Gray to Color

CS 520 Assignment-4

Haotian Xu(hx105), Chaoji Zuo(cz296), Xuenan Wang(xw336)

Spring 2019

Gray to Color

CS 520 Assignment-4

1. Representing The Process

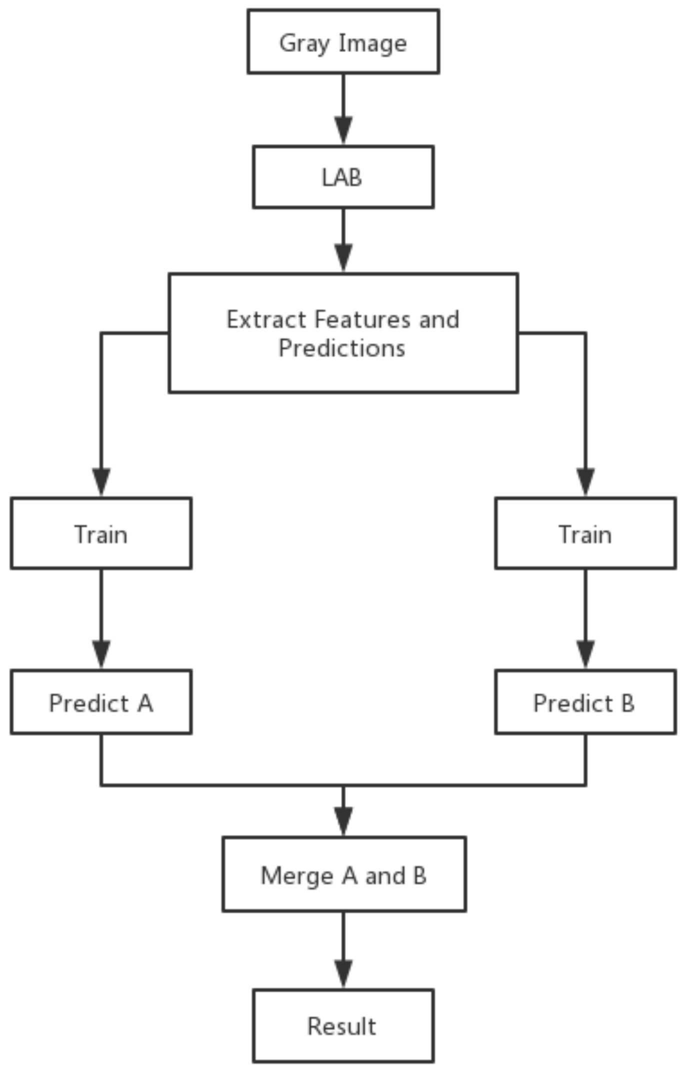


Figure.1 Flow Chart of Whole Algorithm

2. Data

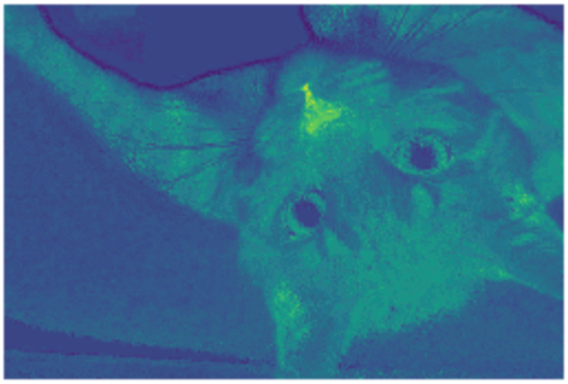
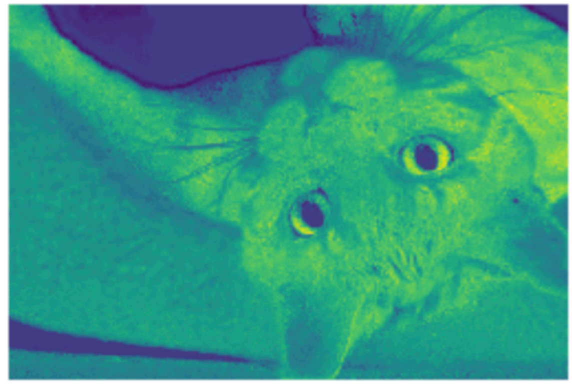
Input data is a RGB image which is actually a 3-D metric. Take Figure 2 for example, channel Red is shown as Figure 3(left), channel Green is shown as Figure 3(middle) and channel Blue is shown as Figure 3(right).

Figure.2 Color Image(Left) v.s. Gray Image(Right)

Figure.3 Red Channel(Left), Green Channel(Middle), Blue Channel(Right)

And we found that if we use LAB color space, we just need to recolor 2 channel a (from green to red) and b (from blue to yellow). In this way, we can do less predication.

Figure.4 a channel(left), b channel(right)

3. Evaluating The Model

To evaluating the model, the most directly method is watch the output, if a people think the output picture is real, it would be successful. And we also came up with a more mathematical way to evaluating the output, calculate the 2 norm of the subtract of true picture and recolor picture.

4. Training The Model

We have different approaches so we have different models, and we have

5. Assessing The Final Project

Text goes here.

6. Bonus

Text goes here.

Other Thoughts:

For coloring an image, we as humans will first recognize the which kind of objects that are represented in the image then do the coloring. Therefore, it may help to do a better job if computer first recognize these images.

The reason we think image recognition might work is that it will provide more features value and increase the accuracy for the coloring. We first take grey scale and RGB channel value as our feature vectors. If we import another feature vector which would be edges, then we basically doing the recognition and color at the same time. However, edges are not like color features. If we take the binary image, we might result it a bunch white area and black area; therefore, we also need to do image sharpening.

In image recognition-coloring, it might be better if we keep the constancy of our image objects, which means we should take those objects with common colors and shape. For example, a tree would be a great train data, but cars might be not.

Source Code

```
#!/Usr/Bin/Env Python3
# -*- Coding: Utf-8 -*-
"""
Created On Mon Mar 25 22:49:49 2019

@Author: Himmel
"""

Import Random
Import Numpy As Np
Import Matplotlib.Pyplot As Plt

Class Map:
    Def __Init__(Self, Size):
```

```
        Self.Size = Size;
        Self.Terrain = {}
        For X In Range(Self.Size):
            For Y In Range(Self.Size):
                Self.Terrain[(X, Y)] = 0
        Self.Target = (0,0)
        Self.Typereport = ("", "")

    Def Generatemap(Self):
        Locations = List(Self.Terrain.Keys())
        Random.Shuffle(Locations)
        For Coordinate In Locations:
            Probability = Random.Random()
            If Probability <= 0.2:
                Self.Terrain[Coordinate] = 1000 #Flat
            Elif Probability <= 0.5:
                Self.Terrain[Coordinate] = 800 #Hilly
            Elif Probability <= 0.8:
                Self.Terrain[Coordinate] = 500 #Forested
            Else:
                Self.Terrain[Coordinate] = 0   #Cave
        Random.Shuffle(Locations)
        Self.Target = Random.Choice(Locations)

    Def Printmap(Self):
        Graph = Np.Zeros((Self.Size, Self.Size), Dtype =
Int)
        For (X, Y) In Self.Terrain.Keys():
            Graph[X, Y] = Self.Terrain[(X, Y)]
        Plt.Figure(Figsize=(7.5,7.5))

Plt.Pcolor(Graph[::-1],Edgecolors='Black',Cmap='Gist
_Earth', Linewidths=2)
        Plt.Xticks([]), Plt.Yticks([])
        Plt.Tight_Layout()
        Plt.Show()

    Def Targetmove(Self):
        Choice = []
        (X, Y) = Self.Target
        If X - 1 >= 0:
            Choice.Append((X - 1, Y))
        If Y - 1 >= 0:
            Choice.Append((X, Y - 1))
        If X + 1 < Self.Size:
            Choice.Append((X + 1, Y))
        If Y + 1 < Self.Size:
            Choice.Append((X, Y + 1))
        Self.Target = Random.Choice(Choice)
        Self.Typereport = (Self.Terrain[(X, Y)],
Self.Terrain[Self.Target])

Class Searchrobot:
    Def __Init__(Self, Map):
        Self.Map = Map
        Self.Belief = {} #Containing Belief
        Self.Choices = []
        Self.Probability = {} #Finding Probability
```

```
        For (X, Y) In Self.Map.Terrain.Keys():
            Self.Belief[(X, Y)] = 1/(Self.Map.Size ** 2)
        Self.Observation = ()
        Self.Ptype = 0

    Def Randompick(Self):
        '''
        Choosing The Locations With The Highest
Probability Of Having Target.
        Then Take The Best Terrain(Order: Flat, Hilly,
Forest, Cave).
        Randomly Picking One From The These Choices
        '''
        Highestprobability = Max(Self.Belief.Values())
        Choices = {}
        For Coordinate In Self.Belief.Keys():
            If Self.Belief[Coordinate] ==
Highestprobability:
                Choices[Coordinate] =
Self.Map.Terrain[Coordinate]
        Betterchoices = []
        Terrain = Max(Choices.Values())
        For Coordinate In Choices.Keys():
            If Choices[Coordinate] == Terrain:
                Betterchoices.Append(Coordinate)
        Return Random.Choice(List(Choices.Keys()))

    Def Specificpick(Self):
        '''
        Choose A Certain Kind Of Area, Based On The
Type Report.
        Pick The Location With Highest Probability
        '''
        If Self.Map.Typereport == ("",""):
            Return Self.Randompick()
        (Origination, Destination) = Self.Map.Typereport
        Choices = {}
        Self.Choices = []
        For Coordinate In Self.Belief.Keys():
            If Self.Map.Terrain[Coordinate] ==
Destination:
                (X, Y) = Coordinate
                If Self.Choices == []:
                    If X - 1 >= 0 And Self.Map.Terrain[(X -
1, Y)] == Origination:
                        Choices[Coordinate] =
Self.Belief[Coordinate]
                    Elif Y - 1 >= 0 And Self.Map.Terrain[(X,
Y - 1)] == Origination:
                        Choices[Coordinate] =
Self.Belief[Coordinate]
                    Elif X + 1 < Self.Map.Size And
Self.Map.Terrain[(X + 1, Y)] == Origination:
                        Choices[Coordinate] =
Self.Belief[Coordinate]
                    Elif Y + 1 < Self.Map.Size And
Self.Map.Terrain[(X, Y + 1)] == Origination:
```

```
                Choices[Coordinate] =
Self.Belief[Coordinate]
            Else:
                If (X - 1, Y) In Self.Choices:
                    Choices[Coordinate] =
Self.Belief[Coordinate]
                Elif (X, Y - 1) In Self.Choices:
                    Choices[Coordinate] =
Self.Belief[Coordinate]
                Elif (X + 1, Y) In Self.Choices:
                    Choices[Coordinate] =
Self.Belief[Coordinate]
                Elif (X, Y + 1) In Self.Choices:
                    Choices[Coordinate] =
Self.Belief[Coordinate]
        Self.Choices = List(Choices.Keys())
        Return Random.Choice(Self.Choices)


    Def Search(Self, Location):
        '''
        Pick A Location With Highest Probability And
Search It.
        Give The False Negative Table To Generate
Certain Feeback
        '''
        (X, Y) = Location
        Self.Observation = (X, Y)
        Probability = Random.Random()
        If Self.Map.Terrain[(X, Y)] == 1000:
            Self.Ptype = 0.1
            If Probability <= 0.1 Or (X, Y) !=
Self.Map.Target:
                Return "Failure"
            Else:
                Return "Success"
        Elif Self.Map.Terrain[(X, Y)] == 800:
            Self.Ptype = 0.3
            If Probability <= 0.3 Or (X, Y) !=
Self.Map.Target:
                Return "Failure"
            Else:
                   Return "Success"
        Elif Self.Map.Terrain[(X, Y)] == 500:
            Self.Ptype = 0.7
            If Probability <= 0.7 Or (X, Y) !=
Self.Map.Target:
                Return "Failure"
            Else:
                Return "Success"
        Else:
            Self.Ptype = 0.9
            If Probability <= 0.9 Or (X, Y) !=
Self.Map.Target:
                Return "Failure"
            Else:
                Return "Success"

    Def Stationarysearch(Self):
        '''
        If Agent Currently Does Not Give Us The
Positive(Success) Feeback, Keep Searching
        Updating The Belief Of Each Location During
The Searching Process
        '''
        Number = 0
        While Self.Search(Self.Randompick()) !=
"Success":
            Denominator = 1 -
Self.Belief[Self.Observation] +
Self.Belief[Self.Observation] * Self.Ptype
            For Coordinate In Self.Belief.Keys():
                If Coordinate == Self.Observation:
                    If Self.Map.Terrain[Coordinate] == 1000:
                        Self.Belief[Coordinate] *= 0.1
                    Elif Self.Map.Terrain[Coordinate] ==
800:
                        Self.Belief[Coordinate] *= 0.3
                    Elif Self.Map.Terrain[Coordinate] ==
500:
                        Self.Belief[Coordinate] *= 0.7
                    Else:
                        Self.Belief[Coordinate] *= 0.9
                Self.Belief[Coordinate] =
Self.Belief[Coordinate]/Denominator
            Number += 1
        Print(Self.Observation)
        If Self.Map.Terrain[Self.Map.Target] == 1000:
            Print("Target Is At Flat Terrain")
        Elif Self.Map.Terrain[Self.Map.Target] == 800:
            Print("Target Is At Hilly Terrain")
        Elif Self.Map.Terrain[Self.Map.Target] == 500:
            Print("Target Is At Forested Terrain")
        Else:
            Print("Target Is In A Cave")
        Print("Congratulations! You've Found The
Target!")
        Return Number

    Def Movingsearch(Self):
        '''
        Each Search-Failure Will Make The Target
Moving Into A Neighbor Location
        The New Probabilities Will Be Updated The
Same As Stationary Search
        '''
        Number = 0
        While Self.Search(Self.Specificpick()) != "Success":
            Denominator = 1 -
Self.Belief[Self.Observation] +
Self.Belief[Self.Observation] * Self.Ptype
            For Coordinate In Self.Belief.Keys():
                If Coordinate == Self.Observation:
                    If Self.Map.Terrain[Coordinate] == 1000:
                        Self.Belief[Coordinate] *= 0.1
```

```
                Elif Self.Map.Terrain[Coordinate] ==
800:
                    Self.Belief[Coordinate] *= 0.3
                Elif Self.Map.Terrain[Coordinate] ==
500:
                    Self.Belief[Coordinate] *= 0.7
                Else:
                    Self.Belief[Coordinate] *= 0.9
            Self.Belief[Coordinate] =
Self.Belief[Coordinate]/Denominator
            Self.Map.Targetmove()
            Number += 1
        Print(Self.Observation)
        If Self.Map.Terrain[Self.Map.Target] == 1000:
            Print("Target Is At Flat Terrain")
        Elif Self.Map.Terrain[Self.Map.Target] == 800:
            Print("Target Is At Hilly Terrain")
        Elif Self.Map.Terrain[Self.Map.Target] == 500:
            Print("Target Is At Forested Terrain")
        Else:
            Print("Target Is In A Cave")
        Print("Congratulations! You've Found The
Target!")
        Return Number

    Def Regression(Self):
        For Coordinate In Self.Belief.Keys():
            Self.Belief[Coordinate] = 1/(Self.Map.Size ** 2)


M = Map(50)
Stationary = []
Moving = []
'''
For I In Range(50):
    M.Generatemap()
    Agent = Searchrobot(M)
    Stationary.Append(Agent.Stationarysearch())
    #Agent.Regression()
    #Moving.Append(Agent.Movingsearch())

Mean= Sum(Stationary) // 50
Std = 0
For Number In Stationary:
    Std += (Number - Mean) ** 2
Std = (Std / 50)** 0.5

Normaldistribution = Np.Random.Normal(Mean, Std,
100000)
Plt.Hist(Normaldistribution, Bins=100, Normed=True)
Plt.Show()
Plt.Xlabel("The Order Of Experiment")
Plt.Ylabel("The Number Of Steps To Find Target")
Plt.Plot(Range(1,51), Stationary)
'''
M.Generatemap()
For I In Range(100):
    Agent = Searchrobot(M)
#Stationary.Append(Agent.Stationarysearch())
    Moving.Append(Agent.Movingsearch())
Mean = Sum(Moving) // 100
Std = 0
For Number In Moving:
    Std += (Number - Mean) ** 2
Std = (Std / 100)** 0.5
Normaldistribution = Np.Random.Normal(Mean, Std,
100000)
Plt.Hist(Normaldistribution, Bins=100, Normed=True)
Plt.Show()

Plt.Xlabel("The Order Of Experiment")
Plt.Ylabel("The Number Of Steps To Find Target")
Plt.Plot(Range(1,101), Moving)
```