Bone_Age_Detection_model

November 13, 2021

[ ]:

```

```

[2]:

```
!pip install -q albumentations==0.4.6
```

```
    || 117 kB 5.5 MB/s
    || 948 kB 23.9 MB/s
 Building wheel for albumentations (setup.py) ... done
```

0.1 Imports

[3]:

```
import os
import torch
import torch.nn as nn
import torchvision
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
import torchvision.transforms.functional as TF
from torchvision.transforms import transforms as T
import torch.optim as optim

import albumentations as A
from albumentations.pytorch import ToTensorV2
from tqdm import tqdm

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import cv2
from PIL import Image
```

0.2 Customized Dataset

[4]:

```
y=[]
for i in range(1,81):# 80 images
  v=f'img_{i}.png'
  y.append(v)
class Data(Dataset):#Dataset stores the samples and their corresponding labels
```

```
    def __init__(self, image_dir, mask_dir, transform=None):
        self.image_dir = image_dir
        self.mask_dir = mask_dir
        self.transform = transform
        self.images = []

        for i in y:
          for j in os.listdir(image_dir):
            if i==j:
              self.images.append(j)

    def __len__(self):#The __len__ function returns the number of samples in␣
↪our dataset.
        return len(self.images)

    def __getitem__(self, index):#The __getitem__ function loads and returns a␣
↪sample from the dataset at the given index idx. Based on the index, it␣
↪identifies the images location on disk, converts that to a tensor using␣
↪read_image
        img_path = os.path.join(self.image_dir, self.images[index])
        mask_path = os.path.join(self.mask_dir, self.images[index])
        image = np.array(Image.open(img_path).convert("RGB"))
        mask = np.array(Image.open(mask_path).convert("L"), dtype=np.float32)
        mask[mask == 255.0] = 1.0

        if self.transform is not None:
            augmentations = self.transform(image=image, mask=mask)
            image = augmentations["image"]
            mask = augmentations["mask"]


        return image, mask
```

0.3 Unet Model

[5]:
```
import torch
import torch.nn as nn
import torchvision.transforms.functional as TF

class DoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DoubleConv, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 3, 1, 1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
```

```python
            nn.Conv2d(out_channels, out_channels, 3, 1, 1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
        )

    def forward(self, x):
        return self.conv(x)

class UNET(nn.Module):
    def __init__(
            self, in_channels=3, out_channels=1, features=[64, 128, 256, 512],
    ):
        super(UNET, self).__init__()
        self.ups = nn.ModuleList()
        self.downs = nn.ModuleList()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Down part of UNET
        for feature in features:
            self.downs.append(DoubleConv(in_channels, feature))
            in_channels = feature

        # Up part of UNET
        for feature in reversed(features):
            self.ups.append(
                nn.ConvTranspose2d(
                    feature*2, feature, kernel_size=2, stride=2,
                )
            )
            self.ups.append(DoubleConv(feature*2, feature))

        self.bottleneck = DoubleConv(features[-1], features[-1]*2)
        self.final_conv = nn.Conv2d(features[0], out_channels, kernel_size=1)

    def forward(self, x):
        skip_connections = []

        for down in self.downs:
            x = down(x)
            skip_connections.append(x)
            x = self.pool(x)

        x = self.bottleneck(x)
        skip_connections = skip_connections[::-1]

        for idx in range(0, len(self.ups), 2):
            x = self.ups[idx](x)
```

```
            skip_connection = skip_connections[idx//2]

            if x.shape != skip_connection.shape:
                x = TF.resize(x, size=skip_connection.shape[2:])

            concat_skip = torch.cat((skip_connection, x), dim=1)
            x = self.ups[idx+1](concat_skip)

        return self.final_conv(x)

def test():
    x = torch.randn((3, 1, 161, 161))
    model = UNET(in_channels=1, out_channels=1)
    preds = model(x)
    assert preds.shape == x.shape

if __name__ == "__main__":
    test()
```

[6]:
```
#Device = "cuda" if torch.cuda.is_available() else "cpu"
model=UNET(3,1)
model=torch.load('/content/drive/MyDrive/Bone_Age_Detection /Saved_model/model1.
 ↪pth', map_location=torch.device('cpu'))
model.eval()
print('model loaded')
```

```
model loaded
```

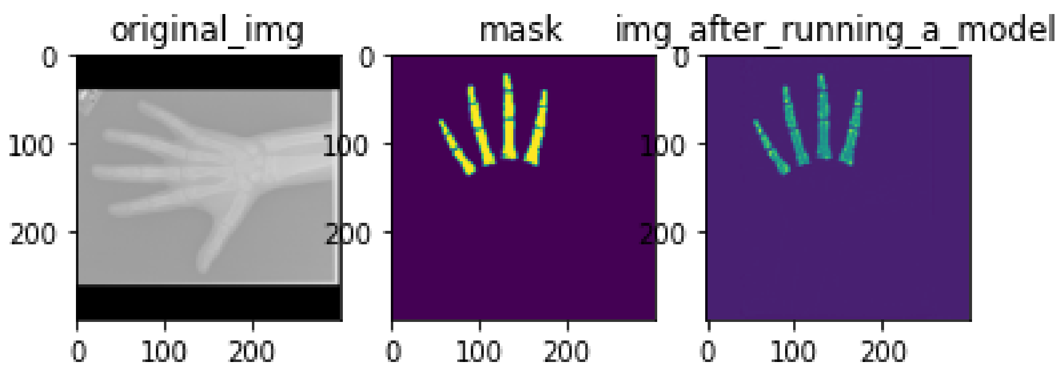
0.4 Visualize image and mask

[7]:
```
IMG_DIR='/content/drive/MyDrive/Bone_Age_Detection /50images'
LABEL_DIR='/content/drive/MyDrive/Bone_Age_Detection /50label'
VAL_IMG_DIR='/content/drive/MyDrive/Bone_Age_Detection /Val_data'
VAL_MASK_DIR='/content/drive/MyDrive/Bone_Age_Detection /Val_masks'
Device = "cuda" if torch.cuda.is_available() else "cpu"
#BATCH_SIZE=1# to display img change batch_size to 1





train_transforms = A.Compose([A.Resize(height=300, width=300), A.Normalize(
                mean=[0.0, 0.0, 0.0],
                std=[1.0, 1.0, 1.0],
                max_pixel_value=255.0), ToTensorV2()])
val_transforms = A.Compose([A.Resize(height=300, width=300), A.Normalize(
                mean=[0.0, 0.0, 0.0],
                std=[1.0, 1.0, 1.0],
```

```
                max_pixel_value=255.0), ToTensorV2()])

train_data=Data(IMG_DIR, LABEL_DIR, train_transforms)
train_load=DataLoader(train_data, batch_size=1, shuffle=True)#DataLoader wraps␣
↪an iterable around the Dataset to enable easy access to the samples.

val_data=Data(VAL_IMG_DIR, VAL_MASK_DIR, val_transforms)
val_load=DataLoader(train_data, batch_size=1, shuffle=False)

batch= next(iter(train_load)) #Each iteration below returns a batch of␣
↪train_features and train_labels (containing batch_size features and labels␣
↪respectively). Because we specified shuffle=True, after we iterate over all␣
↪batches the data is shuffled
image, label = batch
image=image.to(Device)
#print(image.shape)
image=image
mm=torch.sigmoid(model(image))

#print('img_dim_after_running_model',mm.shape)

with torch.no_grad(): #disable gradient calculation. when you are sure that you␣
↪will not call Tensor.backward(). It will reduce memory consumption for␣
↪computations
  m=(model(image)).squeeze(0).squeeze(0).cpu().numpy()#squeeze removes axes␣
↪that have length of 1


image=image.cpu()

#print('img_shape: ',m.shape)
#print('label_shape: ',label.shape)

image=image.squeeze(0)
image=image.numpy().transpose(2,1,0)#diplayed img is rotated bcz of transpose␣
↪but img passed in model will be upright
label=label.squeeze(0).numpy()

plt.subplot(1,3,1)
plt.title('original_img')
plt.imshow(image)#(300, 300, 3)

plt.subplot(1,3,2)
plt.title('mask')
plt.imshow(label)#(300, 300)

plt.subplot(1,3,3)
```

```python
plt.title('img_after_running_a_model')
plt.imshow(m)
```

[7]: <matplotlib.image.AxesImage at 0x7f5bad0ade50>



0.5 Hyperparameters

[8]:
```python
Device = "cuda" if torch.cuda.is_available() else "cpu"
BATCH_SIZE=20
NUM_EPOCHS=150
LEARNING_RATE=0.0001

loss_fn = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)

#model=UNET(3,1).to(Device)
#model.load_state_dict(torch.load('/content/gdrive/MyDrive/Bone_Age_Detection /
↪Saved_model/model_weights.pth'))
#optimizer.load_state_dict(torch.load('/content/gdrive/MyDrive/
↪Bone_Age_Detection /preds/optimizer.pth'))
model.eval()

train_data=Data(IMG_DIR, LABEL_DIR, train_transforms)
train_loader=DataLoader(train_data, batch_size=BATCH_SIZE, shuffle=True)

val_data=Data(VAL_IMG_DIR, VAL_MASK_DIR, val_transforms)
val_loader=DataLoader(train_data, batch_size=1, shuffle=False)
```

0.6 Check Accuracy

[9]:
```
def check_accuracy(loader, model, device="cuda"):
    num_correct = 0
    num_pixels = 0
    dice_score = 0
    model.eval()
#model.eval() is a kind of switch for some specific layers/parts of the model␣
 ↪that behave differently during training and inference (evaluating) time. It␣
 ↪will turn off  Dropouts Layers, BatchNorm Layers etc.
#common practice for evaluating/validation is using torch.no_grad() in pair␣
 ↪with model.eval() to turn off gradients computation.
#we need to turn back to training mode after eval step.

    with torch.no_grad():#we don't use gradients during evaluation, so turning␣
 ↪off the autograd will speed up execution and will reduce memory usage
        for x, y in loader:
            x = x.to(Device)
            y = y.to(Device).unsqueeze(1)
            preds = torch.sigmoid(model(x))#
            preds = (preds > 0.5).float()
            num_correct += (preds == y).sum()
            num_pixels += torch.numel(preds)#calculate number of elements in a␣
 ↪tensor
            dice_score += (2 * (preds * y).sum()) / (#calculate area of overlap␣
 ↪between mask and prediction
                (preds + y).sum() + 1e-8
            )

    print(f"Got {num_correct}/{num_pixels} with acc {num_correct/num_pixels*100:
 ↪.2f}")
    print(f"Dice score: {dice_score/len(loader)}")
    model.train()

#turn off evaluation mode by running model.train(). You should use it when␣
 ↪running your model as an inference engine - i.e. when testing, validating,␣
 ↪and predicting
```

0.7 Train

[ ]:
```
def main():

    scaler = torch.cuda.amp.GradScaler()

    for epoch in range(NUM_EPOCHS):
      loop = tqdm(train_loader)
      #tqdm is a Python library that allows you to output a smart progress bar␣
 ↪by wrapping around any iterable
```

```
    #tqdm progress bar gives us information that includes the task completion␣
↪percentage, number of iterations complete, time elapsed, estimated time␣
↪remaining,
    # and the iterations completed per second.

    for batch_idx, (data, targets) in enumerate(loop):
        l=0
        data = data.to(device=Device)
        targets = targets.float().unsqueeze(1).to(device=Device)

        # forward
        with torch.cuda.amp.autocast():
            predictions =model(data)
            loss = loss_fn(predictions, targets)#predictions and target␣
↪should be of same size

        # backward
        optimizer.zero_grad()# zero out the gradients that are held in the␣
↪grad attribute of weights
        scaler.scale(loss).backward()#calculate gradient which are used to␣
↪update weight
        scaler.step(optimizer)#Update the weights
        scaler.update()
        l+=loss.item()

        # update tqdm loop
        loop.set_postfix(loss=loss.item())#set_postfix to add values directly␣
↪to the bar.
    check_accuracy(val_loader, model, device=Device)

    print('Epoch: ',epoch+1)

main()
```

```
100%|| 4/4 [02:03<00:00, 30.93s/it, loss=0.545]

Got 7035310/7200000 with acc 97.71
Dice score: 0.3176306188106537
Epoch:  1

100%|| 4/4 [00:19<00:00,  4.80s/it, loss=0.0466]

Got 6987373/7200000 with acc 97.05
Dice score: 0.0
Epoch:  2

100%|| 4/4 [00:19<00:00,  4.76s/it, loss=0.0467]
```

```
Got 6987373/7200000 with acc 97.05
Dice score: 0.0
Epoch:  3

100%|| 4/4 [00:19<00:00,  4.78s/it, loss=0.0436]

Got 6987373/7200000 with acc 97.05
Dice score: 0.0
Epoch:  4

100%|| 4/4 [00:19<00:00,  4.76s/it, loss=0.0412]

Got 6987386/7200000 with acc 97.05
Dice score: 0.0001861993077909574
Epoch:  5

100%|| 4/4 [00:19<00:00,  4.76s/it, loss=0.0398]

Got 6993156/7200000 with acc 97.13
Dice score: 0.04901351407170296
Epoch:  6

100%|| 4/4 [00:19<00:00,  4.75s/it, loss=0.0385]

Got 7038470/7200000 with acc 97.76
Dice score: 0.3241806924343109
Epoch:  7

100%|| 4/4 [00:19<00:00,  4.76s/it, loss=0.0369]

Got 7119328/7200000 with acc 98.88
Dice score: 0.7126339673995972
Epoch:  8

100%|| 4/4 [00:18<00:00,  4.75s/it, loss=0.0361]

Got 7165857/7200000 with acc 99.53
Dice score: 0.8944363594055176
Epoch:  9

100%|| 4/4 [00:19<00:00,  4.77s/it, loss=0.0355]

Got 7181330/7200000 with acc 99.74
Dice score: 0.9465829730033875
Epoch:  10

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0347]

Got 7188875/7200000 with acc 99.85
Dice score: 0.9698479771614075
Epoch:  11
```

```
100%|| 4/4 [00:19<00:00,  4.75s/it, loss=0.0343]

Got 7191455/7200000 with acc 99.88
Dice score: 0.9775075316429138
Epoch:  12

100%|| 4/4 [00:18<00:00,  4.74s/it, loss=0.0337]

Got 7192994/7200000 with acc 99.90
Dice score: 0.9816795587539673
Epoch:  13

100%|| 4/4 [00:19<00:00,  4.77s/it, loss=0.033]

Got 7193955/7200000 with acc 99.92
Dice score: 0.984163224697113
Epoch:  14

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0322]

Got 7194782/7200000 with acc 99.93
Dice score: 0.9864413142204285
Epoch:  15

100%|| 4/4 [00:18<00:00,  4.75s/it, loss=0.0316]

Got 7195254/7200000 with acc 99.93
Dice score: 0.987678050994873
Epoch:  16

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0318]

Got 7195140/7200000 with acc 99.93
Dice score: 0.9875271916389465
Epoch:  17

100%|| 4/4 [00:19<00:00,  4.75s/it, loss=0.0313]

Got 7196291/7200000 with acc 99.95
Dice score: 0.9903525710105896
Epoch:  18

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0309]

Got 7196436/7200000 with acc 99.95
Dice score: 0.9908224940299988
Epoch:  19

100%|| 4/4 [00:18<00:00,  4.74s/it, loss=0.0301]
```

```
Got 7196270/7200000 with acc 99.95
Dice score: 0.9904324412345886
Epoch:  20

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0298]

Got 7197102/7200000 with acc 99.96
Dice score: 0.992512047290802
Epoch:  21

100%|| 4/4 [00:18<00:00,  4.74s/it, loss=0.0293]

Got 7197157/7200000 with acc 99.96
Dice score: 0.992719829082489
Epoch:  22

100%|| 4/4 [00:18<00:00,  4.74s/it, loss=0.0295]

Got 7197432/7200000 with acc 99.96
Dice score: 0.9933831095695496
Epoch:  23

100%|| 4/4 [00:19<00:00,  4.75s/it, loss=0.0291]

Got 7197822/7200000 with acc 99.97
Dice score: 0.994442880153656
Epoch:  24

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0284]

Got 7197574/7200000 with acc 99.97
Dice score: 0.9938076138496399
Epoch:  25

100%|| 4/4 [00:18<00:00,  4.75s/it, loss=0.0282]

Got 7198140/7200000 with acc 99.97
Dice score: 0.9952195286750793
Epoch:  26

100%|| 4/4 [00:18<00:00,  4.74s/it, loss=0.0275]

Got 7198268/7200000 with acc 99.98
Dice score: 0.9955641627311707
Epoch:  27

100%|| 4/4 [00:18<00:00,  4.74s/it, loss=0.0282]

Got 7197943/7200000 with acc 99.97
Dice score: 0.9948291182518005
Epoch:  28
```

```
100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0273]

Got 7198492/7200000 with acc 99.98
Dice score: 0.9960228204727173
Epoch:  29

100%|| 4/4 [00:18<00:00,  4.75s/it, loss=0.0269]

Got 7198039/7200000 with acc 99.97
Dice score: 0.9951034784317017
Epoch:  30

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0263]

Got 7198655/7200000 with acc 99.98
Dice score: 0.996508777141571
Epoch:  31

100%|| 4/4 [00:18<00:00,  4.74s/it, loss=0.0262]

Got 7198735/7200000 with acc 99.98
Dice score: 0.9967333078384399
Epoch:  32

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.026]

Got 7198554/7200000 with acc 99.98
Dice score: 0.9963086247444153
Epoch:  33

100%|| 4/4 [00:19<00:00,  4.76s/it, loss=0.0255]

Got 7198942/7200000 with acc 99.99
Dice score: 0.9972529411315918
Epoch:  34

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0259]

Got 7198773/7200000 with acc 99.98
Dice score: 0.9968667030334473
Epoch:  35

100%|| 4/4 [00:18<00:00,  4.75s/it, loss=0.0253]

Got 7199005/7200000 with acc 99.99
Dice score: 0.9974271059036255
Epoch:  36

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0253]
```

```
Got 7198916/7200000 with acc 99.98
Dice score: 0.9972519874572754
Epoch:  37

100%|| 4/4 [00:18<00:00,  4.75s/it, loss=0.0246]

Got 7199076/7200000 with acc 99.99
Dice score: 0.9975990653038025
Epoch:  38

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0244]

Got 7198974/7200000 with acc 99.99
Dice score: 0.9973611235618591
Epoch:  39

100%|| 4/4 [00:18<00:00,  4.74s/it, loss=0.0245]

Got 7199147/7200000 with acc 99.99
Dice score: 0.9977502226829529
Epoch:  40

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.024]

Got 7199159/7200000 with acc 99.99
Dice score: 0.9978005290031433
Epoch:  41

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0242]

Got 7198985/7200000 with acc 99.99
Dice score: 0.9973844885826111
Epoch:  42

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0237]

Got 7199231/7200000 with acc 99.99
Dice score: 0.9979600310325623
Epoch:  43

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0235]

Got 7199111/7200000 with acc 99.99
Dice score: 0.9977933764457703
Epoch:  44

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0233]

Got 7199247/7200000 with acc 99.99
Dice score: 0.9980192184448242
Epoch:  45
```

```
100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0229]

Got 7199190/7200000 with acc 99.99
Dice score: 0.9979438781738281
Epoch:  46

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0225]

Got 7199180/7200000 with acc 99.99
Dice score: 0.9978954195976257
Epoch:  47

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0228]

Got 7199261/7200000 with acc 99.99
Dice score: 0.9980871081352234
Epoch:  48

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0223]

Got 7199179/7200000 with acc 99.99
Dice score: 0.9979234933853149
Epoch:  49

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0224]

Got 7199311/7200000 with acc 99.99
Dice score: 0.9982301592826843
Epoch:  50

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0217]

Got 7199348/7200000 with acc 99.99
Dice score: 0.9983268976211548
Epoch:  51

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0216]

Got 7199305/7200000 with acc 99.99
Dice score: 0.9982919692993164
Epoch:  52

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0217]

Got 7199377/7200000 with acc 99.99
Dice score: 0.9983788728713989
Epoch:  53

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0211]
```

```
Got 7199332/7200000 with acc 99.99
Dice score: 0.9982988238334656
Epoch:  54

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0212]

Got 7199405/7200000 with acc 99.99
Dice score: 0.9984545707702637
Epoch:  55

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0211]

Got 7199410/7200000 with acc 99.99
Dice score: 0.9985069632530212
Epoch:  56

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0206]

Got 7199381/7200000 with acc 99.99
Dice score: 0.9984130859375
Epoch:  57

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0207]

Got 7199366/7200000 with acc 99.99
Dice score: 0.9983645677566528
Epoch:  58

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0203]

Got 7199450/7200000 with acc 99.99
Dice score: 0.9985690116882324
Epoch:  59

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0201]

Got 7199437/7200000 with acc 99.99
Dice score: 0.9985697865486145
Epoch:  60

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0201]

Got 7199505/7200000 with acc 99.99
Dice score: 0.9987292289733887
Epoch:  61

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0197]

Got 7199492/7200000 with acc 99.99
Dice score: 0.9986969828605652
Epoch:  62
```

```
100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0195]

Got 7199514/7200000 with acc 99.99
Dice score: 0.998759925365448
Epoch:  63

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0195]

Got 7199474/7200000 with acc 99.99
Dice score: 0.998637318611145
Epoch:  64

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0192]

Got 7199515/7200000 with acc 99.99
Dice score: 0.9987678527832031
Epoch:  65

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0191]

Got 7199507/7200000 with acc 99.99
Dice score: 0.9987370371818542
Epoch:  66

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0188]

Got 7199519/7200000 with acc 99.99
Dice score: 0.9987527132034302
Epoch:  67

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0187]

Got 7199536/7200000 with acc 99.99
Dice score: 0.9987985491752625
Epoch:  68

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0185]

Got 7199562/7200000 with acc 99.99
Dice score: 0.998873233795166
Epoch:  69

100%|| 4/4 [00:18<00:00,  4.74s/it, loss=0.0186]

Got 7199532/7200000 with acc 99.99
Dice score: 0.9987999796867371
Epoch:  70

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0182]
```

```
Got 7199524/7200000 with acc 99.99
Dice score: 0.9987598657608032
Epoch:  71

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0179]

Got 7199553/7200000 with acc 99.99
Dice score: 0.998868465423584
Epoch:  72

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0182]

Got 7199427/7200000 with acc 99.99
Dice score: 0.9985430836677551
Epoch:  73

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0181]

Got 7199555/7200000 with acc 99.99
Dice score: 0.9988415837287903
Epoch:  74

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0181]

Got 7199531/7200000 with acc 99.99
Dice score: 0.9987978339195251
Epoch:  75

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0174]

Got 7199496/7200000 with acc 99.99
Dice score: 0.9987107515335083
Epoch:  76

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0177]

Got 7199556/7200000 with acc 99.99
Dice score: 0.998859703540802
Epoch:  77

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.017]

Got 7199563/7200000 with acc 99.99
Dice score: 0.9988781809806824
Epoch:  78

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.017]

Got 7199583/7200000 with acc 99.99
Dice score: 0.998935341835022
Epoch:  79
```

```
100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.017]

Got 7199545/7200000 with acc 99.99
Dice score: 0.9988075494766235
Epoch:  80

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0168]

Got 7199335/7200000 with acc 99.99
Dice score: 0.9983145594596863
Epoch:  81

100%|| 4/4 [00:18<00:00,  4.74s/it, loss=0.0168]

Got 7199507/7200000 with acc 99.99
Dice score: 0.9986869692802429
Epoch:  82

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0169]

Got 7199431/7200000 with acc 99.99
Dice score: 0.9985559582710266
Epoch:  83

100%|| 4/4 [00:18<00:00,  4.74s/it, loss=0.0164]

Got 7199519/7200000 with acc 99.99
Dice score: 0.998752236366272
Epoch:  84

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0161]

Got 7199551/7200000 with acc 99.99
Dice score: 0.998847484588623
Epoch:  85

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0162]

Got 7199567/7200000 with acc 99.99
Dice score: 0.998859703540802
Epoch:  86

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0157]

Got 7199543/7200000 with acc 99.99
Dice score: 0.9988029599189758
Epoch:  87

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0158]
```

```
Got 7199592/7200000 with acc 99.99
Dice score: 0.998946487903595
Epoch:  88

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0158]

Got 7199616/7200000 with acc 99.99
Dice score: 0.999020516872406
Epoch:  89

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0157]

Got 7199568/7200000 with acc 99.99
Dice score: 0.9989021420478821
Epoch:  90

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0154]

Got 7199541/7200000 with acc 99.99
Dice score: 0.9988512992858887
Epoch:  91

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0154]

Got 7199623/7200000 with acc 99.99
Dice score: 0.9990567564964294
Epoch:  92

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0152]

Got 7199628/7200000 with acc 99.99
Dice score: 0.9990532994270325
Epoch:  93

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0151]

Got 7199615/7200000 with acc 99.99
Dice score: 0.9990069270133972
Epoch:  94

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0151]

Got 7199647/7200000 with acc 100.00
Dice score: 0.9990585446357727
Epoch:  95

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.015]

Got 7199661/7200000 with acc 100.00
Dice score: 0.9991265535354614
Epoch:  96
```

```
100%|| 4/4 [00:18<00:00,  4.69s/it, loss=0.0147]

Got 7199675/7200000 with acc 100.00
Dice score: 0.9991744160652161
Epoch:  97

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0147]

Got 7199649/7200000 with acc 100.00
Dice score: 0.9991083145141602
Epoch:  98

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0145]

Got 7199651/7200000 with acc 100.00
Dice score: 0.9991223216056824
Epoch:  99

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0148]

Got 7199636/7200000 with acc 99.99
Dice score: 0.9990660548210144
Epoch:  100

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0142]

Got 7199578/7200000 with acc 99.99
Dice score: 0.9988829493522644
Epoch:  101

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0141]

Got 7199607/7200000 with acc 99.99
Dice score: 0.9989585280418396
Epoch:  102

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.014]

Got 7199649/7200000 with acc 100.00
Dice score: 0.9990549087524414
Epoch:  103

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0137]

Got 7199658/7200000 with acc 100.00
Dice score: 0.9991490244865417
Epoch:  104

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0138]
```

```
Got 7199653/7200000 with acc 100.00
Dice score: 0.9991070628166199
Epoch:  105

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0141]

Got 7199670/7200000 with acc 100.00
Dice score: 0.9991265535354614
Epoch:  106

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0135]

Got 7199671/7200000 with acc 100.00
Dice score: 0.9991270303726196
Epoch:  107

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0134]

Got 7199672/7200000 with acc 100.00
Dice score: 0.9991514086723328
Epoch:  108

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0133]

Got 7199717/7200000 with acc 100.00
Dice score: 0.9992529153823853
Epoch:  109

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0131]

Got 7199715/7200000 with acc 100.00
Dice score: 0.9992575645446777
Epoch:  110

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.013]

Got 7199740/7200000 with acc 100.00
Dice score: 0.9993292689323425
Epoch:  111

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0129]

Got 7199707/7200000 with acc 100.00
Dice score: 0.999272346496582
Epoch:  112

100%|| 4/4 [00:18<00:00,  4.68s/it, loss=0.0129]

Got 7199753/7200000 with acc 100.00
Dice score: 0.9993577003479004
Epoch:  113
```

```
100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0139]

Got 7199706/7200000 with acc 100.00
Dice score: 0.9992371797561646
Epoch:  114

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0127]

Got 7199711/7200000 with acc 100.00
Dice score: 0.9992603659629822
Epoch:  115

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0126]

Got 7199690/7200000 with acc 100.00
Dice score: 0.9992374777793884
Epoch:  116

100%|| 4/4 [00:18<00:00,  4.68s/it, loss=0.0128]

Got 7199692/7200000 with acc 100.00
Dice score: 0.9992191195487976
Epoch:  117

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0126]

Got 7199695/7200000 with acc 100.00
Dice score: 0.9992397427558899
Epoch:  118

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0125]

Got 7199695/7200000 with acc 100.00
Dice score: 0.999234676361084
Epoch:  119

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0124]

Got 7199727/7200000 with acc 100.00
Dice score: 0.9992827773094177
Epoch:  120

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0124]

Got 7199701/7200000 with acc 100.00
Dice score: 0.9992201924324036
Epoch:  121

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.012]
```

```
Got 7199743/7200000 with acc 100.00
Dice score: 0.9993358850479126
Epoch:  122

100%|| 4/4 [00:18<00:00,  4.69s/it, loss=0.0122]

Got 7199775/7200000 with acc 100.00
Dice score: 0.9993932843208313
Epoch:  123

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0118]

Got 7199735/7200000 with acc 100.00
Dice score: 0.9993115663528442
Epoch:  124

100%|| 4/4 [00:18<00:00,  4.69s/it, loss=0.0118]

Got 7199741/7200000 with acc 100.00
Dice score: 0.999333381652832
Epoch:  125

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0117]

Got 7199725/7200000 with acc 100.00
Dice score: 0.9992983937263489
Epoch:  126

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0117]

Got 7199757/7200000 with acc 100.00
Dice score: 0.999372661113739
Epoch:  127

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0116]

Got 7199700/7200000 with acc 100.00
Dice score: 0.9992368817329407
Epoch:  128

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0116]

Got 7199770/7200000 with acc 100.00
Dice score: 0.999403178691864
Epoch:  129

100%|| 4/4 [00:18<00:00,  4.73s/it, loss=0.0115]

Got 7199772/7200000 with acc 100.00
Dice score: 0.9994295239448547
Epoch:  130
```

```
100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0112]

Got 7199730/7200000 with acc 100.00
Dice score: 0.9992548227310181
Epoch:  131

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0113]

Got 7199702/7200000 with acc 100.00
Dice score: 0.9992626309394836
Epoch:  132

100%|| 4/4 [00:18<00:00,  4.69s/it, loss=0.0114]

Got 7199755/7200000 with acc 100.00
Dice score: 0.9993782043457031
Epoch:  133

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0111]

Got 7199751/7200000 with acc 100.00
Dice score: 0.9993533492088318
Epoch:  134

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0111]

Got 7199768/7200000 with acc 100.00
Dice score: 0.9993847012519836
Epoch:  135

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0108]

Got 7199772/7200000 with acc 100.00
Dice score: 0.9994075894355774
Epoch:  136

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0107]

Got 7199808/7200000 with acc 100.00
Dice score: 0.9994980692863464
Epoch:  137

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0108]

Got 7199810/7200000 with acc 100.00
Dice score: 0.9995046854019165
Epoch:  138

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0107]
```

```
Got 7199778/7200000 with acc 100.00
Dice score: 0.9993840456008911
Epoch:  139

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0108]

Got 7199809/7200000 with acc 100.00
Dice score: 0.9994916915893555
Epoch:  140

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0107]

Got 7199798/7200000 with acc 100.00
Dice score: 0.9994677901268005
Epoch:  141

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0105]

Got 7199811/7200000 with acc 100.00
Dice score: 0.9994888305664062
Epoch:  142

100%|| 4/4 [00:18<00:00,  4.70s/it, loss=0.0105]

Got 7199797/7200000 with acc 100.00
Dice score: 0.9994634985923767
Epoch:  143

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0106]

Got 7199805/7200000 with acc 100.00
Dice score: 0.9994988441467285
Epoch:  144

100%|| 4/4 [00:18<00:00,  4.69s/it, loss=0.0103]

Got 7199810/7200000 with acc 100.00
Dice score: 0.9995201230049133
Epoch:  145

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0104]

Got 7199808/7200000 with acc 100.00
Dice score: 0.9994863867759705
Epoch:  146

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.0102]

Got 7199794/7200000 with acc 100.00
Dice score: 0.999434769153595
Epoch:  147
```

```
100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.01]

Got 7199804/7200000 with acc 100.00
Dice score: 0.9994713664054871
Epoch:  148

100%|| 4/4 [00:18<00:00,  4.71s/it, loss=0.01]

Got 7199818/7200000 with acc 100.00
Dice score: 0.9995250701904297
Epoch:  149

100%|| 4/4 [00:18<00:00,  4.72s/it, loss=0.0101]

Got 7199819/7200000 with acc 100.00
Dice score: 0.9995226860046387
Epoch:  150
```

[ ]:
```
torch.save(model.state_dict(), '/content/gdrive/MyDrive/Bone_Age_Detection /
↪Saved_model/model_weights.pth')
# state_dict is simply a Python dictionary object that maps each layer to its␣
↪parameter tensor.
```

[ ]:
```
torch.save(model, '/content/gdrive/MyDrive/Bone_Age_Detection /Saved_model/
↪model1.pth')
```

[ ]:
```
torch.save(optimizer.state_dict(), '/content/gdrive/MyDrive/Bone_Age_Detection /
↪Saved_model/optimizer.pth')
```

0.8 Save predictions and mask

[ ]:
```
val_transforms = A.Compose([ A.Resize(height=300, width=300),A.
↪Normalize(mean=[0.0, 0.0, 0.0], std=[1.0, 1.0, 1.0],max_pixel_value=255.
↪0),ToTensorV2()])
val_trans=T.Compose([T.ToTensor(), T.Resize(300)])


def save_pred(img_path, mask_path,folder):

        image = Image.open(img_path).convert("RGB")#
        mask = Image.open(mask_path).convert("L")#

        image=val_trans(image)#torch.Size([3, 300, 300])
        mask=val_trans(mask)#torch.Size([3, 300, 300])
        mask[mask == 255.0] = 1.0

        image, mask = image.unsqueeze(0).to(Device), mask.unsqueeze(0).
↪to(Device)
```

```
        model.eval()
        with torch.no_grad():
            preds =torch.sigmoid(model(image))
            preds = (preds > 0.5).float()
        torchvision.utils.save_image(preds, f"{folder}/pred_{i}.png")
        #torchvision.utils.save_image(image, f"{folder}{i}.png")

        model.train()
for i in range(1,12):
  save_pred(f'/content/drive/MyDrive/Bone_Age_Detection /Val_data/img_{i}.
 ↪png',f'/content/drive/MyDrive/Bone_Age_Detection /Val_masks/img_{i}.png',f'/
 ↪content/drive/MyDrive/Bone_Age_Detection /pred/')
```

0.9 Merge for a test image

[ ]:
```
def merg_img(original_image, prediction):#this are PIL images RGB
  r1,g1,b1= original_image.split()
  r2,g2,b2= prediction.split()
  new_img=Image.merge('RGB', (r2,g1,b2))
  #plt.imshow(new_img)
  new_img.save(f'/content/drive/MyDrive/Bone_Age_Detection /MP/mergedpred{1}.
 ↪png')

for i in range(1,12):
    original=Image.open(f'/content/drive/MyDrive/Bone_Age_Detection /Val_data/
 ↪img_{i}.png').convert('RGB')
    pred=Image.open(f'/content/drive/MyDrive/Bone_Age_Detection /prediction2/
 ↪pred_{1}.png').convert('RGB')

    merg_img(original, pred)
```

0.9.1 Add bounding box to merged predictions

[ ]:
```
def b_box(path1,path2):
  img_path=os.listdir(path1)
  img_path1=os.listdir(path2)
  i=0
  for j in range(len(img_path)):
    i+=1
    image = cv2.imread(path1+'/'+img_path[j])
    img_to_add_bbox=cv2.imread(path2+'/'+img_path1[j])
    gray =cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    gray1 =cv2.cvtColor(img_to_add_bbox, cv2.COLOR_BGR2GRAY)#
    ret,binary = cv2.threshold(gray,127,255,cv2.THRESH_BINARY)
```

```
    contours,hierarchy = cv2.findContours(binary,cv2.RETR_EXTERNAL,cv2.
↪CHAIN_APPROX_SIMPLE)#contours gives dim for each element in prediction(each␣
↪bone)
    boxes = []
    for c in contours:
        (x, y, w, h) = cv2.boundingRect(c)
        boxes.append([x,y, x+w,y+h])

    boxes = np.asarray(boxes)
    left, top = np.min(boxes, axis=0)[:2]
    right, bottom = np.max(boxes, axis=0)[2:]
    cv2.rectangle(img_to_add_bbox, (left,top), (right,bottom), (255, 0, 0), 2)
    crop_img=img_to_add_bbox[top:bottom,left:right]

    #save images with bounding box
    cv2.imwrite(f'/content/gdrive/MyDrive/Bone_Age_Detection /bounded_ped/
↪bbox{i}.png',img_to_add_bbox)
    #save cropped images
    cv2.imwrite(f'/content/gdrive/MyDrive/Bone_Age_Detection /cropped_pred/
↪crop_img{i}.png',crop_img)


path1='/content/gdrive/MyDrive/Bone_Age_Detection /prediction2'#path for␣
↪prediction obtained from val_data
path2='/content/gdrive/MyDrive/Bone_Age_Detection /MP'#path for merged␣
↪predictions and original val_img
#b_box(path1,path2)
```

0.10 Export to ONNX

[ ]:
```
!pip install onnx onnxruntime
```

```
Collecting onnx
  Downloading onnx-1.10.1-cp37-cp37m-
manylinux_2_12_x86_64.manylinux2010_x86_64.whl (12.3 MB)
     || 12.3 MB 4.2 MB/s
Collecting onnxruntime
  Downloading onnxruntime-1.9.0-cp37-cp37m-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (4.8 MB)
     || 4.8 MB 40.6 MB/s
Requirement already satisfied: numpy>=1.16.6 in /usr/local/lib/python3.7
/dist-packages (from onnx) (1.19.5)
Requirement already satisfied: typing-extensions>=3.6.2.1 in
/usr/local/lib/python3.7/dist-packages (from onnx) (3.7.4.3)
Requirement already satisfied: protobuf in /usr/local/lib/python3.7/dist-
packages (from onnx) (3.17.3)
```

```
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages
(from onnx) (1.15.0)
Requirement already satisfied: flatbuffers in /usr/local/lib/python3.7/dist-
packages (from onnxruntime) (1.12)
Installing collected packages: onnxruntime, onnx
Successfully installed onnx-1.10.1 onnxruntime-1.9.0
```

[ ]:
```
batch= next(iter(val_loader)) #Each iteration below returns a batch of␣
↪train_features and train_labels (containing batch_size features and labels␣
↪respectively). Because we specified shuffle=True, after we iterate over all␣
↪batches the data is shuffled
image, label = batch
image=image.to(Device)
```

[ ]:
```
import torch.onnx
import onnx
from onnx import version_converter, helper
batch_size=20
#model_path='/content/gdrive/MyDrive/Bone_Age_Detection /Saved_model/model1.
↪pth'
#model=torch.load(model_path)
#model.eval()

output=model(image)
torch.onnx.export(model,image, '/content/gdrive/MyDrive/Bone_Age_Detection /
↪export3.onnx',
                  export_params=True, opset_version=11,
                  do_constant_folding=True,
                  input_names = ['input'],
                  output_names = ['output'],
                  dynamic_axes={'input' : {0 : 'batch_size'},'output' : {0 :␣
↪'batch_size'}})


onnx_model = onnx.load("/content/gdrive/MyDrive/Bone_Age_Detection /export3.
↪onnx")
onnx.checker.check_model(onnx_model)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:62: TracerWarning:
Converting a tensor to a Python boolean might cause the trace to be incorrect.
We can't record the data flow of Python values, so this value will be treated as
a constant in the future. This means that the trace might not generalize to
other inputs!
```

[ ]:
```

```

```
batch= next(iter(val_load)) #Each iteration below returns a batch of␣
↪train_features and train_labels (containing batch_size features and labels␣
↪respectively). Because we specified shuffle=True, after we iterate over all␣
↪batches the data is shuffled
image, label = batch
image=image.to(Device)
img=image
```

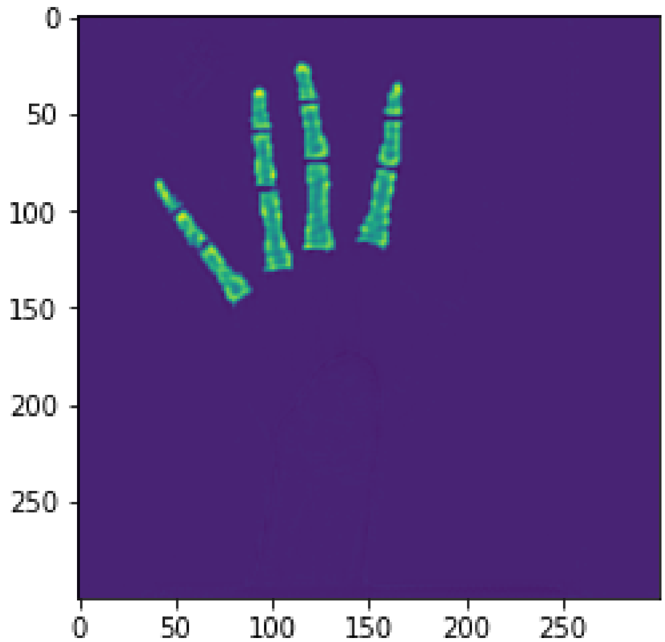
[ ]:
```
import onnxruntime

ort_session = onnxruntime.InferenceSession("/content/gdrive/MyDrive/
↪Bone_Age_Detection /export.onnx")
def to_numpy(tensor):
    return tensor.detach().cpu().numpy() if tensor.requires_grad else tensor.
↪cpu().numpy()

# compute ONNX Runtime output prediction
ort_inputs = {ort_session.get_inputs()[0].name: to_numpy(img)}
ort_outs = ort_session.run(None, ort_inputs)

# compare ONNX Runtime and PyTorch results
np.testing.assert_allclose(to_numpy(output), ort_outs[0], rtol=1e-01,␣
↪atol=1e-01)

print("Exported model has been tested with ONNXRuntime, and the result looks␣
↪good!")
```

```
Exported model has been tested with ONNXRuntime, and the result looks good!
```

[ ]:
```
img_out_y = ort_outs[0]
img_out_y=img_out_y.squeeze().squeeze()
print(img_out_y.shape)
plt.imshow(img_out_y)
```

```
(300, 300)
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7f128c4e5350>
```



[10]:
```
import os
```

[13]:
```
os.getcwd()
```

[13]: '/content/drive/MyDrive'

[12]:
```
os.chdir('/content/drive/MyDrive')
```

[21]:
```
!jupyter nbconvert --to PDF "Bone_Age_Detection_model.ipynb"
```

```
[NbConvertApp] Converting notebook Bone_Age_Detection_model.ipynb to PDF
[NbConvertApp] Support files will be in Bone_Age_Detection_model_files/
[NbConvertApp] Making directory ./Bone_Age_Detection_model_files
[NbConvertApp] Making directory ./Bone_Age_Detection_model_files
[NbConvertApp] Writing 117777 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: [u'xelatex', u'./notebook.tex',
'-quiet']
[NbConvertApp] Running bibtex 1 time: [u'bibtex', u'./notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 111308 bytes to Bone_Age_Detection_model.pdf
```