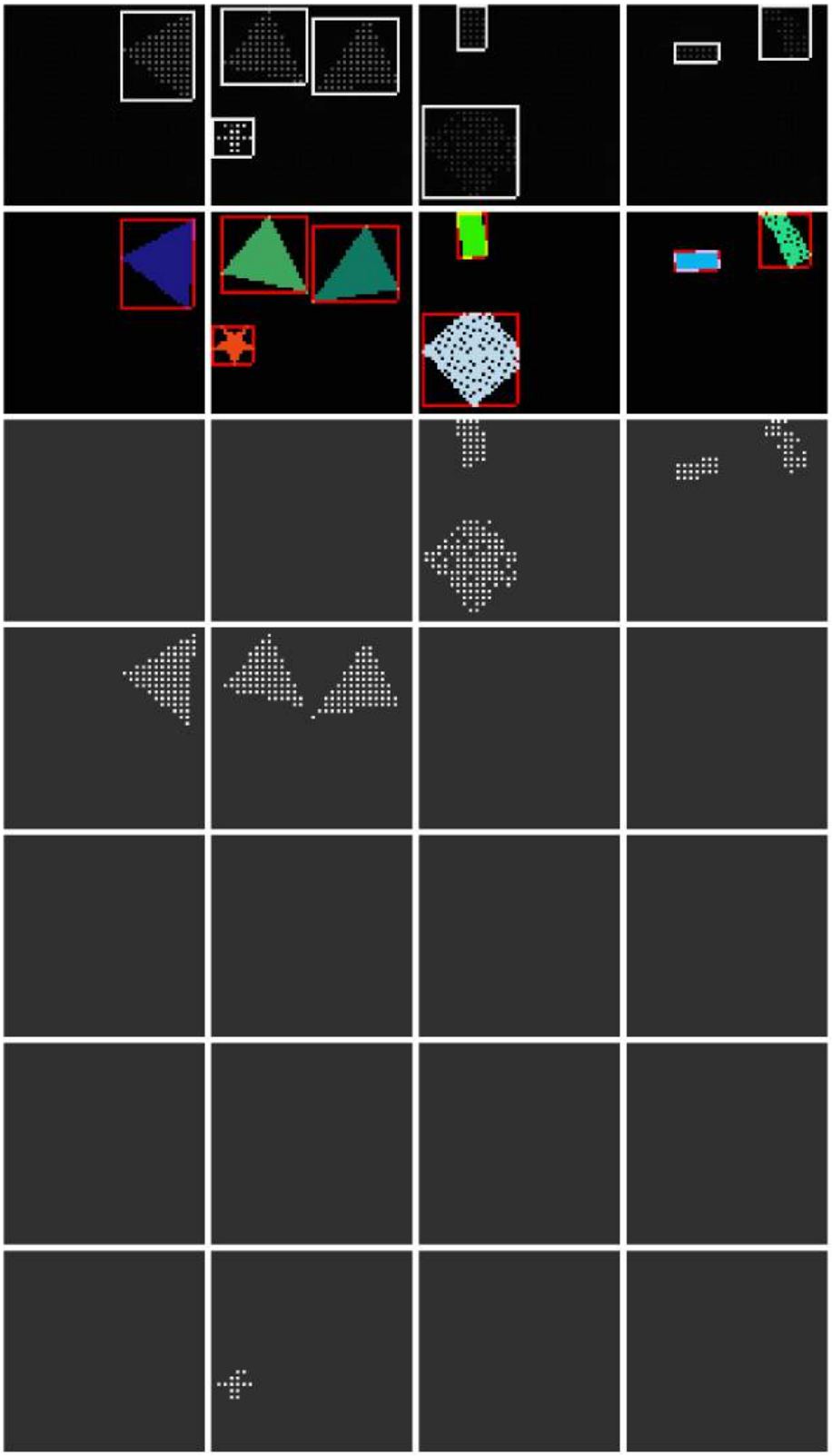
Batch 1



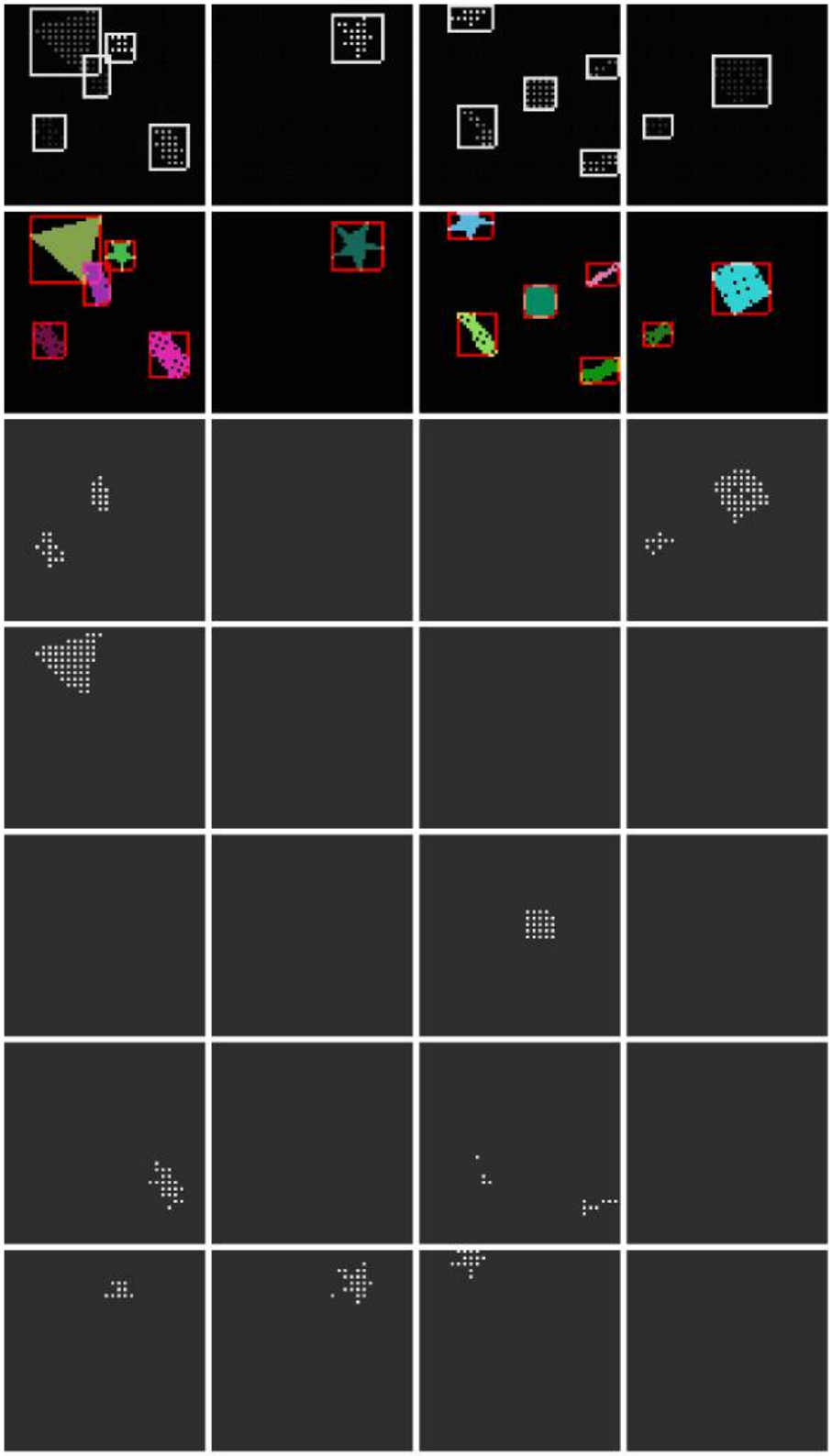
Batch 51



Batch 101

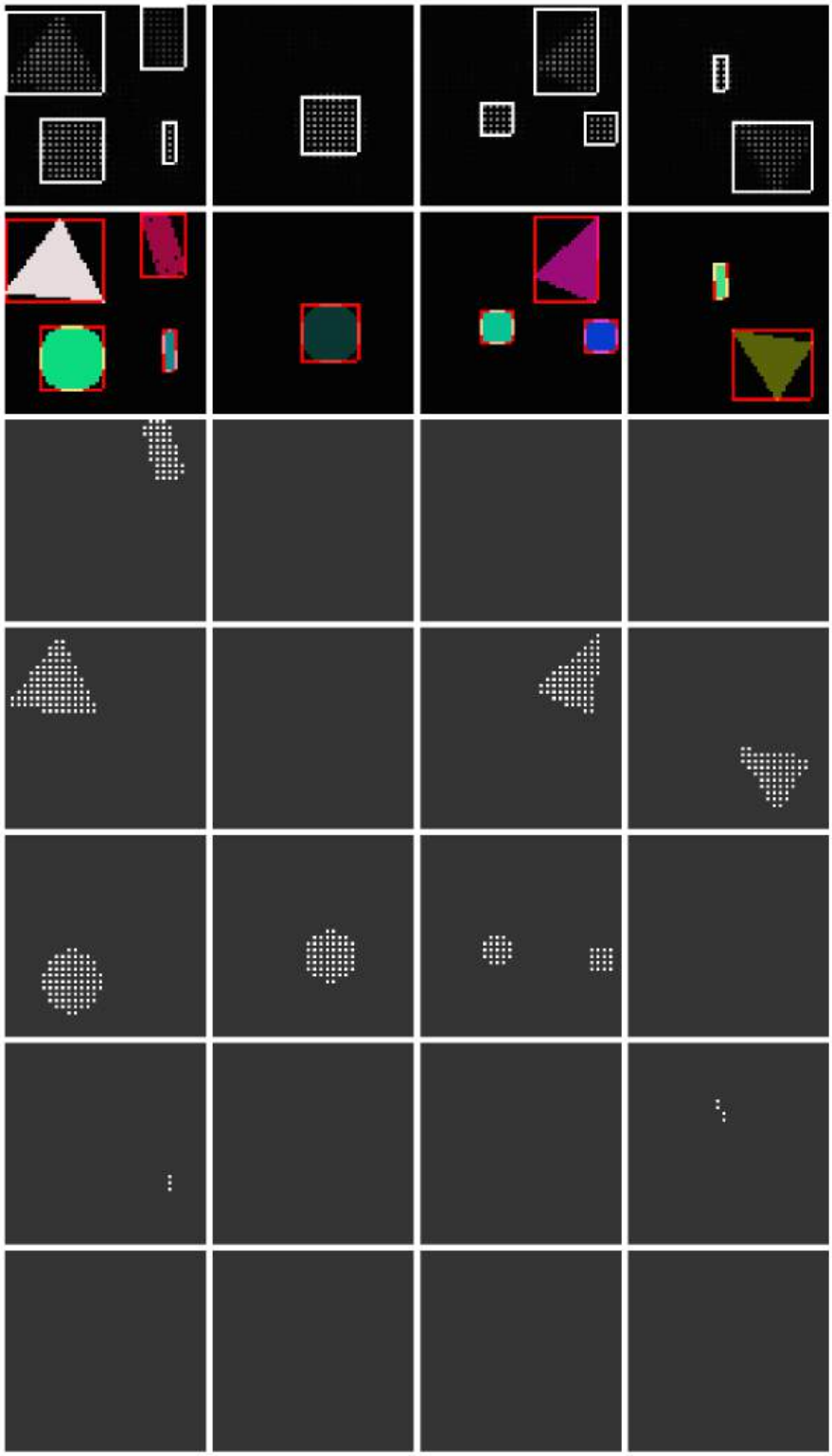


Batch 151

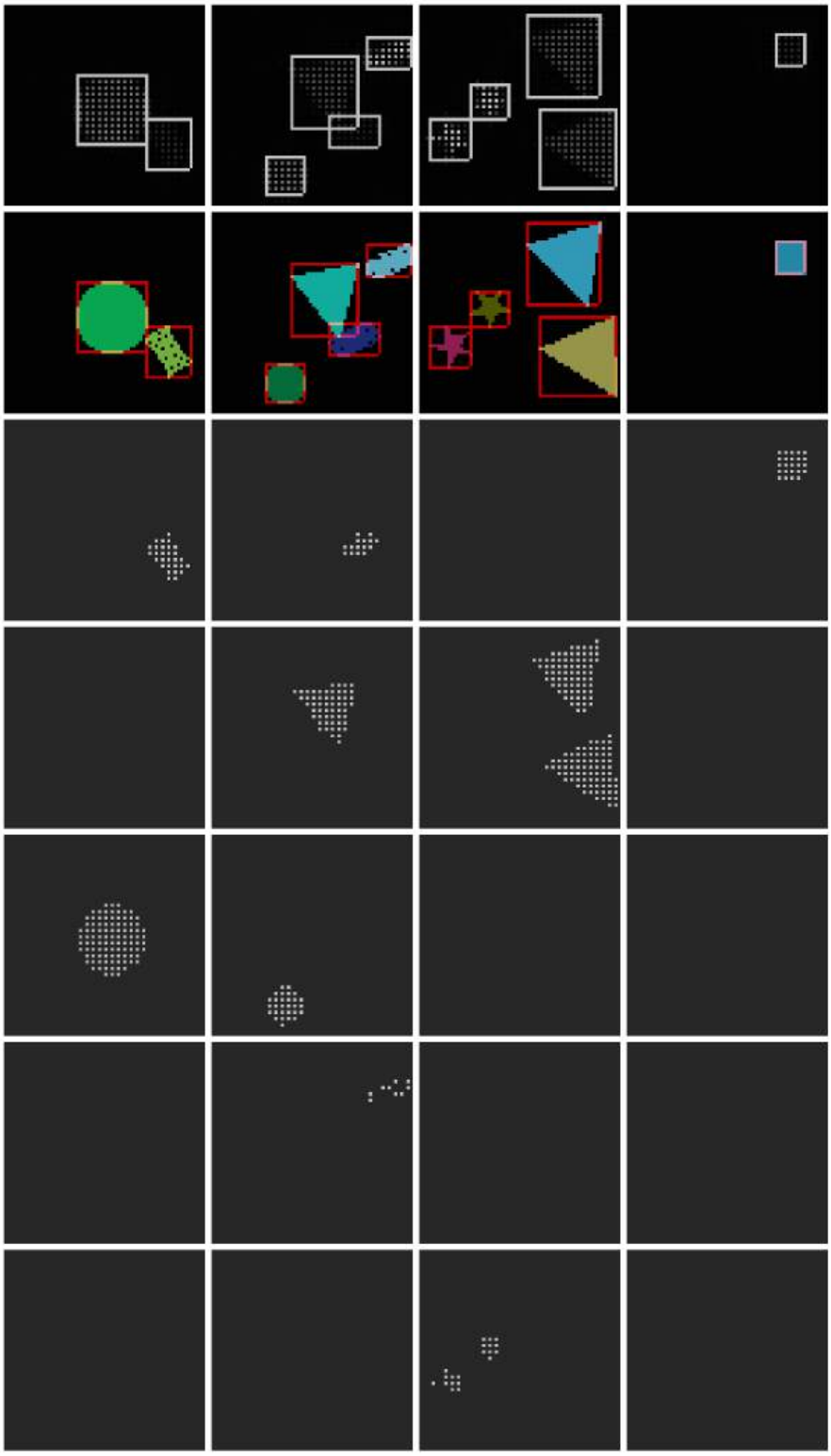


Batch 201

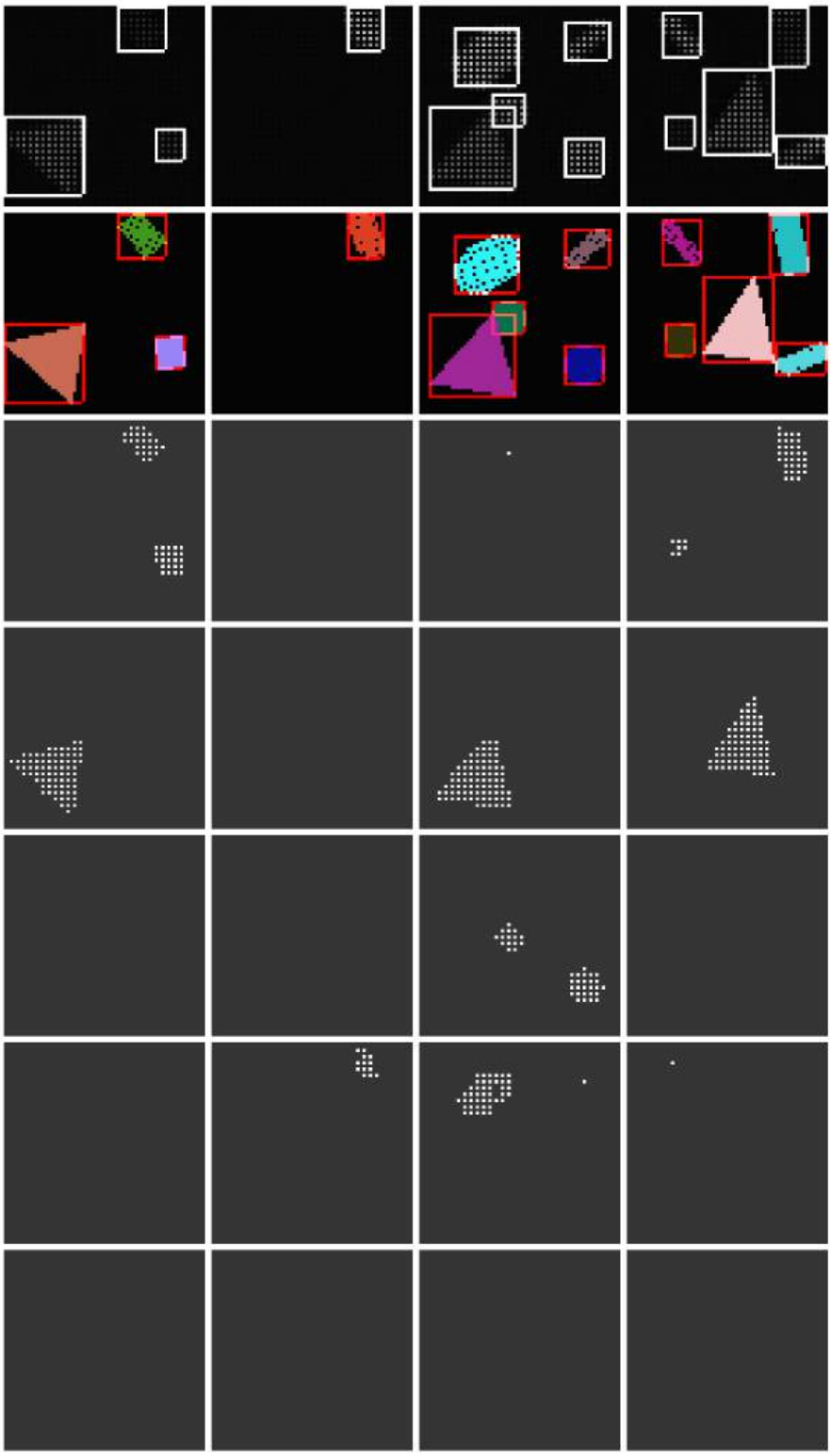
Output of learning rate 1e-5, epoch 6, batch 4



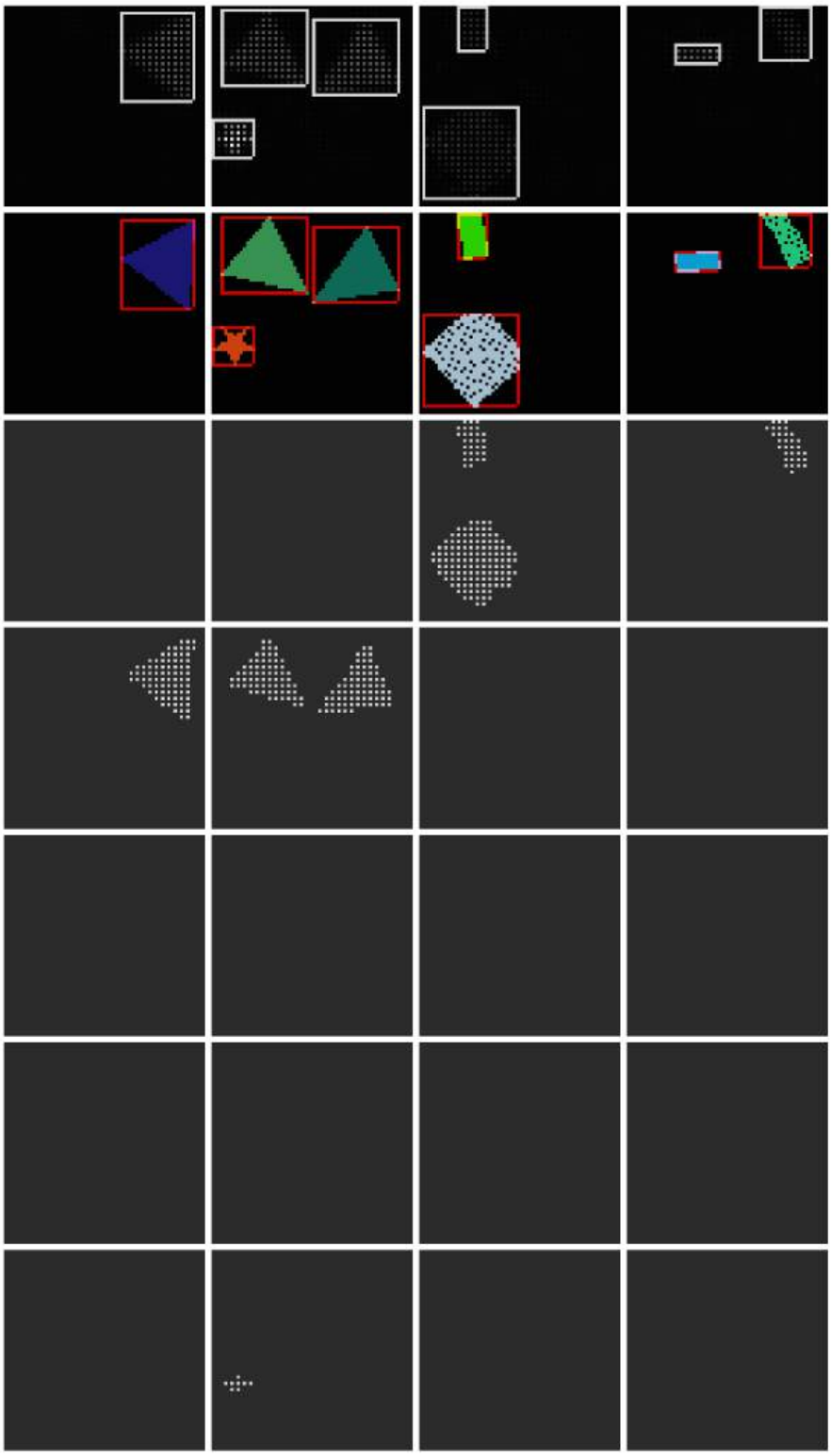
Batch 1



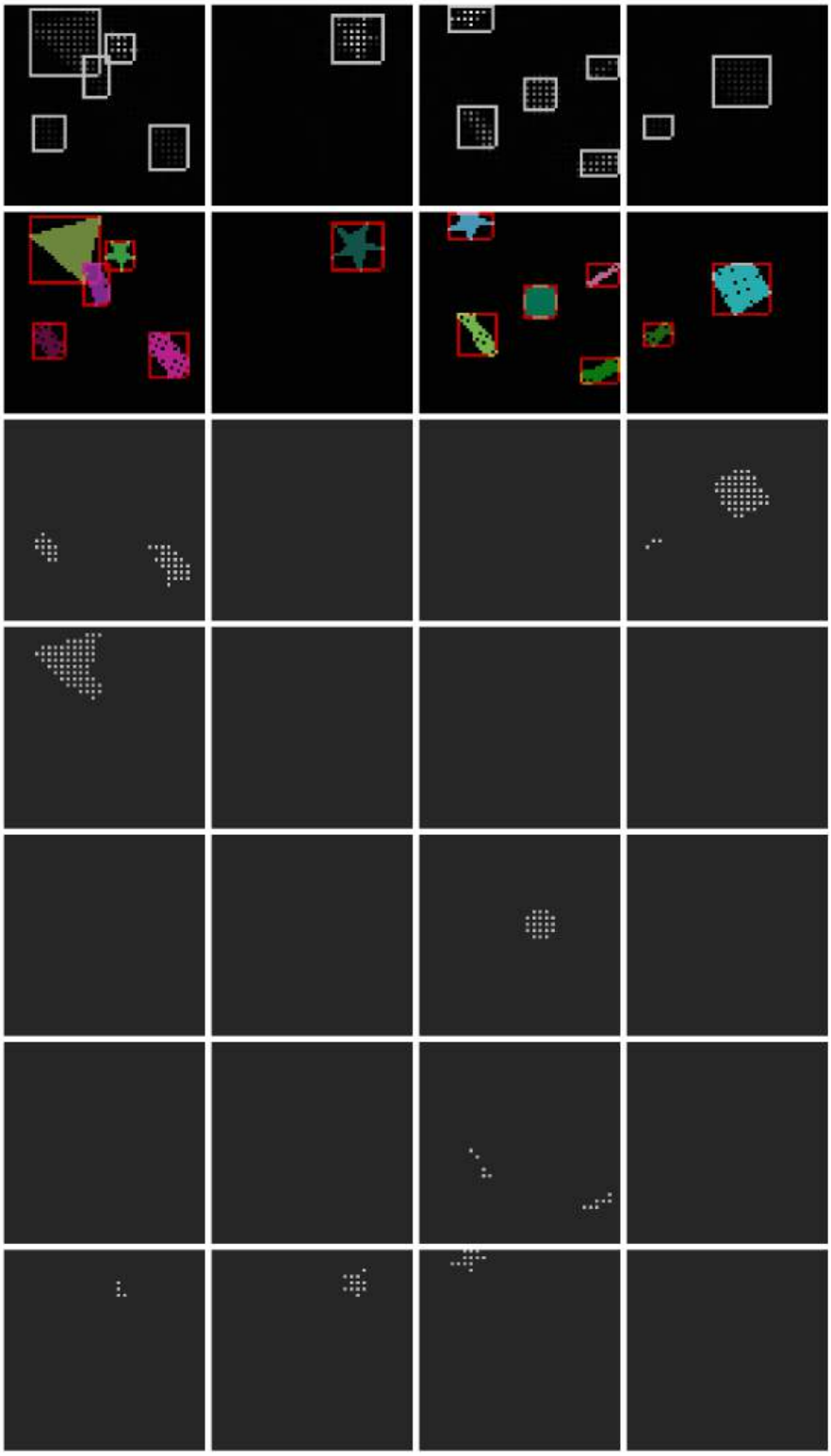
Batch 51



Batch 101

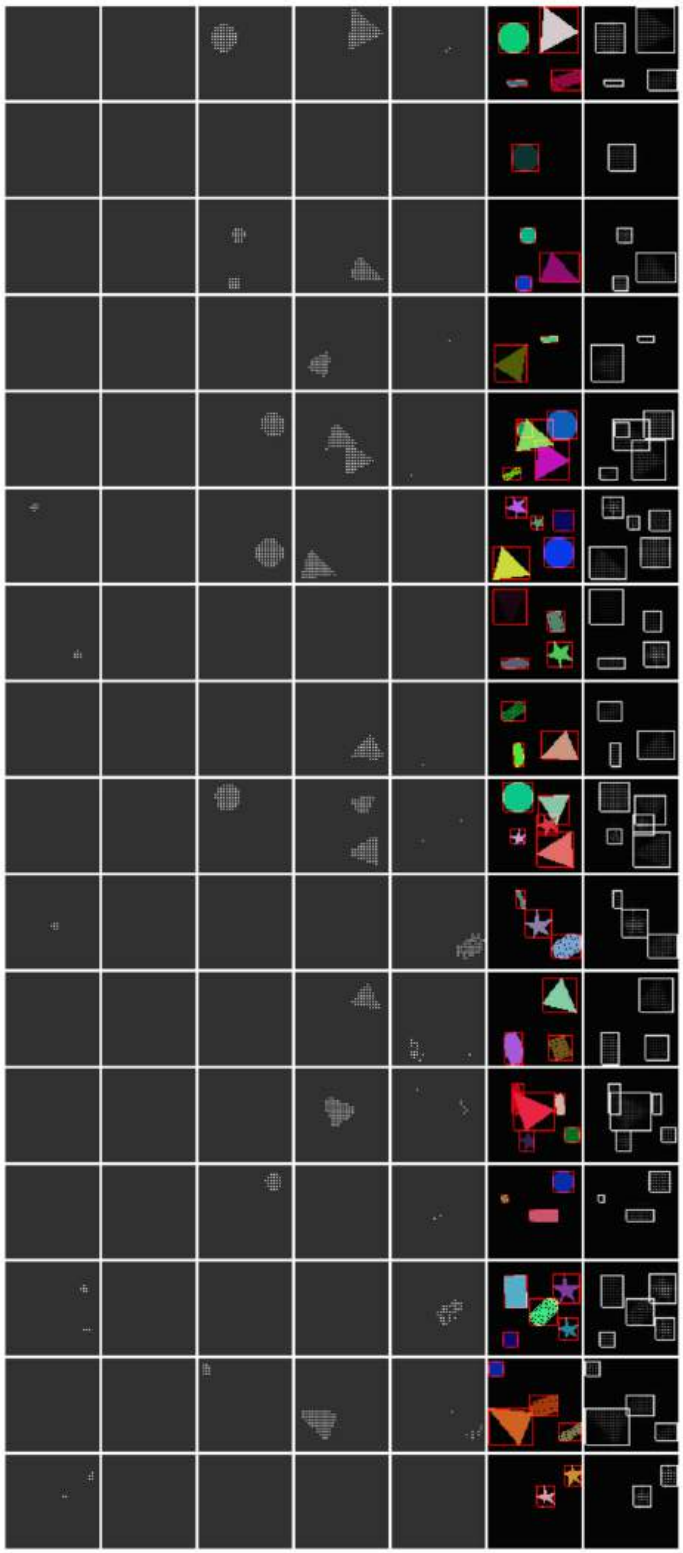


Batch 151



Batch 201

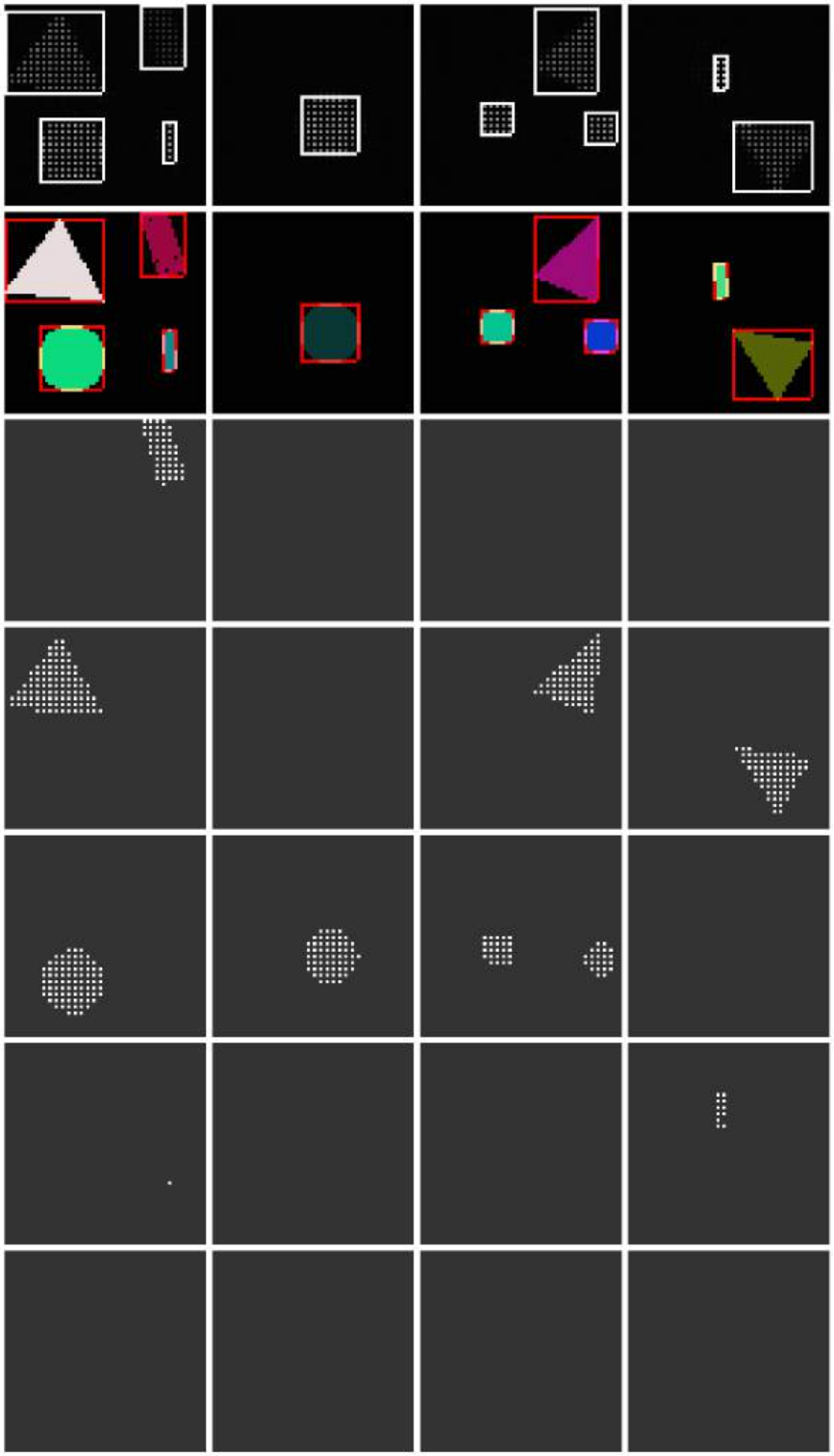
Output of learning rate 1e-5, epoch 6, batch 16



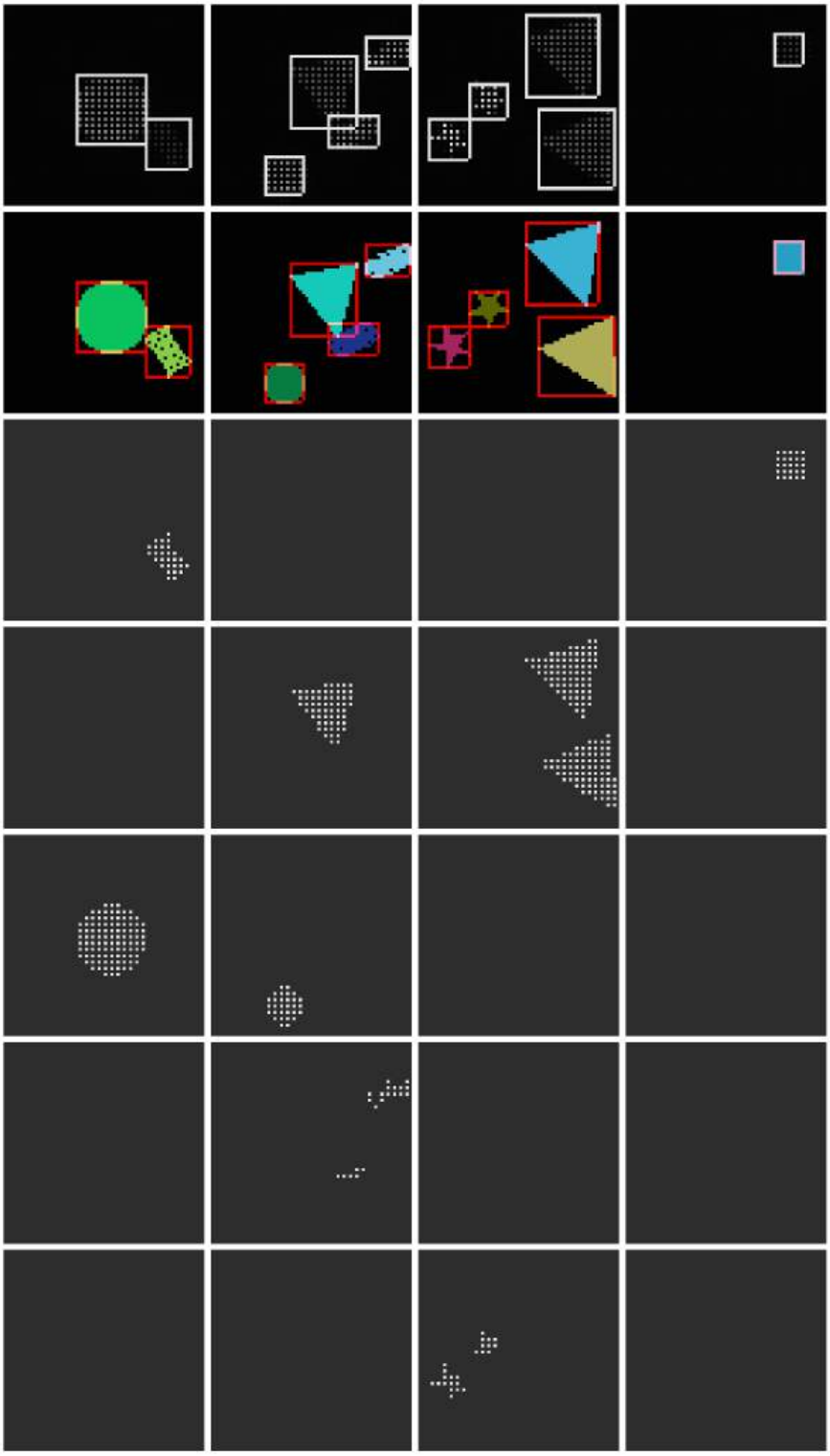
Batch 1

Batch 51

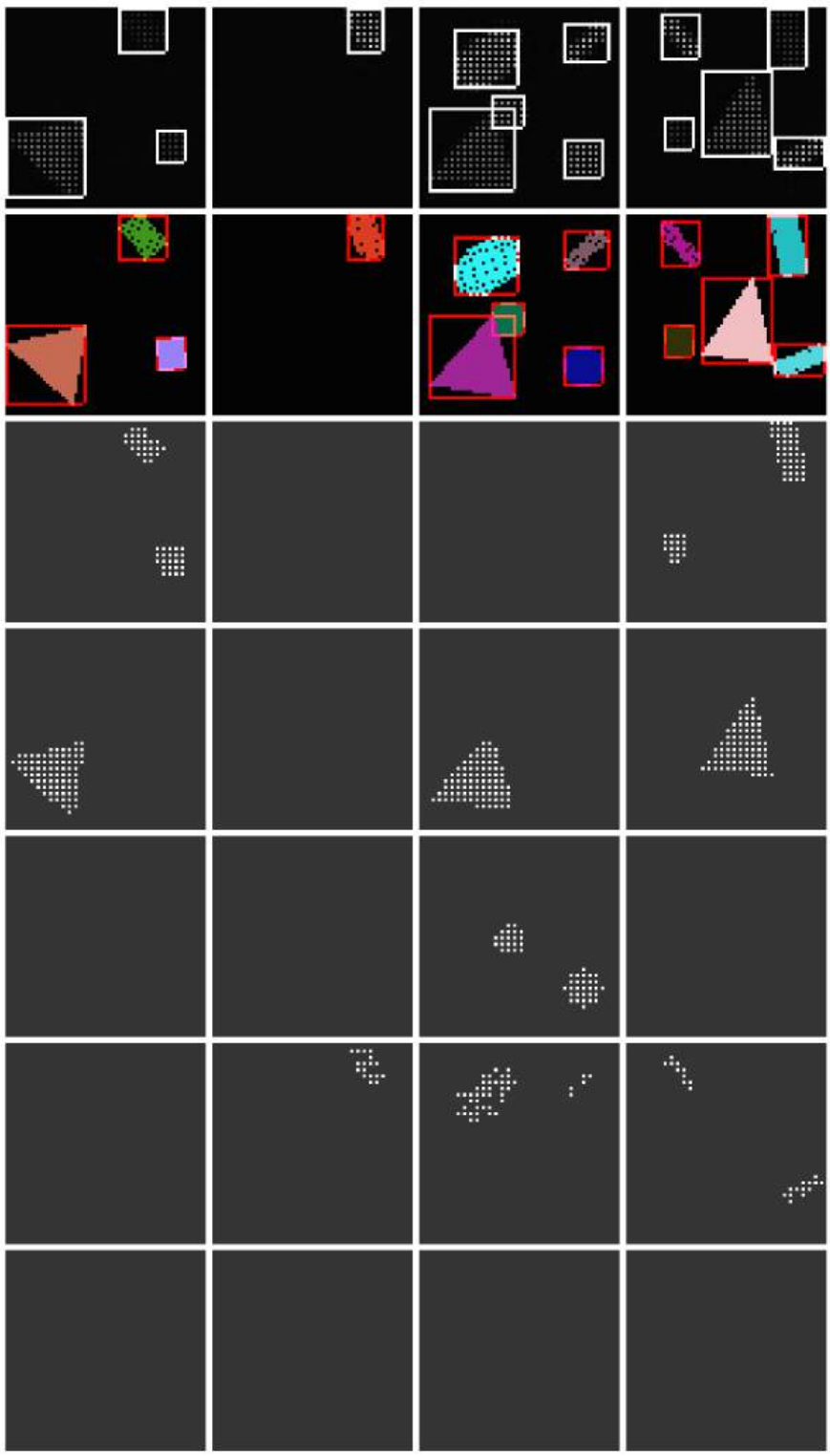
Output of learning rate 1e-5, epoch 12, batch 4



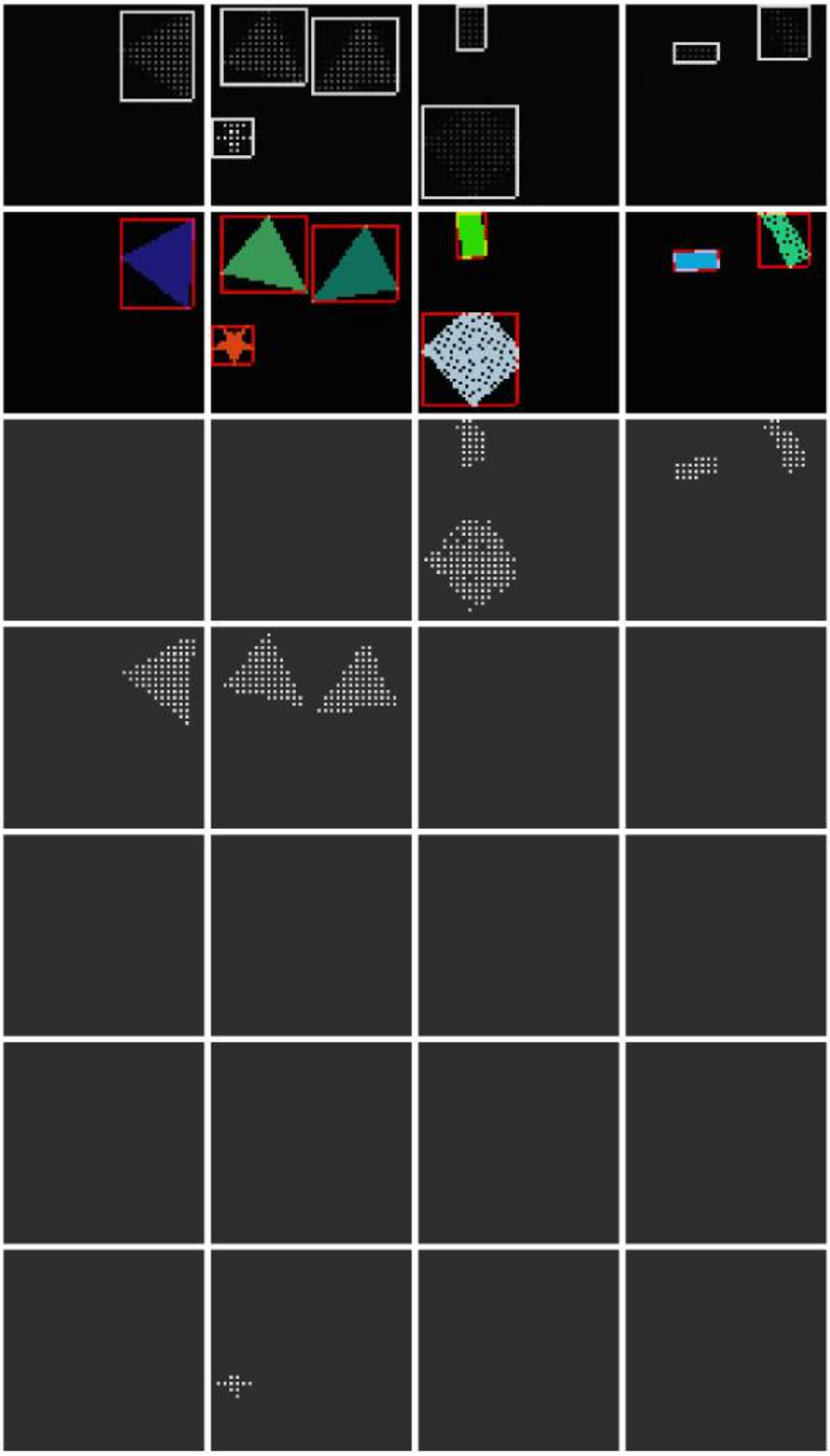
Batch 1



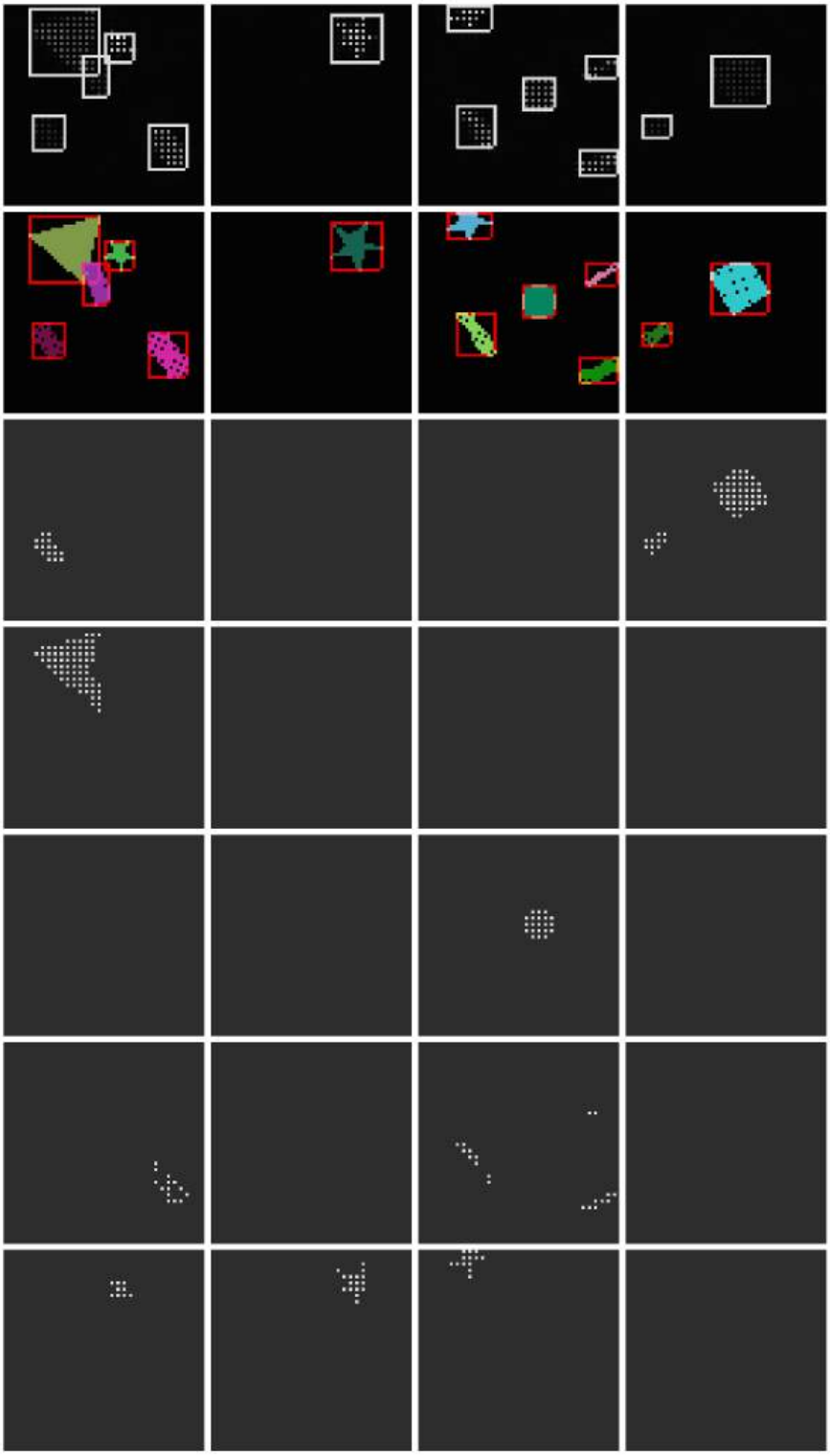
Batch 51



Batch 101

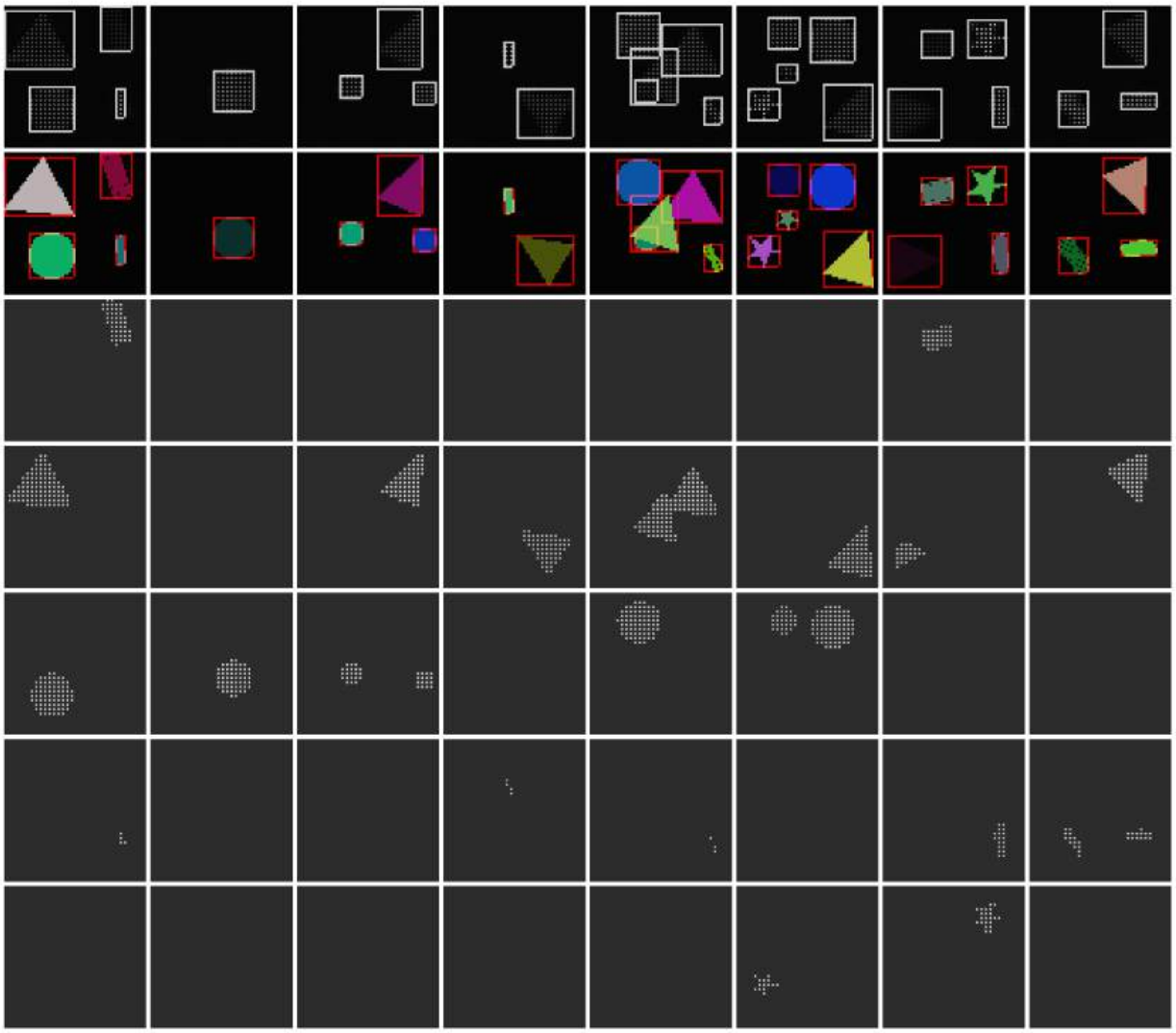


Batch 151

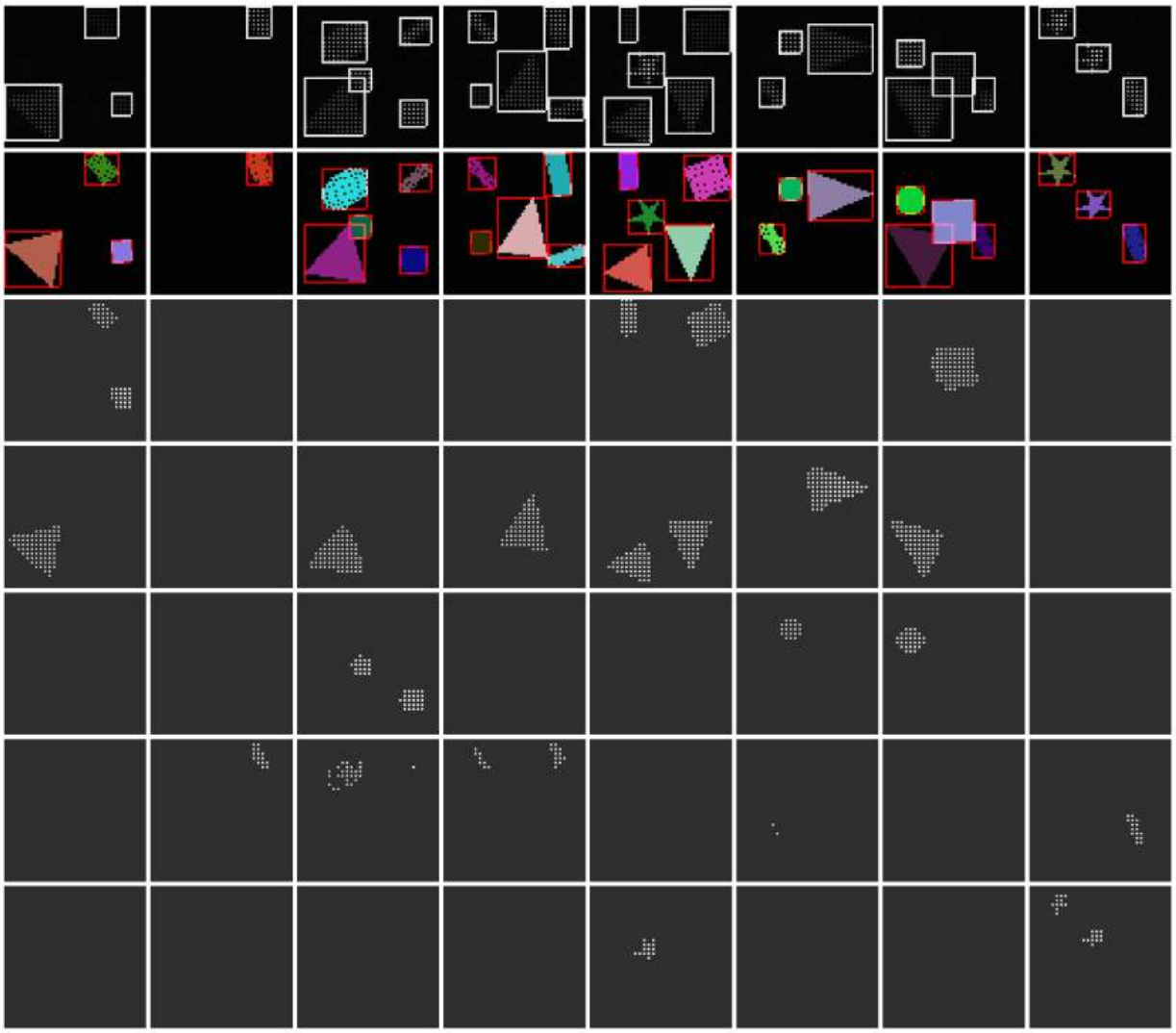


Batch 201

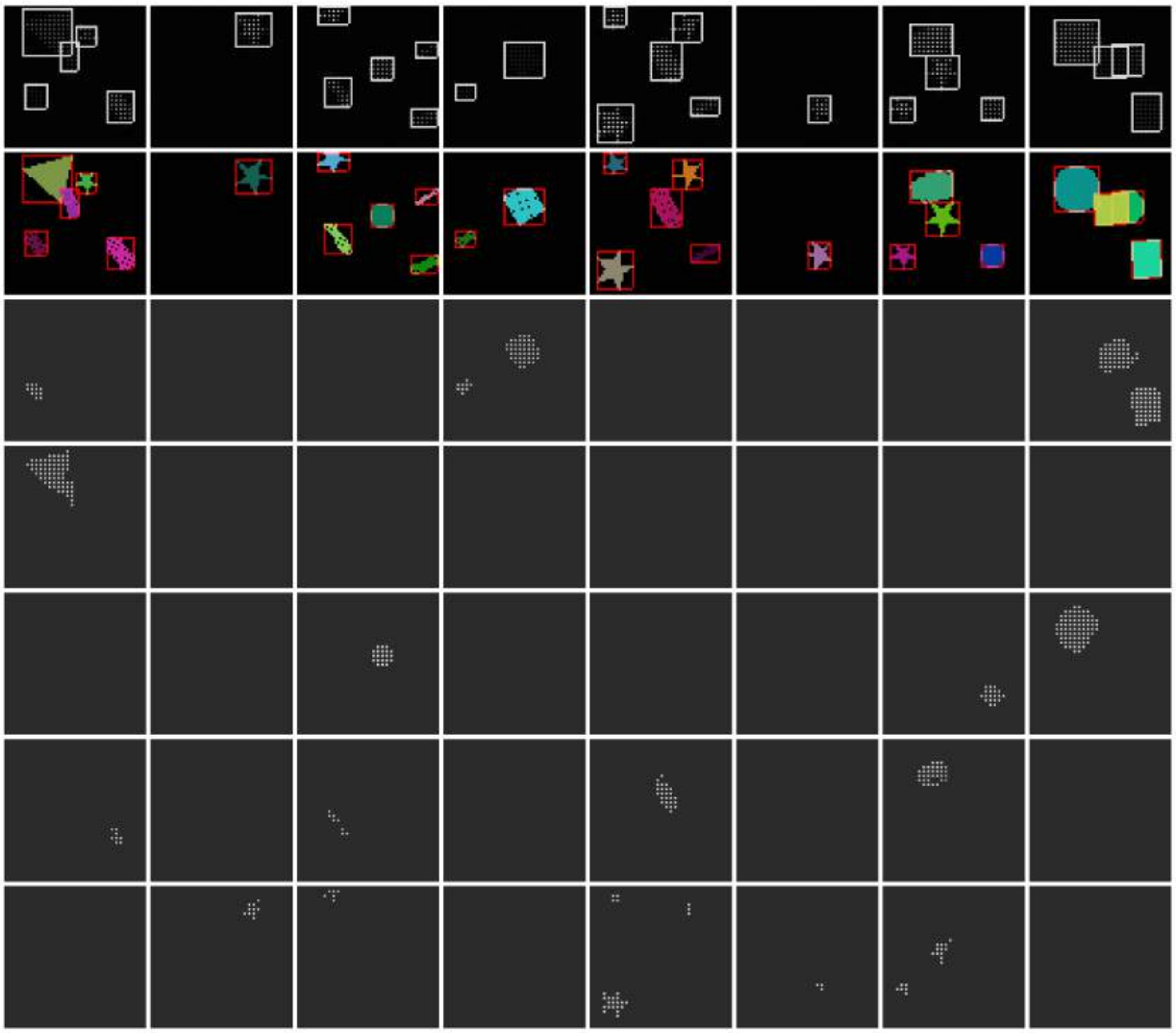
Output of learning rate 1e-5, epoch 12, batch 8



Batch 1



Batch 51

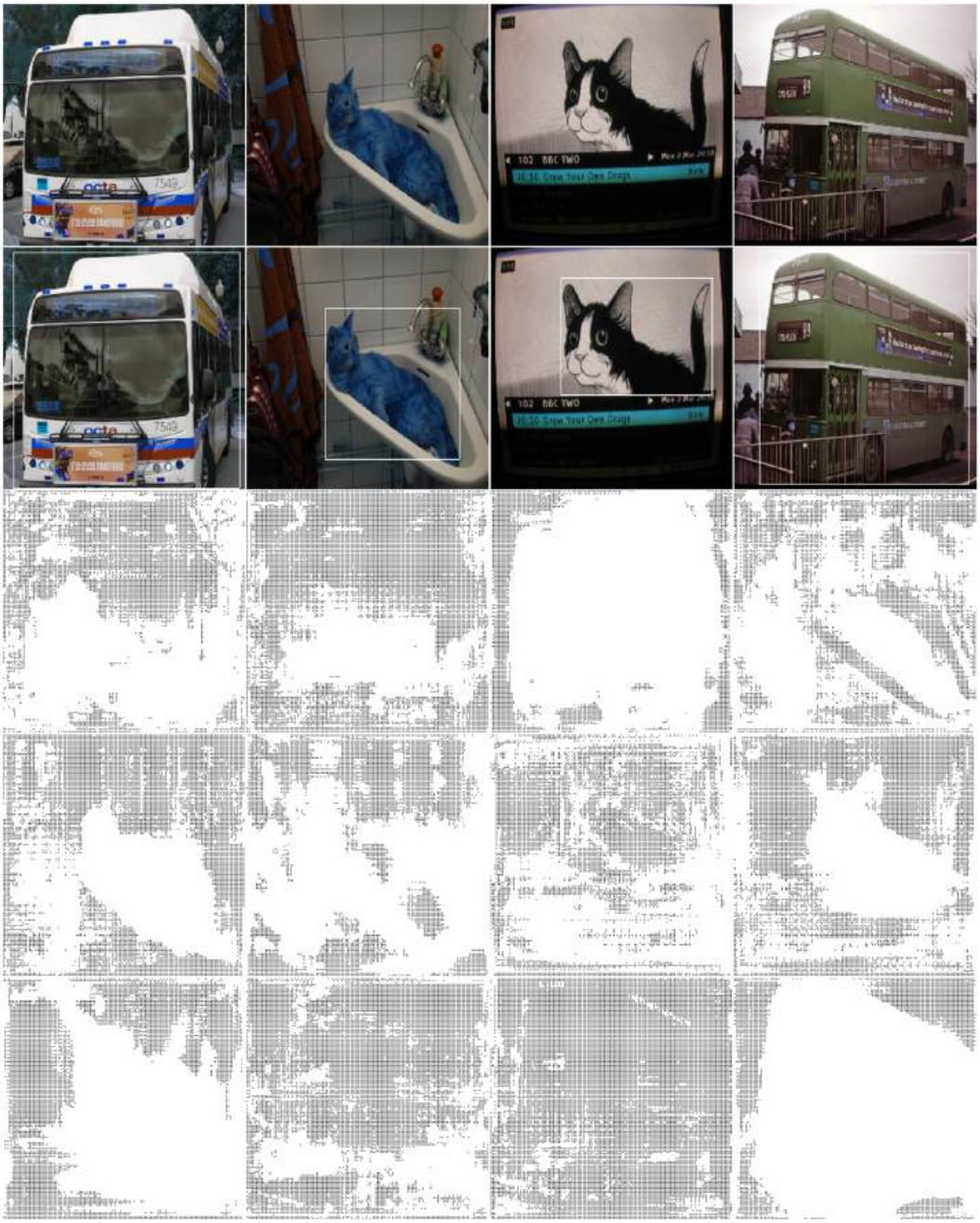


Batch 101

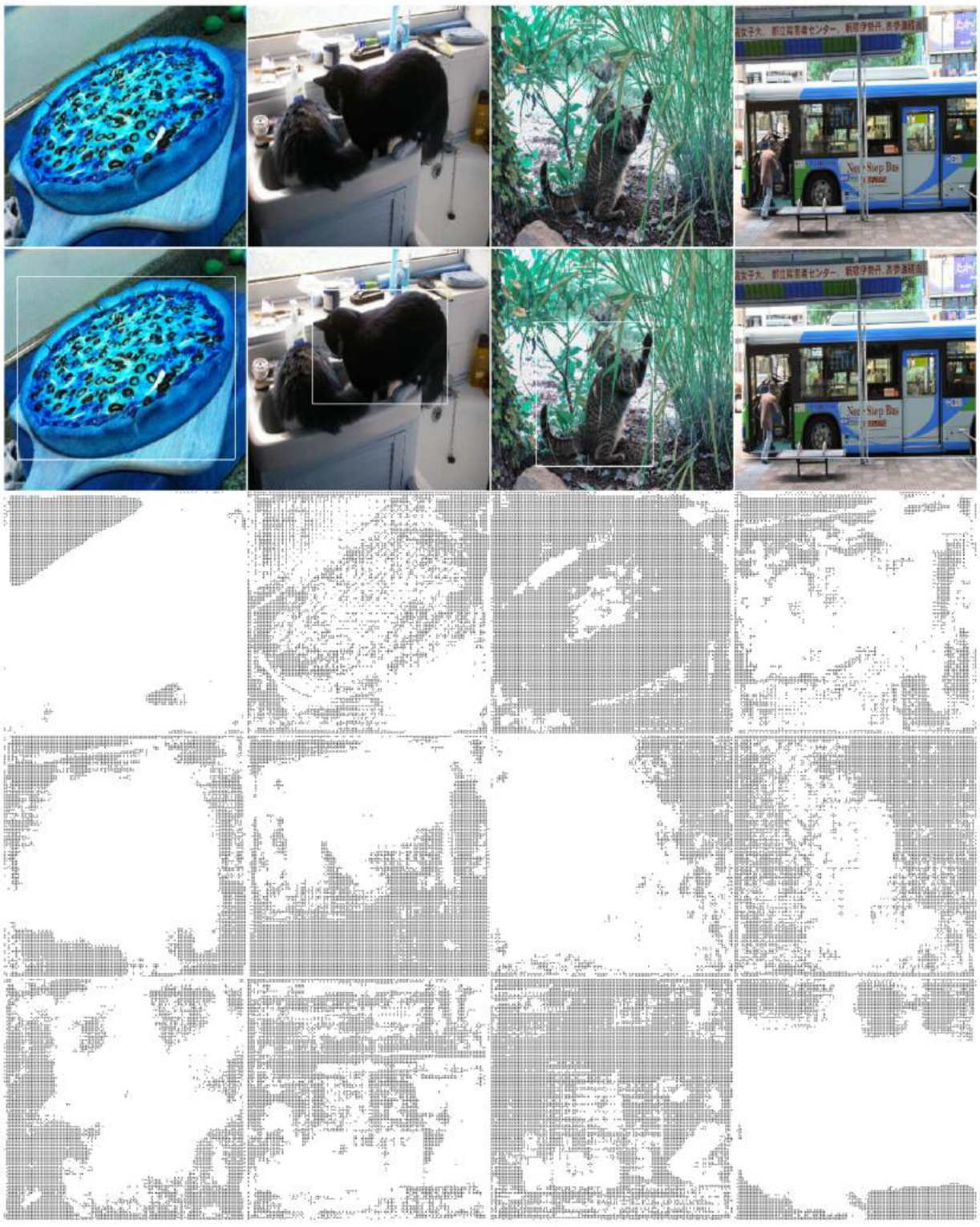
Test Results of MSCOCO

As I have already mentioned in the source code, while writing a report I found out I have made a possible mess in the demonstration of the original image, which is why the following test results will show different color than what is originally present in the COCO dataset. Given the time constraint, I have decided not to pursue this issue, as I will have to rerun all the MSCOCO based codes again.

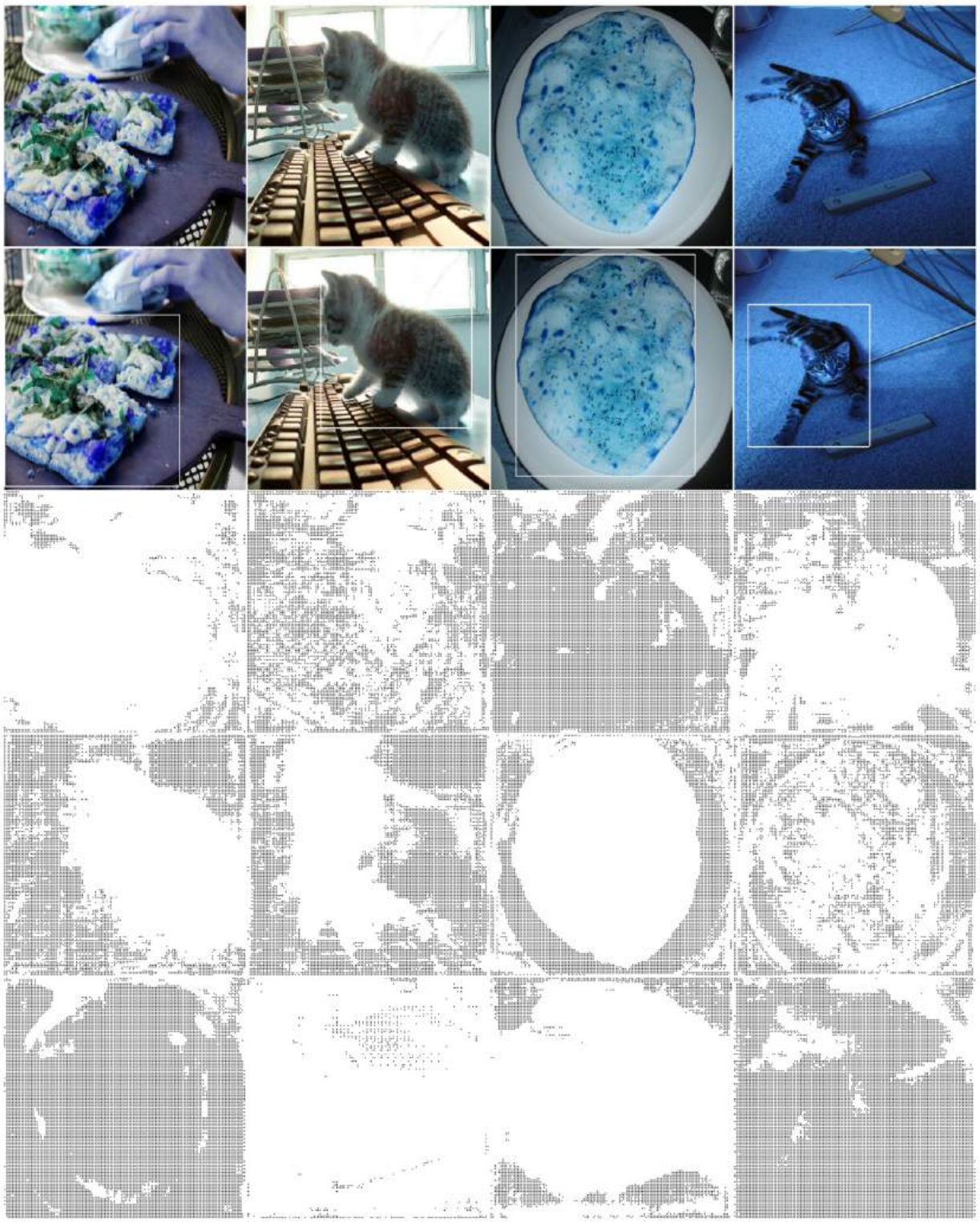
Output of learning rate 1e-4, epoch 20, batch 4



Batch 1



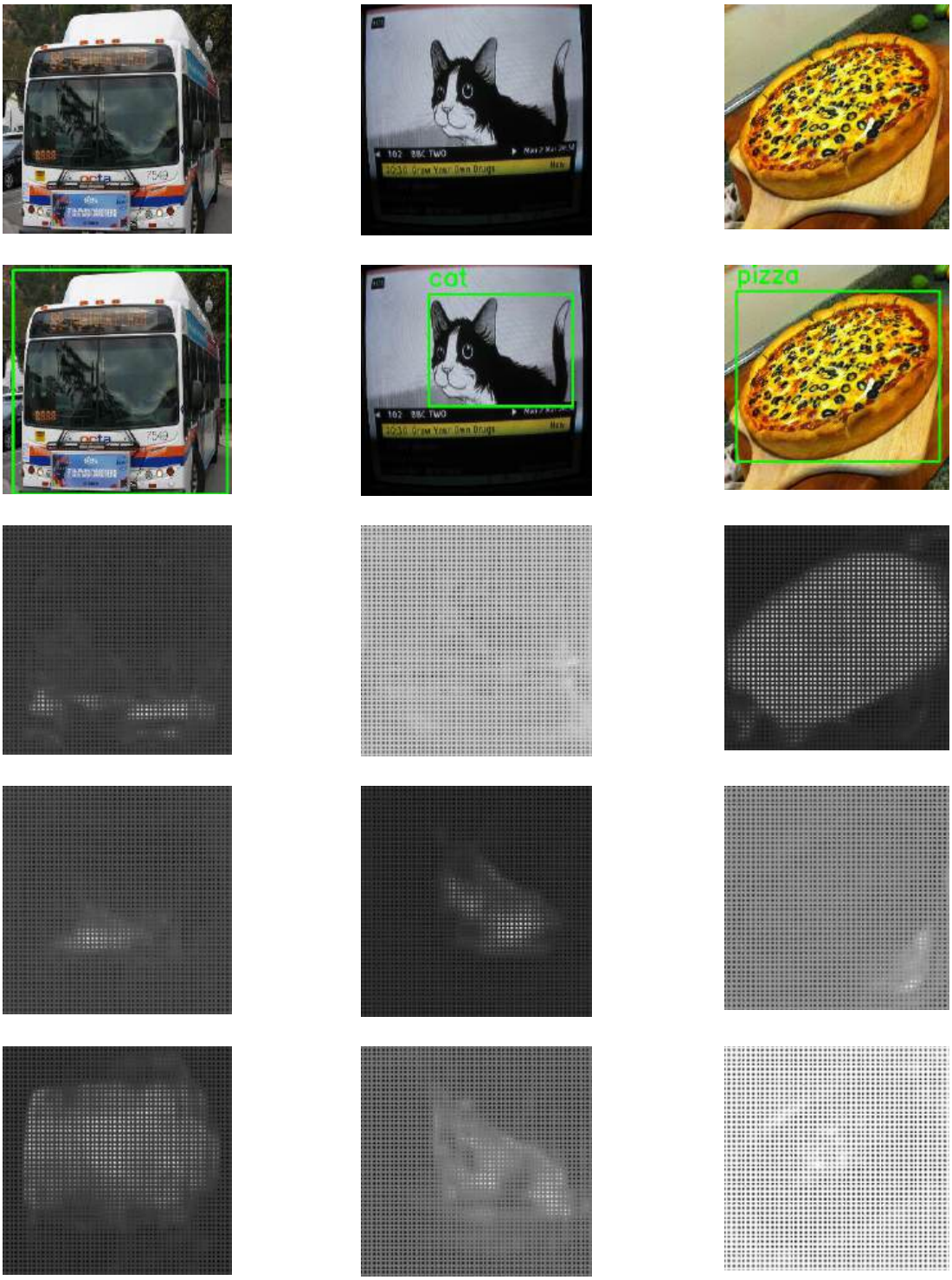
Batch 301



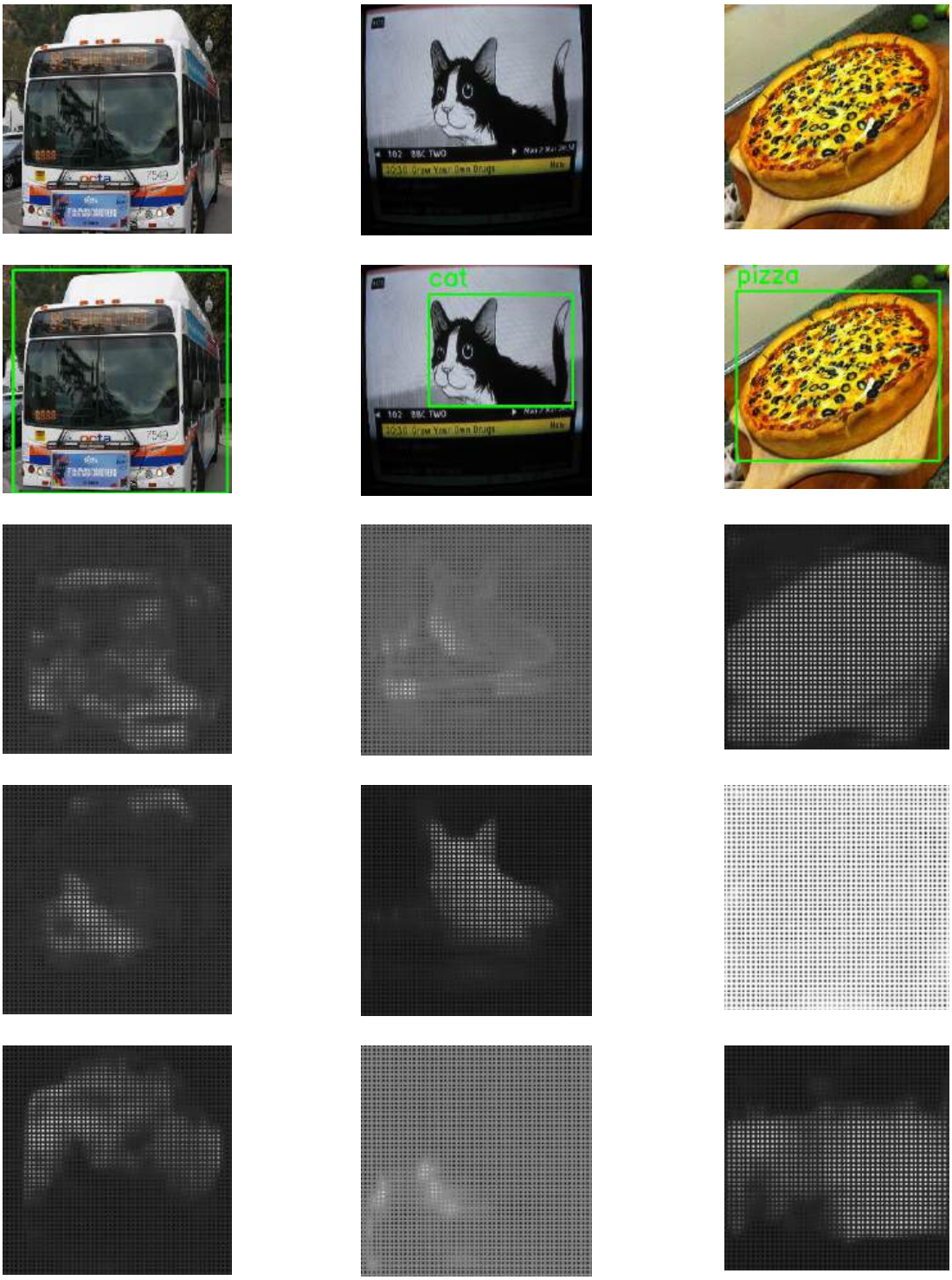
Batch 501

I have looked at the original images and handpicked 3 images that I will use to compare the results of different hyperparameter tuning. I am not considering batch here, meaning different images are from different batches. I will be displaying these handcrafted results from now on.

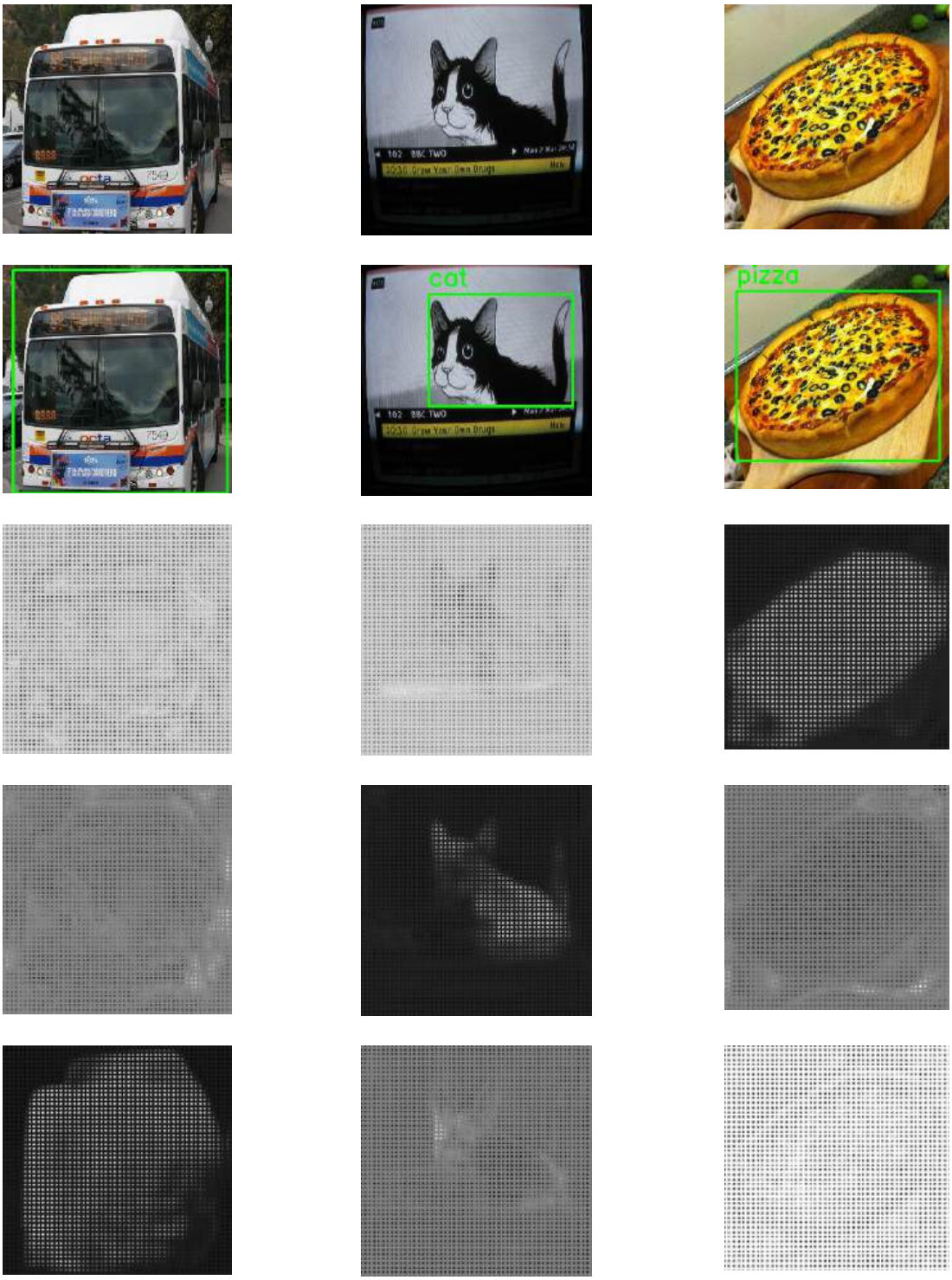
Output of learning rate 1e-4, epoch 20, batch 4



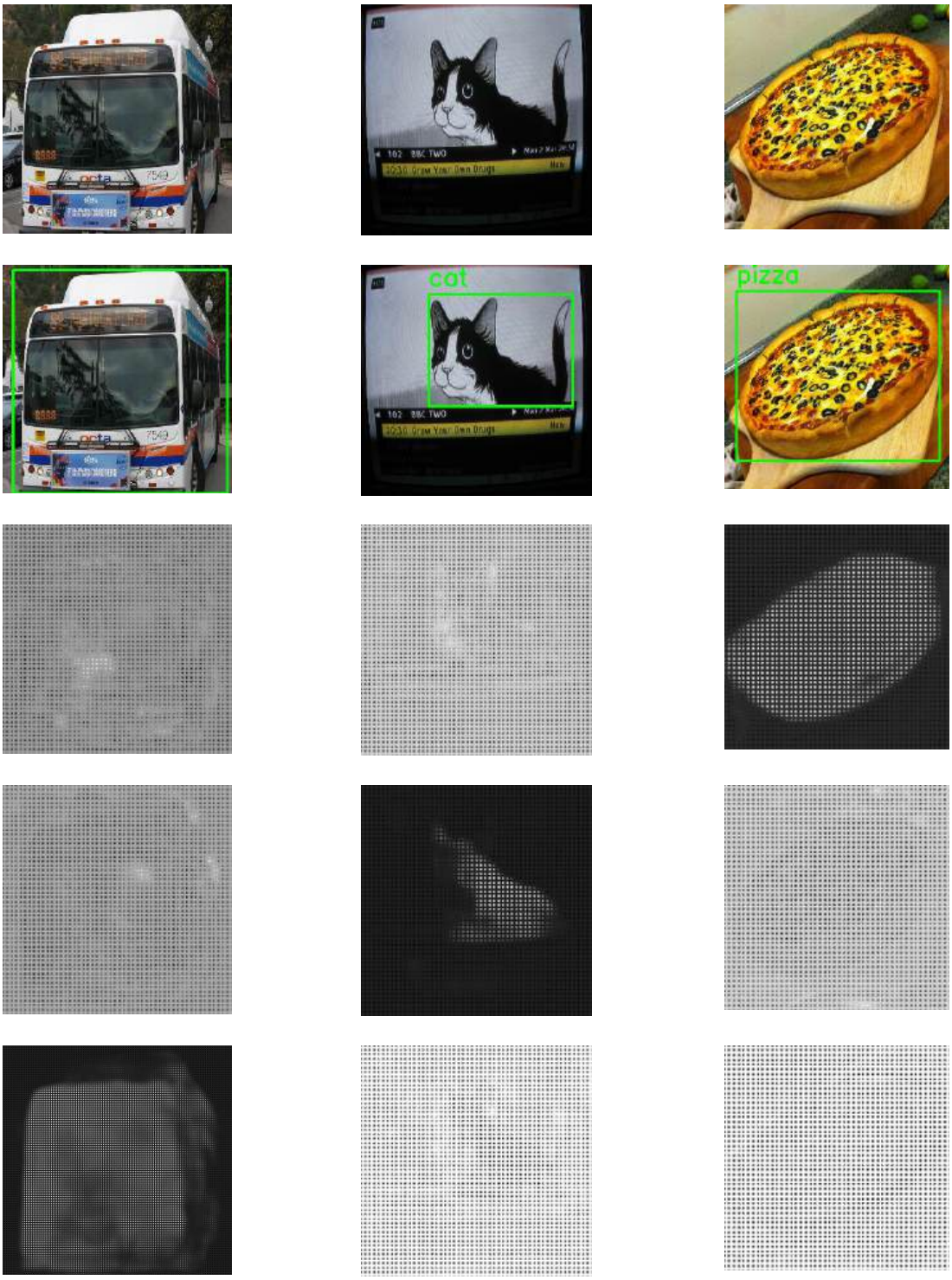
Output of learning rate 1e-4, epoch 30, batch 4



Output of learning rate 1e-5, epoch 20, batch 4



Output of learning rate 1e-5, epoch 30, batch 4



Brief understanding of mUnet and how it carries out semantic segmentation of an image.

From the basic mUnet, the architecture has an encoder that reduces spatial resolution while increasing the channel depth and the decoder that reconstructs the feature maps to the original image dimension with the help of skip connections to regain the fine details from the earlier layers. It is basically following and modifying the motivation of U-net. With the addition of the ASPP module at the bottleneck, right before the decoder, the model can examine the feature maps at multiple scales which can improve the model performance for detecting objects with varying sizes and shapes.

Introducing Dice Loss

I created my own dice loss function with the skeleton provided by the TA in the guideline. Also, had a slight help from previous year's solution 1. Here is my dice loss function:

```
  '''
  This is the required dice_function of which the skeleton was provided
  '''
  def dice_loss(preds: torch.Tensor, ground_truth: torch.Tensor, epsilon=1e-6):
    # Flattened the prediction and ground truth vector first
    # Got this idea from the first solution of Spring 2024 Page 4
    #
https://engineering.purdue.edu/DeepLearn/2_best_solutions/2024/Homeworks/HW7/2BestSolu
tions/1.pdf
    preds_flat = preds.view(preds.size(0), preds.size(1), -1)
    ground_truth_flat = ground_truth.view(ground_truth.size(0), ground_truth.size(1),
-1)
    # Step 1: Compute Dice Coefficient
    numerator = torch.sum(preds_flat * ground_truth_flat, dim=-1)
    denominator = torch.sum(preds_flat ** 2, dim=-1) + torch.sum(ground_truth_flat **
2, dim=-1)
    # Step 2: dice_coefficient = 2*numerator / (denominator + epsilon)
    dice_coefficient = (2.0 * numerator) / (denominator + epsilon)  # Shape:
[batch_size, num_classes]
    # Step 3: Compute dice_loss  = 1 - dice_coefficient
    dice_loss = 1.0 - torch.mean(dice_coefficient)
    return dice_loss
```

Dice Loss Only Analysis

Training Loss of PurdueShapes5MultiObjectDataset

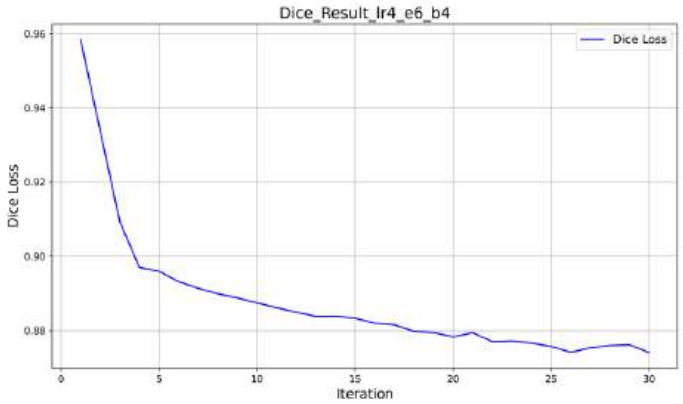


Fig: Loss curve for learning rate 1e-4, epoch 6, batch 4

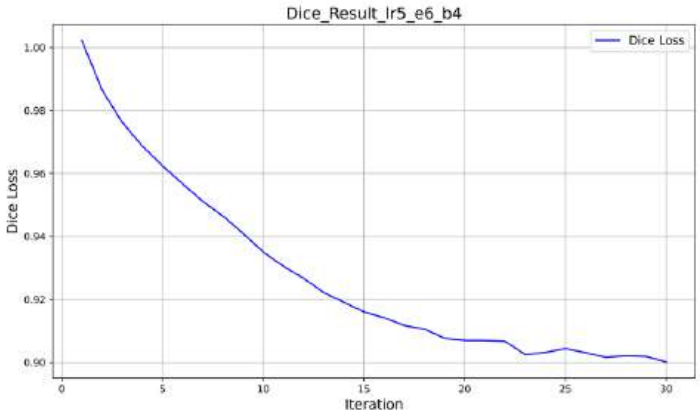


Fig: Loss curve for learning rate 1e-5, epoch 6, batch 4

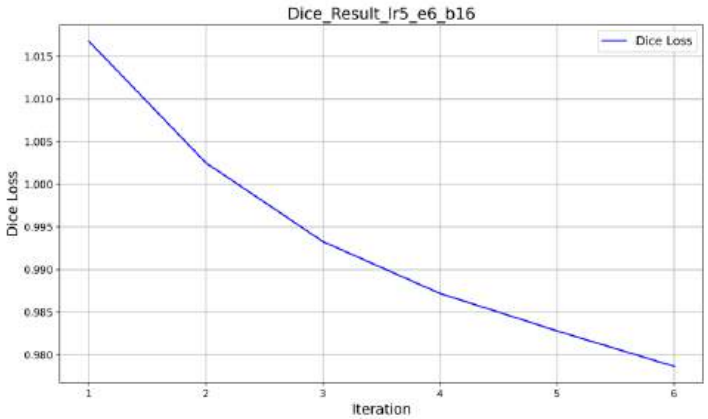


Fig: Loss curve for learning rate 1e-5, epoch 6, batch 16

Fig: Loss curve for learning rate 1e-5, epoch 12, batch 4

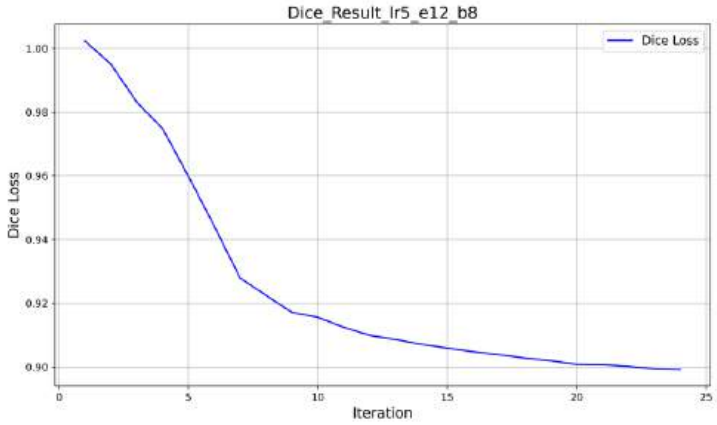


Fig: Loss curve for learning rate 1e-5, epoch 12, batch 8

Training Loss of MSCOCO

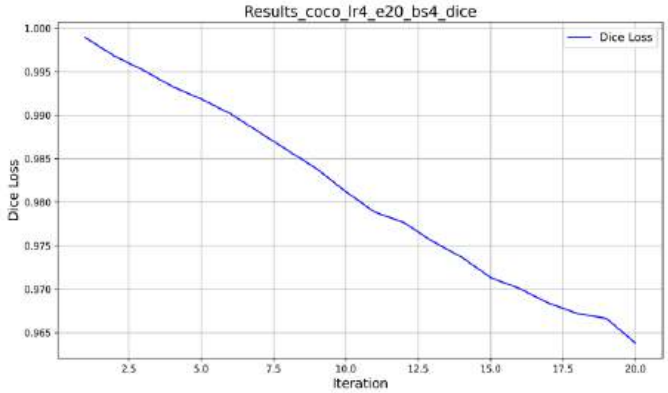


Fig: Loss curve for learning rate 1e-4, epoch 20, batch 4

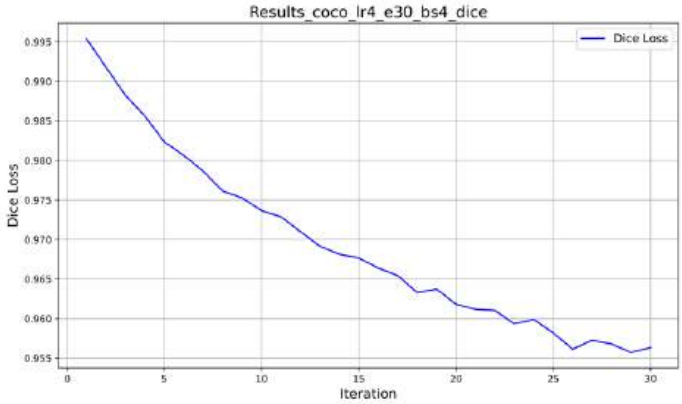


Fig: Loss curve for learning rate 1e-4, epoch 30, batch 4

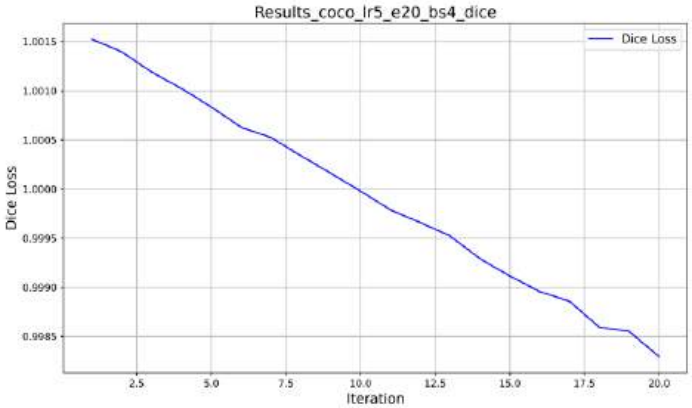


Fig: Loss curve for learning rate 1e-5, epoch 20, batch 4

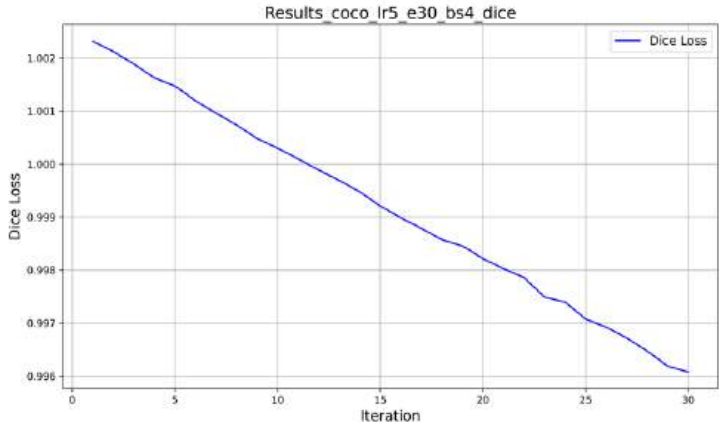
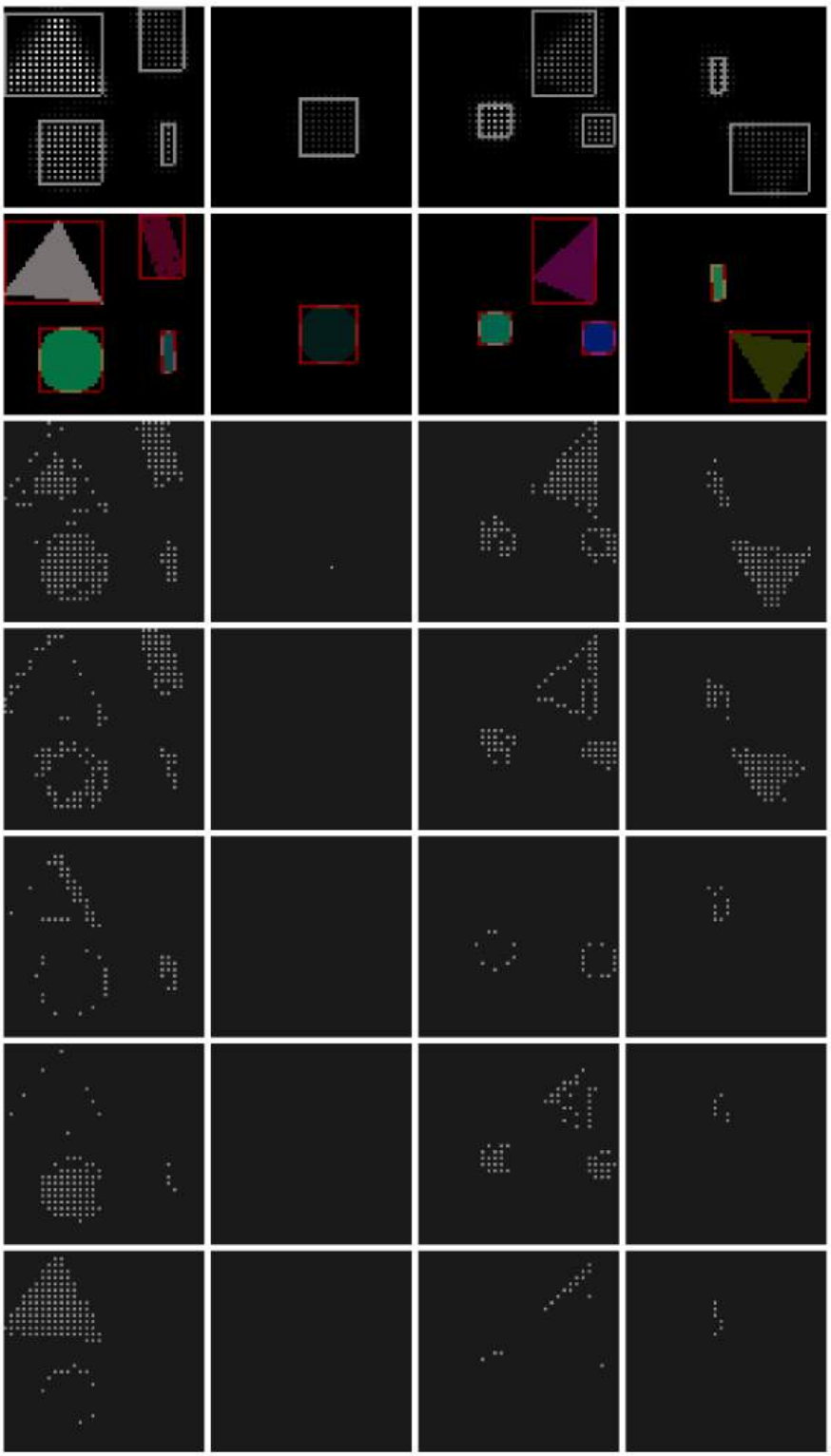


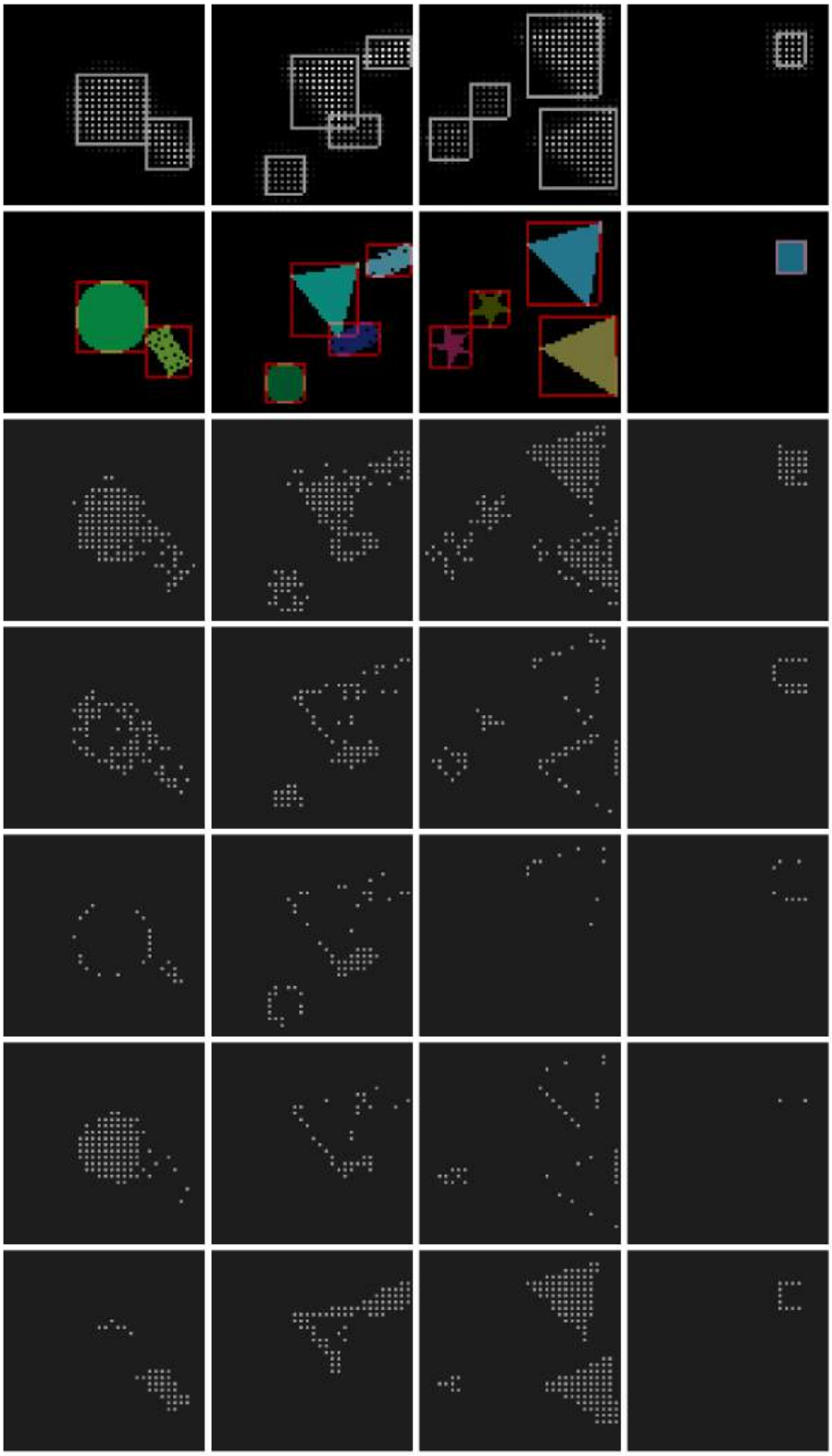
Fig: Loss curve for learning rate 1e-5, epoch 30, batch 4

Test Results of PurdueShapes5MultiObjectDataset

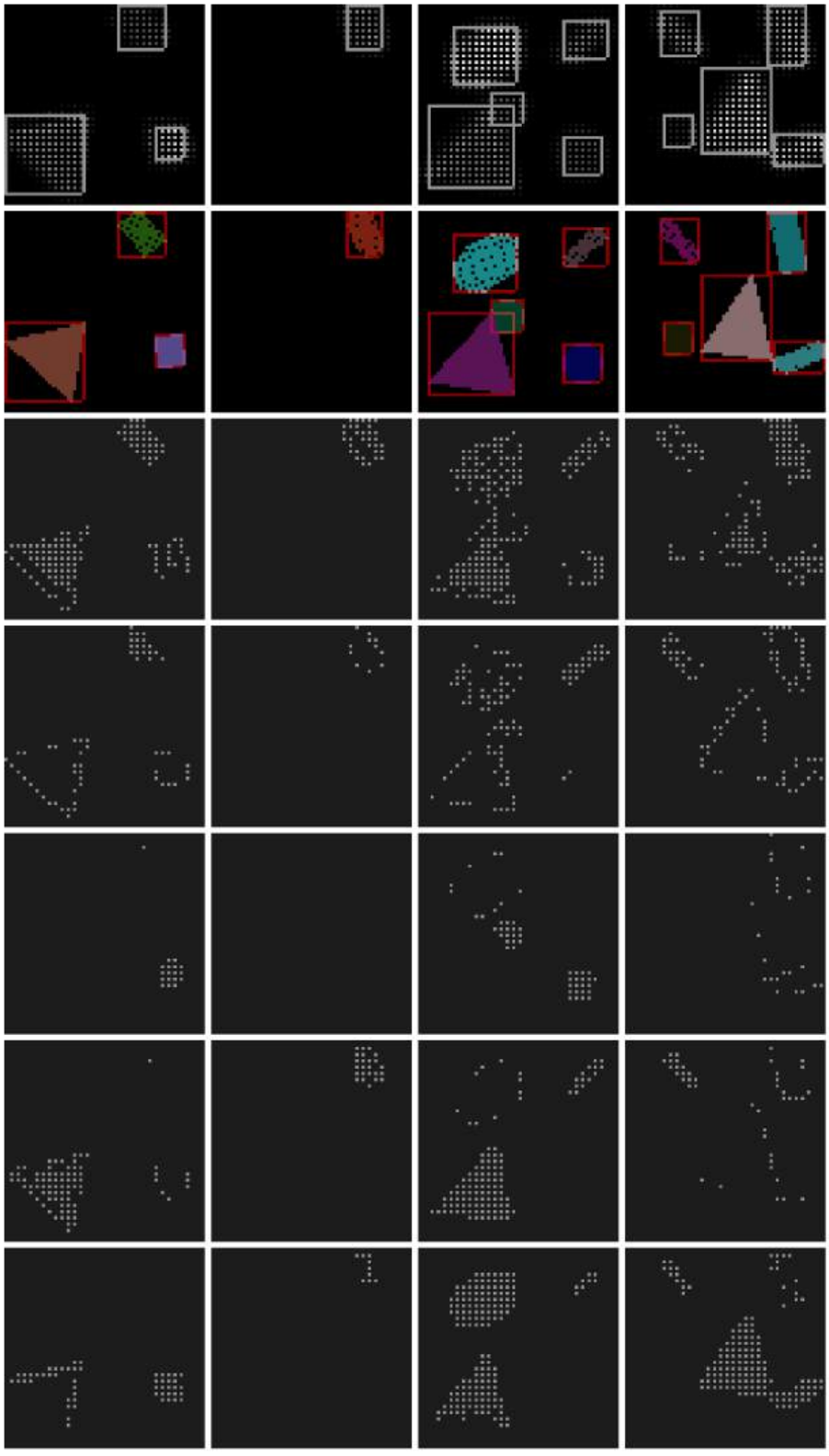
Output of learning rate 1e-4, epoch 6, batch 4



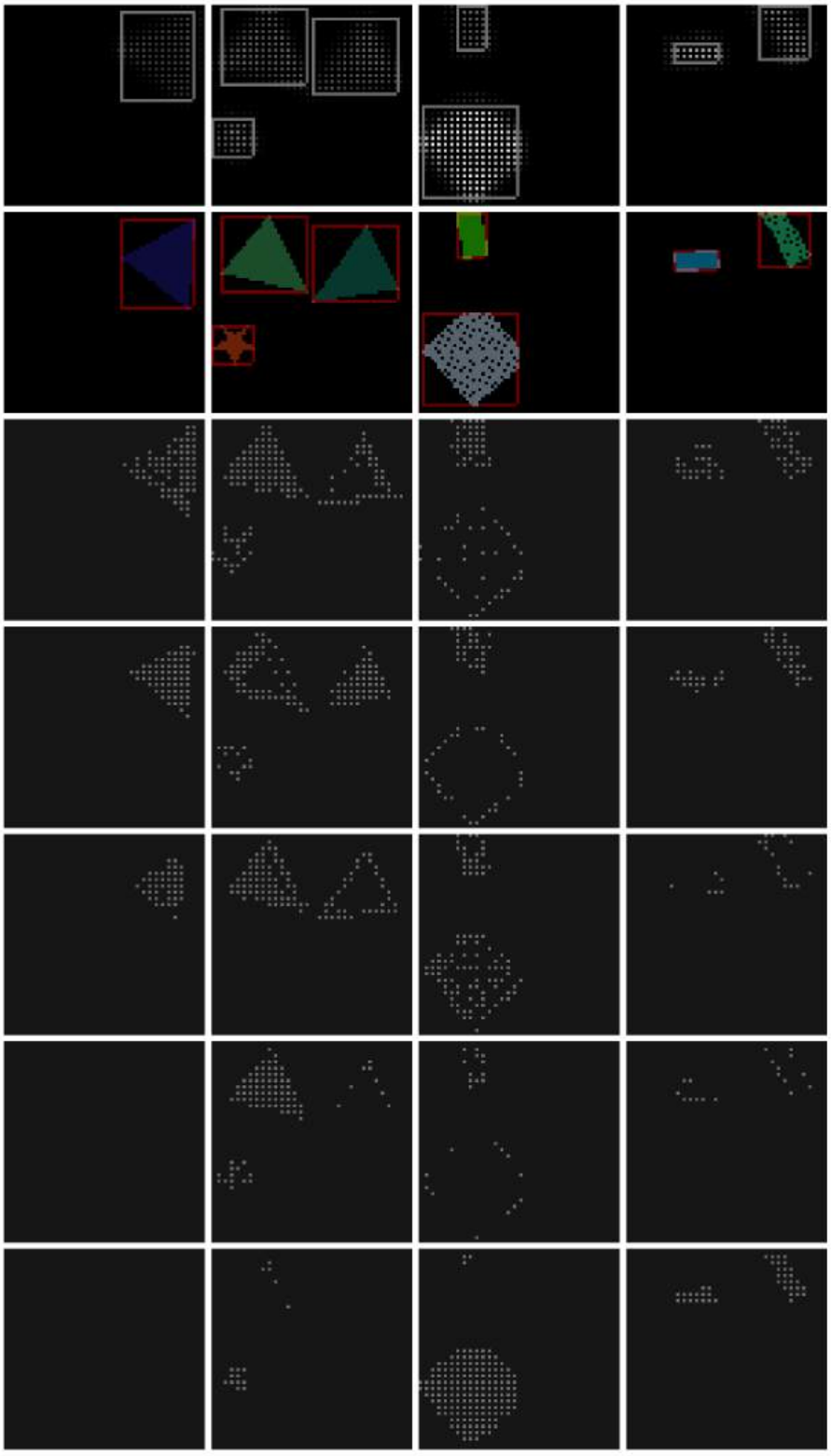
Batch 1



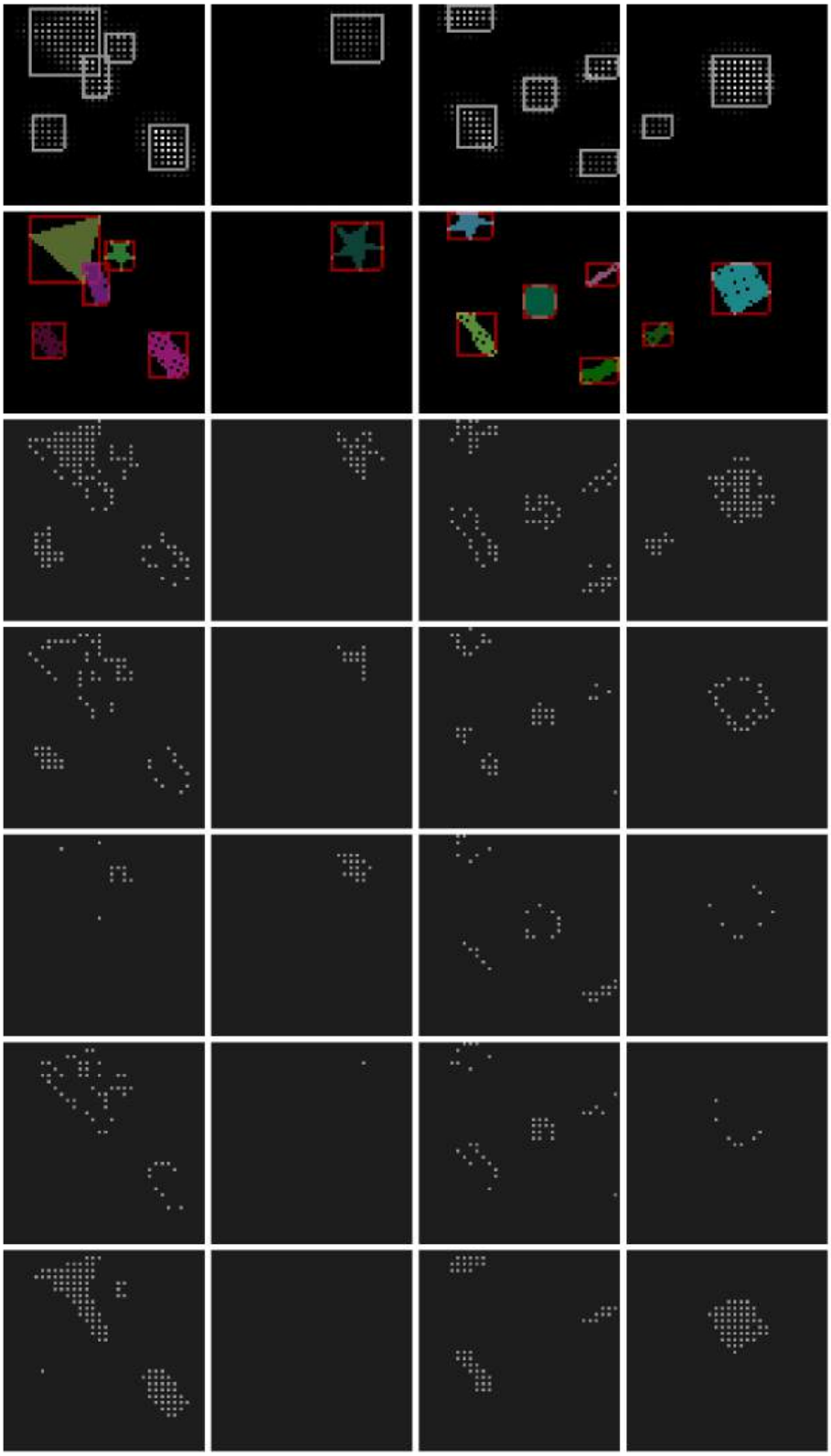
Batch 51



Batch 101

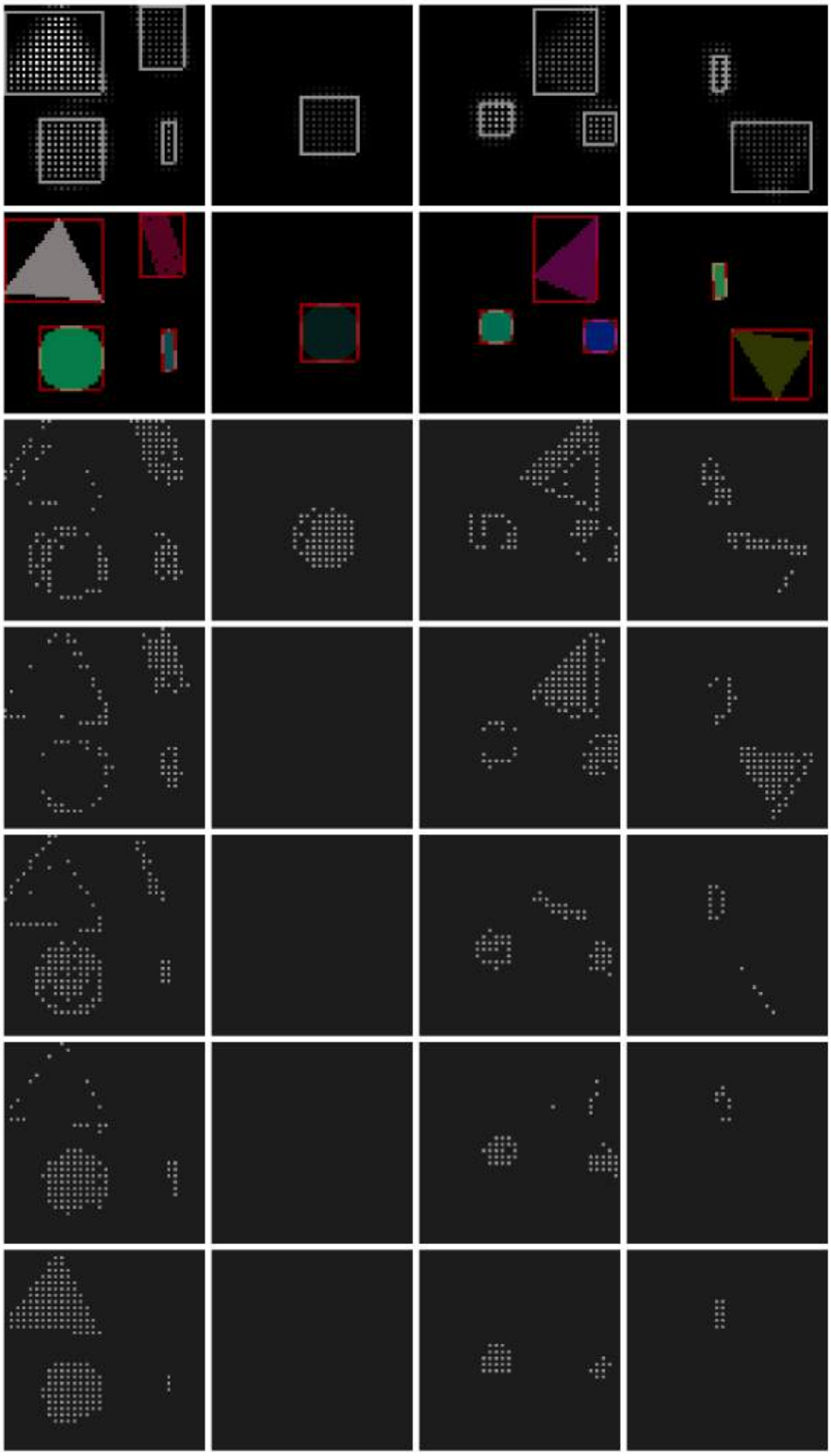


Batch 151

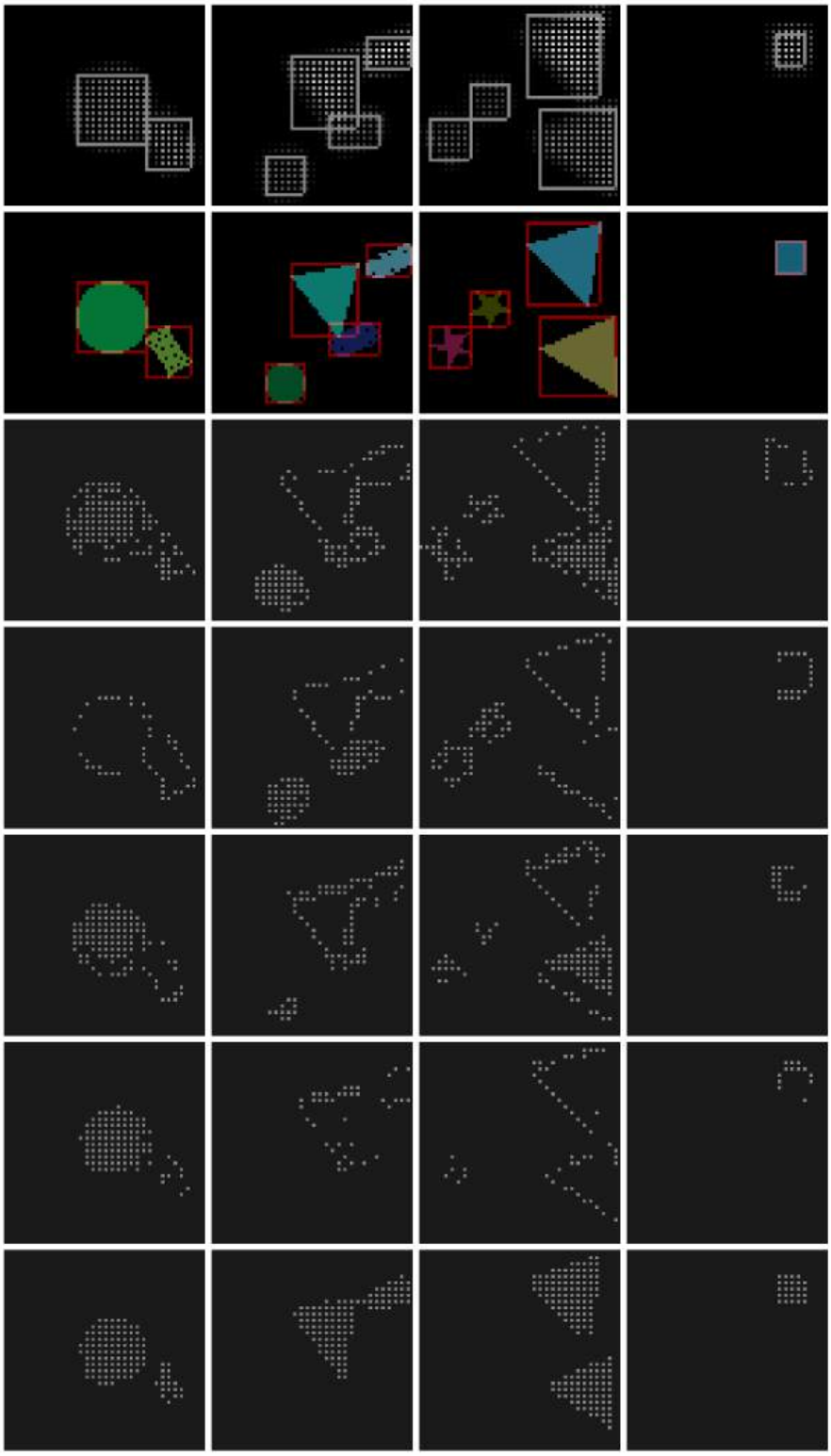


Batch 201

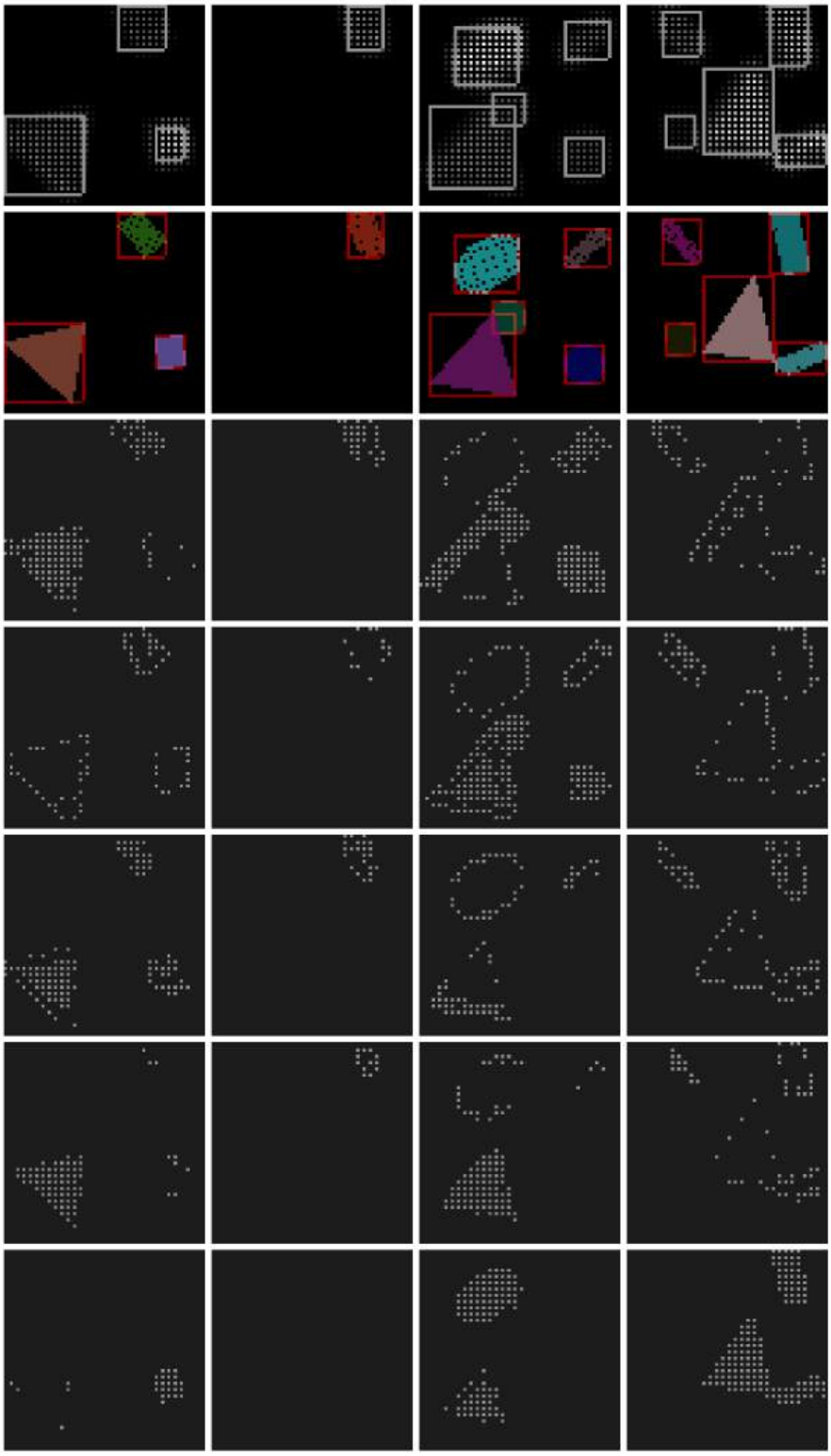
Output of learning rate 1e-5, epoch 6, batch 4



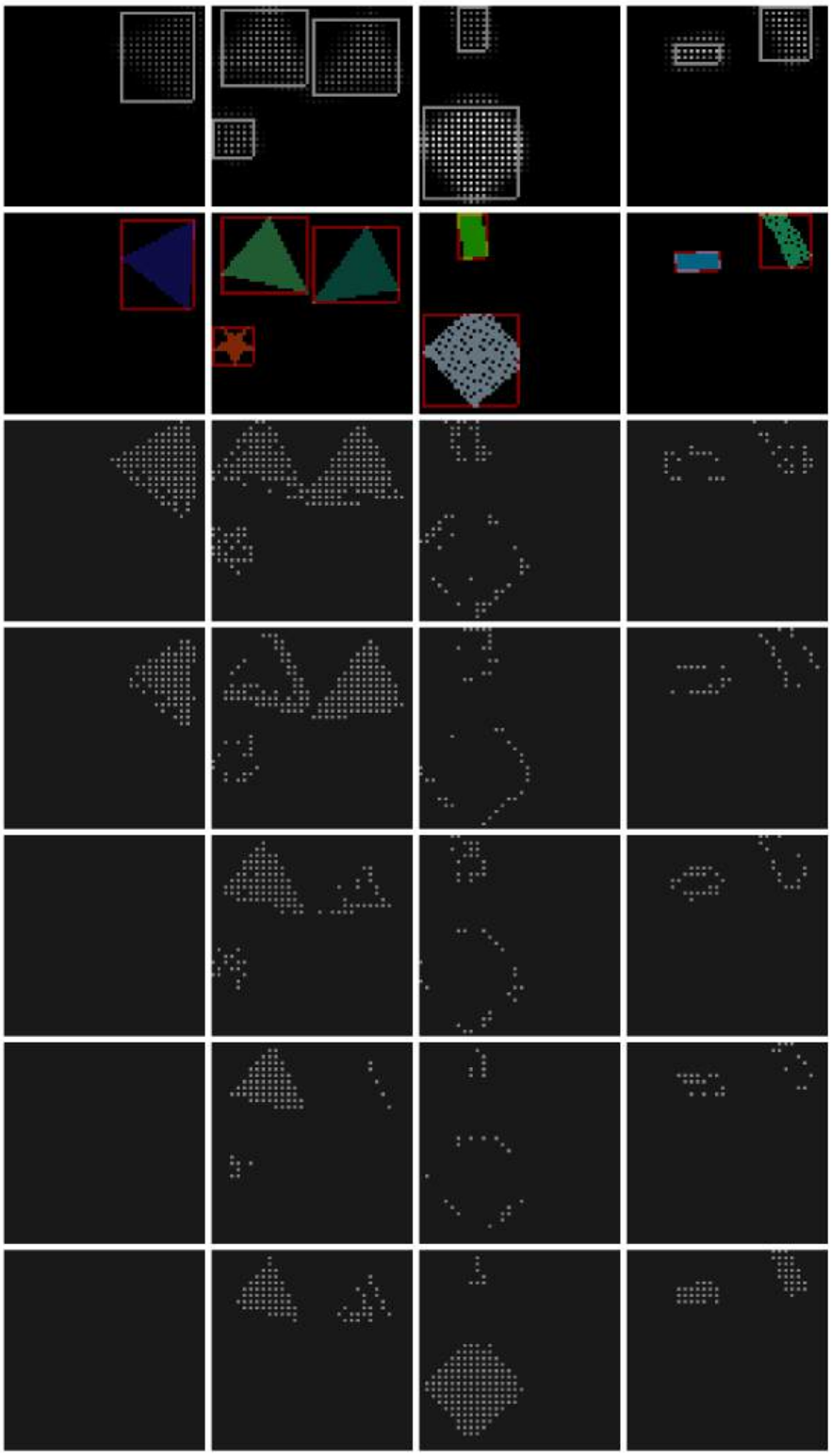
Batch 1



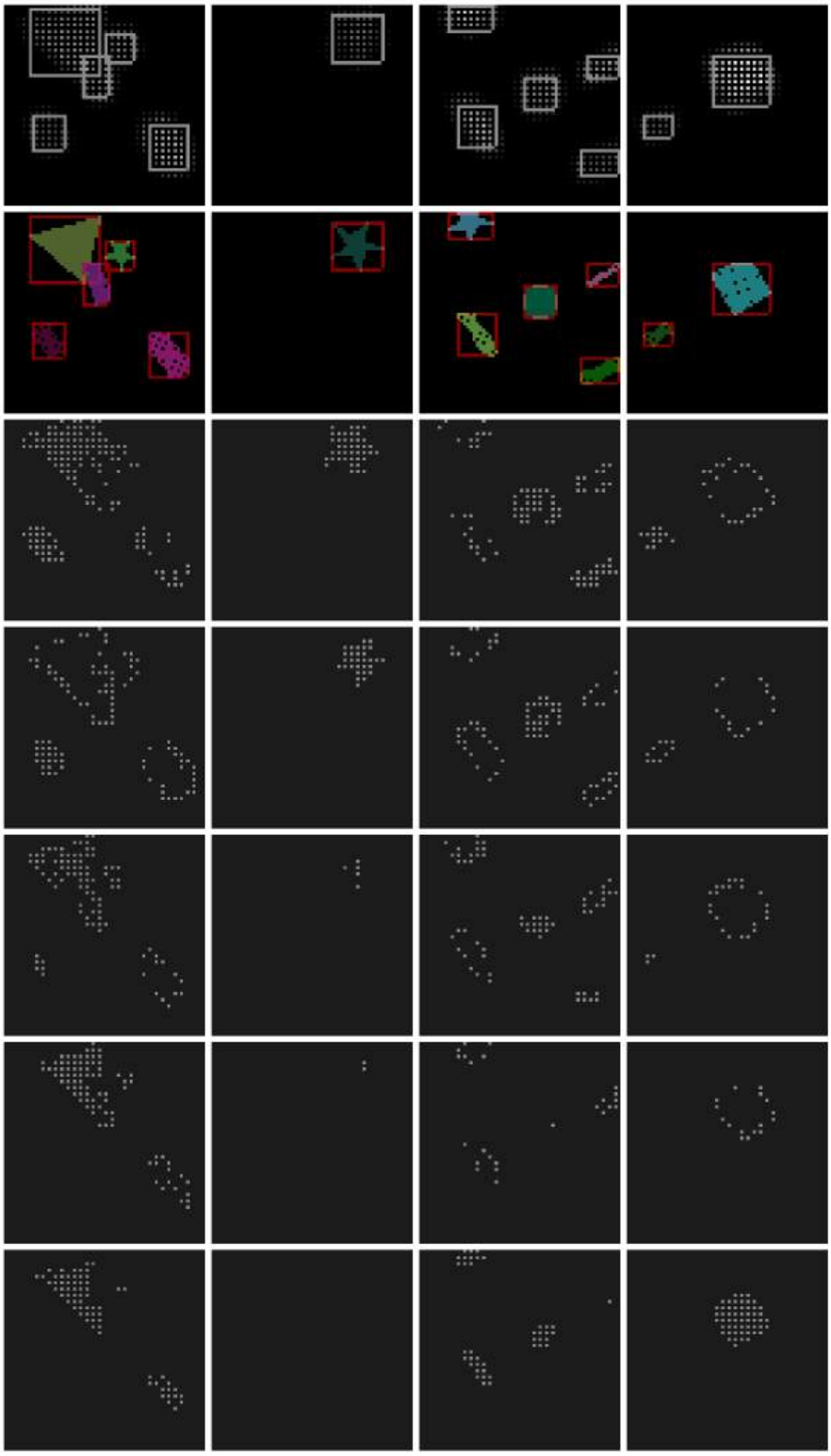
Batch 51



Batch 101

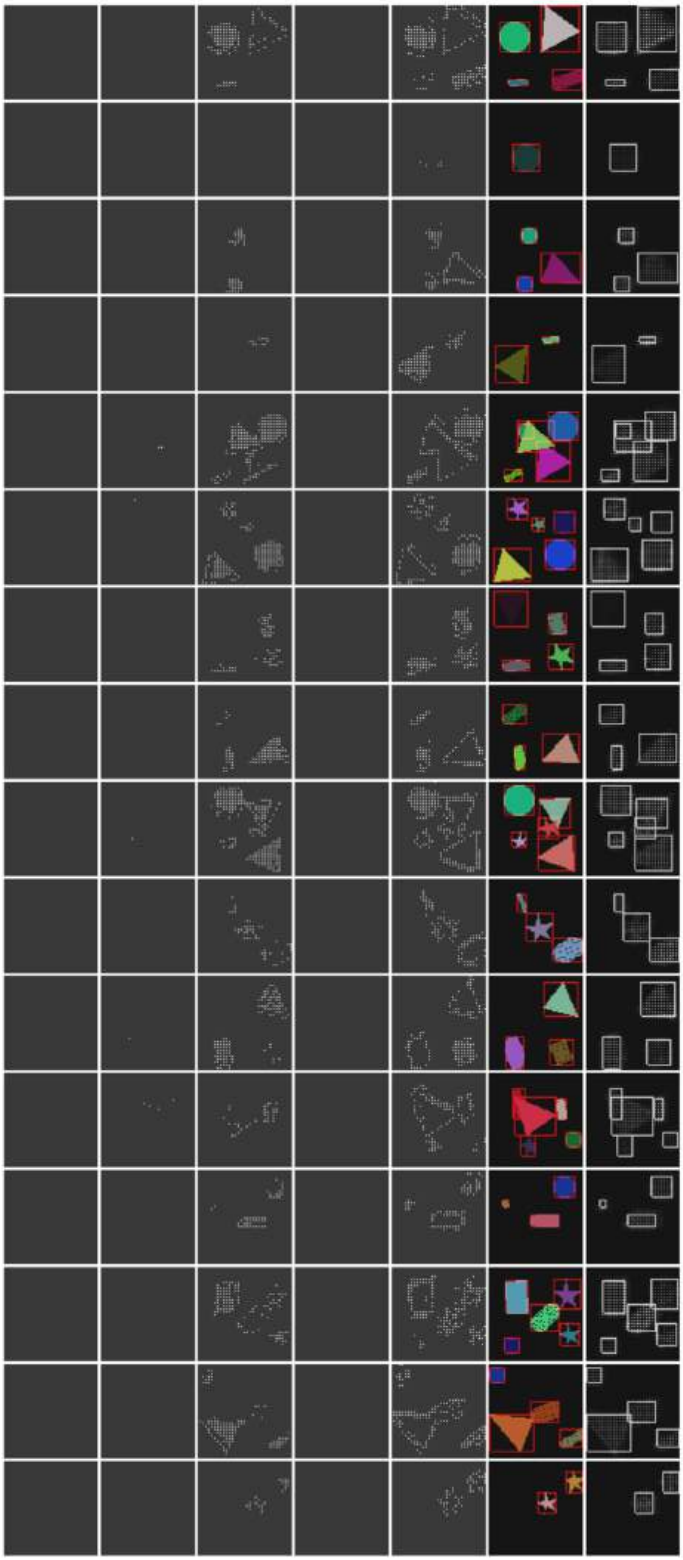


Batch 151



Batch 201

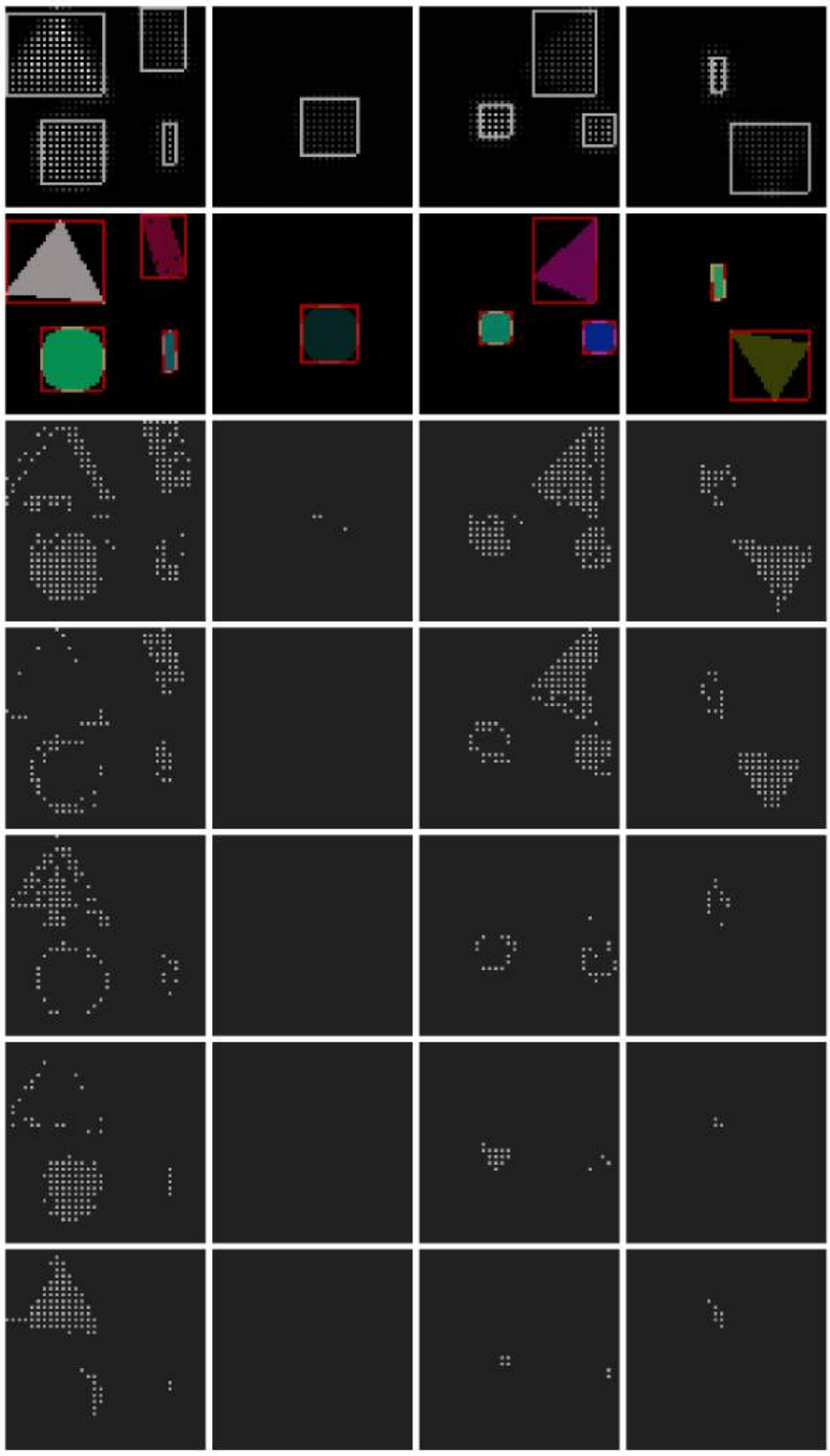
Output of learning rate 1e-5, epoch 6, batch 16



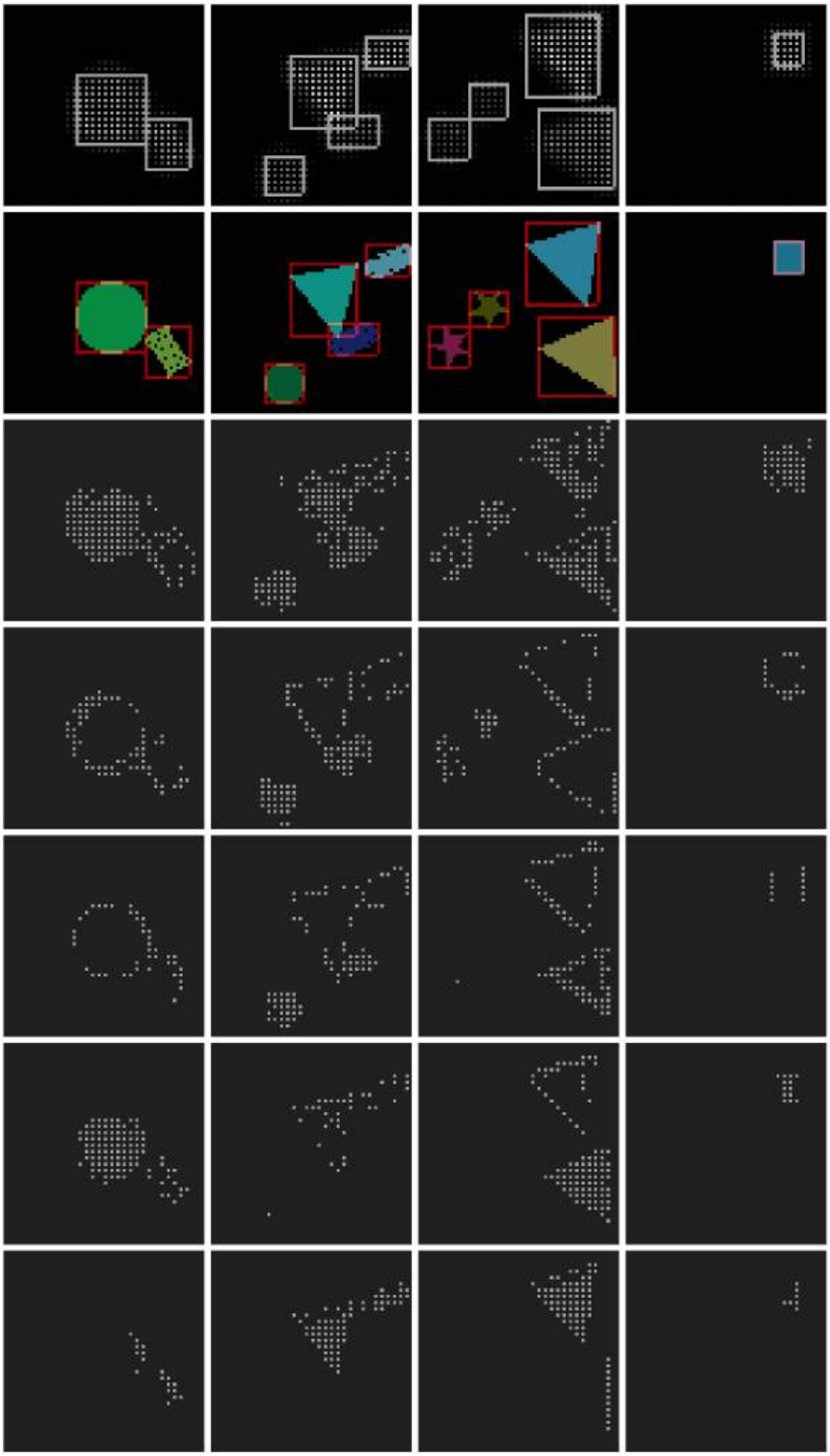
Batch 1

Batch 51

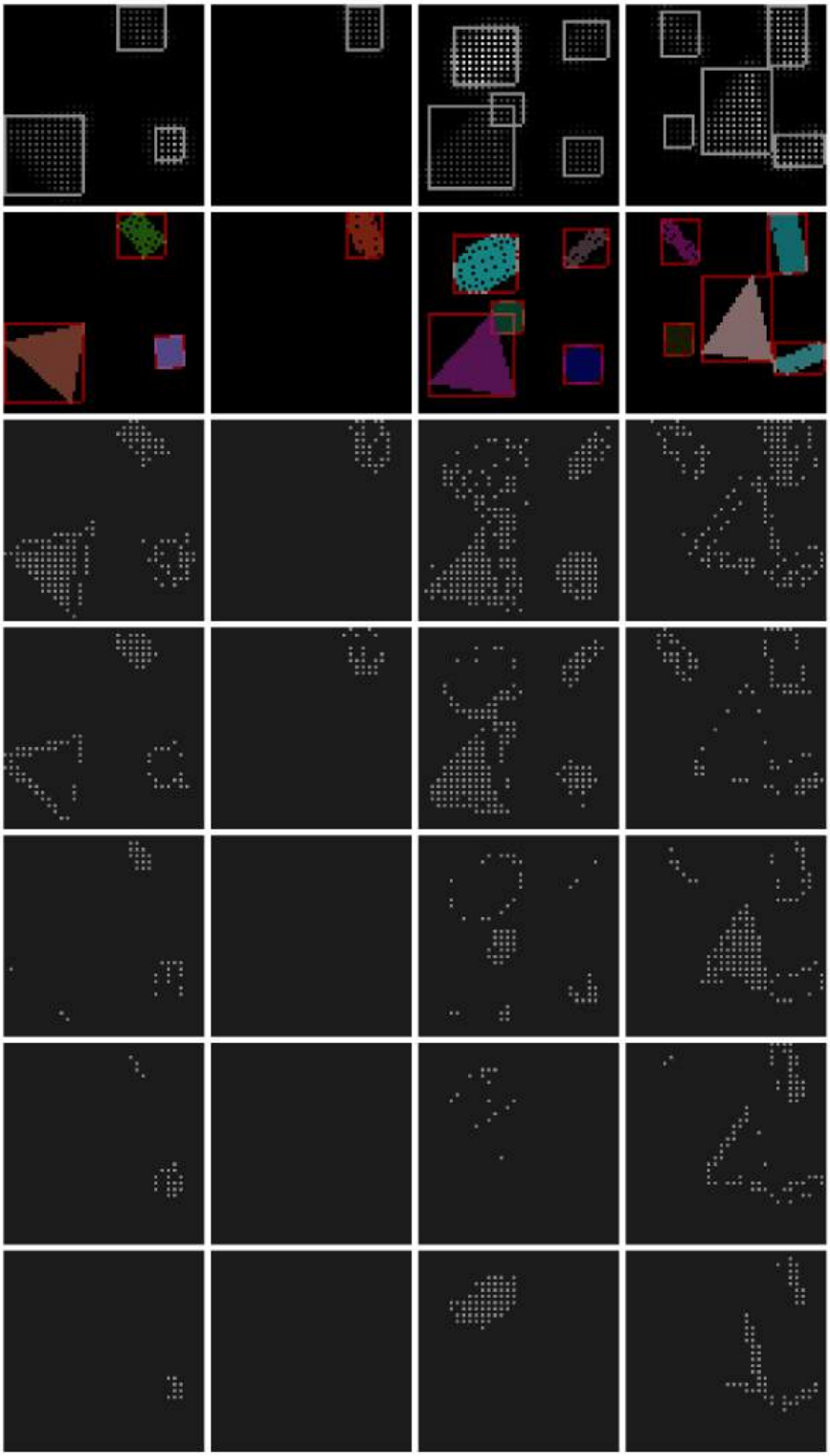
Output of learning rate 1e-5, epoch 12, batch 4



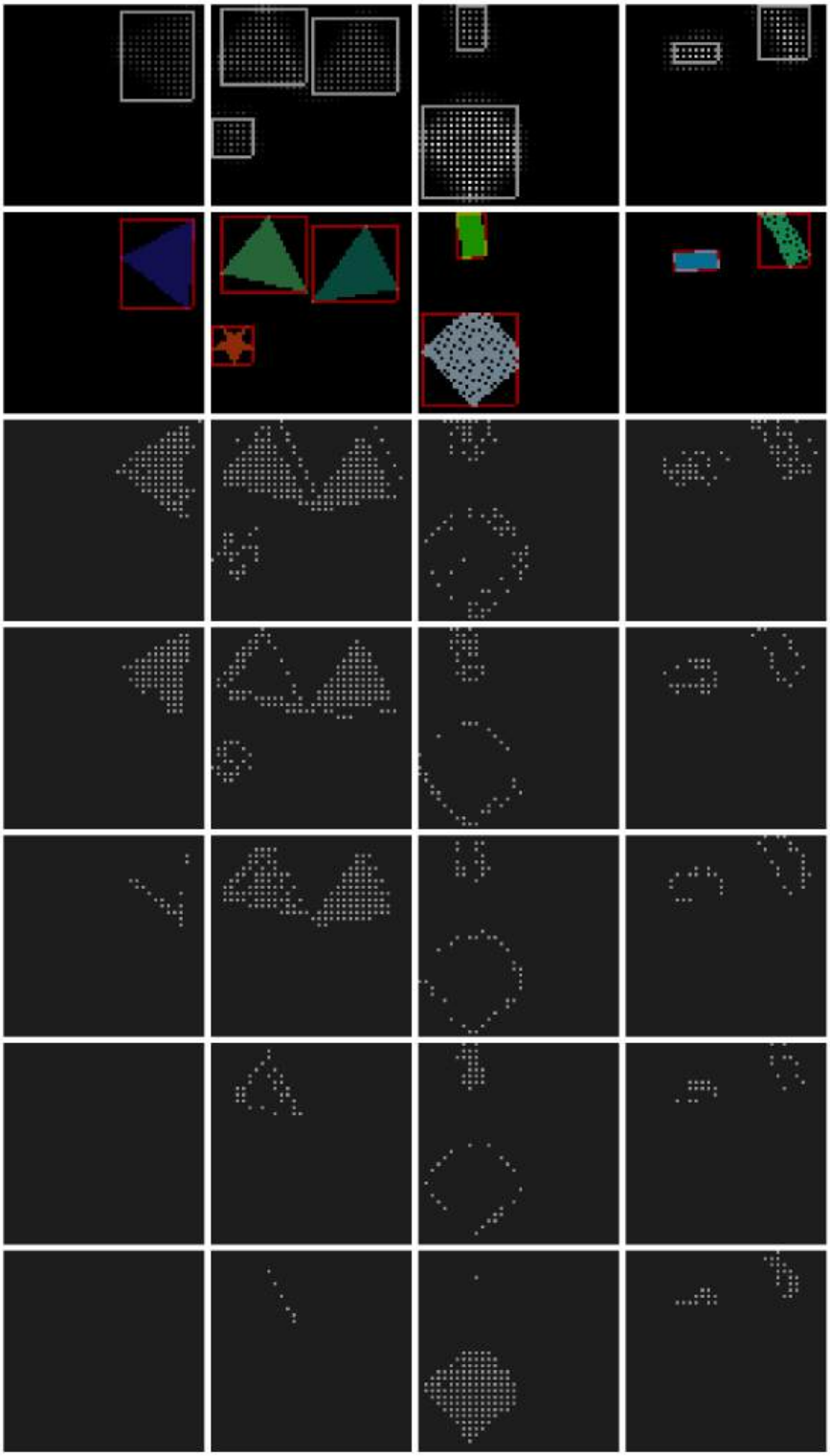
Batch 1



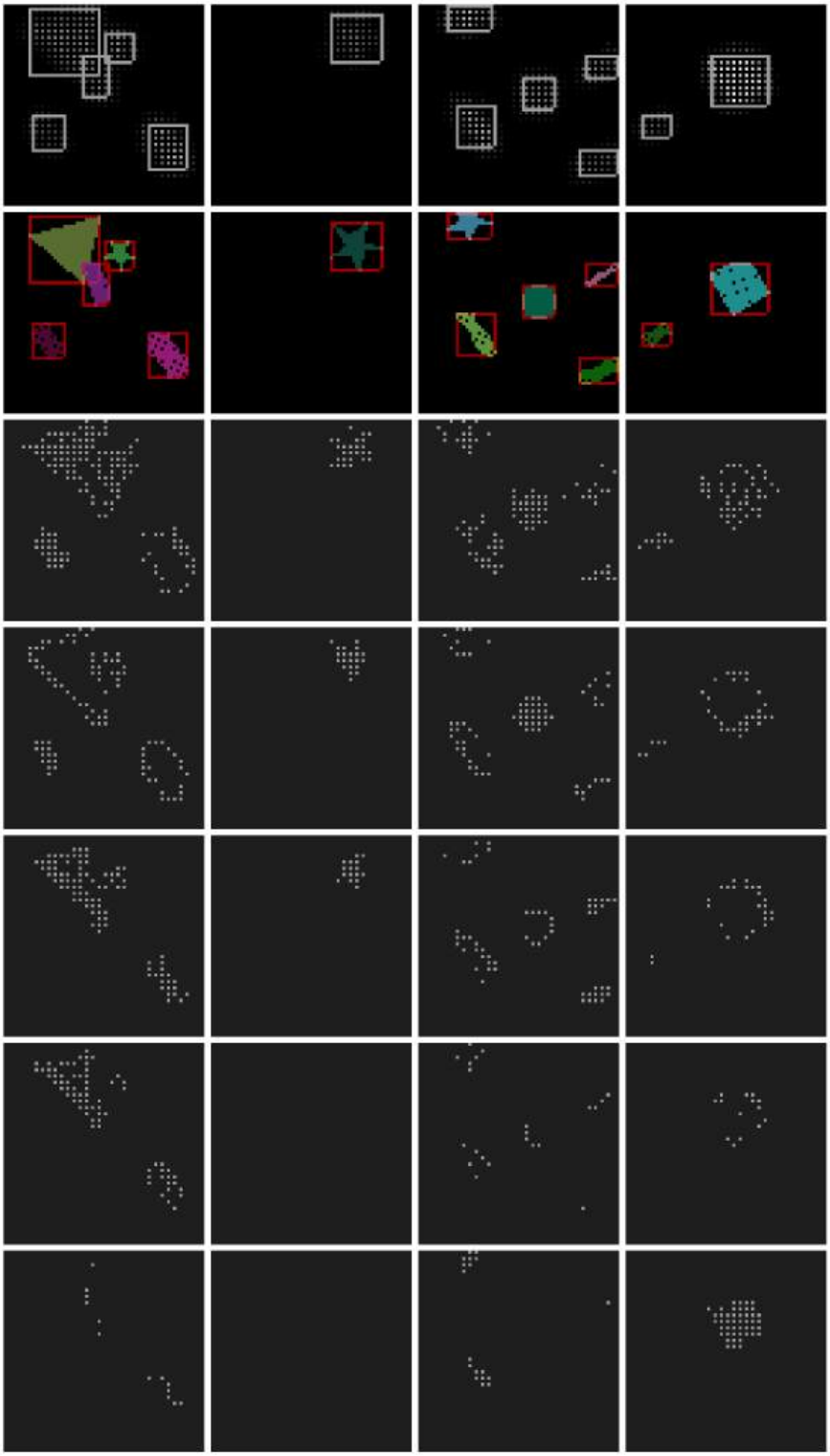
Batch 51



Batch 101

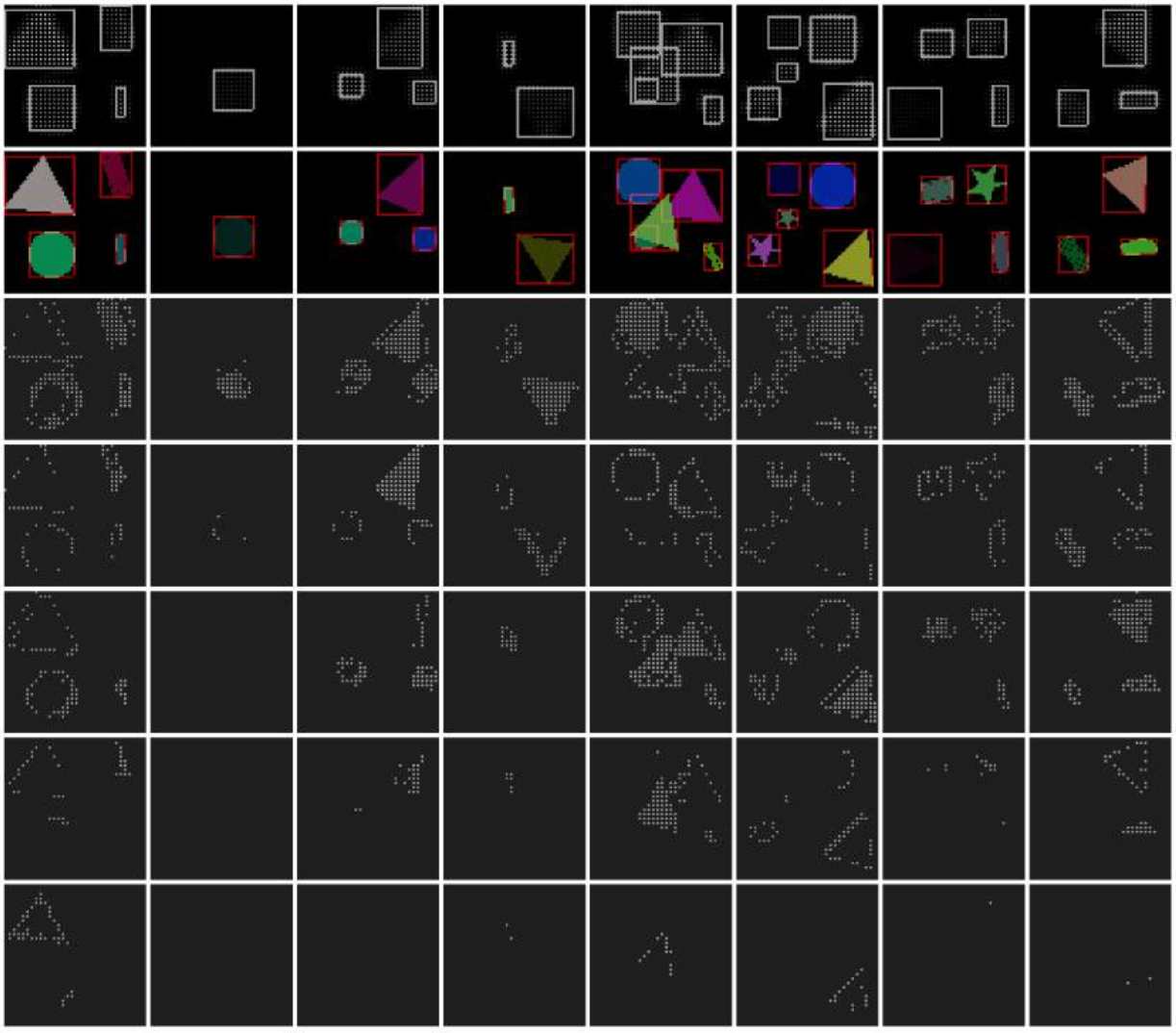


Batch 151

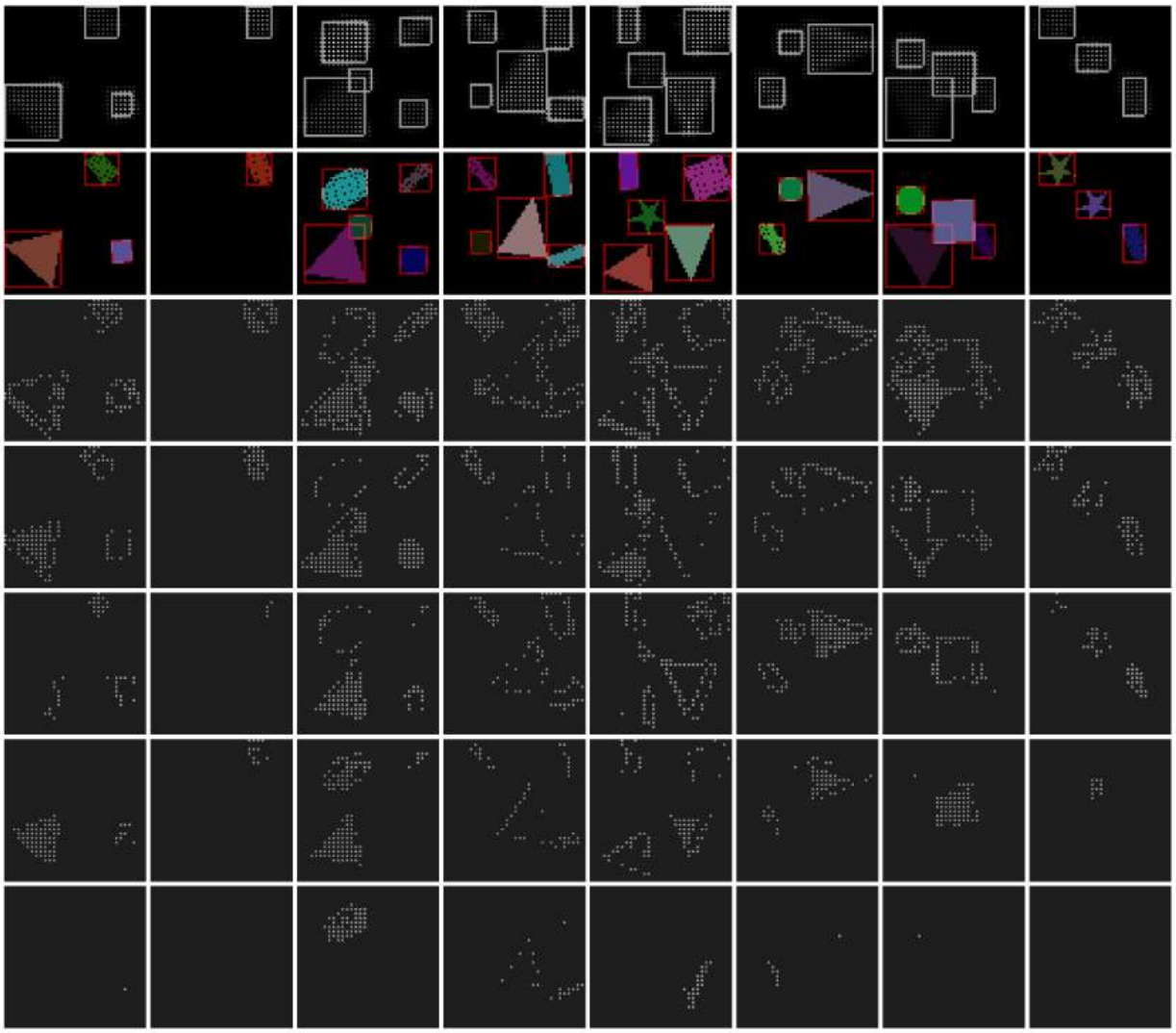


Batch 201

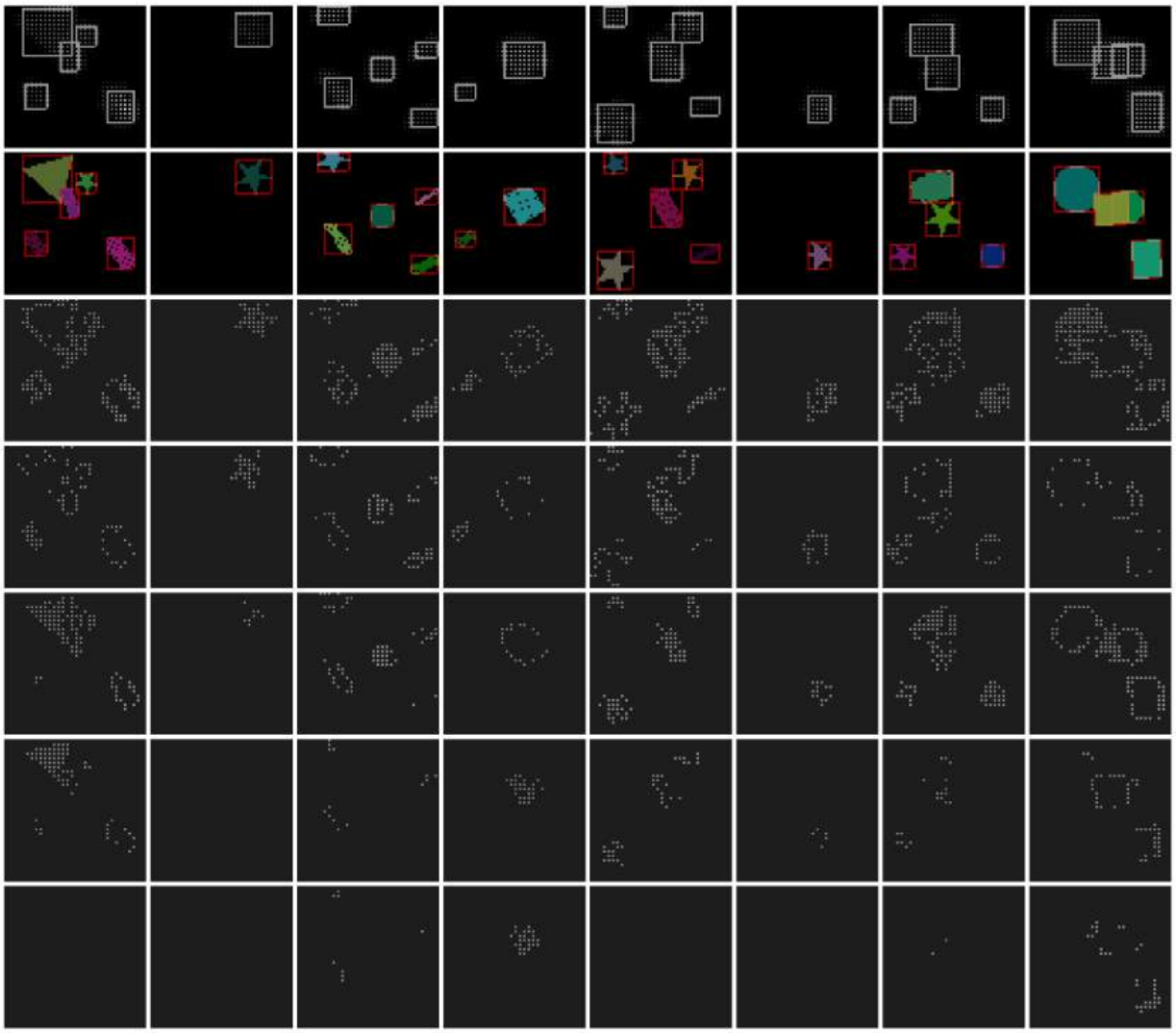
Output of learning rate 1e-5, epoch 12, batch 8



Batch 1



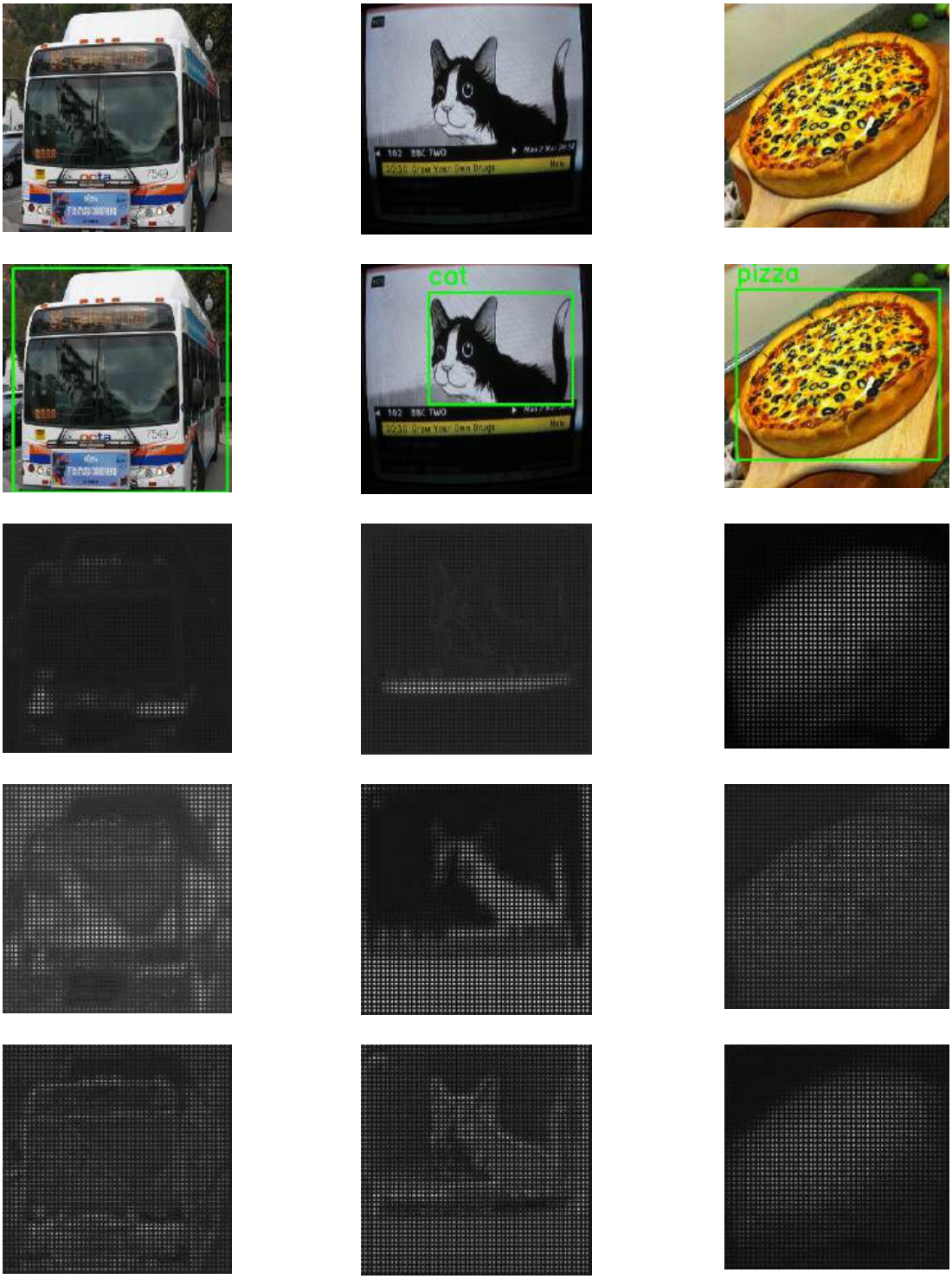
Batch 51



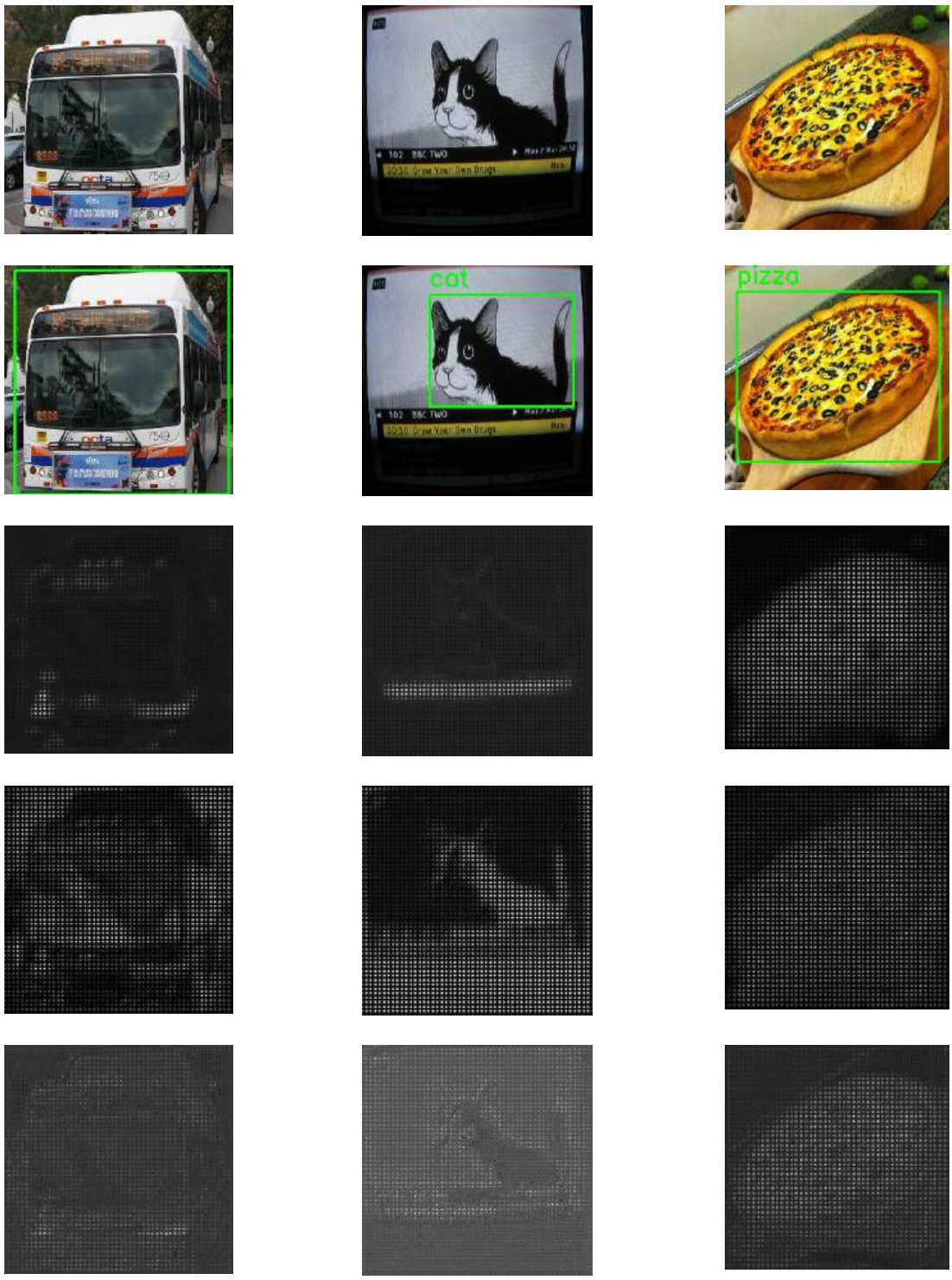
Batch 101

Test Results of MSCOCO

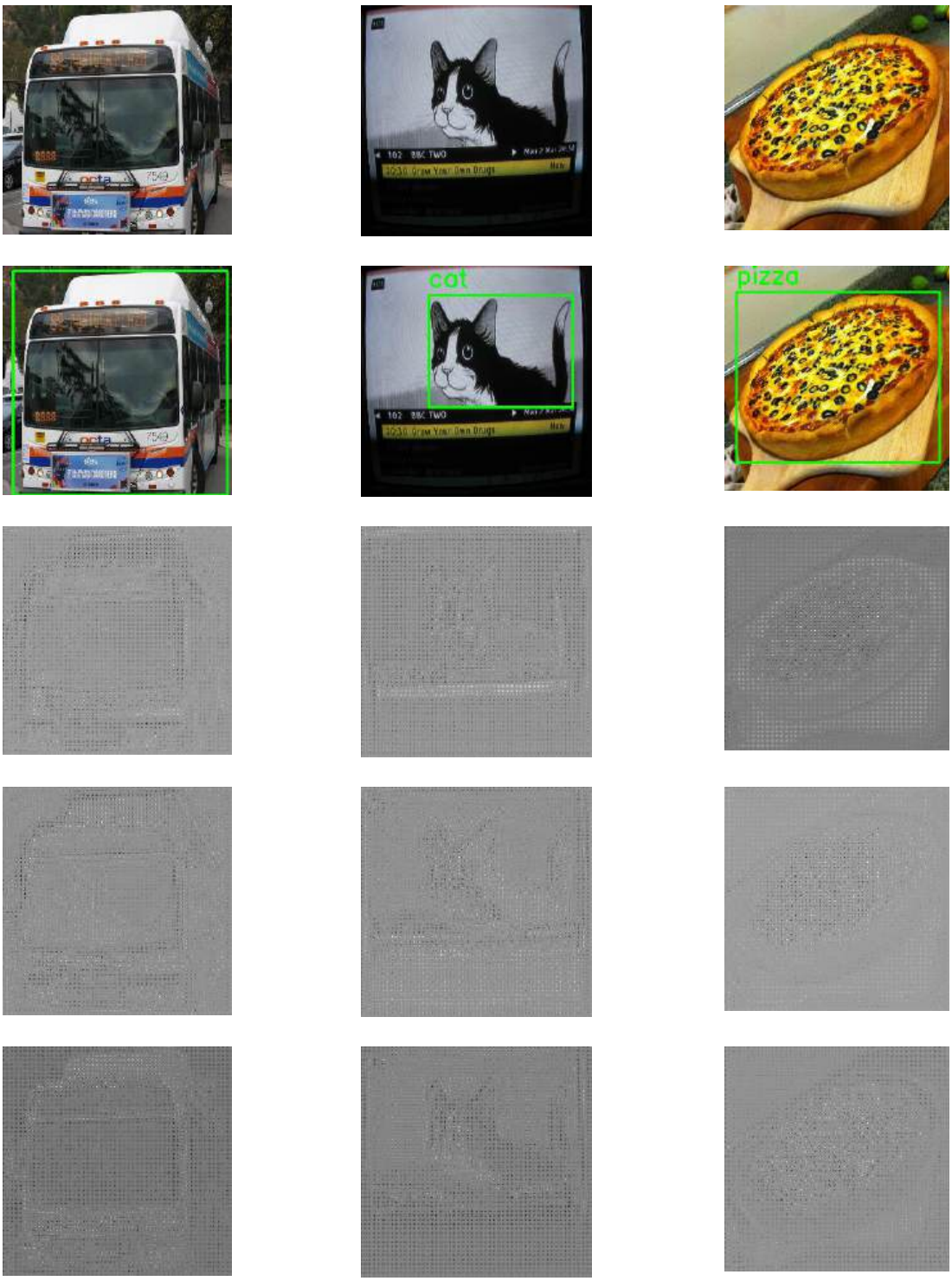
Output of learning rate 1e-4, epoch 20, batch 4



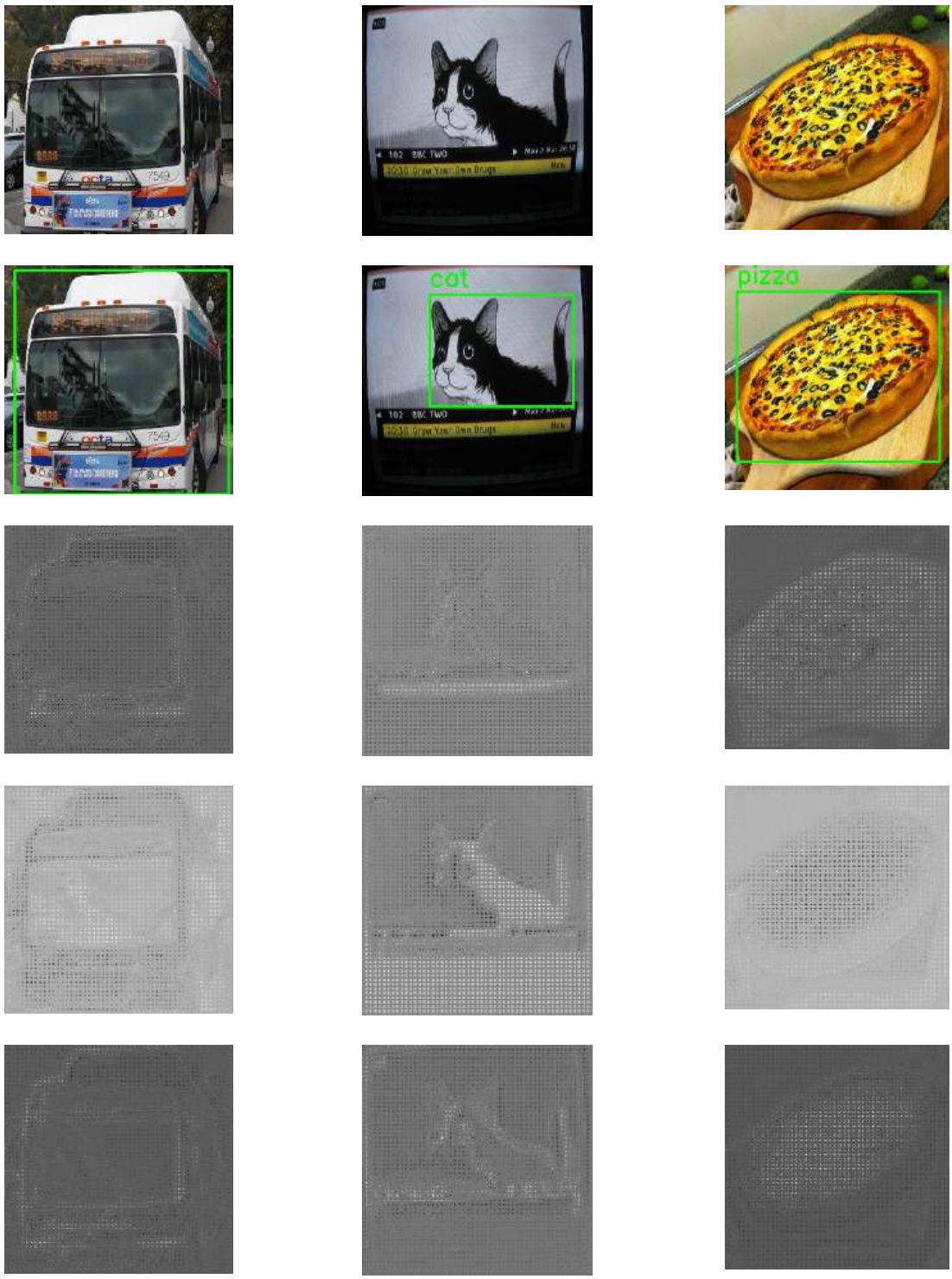
Output of learning rate 1e-4, epoch 30, batch 4



Output of learning rate 1e-5, epoch 20, batch 4



Output of learning rate 1e-5, epoch 30, batch 4



Disclaimer

When I was writing a report, the initial report exceeded 145 pages, and the software crashed twice. I did not want to face any further crashes, therefore split the report in two parts. The MSE+Dice and the rest of the report was written in a separate report which I intend to merge before submitting. You may find page discrepancy in the report for this.

When using Dice+MSE loss, do you think there should be a scaling factor to scale the Dice Loss? Why or why not?

Yes, As the Dice is bounded by [0 to 1], and the MSE loss that I got was in the order of magnitude of 2, I think both the losses should be of equal magnitude for the model to be unbiased to any of them. So, either we have to scale the MSE down or scale the Dice loss up. I chose the second option and created the total loss to be MSE+ αDice. I varied α value to be 100 to have them in the same magnitude and also played with value of 1 to see the difference.

*Mixed loss (MSE+ 1*Dice) Analysis*

Training Loss of PurdueShapes5MultiObjectDataset

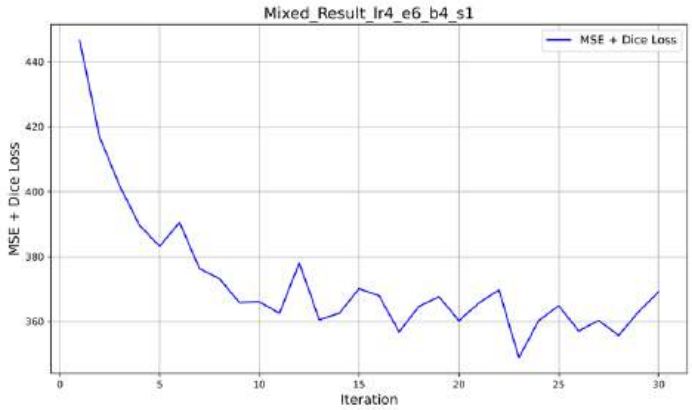


Fig: Loss curve for learning rate 1e-4, epoch 6, batch 4

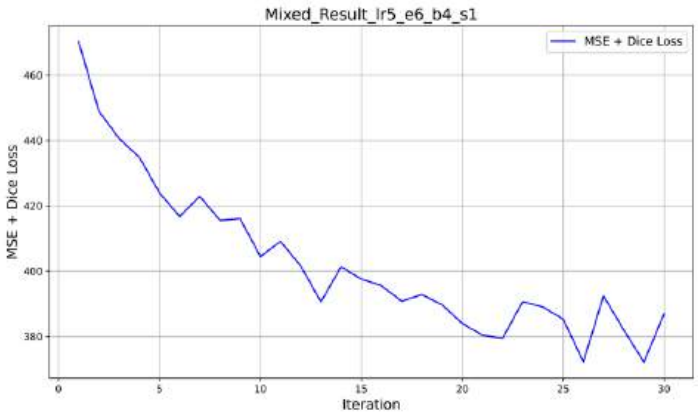


Fig: Loss curve for learning rate 1e-5, epoch 6, batch 4

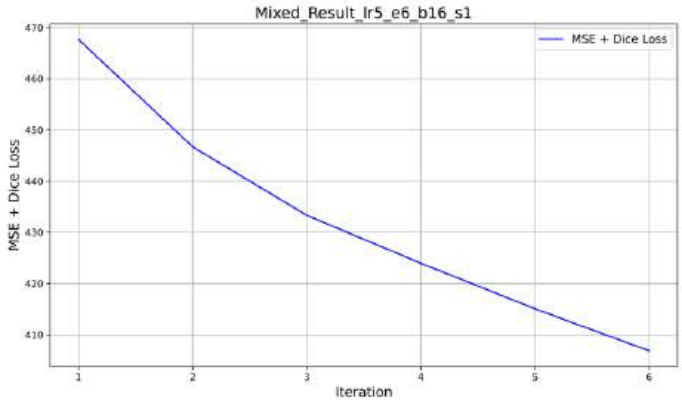


Fig: Loss curve for learning rate 1e-5, epoch 6, batch 16

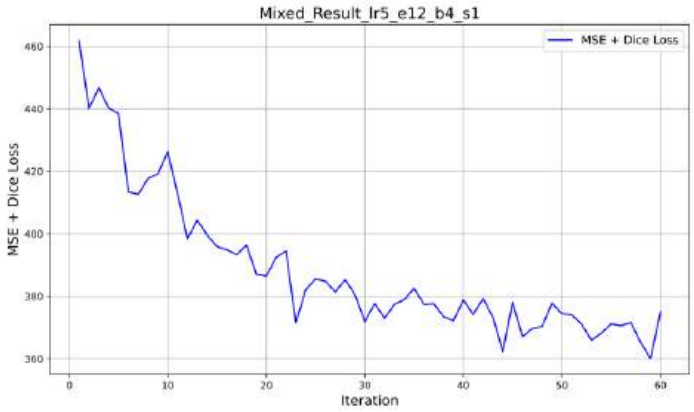


Fig: Loss curve for learning rate 1e-5, epoch 12, batch 4

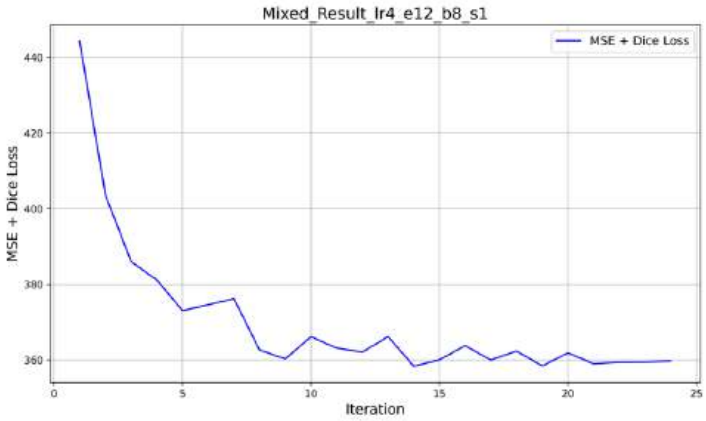


Fig: Loss curve for learning rate 1e-5, epoch 12, batch 8

Training Loss of MSCOCO

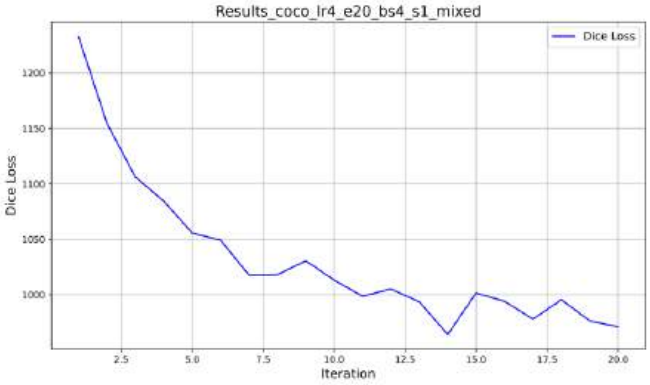


Fig: Loss curve for learning rate 1e-4, epoch 20, batch 4

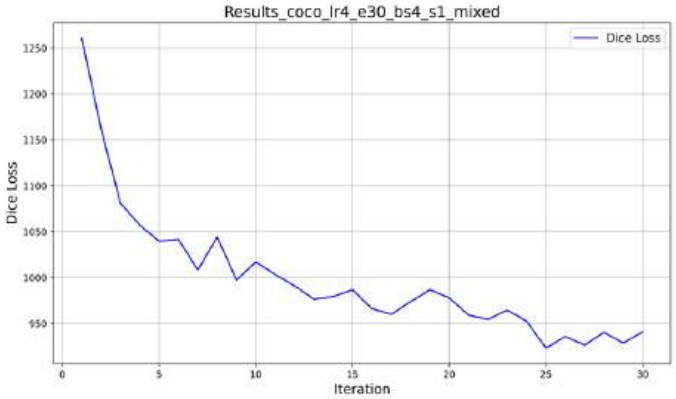


Fig: Loss curve for learning rate 1e-4, epoch 30, batch 4

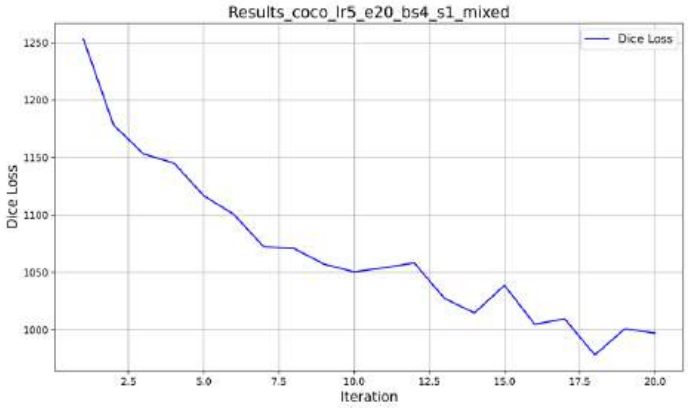


Fig: Loss curve for learning rate 1e-5, epoch 20, batch 4

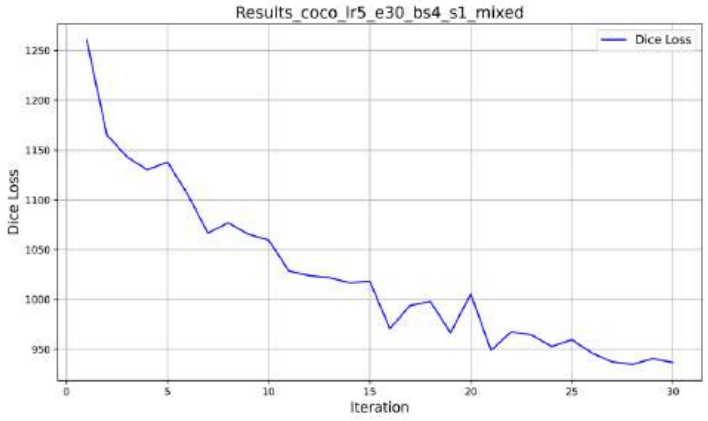
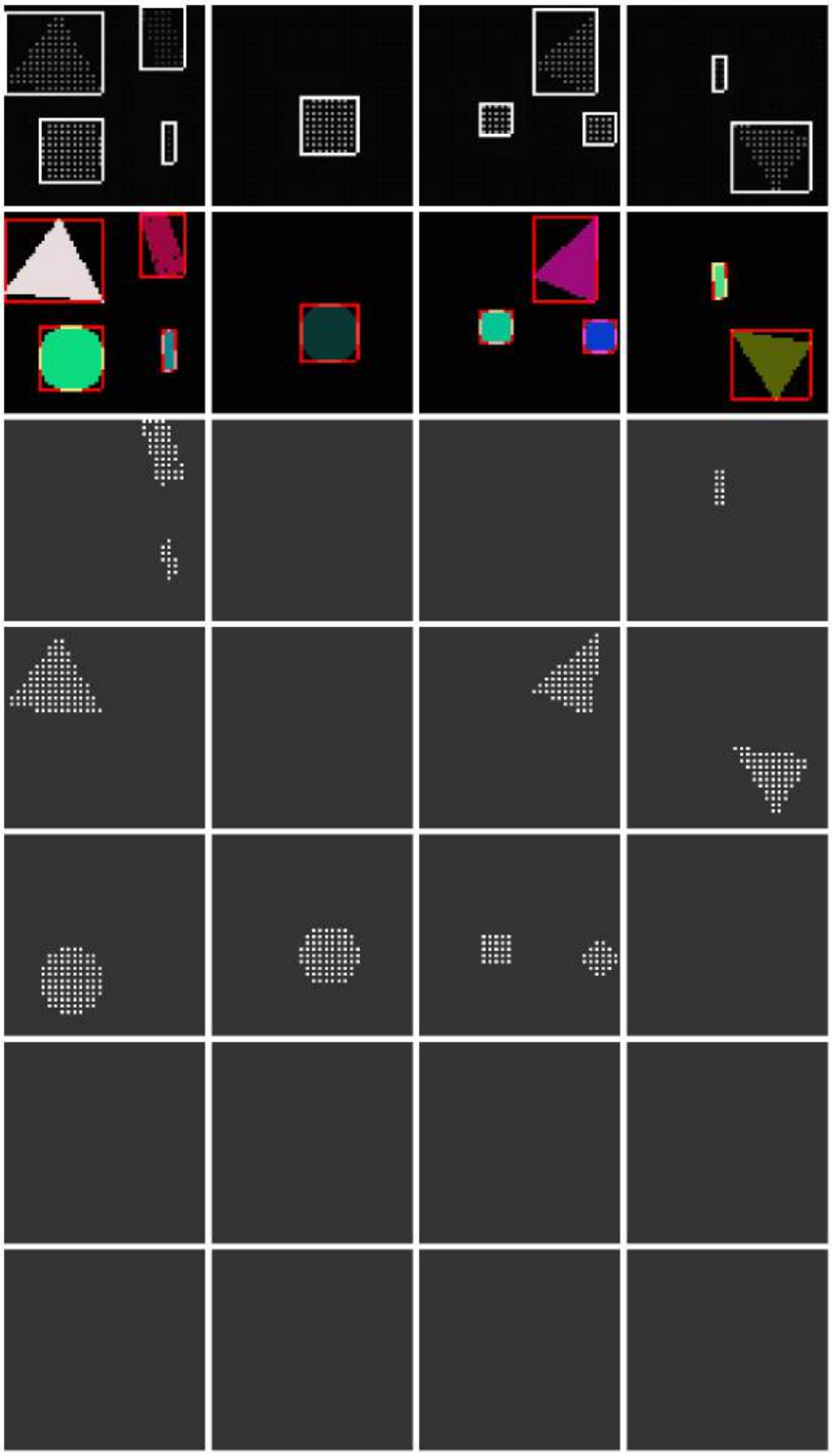


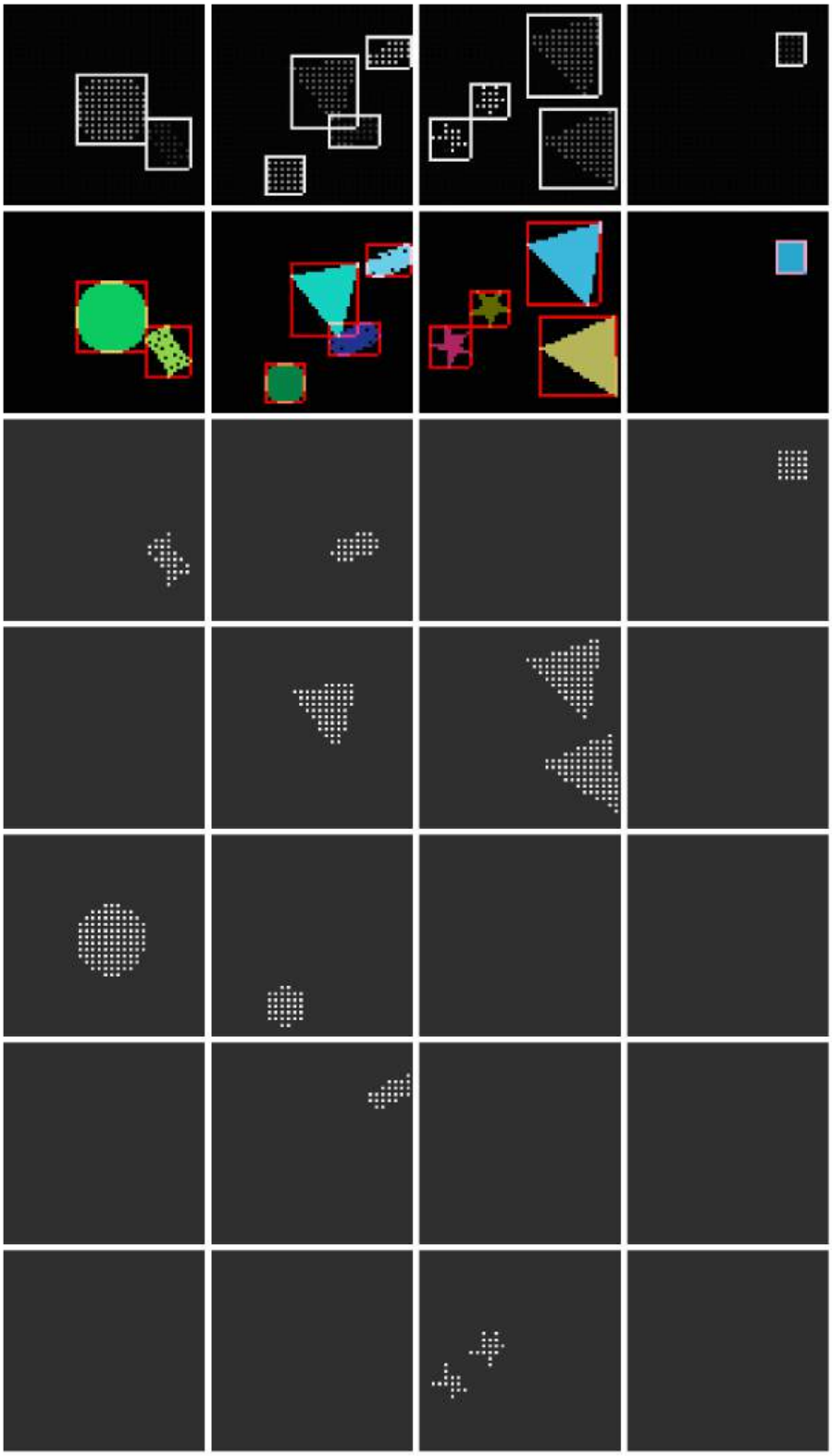
Fig: Loss curve for learning rate 1e-5, epoch 30, batch 4

Test Results of PurdueShapes5MultiObjectDataset

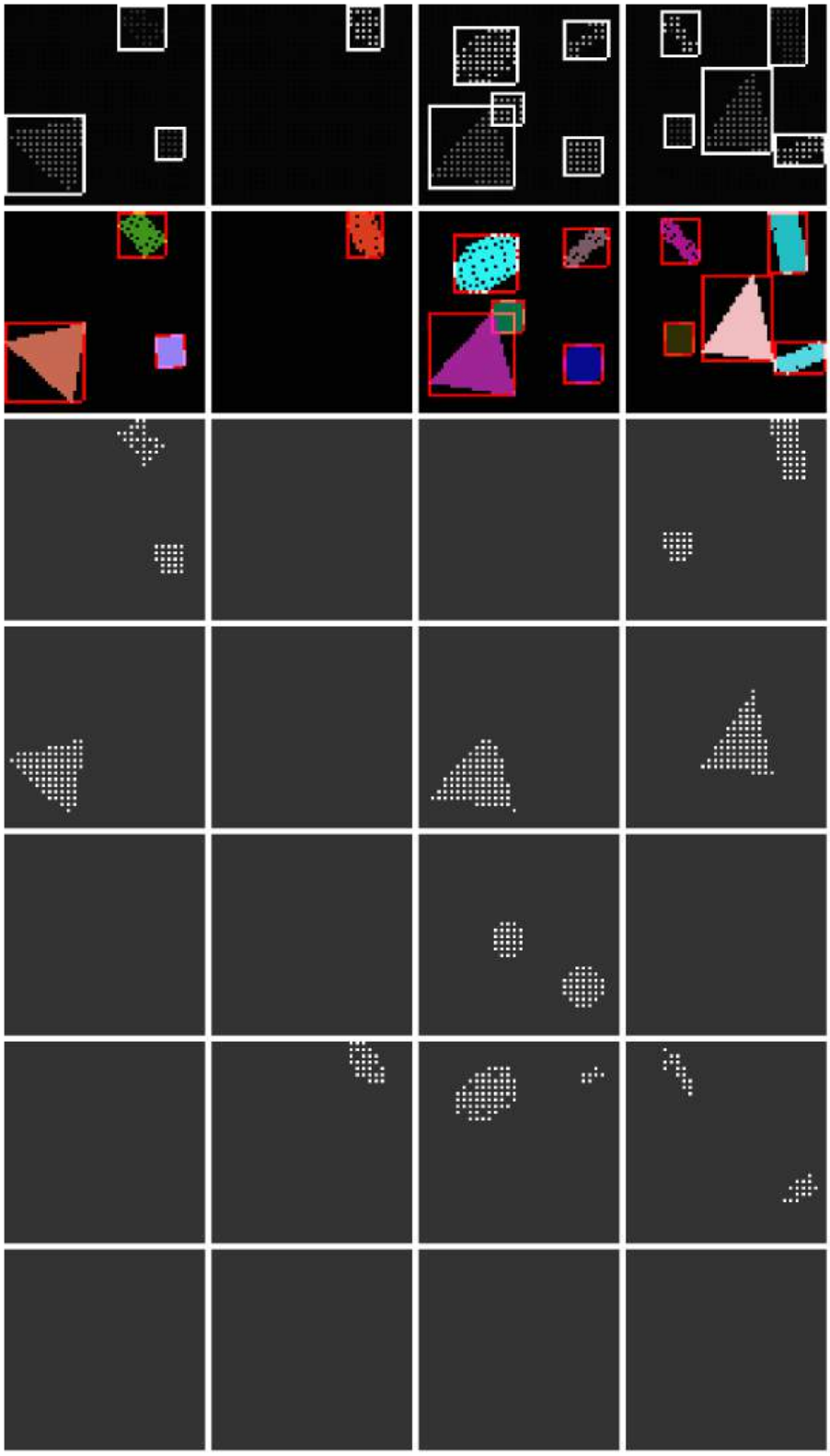
Output of learning rate 1e-4, epoch 6, batch 4



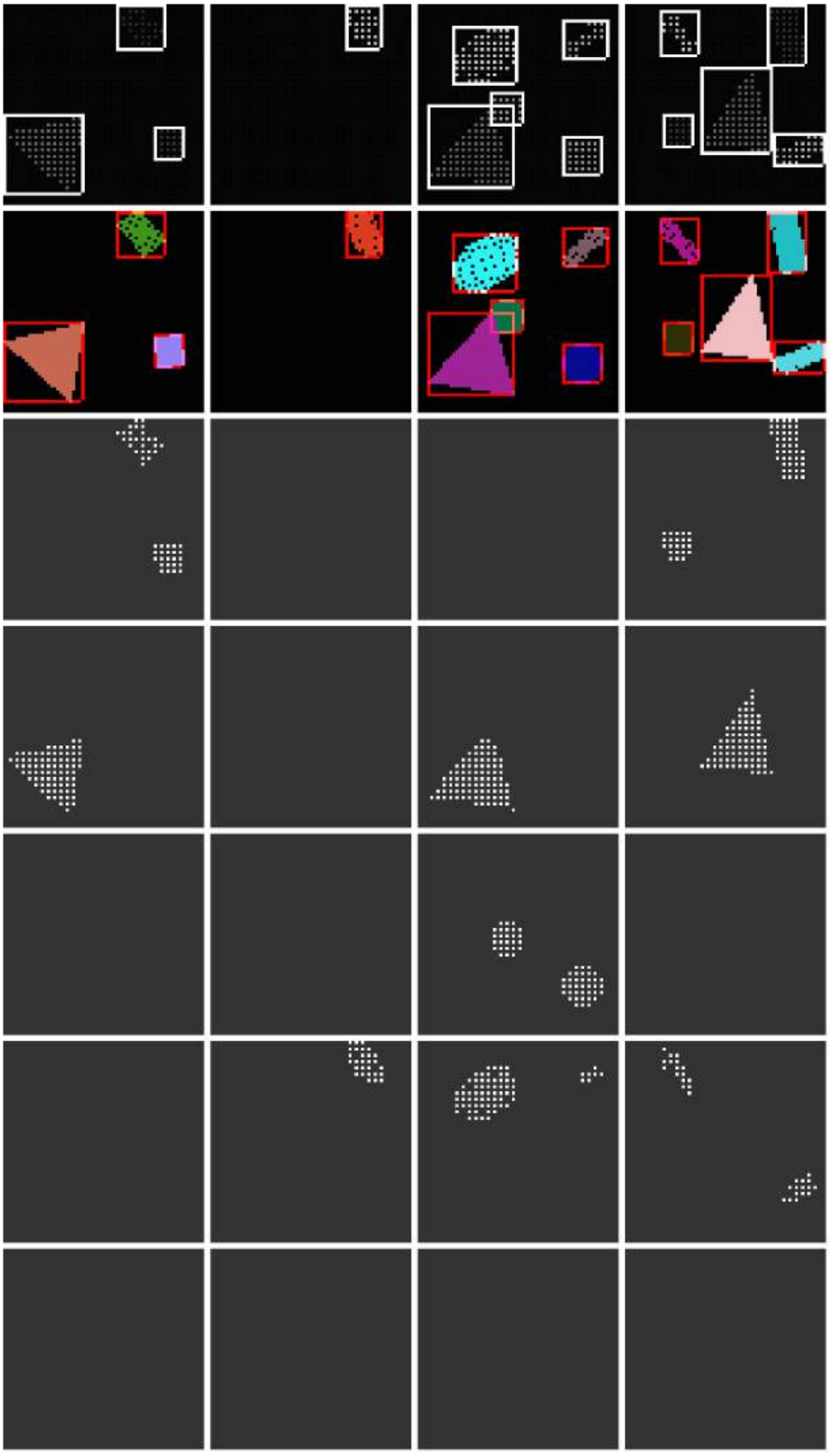
Batch 1



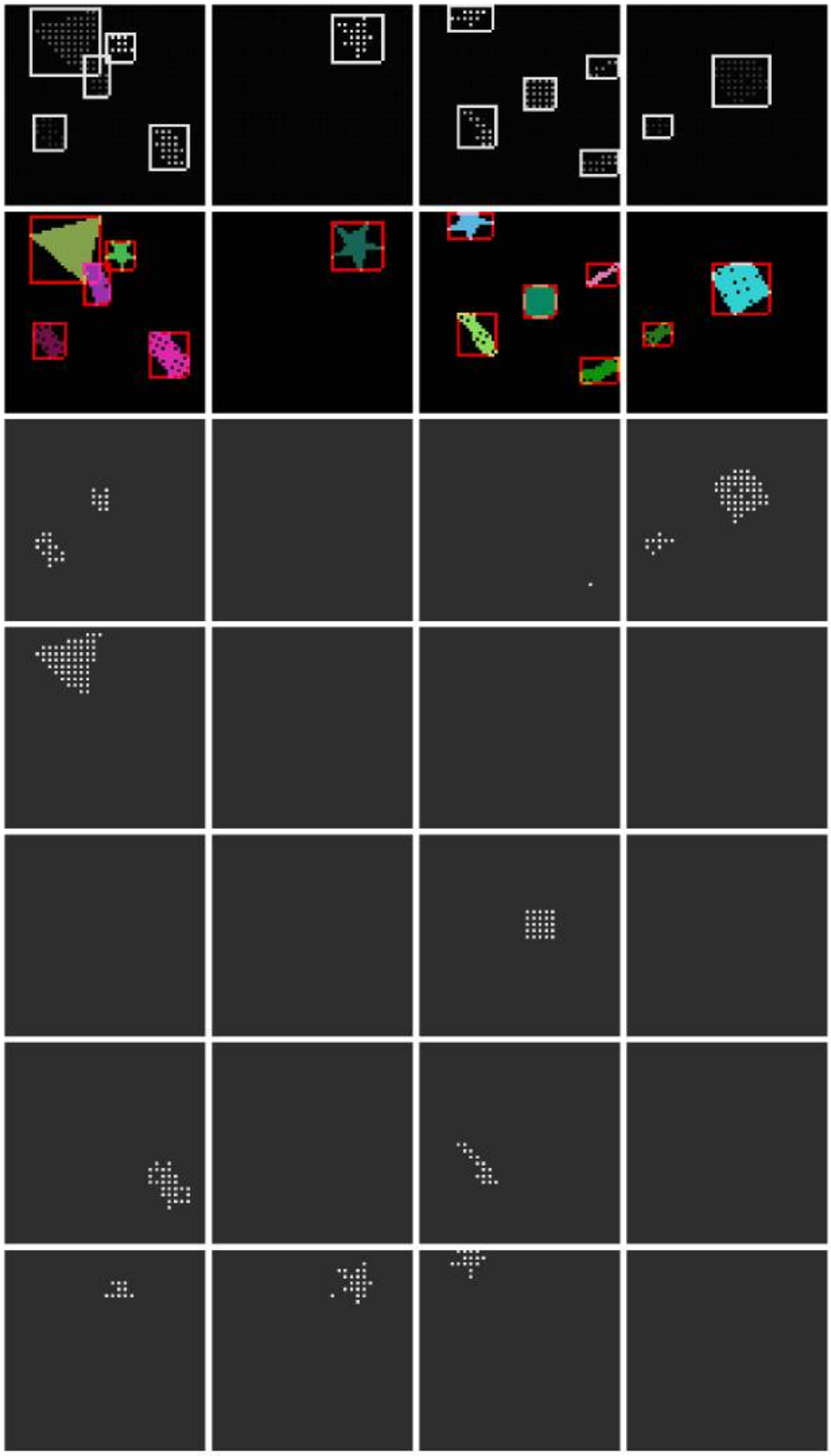
Batch 51



Batch 101

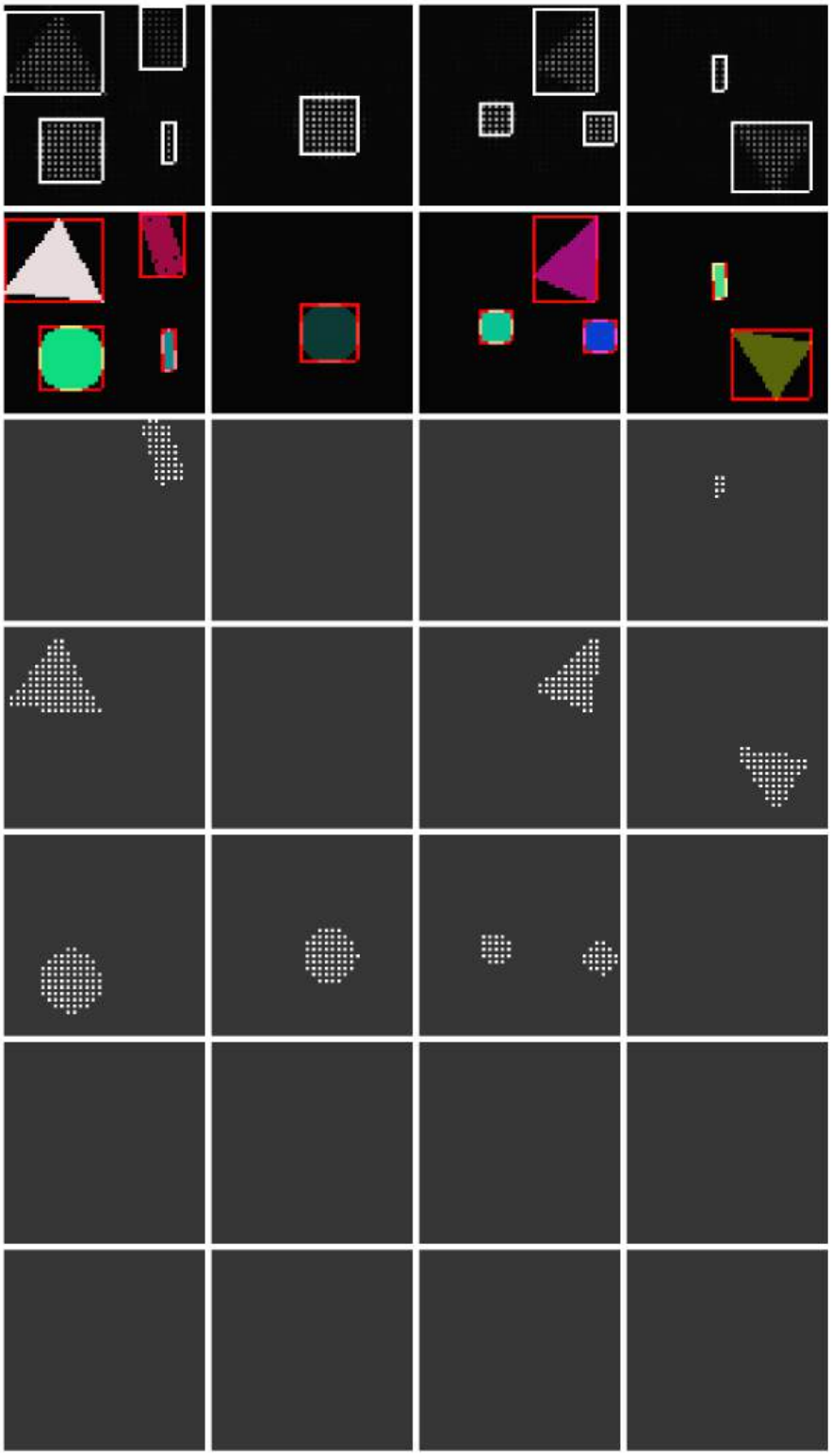


Batch 151

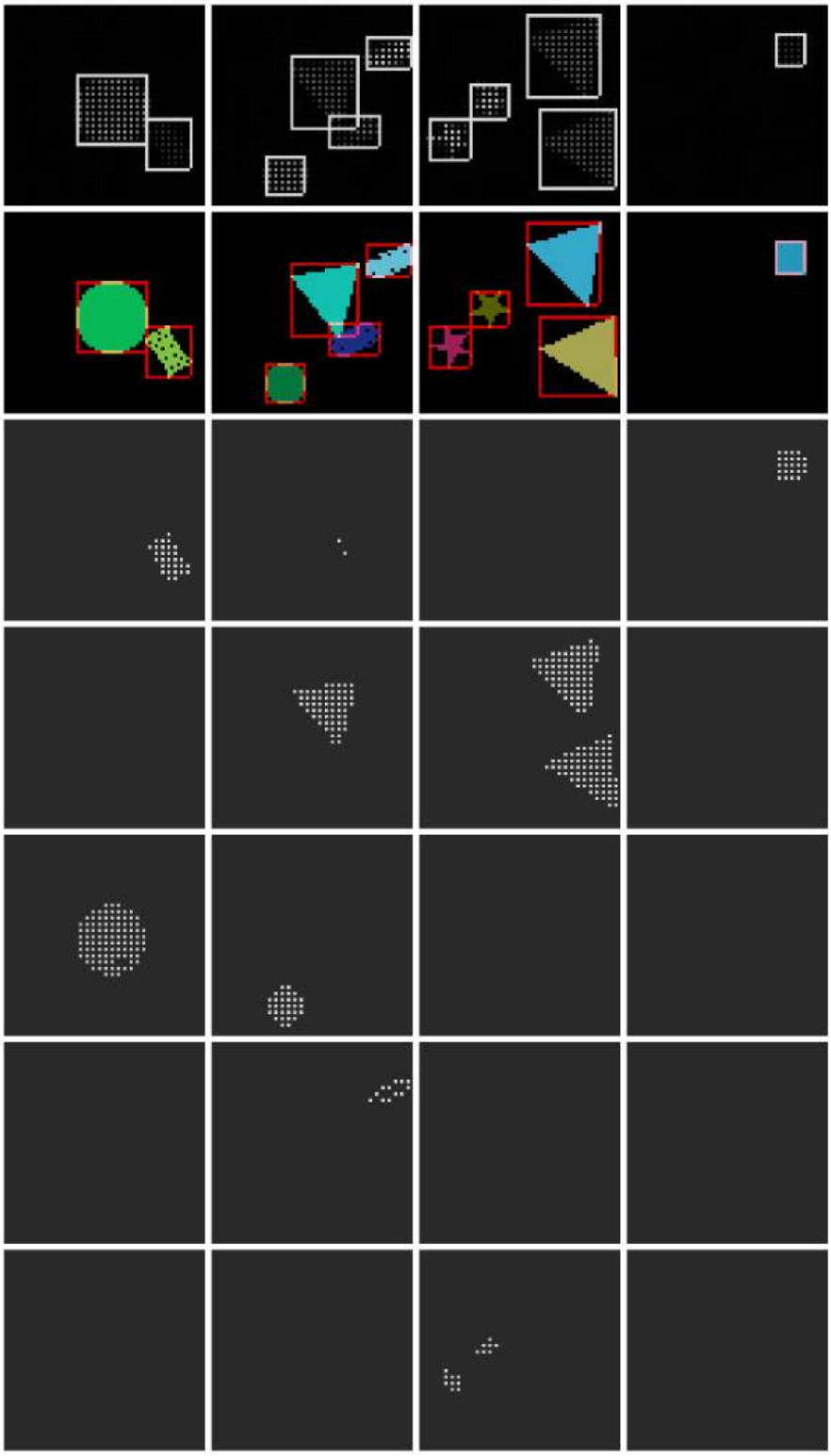


Batch 201

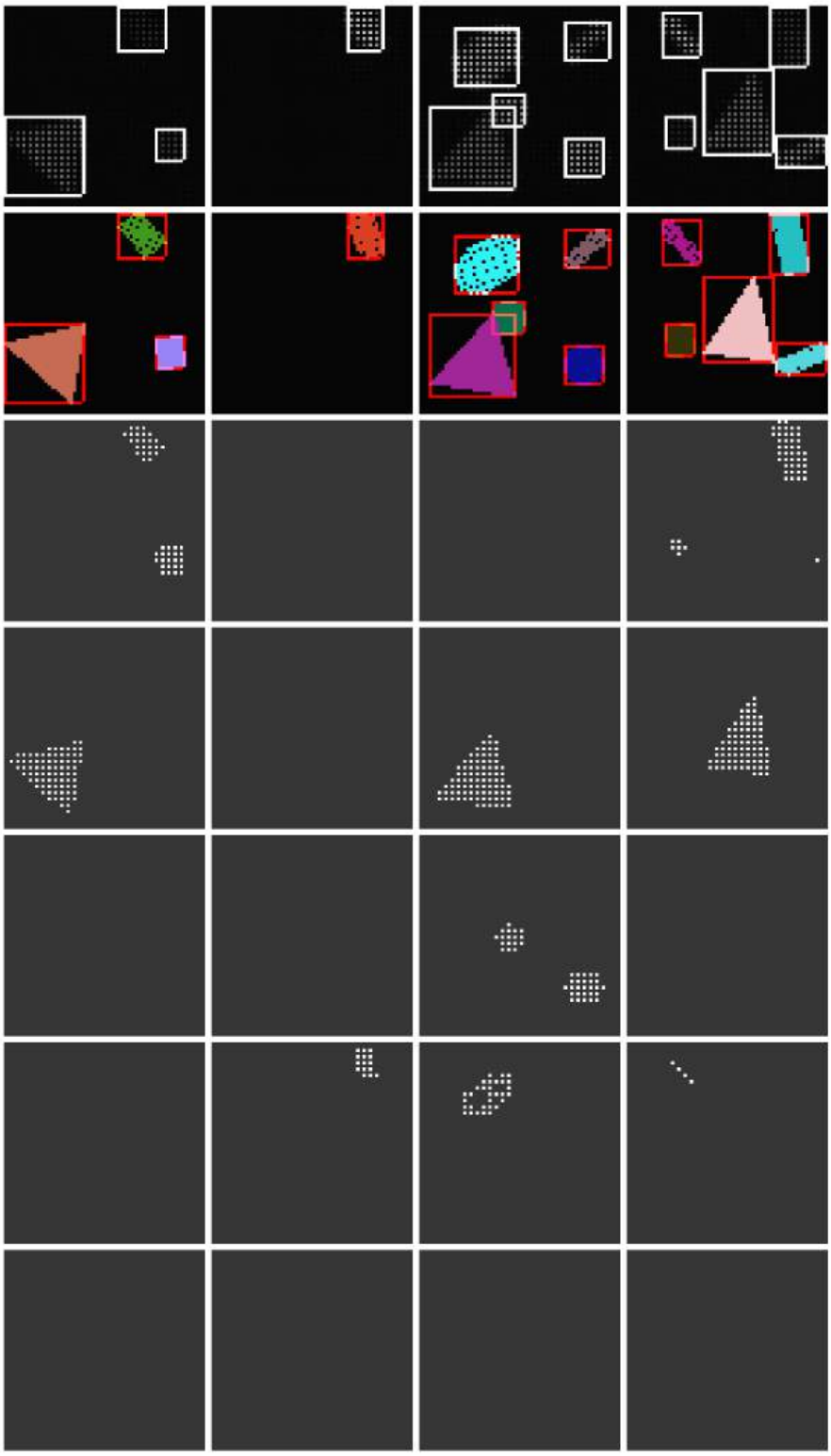
Output of learning rate 1e-5, epoch 6, batch 4



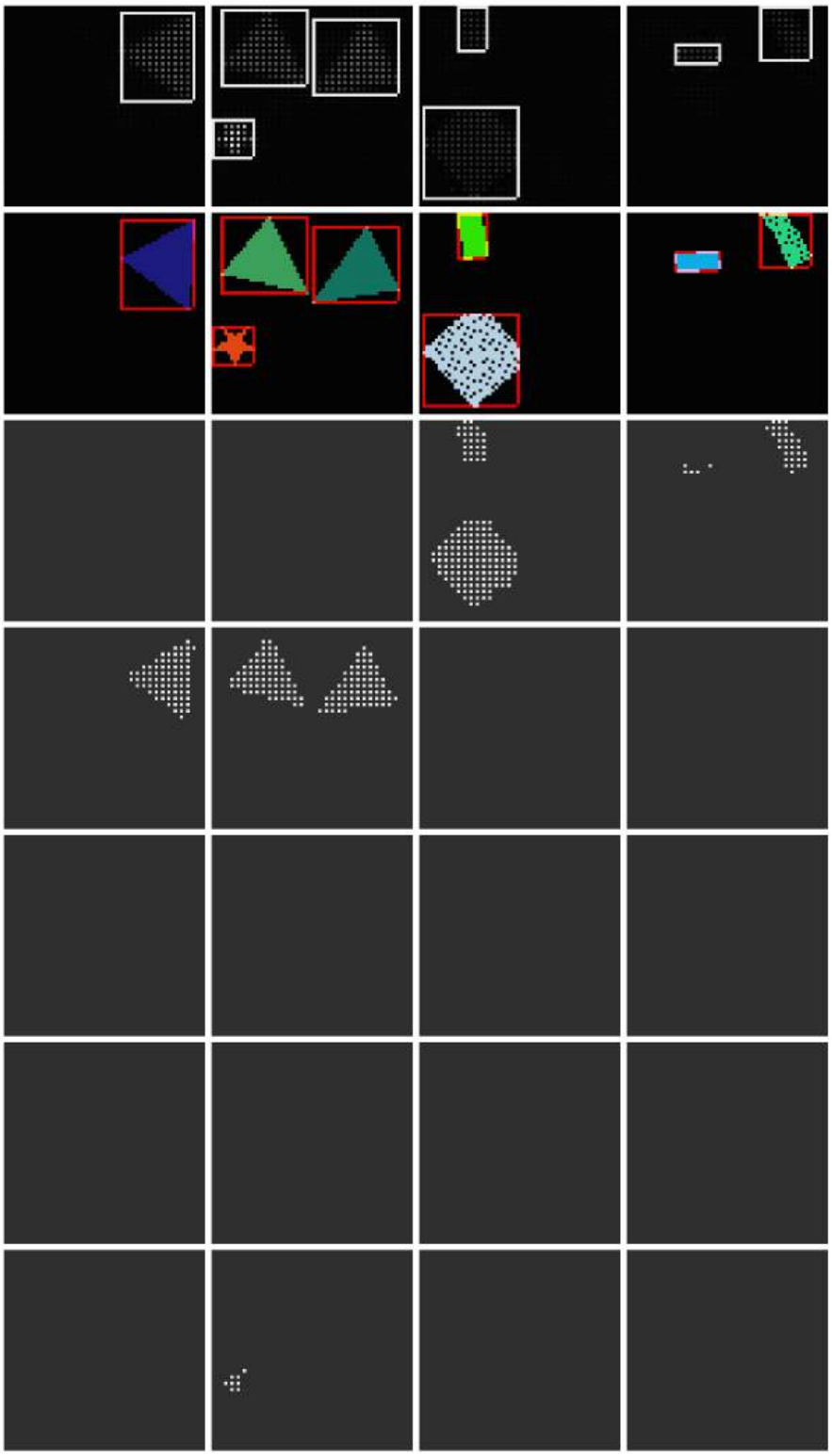
Batch 1



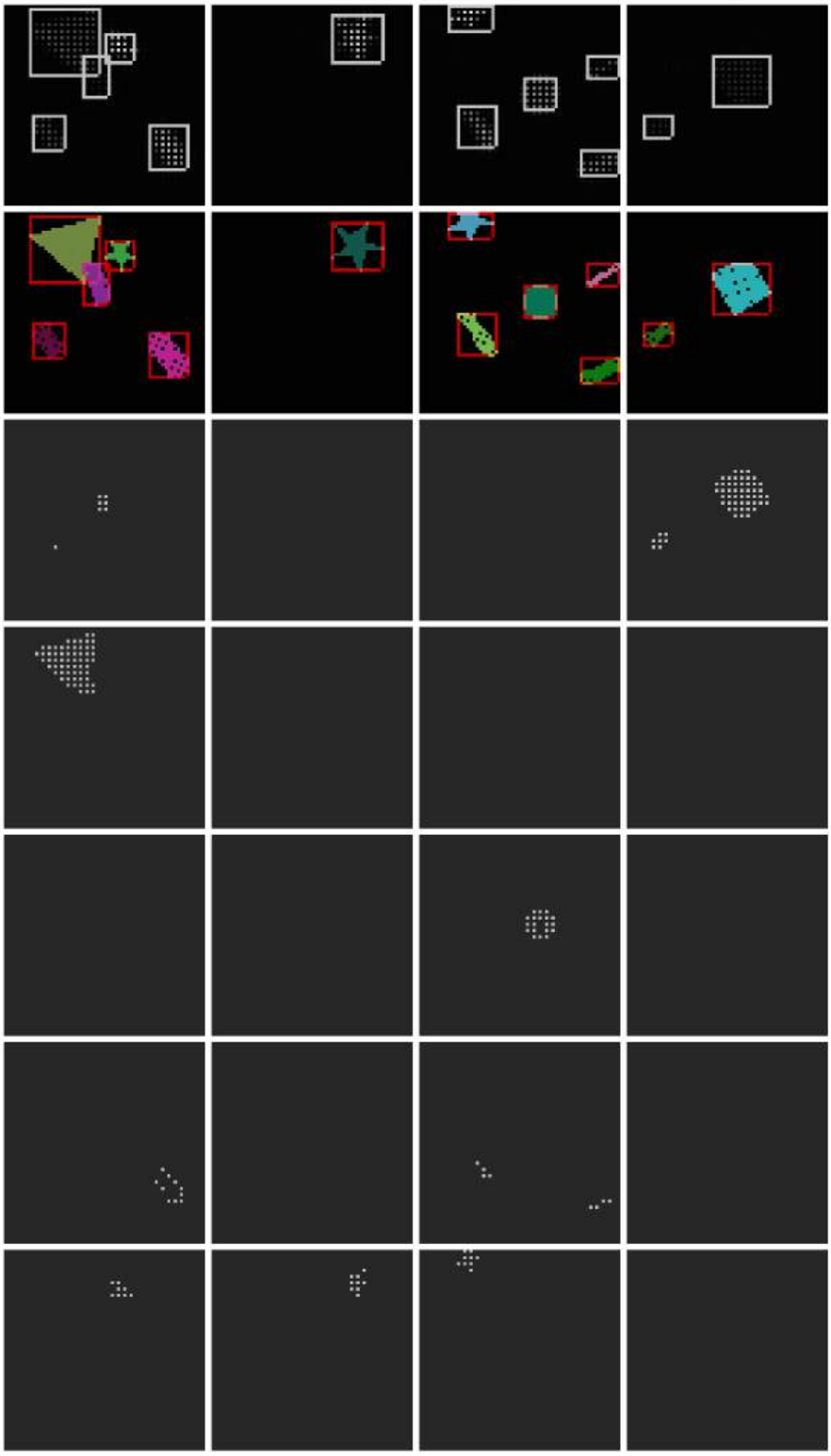
Batch 51



Batch 101

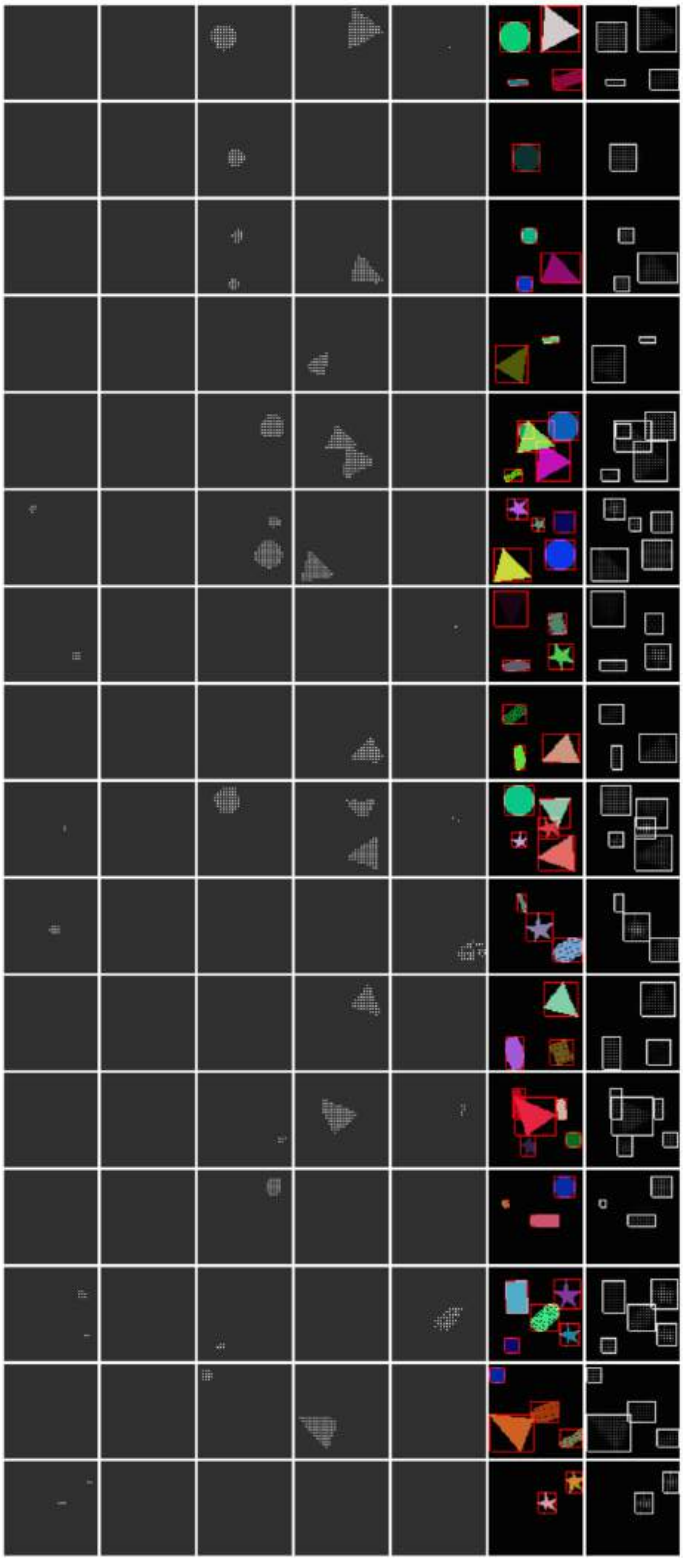


Batch 151

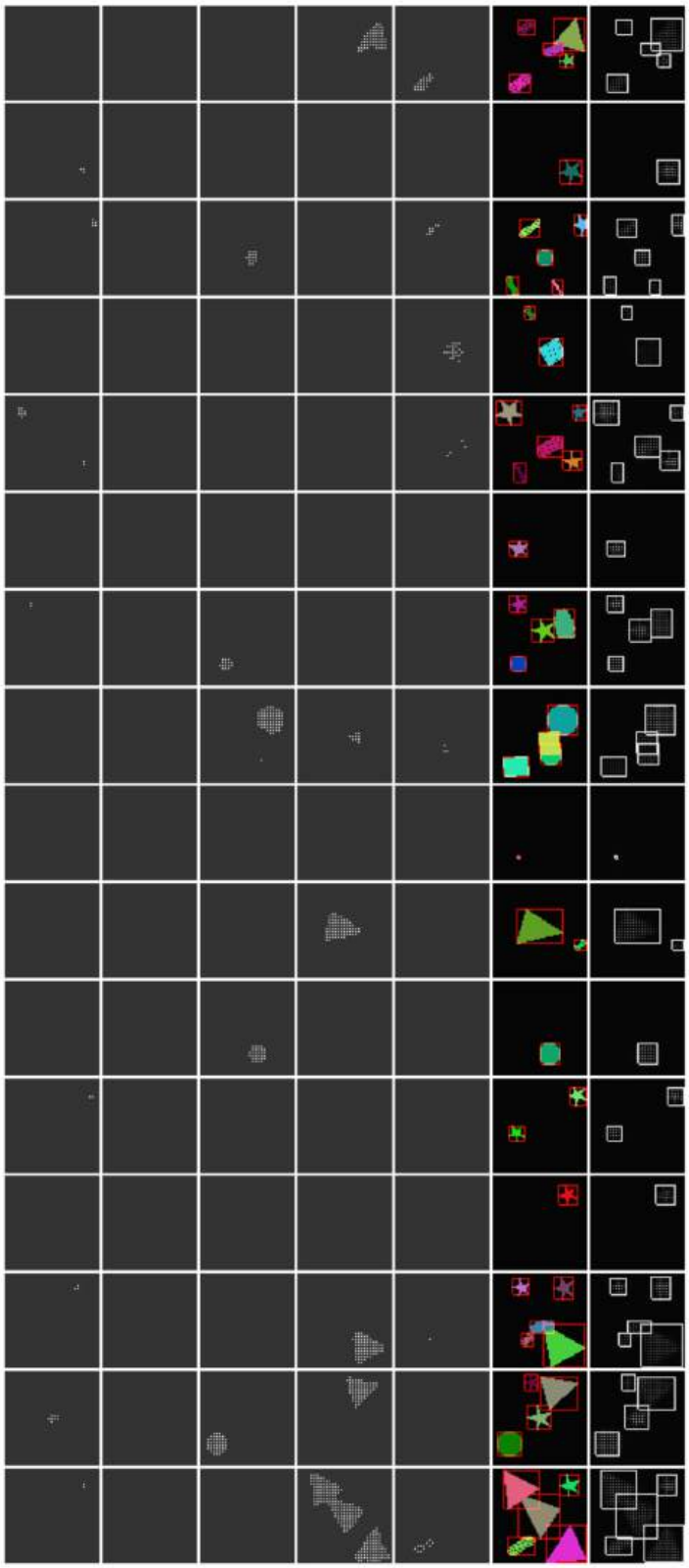


Batch 201

Output of learning rate 1e-5, epoch 6, batch 16

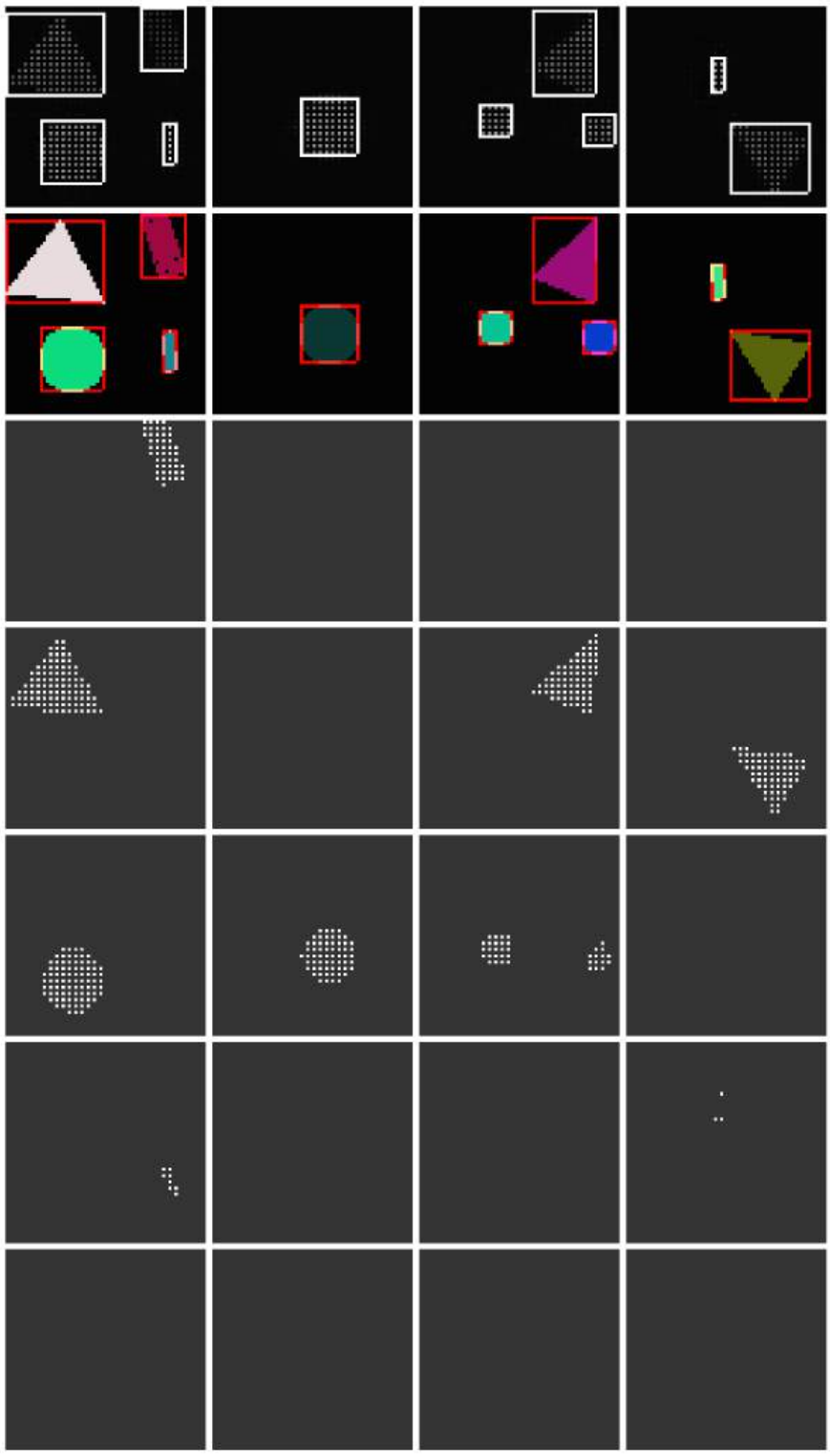


Batch 1

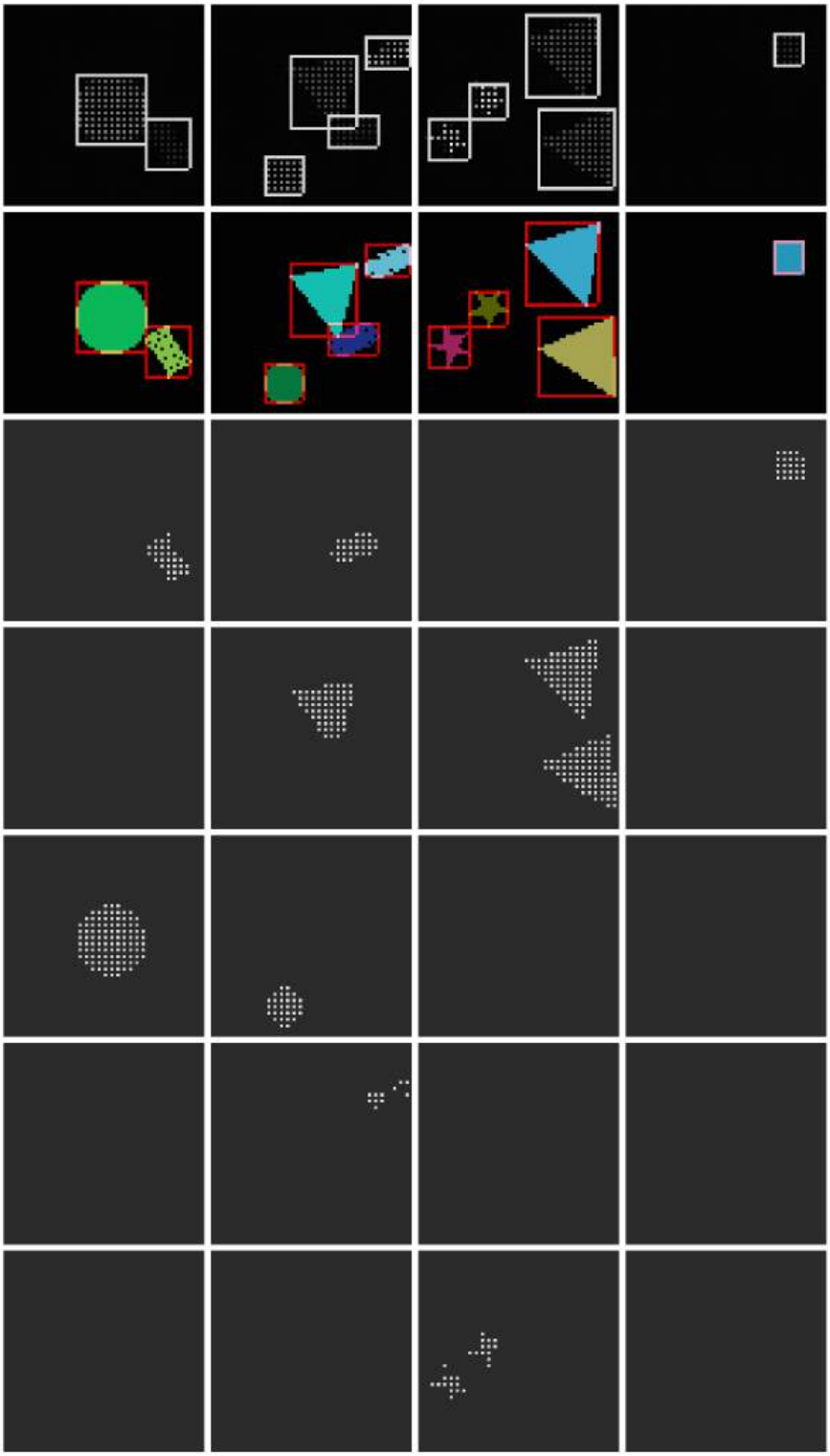


Batch 51

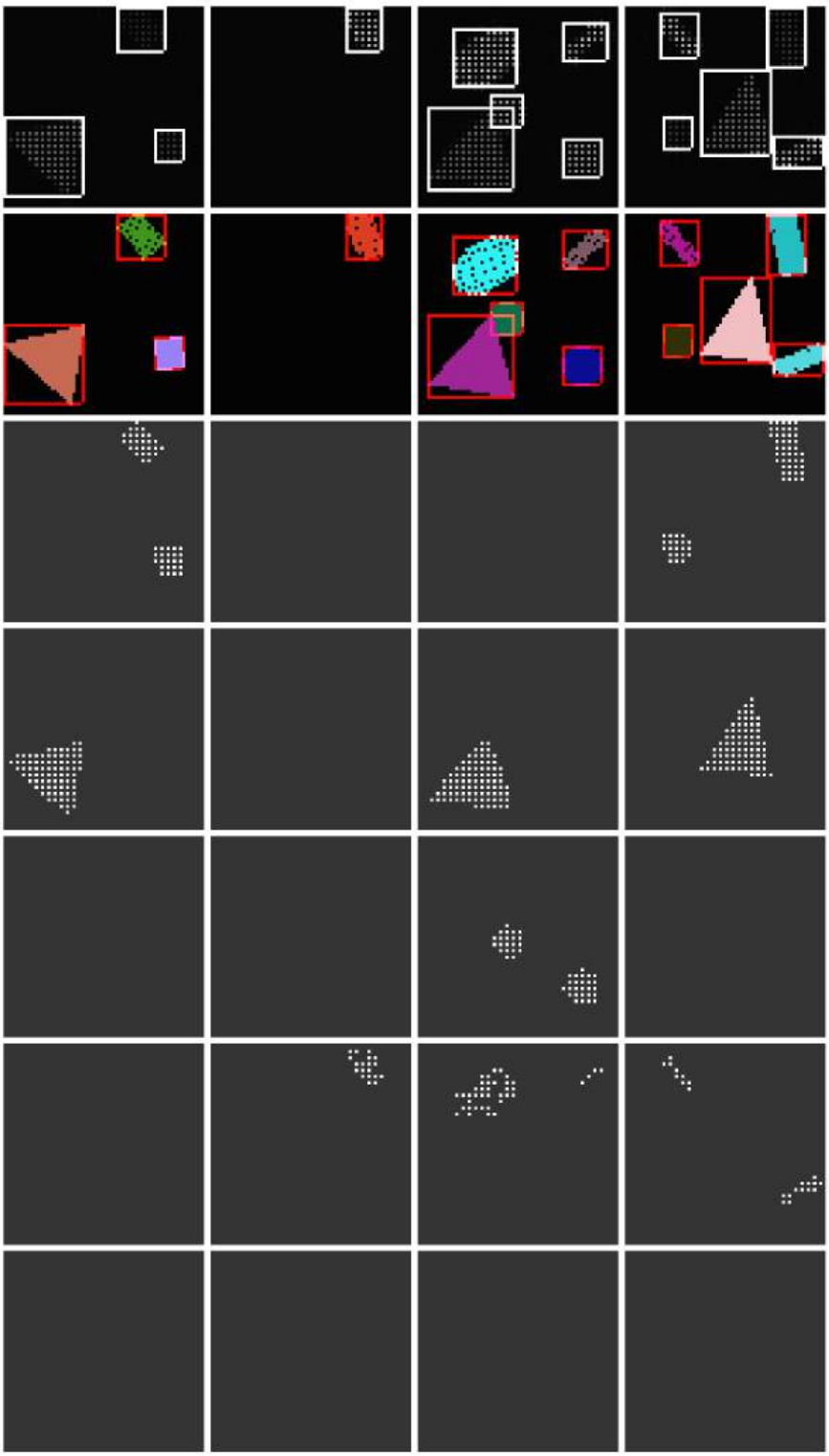
Output of learning rate 1e-5, epoch 12, batch 4



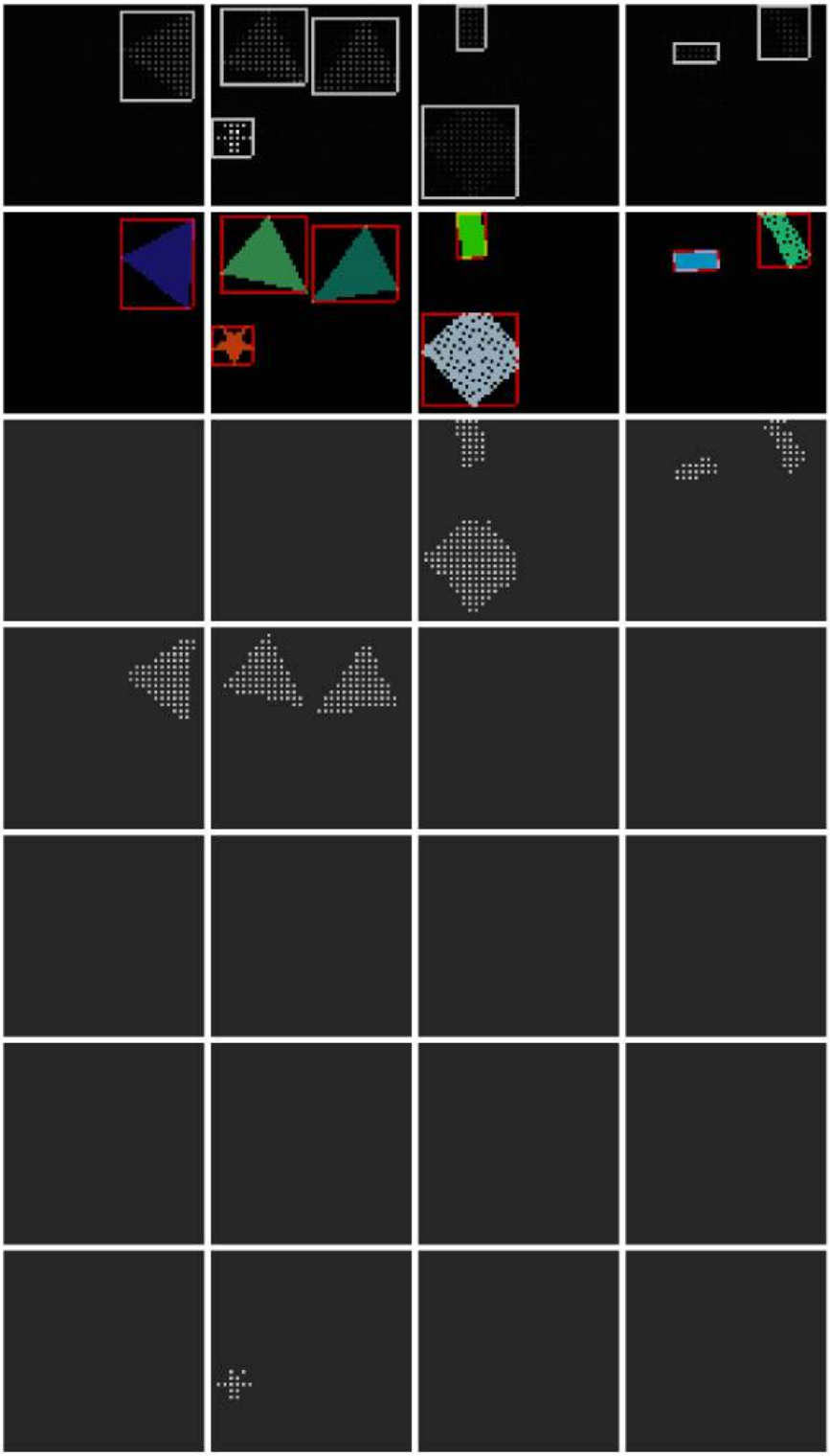
Batch 1



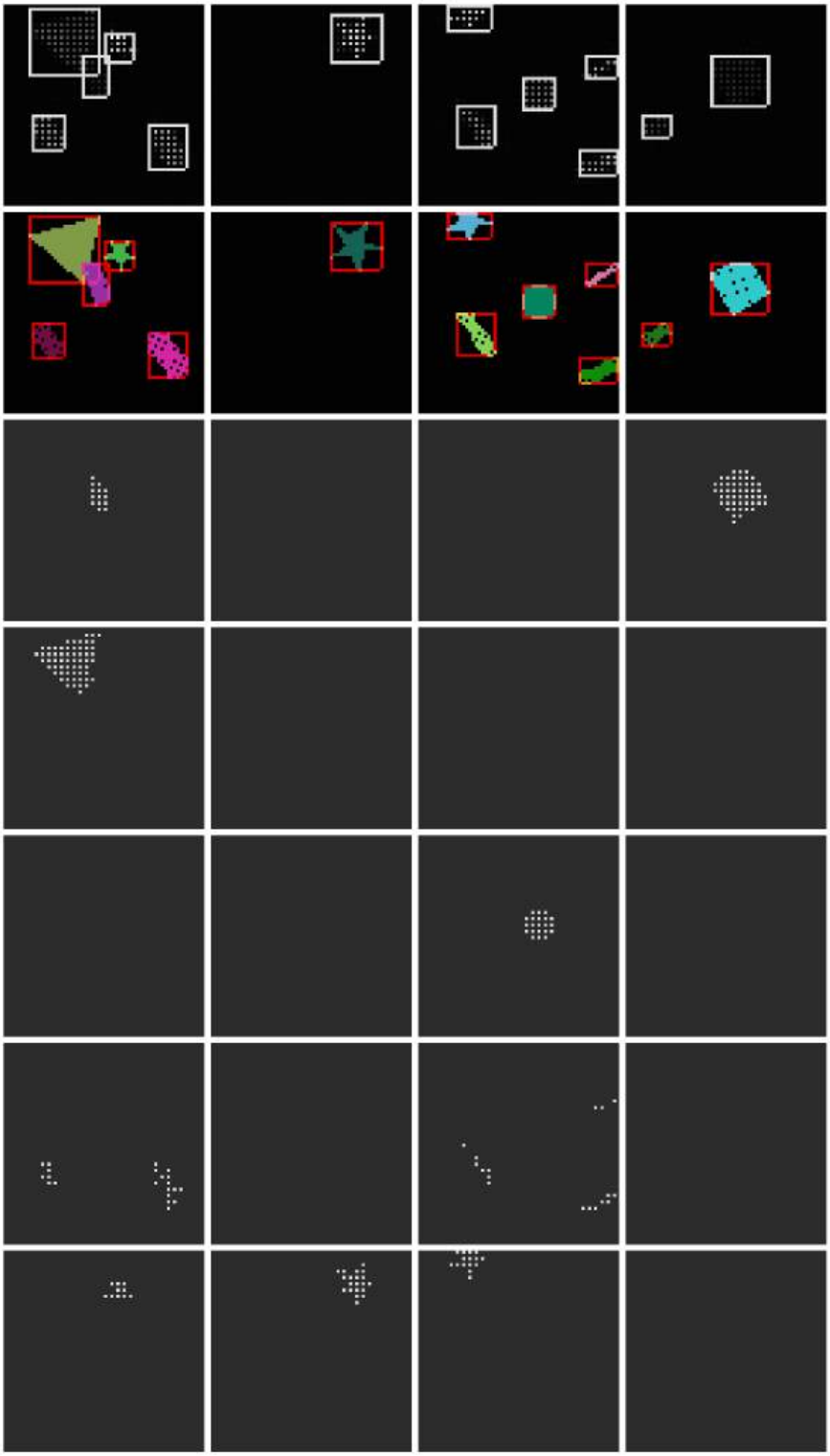
Batch 51



Batch 101

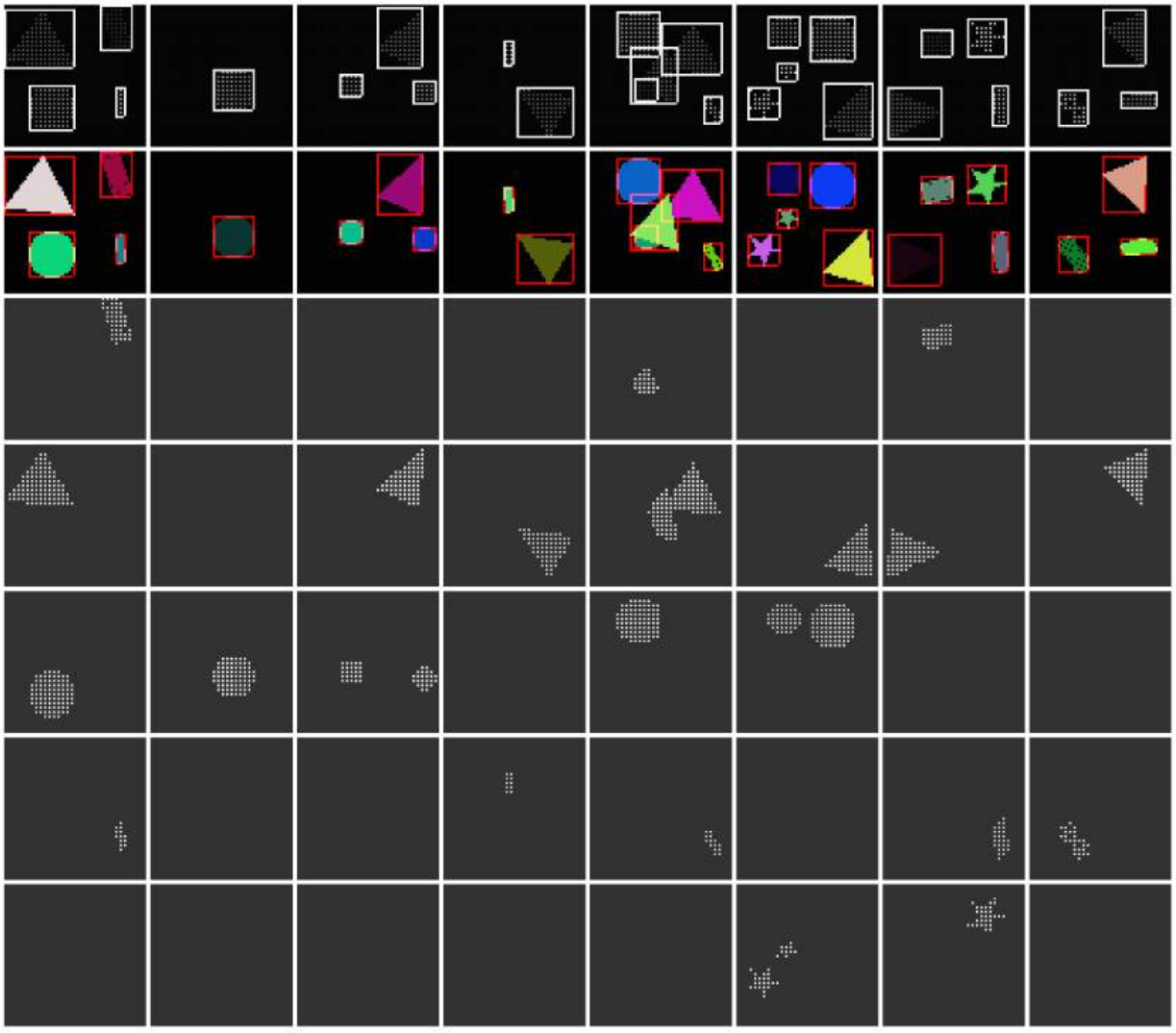


Batch 151

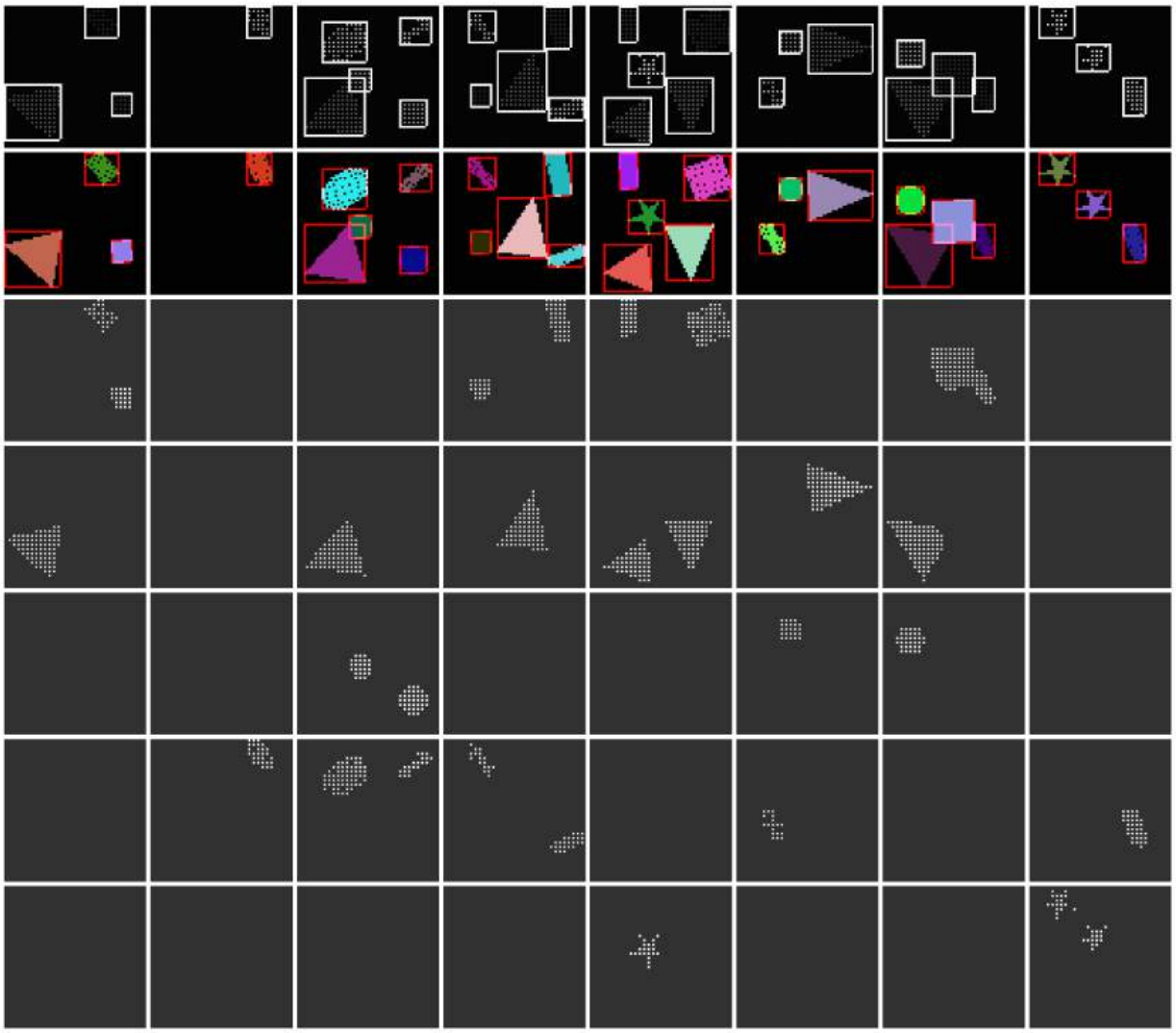


Batch 201

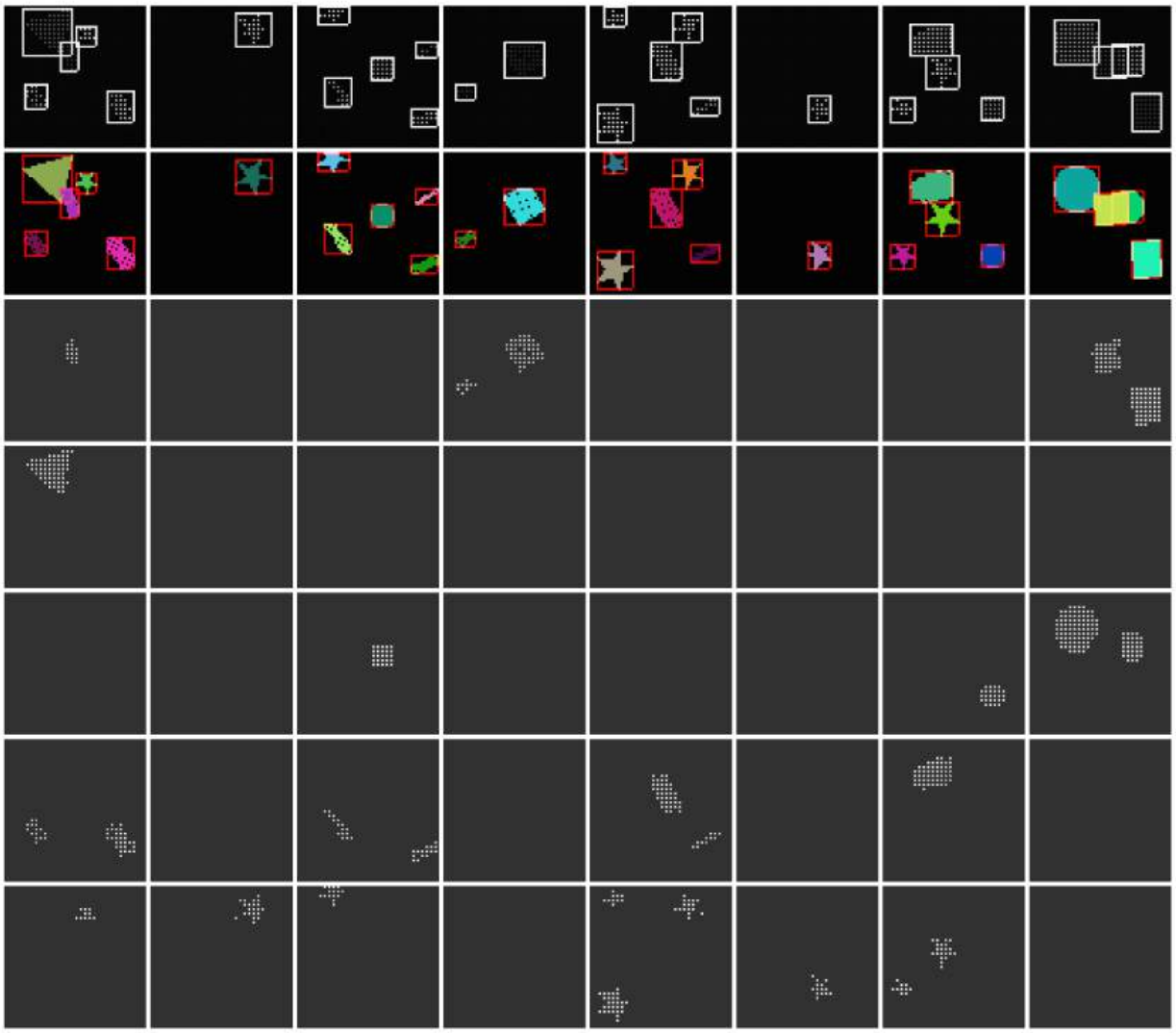
Output of learning rate 1e-4, epoch 12, batch 8



Batch 1



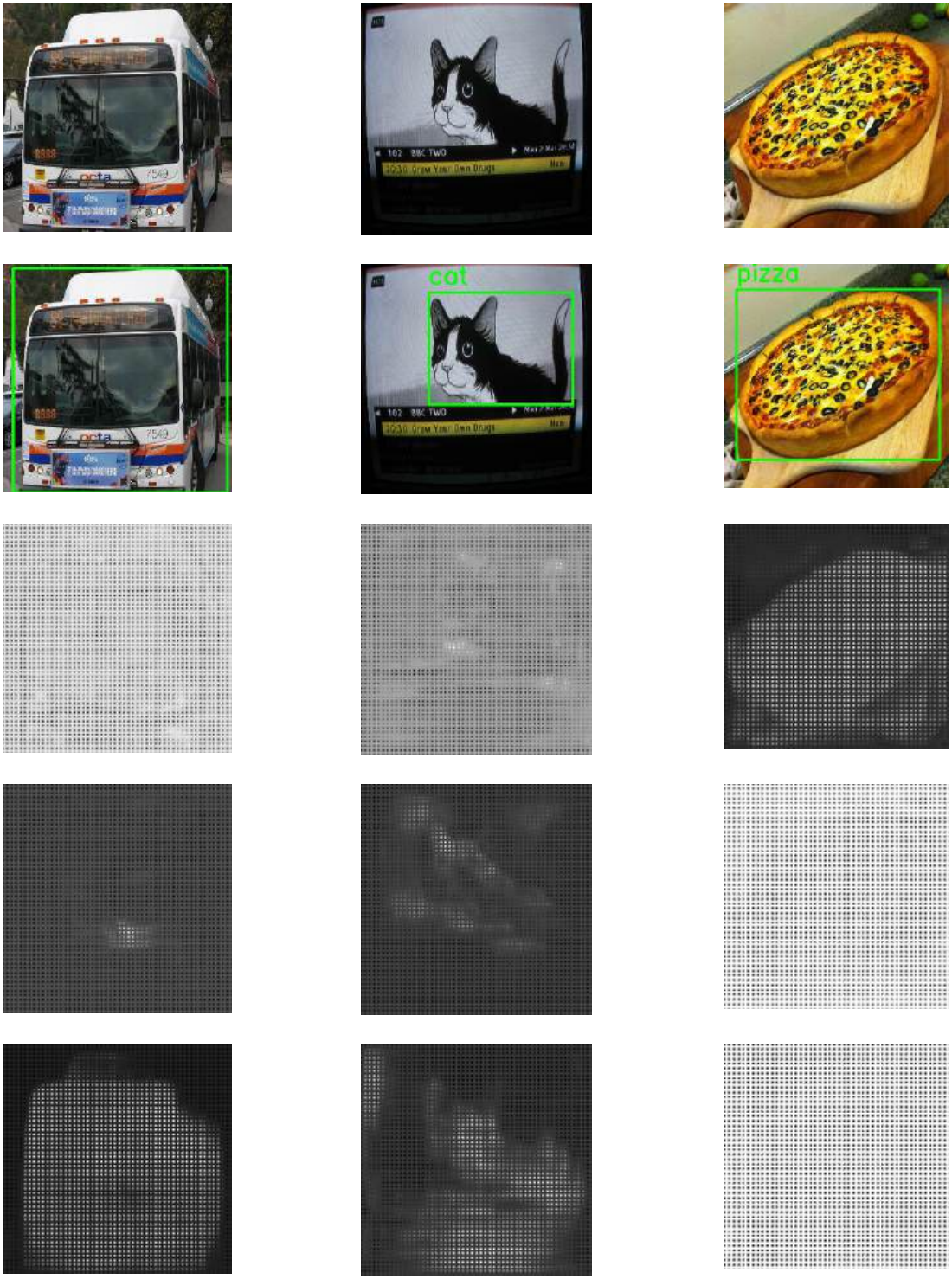
Batch 51



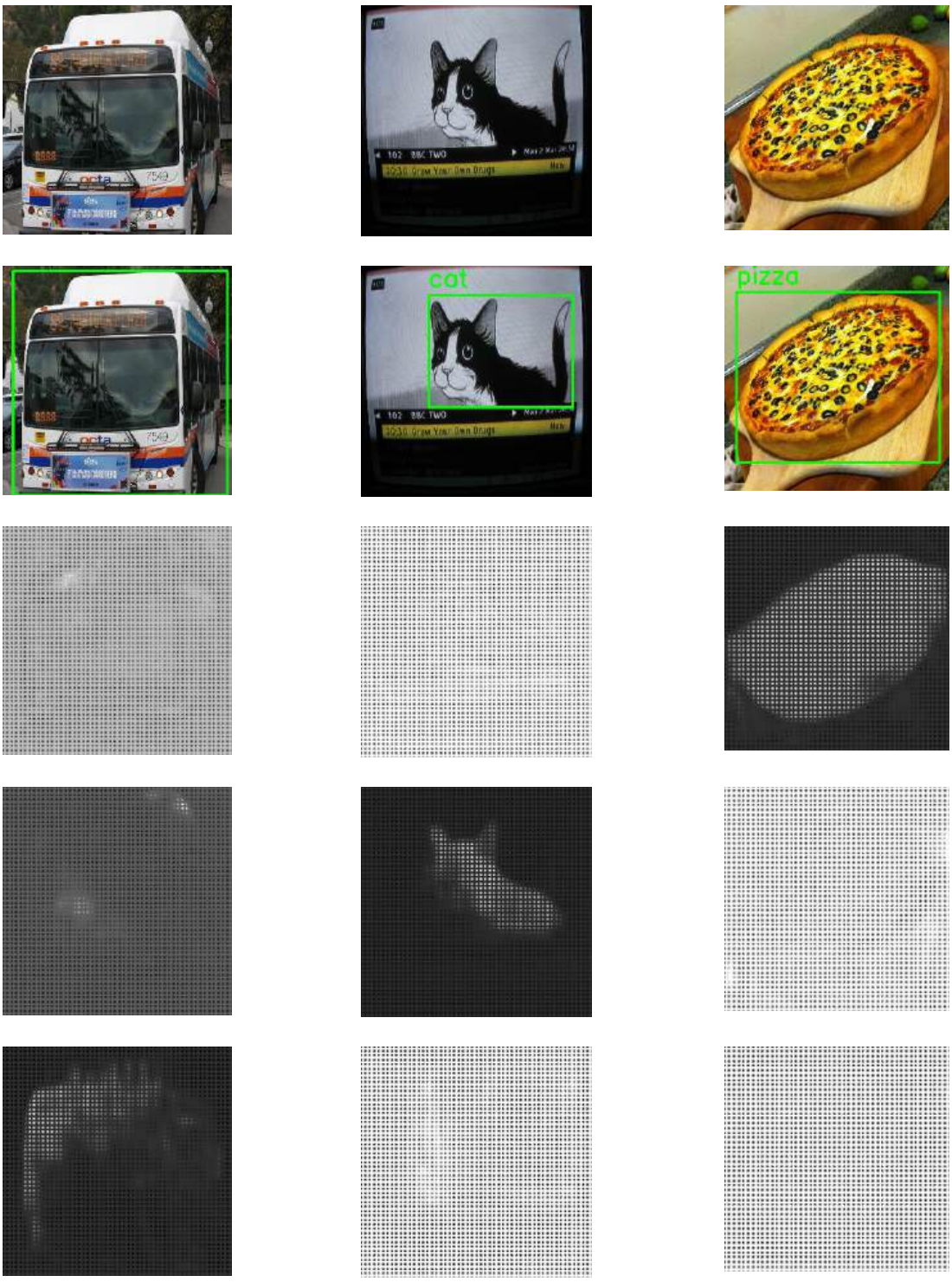
Batch 101

Test Results of MSCOCO

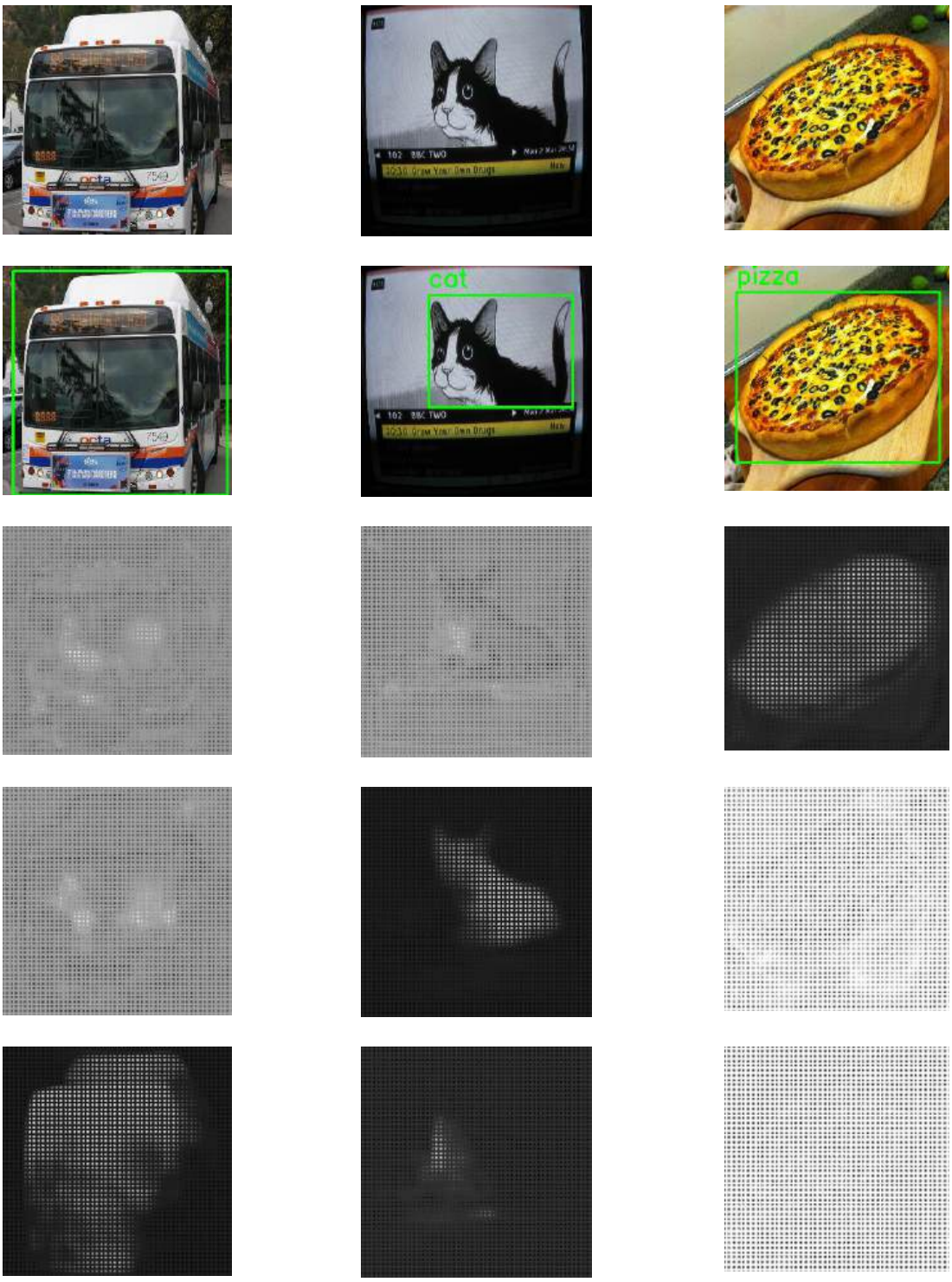
Output of learning rate 1e-4, epoch 20, batch 4



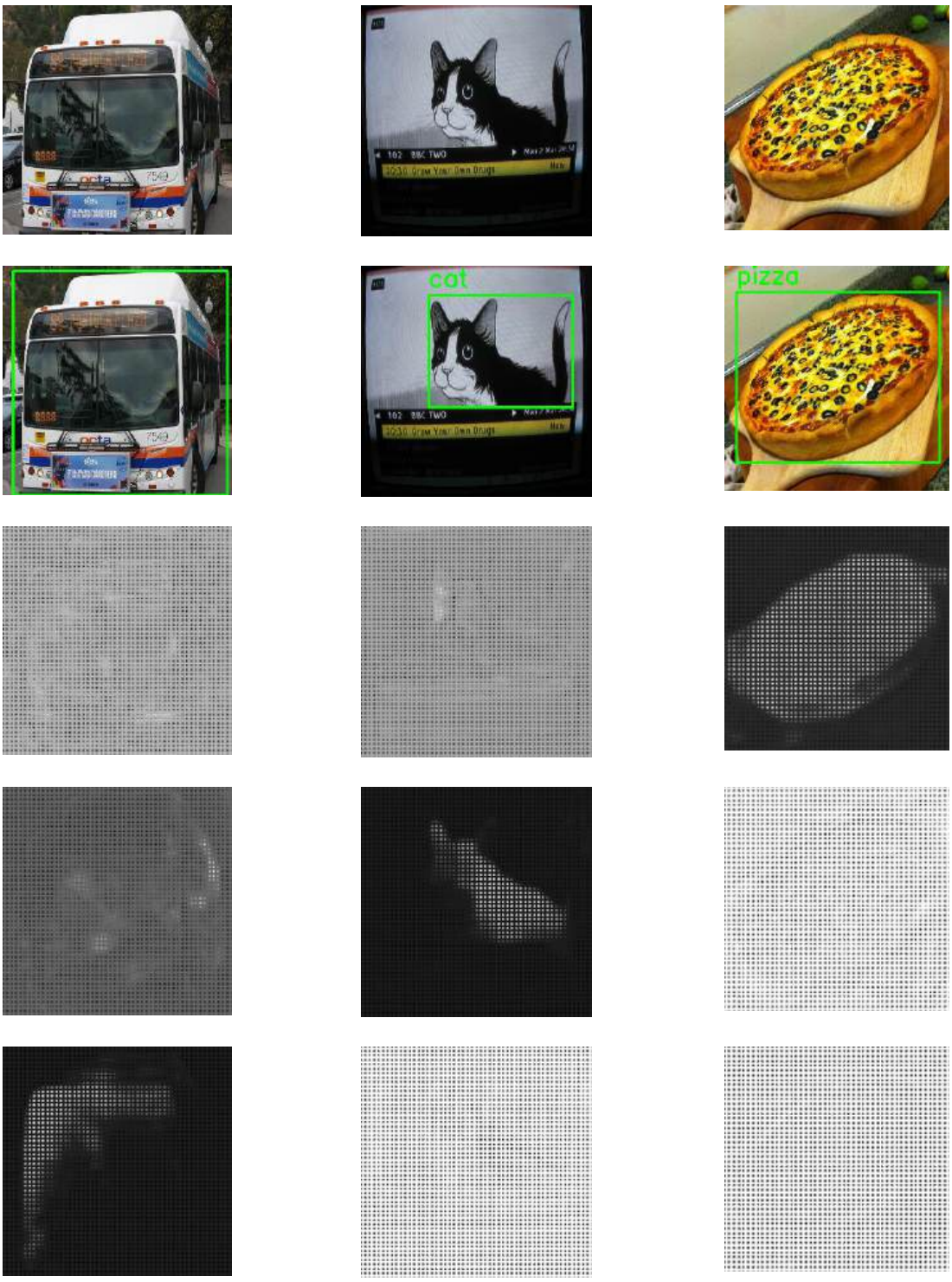
Output of learning rate 1e-4, epoch 30, batch 4



Output of learning rate 1e-5, epoch 20, batch 4



Output of learning rate 1e-5, epoch 30, batch 4



*Mixed loss (MSE+ 100*Dice) Analysis*

Training Loss of PurdueShapes5MultiObjectDataset

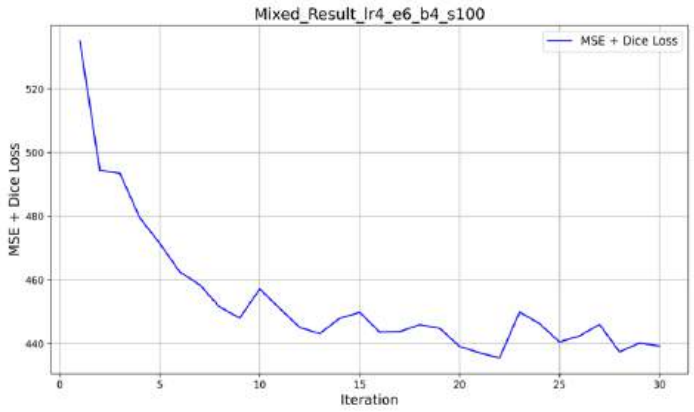


Fig: Loss curve for learning rate 1e-4, epoch 6, batch 4

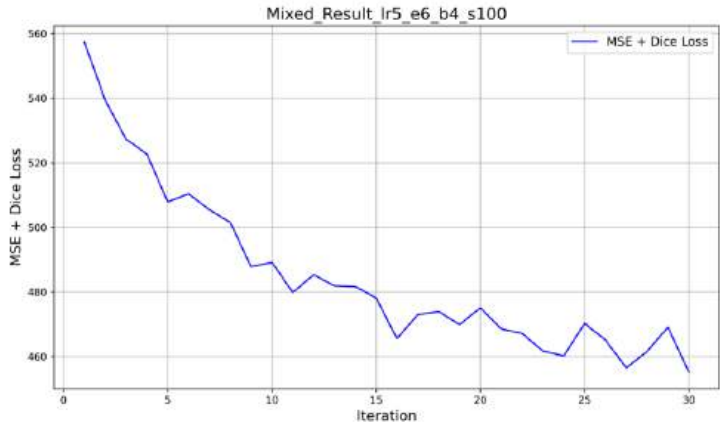


Fig: Loss curve for learning rate 1e-5, epoch 6, batch 4

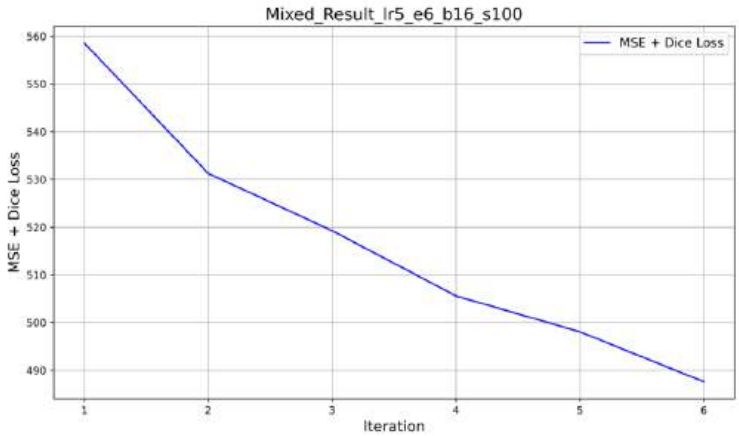


Fig: Loss curve for learning rate 1e-5, epoch 6, batch 16

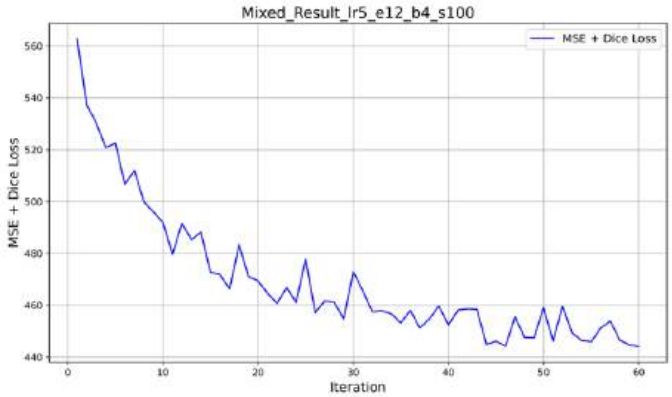


Fig: Loss curve for learning rate 1e-5, epoch 12, batch 4

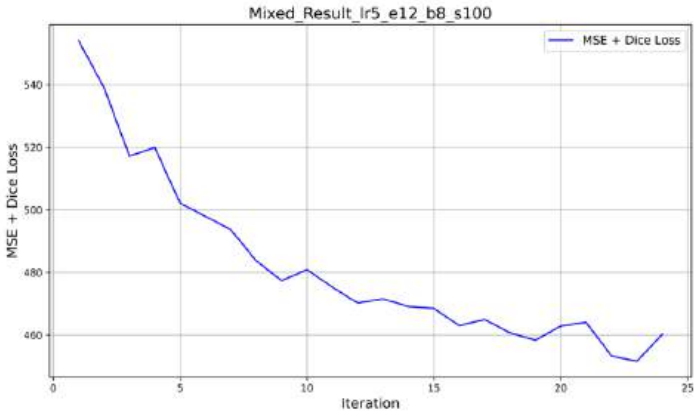


Fig: Loss curve for learning rate 1e-5, epoch 12, batch 8

Training Loss of MSCOCO

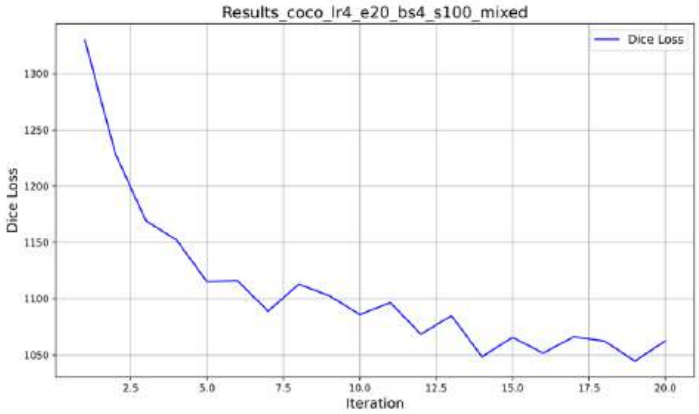


Fig: Loss curve for learning rate 1e-4, epoch 20, batch 4

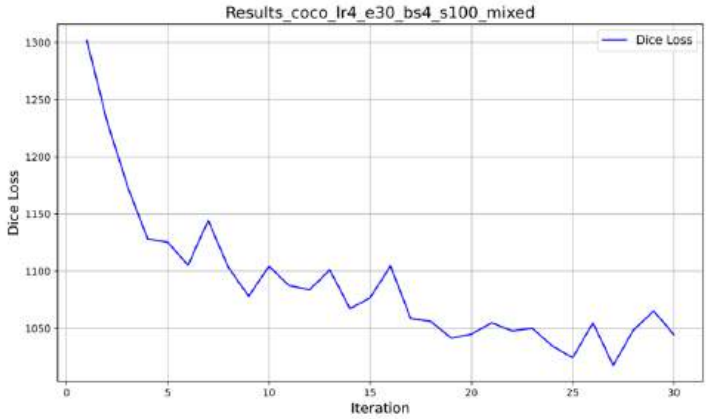


Fig: Loss curve for learning rate 1e-4, epoch 30, batch 4

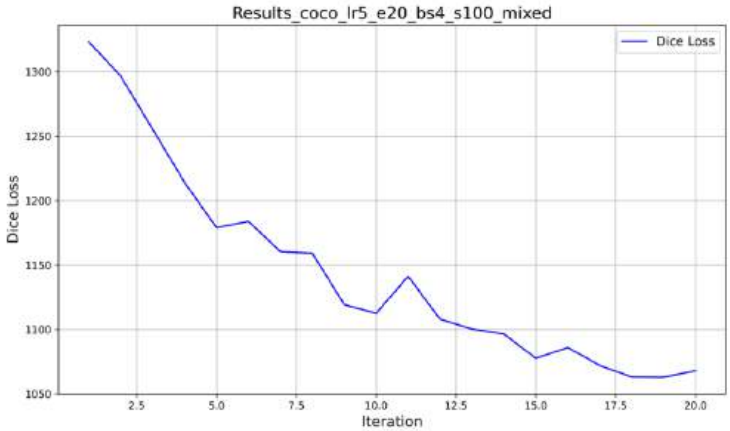


Fig: Loss curve for learning rate 1e-5, epoch 20, batch 4

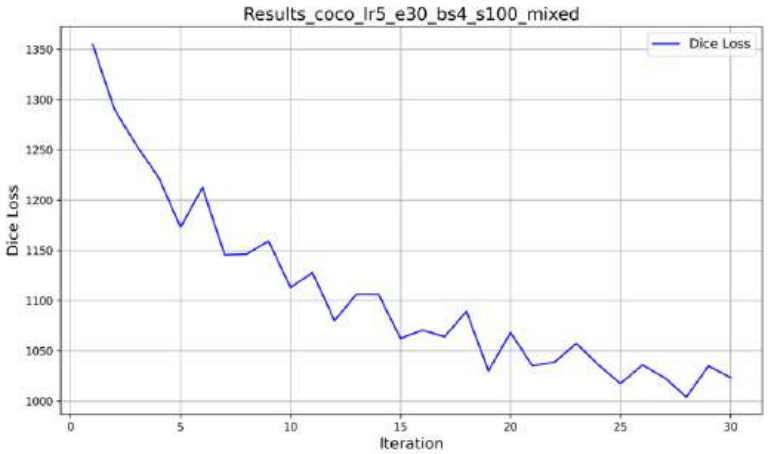
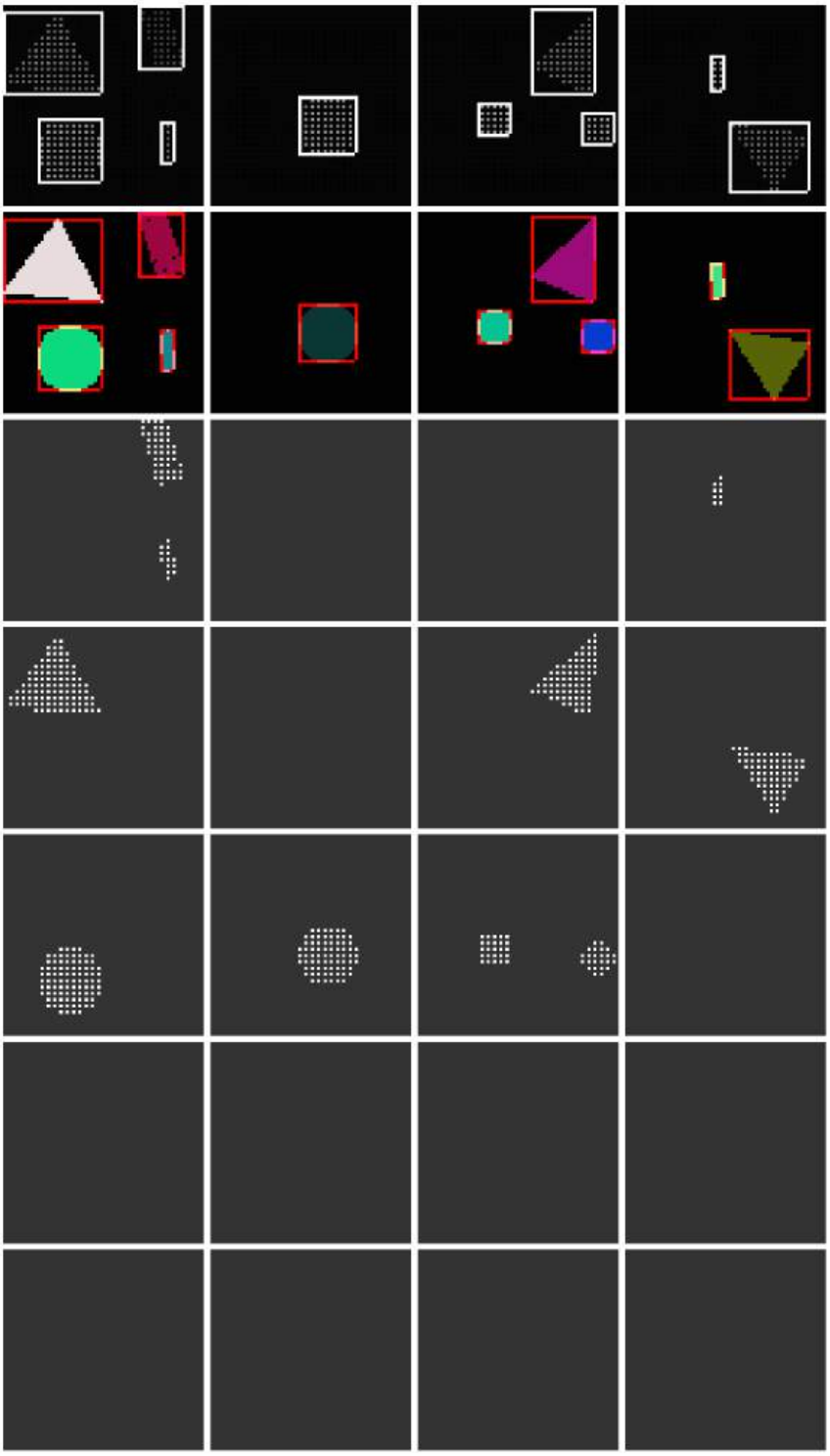


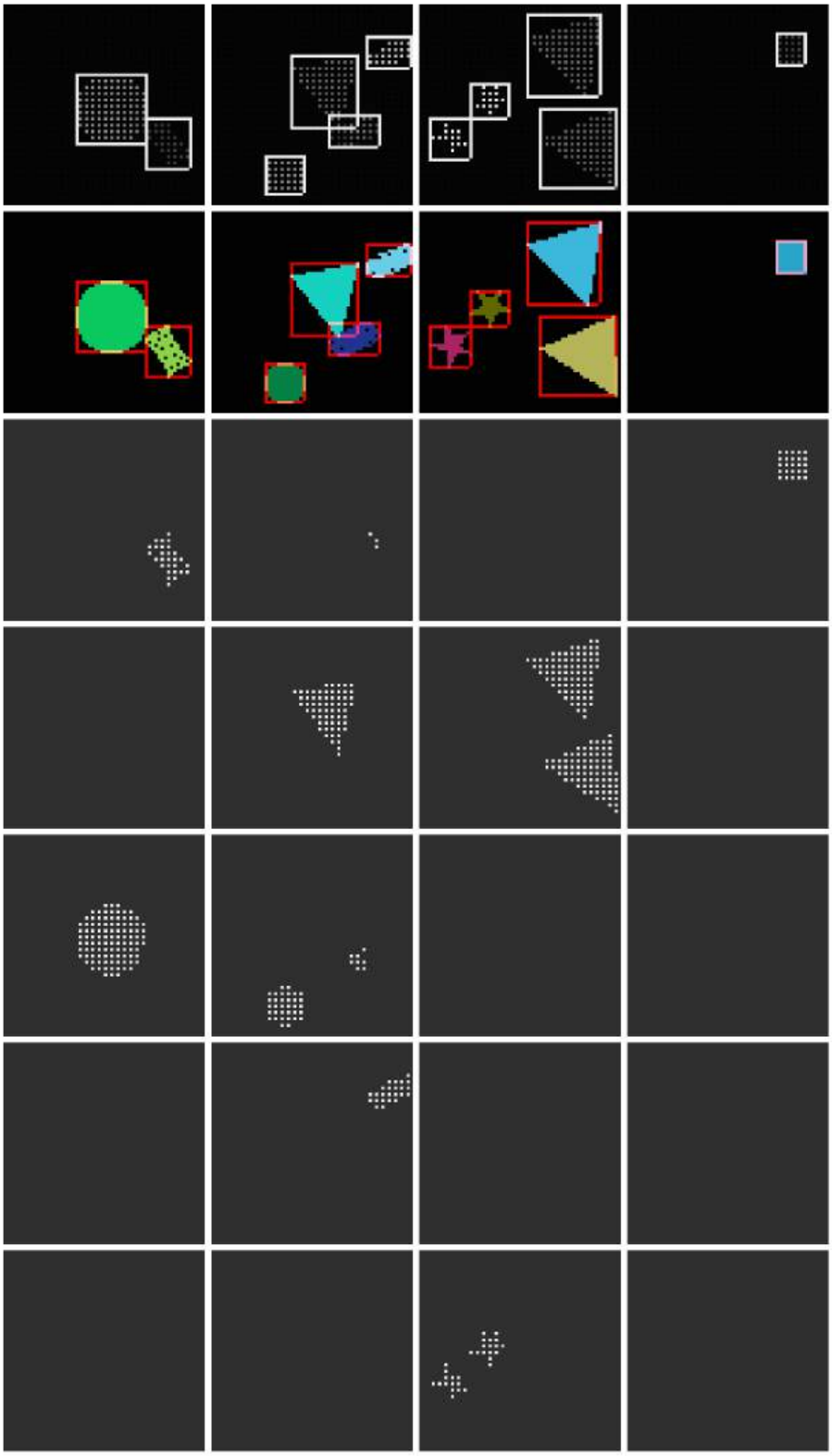
Fig: Loss curve for learning rate 1e-5, epoch 30, batch 4

Test Results of PurdueShapes5MultiObjectDataset

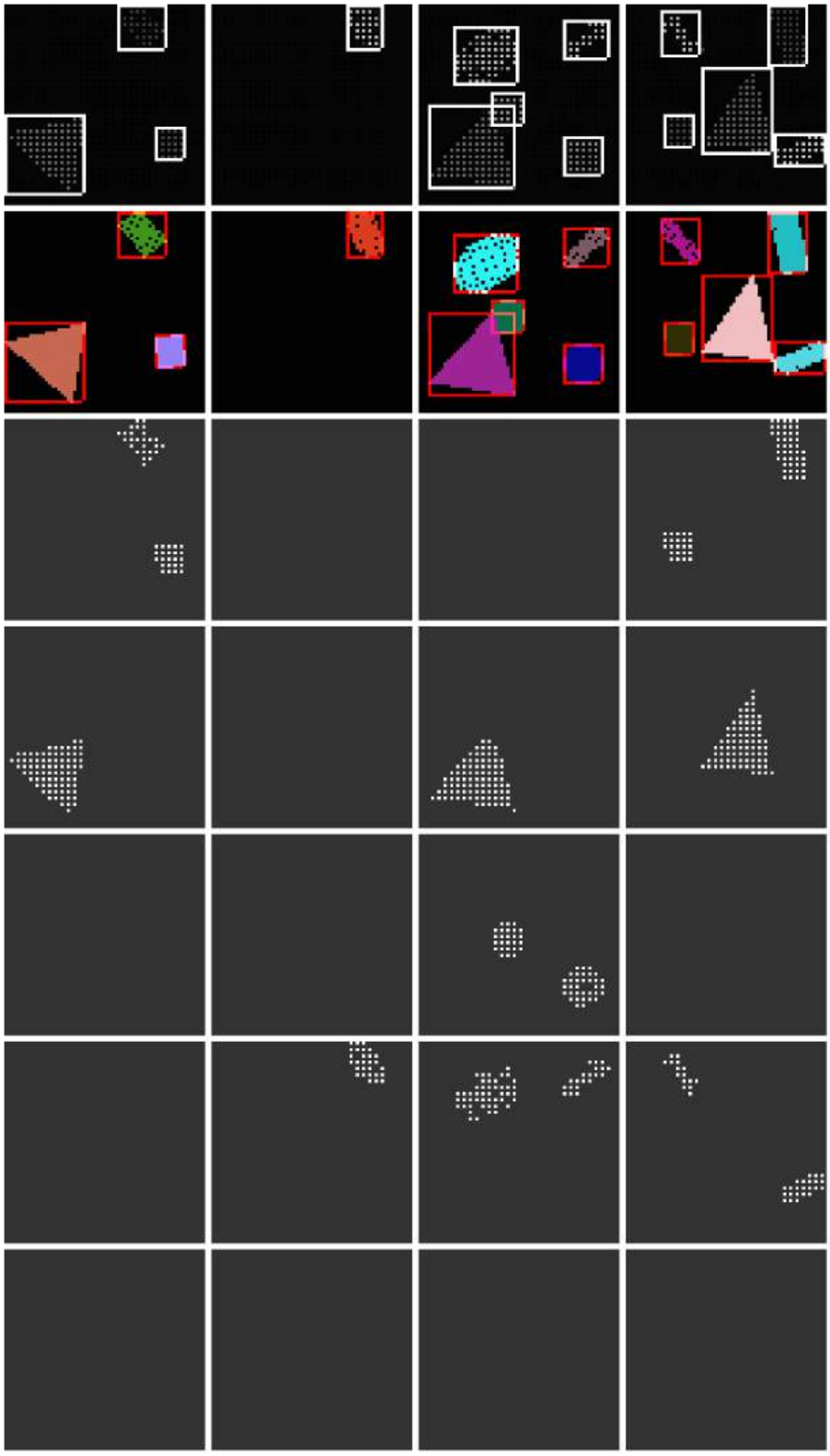
Output of learning rate 1e-4, epoch 6, batch 4



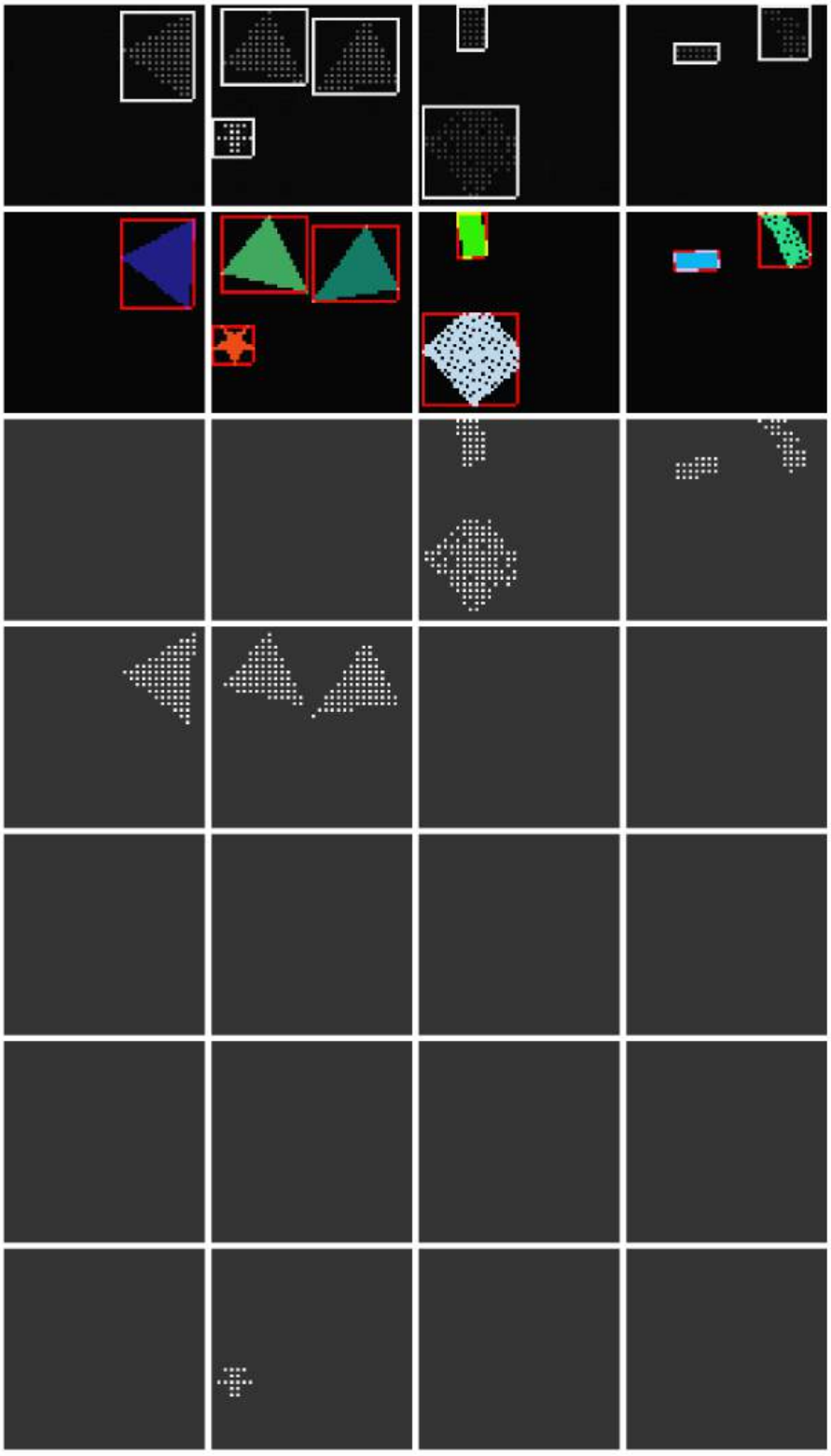
Batch 1



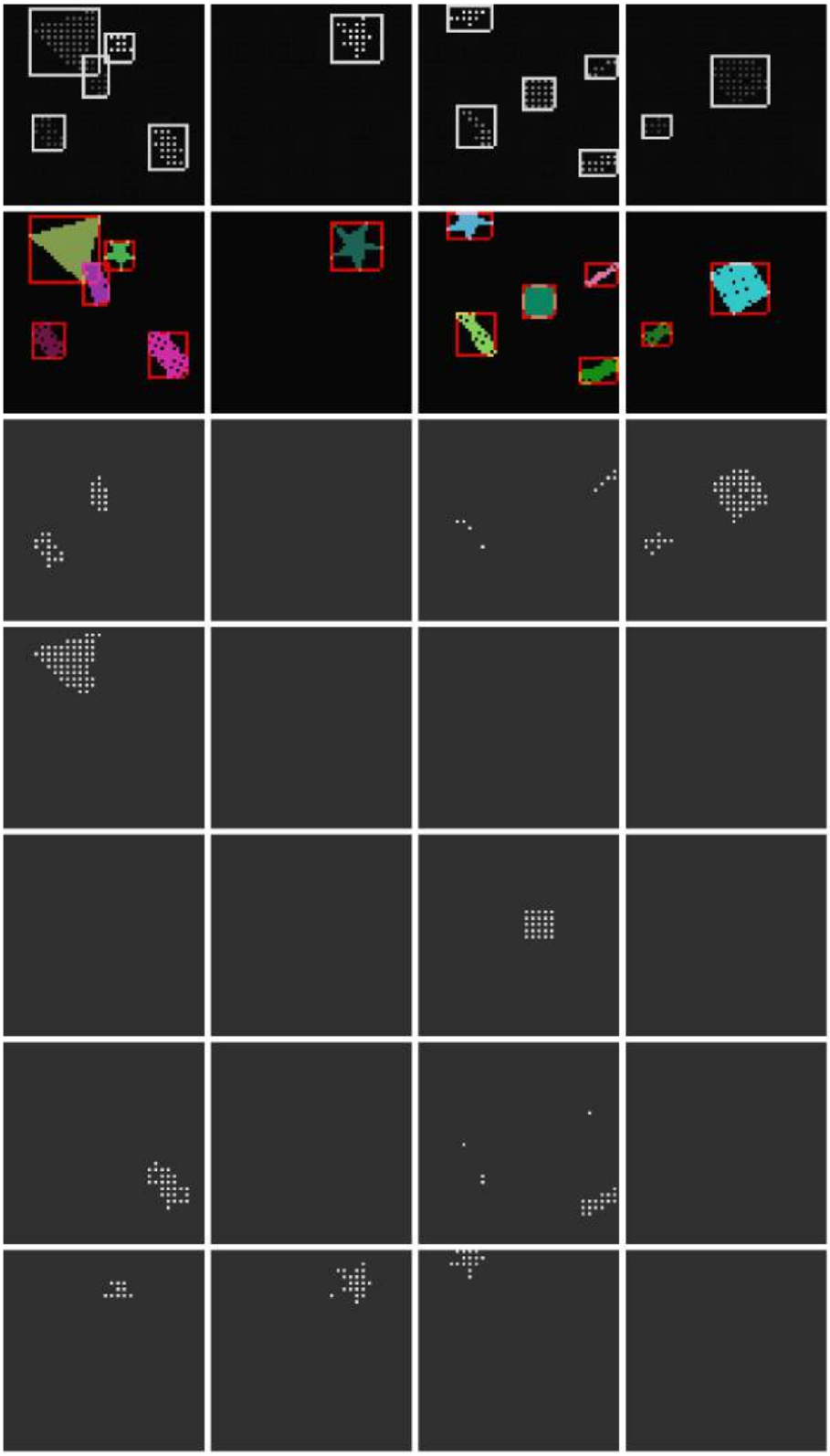
Batch 51



Batch 101

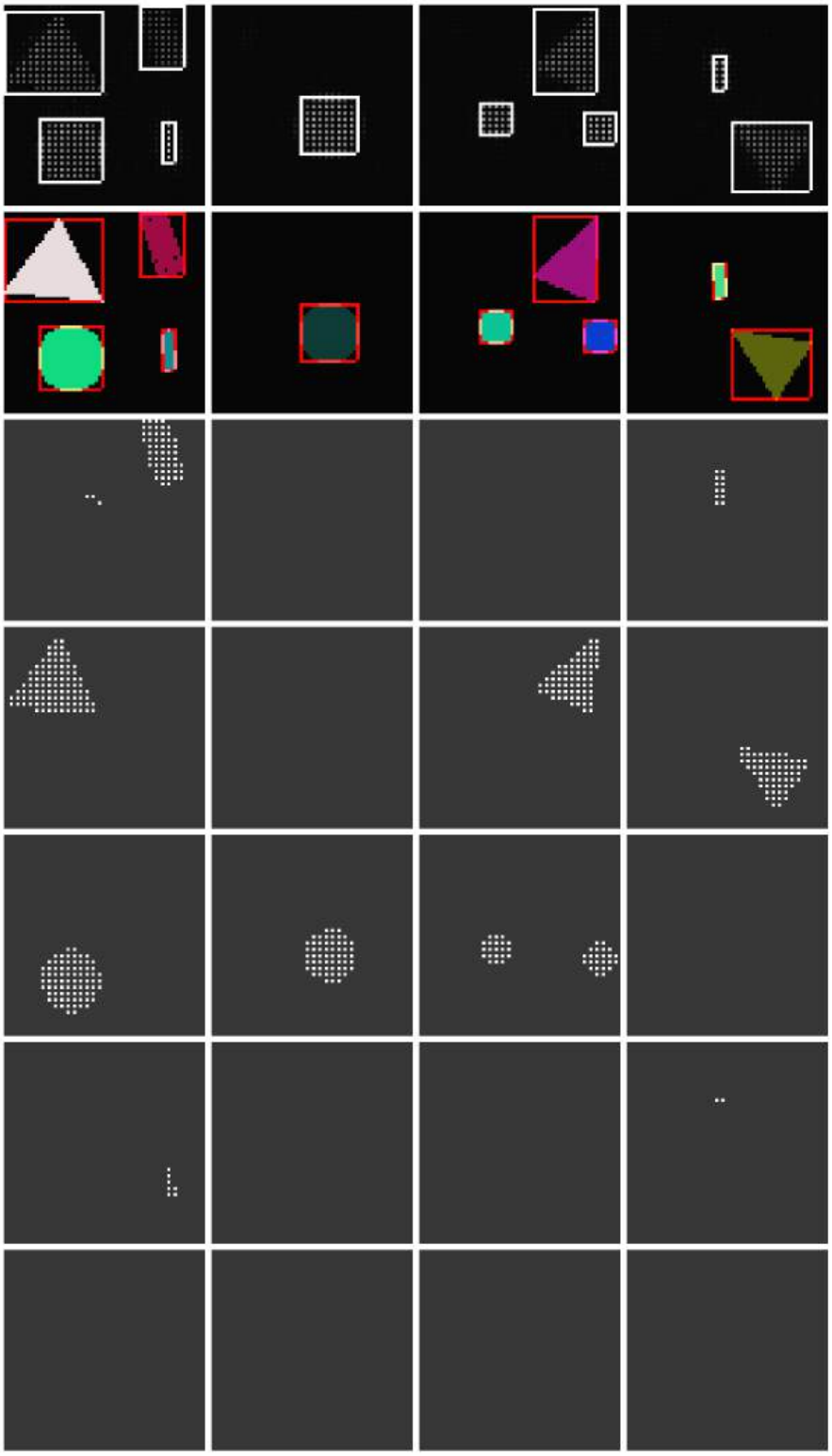


Batch 151

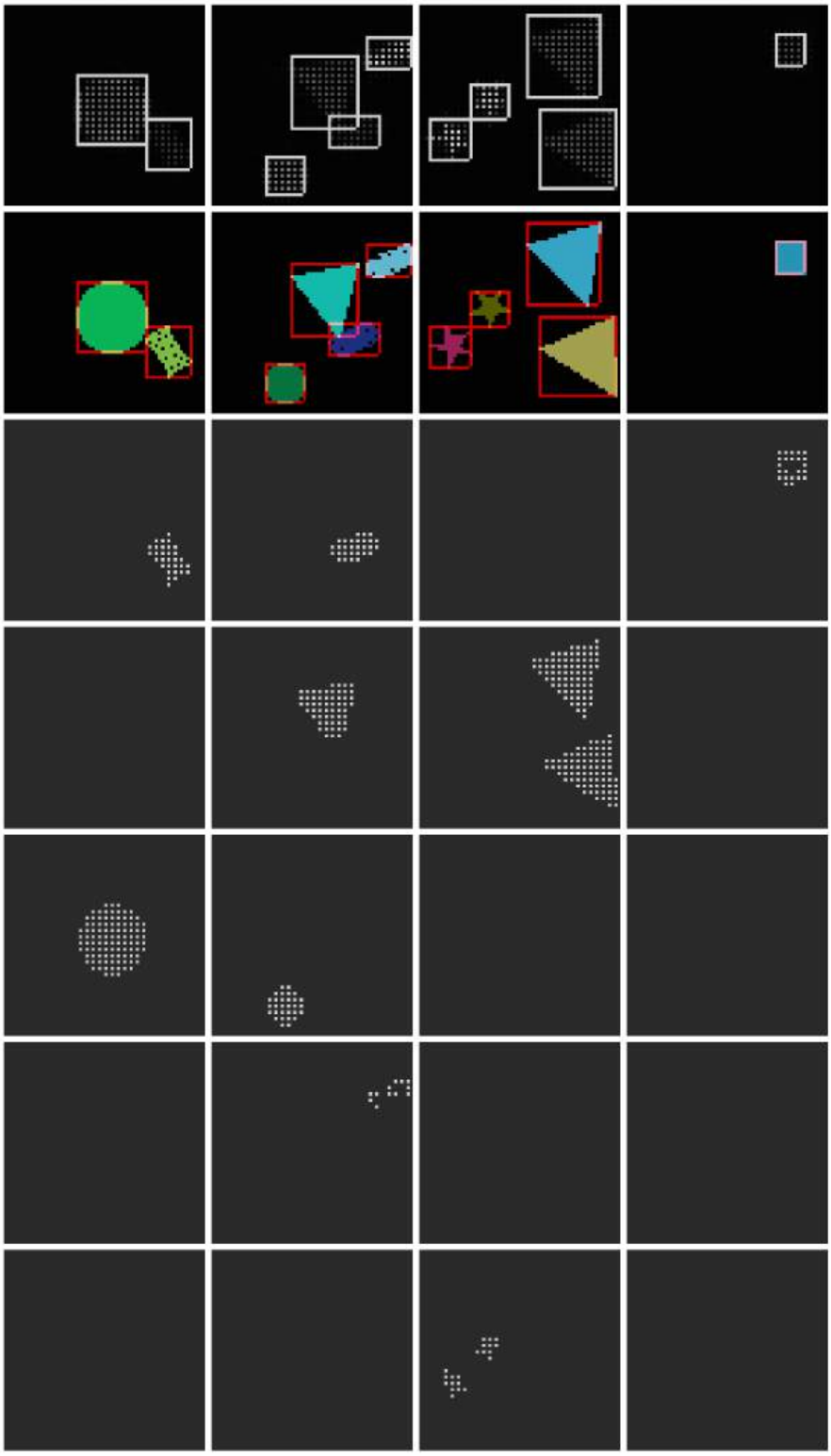


Batch 201

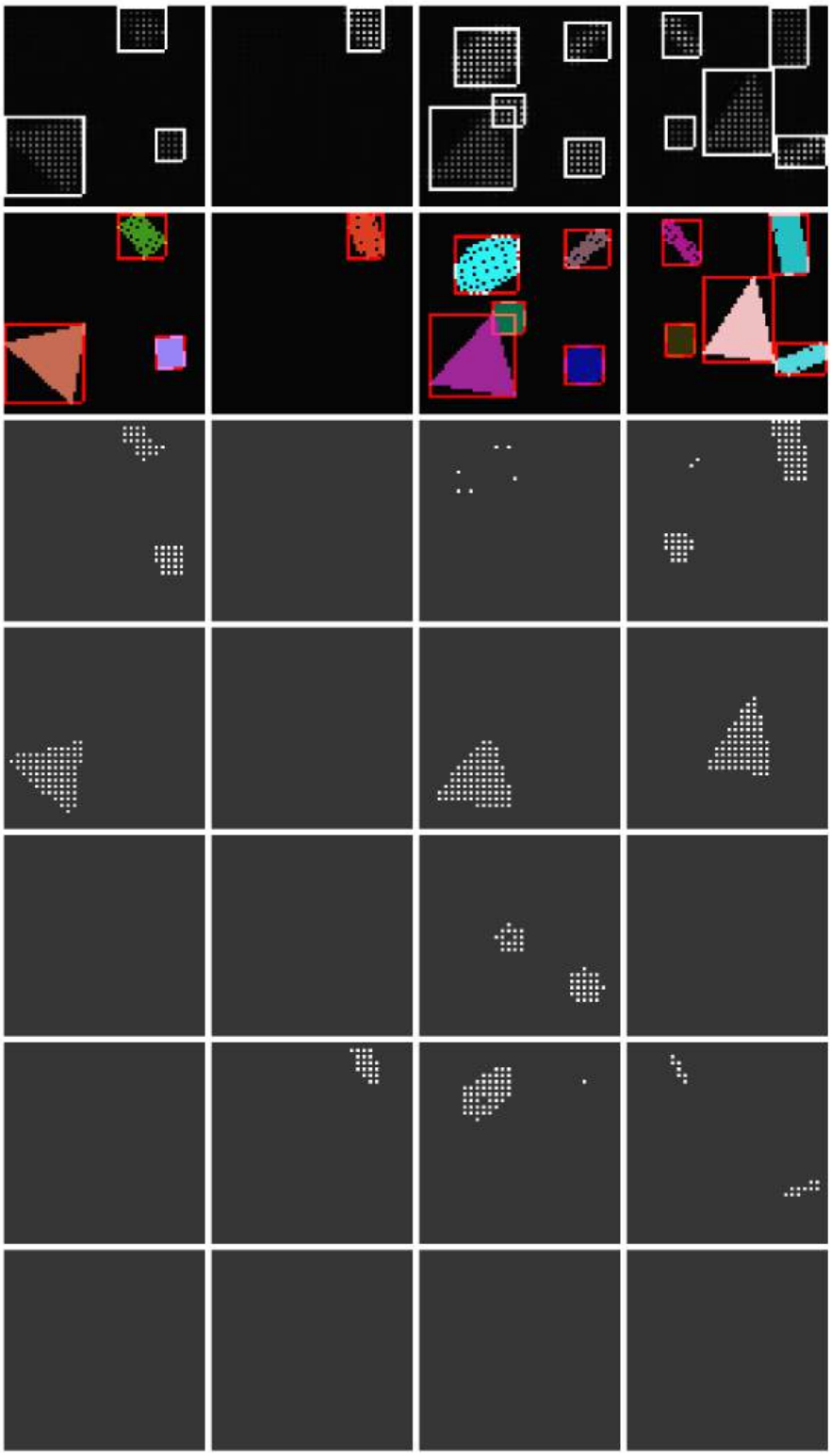
Output of learning rate 1e-5, epoch 6, batch 4



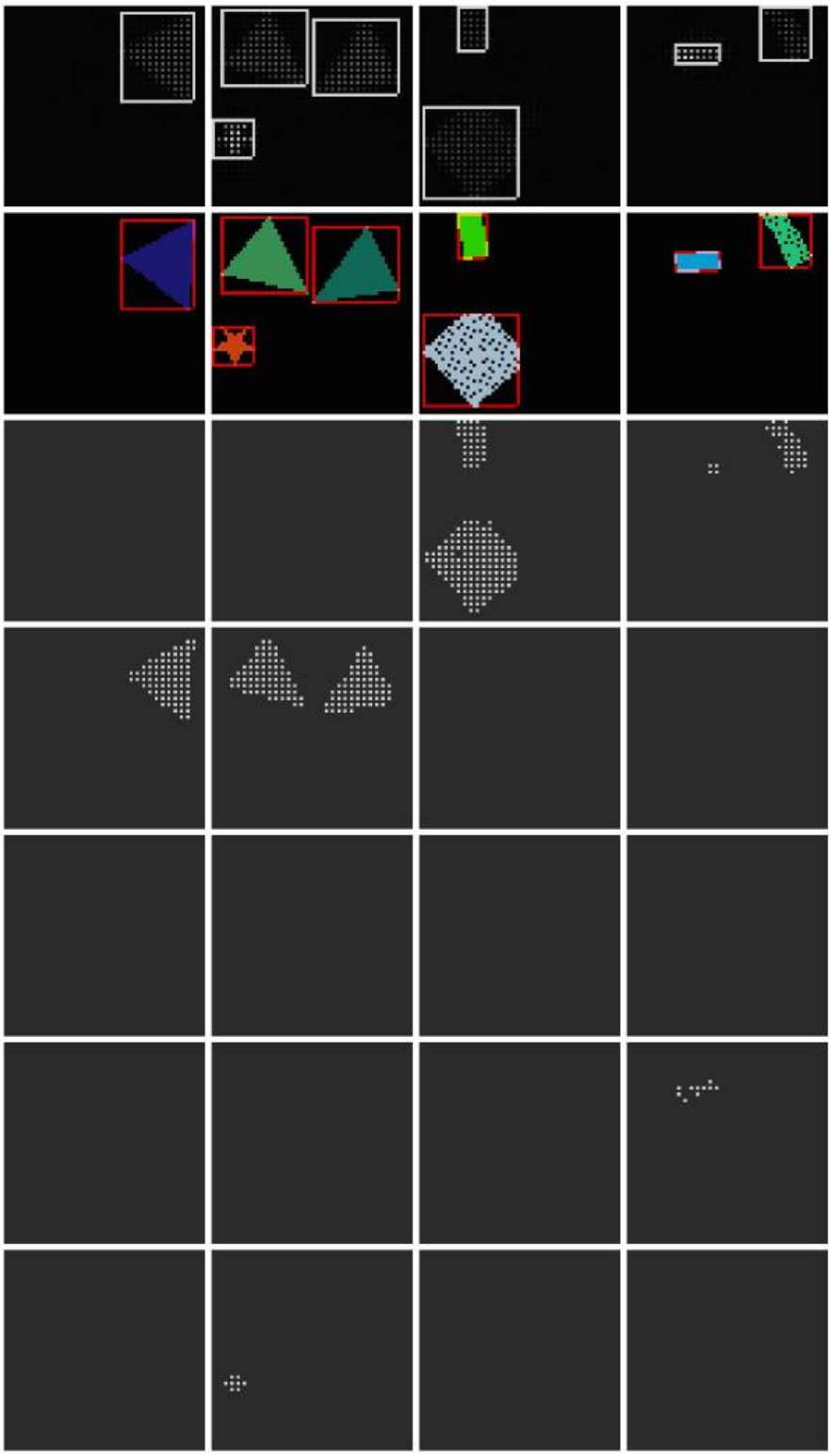
Batch 1



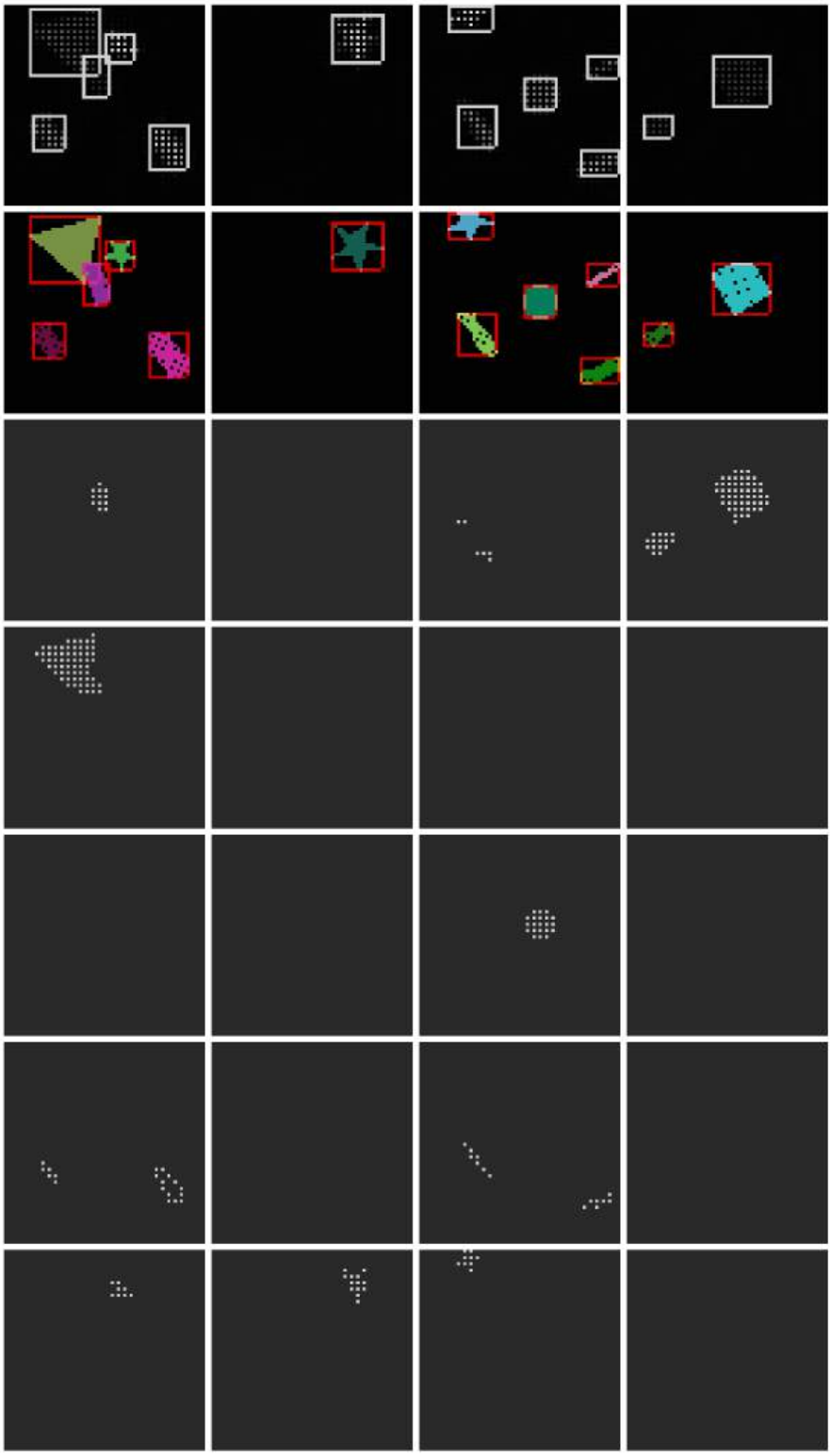
Batch 51



Batch 101

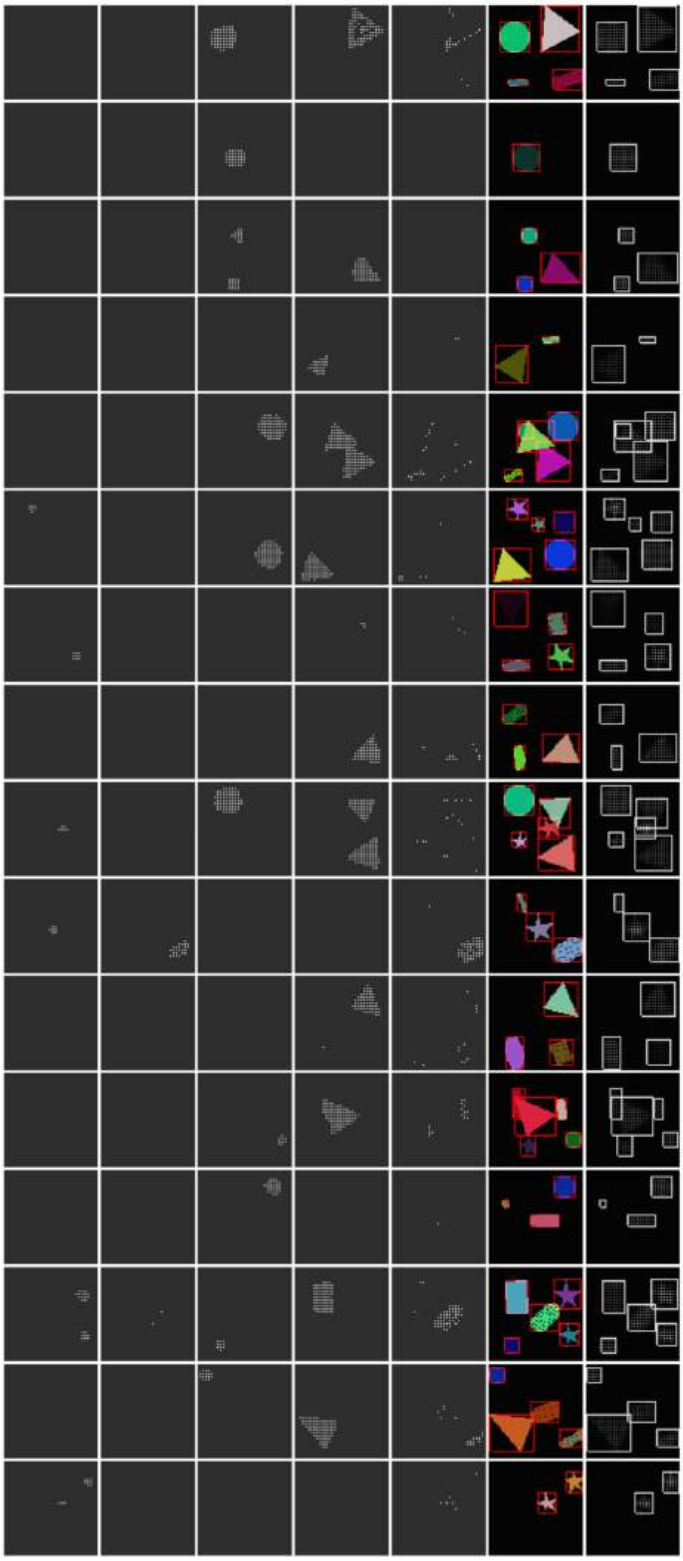


Batch 151

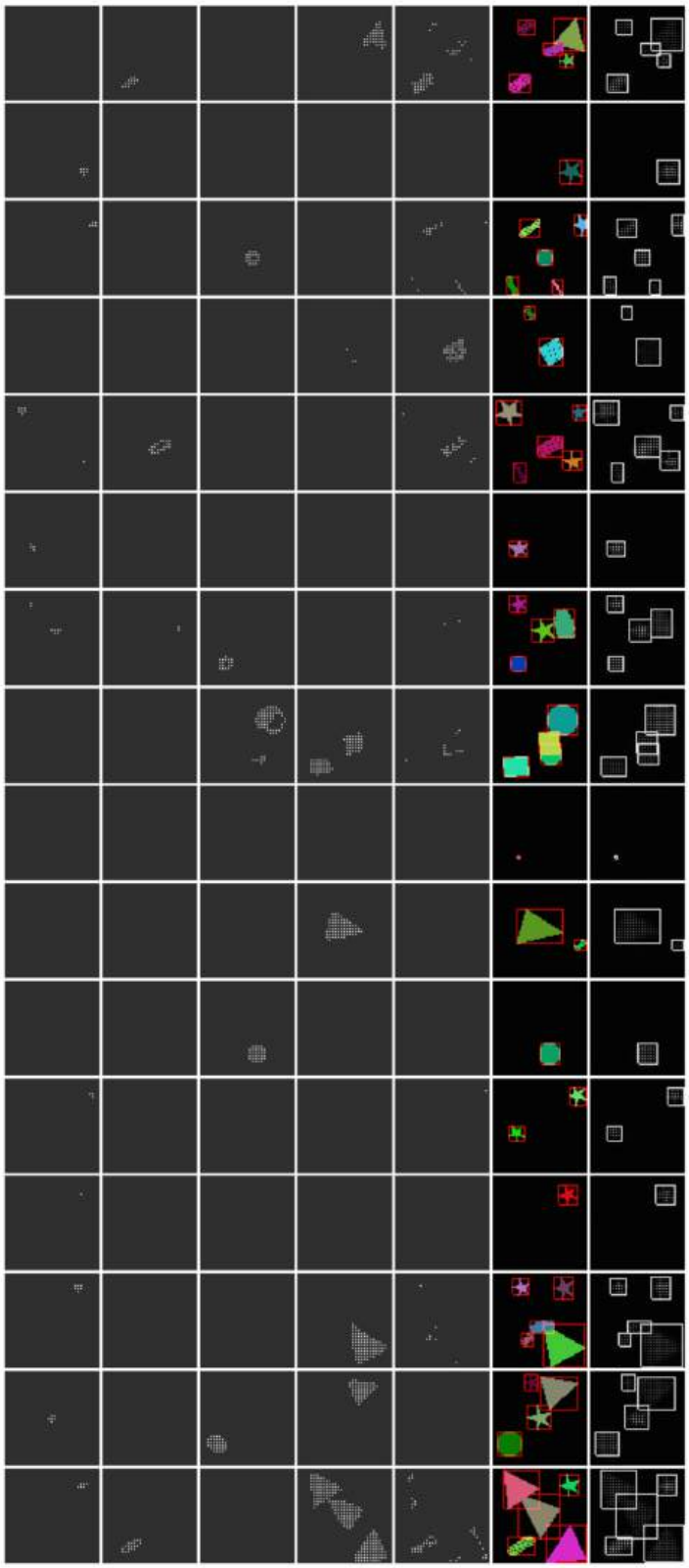


Batch 201

Output of learning rate 1e-5, epoch 6, batch 16

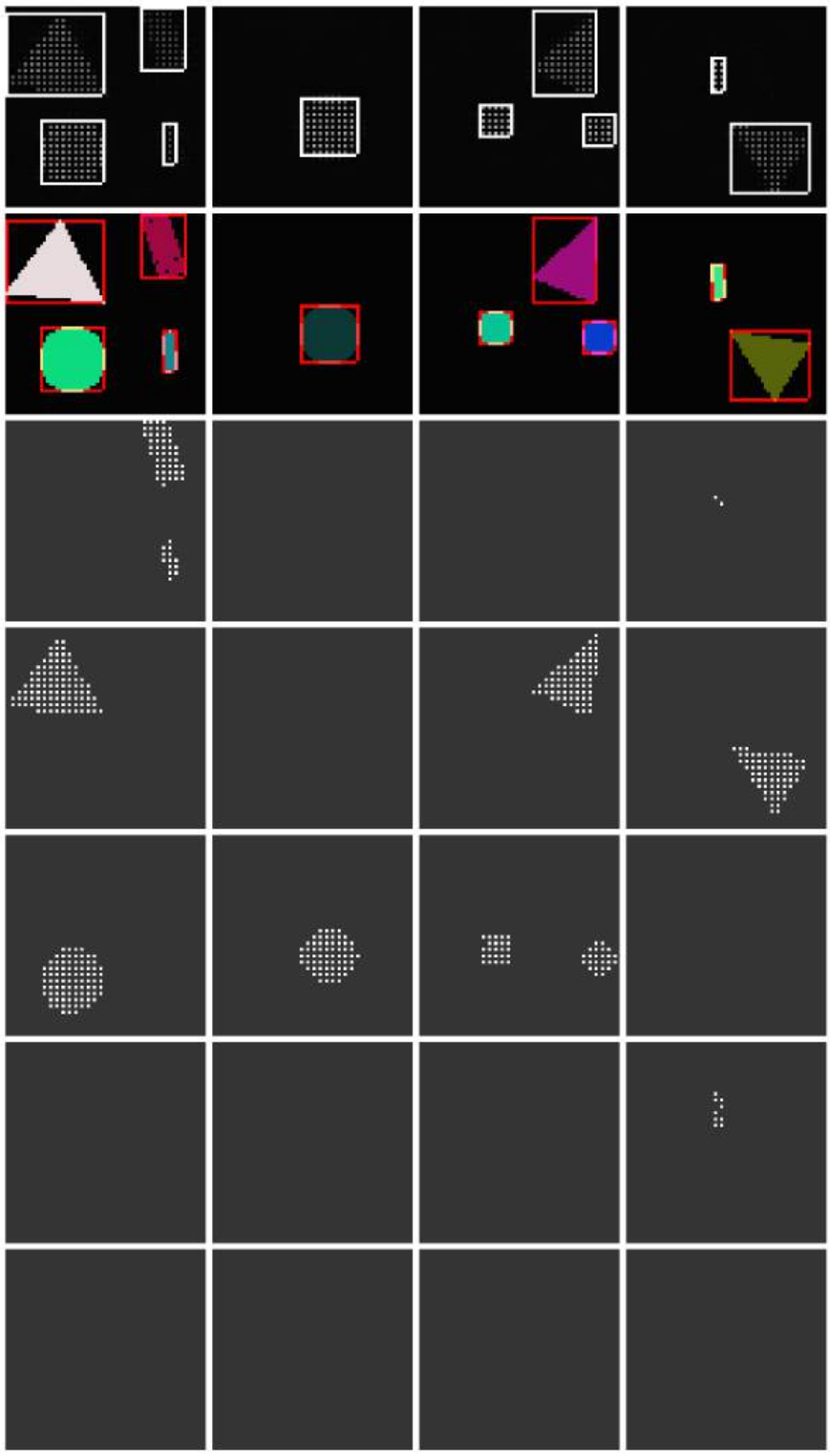


Batch 1

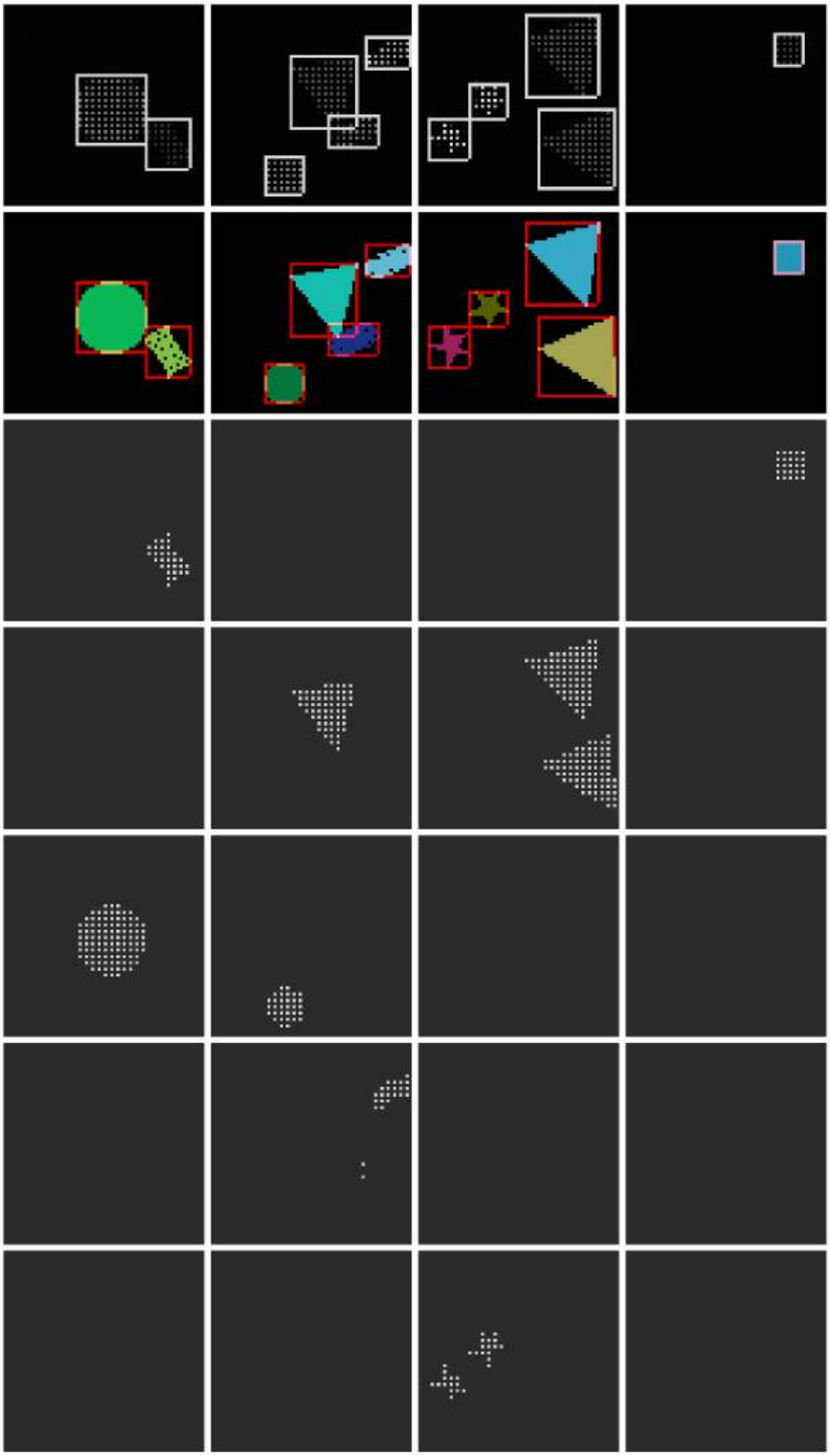


Batch 51

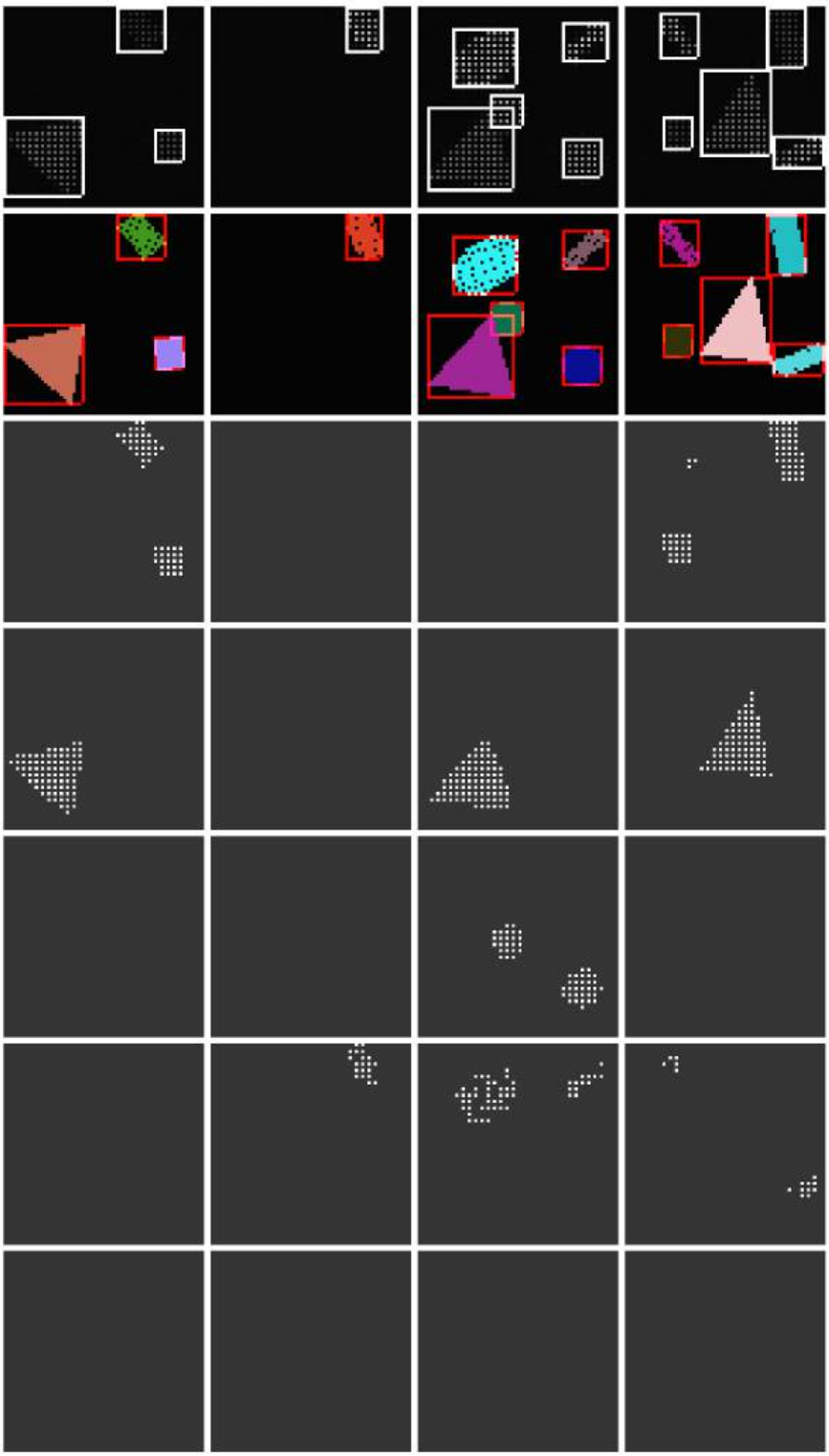
Output of learning rate 1e-5, epoch 12, batch 4



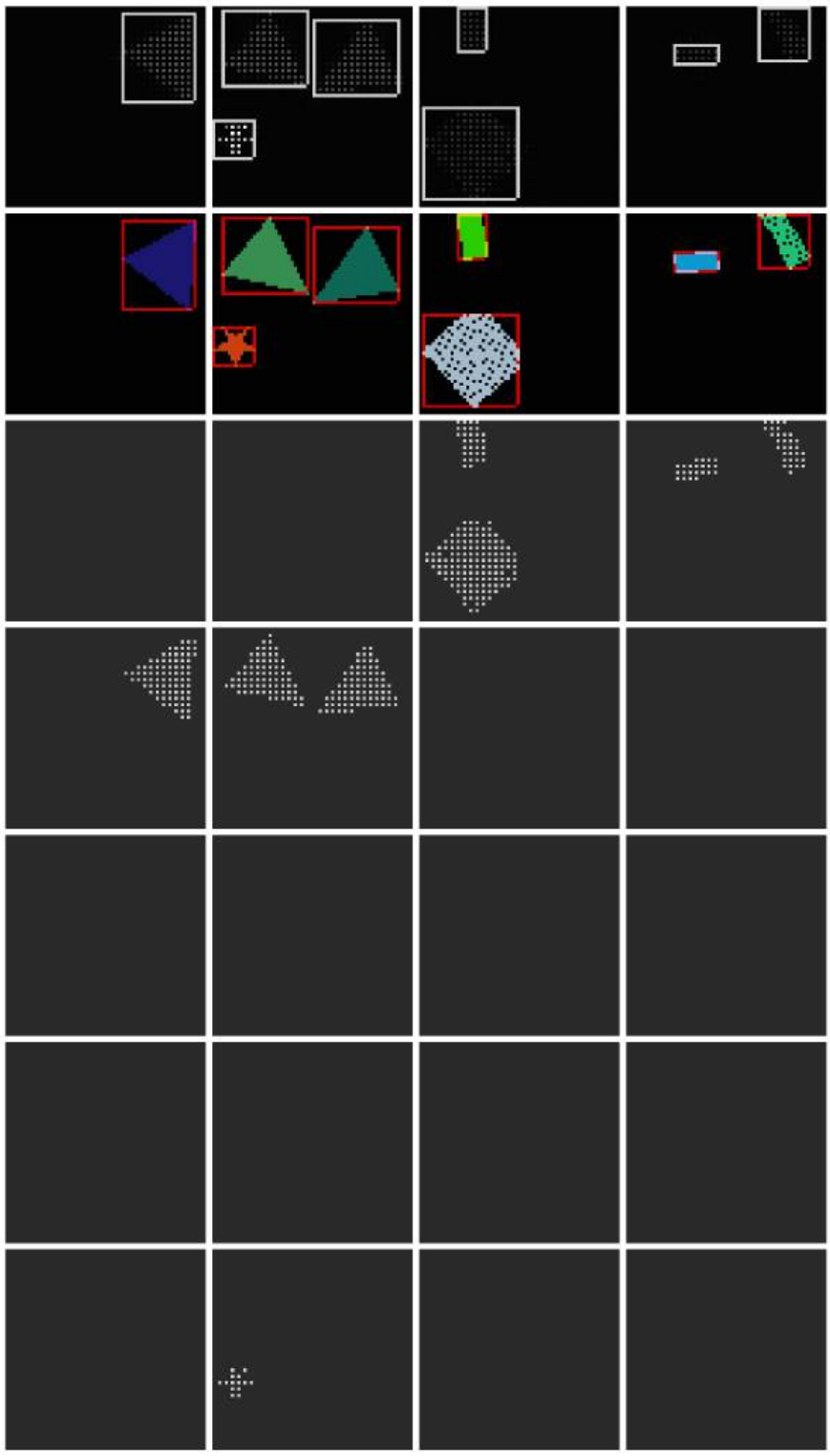
Batch 1



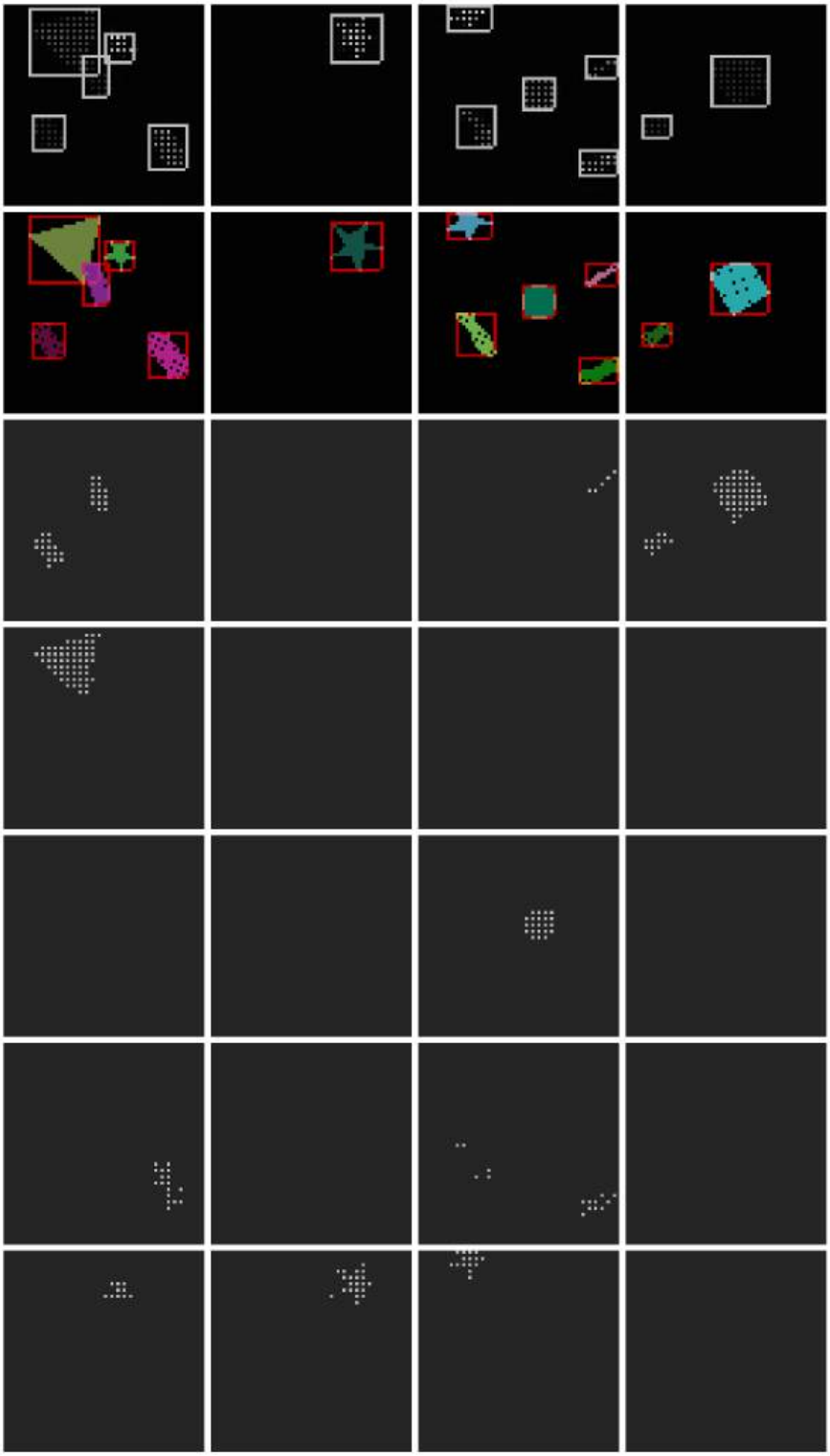
Batch 51



Batch 101

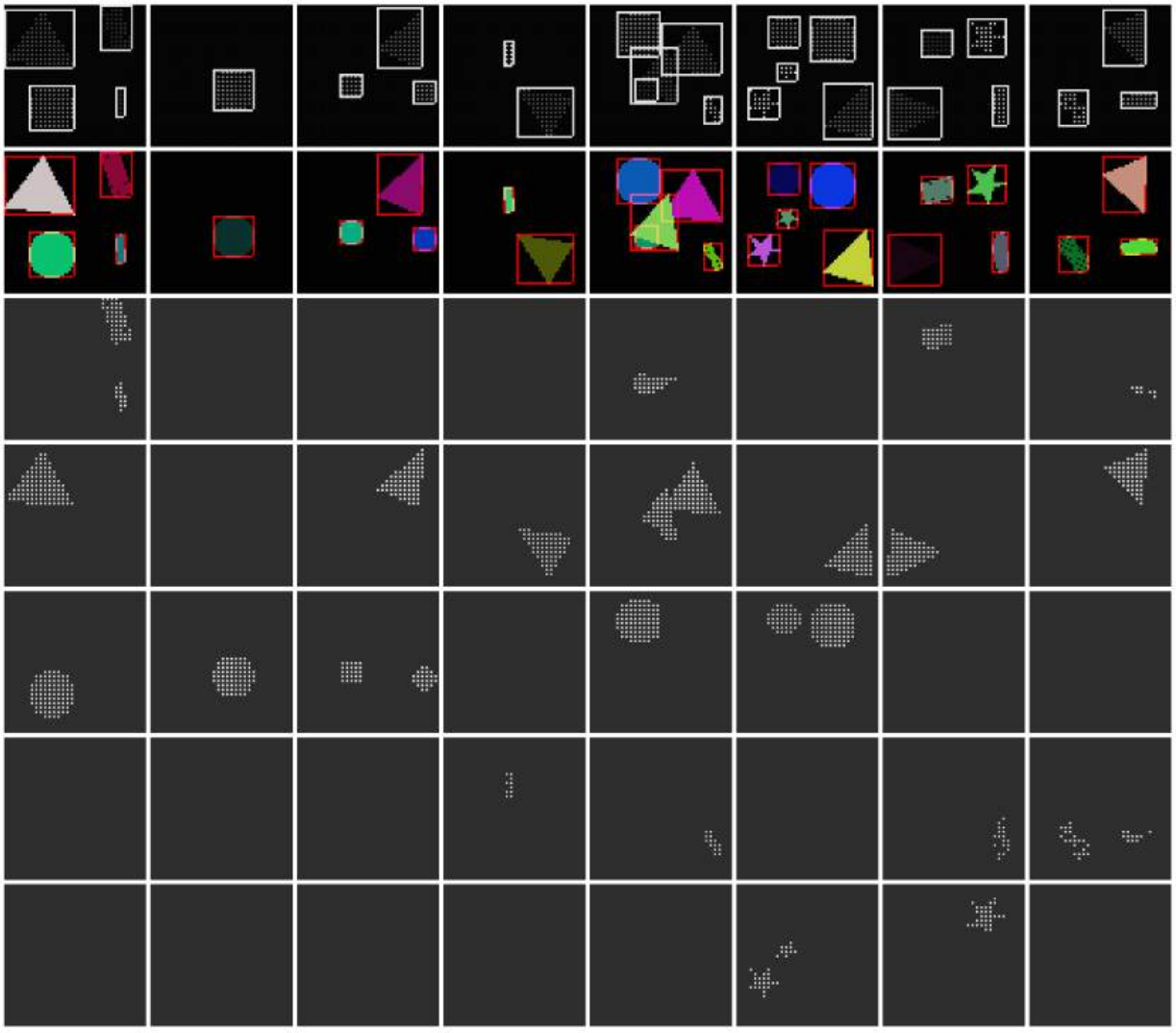


Batch 151

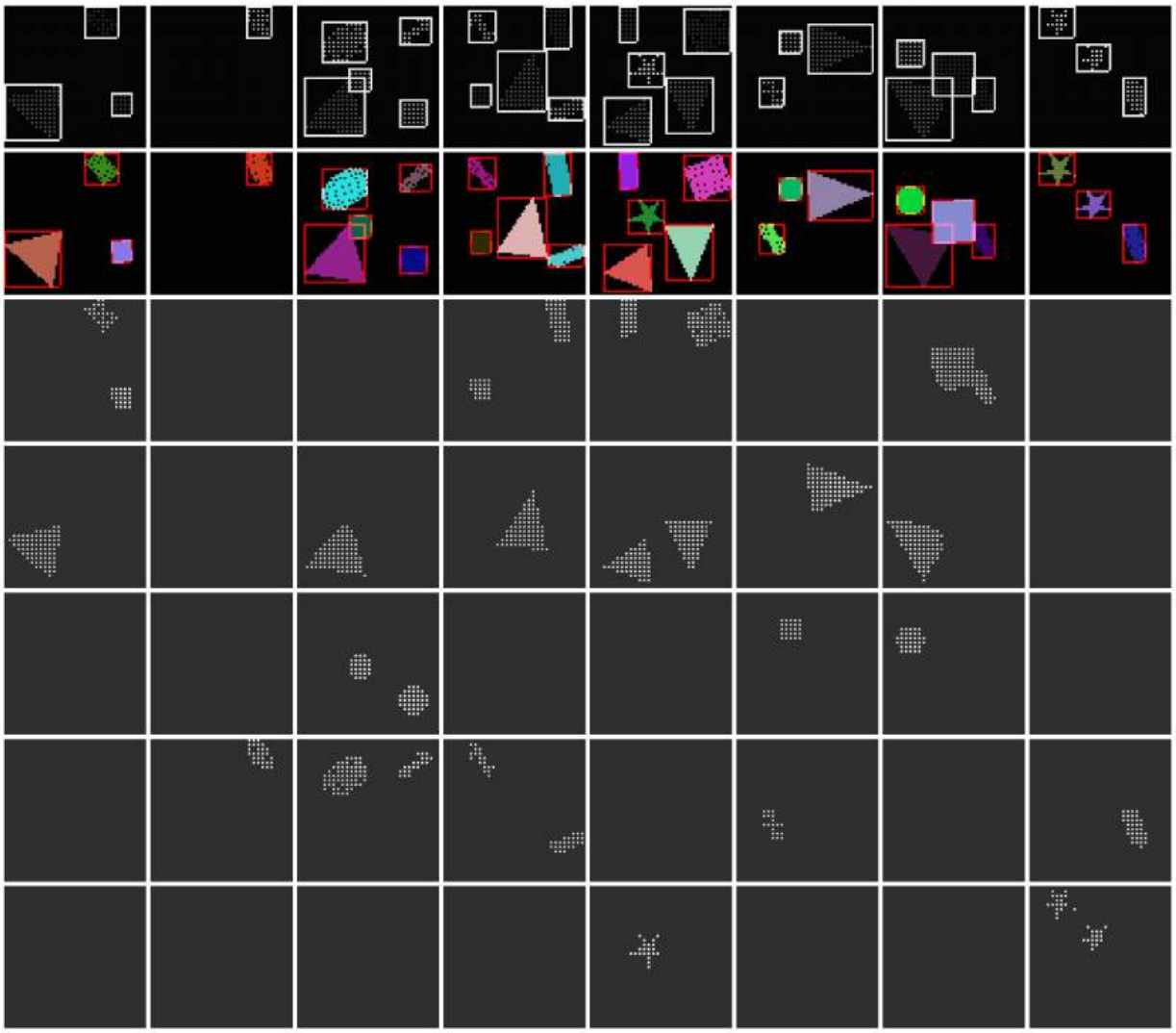


Batch 201

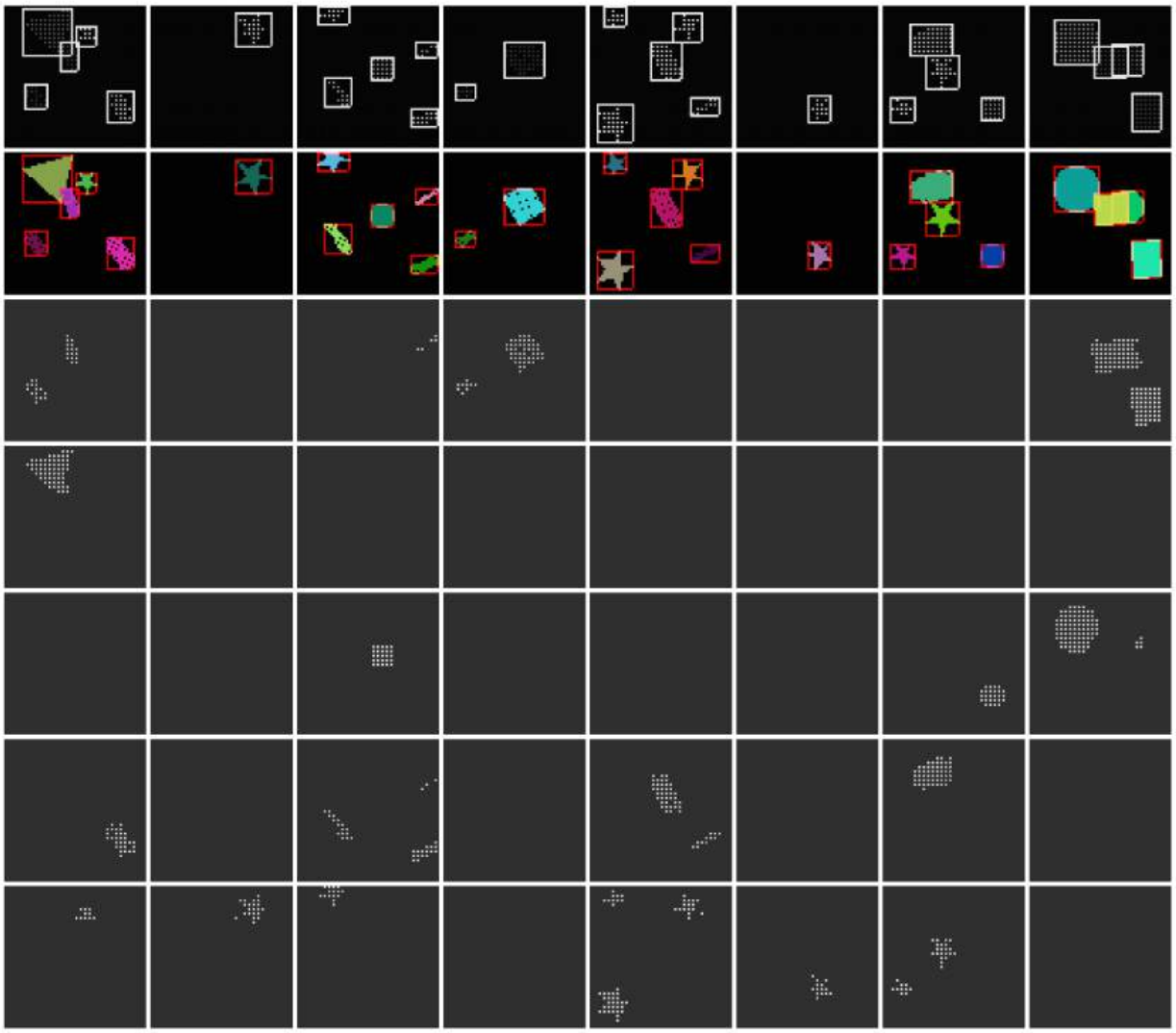
Output of learning rate 1e-4, epoch 12, batch 8



Batch 1



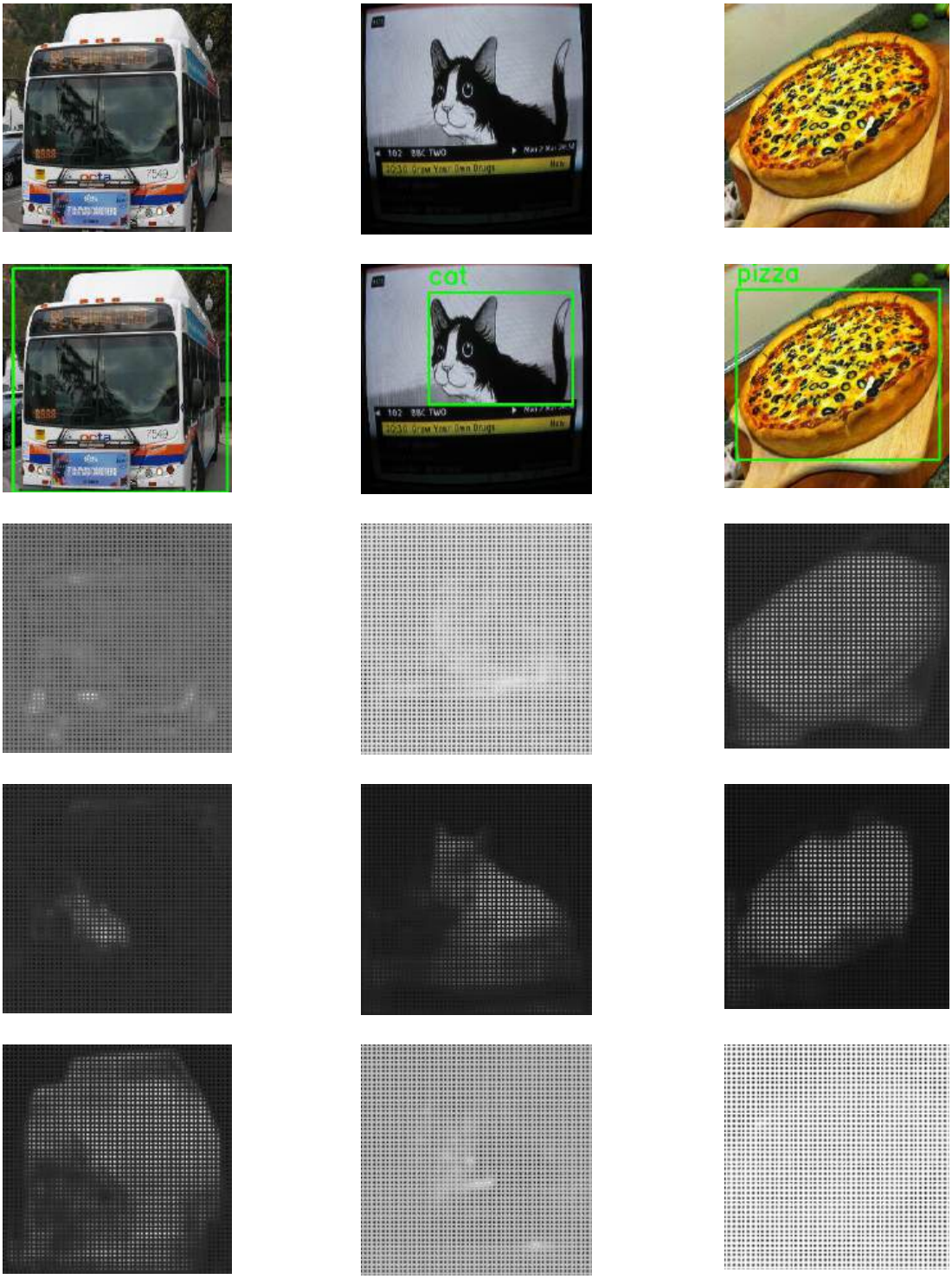
Batch 51



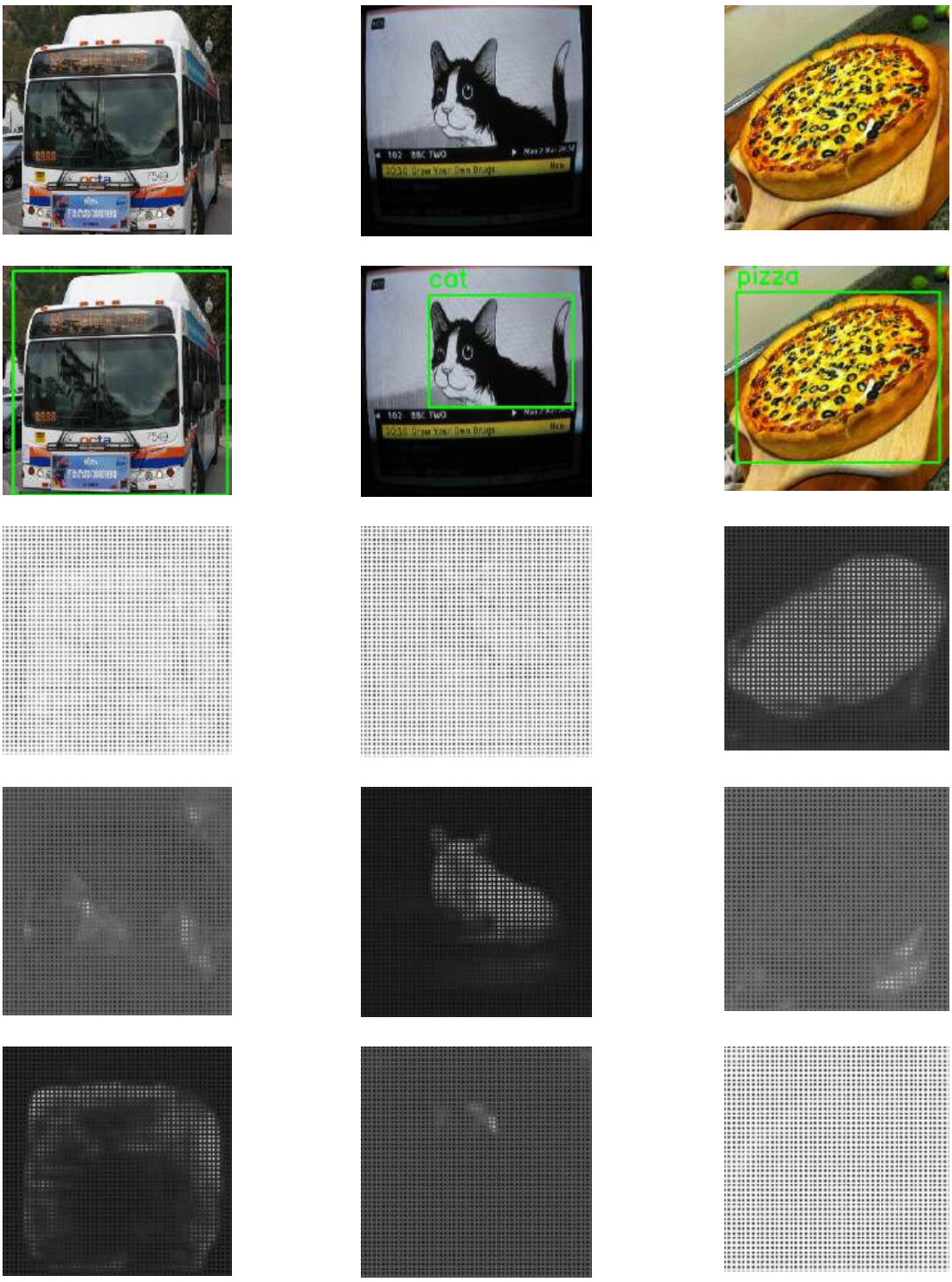
Batch 101

Test Results of MSCOCO

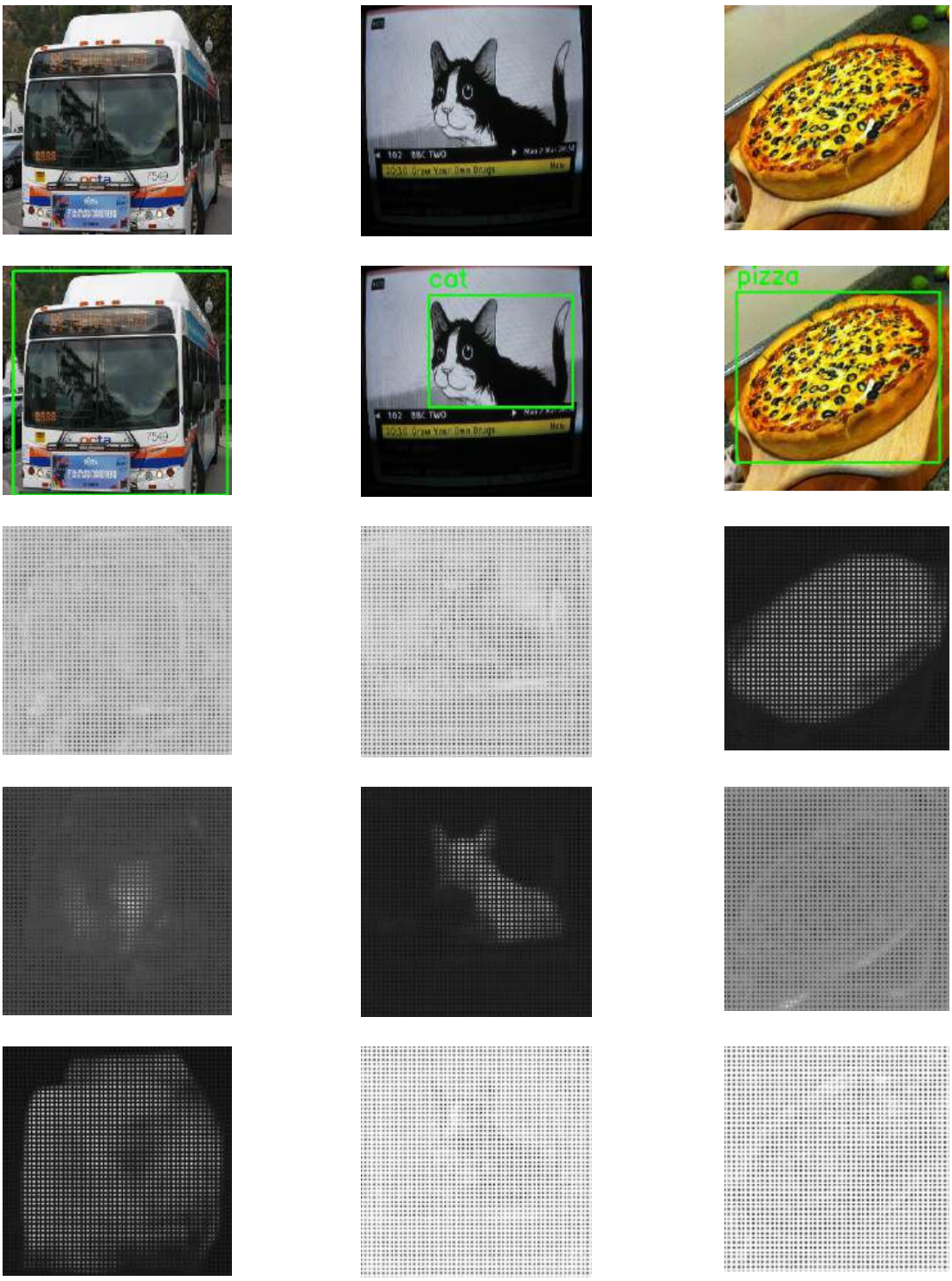
Output of learning rate 1e-4, epoch 20, batch 4



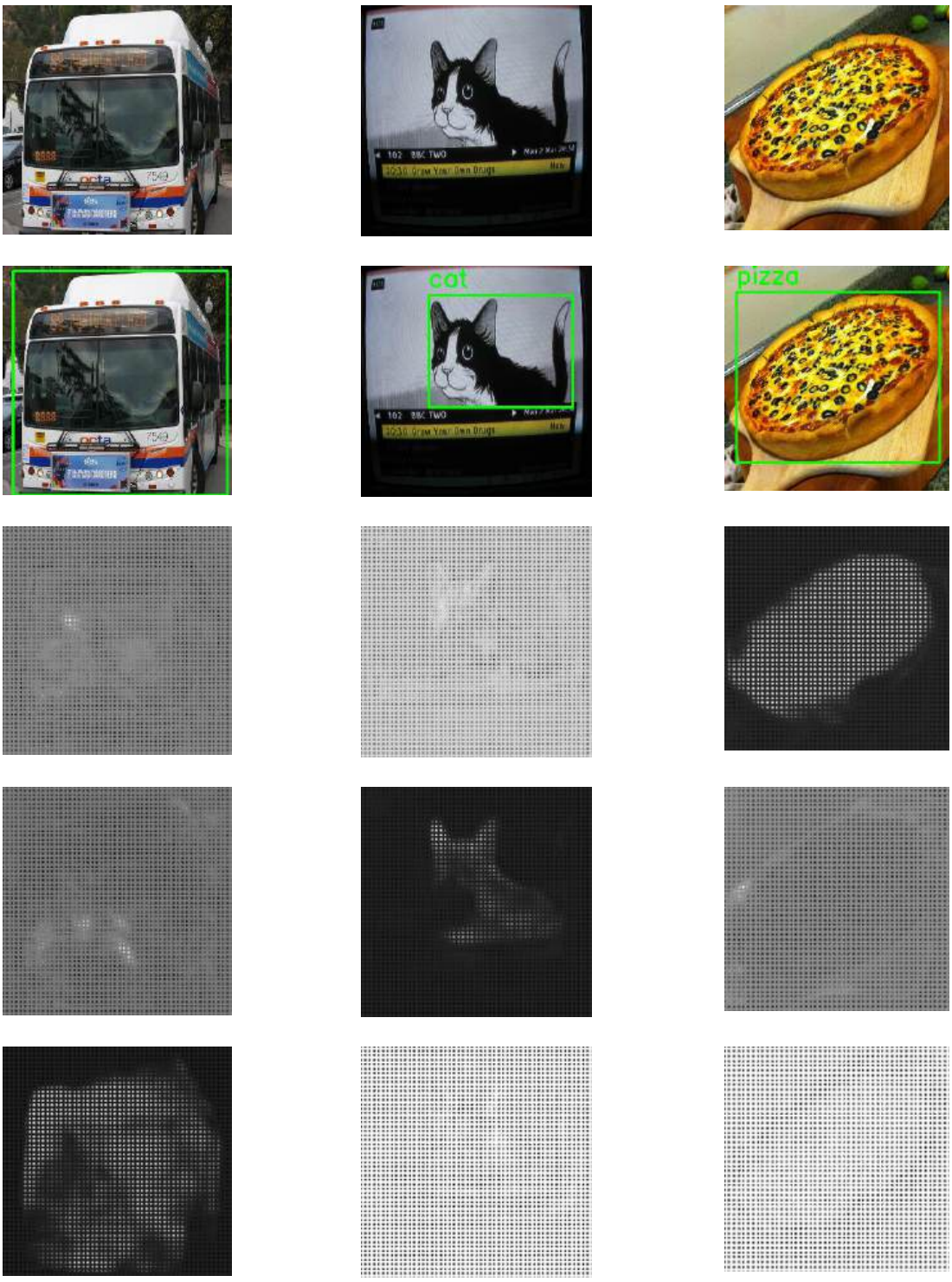
Output of learning rate 1e-4, epoch 30, batch 4



Output of learning rate 1e-5, epoch 20, batch 4



Output of learning rate 1e-5, epoch 30, batch 4



Best and Worst-case Training-loss vs. iterations for 4 cases

Case 1: MSE only

Best Case for PurdueShapes5MultiObjectDataset

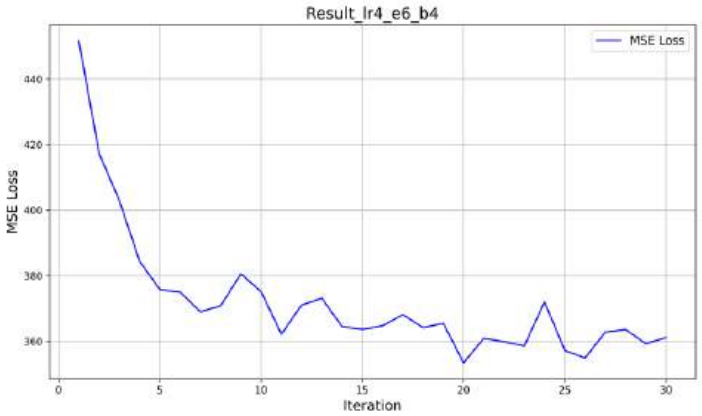
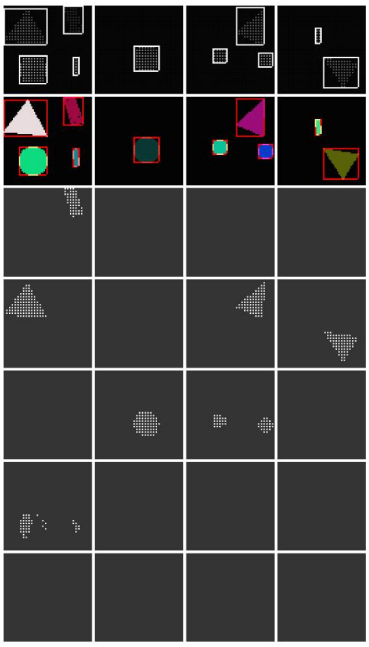
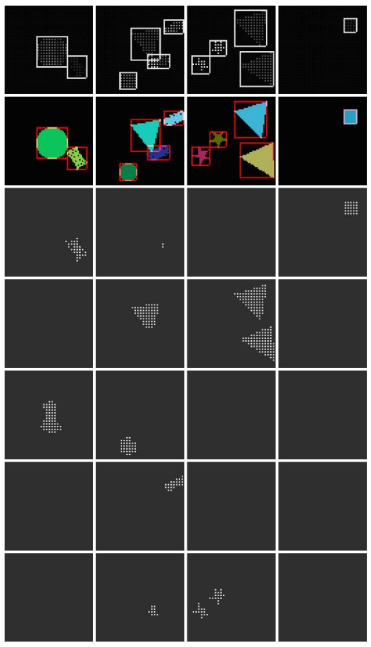


Fig: Loss curve for learning rate 1e-4, epoch 6, batch 4

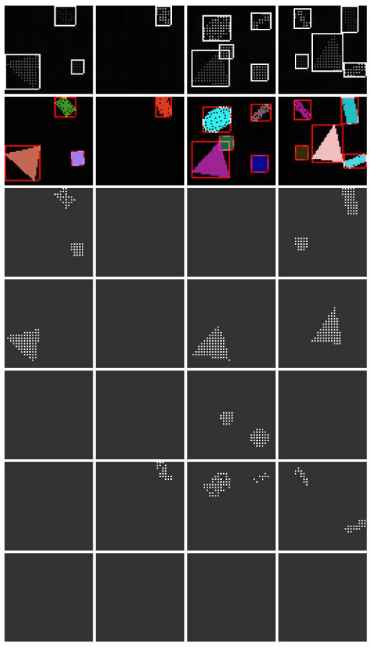
Test Output of learning rate 1e-4, epoch 6, batch 4



Batch 1



Batch 51



Batch 101

Best Case for MSCOCO

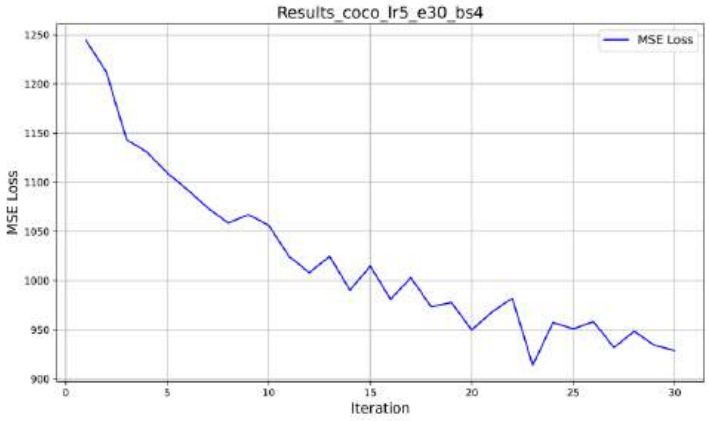
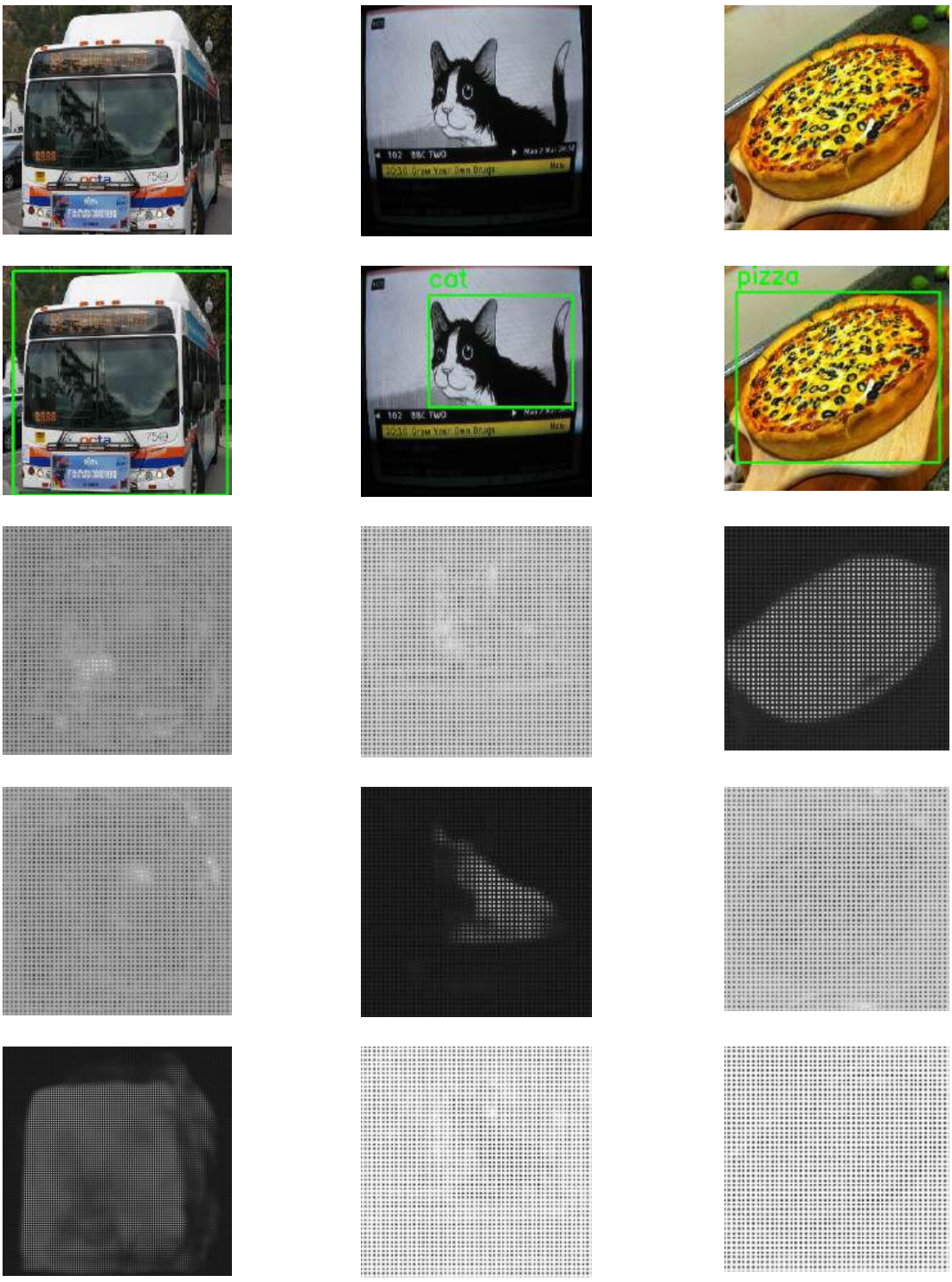


Fig: Loss curve for learning rate 1e-5, epoch 30, batch 4

Output of learning rate 1e-5, epoch 30, batch 4



Worst Case for PurdueShapes5MultiObjectDataset

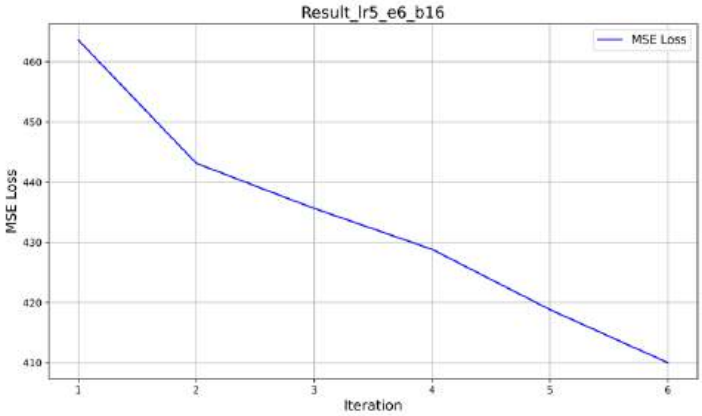
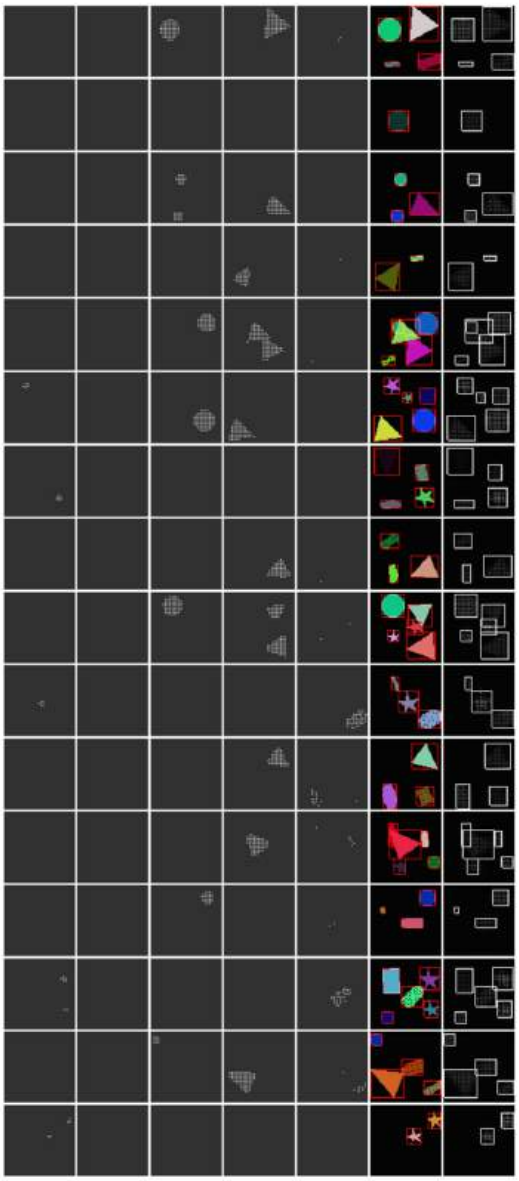


Fig: Loss curve for learning rate 1e-5, epoch 6, batch 16

Output of learning rate 1e-5, epoch 6, batch 16



Batch 1

Batch 51

Worst Case for MSCOCO

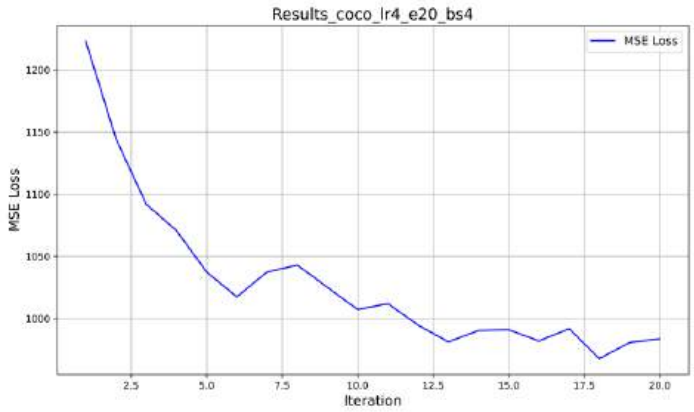
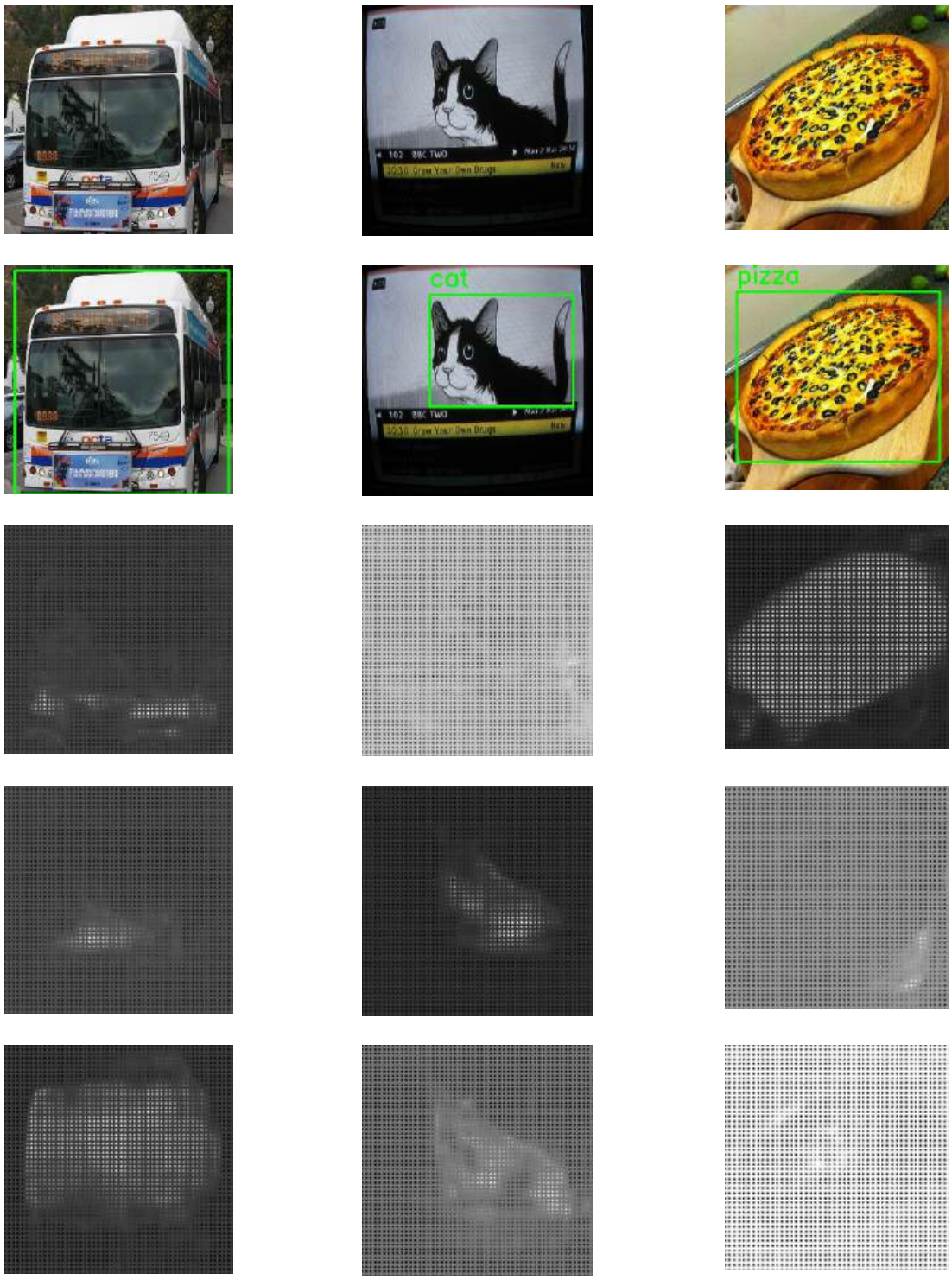


Fig: Loss curve for learning rate 1e-4, epoch 20, batch 4

Test Output of learning rate 1e-4, epoch 20, batch 4



Case 2: Dice Loss only

Best Case for PurdueShapes5MultiObjectDataset

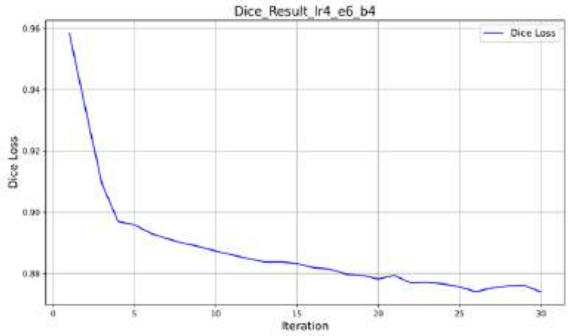
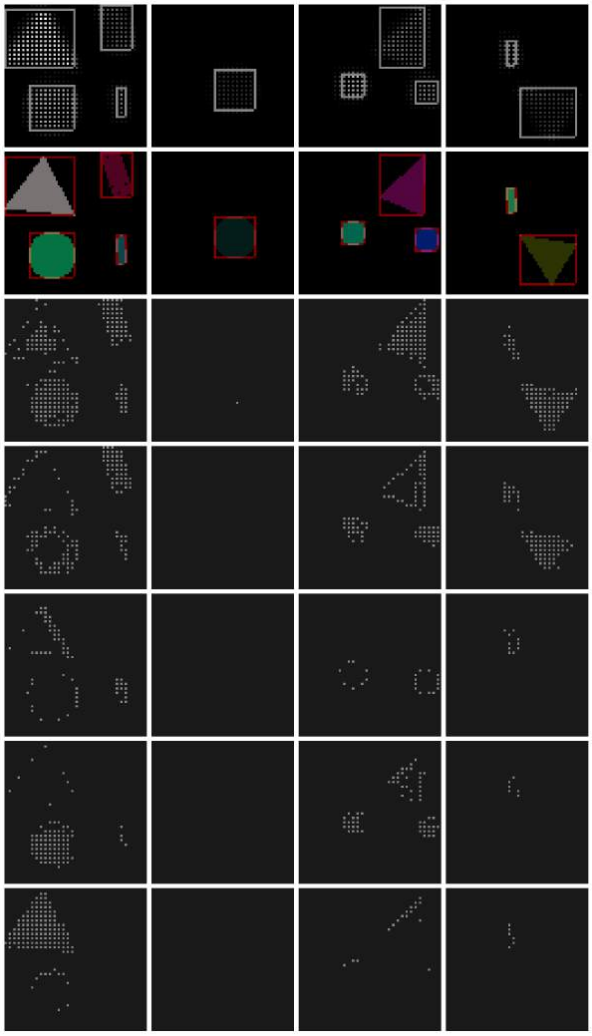


Fig: Loss curve for learning rate 1e-4, epoch 6, batch 4

Output of learning rate 1e-4, epoch 6, batch 4



Batch 1

Best Case for MSCOCO

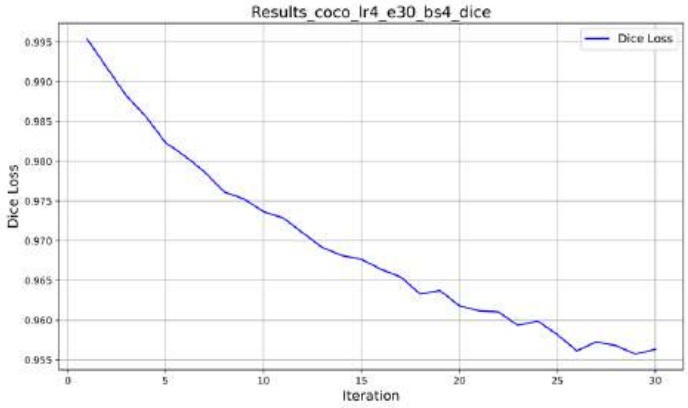
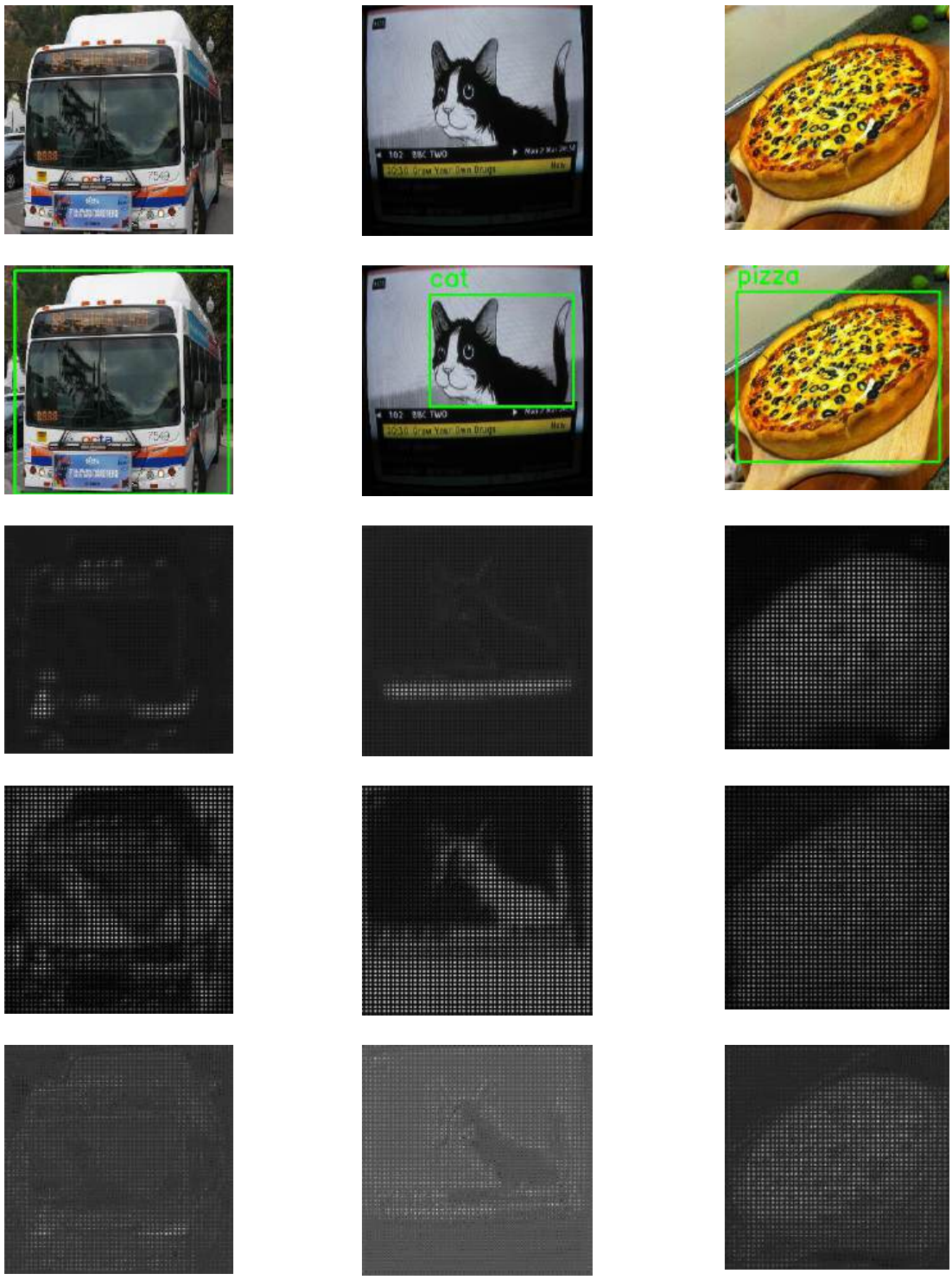


Fig: Loss curve for learning rate 1e-4, epoch 30, batch 4

Output of learning rate 1e-4, epoch 30, batch 4



Worst Case for PurdueShapes5MultiObjectDataset

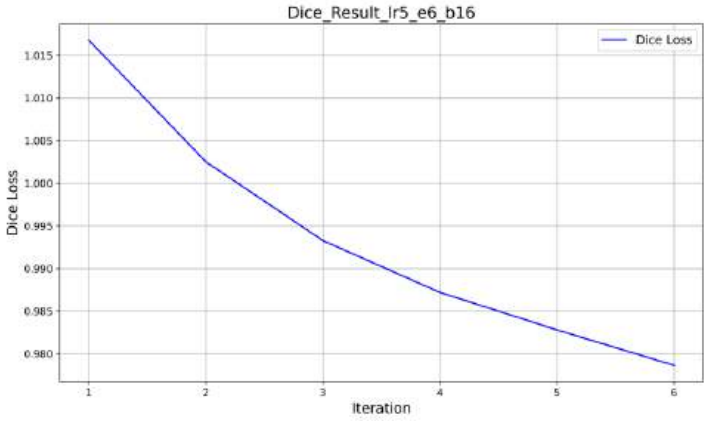
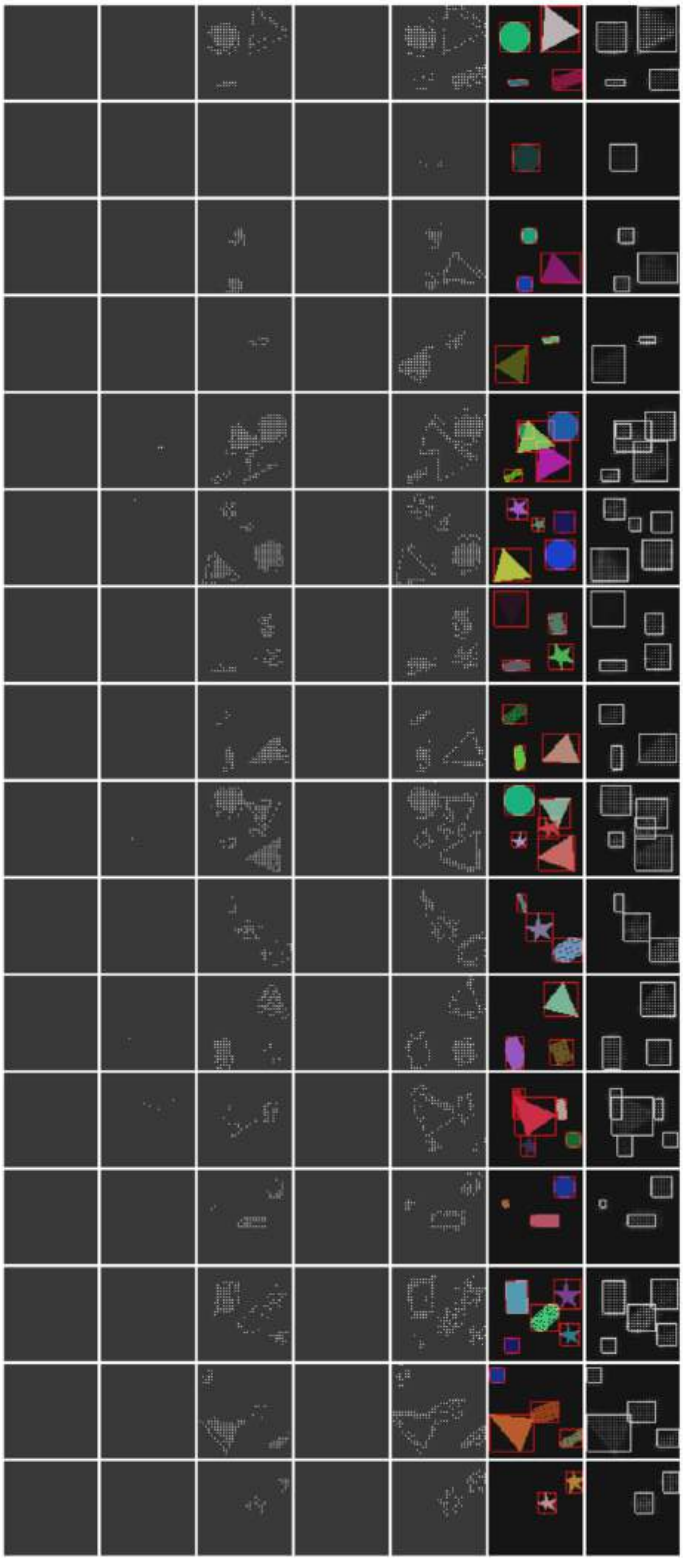


Fig: Loss curve for learning rate 1e-5, epoch 6, batch 16

Output of learning rate 1e-5, epoch 6, batch 16



Batch 1

Worst Case for MSCOCO

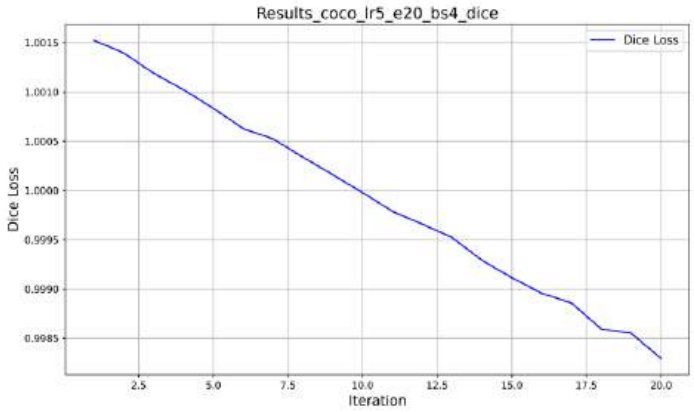
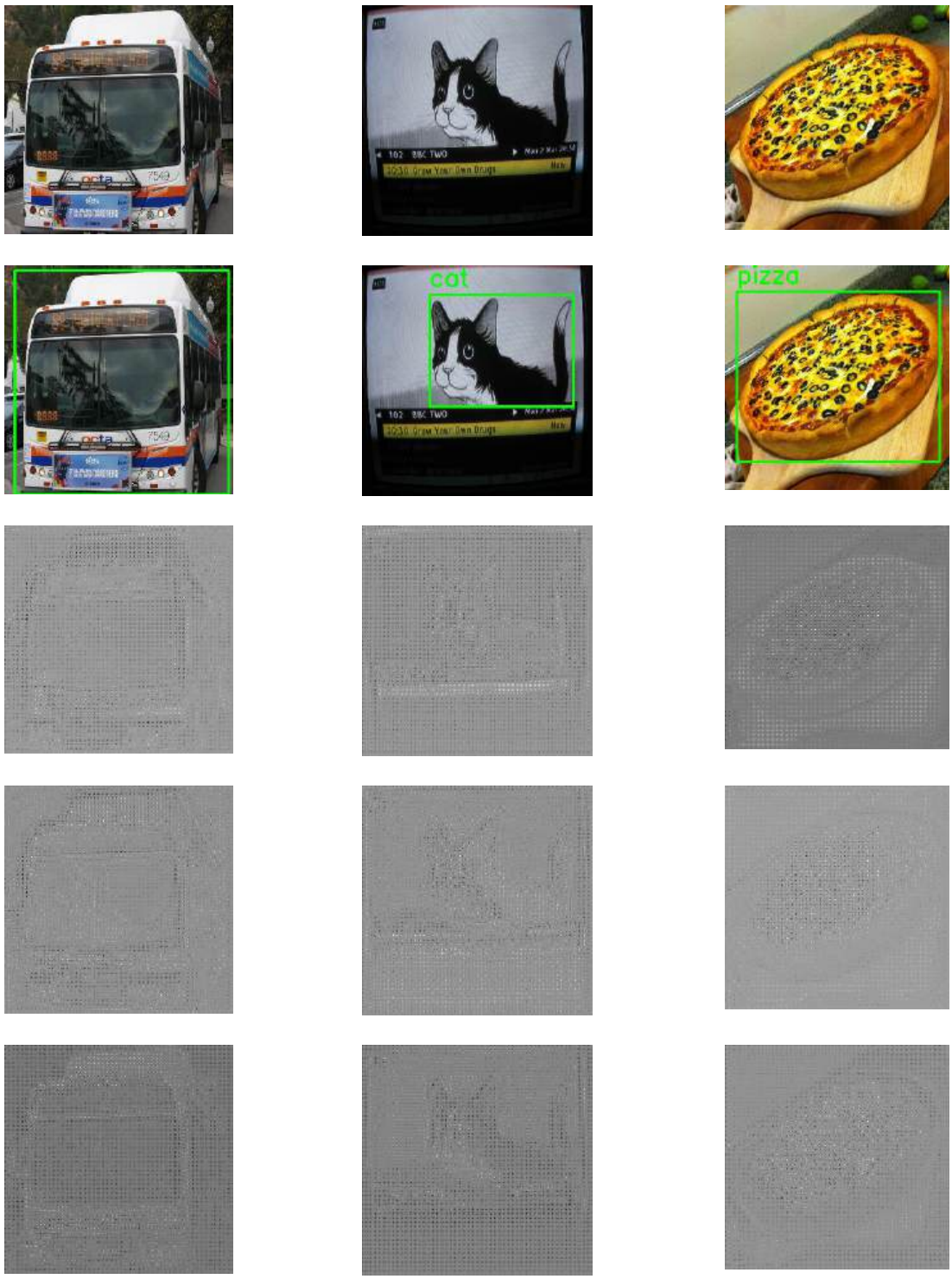


Fig: Loss curve for learning rate 1e-5, epoch 20, batch 4

Output of learning rate 1e-5, epoch 20, batch 4



Case 3: MSE+Dice with Scale of 1

Best Case for PurdueShapes5MultiObjectDataset

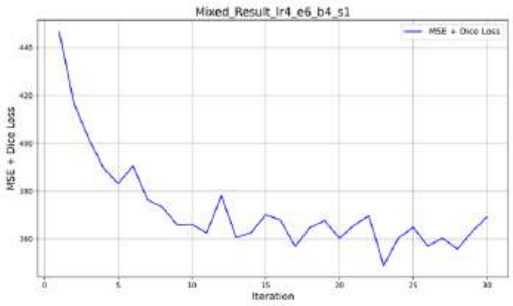
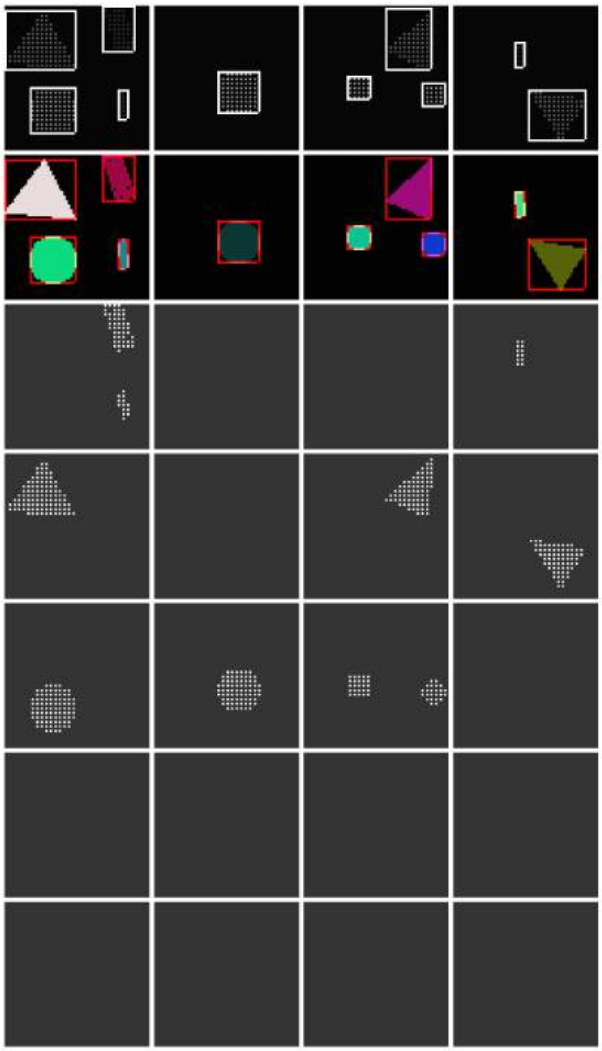


Fig: Loss curve for learning rate 1e-4, epoch 6, batch 4

Output of learning rate 1e-4, epoch 6, batch 4



Batch 1

Best Case for MSCOCO

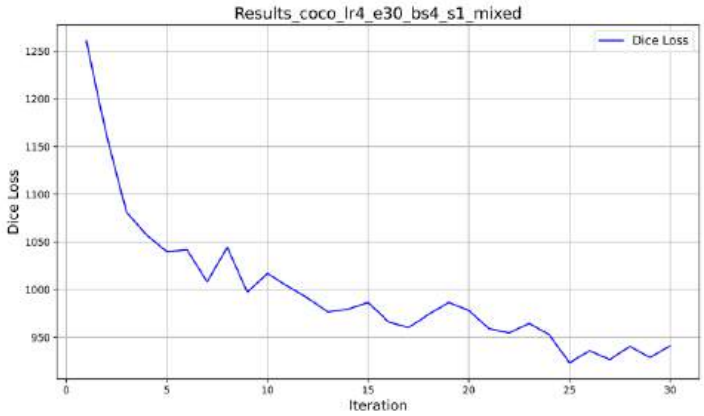
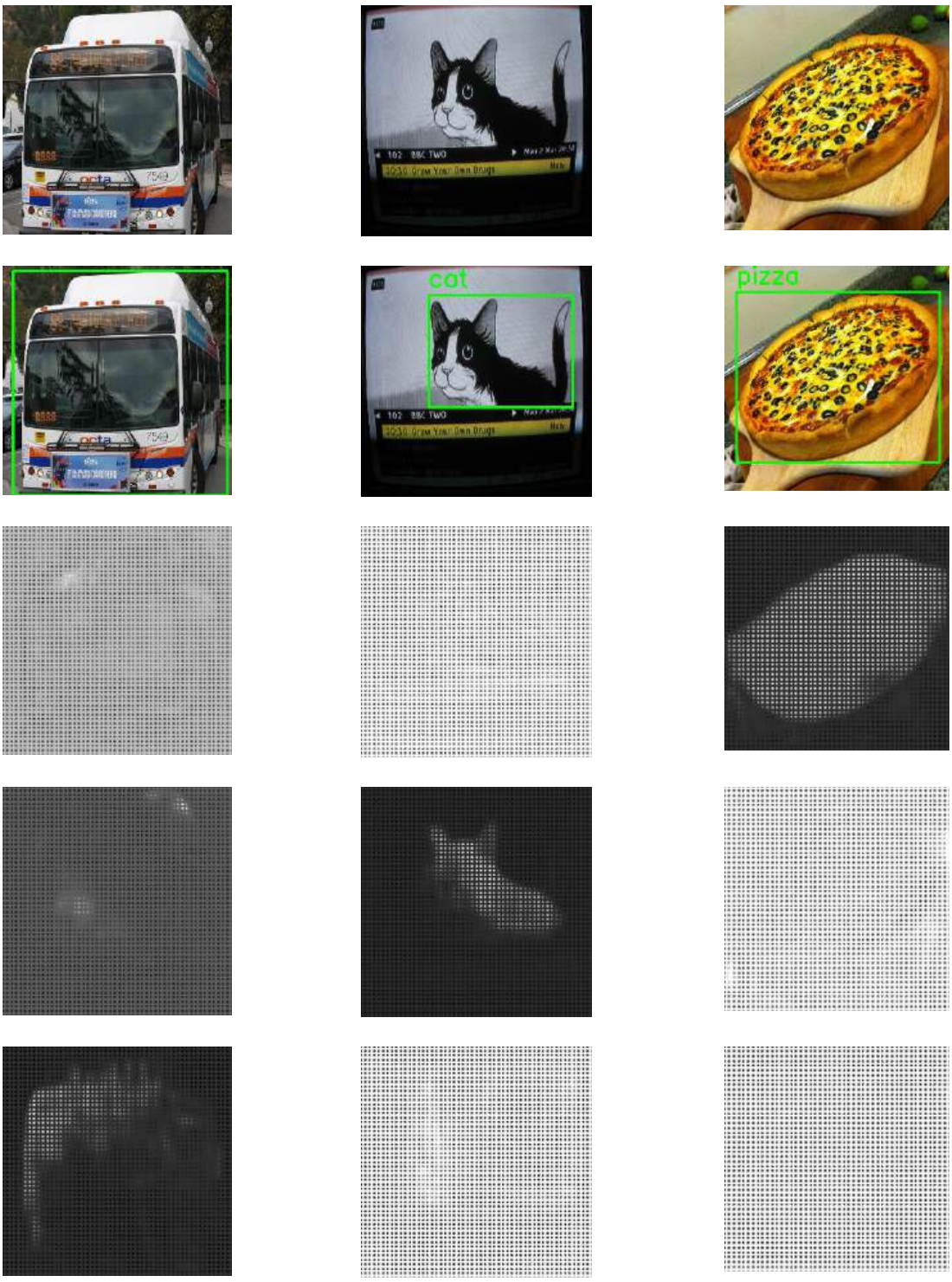


Fig: Loss curve for learning rate 1e-4, epoch 30, batch 4

Output of learning rate 1e-4, epoch 30, batch 4



Worst Case for PurdueShapes5MultiObjectDataset

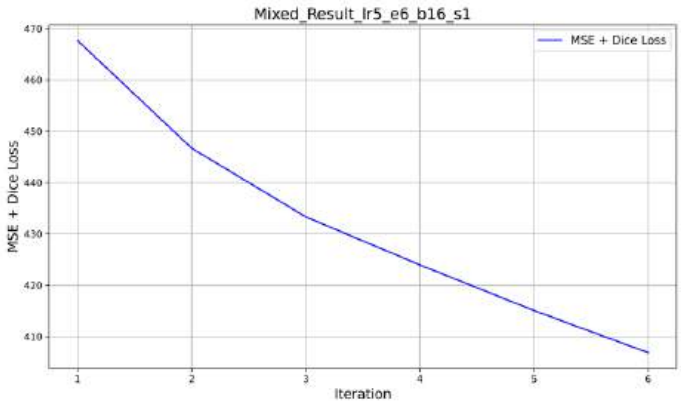
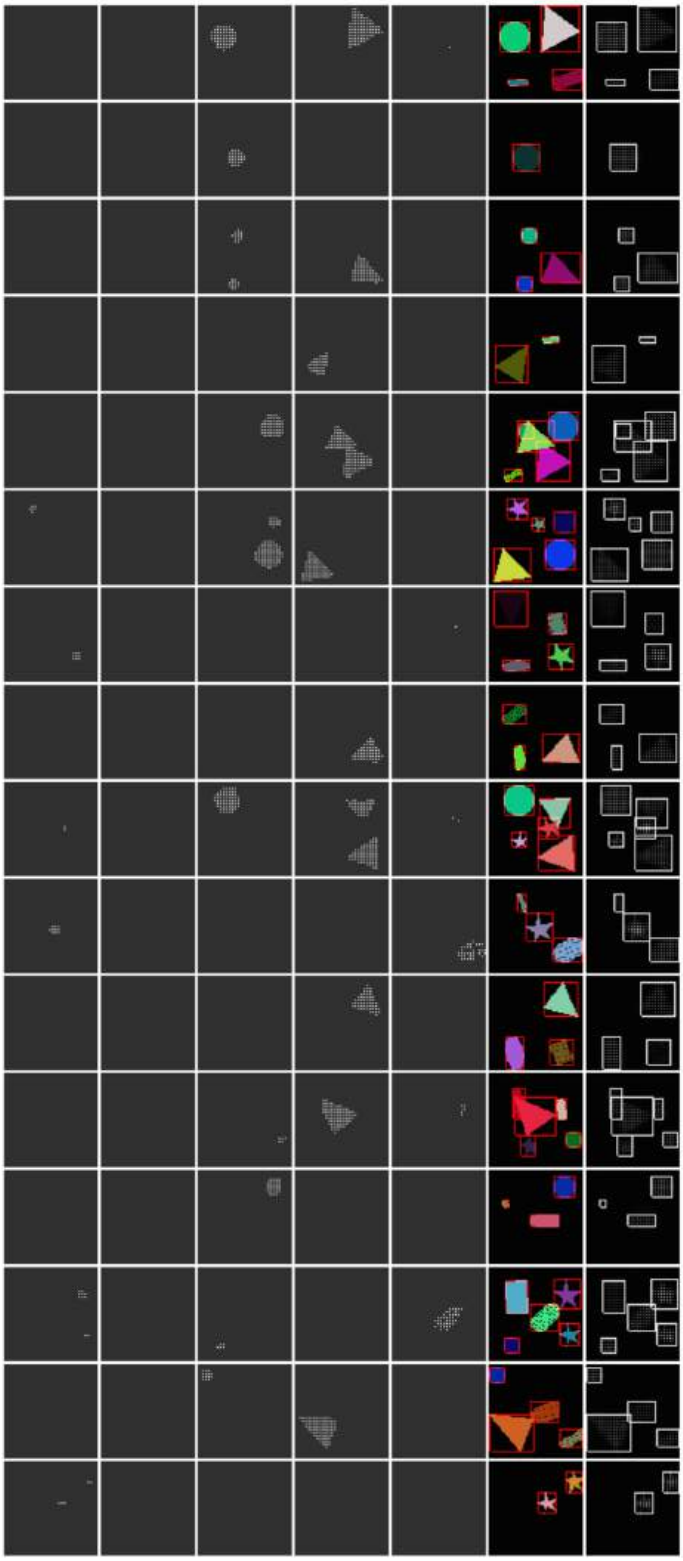


Fig: Loss curve for learning rate 1e-5, epoch 6, batch 16

Output of learning rate 1e-5, epoch 6, batch 16



Batch 1

Worst Case for MSCOCO

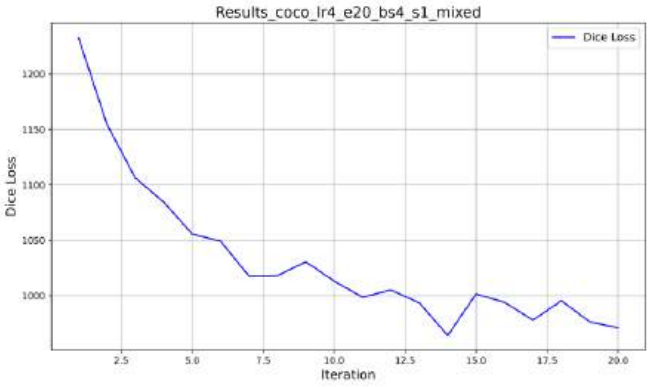
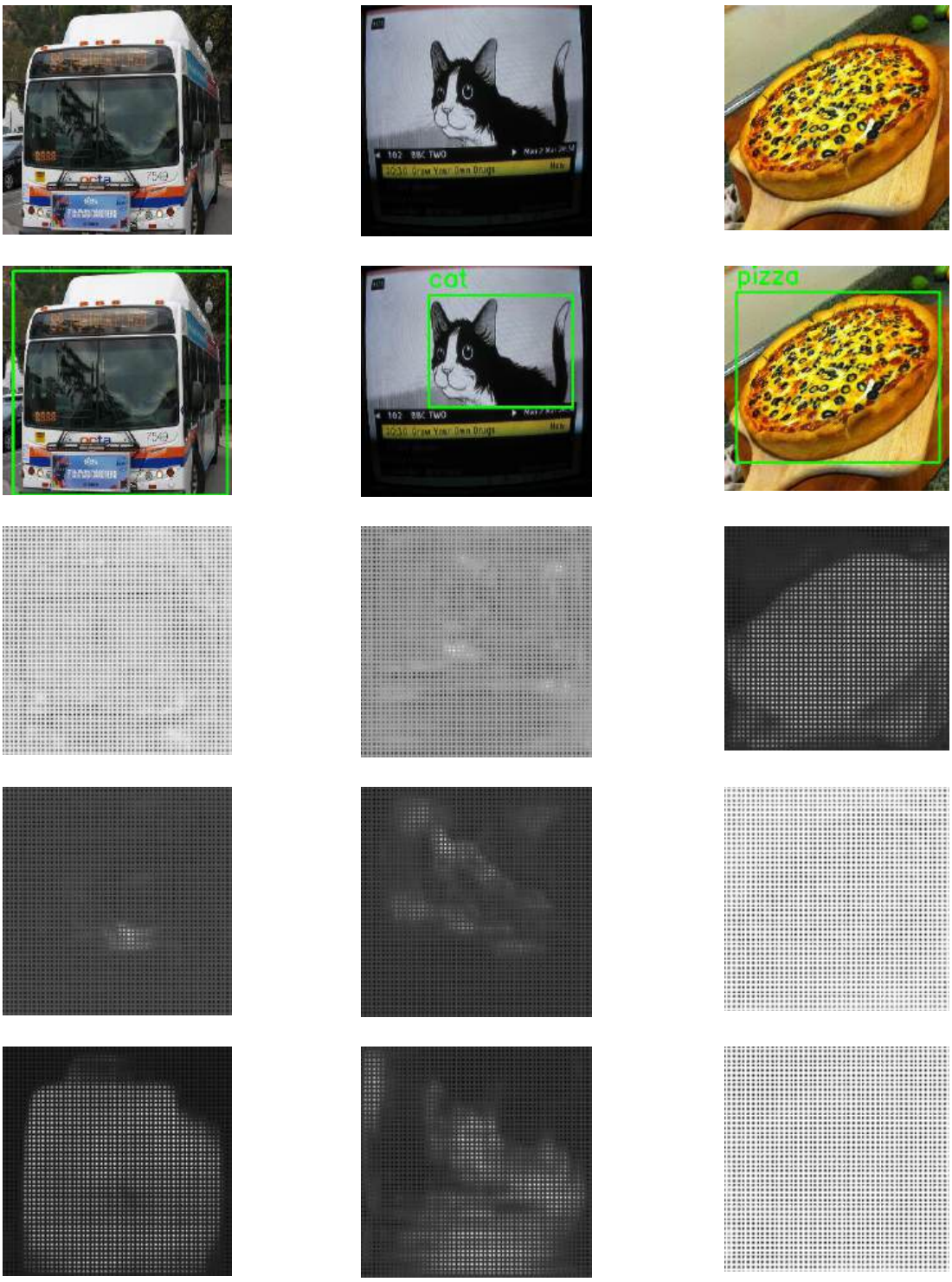


Fig: Loss curve for learning rate 1e-4, epoch 20, batch 4

Output of learning rate 1e-4, epoch 20, batch 4



Case 4: MSE+Dice with Scale of 100

Best Case for PurdueShapes5MultiObjectDataset

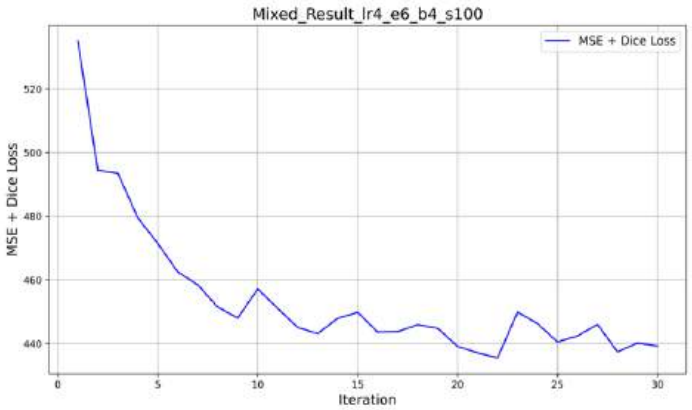
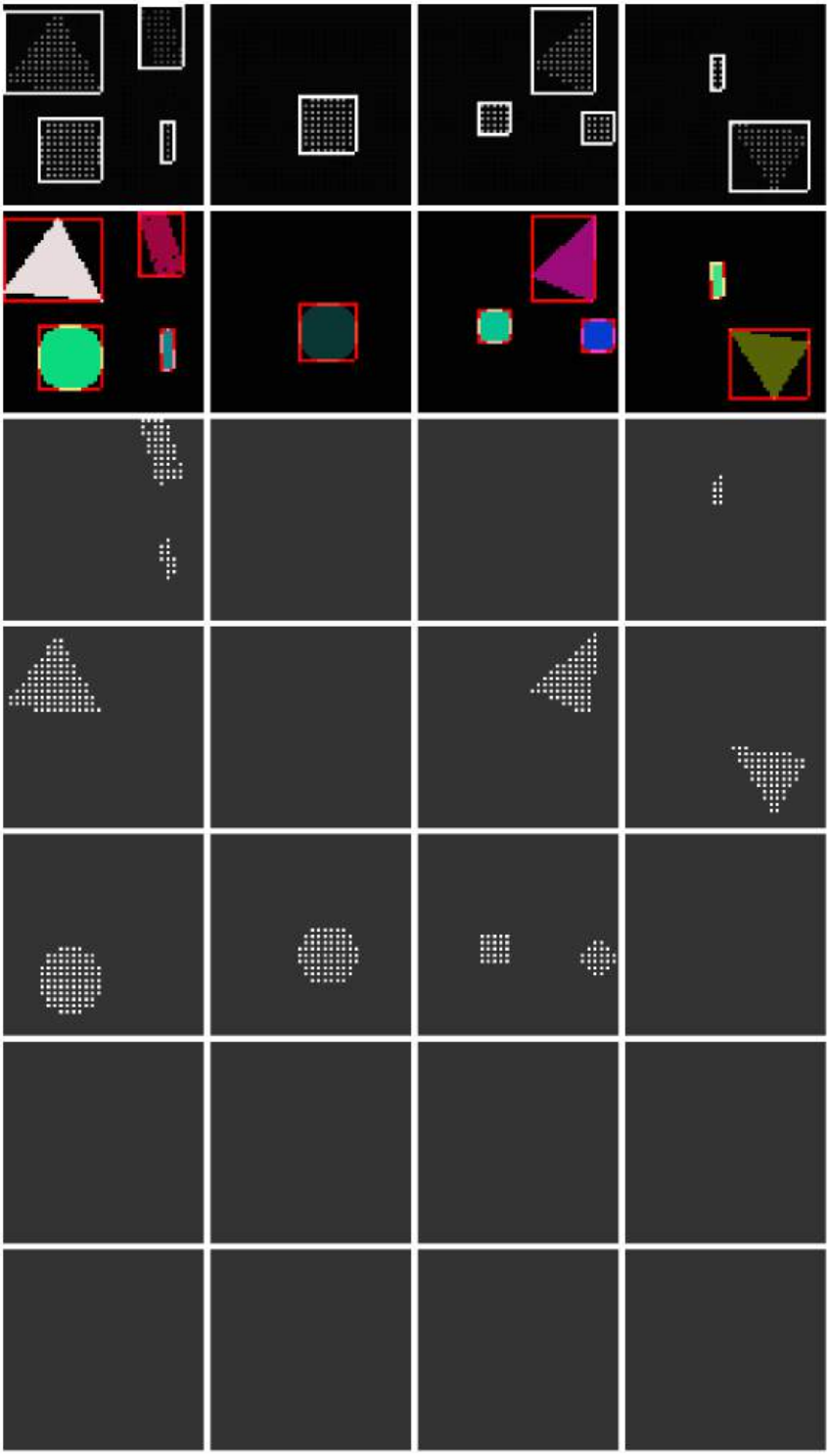


Fig: Loss curve for learning rate 1e-4, epoch 6, batch 4

Output of learning rate 1e-4, epoch 6, batch 4



Batch 1

Best Case for MSCOCO

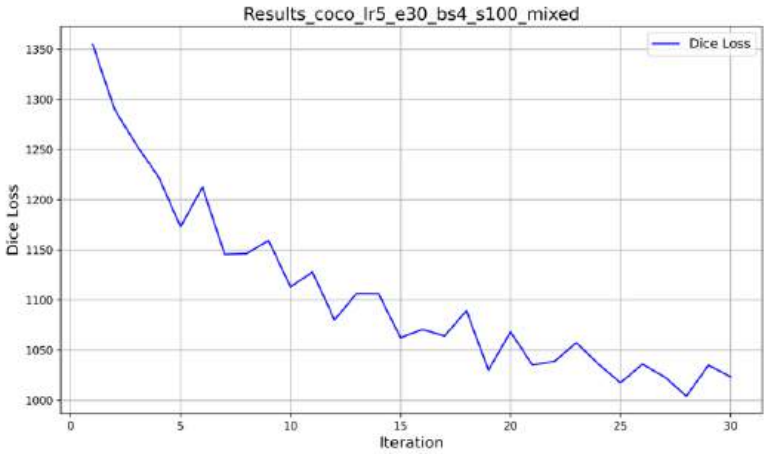
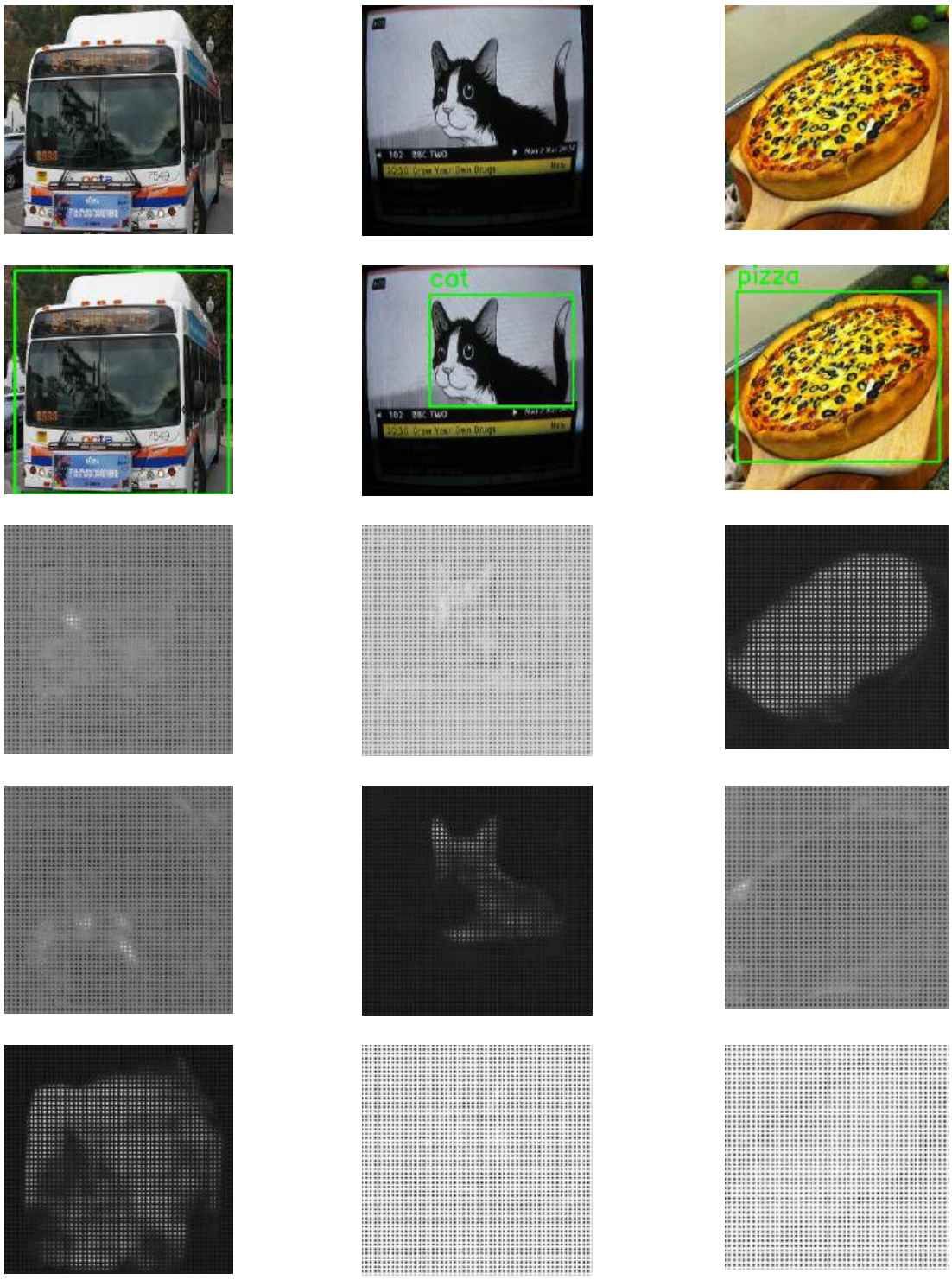


Fig: Loss curve for learning rate 1e-5, epoch 30, batch 4

Output of learning rate 1e-5, epoch 30, batch 4



Worst Case for PurdueShapes5MultiObjectDataset

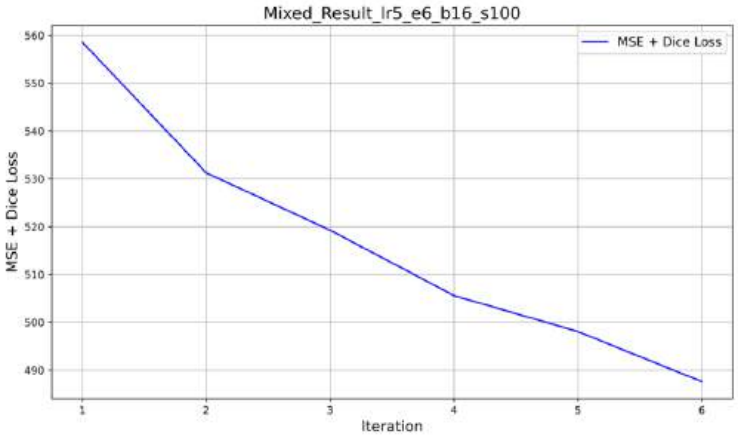
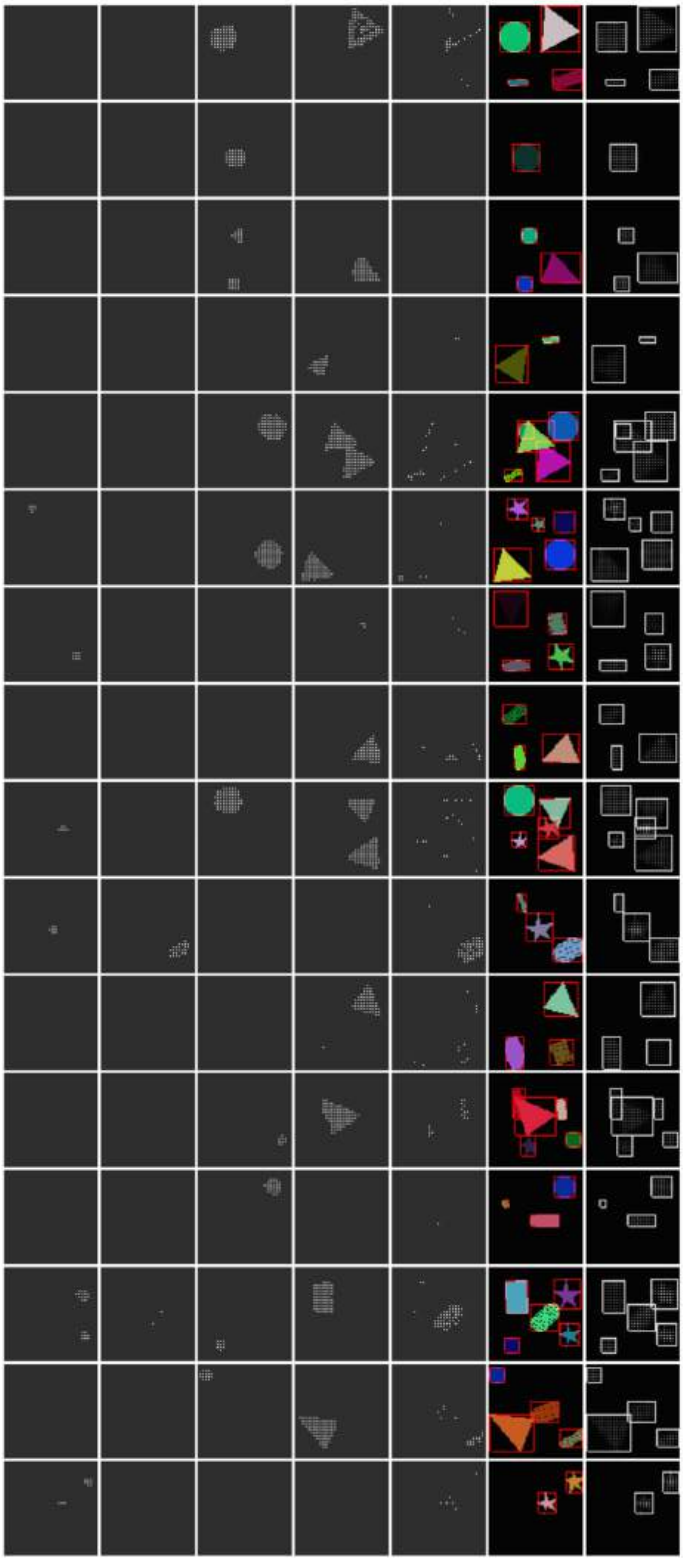


Fig: Loss curve for learning rate 1e-5, epoch 6, batch 16

Output of learning rate 1e-5, epoch 6, batch 16



Batch 1

Worst Case for MSCOCO

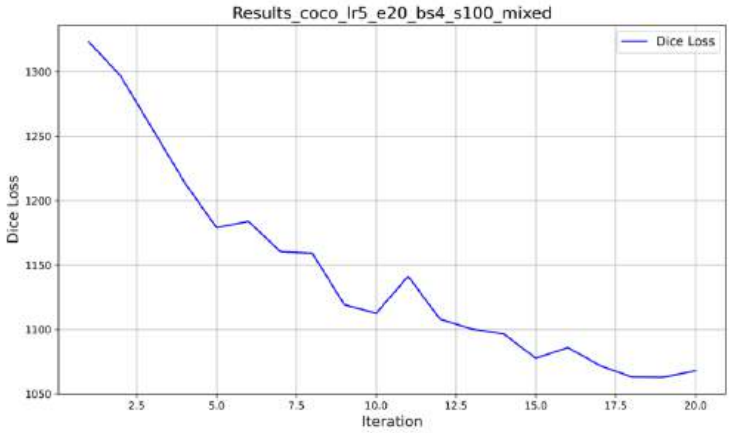
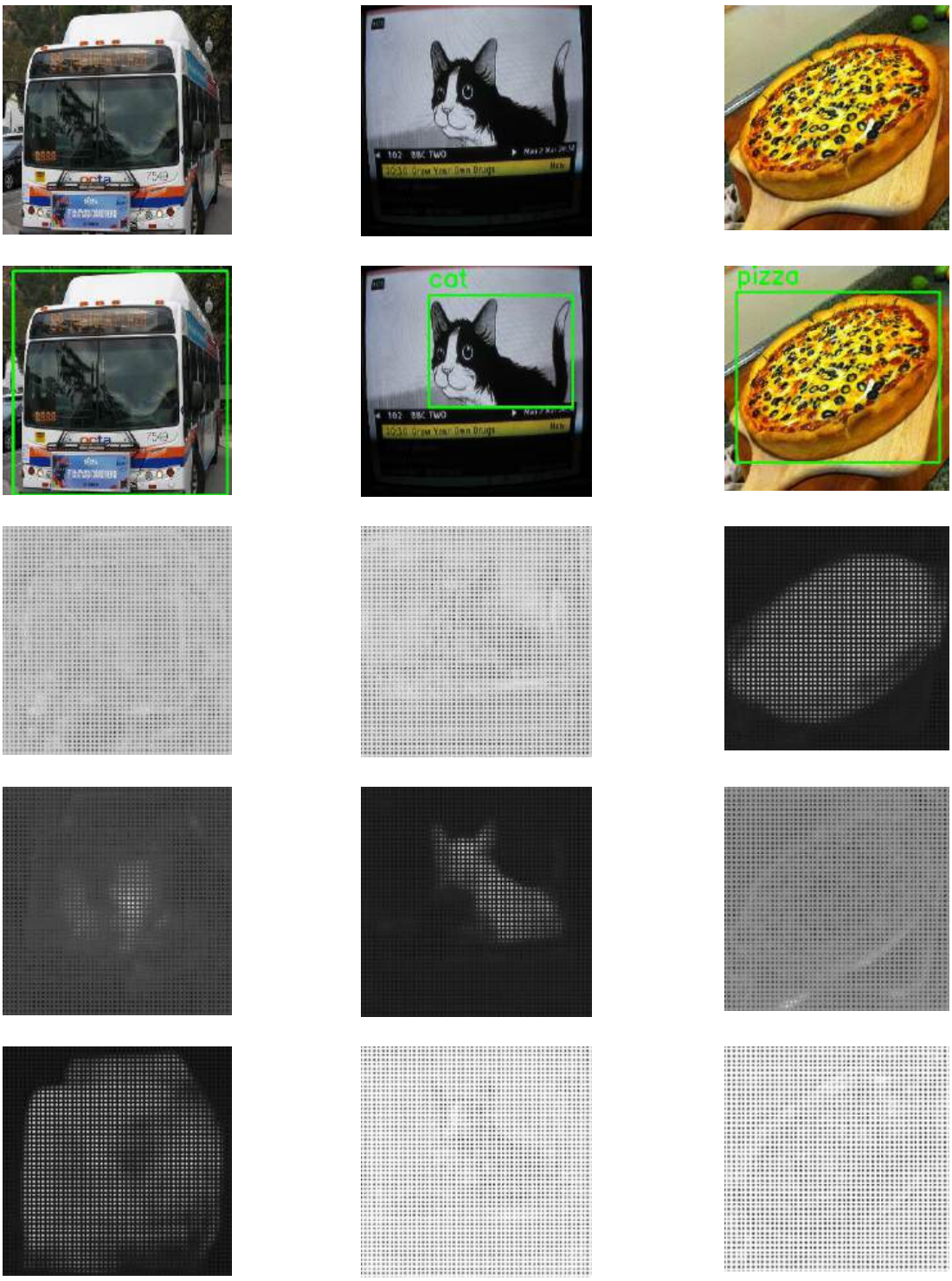


Fig: Loss curve for learning rate 1e-5, epoch 20, batch 4

Output of learning rate 1e-5, epoch 20, batch 4



Summary of Hyperparameters

Case	Dataset	Best/Worst	LR	Epoch	Batch
1 (MSE)	Purdue	Best	1e-4	6	4
	COCO	Best	1e-5	30	4
	Purdue	Worst	1e-5	6	16
	COCO	Worst	1e-4	20	4
2 (dice)	Purdue	Best	1e-4	6	4
	COCO	Best	1e-4	30	4
	Purdue	Worst	1e-5	6	16
	COCO	Worst	1e-5	20	4
3 (scale 1)	Purdue	Best	1e-4	6	4
	COCO	Best	1e-4	30	4
	Purdue	Worst	1e-5	6	16
	COCO	Worst	1e-4	20	4
4 (scale 100)	Purdue	Best	1e-4	6	4
	COCO	Best	1e-5	30	4
	Purdue	Worst	1e-5	6	16
	COCO	Worst	1e-5	20	4

Insights into potential factors contributing to the observed variations in performance.

The most potential factor is I think the epoch, the longer time I ran, the better result I got. Also, increasing the batch size was not a good intuition, it gave me the worst results.

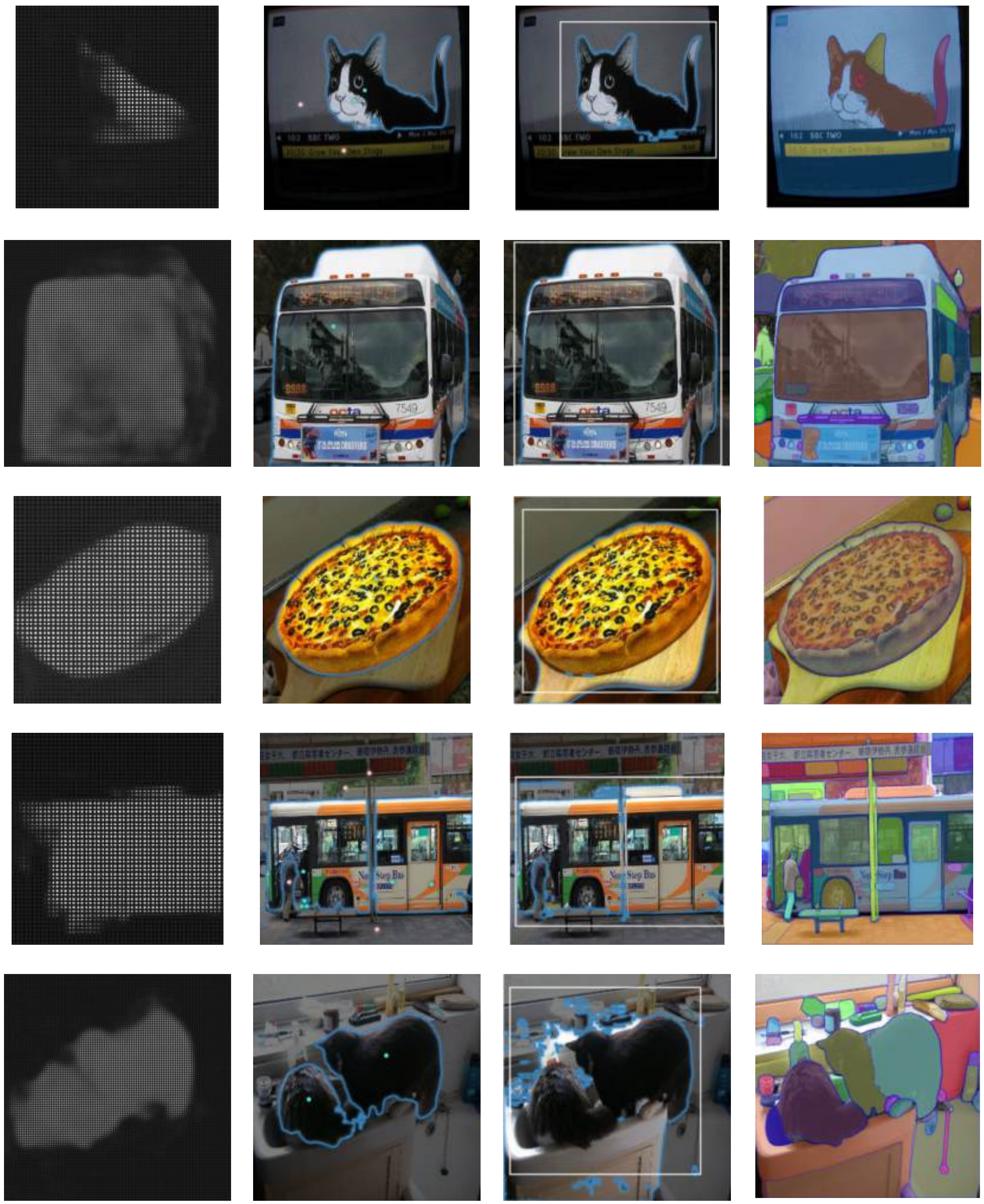
Qualitative observations on the model test results for MSE loss vs Dice loss vs. Dice+MSE loss

By the look of the test outputs, introducing dice loss and scaling it properly has caused the model to improve slightly. MSE loss alone is the significant loss function that improves the model, dice loss on the other hand does not generate good result.

Bonus

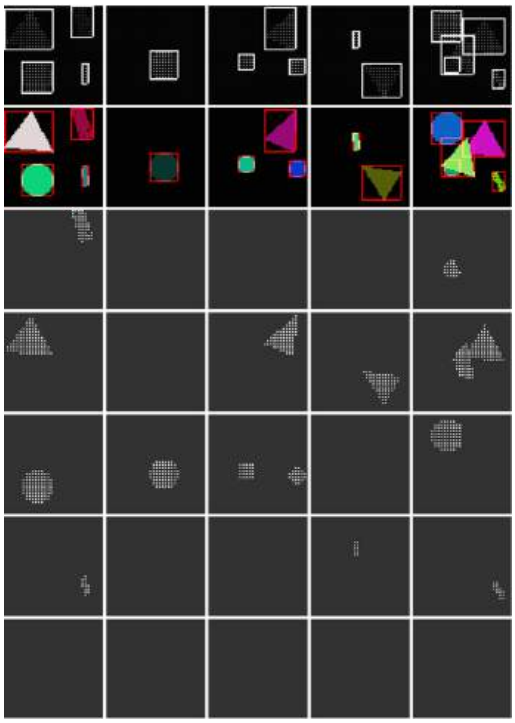
MSCOCO images side by side of SAM and mUNet

mUNet | Hover and click | Box | Everything

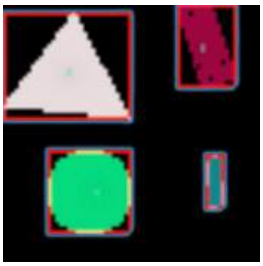
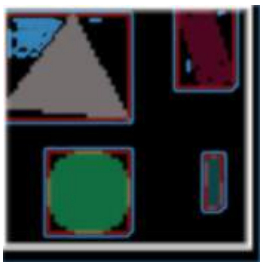
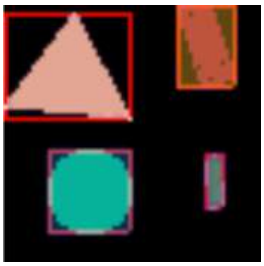
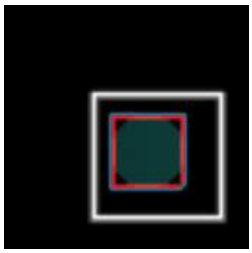
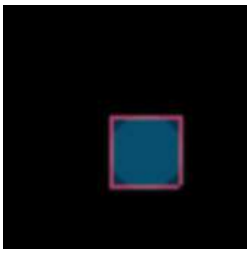


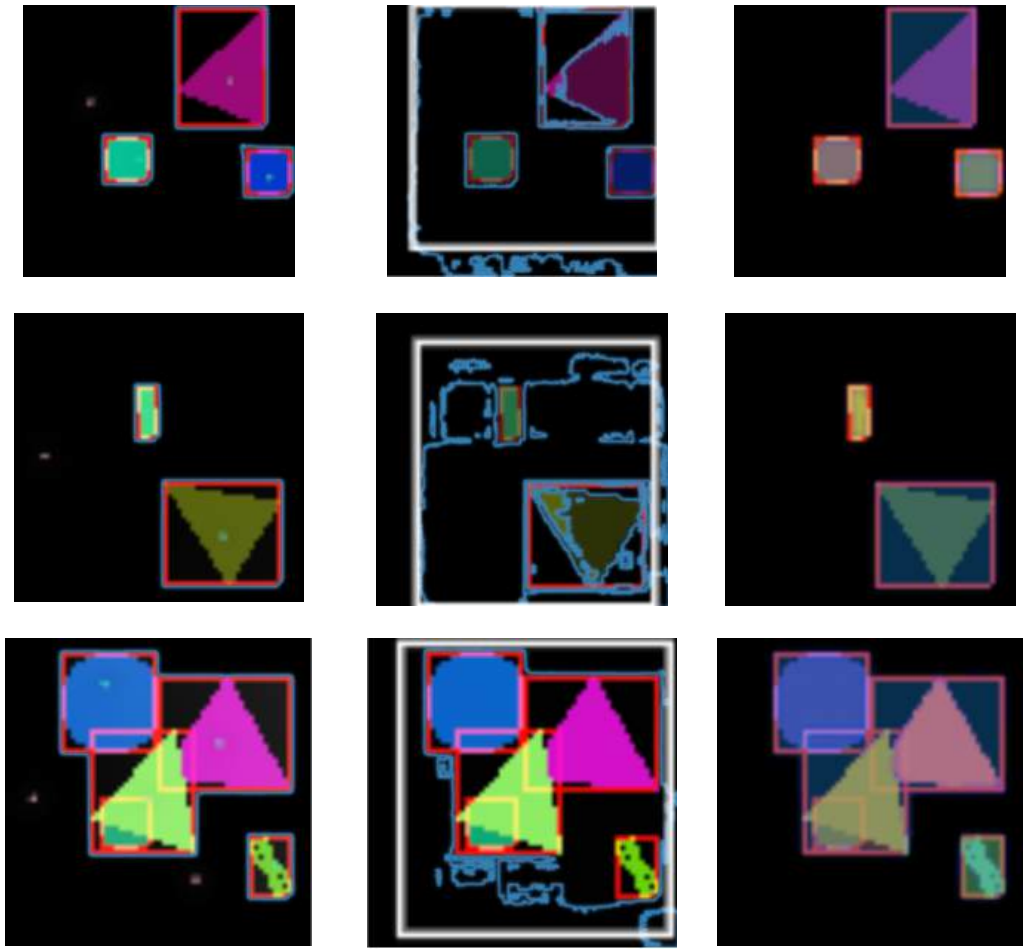
PurdueShapes5MultiObjectDataset images side by side of SAM and mUNet

This is the mUNet output:



This is the corresponding SAM output:

Hover and click	Box	Everything
		
		



Qualitative Observations (Edge Accuracy, Completeness, FP, FN)

If compared with Box method of SAM, I think my model is superior in the sense of edge accuracy, however SAM is superior in Completeness for most of the cases. Although the box method includes some unnecessary parts from the image which is especially prevalent in the PurdueShapes5MultiObjectDataset. Also, the everything method is unsuitable in this case as it tries to break the object into several segments (e.g. different parts of bus). But, for overall performance, the hover and click method of SAM is the best among my method and the three options of SAM.