Report for Part II

Team: Hao-Yu Liao, Shuaizhou Hu, Xinyao Zhang

- **Instruction on running the programs**

The 'ReadMeFirst.txt' file explains contents of each file and fold. The 'Programs' file contains all codes for each model. The 'TrainedModel' file contains parameters which lead to the best performance of each model. The ' TrainTestImagesRes' file contains all image results of each model. The 'TrainTestLog' file contains all MSE results of each model. 6 models are trained as the following description:

1. Regressor.ipynb: Regressor model to output mean of a* and b*.
2. ColorizingRelu.ipynb: Colorizing model to output colorizing images from gray images. by ReLU function in last layer.
3. ColorizingSigmoid.ipynb: Colorizing model to output colorizing images from gray images. by Sigmoid function in last layer.
4. ColorizingTanh.ipynb: Colorizing model to output colorizing images from gray images. by Tanh function in last layer.
5. RegressorColorizing.ipynb: Combining regressor and Colorizing model together. The input is L*, mean of scalar a* and b*. The output is matrix a* and b* for colorizing images.
6. ColorizingSigmoidChangeFM.ipynb: Changing different number of feature maps as the best architecture by using Sigmoid function in the last layer in this study.

- **Implementation the programs**

We will discuss how to implement GPU programs to train the deep learning models.

1. **Set Up Code Base**

   We use Anaconda 64-bit to build the environment for applying CUDA to run GPU and use the Jupyter notebook from Anaconda to build PyTorch programs. We recommend using Anaconda with the Jupyter notebook to run our programs with the same environmental setting.

2. **Load the Dataset**

   Load the image dataset, set the default Torch datatype to 32-bit float, and randomly shuffle the data

```
dataBGR = np.array(dataBGR, dtype = np.float32) #Change data type into float 32 in whole program.
dataLAB = np.array(dataLAB, dtype = np.float32) #Change data type into float 32 in whole program.
```

```
dataLAB = dataLAB / 255 ##Change reange into [0, 1]
dataLAB = np.moveaxis(dataLAB, -1, 1) #Reshape channeL from [B, H, W, C] to [B, C, H, W]
dataLAB = dataLAB[torch.randperm(dataLAB.shape[0],generator=torch.random.manual_seed(42))] #shuffle with random seed
lengths = [int(len(dataLAB)*0.9), int(len(dataLAB)*0.1)]
trainLAB, testLAB = random_split(dataLAB, lengths ,generator=torch.random.manual_seed(42)) #Shuffle data with random
trainLAB = np.array(trainLAB)
testLAB = np.array(testLAB)
```

3. **Augment your dataset**

   Three "augmentation" operations, including horizontal flips, random crops and scalings, to be applied to training data. The number of training data will be from 675 to 6750.

```
transform = torchvision.transforms.Compose([ #Data aguement.
    torchvision.transforms.ToPILImage(),
    torchvision.transforms.RandomHorizontalFlip(), #Randomly Horizontal Flip.
    torchvision.transforms.RandomResizedCrop((128,128),scale=(0.6, 1.0)), #Scaling between [0.6 1.0] and randomly crop.
    #scale (tuple of python:float) – scale range of the cropped image before resizing, relatively to the origin image from pytorch office.
    torchvision.transforms.ToTensor()
])
trainLABTensorx10Aug = augementData(trainLABTensorx10,transform) #Augement train data.
```

   In order to reduce overfitting, torch Tensor class that is 10x larger along the first dimension as the original image dataset, but has the same sizes for the other dimensions.

```
trainLABTensor = torch.tensor(trainLAB)
trainLABTensorx10 = torch.clone(trainLABTensor)
for i in range(9): #Increasing training data from 675 to 6750.
    trainLABTensorx10 = torch.cat((trainLABTensorx10, torch.clone(trainLABTensor)), 0)
```

4. **Convert your images to L ∗ a ∗ b∗ color space**

   Convert images into L*a*b color space and save them.

```
img_dir = "./face_images/"
_, _, files = next(os.walk(img_dir)) #Get each image's path.
dataBGR = []
dataLAB = []
for f1 in files:
    img = cv2.imread(img_dir+f1)
    dataBGR.append(img) #Cover into BGR and save.
    imageLAB = cv2.cvtColor(img, cv2.COLOR_BGR2LAB) #Cover into LAB and save.
    dataLAB.append(imageLAB)
```

5. **Build a simple regressor**

   Input is only the L∗ channel and output is mean a∗ and mean b*.

```
trainLx10 = [] #Input L*
trainAx10 = [] #Matrix a* #Label
trainBx10 = [] #Matrix b* #Label
trainAvg_ax10 = [] #scalar mean a* #Label
trainAvg_bx10 = [] #scalar mean b* #Label
i = 0
for t in trainLABTensorx10Aug:
    meanA = torch.mean(t[1]) #Get mean of a*.
    meanB = torch.mean(t[2]) #Get mean of b*.
```

   There are 7 modules, where each module consists of a SpatialConvolution layer followed by the hidden layer=3, stridea =1, padding =1, and ReLU activation function. The sizes of the images as they go through the CNN are decreasing as 128,64,32,16, 8, 4, 2.

```
# Defining 1st 2D convolution layer
Conv2d(1, 3, kernel_size=3, stride=1, padding=1), #128@3
BatchNorm2d(3), #Batch normailzation.
ReLU(inplace=True),
MaxPool2d(kernel_size=2, stride=2),
```

6. **Colorize the image**

   There are 5 downsampling layers and set the initial spatial resolutions at different parts of the CNN as 128,64,32,16,8,8,16,32,64,128 (N=5).

```
netClo = Sequential( #Define  colorizing model.
            # Defining 1st 2D convolution layer
            Conv2d(3, 3, kernel_size=3, stride=1, padding=1), #128@3
            BatchNorm2d(3), #Batch normailzaiton.
            ReLU(inplace=True),
            MaxPool2d(kernel_size=2, stride=2),
```

7. **Batch Normalization**

   Inserte batch normalization directly after each SpatialConvolution layer. Also, divide training dataset into mini-batches of 100 images each.

8. **GPU computing**

   Speed up the network by using the GPU.

```
netReg = netReg.cuda() #Apply GPU.
```

```
inputs, labels = data[0].cuda(), data[1].cuda() #Apply GPU in computing.
```

9. **Training**

   90% of the images can be in the training part, and 10% of the images can be in the testing part. To evaluate the test images, calculate a numerical mean square error value.

```
Train epoch: 1, MSE_Loss: 0.0163954084
Train epoch: 2, MSE_Loss: 0.0002786994
Train epoch: 3, MSE_Loss: 0.0002475026
Train epoch: 4, MSE_Loss: 0.0002318625
Train epoch: 5, MSE_Loss: 0.0002237538
```

10. **Testing**

75 images will be used for testing results. Three results including ground truth, colorizing, and grayscale will be outputted.

Ground truth

Colorizing from grayscale

Grayscale

- **Evaluation results**

We train model with 1000 epoch. The stochastic gradient descent is used to find the best parameters, the learn rate is 0.1, and the momentum is 0.9. The size of mini bath is 100. Therefore, the program will run 68 times for each epoch because of 6750 train samples due to augment. Also, we make sure training samples are the same for different models.

1. **Regressor results**

Train epoch: 1	MSE: 0.0163966381
Train epoch from 2 - 999	...
Train epoch: 1000	MSE: 0.0001169633
Test: 75 samples	MSE: 0.0001194611

2. **Colorizing with a Relu activation function as last layer**

Train epoch: 1	MSE: 0.0221731979
Train epoch from 2 - 999	...
Train epoch: 1000	MSE: 0.0003357466
Test: 75 samples	MSE: 0.0004444391

Prediction image using test dataset (ReLU)

3. **Colorizing with a Sigmoid activation function as last layer**

Train epoch: 1	MSE: 0.0020534776
Train epoch from 2 - 999	...
Train epoch: 1000	MSE: 0.0003374989
Test: 75 samples	MSE: 0.0004205852

Prediction image using test dataset (Sigmoid)

In conclusion, colorizing with a Sigmoid activation function has the better performance than colorizing with a Relu activation function. As the SIGGRAPH [1] paper mentioned, they used mean square loss function to train the model. Also, they assigned each layer with ReLU function, and output layer is Sigmoid function. Therefore, comparing the ReLU and Sigmoid function in Section, the Sigmoid has the better performance.

- **Extra Credit**

1. **Colorizing with a Tanh activation function**

Train epoch: 1	MSE: 0.0069492882
Train epoch from 2 - 999	...
Train epoch: 1000	MSE: 0.0013039349
Test: 75 samples	MSE: 0.0016880839

Prediction image using test dataset (Tanh)

In conclusion, colorizing with a Tanh activation function is not good than colorizing with a ReLU or Sigmoid activation functions as shown in the previous section. The output range of Tanh activation is -1 to 1 in the last layer. However, all the previous layers apply ReLU activation function with output range 0 to 1 for L* input. We think if we use

both ReLU activation function in the previous layer and Tanh activation function in the last layer, the results will be poor performance.

2. Changing the number of feature maps

The previous 2D convolution layer is defined as 'Conv2d(3, 3, kernel_size=3, stride=1, padding=1)' in all layers. That means that each layer has 3 feature maps. However, after changing the number of feature maps, the first 2D convolution layer is defined as 'Conv2d(1, 64, kernel_size=3, stride=1, padding=1)'. We change channels from 1 to 64. Then, the second 2D convolution layer is defined as 'Conv2d(64, 128, kernel_size=3, stride=1, padding=1)'. And so on. The below code figure shows the overall changing. The only thing is that we change the number of feature maps, but we still have the same number of hidden layers. We review the paper [2] to change the number of feature maps.

```
= Sequential(  #Define the model. The number of each layer's feature maps is different.
        # Defining 1st 2D convolution layer
        Conv2d(1, 64, kernel_size=3, stride=1, padding=1), #128@3
        BatchNorm2d(64), #Batch normailzation.
        ReLU(inplace=True),
        MaxPool2d(kernel_size=2, stride=2),
        # Defining 2nd 2D convolution layer
        Conv2d(64, 128, kernel_size=3, stride=1, padding=1), #64@3
        BatchNorm2d(128),
        ReLU(inplace=True),
        MaxPool2d(kernel_size=2, stride=2),
        # Defining 3rd 2D convolution layer
        Conv2d(128, 256, kernel_size=3, stride=1, padding=1), #32@3
        BatchNorm2d(256),
        ReLU(inplace=True),
        MaxPool2d(kernel_size=2, stride=2),
        # Defining 4th 2D convolution layer
        Conv2d(256, 512, kernel_size=3, stride=1, padding=1), #16@3
        BatchNorm2d(512),
        ReLU(inplace=True),
        MaxPool2d(kernel_size=2, stride=2),
        ConvTranspose2d(512, 512, 4, stride=2, padding=1), #@16@3
        BatchNorm2d(512),
        ReLU(inplace=True),
        ConvTranspose2d(512, 256, 4, stride=2, padding=1), #@32@3
        BatchNorm2d(256),
        ReLU(inplace=True),
        ConvTranspose2d(256, 64, 4, stride=2, padding=1), #@64@3
        BatchNorm2d(64),
        ReLU(inplace=True),
        ConvTranspose2d(64, 2, 4, stride=2, padding=1), #@128@3
        Sigmoid()
    )
```

	After changing feature maps	Before changing feature maps
Train epoch: 1	MSE: 0.0071906213	MSE: 0.0020534776
Train epoch from 2 - 999	...	...
Train epoch: 1000	MSE: 0.0001394850	MSE: 0.0003374989
Test: 75 samples	MSE: 0.0002738693	MSE: 0.0004205852

Input grayscale Image

Prediction image using test dataset (After changing feature maps)

Prediction image using test dataset (Before changing feature maps)

True label image

After Changing feature maps, the performance is better. The best model can predict the color skin more accurately.

3. Paper reading

Colorization using a regressor tends to look more desaturated than using a classifier. Adopting loss functions inherited from standard regression problems will cause predict conservatively [3], [4], which is aimed to minimize Euclidean error between an estimate and the ground truth. And to solve the Euclidean loss optimally is using the mean of the color space set, this average effect makes results grayish and desaturated.

Using a classifier to predict a distribution of possible colors for each pixel can model the multimodal nature [5] of the colorization problem appropriately. And this model tends to utilize the full diversity of the large-scale training data. Therefore, it can produce better colorization results that are more vivid and sensually realistic [2]. The result of our project also proves this conclusion.

- **Discussion**

In this study, we compare different activation functions including Sigmoid, ReLU, and Tanh in the last layer. We found with the Sigmoid activation function the model can perform better than other activation functions. Also, we can use large learning rate to train the model because of bath normalization. In addition, the best model is the Sigmoid function with changing different number of feature maps.

- **Reference:**

[1] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Let There Be Color! Joint End-to-End Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification," *ACM Trans. Graph.*, vol. 35, no. 4, Jul. 2016, doi: 10.1145/2897824.2925974.

[2] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *European conference on computer vision*, 2016, pp. 649–666.

[3] Z. Cheng, Q. Yang, and B. Sheng, "Deep colorization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 415–423.

[4] R. Dahl, "Automatic colorization." 2016.

[5] G. Charpiat, M. Hofmann, and B. Schölkopf, "Automatic image colorization via multimodal predictions," in *European conference on computer vision*, 2008, pp. 126–139.