Assignment 4
Introduction to Artificial Intelligence

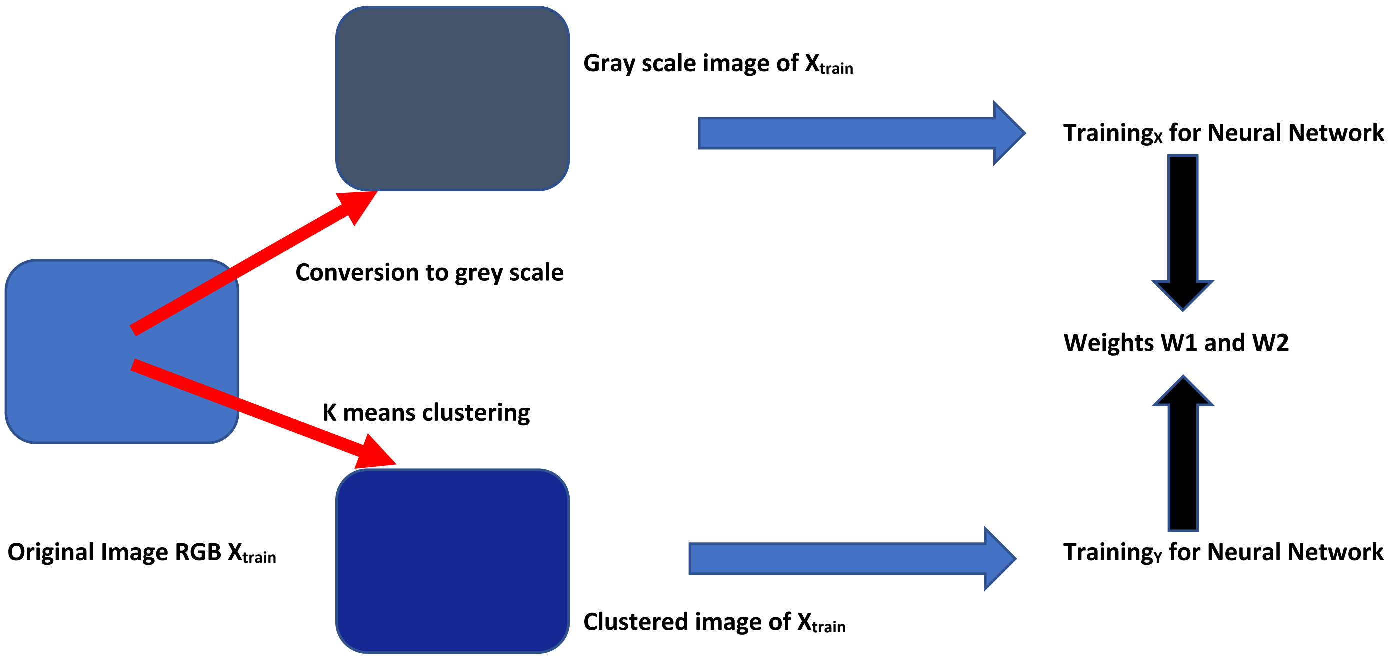
BY

Kartik Rattan, 187001624
Mareesh Kumar Issar, 186000297

Date: December 14, 2019

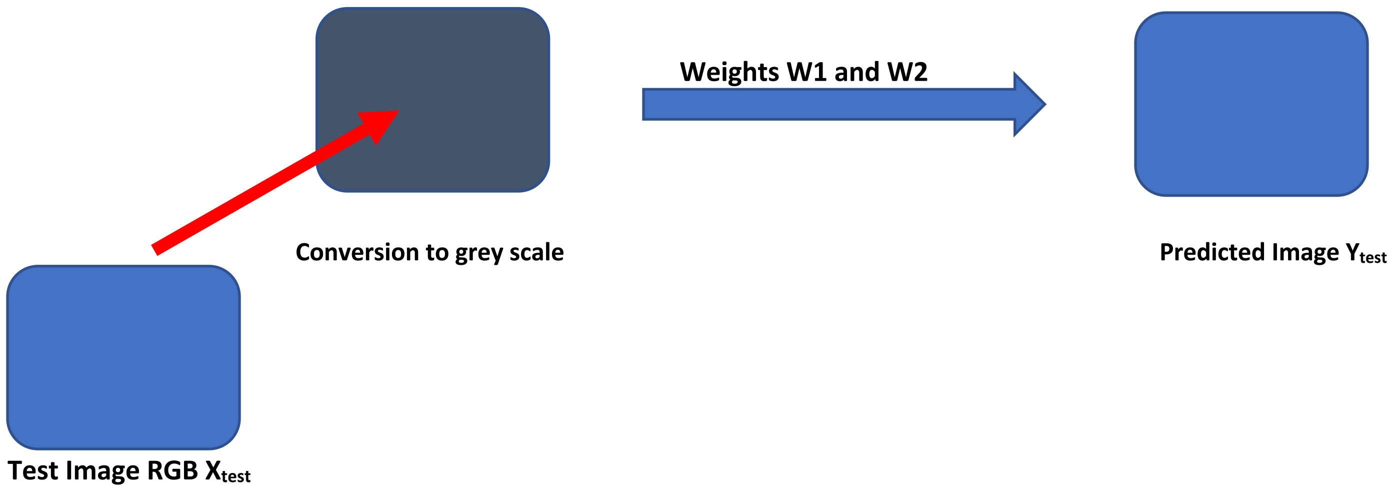
PLAN OF ACTION

1. **STEP 1: Training a system to learn a gray scale image and its corresponding RGB colors**



Gray scale image of X_{train}
$Training_X$ for Neural Network
Conversion to grey scale
Weights W1 and W2
K means clustering
Original Image RGB X_{train}
$Training_Y$ for Neural Network
Clustered image of X_{train}

2. **STEP 2: Predict RGB image from a gray scale image using the weights from our system**



Ques1. Representing the Process: How can you represent the coloring process in a way that a computer can handle? What spaces are you mapping between? What maps do you want to consider? Note that mapping from a single grayscale value gray to a corresponding color (r, g, b) on a pixel by pixel basis, you do not have enough information in a single gray value to reconstruct the correct color (usually).

Solution:

For a human eye, detection can be best done visually. Unfortunately, the computer still deals with numbers.

For the computer system, the image is divided into smaller areas knows as pixels. The pixel can be defined as a number (gray scale) or a set of numbers (RGB image). Usually, the RGB which stands for Red, Green and Blue, where pixel is made from 3 numbers in the range 0 to 255. A pixel with value [0 0 0] is black, and a pixel with value [255 255 255] is white. In gray scale, the pixel consists of a single number in the range 0 to 255. An RGB pixel can be converted into a gray scale pixel using the formula:

$$\textbf{gray} = 0.21 \times r + 0.72 \times g + 0.07 \times b \qquad \textbf{eq.1}$$

The assignment asks us to map Gray scale pixel → RGB scale Pixel, i.e.

[245] → [127 116 209]

But from eq.1 we realize that multiple values for gray could lead to RGB scale. **Multiple combinations for RGB can lead to a gray scale pixel**. Also, since eq.1 has decimal constants, **rounding the values to get an integer gray scale value also leads to same gray pixel from multiple RGB pixels.**

Thus, we need a higher dimension of a pixel which distinguishes between two gray scale pixels. We decided to take maps of 3X3, which describes a single gray scale pixel along with its 8 neighbors. For instance, the following shows us the representation of a gray scale image in a higher 3X3 dimensions.

Neighbor1	Neighbor2	Neighbor3
Neighbor4	PIXEL in Gray	Neighbor5
Neighbor6	Neighbor7	Neighbor8

↓ Conversion from 3X3 to 1X9

Neighbor1	Neighbor2	Neighbor3	Neighbor4	PIXEL in gray	Neighbor5	Neighbor6	Neighbor7	Neighbor8

↓ **Mapping**

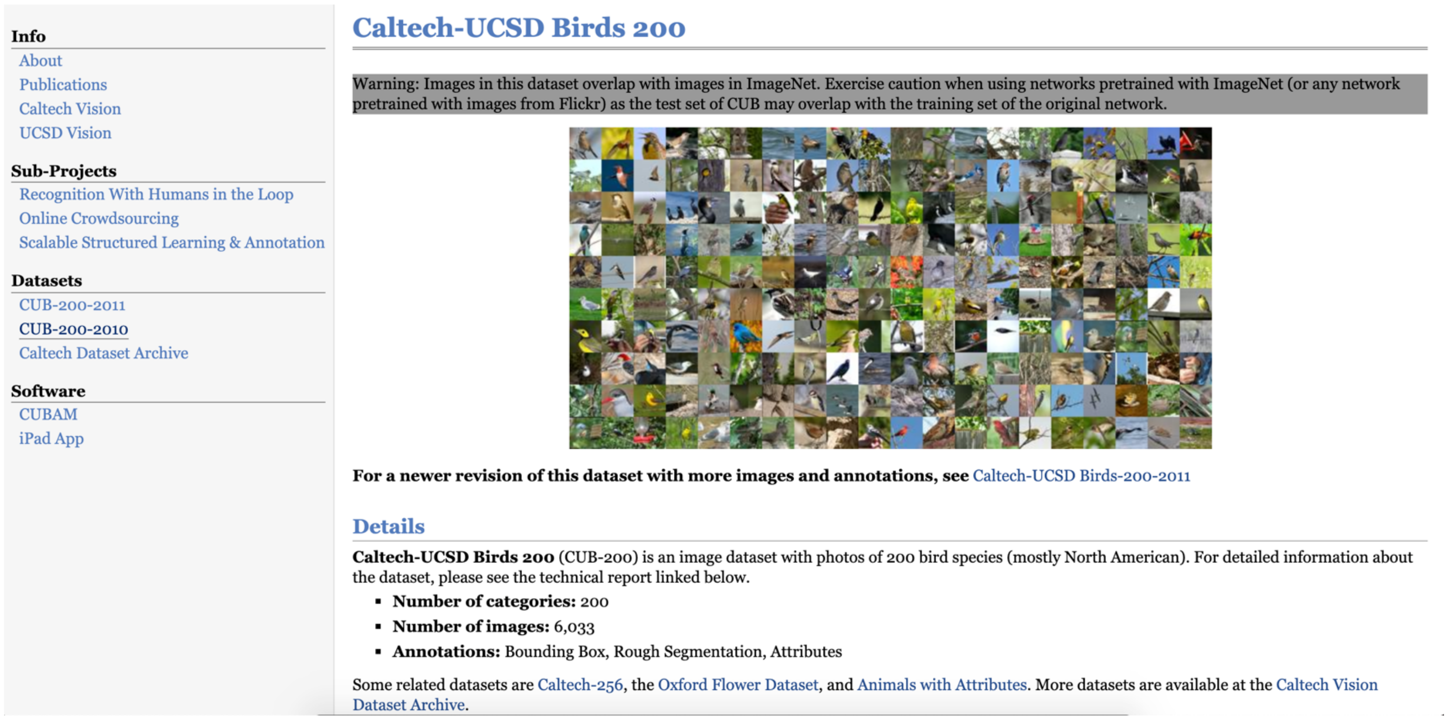
RED	GREEN	BLUE

Intuitively, we realize that as the number of pixels or dimensions increase, it is able to capture more information. A single blue pixel cannot give us much information, but a collection of blue pixels might tell us that we are talking about the sky!

Ques2. Data: Where are you getting your data from to train/build your model? What kind of pre-processing might you consider doing?

Solution:

The image data that we use to train and build the model is downloaded from: http://www.vision.caltech.edu/visipedia/CUB-200.html


Info
About
Publications
Caltech Vision
UCSD Vision

Sub-Projects
Recognition With Humans in the Loop
Online Crowdsourcing
Scalable Structured Learning & Annotation

Datasets
CUB-200-2011
CUB-200-2010
Caltech Dataset Archive

Software
CUBAM
iPad App

Caltech-UCSD Birds 200

Warning: Images in this dataset overlap with images in ImageNet. Exercise caution when using networks pretrained with ImageNet (or any network pretrained with images from Flickr) as the test set of CUB may overlap with the training set of the original network.

For a newer revision of this dataset with more images and annotations, see Caltech-UCSD Birds-200-2011

Details

Caltech-UCSD Birds 200 (CUB-200) is an image dataset with photos of 200 bird species (mostly North American). For detailed information about the dataset, please see the technical report linked below.

- **Number of categories:** 200
- **Number of images:** 6,033
- **Annotations:** Bounding Box, Rough Segmentation, Attributes

Some related datasets are Caltech-256, the Oxford Flower Dataset, and Animals with Attributes. More datasets are available at the Caltech Vision Dataset Archive.

For instance, some of the images that we used frequently for our training are:

Though it was not necessary, to avoid changing the dimensions for the various matrices we generated with respect to the images, we converted all the images in the dimension 240 X 420. This means that each of our image had 100800 pixels in total defining the image.

The pre-processing we considered was using K – Means clustering. The intuition behind using K – means clustering was to reduce the search space for our dataset. Each pixel has color specified by a (red, blue, green) value, and each value is an integer between 0 and 255. This gives a total of 16,777,216 possible colors, but in fact the picture contains much fewer colors. Thus, **we reduce the number of pixels** a gray scale image can be mapped to. We explain the K – means clustering in the next section.

Ques3. Evaluating the Model: Given a model for moving from grayscale images to color images (whatever spaces you are mapping between), how can you evaluate how good your model is? How can you assess the error of your model (hopefully in a way that can be learned from)? Note there are at least two things to consider when thinking about the error in this situation: numerical/quantified error (in terms of deviation between predicted and actual) and perceptual error (how good do humans find the result of your program).

Solution:

1. **K-Means Clustering**

We begin our discussion with K-means clustering. As mentioned, the idea behind using K-means clustering is simple: Each pixel has color specified by a (red, blue, green) value, and each value is an integer between 0 and 255. This gives a total of 16,777,216 possible colors. Instead of using such a range of pixels to define our search space, we reduce the search space altogether.

K-means clustering gives us the dominant colors present in an image. Thus, each gray pixel after training in the ANN model defined below has less options to choose from. But then the question arises:

How many dominant colors are present in an image?

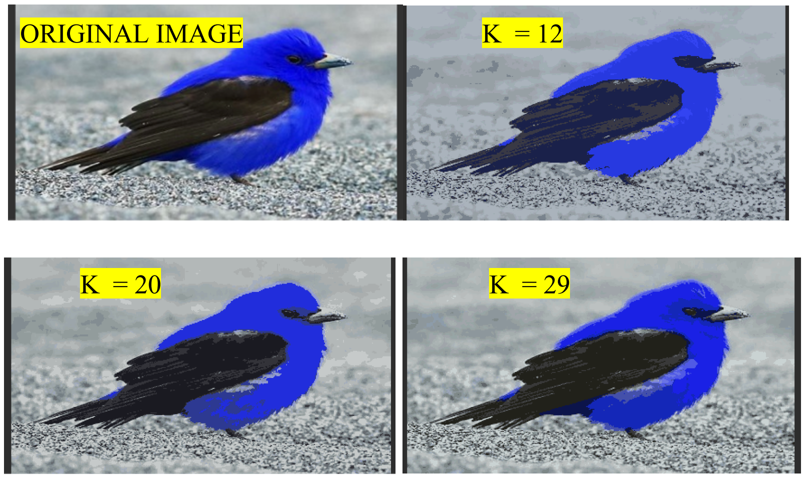
This translate to a general but important question in K-means clustering, **the number of K clusters we choose.** This can be achieved by the **elbow method**. The elbow method as defined by Wikipedia

*"The **elbow method** is a heuristic **method** of interpretation and validation of consistency within cluster analysis designed to help find the appropriate number of clusters in a dataset."*

The elbow method can also be defined in a human or perpetual model!

1. Select various values of clusters K.
2. Select the cluster which best describes the given image.

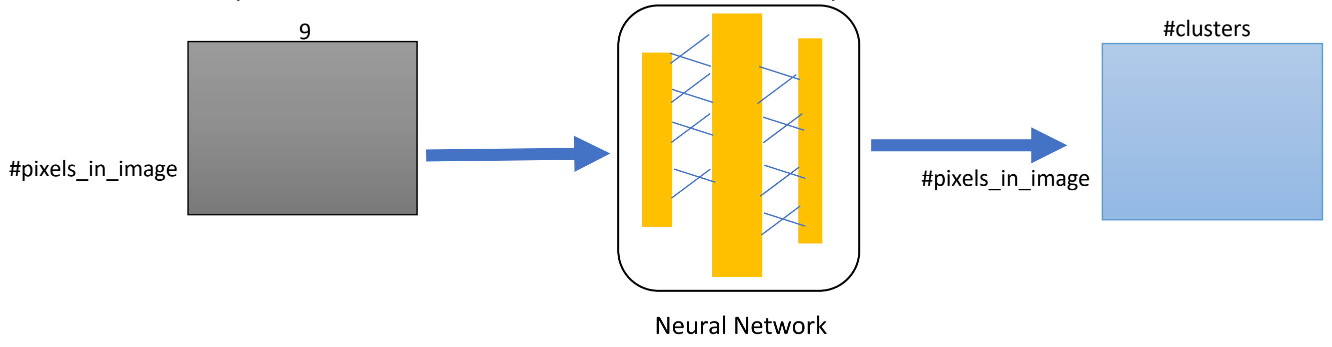
For instance, the following images are developed using different values of K:



As we can see, the image with clusters K = 29 resembles the picture the most, we chose K = 29 for the number of clusters in our K-means clustering algorithm.

Neural Network Architecture

We have used a 2-layer neural network with 60 nodes in the hidden layer



The input to the neural network is a matrix of dimensions **N(#pixels_in_image) X 9**. Each row of the input corresponds to a gray-scale pixel value and the value of its eight neighbors. Since a single grayscale pixel value cannot fully capture the mapping from grayscale to a RGB, we take a pixel along with its neighbors as our input. For the missing neighbors for the pixels at the image boundary we have used zero padding. We also scale all the grayscale values by 255 before they are fed into out neural network.

We have used **sigmoid activation function** for the nodes present in the hidden layer and **SoftMax activation function** for the nodes present at the output layer.

$$sigmoid(x) = \frac{1}{1+e^{-x}}$$

$$soft\max(z_j) = \frac{\exp(z_j)}{\sum_j \exp(z_j)}$$

The output layer consists of nodes=#clusters and the output is a **N(#pixels_in_image) X K(#clusters)** matrix. We compare the output of our ANN with the **one hot encoded** output of the image generated after clustering process is performed on the original image.

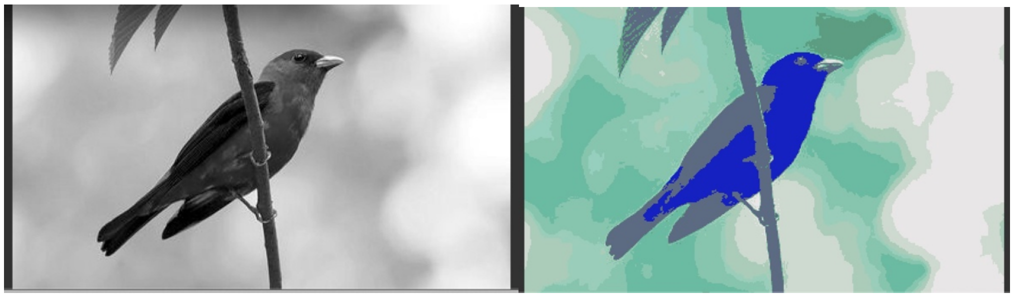
To evaluate our model, we measure the loss using **cross entropy error function** (which is common choice for loss function for multi-class classification problems) given by

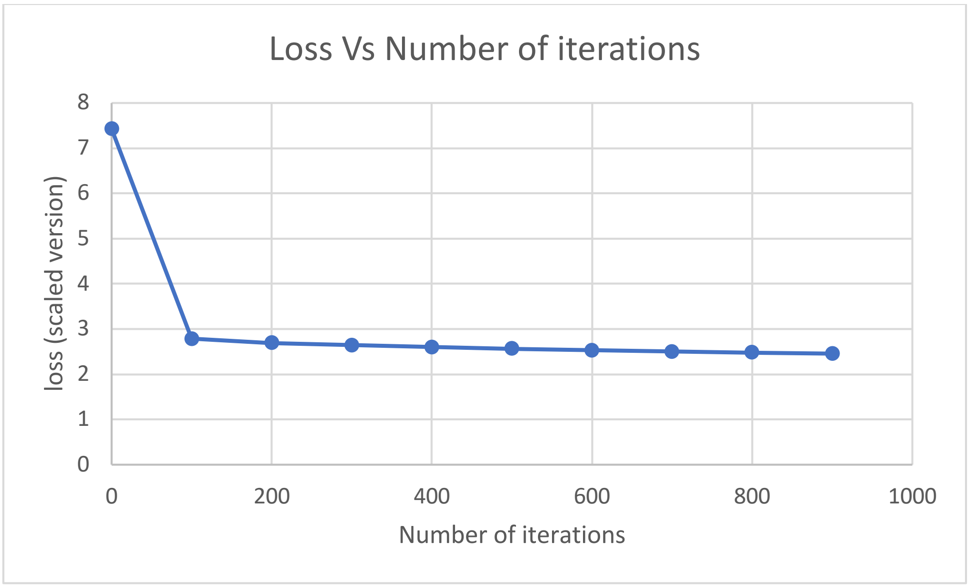
$$L = -\sum_i t_i \log(y_i)$$

For each set of 9 input grayscale values, where t_i represents the target value, y_i denotes the output value from the ANN and L is our loss function.

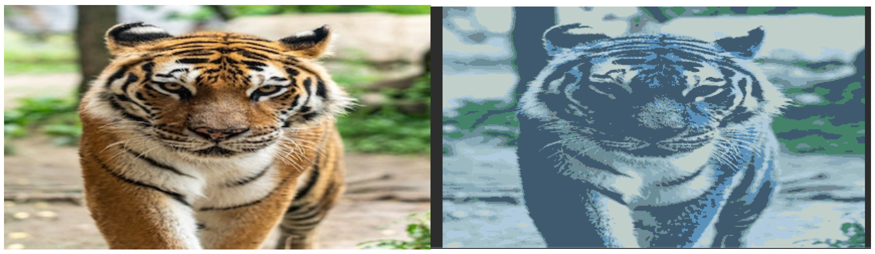
RESULTS:

GRAY SCALE TEST IMAGE PREDICTED OUTPUT

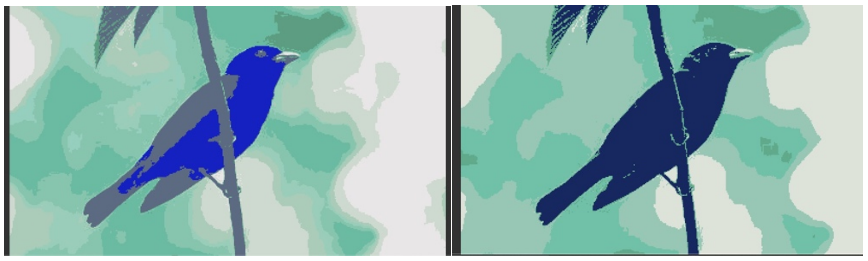




For evaluating the perceptual error, we generate the output image after coloring and compare it visually with the colors present in our training dataset. This error can further be split into two categories, first when there is error in the colors used, for example while **coloring the tiger, with the wrong colors**.



Another category is related to the shape of the object, for example the shape of the bird in our results. This shape is not visible due to reduced color palette after clustering, which leads to loss in information **related to textures.** Note that this approach is susceptible to the colors present in the image as well because we perceive different shades of green much more strongly the different shades of red. This can be seen below in the increasing order for perpetual error. **The first image depicts the original bird more than the last image**.



Ques4. Training the Model: Representing the problem is one thing, but can you train your model in a computationally tractable manner? What algorithms did you draw on? How did you determine convergence? How did you avoid overfitting?

Solution:

To train the model in a computationally tractable manner, we reduced to output space of the neural network from predicting any of the possible combination of RGB **(255 X 255 X 255)** to a distinct color palette of **K** colors that is one hot encoded. Also, we are using just one hidden layer in our model.

To generate the predicted output from the inputs we use forward propagation algorithm with **sigmoid activation function** at the **hidden layer** and **SoftMax activation function** at the **output layer**. For training the network we use **back propagation algorithm** where we calculate

$$\frac{\partial L}{\partial w_{ij}^{l-1}}$$

recursively for each layer starting from the output layer and propagating it backwards towards the input layer for using it to update the weights according to the equation

$$w_{new} = w_{old} - lr\nabla_{w_{old}} L$$

where lr is the learning rate, L is the loss function

For determining the convergence, we used accuracy as metric where accuracy is given by the formula:

Accuracy = Number of correct predictions / Total number of predictions

To avoid overfitting, we use the technique of early termination, where we track the change of error over time for both the training and testing. The best point of termination is the point where the test error starts increasing while the training error is still low and decreasing. We also utilize the concept of target error where we can stop the training as soon as the target error value is reached

Ques5. Assessing the Final Project: How good is your final program, and how can you determine that? How did you validate it? What is your program good at, and what could use improvement? Do your program's mistakes 'make sense'? What happens if you try to color images unlike anything the program was trained on? What kind of models and approaches, potential improvements and fixes, might you consider if you had more time and resources?

Solution:

There are two metrics that can be used for determining how good our final program is:

1. Numerical- By calculating the loss function with testing data as the input.
2. Visual- By visually inspecting the colors and their textures from the colored image.

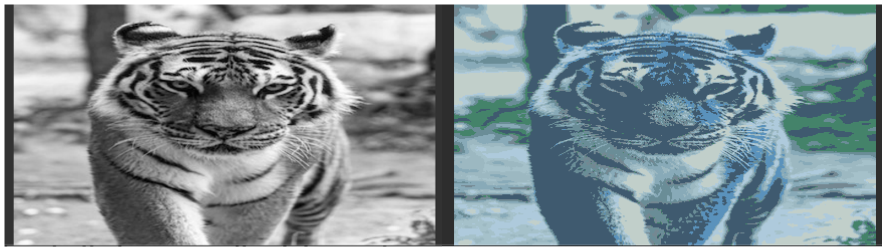
Our program is currently good at clustering the colors of an image to only significant colors and coloring a new unseen greyscale image that is similar to our training dataset to a corresponding colored image.

Our program could use a larger window size at the input i.e. have more information regarding a pixel and its neighbors. This would improve our program's ability to identify textures.

Our model is currently data specific as it is highly dependent on the data for example if our model was trained only on beach images (containing mostly blue and white color). It will perform poorly on an image of a tiger (contains mostly orange and black colors)

If we had more time and resources, we would have experimented with various activation functions, using larger number of hidden layers. Since our training data consisted of training our network from 3X3 window around a pixel from images of similar kind, we could improve this by training our network with 3X3 window around a pixel but from different images in a round robin fashion taking 3X3 window for the same pixel position but from different images. We would also experiment with different window sizes.

The major problems with this approach is that the model learns the textures or recognize the pixel mappings instead of the image. For instance, the model has been trained to recognize a blue bird. Instead, if we test our model on the image of a tiger, the tiger will be predicted as a blue tiger. This can be seen as follows:



The problem is the neural network does not learn the objects in the image. That is why it is not able to distinguish between a bird and a tiger. One solution is to train our model with **different types of images**. Another solution for this can be using a **convolutional neural network**, which learns the patterns of the image.

Bonus

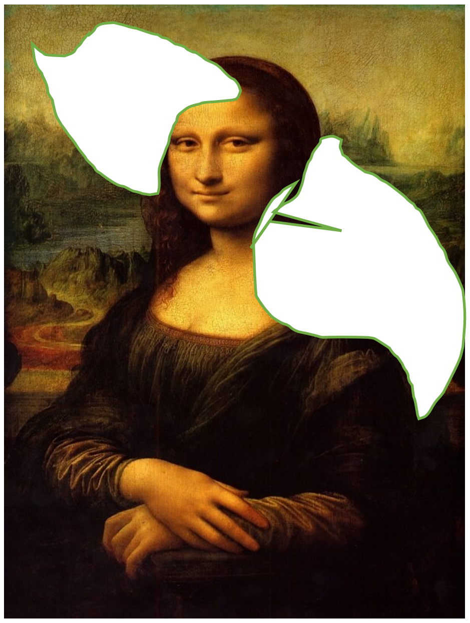
Imagine trying to reconstruct damaged images after an event like a fire. How could you build a system to 'in-paint' or fill in holes/missing areas in images? Consider the five areas outlined above in designing such a thing. Build it.

This problem can be best explained using the fire caused in **Notre-Dame Cathedral use case**. Several precious artifacts, including paintings, were burnt due to the devastation caused by the fire. Several years down the line, if someone wants to recreate the pictures, they use the following approach:

INPUT DATA SET:

X_{TRAIN}:

To mimic the burnt paintings/images, we can create white blobs in X_{train} which acts as the burnt portions. We can make the pixel values at several places as [255 255 255] in RGB scale. For instance, the following image of Mona Lisa can mimic a burnt Mona Lisa image.



Burnt image of Mona Lisa which acts as the X_{train}

Y_{TRAIN}:

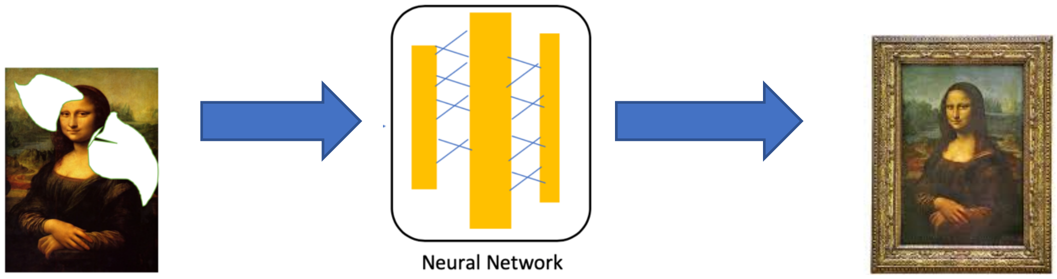
The Y_{TRAIN} can be several images that can be taken by other people in the past or copies of the same image found in different museums which can be used to predict the white blobs. For instance, the following images can be used to predict the white blobs (depicting fires):

Now using the above images, **the above problem now becomes a regression problem**. Also, **the information gained by mappings of 3X3 or other higher dimensions** by incorporating the neighbors of pixel to define that particular pixel is the very significant here.

The pixels on the neck of Mona Lisa which got burned can be learned from the pixels of the neck in the in the Y_{TRAIN} images.

The model can be used as follows:

1. We can again use the K – means clustering to reduce the mapping space for the pixels. But to retain the richness of texture in the image, we do not use K – means. We want the image to be as close to the original as possible.
2. The ANN can be used to learn the about the RGB colors for the pixels that were lost due to the burning of the image.



INPUT Y_{train} for comparison

Input X_{train}