CS440 Project 4: Colorization

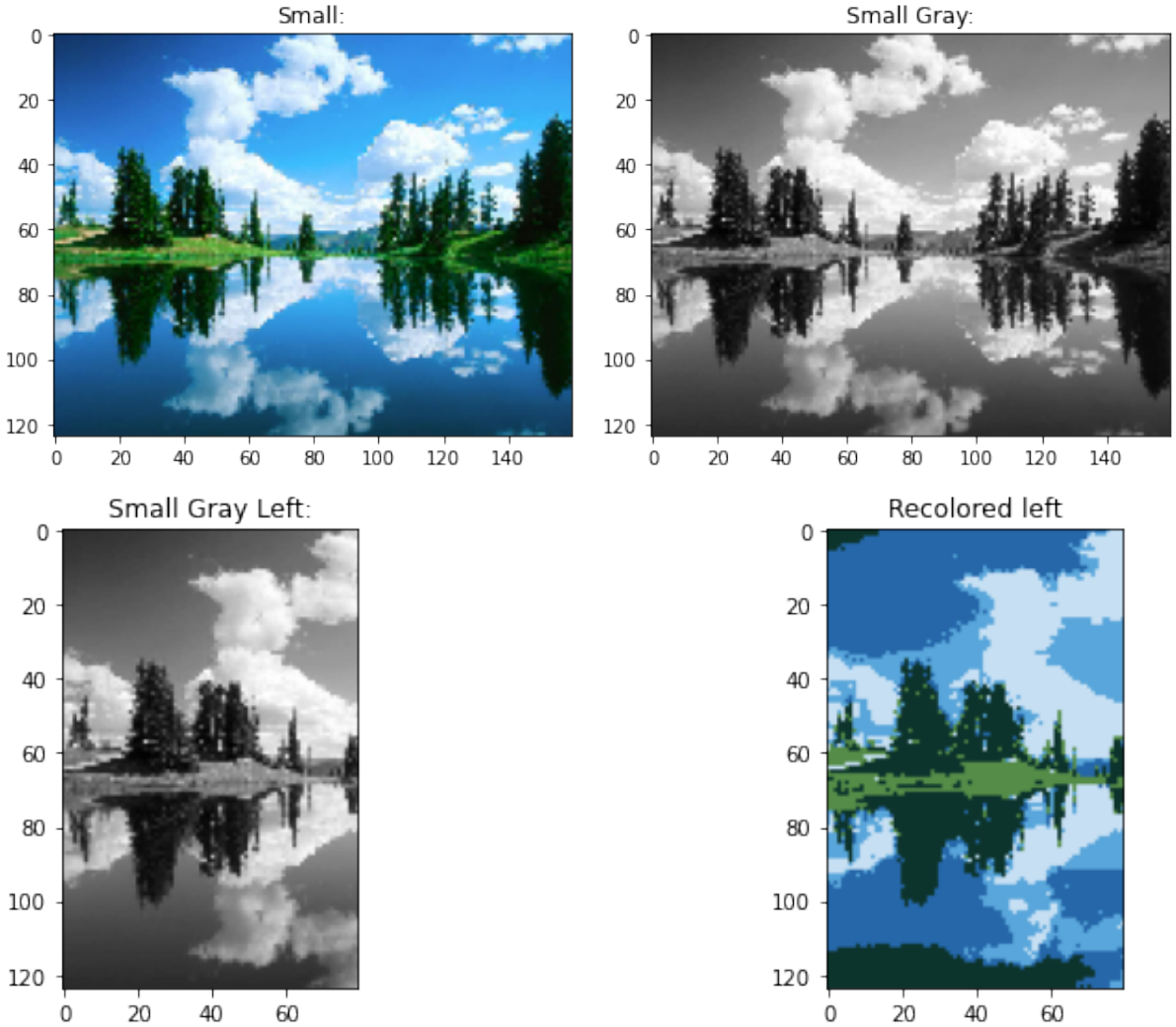
Achintya Singh, Tapan Patel

May 4, 2021

Exercise 1

How good is the final result? How could you measure the quality of the final result? Is it numerically satisfying, if not visually?

The final result seems surprisingly acceptable. We are comparing the predicted colors with the group truth and using mean squared error(MSE) as a measure to rate the agent. It is numerically satisfying as well.



Exercise 2

A specification of your solution, including describing your input space, output space, model space, error / loss function, and learning algorithm.

Input space is 3x3 matrix for each pixel in training set. Output is its corresponding R,G,B values (of middle pixel) Model space is weights/coeff of each power of each feature(3x3=9) Error function is Mean difference. Learning Algorithm is Gradient Descent on Polynomial Regression.

Exercise 3

How did you choose the parameters (structure, weights, any decisions that needed to be made) for your model?

Degree was chosen such that it can cover the needed info, we can consider all directions and say degree = feature to be conservative.

Exercise 4

Any pre-processing of the input data or output data that you used.

Yes, we made all values scaled in 0-1 before input and output to and from the model.

Exercise 5

How did you handle training your model? How did you avoid overfitting?

For the kind of model I'm using, I can reduce learning rate, add normalization and reduce degrees to avoid overfitting, if encountered.

Exercise 6

An evaluation of the quality of your model compared to the basic agent. How can you quantify and qualify the differences between their performance? How can you make sure that the comparison is 'fair' ?

The evaluation metric we used was MSE of corresponding pixel difference with the original image. The basic agent still has better results due to poor implementation of improved agent and we can quantify that using the evaluation metric itself and it also takes more time to execute. The comparison is consider fair since both are compared with same standard ground truth.

Exercise 7

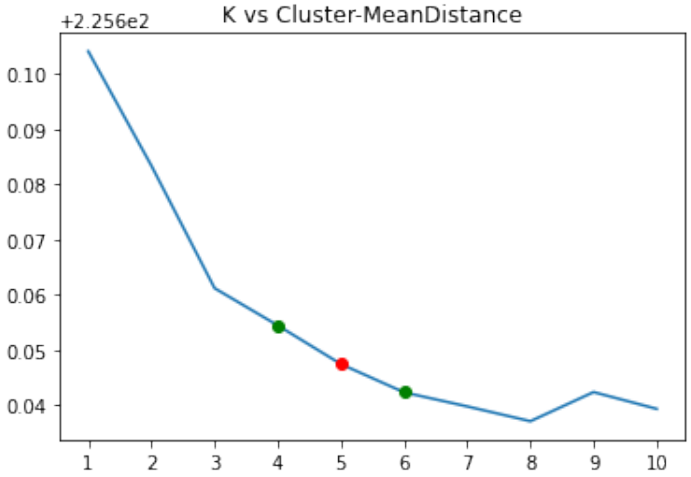
How might you improve your model with sufficient time, energy, and resources?

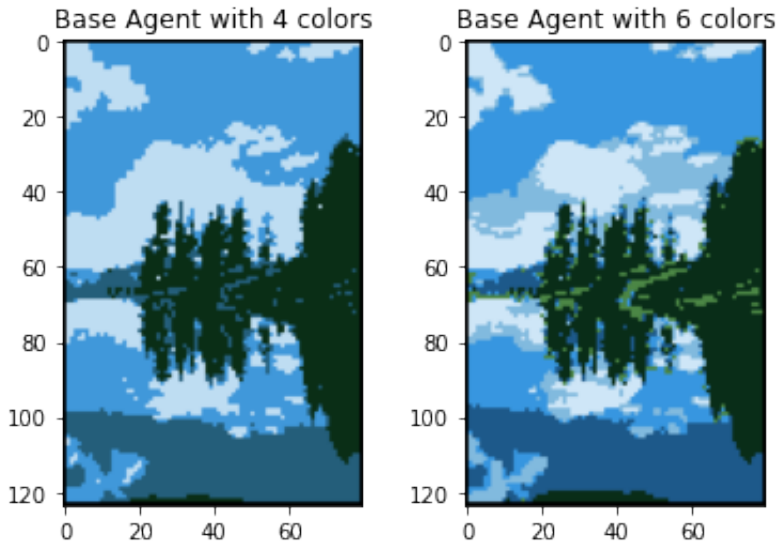
We will improve the implementation of Polynomial Regression and also consider limiting the search in 0-255 values of each channel such that they are able to satisfy the equation:

$$g = 0.21r + 0.72g + 0.07b$$

Exercise 8

Bonus 1: Instead of doing a 5-nearest neighbor based color selection, what is the best number of representative colors to pick for your data? How did you arrive at that number? Justify yourself and be thorough.



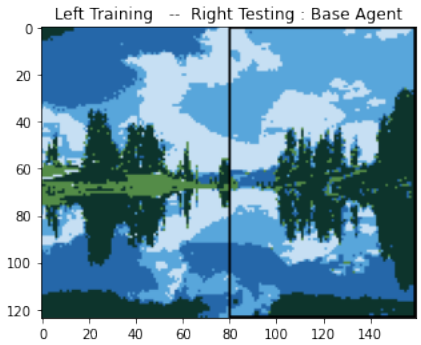


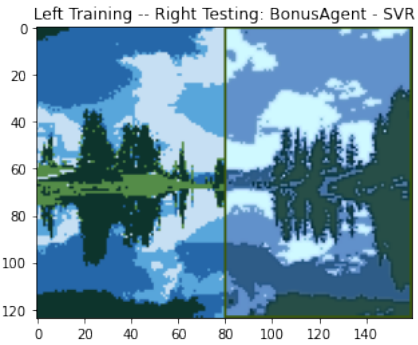
Base Agent Error (4 colors): 21.109085546985817
Base Agent Error (6 colors): 22.451747614131328

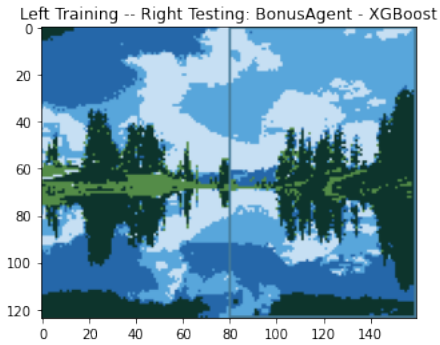
For our image, it really turned out to be 5 representative colors. We used the elbow method to find the best K. In this method, we plotted cluster mean-distance (mean distance of points from their cluster center) vs. K. This tells us for each K how much concise the cluster would be within itself. The elbow in this plot gives us the best of both worlds (i.e. Cluster size and reasonably small K). Had we picked a smaller value i.e. 4, we would have missed on 1 important color and large clusters(Missing 1 Green shade, merged with Waters). Had we picked a larger value (i.e. 6), we would have increased 1 color which was not really needed(Extra Sky color). We have also shown the base agent performs best only on the K=5 compared to K=4 and K=6. On top of this, even a quick eye check and counting the number of major colors give 5: White(cloud), Light Blue(sky), Dark Blue(water), and two shades of Green(trees).

Exercise 9

Bonus 2: Research a ML framework like scikit-learn or tensorflow, and build a solution to this problem using this framework that beats your improved agent.







We were eager enough to implement SVR(Support Vector Regression) to overcome the improved agent's shortcomings and found out that it is performing slightly better than the improved agent. We again used the same evaluation metric(MSE against ground truth). It was also seen that XGBoost(eXtreme Gradient Boosting) performs even better and faster - given its tree nature. So, XGBoost bags the place for best algorithm, second to SVR, followed by Polynomial Regression and Frequency(Base agent).

Sklearn:

```
def improved_agent2(img,new_colors):
    from sklearn.svm import SVR
    #source images: left and right
    recolored=recolor(L(img),new_colors)
    gl=g(recolor(L(img),new_colors)) # gray left
    gr=g(recolor(R(img),new_colors)) # gray right
    # extracting features
    glp=extract_patches(gl,3)
    glp_vals=np.array(list(glp.values()))
    glp_keys=list(glp.keys())
    grp=extract_patches(gr,3)
    flat_shape=gl.shape[0]*gl.shape[1]
    # training data
    X=glp_vals.reshape(flat_shape,9)
    #3-channel target vars: training
    Rtr=np.array([recolored[p[0],p[1]][0] for p in glp_keys])
    Gtr=np.array([recolored[p[0],p[1]][1] for p in glp_keys])
    Btr=np.array([recolored[p[0],p[1]][2] for p in glp_keys])
    # test data
    glr_vals=np.array(list(grp.values()))
    Xt=glr_vals.reshape(flat_shape,9)

    # predict per channel
    y_pred = lambda
        y,Xt:SVR(kernel='poly',degree=3,C=0.000001,coef0=1000).fit(X,y).predict(Xt)
    def lim(Yt): # inner limiting func.
        Yt[Yt>1]=1
        Yt[Yt<0]=0
```

```
        return Yt*255

    #Training and prediction
    Rt=lim(y_pred(Rtr,Xt))
    Gt=lim(y_pred(Gtr,Xt))
    Bt=lim(y_pred(Btr,Xt))

    glT=(np.array(list(zip(Rt,Gt,Bt)))).reshape([*gr.shape,3])/255 # merging 3-channel
        predictions
    return glT
```

XGBoost:

```
def improved_agent3(img,new_colors):
    import xgboost as xgb
    #source images: left and right
    recolored=recolor(L(img),new_colors)
    gl=g(recolor(L(img),new_colors)) # gray left
    gr=g(recolor(R(img),new_colors)) # gray right
    # extracting features
    glp=extract_patches(gl,3)
    glp_vals=np.array(list(glp.values()))
    glp_keys=list(glp.keys())
    grp=extract_patches(gr,3)
    flat_shape=gl.shape[0]*gl.shape[1]
    # training data
    X=glp_vals.reshape(flat_shape,9)
    #3-channel target vars: training
    Rtr=np.array([recolored[p[0],p[1]][0] for p in glp_keys])
    Gtr=np.array([recolored[p[0],p[1]][1] for p in glp_keys])
    Btr=np.array([recolored[p[0],p[1]][2] for p in glp_keys])
    # test data
    glr_vals=np.array(list(grp.values()))
    Xt=glr_vals.reshape(flat_shape,9)

    # predict per channel
    y_pred = lambda y,Xt:xgb.XGBRegressor(verbosity=0).fit(X, y).predict(Xt)
    def lim(Yt): # inner limiting func.
        Yt[Yt>1]=1
        Yt[Yt<0]=0
        return Yt*255

    #Training and prediction
    Rt=lim(y_pred(Rtr,Xt))
    Gt=lim(y_pred(Gtr,Xt))
    Bt=lim(y_pred(Btr,Xt))

    glT=(np.array(list(zip(Rt,Gt,Bt)))).reshape([*gr.shape,3])/255 # merging 3-channel
        predictions
    return glT
```