Automated Detection and Analysis for Diagnosis in Cephalometric X-ray Image Using Convolutional Neural Network

June 30, 2020

T.C.KONYA TEKNİK ÜNİVERSİTESİLİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

Dental X-ray Image Analysis

Cephalometric Keypoints Detection using CNN Run

1 Automated Detection and Analysis for Diagnosis in Cephalometric X-ray Image Using Convolutional Neural Network

This project is build on top of the Wang Cwei dataset which can be found in the link: https://figshare.com/s/37ec464af8e81ae6ebbf

1.0.1 The scope of this work is to read the images and display the dots annotated by a professional medical doctor

[46]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
from skimage import io, transform, img_as_float
from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
import torchvision.datasets as dsets
from torchvision.transforms import ToTensor
from torchvision import datasets, transforms, models # add models to the list
from torch.utils.data import Dataset, DataLoader, TensorDataset
```

```
from torchvision import models
from torch.autograd import Variable
import torch.nn.functional as F
from torchvision.utils import make_grid
import time
import random

# ignore harmless warnings
import warnings
warnings.filterwarnings("ignore")
```

Lets perfom this operation on one image and sclale before performing a similar operation on multiple images

[91]:
```
# Take a look at one of the image samples and labels

#NOTE: THE IMAGE FOLDERS HAS BEEN MODIFIED AND SEPERATED INTO TRAIN AND TEST␣
 ↪FOLDERS SETS
SAMPLE_PATH = "data/RawImage/Train/TrainingData/005.bmp"
TXT_PATH = "data/AnnotationsByMD/400_senior/005.txt"
# import sample image
img = io.imread(SAMPLE_PATH, as_gray=True)
img
```

[91]:
```
array([[0.94509804, 0.94509804, 0.94509804, …, 0.94901961, 0.94117647,
        0.00392157],
       [0.94509804, 0.94509804, 0.94509804, …, 0.94901961, 0.94117647,
        0.00392157],
       [0.94509804, 0.94509804, 0.94509804, …, 0.94901961, 0.94117647,
        0.00392157],
       …,
       [0.94117647, 0.94117647, 0.94117647, …, 0.89803922, 0.89411765,
        0.        ],
       [0.94509804, 0.94509804, 0.94509804, …, 0.90588235, 0.90980392,
        0.        ],
       [0.        , 0.        , 0.        , …, 0.00784314, 0.        ,
        0.01176471]])
```

[92]:
```
img.shape
```

[92]:
```
(2400, 1935)
```

[93]:
```
# import sample coordinates from text as tuples
def extract_labels_from_txt(path):
    with open(path, "r") as f:
        # only first 19 are actual coords in dataset label files
        coords_raw = f.readlines()[:19]
```

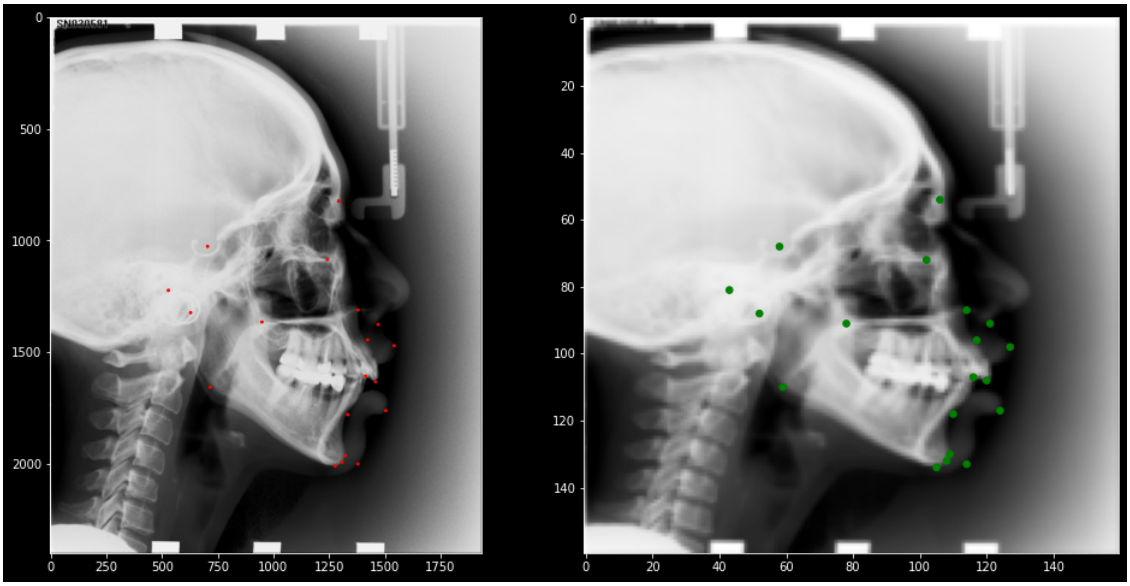
```
        coords_raw = [tuple([int(float(s)) for s in t.split(",")]) for t in␣
↪coords_raw]
        return coords_raw
```

[94]:
```
coords_raw = extract_labels_from_txt(TXT_PATH)
coords_raw
```

[94]:
```
[(705, 1026),
 (1294, 823),
 (1243, 1085),
 (529, 1223),
 (1423, 1445),
 (1334, 1780),
 (1324, 1964),
 (1278, 2010),
 (1309, 1993),
 (717, 1657),
 (1414, 1607),
 (1459, 1633),
 (1542, 1471),
 (1505, 1762),
 (1470, 1376),
 (1380, 2001),
 (949, 1365),
 (1380, 1311),
 (629, 1323)]
```

[95]:
```
plt.rcParams["figure.figsize"] = [32,18]
plt.style.use(['dark_background'])
fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 1, 1)
ax1.imshow(img, cmap="gray")
# also plot resized image for later
orig_y, orig_x = img.shape[:2]
SCALE = 15

# for rescale, use same target for both x&y axis
rescaled_img = transform.resize(img,(orig_y/SCALE,orig_y/SCALE))
ax2.imshow(rescaled_img, cmap="gray")

for c in coords_raw:
    # add patches to original image
    # could also just plt.scatter() but less control then
    ax1.add_patch(plt.Circle(c, 5, color='r'))
    # and rescaled marks to resized images
    x,y = c
```

```
    x = int(x*(orig_y*1.0/orig_x)/SCALE)
    y = int(y/SCALE)
    ax2.add_patch(plt.Circle((x,y), 1, color='g'))

plt.show()
```

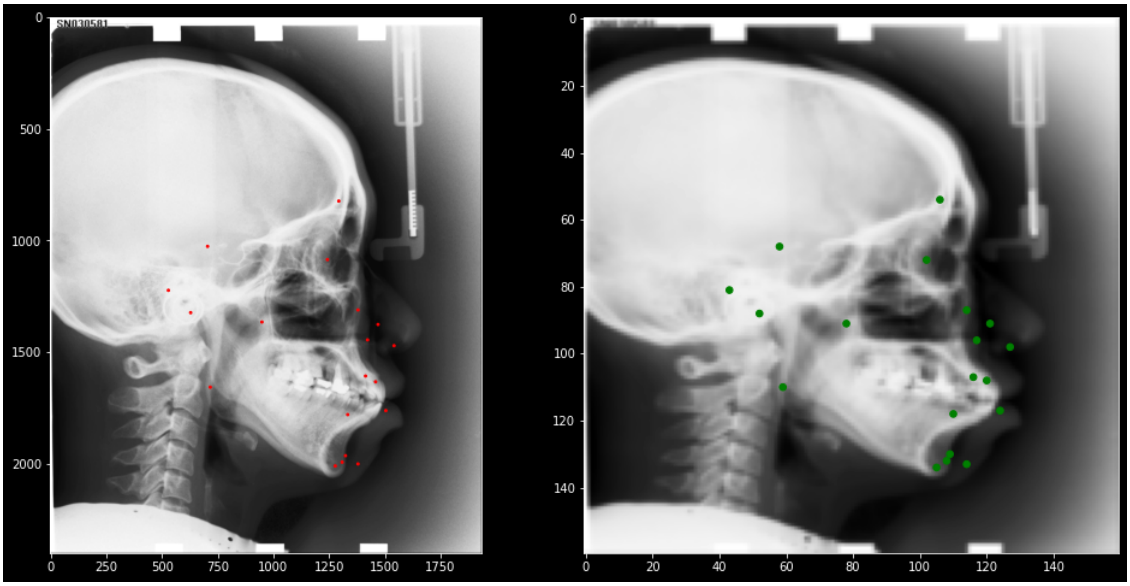


[96]:
```
def print_image(img,labels):
    print(img.shape)
    plt.rcParams["figure.figsize"] = [32,18]
    fig = plt.figure()
    ax1 = fig.add_subplot(2, 2, 1)
    ax2 = fig.add_subplot(2, 1, 1)
    ax1.imshow(img, cmap="gray")
    # also plot resized image for later
    orig_y, orig_x = img.shape[:2]
    SCALE = 15

    # for rescale, use same target for both x&y axis
    rescaled_img = transform.resize(img,(orig_y/SCALE,orig_y/SCALE))
    ax2.imshow(rescaled_img, cmap="gray")


    for c in coords_raw:
        # add patches to original image
        # could also just plt.scatter() but less control then
        ax1.add_patch(plt.Circle(c, 5, color='r'))
        # and rescaled marks to resized images
        x,y = c
```

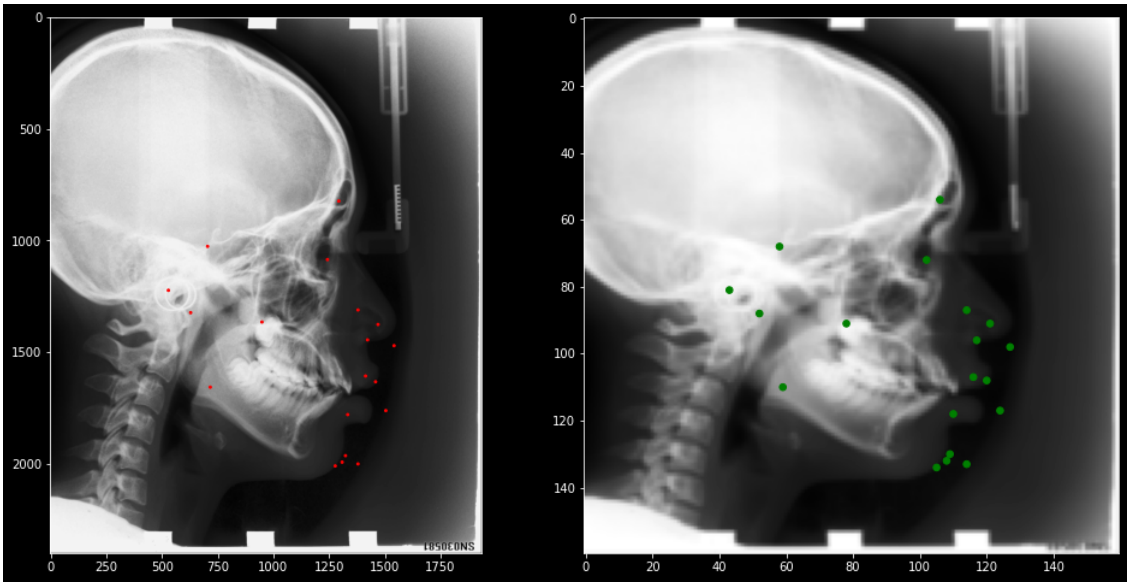
```
        x = int(x*(orig_y*1.0/orig_x)/SCALE)
        y = int(y/SCALE)
        ax2.add_patch(plt.Circle((x,y), 1, color='g'))

    plt.show()
```

[98]:
```
BASE_IMAGE_PATH='data/RawImage/Train/TrainingData/'
BASE_CORD_PATH='data/AnnotationsByMD/400_senior/'

def display_image_and_cord(image_number,img_path, cord_path):
    data = []
    target = []
    for i, fi in enumerate(os.listdir(img_path)):
           if i<image_number:
                loop_img = io.imread(img_path + fi, as_gray=True)
                lf = fi[:-4] + ".txt"
                loop_labels = extract_labels_from_txt(cord_path + lf)

                loop_labels = (np.array(loop_labels))
                print(loop_img)
                print_image(loop_img,loop_labels)



display_image_and_cord(10,BASE_IMAGE_PATH, BASE_CORD_PATH)
```

```
[[1.         1.         1.         … 0.99607843 1.         0.        ]
 [1.         1.         1.         … 0.99607843 1.         0.        ]
 [1.         1.         1.         … 0.99607843 1.         0.        ]
 …
 [1.         1.         1.         … 0.87058824 0.88235294 0.        ]
 [1.         1.         1.         … 0.89411765 0.87058824 0.        ]
 [0.         0.         0.         … 0.         0.01176471 0.        ]]
(2400, 1935)
```

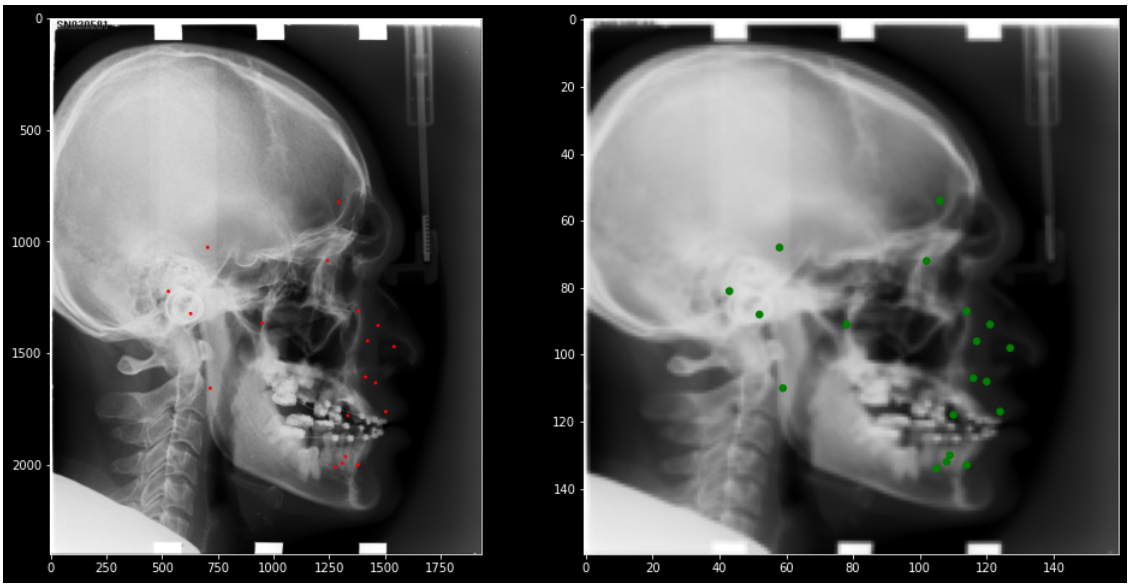


```
[[0.06666667 0.06666667 0.0627451  … 0.95686275 0.95686275 0.        ]
 [0.0627451  0.0627451  0.05882353 … 0.95686275 0.95686275 0.        ]
 [0.05490196 0.05490196 0.05098039 … 0.95686275 0.95686275 0.        ]
 …
 [0.95686275 0.95686275 0.95686275 … 0.94509804 0.95294118 0.        ]
 [0.95686275 0.95686275 0.95686275 … 0.96078431 0.96078431 0.00392157]
 [0.         0.         0.         … 0.01176471 0.         0.00784314]]
(2400, 1935)
```

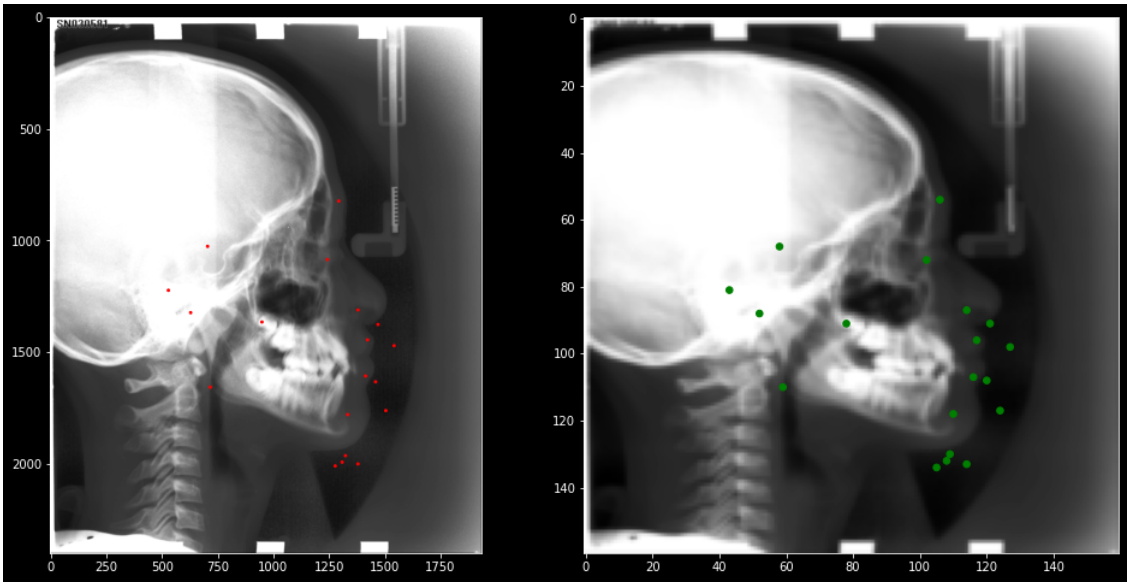


```
[[0.99607843 0.99607843 0.99607843 … 0.99607843 1.         0.        ]
```

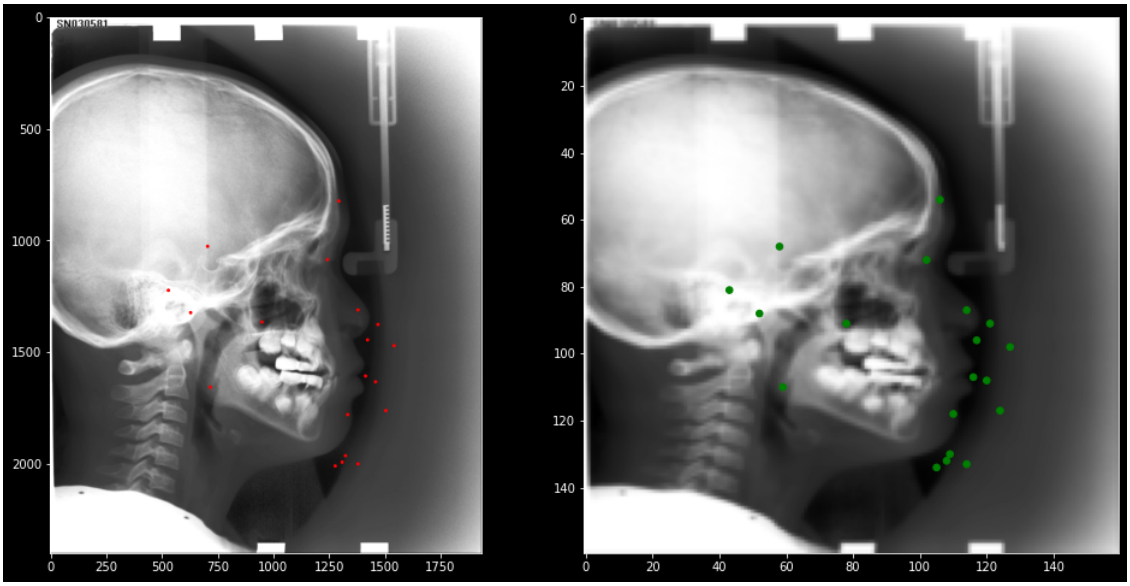
```
 [0.99607843 0.99607843 0.99607843 … 0.99607843 1.         0.        ]
 [0.99607843 0.99607843 0.99215686 … 0.99607843 1.         0.        ]
 …
 [1.         1.         1.         … 0.34509804 0.41960784 0.        ]
 [1.         1.         1.         … 0.39215686 0.44705882 0.01176471]
 [0.         0.         0.         … 0.01568627 0.         0.        ]]
(2400, 1935)
```




```
[[1.         1.         1.         … 0.99607843 1.         0.        ]
 [1.         1.         1.         … 0.99607843 1.         0.        ]
 [1.         1.         1.         … 0.99607843 1.         0.        ]
 …
 [1.         1.         1.         … 0.96078431 0.96078431 0.        ]
 [1.         1.         1.         … 0.98823529 0.98823529 0.01568627]
 [0.         0.         0.         … 0.00784314 0.         0.00392157]]
(2400, 1935)
```

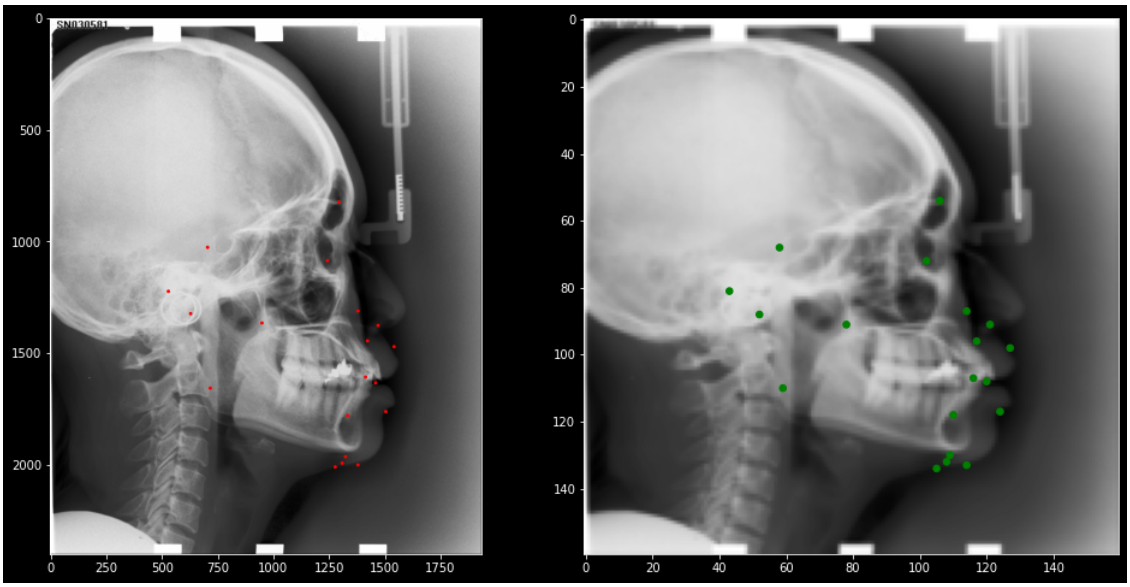


```
[[1.         1.         1.         … 0.99607843 1.         0.        ]
 [1.         1.         1.         … 0.99607843 1.         0.        ]
 [1.         1.         1.         … 0.99607843 1.         0.        ]
 …
 [1.         1.         1.         … 0.97647059 1.         0.01960784]
 [1.         1.         1.         … 1.         0.97254902 0.        ]
 [0.         0.         0.         … 0.         0.02352941 0.        ]]
(2400, 1935)
```

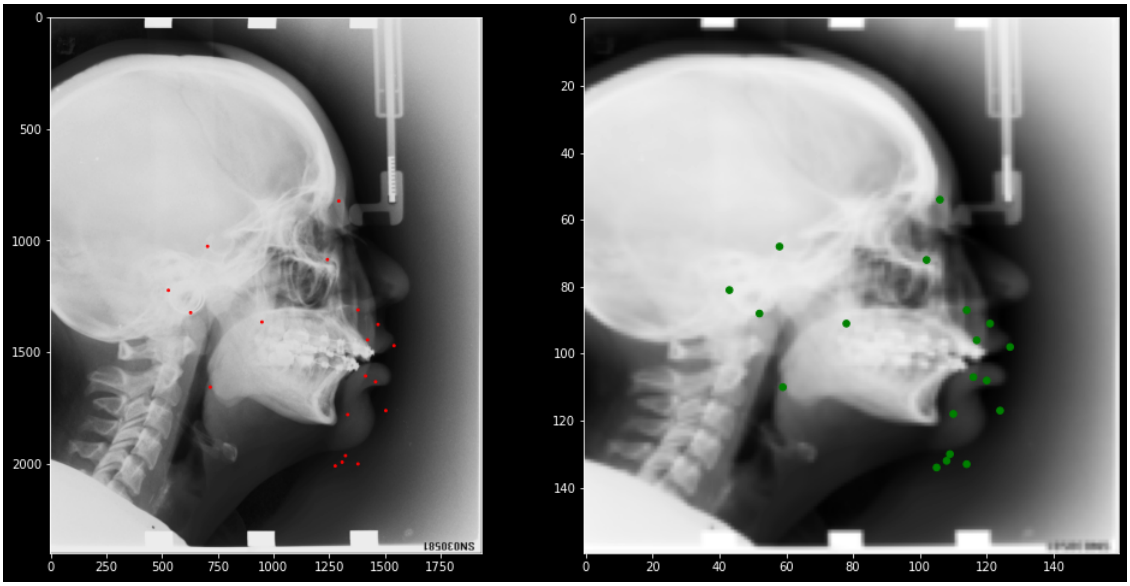


```
[[0.99607843 0.99607843 0.99607843 … 0.99607843 0.99607843 0.00392157]
```

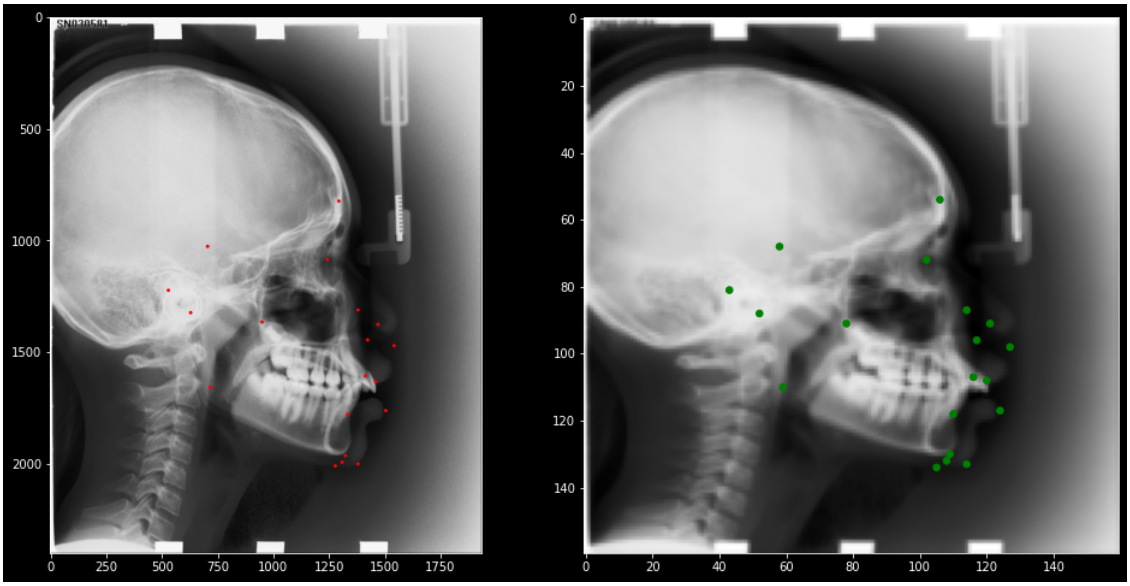
```
 [0.99607843 0.99607843 0.99607843 … 0.99607843 0.99607843 0.00392157]
 [0.99607843 0.99607843 0.99607843 … 0.99607843 0.99607843 0.00392157]
 …
 [0.99607843 0.99607843 0.99607843 … 0.75294118 0.77647059 0.        ]
 [0.99607843 0.99607843 0.99607843 … 0.72156863 0.74509804 0.00392157]
 [0.         0.         0.         … 0.         0.         0.01960784]]
(2400, 1935)
```




```
[[0.11372549 0.11372549 0.11372549 … 0.67843137 0.6745098  0.00392157]
 [0.11372549 0.11372549 0.11372549 … 0.67843137 0.6745098  0.00392157]
 [0.11372549 0.11372549 0.11372549 … 0.67843137 0.6745098  0.00392157]
 …
 [0.67843137 0.67843137 0.67843137 … 0.67843137 0.69803922 0.        ]
 [0.67058824 0.67058824 0.67058824 … 0.68235294 0.67058824 0.01568627]
 [0.00392157 0.00392157 0.00392157 … 0.         0.         0.        ]]
(2400, 1935)
```

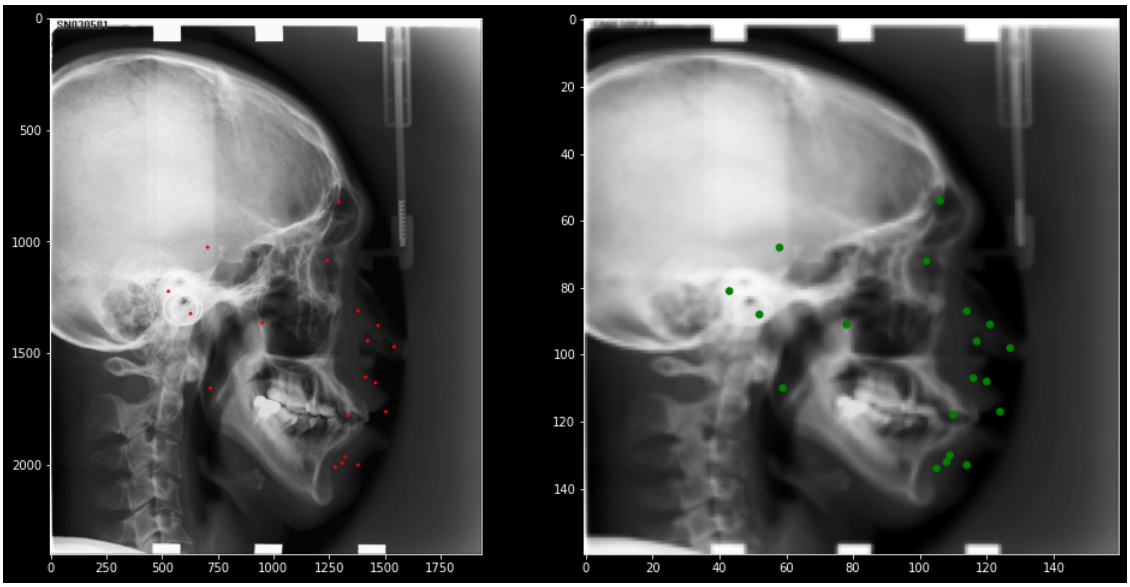


```
[[0.96862745 0.96862745 0.96862745 … 0.97254902 0.96470588 0.        ]
 [0.96862745 0.96862745 0.96862745 … 0.97254902 0.96470588 0.        ]
 [0.96862745 0.96862745 0.96862745 … 0.97254902 0.96470588 0.        ]
 …
 [0.97254902 0.97254902 0.97254902 … 0.88235294 0.89019608 0.        ]
 [0.96862745 0.96862745 0.96862745 … 0.89019608 0.89803922 0.        ]
 [0.         0.         0.         … 0.         0.         0.01176471]]
(2400, 1935)
```

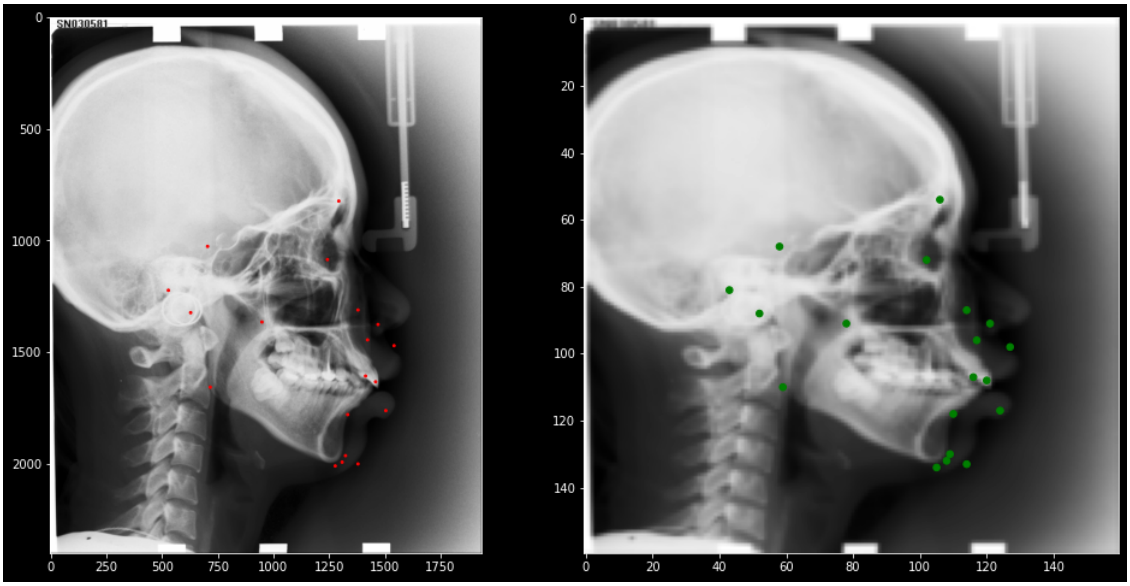


```
[[0.98039216 0.98039216 0.98039216 … 0.98039216 0.98431373 0.        ]
```

```
 [0.98039216 0.98039216 0.98039216 … 0.98039216 0.98431373 0.        ]
 [0.98039216 0.98039216 0.98039216 … 0.98039216 0.98431373 0.        ]
 …
 [0.97647059 0.97647059 0.97647059 … 0.43137255 0.45490196 0.        ]
 [0.98039216 0.98039216 0.98039216 … 0.44313725 0.40784314 0.05098039]
 [0.         0.         0.         … 0.         0.02745098 0.        ]]
(2400, 1935)
```



```
[[0.99607843 0.99607843 0.99607843 … 0.99607843 0.99607843 0.00392157]
 [0.99607843 0.99607843 0.99607843 … 0.99607843 0.99607843 0.00392157]
 [0.99607843 0.99607843 0.99607843 … 0.99607843 0.99607843 0.00392157]
 …
 [0.99607843 0.99607843 0.99607843 … 0.77254902 0.84313725 0.        ]
 [0.99607843 0.99607843 0.99607843 … 0.83921569 0.81960784 0.00392157]
 [0.         0.         0.         … 0.         0.01568627 0.00784314]]
(2400, 1935)
```



1.1 Define transforms

In the previous section we looked at a variety of transforms available for data augmentation (rotate, flip, etc.) and normalization. Here we'll combine the ones we want, including the recommended normalization parameters for mean and std per channel.

[99]:
```
train_transform = transforms.Compose([
        transforms.Resize(224),              # resize shortest side to 224 pixels
        transforms.CenterCrop(224),          # crop longest side to 224 pixels␣
 ↪at center
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ])

test_transform = transforms.Compose([
        transforms.Resize(224),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ])
```

[100]:
```
root = 'data/RawImage/'

train_data = datasets.ImageFolder(os.path.join(root, 'Train'),␣
 ↪transform=train_transform)
```

```
test_data = datasets.ImageFolder(os.path.join(root, 'Test'),␣
↪transform=test_transform)
```

[101]:
```
train_data
```

[101]:
```
Dataset ImageFolder
    Number of datapoints: 150
    Root location: data/RawImage/Train
    StandardTransform
Transform: Compose(
               Resize(size=224, interpolation=PIL.Image.BILINEAR)
               CenterCrop(size=(224, 224))
               ToTensor()
               Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
           )
```

[102]:
```
test_data
```

[102]:
```
Dataset ImageFolder
    Number of datapoints: 250
    Root location: data/RawImage/Test
    StandardTransform
Transform: Compose(
               Resize(size=224, interpolation=PIL.Image.BILINEAR)
               CenterCrop(size=(224, 224))
               ToTensor()
               Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
           )
```

[103]:
```
torch.manual_seed(42)
train_loader = DataLoader(train_data, batch_size=10, shuffle=True)
test_loader = DataLoader(test_data, batch_size=10, shuffle=True)
```

[104]:
```
class_names = test_data.classes

print(class_names)
print(f'Training images available: {len(train_data)}')
print(f'Testing images available:  {len(test_data)}')
```

```
['Test1Data', 'Test2Data']
Training images available: 150
Testing images available:  250
```

1.2 Display a batch of images

To verify that the training loader selects cat and dog images at random, let's show a batch of loaded images. Recall that imshow clips pixel values <0, so the resulting display lacks contrast. We'll

apply a quick inverse transform to the input tensor so that images show their ”true” colors.

[105]:
```
# Grab the first batch of 10 images
for images,labels in train_loader:
    break
```

[106]:
```
images
```

[106]:
```
tensor([[[[ 2.2489,  2.2318,  0.4508,  …,  2.0263,  2.0605,  1.7523],
          [ 2.2489,  2.1975,  0.3481,  …,  2.0263,  2.0434,  1.7352],
          [ 2.2489,  2.1804,  0.1768,  …,  2.0092,  2.0605,  1.7694],
          …,
          [ 2.1804, -0.3541, -1.9467,  …,  0.9132,  0.9988,  0.8447],
          [ 2.1975, -0.2856, -1.9295,  …,  0.9646,  1.0159,  0.8276],
          [ 2.1975, -0.1999, -1.8953,  …,  0.9646,  1.0673,  0.8961]],

         [[ 2.4286,  2.4111,  0.5903,  …,  2.2010,  2.2360,  1.9209],
          [ 2.4286,  2.3761,  0.4853,  …,  2.2010,  2.2185,  1.9034],
          [ 2.4286,  2.3585,  0.3102,  …,  2.1835,  2.2360,  1.9384],
          …,
          [ 2.3585, -0.2325, -1.8606,  …,  1.0630,  1.1506,  0.9930],
          [ 2.3761, -0.1625, -1.8431,  …,  1.1155,  1.1681,  0.9755],
          [ 2.3761, -0.0749, -1.8081,  …,  1.1155,  1.2206,  1.0455]],

         [[ 2.6400,  2.6226,  0.8099,  …,  2.4134,  2.4483,  2.1346],
          [ 2.6400,  2.5877,  0.7054,  …,  2.4134,  2.4308,  2.1171],
          [ 2.6400,  2.5703,  0.5311,  …,  2.3960,  2.4483,  2.1520],
          …,
          [ 2.5703, -0.0092, -1.6302,  …,  1.2805,  1.3677,  1.2108],
          [ 2.5877,  0.0605, -1.6127,  …,  1.3328,  1.3851,  1.1934],
          [ 2.5877,  0.1476, -1.5779,  …,  1.3328,  1.4374,  1.2631]]],


        [[[ 1.1015, -1.6213, -2.0152,  …,  1.7352,  1.7523,  1.4954],
          [ 1.1187, -1.6042, -2.0152,  …,  1.7352,  1.7523,  1.4954],
          [ 1.1358, -1.6042, -1.9638,  …,  1.7180,  1.7523,  1.4954],
          …,
          [ 2.1975,  2.1804,  1.4783,  …,  0.5022,  0.5707,  0.4166],
          [ 2.2147,  2.1804,  1.5297,  …,  0.5364,  0.5878,  0.4337],
          [ 2.2147,  2.1804,  1.5639,  …,  0.5364,  0.6049,  0.4508]],

         [[ 1.2556, -1.5280, -1.9307,  …,  1.9034,  1.9209,  1.6583],
          [ 1.2731, -1.5105, -1.9307,  …,  1.9034,  1.9209,  1.6583],
          [ 1.2906, -1.5105, -1.8782,  …,  1.8859,  1.9209,  1.6583],
          …,
          [ 2.3761,  2.3585,  1.6408,  …,  0.6429,  0.7129,  0.5553],
          [ 2.3936,  2.3585,  1.6933,  …,  0.6779,  0.7304,  0.5728],
```

```
  [ 2.3936,  2.3585,  1.7283,  …,  0.6779,  0.7479,  0.5903]],

 [[ 1.4722, -1.2990, -1.6999,  …,  2.1171,  2.1346,  1.8731],
  [ 1.4897, -1.2816, -1.6999,  …,  2.1171,  2.1346,  1.8731],
  [ 1.5071, -1.2816, -1.6476,  …,  2.0997,  2.1346,  1.8731],
  …,
  [ 2.5877,  2.5703,  1.8557,  …,  0.8622,  0.9319,  0.7751],
  [ 2.6051,  2.5703,  1.9080,  …,  0.8971,  0.9494,  0.7925],
  [ 2.6051,  2.5703,  1.9428,  …,  0.8971,  0.9668,  0.8099]]],


[[[ 2.2318,  2.2318,  1.2899,  …,  1.6838,  1.7009,  1.4783],
  [ 2.2318,  2.2147,  1.2214,  …,  1.6667,  1.7009,  1.4440],
  [ 2.2318,  2.2318,  1.6667,  …,  1.6495,  1.7009,  1.4440],
  …,
  [ 2.1462,  0.1597, -2.0323,  …,  0.5193,  0.6049,  0.4679],
  [ 2.1462,  0.1939, -2.0323,  …,  0.5536,  0.6221,  0.4851],
  [ 2.1462,  0.2453, -2.0323,  …,  0.5707,  0.6221,  0.4851]],

 [[ 2.4111,  2.4111,  1.4482,  …,  1.8508,  1.8683,  1.6408],
  [ 2.4111,  2.3936,  1.3782,  …,  1.8333,  1.8683,  1.6057],
  [ 2.4111,  2.4111,  1.8333,  …,  1.8158,  1.8683,  1.6057],
  …,
  [ 2.3235,  0.2927, -1.9482,  …,  0.6604,  0.7479,  0.6078],
  [ 2.3235,  0.3277, -1.9482,  …,  0.6954,  0.7654,  0.6254],
  [ 2.3235,  0.3803, -1.9482,  …,  0.7129,  0.7654,  0.6254]],

 [[ 2.6226,  2.6226,  1.6640,  …,  2.0648,  2.0823,  1.8557],
  [ 2.6226,  2.6051,  1.5942,  …,  2.0474,  2.0823,  1.8208],
  [ 2.6226,  2.6226,  2.0474,  …,  2.0300,  2.0823,  1.8208],
  …,
  [ 2.5354,  0.5136, -1.7173,  …,  0.8797,  0.9668,  0.8274],
  [ 2.5354,  0.5485, -1.7173,  …,  0.9145,  0.9842,  0.8448],
  [ 2.5354,  0.6008, -1.7173,  …,  0.9319,  0.9842,  0.8448]]],


…,


[[[-1.5014, -1.4672, -1.4329,  …,  1.5297,  1.8208,  1.7180],
  [-1.4672, -1.4329, -1.3987,  …,  1.5125,  1.8208,  1.7180],
  [-1.4329, -1.3987, -1.3987,  …,  1.4954,  1.8037,  1.7180],
  …,
  [-1.7583, -1.7583, -1.7240,  …,  0.6734,  1.6667,  1.7180],
  [-1.7583, -1.7412, -1.7240,  …,  0.7591,  1.7352,  1.7180],
  [-1.7583, -1.7412, -1.7069,  …,  0.9132,  1.8722,  1.7180]],
```

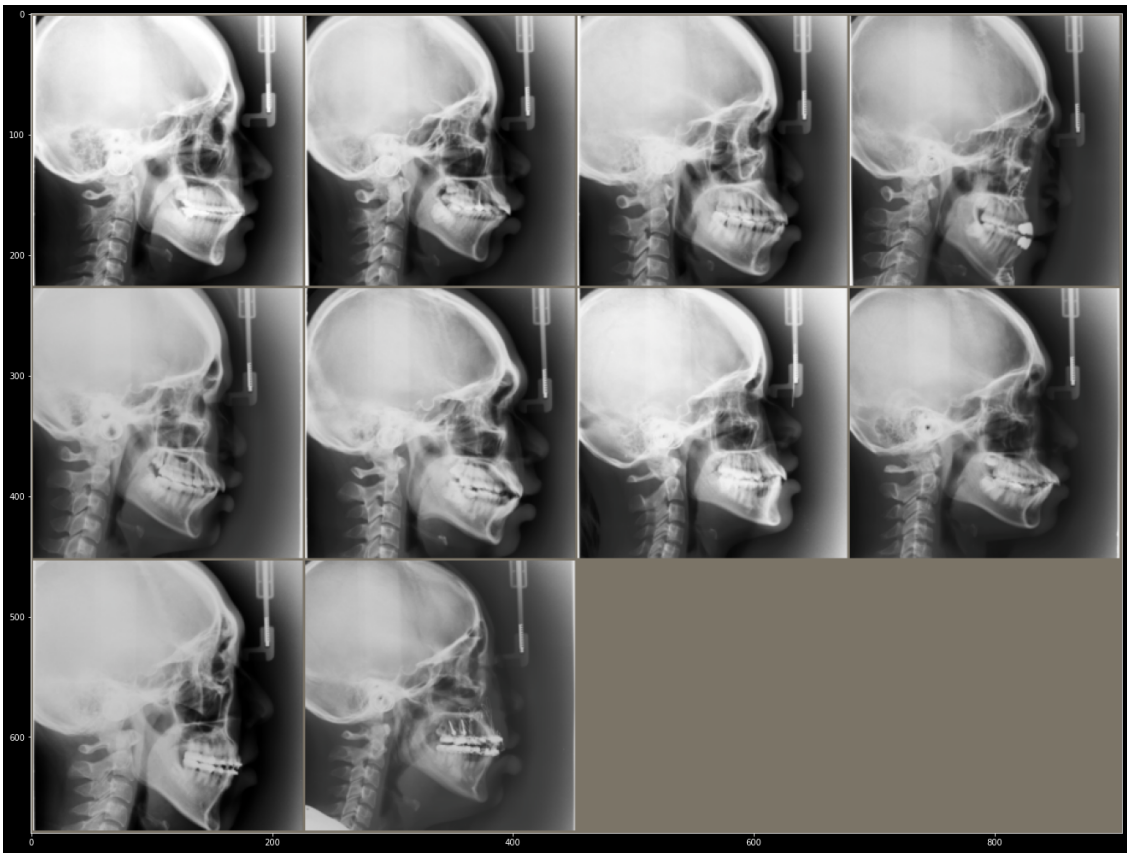
```
 [[-1.4055, -1.3704, -1.3354,  …,  1.6933,  1.9909,  1.8859],
  [-1.3704, -1.3354, -1.3004,  …,  1.6758,  1.9909,  1.8859],
  [-1.3354, -1.3004, -1.3004,  …,  1.6583,  1.9734,  1.8859],
  …,
  [-1.6681, -1.6681, -1.6331,  …,  0.8179,  1.8333,  1.8859],
  [-1.6681, -1.6506, -1.6331,  …,  0.9055,  1.9034,  1.8859],
  [-1.6681, -1.6506, -1.6155,  …,  1.0630,  2.0434,  1.8859]],

 [[-1.1770, -1.1421, -1.1073,  …,  1.9080,  2.2043,  2.0997],
  [-1.1421, -1.1073, -1.0724,  …,  1.8905,  2.2043,  2.0997],
  [-1.1073, -1.0724, -1.0724,  …,  1.8731,  2.1868,  2.0997],
  …,
  [-1.4384, -1.4384, -1.4036,  …,  1.0365,  2.0474,  2.0997],
  [-1.4384, -1.4210, -1.4036,  …,  1.1237,  2.1171,  2.0997],
  [-1.4384, -1.4210, -1.3861,  …,  1.2805,  2.2566,  2.0997]]],


[[[ 1.6495,  1.5982,  1.3413,  …,  1.0331,  1.0673,  0.8447],
  [ 1.6495,  1.5982,  1.4098,  …,  1.0331,  1.0502,  0.8447],
  [ 1.6495,  1.5639,  1.3927,  …,  1.0159,  1.0502,  0.8447],
  …,
  [ 1.4612, -1.0562, -2.0665,  …, -0.0972,  0.0227, -0.0972],
  [ 1.4783, -1.0048, -2.0494,  …, -0.0629,  0.0056, -0.0972],
  [ 1.4954, -0.9192, -2.0494,  …, -0.0116,  0.0398, -0.0287]],

 [[ 1.8158,  1.7633,  1.5007,  …,  1.1856,  1.2206,  0.9930],
  [ 1.8158,  1.7633,  1.5707,  …,  1.1856,  1.2031,  0.9930],
  [ 1.8158,  1.7283,  1.5532,  …,  1.1681,  1.2031,  0.9930],
  …,
  [ 1.6232, -0.9503, -1.9832,  …,  0.0301,  0.1527,  0.0301],
  [ 1.6408, -0.8978, -1.9657,  …,  0.0651,  0.1352,  0.0301],
  [ 1.6583, -0.8102, -1.9657,  …,  0.1176,  0.1702,  0.1001]],

 [[ 2.0300,  1.9777,  1.7163,  …,  1.4025,  1.4374,  1.2108],
  [ 2.0300,  1.9777,  1.7860,  …,  1.4025,  1.4200,  1.2108],
  [ 2.0300,  1.9428,  1.7685,  …,  1.3851,  1.4200,  1.2108],
  …,
  [ 1.8383, -0.7238, -1.7522,  …,  0.2522,  0.3742,  0.2522],
  [ 1.8557, -0.6715, -1.7347,  …,  0.2871,  0.3568,  0.2522],
  [ 1.8731, -0.5844, -1.7347,  …,  0.3393,  0.3916,  0.3219]]],


[[[-1.0562, -1.0390, -0.9877,  …,  1.3242,  1.7180,  1.4783],
  [-1.0390, -0.9534, -0.4911,  …,  1.2899,  1.7180,  1.4783],
  [-0.8678, -0.3198,  0.4508,  …,  1.2899,  1.7180,  1.4783],
  …,
  [ 1.4098,  1.3927,  1.3927,  …,  0.1254,  0.6049,  1.3242],
```

```
 [ 1.4098,  1.4098,  1.3927,  …,  0.1597,  0.6392,  1.3584],
 [ 1.4440,  1.4098,  1.4098,  …,  0.1597,  0.6563,  1.3755]],

[[-0.9503, -0.9328, -0.8803,  …,  1.4832,  1.8859,  1.6408],
 [-0.9328, -0.8452, -0.3725,  …,  1.4482,  1.8859,  1.6408],
 [-0.7577, -0.1975,  0.5903,  …,  1.4482,  1.8859,  1.6408],
 …,
 [ 1.5707,  1.5532,  1.5532,  …,  0.2577,  0.7479,  1.4832],
 [ 1.5707,  1.5707,  1.5532,  …,  0.2927,  0.7829,  1.5182],
 [ 1.6057,  1.5707,  1.5707,  …,  0.2927,  0.8004,  1.5357]],

[[-0.7238, -0.7064, -0.6541,  …,  1.6988,  2.0997,  1.8557],
 [-0.7064, -0.6193, -0.1487,  …,  1.6640,  2.0997,  1.8557],
 [-0.5321,  0.0256,  0.8099,  …,  1.6640,  2.0997,  1.8557],
 …,
 [ 1.7860,  1.7685,  1.7685,  …,  0.4788,  0.9668,  1.6988],
 [ 1.7860,  1.7860,  1.7685,  …,  0.5136,  1.0017,  1.7337],
 [ 1.8208,  1.7860,  1.7860,  …,  0.5136,  1.0191,  1.7511]]])
```

[107]:
```
labels
```

[107]:
```
tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

[108]:
```
# Print the labels
im = make_grid(images, nrow=4)  # the default nrow is 8

# Inverse normalize the images
inv_normalize = transforms.Normalize(
    mean=[-0.485/0.229, -0.456/0.224, -0.406/0.225],
    std=[1/0.229, 1/0.224, 1/0.225]
)
im_inv = inv_normalize(im)

# Print the images
plt.figure(figsize=(32,18))
plt.imshow(np.transpose(im_inv.numpy(), (1, 2, 0)));
```


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