Attribute-based Image Retrieval using Convolutional Neural Networks

Term Project for CS535- Big Data

Report submission date: 30 April 2022

Mridul Banik, Sanket Mayuresh Bhave and Shreya Anilkumar

Table of contents

1. Project Title

Attribute Based Image Retrieval using Convolutional Neural Networks

2. Problem formulation

Image mining deals with digging relevant images and extracting patterns from a targeted database of images. This concept is data mining with the exception that data is only images. Image mining can be useful in many fields like medical images, telescope images, or even satellite images. It helps in finding if the situation in the query image has been identified before and if yes, what is the relation of the content in the query image with the images the system is already aware of. Examples can be a relevant medical condition in an X-ray or even a relevant celestial object and even various social media applications.

In this project, we have implemented an efficient image retrieval system that focuses on high accuracy and tries to identify any (possible) relations with the images already known. Considering that images take up a sizable portion of the total data generated every year due to various social media platforms, they have become an especially important part of our lives. Hence, it seems interesting how images have a trend and what useful information can be extracted just by looking at the images generated.

3. Methodology

The approach to the problem formulation has the following input and output:

- Input to the system: a query image
- Output of the system: Top 'N' relevant images from the dataset.

3.1. Dataset used: CelebA

The dataset used for the image retrieval project is the CelebA dataset. The CelebA dataset is great for training and testing models for face detection, particularly for recognizing facial attributes such as finding people with brown hair, smiling, or wearing glasses. Images cover large pose variations, background clutter, diverse people, supported by many images and rich annotations.

This data was originally collected by researchers at MMLAB, The Chinese University of Hong Kong. This dataset has a total of two hundred thousand (202,599) face images of numerous popular celebrities

The overall size of the dataset is up to 1.45 GB with the major data files consisting of:

- img*align*celeba.zip: All the face images are cropped and aligned.
- list*eval*partition.csv: Recommended partitioning of images into training, validation, testing sets. Images 1-162770 are training, 162771-182637 are validation, 182638-202599 are testing

- listboxceleba.csv: Bounding box information for each image. "x*1" and "y*1" represent the upper left point coordinate of bounding box. "width" and "height" represent the width and height of bounding box
- list*landmarks*align_celeba.csv: Image landmarks and their respective coordinates. There are 5 landmarks: left eye, right eye, nose, left mouth, right mouth
- listatrceleba.csv: Attribute labels for each image. There are 40 attributes. "1" represents positive while "-1" represents negative. The following are the examples of attributes which have been labelled to the images: 5_o_Clock_Shadow, Arched_Eyebrows, Attractive, Bags_Under_Eyes, Bald, Bangs, Big_Lips, Big_Nose, Black_Hair etc.

Based on the project formulation, the files used for this project are only list*eval*partition.csv, listatrceleba.csv and img*align*celeba.zip. A custom dataset has been made using these files to suit the project requirements.

3.2. Data pre-processing and explorative analytics

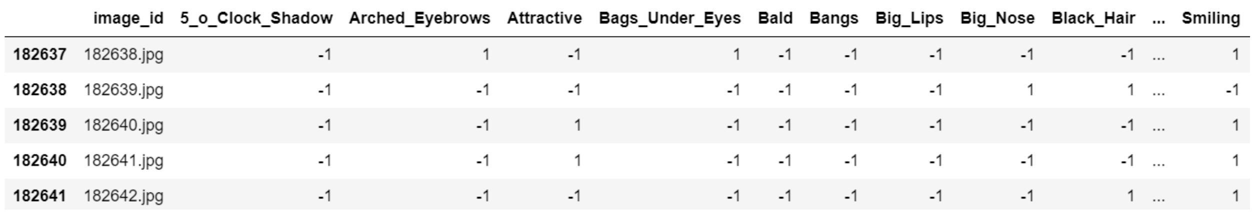
All the images of the CelebA dataset are indexed based on the image id, which is the primary key of the image database. The entire dataset is divided into Train set, Test set and Validation set. All the image dimensions are checked as given below in Fig1.

```
(218, 178, 3)
```

Figure1: Sample image from dataset with its dimensions

All images in the train dataset are resized to (299, 299) to standardize all images with the same dimensions, horizontally flipped to 0.5 units and rotated to 45 degrees. Similarly, all images in the validation set are resized to (299, 299). All images in the database, i.e., train, test and validation set are transformed to tensor.

The CelebA dataset has images mapped to its image ids and image ids mapped to its attributes as shown in Fig 2. For the process of image mining, it is required to directly map images with their respective attributes. Hence to facilitate this process, all images are converted from BGR to RGB color format and image transforms are applied to obtain image with its respective attributes.

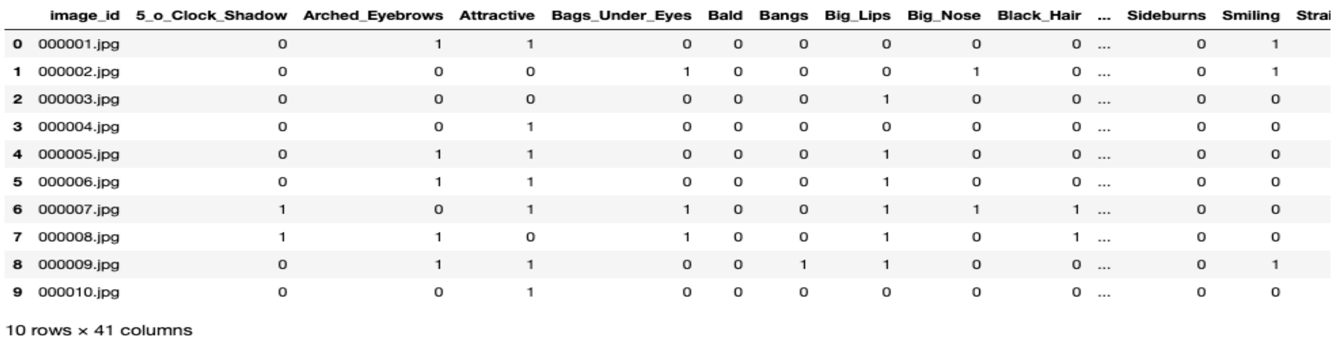


	image_id	5_o_Clock_Shadow	Arched_Eyebrows	Attractive	Bags_Under_Eyes	Bald	Bangs	Big_Lips	Big_Nose	Black_Hair	...	Smiling
182637	182638.jpg	-1	1	-1	1	-1	-1	-1	-1	-1	...	1
182638	182639.jpg	-1	-1	-1	-1	-1	-1	-1	1	1	...	-1
182639	182640.jpg	-1	-1	1	-1	-1	-1	-1	-1	-1	...	1
182640	182641.jpg	-1	-1	1	-1	-1	-1	-1	-1	-1	...	1
182641	182642.jpg	-1	-1	-1	-1	-1	-1	-1	-1	1	...	1

Figure 2: Image ids mapped to its attributes

As it will be much easier to achieve a probability of an attribute present, we have converted our attributes.csv file above such that it will have value 0 if the corresponding attribute is not present in the image, else will have the value 1.

So, our attributes now look like:



	image_id	5_o_Clock_Shadow	Arched_Eyebrows	Attractive	Bags_Under_Eyes	Bald	Bangs	Big_Lips	Big_Nose	Black_Hair	...	Sideburns	Smiling	Strai
0	000001.jpg	0	1	1	0	0	0	0	0	0	...	0	1	
1	000002.jpg	0	0	0	1	0	0	0	1	0	...	0	1	
2	000003.jpg	0	0	0	0	0	0	1	0	0	...	0	0	
3	000004.jpg	0	0	1	0	0	0	0	0	0	...	0	0	
4	000005.jpg	0	1	1	0	0	0	1	0	0	...	0	0	
5	000006.jpg	0	1	1	0	0	0	1	0	0	...	0	0	
6	000007.jpg	1	0	1	1	0	0	1	1	1	...	0	0	
7	000008.jpg	1	1	0	1	0	0	1	0	1	...	0	0	
8	000009.jpg	0	1	1	0	0	1	1	0	0	...	0	1	
9	000010.jpg	0	0	1	0	0	0	0	0	0	...	0	0	

10 rows × 41 columns

Figure 3: Attributes have value only 0 or 1

3.3 Building a Custom Data Loader:

Pytorch loads the dataset into a data loader. The data loader is used to divide the data into batches and even for applying transformations which can be used as a part of preprocessing. We can call it an iterable over the dataset.

In our case for training, we required the data loader to give an output as two fields:

1] image and 2] the corresponding attribute vector for the image.

Whereas for image retrieval purpose, we added another field this two which is image id.

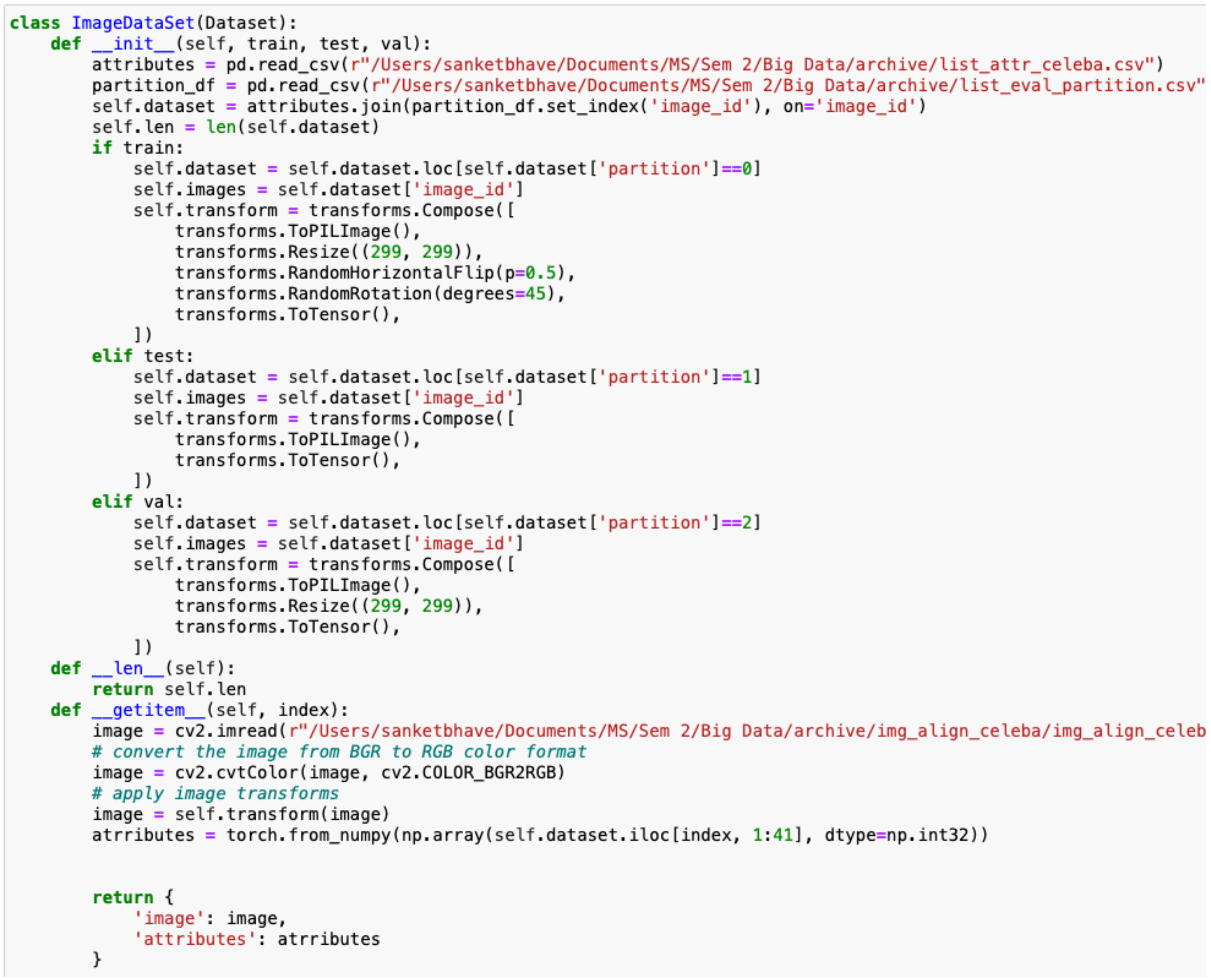
The in-built datasets like MNIST have a predefined data loader which outputs an image and its corresponding label. The CelebA dataset is not built in Pytorch. Unlike MNIST, we have images in a separate folder and our attribute mapping is in a separate CSV file. To map our exact image to the attribute vector, we need to have a data loader and we decided to build our Dataset class for the same.

We expected two things from our custom object:

1] It should give an appropriate train, validation and test set when asked for the same.

2] It should give an image along with its attributes converted to tensors.

Accordingly, we did the same. Below is a snapshot of our class:



```
class ImageDataSet(Dataset):
    def __init__(self, train, test, val):
        attributes = pd.read_csv(r"/Users/sanketbhave/Documents/MS/Sem 2/Big Data/archive/list_attr_celeba.csv")
        partition_df = pd.read_csv(r"/Users/sanketbhave/Documents/MS/Sem 2/Big Data/archive/list_eval_partition.csv"
        self.dataset = attributes.join(partition_df.set_index('image_id'), on='image_id')
        self.len = len(self.dataset)
        if train:
            self.dataset = self.dataset.loc[self.dataset['partition']==0]
            self.images = self.dataset['image_id']
            self.transform = transforms.Compose([
                transforms.ToPILImage(),
                transforms.Resize((299, 299)),
                transforms.RandomHorizontalFlip(p=0.5),
                transforms.RandomRotation(degrees=45),
                transforms.ToTensor(),
            ])
        elif test:
            self.dataset = self.dataset.loc[self.dataset['partition']==1]
            self.images = self.dataset['image_id']
            self.transform = transforms.Compose([
                transforms.ToPILImage(),
                transforms.ToTensor(),
            ])
        elif val:
            self.dataset = self.dataset.loc[self.dataset['partition']==2]
            self.images = self.dataset['image_id']
            self.transform = transforms.Compose([
                transforms.ToPILImage(),
                transforms.Resize((299, 299)),
                transforms.ToTensor(),
            ])
    def __len__(self):
        return self.len
    def __getitem__(self, index):
        image = cv2.imread(r"/Users/sanketbhave/Documents/MS/Sem 2/Big Data/archive/img_align_celeba/img_align_celeb
        # convert the image from BGR to RGB color format
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        # apply image transforms
        image = self.transform(image)
        atrributes = torch.from_numpy(np.array(self.dataset.iloc[index, 1:41], dtype=np.int32))


        return {
            'image': image,
            'attributes': atrributes
        }
```

Figure 4: Image Data Set Class

As you can observe, the __init__ () function returns the corresponding set and the __getitem__ () function returns the mapping of image to the attribute. This __getitem__ () function is used by the DataLoader() class to load the data in batches.

3.4 Convolutional Neural Network Model

The team havs experimented with various conventional deep neural network architectures and implemented transfer learning on them. Some of them that were considered for analysis are listed below:

- VGG16: VGG16 is a convolution neural net (CNN) architecture which was used to win ILSVR(ImageNet) competition in 2014. It is one of the excellent vision model architectures to date. The most unique thing about VGG16 is that instead of having g many hyper-parameters they focused on having convolution layers of 3x3 filter with a stride 1 and always used the same padding and max pool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end, it has 2 FC (Fully Connected) (fully connected layers) followed by a SoftMax for output. The 16 in VGG16 refers to it has 16 layers that have weights. This network is a large network, and it has about 138 million (approx.) parameters.

- Resnet: which is short for Residual Networks, a classic neural network used as a backbone for many computers vision tasks. This model was the winner of the ImageNet challenge in 2015. The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with 150+layers successfully. Prior to ResNet training very deep neural networks were difficult due to the problem of vanishing gradients.

- Inception-V3: This model has obtained an accuracy of about 78% on the ImageNet dataset. Inception-V3 has symmetric and asymmetric blocks and has convolutions, average pooling, max pooling, concatenations, dropouts, and fully connected layers. The model uses SoftMax as a loss function.

- Xception : Xception is a deep convolutional neural network architecture that involves Depth wise Separable Convolutions. It was developed by Google researchers. Google presented an interpretation of Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the depth wise separable convolution operation (a depth wise convolution followed by a pointwise convolution). In this light, a depth wise separable convolution can be understood as an Inception module with a maximally substantial number of towers. This observation leads them to propose a novel deep convolutional neural network architecture inspired by Inception, where Inception modules have been replaced with depth wise separable convolutions.

The team finalized Resnet 18. This model was chosen as it was lighter than all other models as Networks with considerable number (even thousands) of layers can be trained easily without increasing the training error percentage.

3.4.1 ResNet 18 architecture:

ResNet 18 is a 72-layer architecture with 18 deep layers. The architecture of the Resnet 18 network aimed at enabling enormous amounts of convolutional layers to function efficiently.

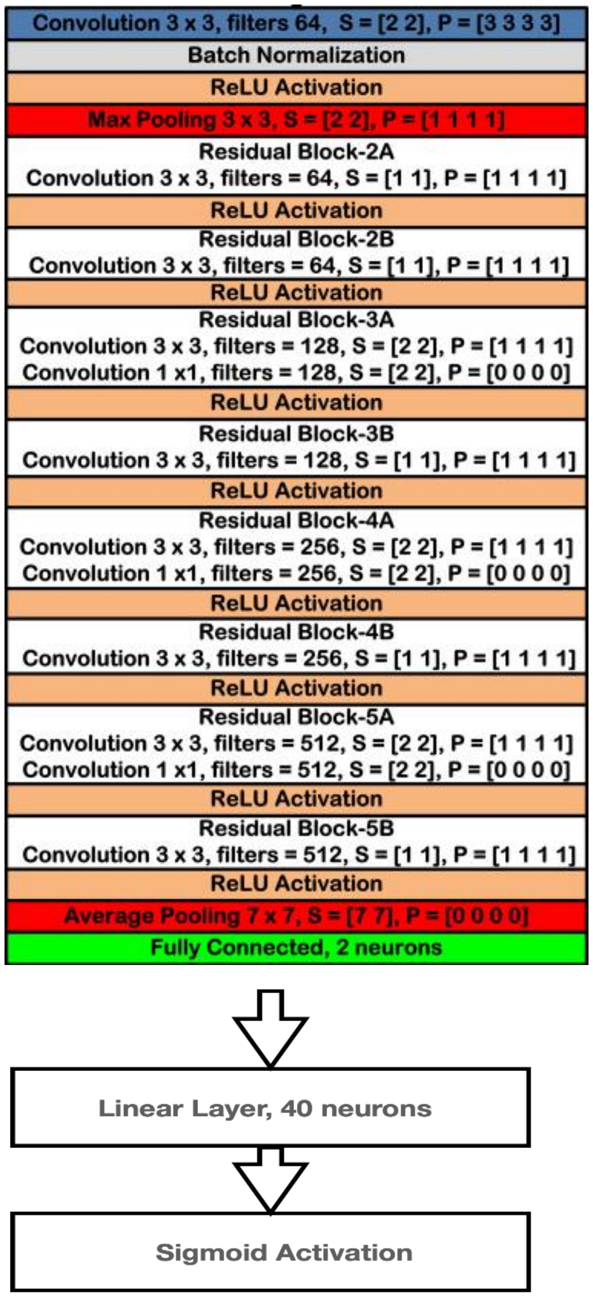

Convolution 3 x 3, filters 64, S = [2 2], P = [3 3 3 3]
Batch Normalization
ReLU Activation
Max Pooling 3 x 3, S = [2 2], P = [1 1 1 1]
Residual Block-2A
Convolution 3 x 3, filters = 64, S = [1 1], P = [1 1 1 1]
ReLU Activation
Residual Block-2B
Convolution 3 x 3, filters = 64, S = [1 1], P = [1 1 1 1]
ReLU Activation
Residual Block-3A
Convolution 3 x 3, filters = 128, S = [2 2], P = [1 1 1 1]
Convolution 1 x1, filters = 128, S = [2 2], P = [0 0 0 0]
ReLU Activation
Residual Block-3B
Convolution 3 x 3, filters = 128, S = [1 1], P = [1 1 1 1]
ReLU Activation
Residual Block-4A
Convolution 3 x 3, filters = 256, S = [2 2], P = [1 1 1 1]
Convolution 1 x1, filters = 256, S = [2 2], P = [0 0 0 0]
ReLU Activation
Residual Block-4B
Convolution 3 x 3, filters = 256, S = [1 1], P = [1 1 1 1]
ReLU Activation
Residual Block-5A
Convolution 3 x 3, filters = 512, S = [2 2], P = [1 1 1 1]
Convolution 1 x1, filters = 512, S = [2 2], P = [0 0 0 0]
ReLU Activation
Residual Block-5B
Convolution 3 x 3, filters = 512, S = [1 1], P = [1 1 1 1]
ReLU Activation
Average Pooling 7 x 7, S = [7 7], P = [0 0 0 0]
Fully Connected, 2 neurons
Linear Layer, 40 neurons
Sigmoid Activation

Figure 5: Sample architecture of Resnet 18

Figure 5 shows a sample of the Resnet 18 architecture. As you can see in the above picture, we added two layers to the existing ResNet 18 architecture. As we have 40 attributes, we need 40 output values, so added a linear layer with 40 neurons. To these neurons, we applied Sigmoid activation function as we wanted to have value for each attribute between 0 and 1.

This architecture developed seemed to be robust and we can replace the upper ResNet 18 layer with any pre-trained or self-built CNN architecture and implement our purpose of image retrieval.

3.4.2 Cost Function:

Finding an appropriate cost function was an incredibly challenging task for us. Selecting the cost function becomes a very crucial task for Deep Learning problems. If not chosen wisely, every effort is a waste. We initially thought Cosine Similarity would be useful, but we did not get a good loss value for our training data. After careful thought and many brainstorming sessions, we found that the problem we are trying to solve is none other than a multi-label classification problem. Then, it was decided that Binary Cross Entropy (BCELoss) can be used for our problem.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1-y_i) \cdot log(1-p(y_i))$$

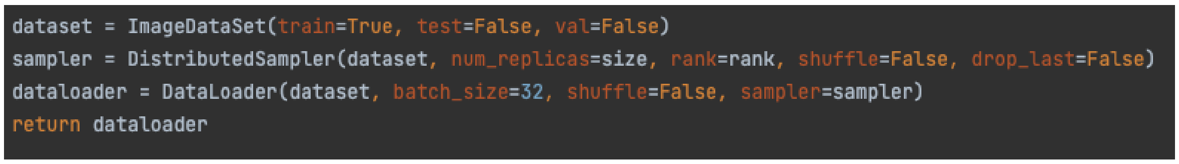
Binary Cross-Entropy / Log Loss

Figure 6: Cost function

3.4 Distributed Training

PyTorch is an optimized tensor library for deep learning using GPUs and CPUs. The team has utilized the distributed package of pytorch to implement distributed training. This has enabled us to easily parallelize the computations across processes and clusters of machines. All essential packages such as multiprocessing, Distributed Sampler, Data Loader, and Distributed Data Parallel as DDP are imported.

The Distributed Sampler samples the training data into the number of nodes working on the same. This DistributedSampler() is used by the DataLoader() to evenly distribute the data among the working nodes, including the master.



```
dataset = ImageDataSet(train=True, test=False, val=False)
sampler = DistributedSampler(dataset, num_replicas=size, rank=rank, shuffle=False, drop_last=False)
dataloader = DataLoader(dataset, batch_size=32, shuffle=False, sampler=sampler)
return dataloader
```

Figure 7: DistributedSampler() used by the DataLoader()

The distributed package uses the Gloo backend. The Gloo backend is quite handy as a development platform, as it is included in the pre-compiled PyTorch binaries and works on both Linux (since

0.2) and macOS (since 1.3). It supports all point-to-point and collective operations on CPU, and all collective operations on GPU.

On setting four environment variables on all machines, all processes will be able to properly connect to the master, obtain information about the other processes, and finally handshake with each other. The environment variables are as follows:

- MASTER_PORT: A free port on the machine that will host the process with rank 0.
- MASTER_ADDR: IP address of the machine that will host the process with rank 0.
- WORLD_SIZE: The total number of processes, so that the master node knows how many workers to wait for.
- RANK: Rank of each process, so they will know whether it is the master process of a worker.

The python APIs majorly used in this project are torch, torch.nn, torch.nn.functional, torch.Tensor. The torch package contains data structures for multi-dimensional tensors and defines mathematical operations over these tensors. Additionally, it provides many utilities for efficient serializing of Tensors and arbitrary types, and other useful utilities. Torch.nn are the basic building blocks for graphs which includes Containers, Convolution Layers, pooling layers, Padding Layers, Non-linear Activations (weighted sum, nonlinearity), Non-linear Activations (other), Normalization Layers, Recurrent Layers, Transformer Layers, Linear Layers, Dropout Layers, Loss Functions etc.

4. Results and Evaluation

The dataset was divided into ‘n’ equal parts, where n is the total number of nodes participating in the training process. This ‘n’ was user configurable and was taken as a command-line argument.

Each node worked on its own partition of the data. We ran our training for 10 epochs. At the end of epoch 10, each node gave the same loss value:

```
epoch 9: 0.19342613032852501
```

Post this, we evaluated our model on validation set which had around 20K samples. We wanted to find the classification loss on the validation set.

Deciding which factor to count was a major challenge for us. We came across a simple metric- Calculating differences between the actual and predicted class values. The reason we fixed this was: multi-label classification can be grouped into 3 groups:

1] Evaluating ranking: Our attributes have no ranking, so this is not a valid measure

2] Hierarchical loss: Our data has no hierarchy.

3] Partition-based loss: This includes many metrics like F1 score, accuracy, hamming loss, exact match ratio, etc. Since, in our cases, we do not pose any stringent condition to determine labels, we cannot exactly say which labels predicted are correct and which are not.

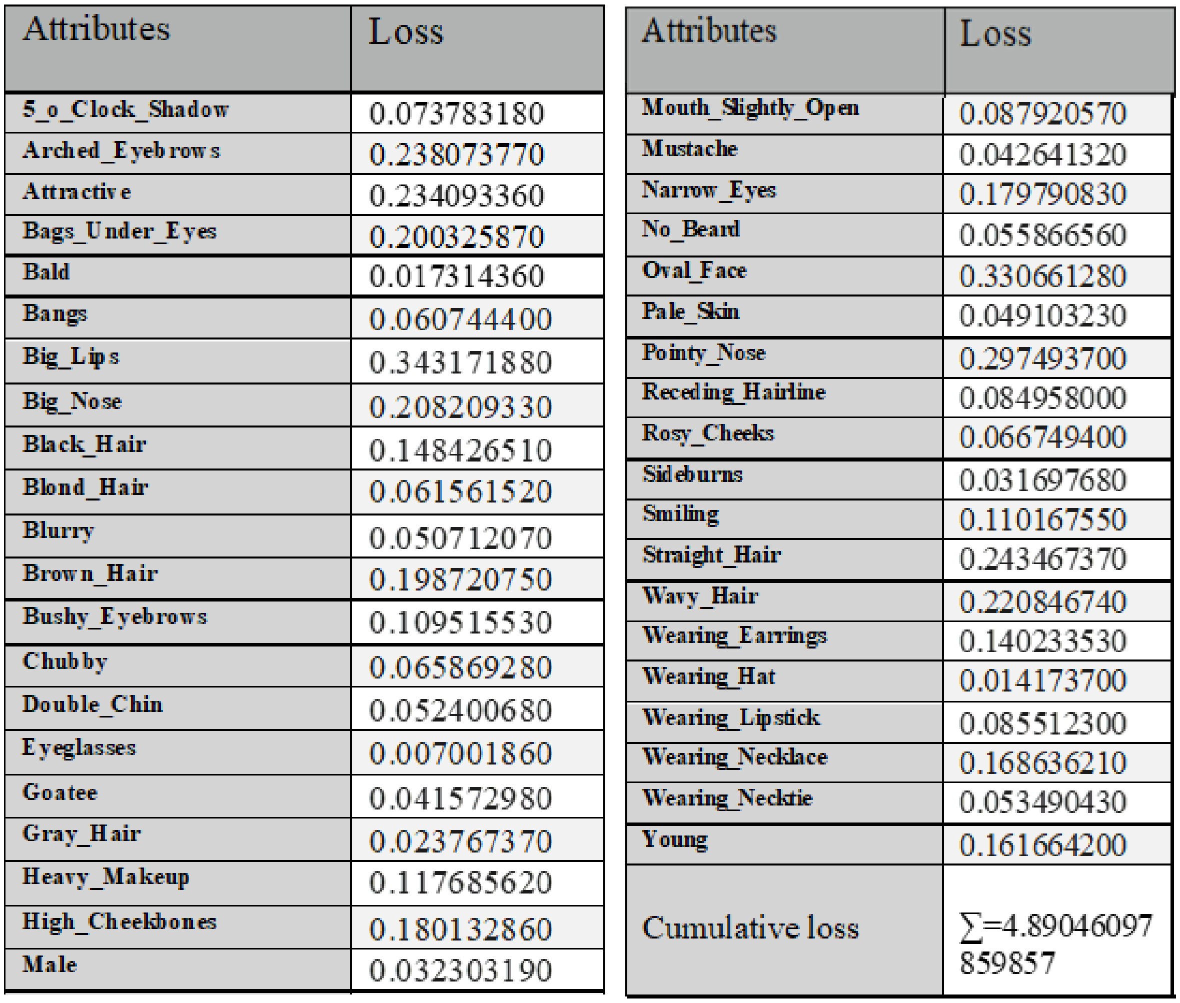
Meaning, we output sigmoid values in our last layer, we cannot determine which labels are predicted confidently and which are not. If we put a condition like:

x>0.5, y=1 else y=0,

We may lose some important insights into the data or gain unnecessary attributes as well.

Due to this issue, we do not know which labels are "correct" and which are not. Hence, metrics like exact match ratio or hamming loss fail to capture the loss. Same is the case with F1 score.

So, we decided to give the average difference as our loss, accordingly, we have the following per-attribute loss:



Attributes	Loss
5_o_Clock_Shadow	0.073783180
Arched_Eyebrows	0.238073770
Attractive	0.234093360
Bags_Under_Eyes	0.200325870
Bald	0.017314360
Bangs	0.060744400
Big_Lips	0.343171880
Big_Nose	0.208209330
Black_Hair	0.148426510
Blond_Hair	0.061561520
Blurry	0.050712070
Brown_Hair	0.198720750
Bushy_Eyebrows	0.109515530
Chubby	0.065869280
Double_Chin	0.052400680
Eyeglasses	0.007001860
Goatee	0.041572980
Gray_Hair	0.023767370
Heavy_Makeup	0.117685620
High_Cheekbones	0.180132860
Male	0.032303190
Mouth_Slightly_Open	0.087920570
Mustache	0.042641320
Narrow_Eyes	0.179790830
No_Beard	0.055866560
Oval_Face	0.330661280
Pale_Skin	0.049103230
Pointy_Nose	0.297493700
Receding_Hairline	0.084958000
Rosy_Cheeks	0.066749400
Sideburns	0.031697680
Smiling	0.110167550
Straight_Hair	0.243467370
Wavy_Hair	0.220846740
Wearing_Earrings	0.140233530
Wearing_Hat	0.014173700
Wearing_Lipstick	0.085512300
Wearing_Necklace	0.168636210
Wearing_Necktie	0.053490430
Young	0.161664200
Cumulative loss	∑=4.89046097859857

Table 1: Loss calculated per attribute

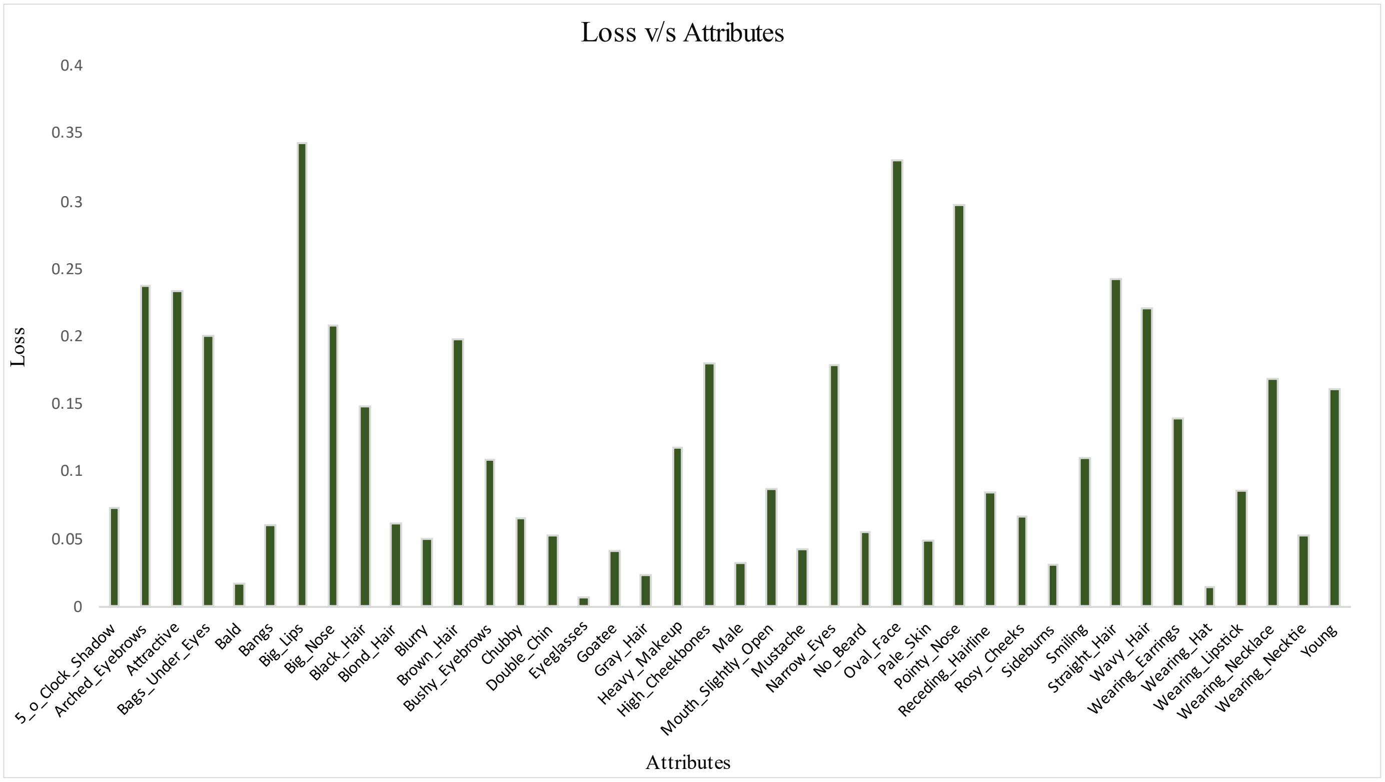


Figure 8: Per Attribute Loss

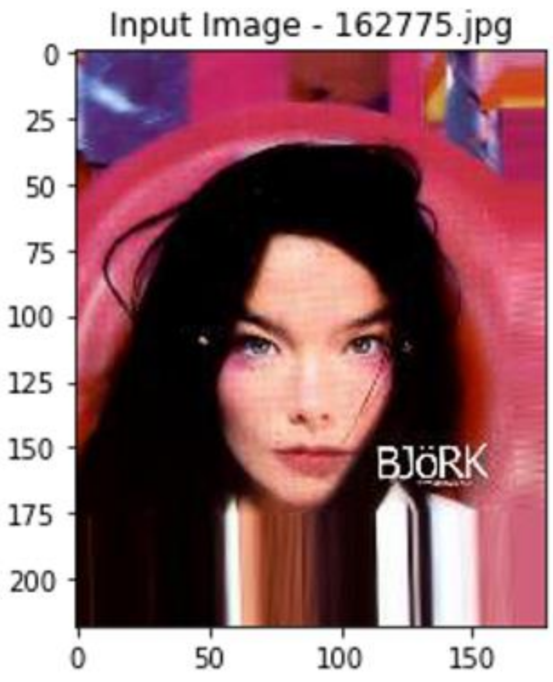
Table 1 shows per-attribute loss calculated for 20k images in the validation set. Note that the loss here is the cumulative average difference between the actual and predicted labels, or confidence scores, as we may say.

Figure 8 shows the corresponding bar chart for each loss. As is visible, our system does not perform well in detecting attributes viz., Big_Lips, Oval_Face, Pointy_Nose. While for some attributes like Bald, Eyeglasses, Sideburns, Wearing_Hat, etc., the model does quite well.

Similarity wise content retrieval:

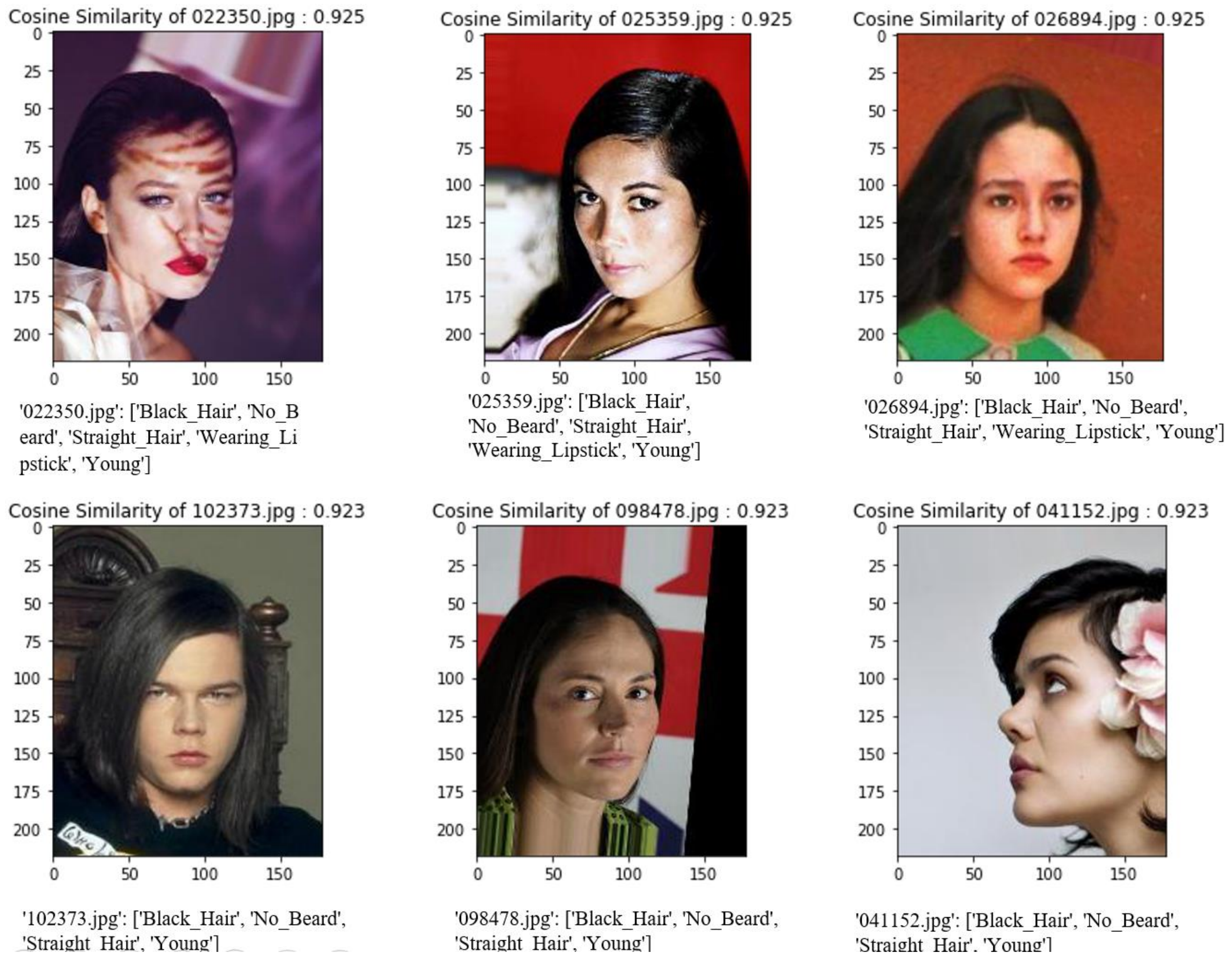
In image retrieval process, we used cosine similarity to compare similarity. We can choose an input image randomly from the test set. After selecting an image given by user, we will take input of how many top similar images the user likes to see. Then we iterate through our train images and store the information of image id and its cosine similarity value by comparing with input image. To get the attribute list of output images, we also stored ground truth of attribute list of corresponding images.

Predicted score for each attribute for the query image:



Attribute names	Probability Score
5_o_Clock_Shadow	2.3482e-03
Arched_Eyebrows	2.9373e-02
Attractive	1.7625e-01
Bags_Under_Eyes	4.2410e-02
Bald	9.7835e-06
Bangs	3.6118e-03
Big_Lips	3.2822e-01
Big_Nose	3.5797e-02
Black_Hair	9.2712e-01
Blond_Hair	6.6448e-04
Blurry	8.9973e-04
Brown_Hair	1.5098e-02
Bushy_Eyebrows	1.1895e-02
Chubby	1.6110e-03
Double_Chin	1.9090e-04
Eyeglasses	1.4368e-04
Goatee	2.2804e-03
Gray_Hair	1.9856e-05
Heavy_Makeup	8.0910e-02
High_Cheekbones	3.1982e-02
Male	4.7578e-02
Mouth_Slightly_Open	1.8520e-01
Mustache	2.5578e-03
Narrow_Eyes	2.8814e-01
No_Beard	9.8894e-01
Oval_Face	1.6015e-02
Pale_Skin	3.3927e-03
Pointy_Nose	8.6048e-02
Receding_Hairline	5.5876e-03
Rosy_Cheeks	4.6326e-04
Sideburns	4.6981e-04
Smiling	5.1269e-03
Straight_Hair	6.4954e-01
Wavy_Hair	5.6471e-02
Wearing_Earrings	1.3266e-02
Wearing_Hat	9.0894e-03
Wearing_Lipstick	4.2764e-01
Wearing_Necklace	1.3232e-02
Wearing_Necktie	3.6507e-04
Young	9.9224e-01

Figure 9: Input image with its respective attributes


Cosine Similarity of 022350.jpg : 0.925
Cosine Similarity of 025359.jpg : 0.925
Cosine Similarity of 026894.jpg : 0.925
Cosine Similarity of 102373.jpg : 0.923
Cosine Similarity of 098478.jpg : 0.923
Cosine Similarity of 041152.jpg : 0.923

```
'022350.jpg': ['Black_Hair', 'No_B
eard', 'Straight_Hair', 'Wearing_Li
pstick', 'Young']

'025359.jpg': ['Black_Hair',
'No_Beard', 'Straight_Hair',
'Wearing_Lipstick', 'Young']

'026894.jpg': ['Black_Hair', 'No_Beard',
'Straight_Hair', 'Wearing_Lipstick', 'Young']

'102373.jpg': ['Black_Hair', 'No_Beard',
'Straight_Hair', 'Young']

'098478.jpg': ['Black_Hair', 'No_Beard',
'Straight_Hair', 'Young']

'041152.jpg': ['Black_Hair', 'No_Beard',
'Straight_Hair', 'Young']
```

Figure 10: Output images

Above is an example show by giving an input of image_id 162775.jpg. After comparing each image's cosine similarity, we are displaying the top six images here. The first three images with 0.925 cosine similarity and the rest three have 0.923 cosine similarity. We can see our input image had probability score in attributes named "Black hair", "No Beard", "Straight hair", "Young". From the output images it is visible that all six images had these attributes ("Black hair", "No Beard", "Straight hair", "Young") which our similar image retrieval system a consistent and reliable system.

4.1 Challenges faced:

Challenge 1: Difficult to map images to attributes:

Mapping images to attributes was a challenging task as the data is separated into folders.

Therefore, we needed to build our own Custom Dataset object to solve this problem.

Challenge 2: CUDA Memory error:

The Lab 120 Machines were busy many a time and hence we could not get them when required. Moreover, we needed them for a large amount of time as it required 5+ hours to train our model. This was beyond our control.

We tried to mitigate this problem effect my starting the training process at night when traffic was less.

Challenge 3: Defining our model:

Initially, we faced blockers while trying to identify how the model needs to be designed (or re-designed) to suit our requirements. Finding an appropriate cost function was also a challenging task.

Looking up references on the internet and checking for relevant documentation helped us to get through it.

Challenge 4: Huge time required for training:

As the dataset is huge, it took a significant amount of time to train the model. Moreover, if anything goes wrong in the middle or at the end, we will encounter that after investing 5+ hours in training. This was a major issue as time was a crunch.

Solution: Increased our patience level.

5. Conclusion

Thus, we successfully built an image retrieval system with the use of ResNet-18 model. We attained a loss of 4.89 on our model. The model we built successfully takes an input image, extracts the attributes from the same and does a similarity match with pre-existed images in our dataset. Furthermore, we did our training in a distributed environment with 4+ nodes using the power of distributed PyTorch packages. As a part of future work, we plan to add image descriptors to our attribute vector to make our query system more efficient.

6. Future Work

As a part of future work, we can append a feature vector to the attribute array to find interesting patterns within the dataset and make our system detect various facial features. These features can then be used for retrieval tasks. To calculate the features, we can make use of existing pre-trained CNNs (Convolutional Neural Network), wherein, the model will output the image descriptor through a linear layer at the end. This linear layer will be immediately after the last convolution layer in our pre-trained architecture.

7. Contributions

A detailed list of all the tasks completed by the team members, Mridul Banik, Sanket Mayuresh Bhave and Shreya Anilkumar are given below in the contributions table. The entire project duration was four weeks.

Week	**Tasks completed by each Team member**
Week 1 April 3 – April 9	Mridul Banik: Explored and understood the Celebrity (CelebA) Dataset (Binary annotation, landmark location, bounding box) and selected the required data based on the project requirement. Understood the concepts of distributed pytorch (Parallel Application). Sanket Mayuresh Bhave: Explored and understood the Celebrity (CelebA) Dataset (Binary annotation, landmark location, bounding box) and selected the required data based on the project requirement. Worked on a Proof of concept using the CelebA dataset and started data preprocessing. Shreya Anilkumar: Explored and understood the Celebrity (CelebA) Dataset (Binary annotation, landmark location, bounding box) and selected the required data based on the project requirement. Understood the concepts of distributed pytorch (Parallel Application).
Week 2 April 10 – April 16	Mridul Banik: Explored and understood on writing distributed applications with pytorch. Researched to choose a model for transfer learning. Sanket Mayuresh Bhave: Worked on a Proof of concept using the CelebA dataset, completed data preprocessing, worked on cosine similarity, train functions etc. Shreya Anilkumar: Explored and understood on writing distributed applications with pytorch. Started working on POC to setup the environment and implement dataset portioning on a dummy dataset.
Week 3 April 17- April 23	Mridul Banik: Worked on Cosine similarity and on prediction. Explored similarity metrics (Distance Mapping) and Selection of deciding factors for image similarity (Threshold). Sanket Mayuresh Bhave: Worked on Distributed Data Parallel, modeling, and training of resnet18. Explored similarity metrics (Distance Mapping) and Selection of deciding factors for image similarity (Threshold). Shreya Anilkumar: Started filling, editing, and reviewing the term project report.

	Explored similarity metrics (Distance Mapping) and Selection of deciding factors for image similarity (Threshold).
Week 4 April 24- April 30	Mridul Banik: Experimentation and evaluation of parameters. Report reviews and editing. Sanket Mayuresh Bhave: Experimentation and evaluation of parameters. Report reviews and editing. Shreya Anilkumar: Filling and completing the term project report. Report reviews and editing.

8. Bibliography

[1] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016 pp. 2818-2826.
doi: 10.1109/CVPR.2016.308
url: https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.308
[2] Chollet, François. "Xception: Deep learning with depthwise separable convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.
[3] Dengsheng Zhang and Guojun Lu, "Evaluation of similarity measurement for image retrieval," International Conference on Neural Networks and Signal Processing, 2003. Proceedings of 2003, 2003, pp. 928-931 Vol.2, doi: 10.1109/ICNNSP.2003.1280752.
[4] H. Abdel-Nabi, G. Al-Naymat and A. Awajan, "Content Based Image Retrieval Approach using Deep Learning," 2019 2nd International Conference on new Trends in Computing Sciences (ICTCS), 2019, pp. 1-8, doi: 10.1109/ICTCS.2019.8923042.
[5] He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.
[6] I. Azhar, I. Afyouni and A. Elnagar, "Facilitated Deep Learning Models for Image Captioning," 2021 55th Annual Conference on Information Sciences and Systems (CISS), 2021, pp. 1-6, doi: 10.1109/CISS50987.2021.9400209.
[7] I. C. Amitha and N. K. Narayanan, "Object Retrieval in Images using SIFT and R-CNN," 2020 International Conference on Innovative Trends in Information Technology (ICITIIT), 2020, pp. 1-5, doi: 10.1109/ICITIIT49094.2020.9071557.
[8] K. A. Shaheer Abubacker and L. K. Indumathi, "Attribute associated image retrieval and similarity reranking," 2010 International Conference on Communication and Computational Intelligence (INCOCCI), 2010, pp. 235-240.

[9] M. Kokare, B. N. Chatterji and P. K. Biswas, "Comparison of similarity metrics for texture image retrieval," TENCON 2003. Conference on Convergent Technologies for Asia-Pacific Region, 2003, pp. 571-575 Vol.2, doi: 10.1109/TENCON.2003.1273228.
[10] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).
[11] S. Li and L. Huang, "Context-based Image Caption using Deep Learning," 2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP), 2021, pp. 820-823, doi: 10.1109/ICSP51882.2021.9408871.
[12] *ImageNet*. http://www.image-net.org
[13] https://infospaces.cs.colostate.edu/watch.php?id=225
[14] https://www.linkedin.com/pulse/content-based-feature-extraction-image-retrieval-using-santosh-sawant/
[15] https://www.mathworks.com/help/deeplearning/ref/resnet18.html
[16] https://medium.com/analytics-vidhya/extracting-attributes-from-image-using-multi-label-classification-based-on-hypotheses-cnn-pooling-e2f9ce80461
[17] https://medium.com/codex/a-comprehensive-tutorial-to-pytorch-distributeddataparallel-1f4b42bb1b51

[18] https://pytorch.org/tutorials/intermediate/dist_tuto.html#distributed-training

[19] https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html
[20] https://pytorch.org/tutorials/intermediate/ddp_tutorial.html

[21] https://www.sciencedirect.com/topics/computer-science/residual-network

[22] https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a