

Assignment - 1

Elkhan Ismayilzada

Solution for Problem 1:

To meet the conditions stated in Problem 1, I utilized the OpenCV package for clock generation. First I made an empty image and filled it with a random background RGB color generated from the Numpy random package. Next, using the circle function of OpenCV, I drew a circle and filled it with a random color. I used the putText function of OpenCV to write the clock numbers on the image with recommended font properties. Leveraging the degree and radius property of the circle, I located the respective position of each number on the image. Following equation is used for positions:

$$x = \text{center}(x) + \text{radius} * \cos(\text{degree}), y = \text{center}(y) + \text{radius} * \sin(\text{degree}).$$

As we have 12 numbers for an hour, neighbor numbers are separated by $360 / 12 = 30$. The same way, clock arms are positioned. Leveraging the randn function of OpenCV package, I added a small noise with mean 0 and standard deviation of 50. Noise values are in $[0, 255]$ range.

Solution for Problem 2:

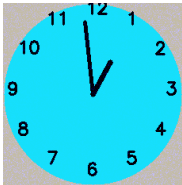
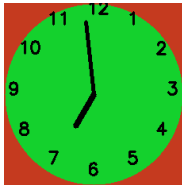
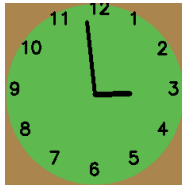
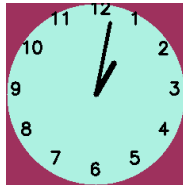
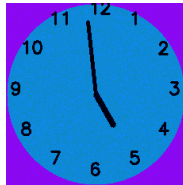
For predicting the hour and minute values from a given image, I employed the imagenet pretrained ResNet18. I attached two Linear layers on top of the last layer of ResNet18 which output one value i.e., hour and minute respectively as shown here.

```
class TimePredictor(nn.Module):
    def __init__(self):
        super(TimePredictor, self).__init__()
        self.model = torchvision.models.resnet18(pretrained=True)
        self.hour = nn.Linear(self.model.fc.out_features, 1)
        self.minute = nn.Linear(self.model.fc.out_features, 1)
```

Solution for Problem 3:

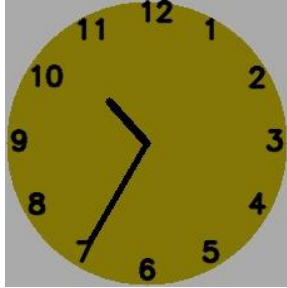
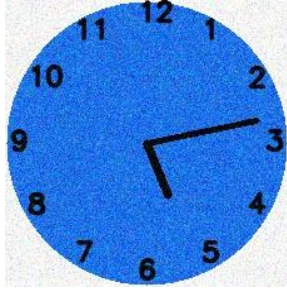
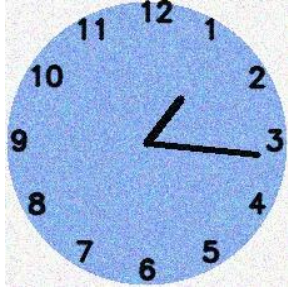
To train our model, I made a dataset loader class. Unlike the traditional way, I yield the images for the train data on the spot. In detail, when I get a request for an image of a certain hour and minute, I call my clock image generation function and return it. This way, I made sure that our model sees every possible clock scenario with random colors and noise during training. Also, as a ground truth label, since we have 720 ($60 * 12$) scenarios, I made sure all of these labels exist during the training. For validation, however, I generated 14400 images that encompass all labels with as many color and noise variations as possible. As a loss function, I employed L1 loss and as an optimizer, I utilized Adam optimizer with a learning rate to be $3e-5$. Before performing the loss, I performed sigmoid on the output of our model. After this, I unnormalize sigmoid values to hour and minute (i.e., by multiplying 12 and 60, respectively) and perform L1 loss between unnormalized sigmoid values and ground truth labels. I trained for around 20 epochs with a

batch size of 8 and obtained around 97% accuracy on my validation dataset. During testing, I simply unnormalized the sigmoid values and rounded. Here are some images in which my model outputs the wrong value:

Image					
Prediction	11h 53m	7h 54m	2h 53m	2h 4m	5h 53m
Ground truth	12h 59m	6h 59m	2h 59m	1h 2m	4h 59m

As can be seen from the examples, it seems that our model struggles to predict accurately when the minute arm is around the 12th tick.

I tested on the example images given by the Professor and I got the following results.

Image			
Prediction	10h 35m	5h 13m	1h 16m
Ground truth	10h 35m	5h 13m	1h 16m

Solution for Problem 4:

My test function accepts an image folder as an argument and prints the output along with the respective image. Additionally, I save the results in a json file (results.json). Here is an example to run my test function:

```
python Prob4.py --folder test_dataset/
```