Assignment-3&4
Report

of

CS-673: Advanced Computer Vision

Submitted to

Dr. Dinesh Singh, Dr. Rohit Saluja

Submitted by

Nandani Sharma (D22180)
Kajal (S23083)

Table of Contents

Question-1

MTCNN

The MTCNN implementation is derived from Facenet's and combined into a single file located under the 'data' folder in relation to the module's directory. By adding it to the MTCNN() constructor during instantiation, it may be overridden.

- **Code:** Code and implementation details can be found on GitHub.
- **Datasets:** CASIA-WebFace, VGGFace2.

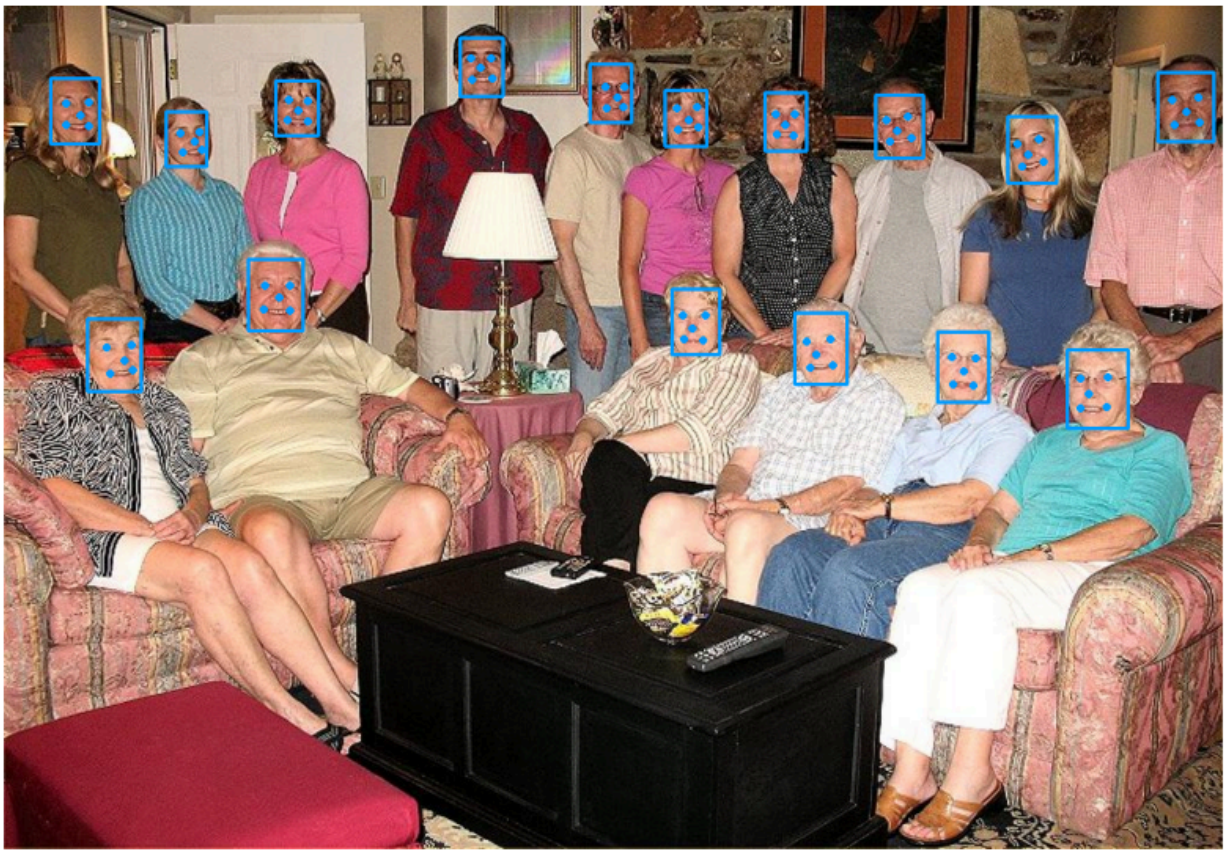


Figure-1: Result of multiple face detection during testing

Inference time per step: 33/33 [==============================] - 0s 8ms/step

FaceNet

Using the facenet-pytorch package, one may create a basic deepfake detector without having to train any models. Additionally, it shows how to load every frame of the movie, locate every face, and compute face embeddings at a pace of more than 30 frames per second, or more than one video every ten seconds.

- **Code:** Code and implementation details can be found on GitHub.
- **Datasets:** VGGFace2.

Output:

- HBox(children=(IntProgress(value=0, max=400), HTML(value='')))
- Frames per second (load+detect+embed): 48.7

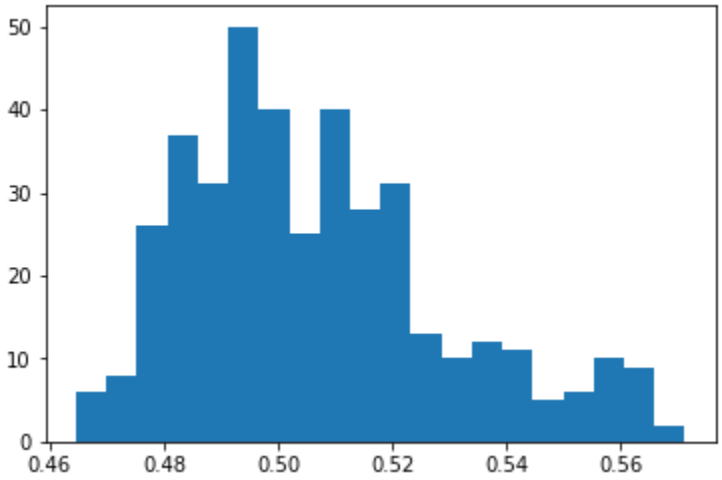


Figure-2: histogram Plot

Face recognition siamese network:

Face Verification: A one-to-one comparison between a template face picture whose identity is being claimed and a query face image.

Face Identification: A one-to-many match that establishes the identification of the query face by comparing it to every template picture stored in the database.

- **Models:** CelebA face Recognition, FaceNet, LFW

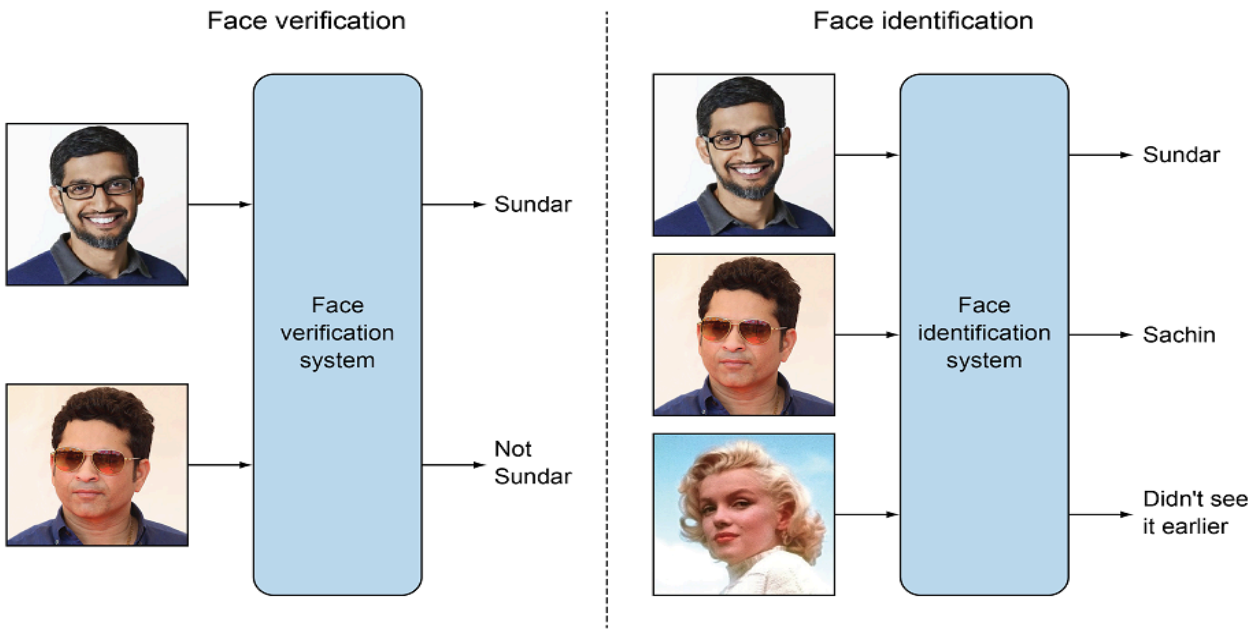


Figure3: Face verification and identification

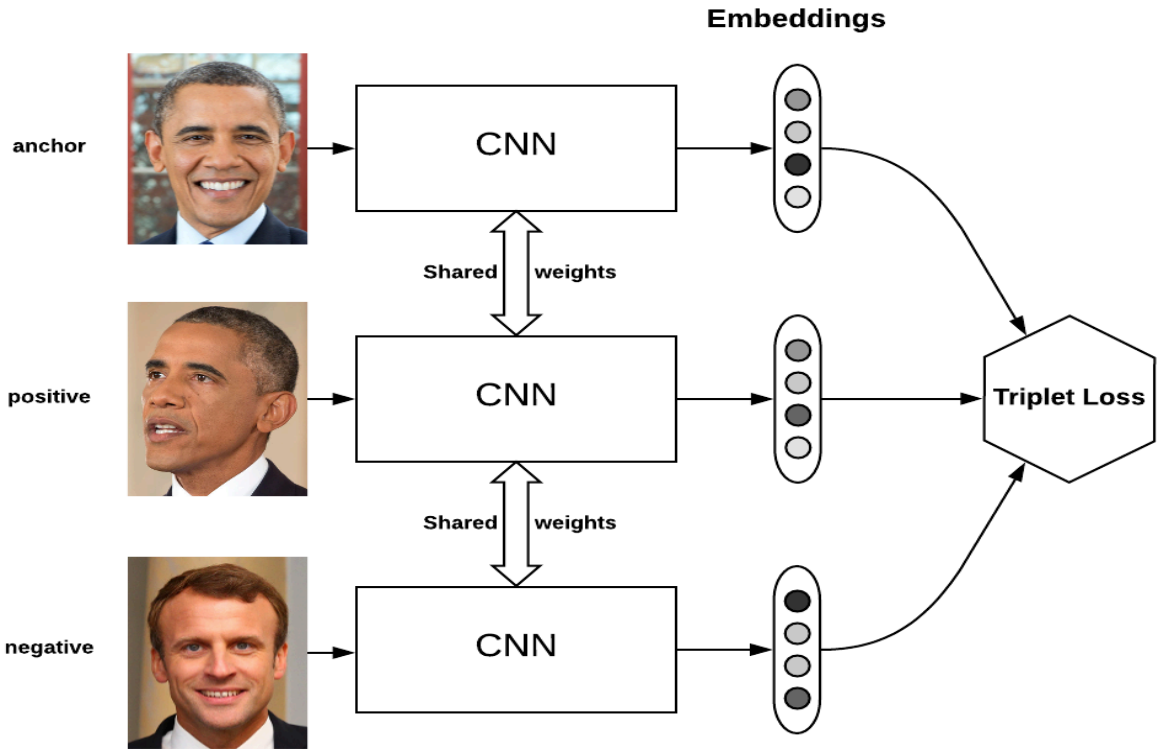


Figure-4: Siamese neural network for Face recognition

- A Siamese neural network is a type of artificial neural network that computes identical output vectors by operating in parallel on two separate input vectors using the same weights.
- Siamese networks may be used for a variety of purposes, such as face recognition, anomaly detection, and duplication detection.
- Although there are several methods for training a Siamese network, we will use the triplet loss function in this case.
- Triplet loss is a type of loss function used in machine learning algorithms, in which a non-matching input (referred to as negative) and a matching input (referred to as positive) are compared with a reference input (referred to as anchor).
- We generate triplets to train our model

1. **Anchor:** An input image
2. **Positive:** Same image as the identity in the input image
3. **Negative:** Different image than the identity in the input image

Output

Epoch 00006: val_loss did not improve from 0.02929

Epoch 00006: ReduceLROnPlateau reducing learning rate to 2.9999999242136255e-05.
Epoch 7/50
448/448 [==============================] - 78s 175ms/step - loss: 0.0015 - val_loss: 0.0380

A threshold of 1.65 gives correct decisions 83.55% of the time.

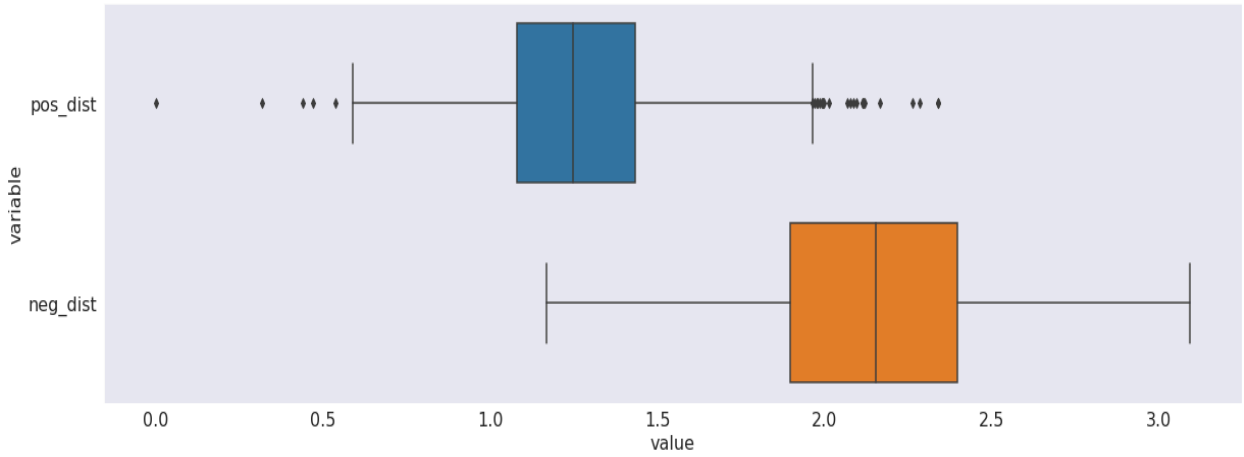


Figure-5: descriptive stats and the IQR Box plots, trying out values between 1.5-2.0 for threshold.

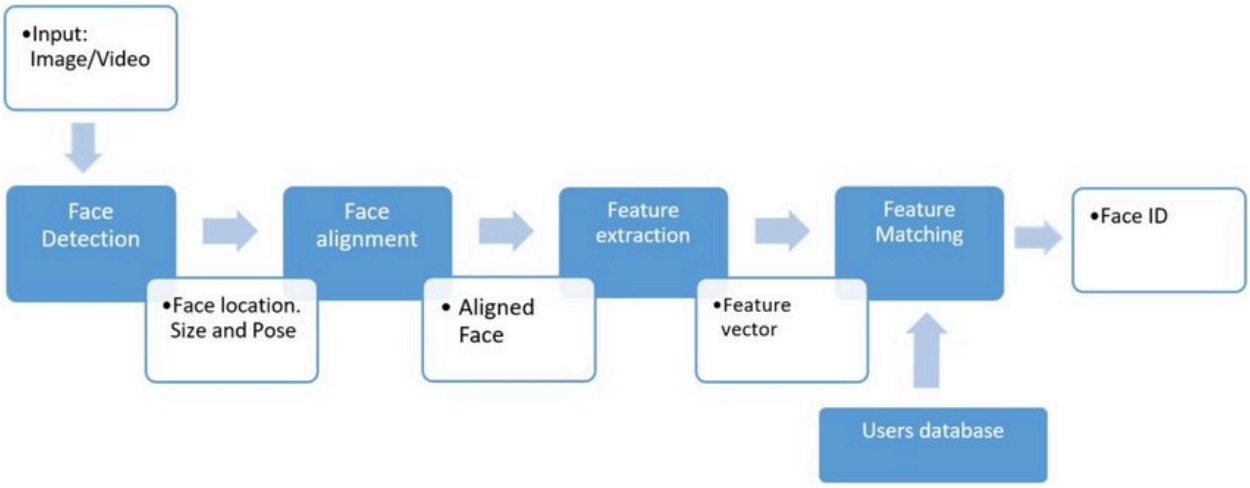


Figure-6: Inference Pipeline for Face Recognition Systems

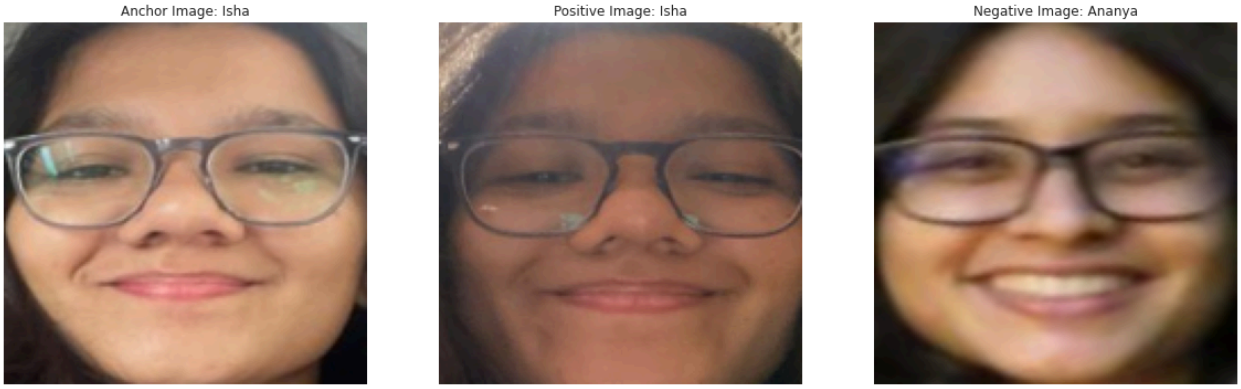


Figure-7: Face Verification

Output

- Distance Between Anchor & Positive: The L2 distance is less than 1.65
- Distance Between Anchor & Negative: The L2 distance is greater than 1.65

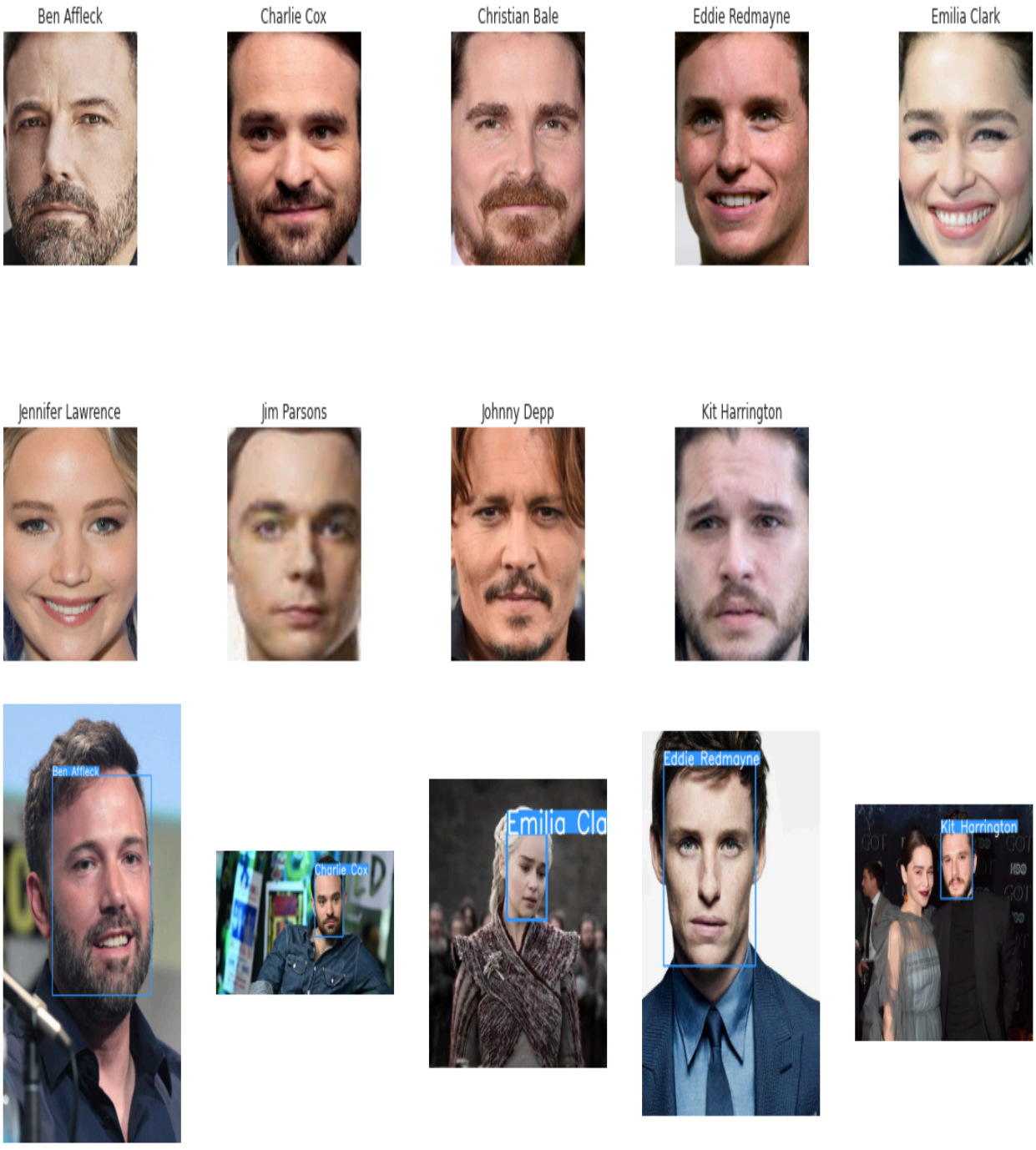


Figure-7: Face Identification Query images

Face-recognition-on-olivetti-dataset: using PCA

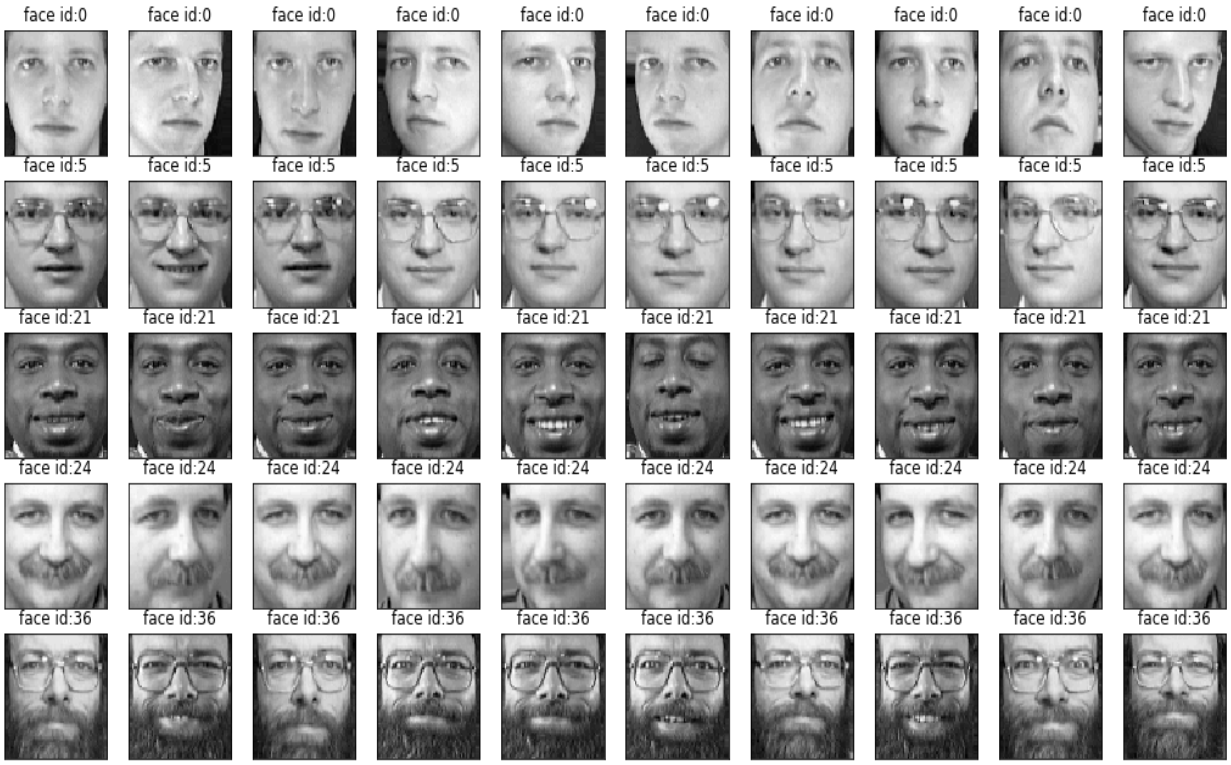


Figure-8: playaround subject_ids to see other people faces

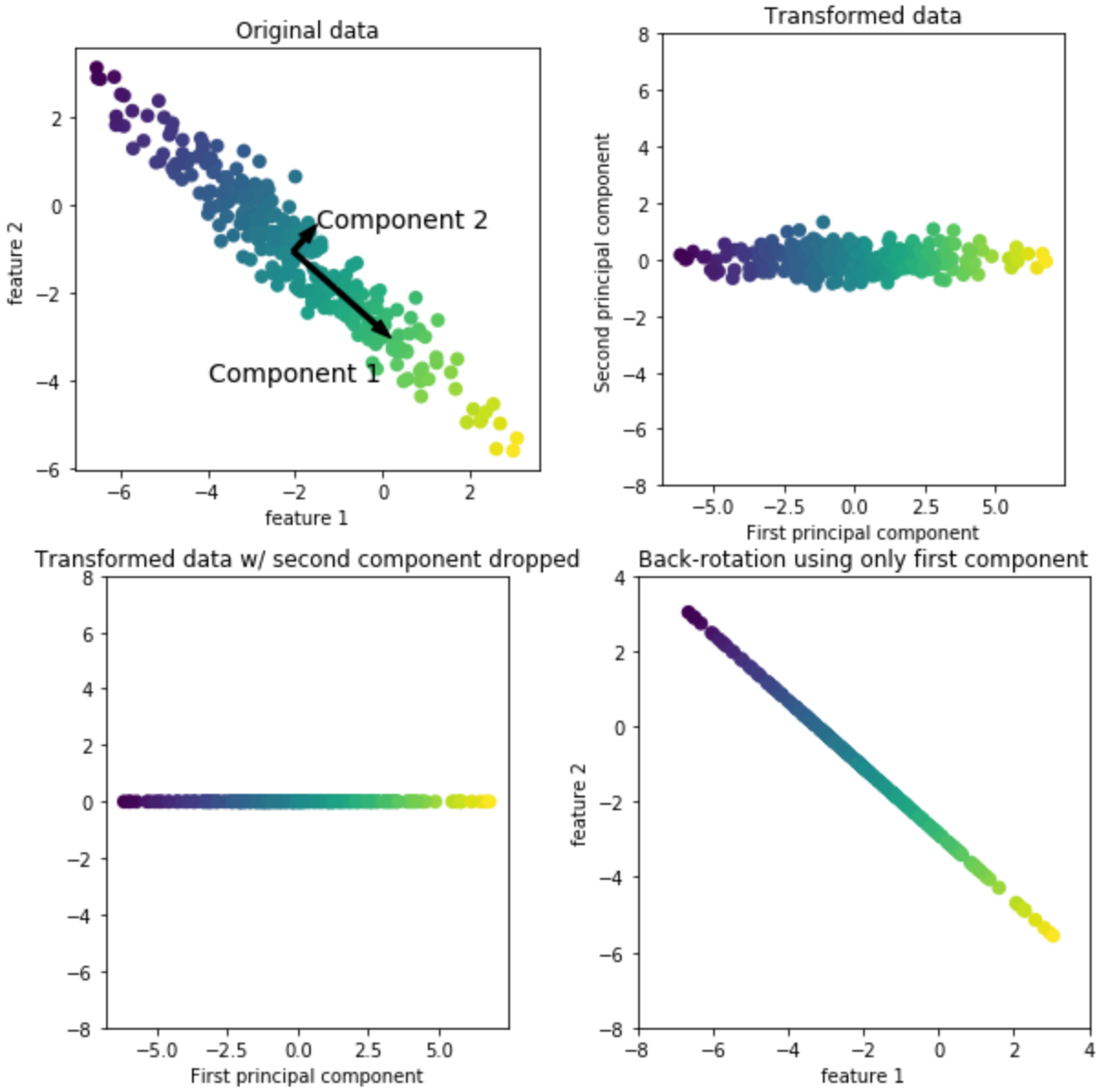


Figure-9: shows a simple example on a synthetic two-dimensional data set

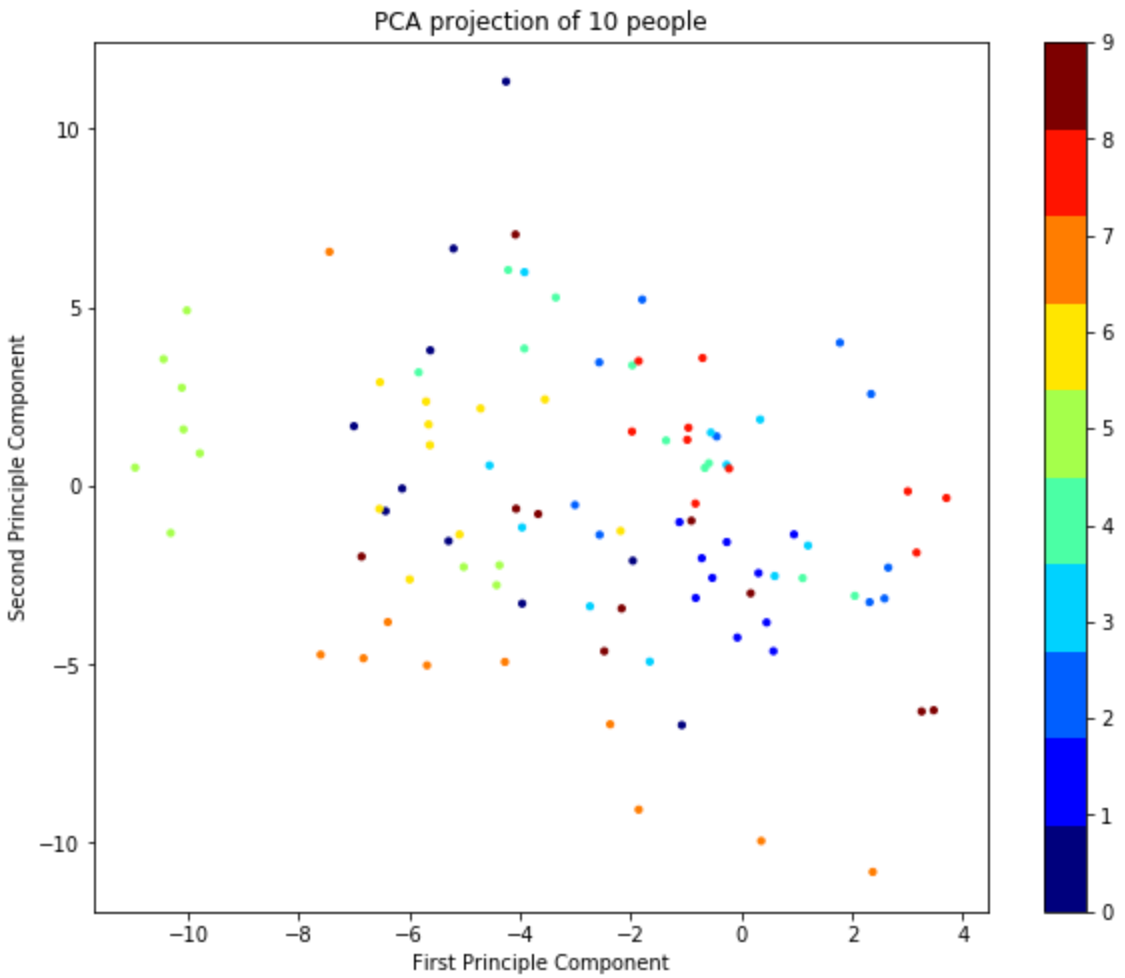


Figure-10: PCA projection of 10 people

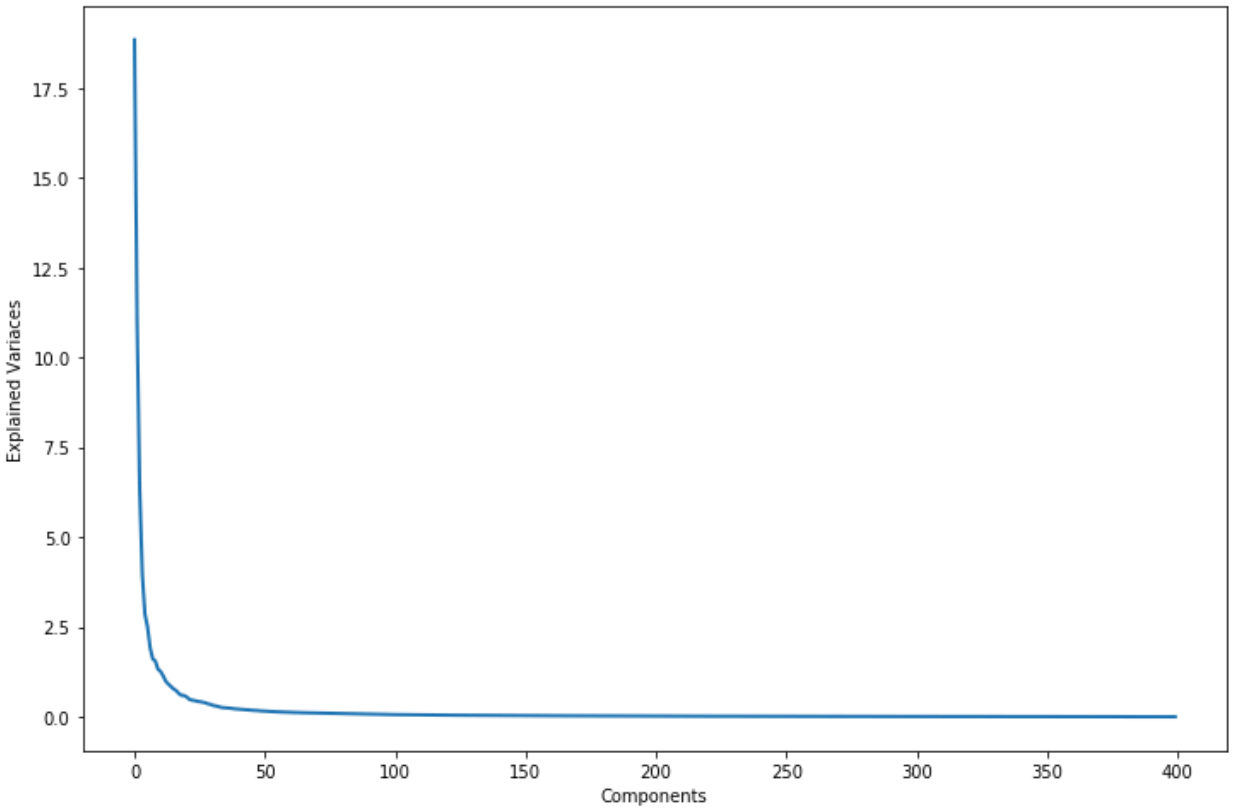


Figure-11: 90 and more PCA components represent the same data

Figure-12: Eigen faces of all people ids

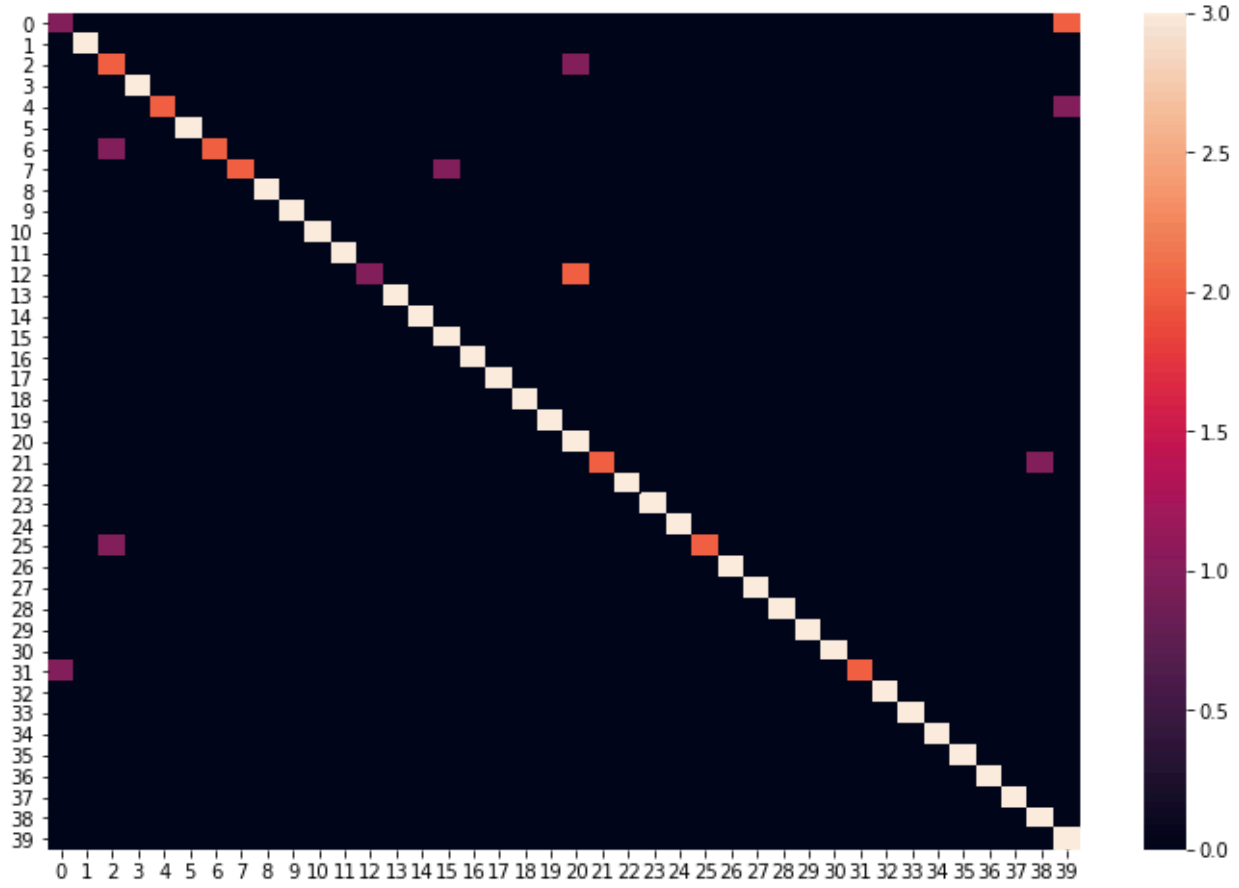


Figure-13: Heatmap confusion matrix

Output

- micro avg 0.94 0.94 0.94 120
- macro avg 0.95 0.94 0.94 120
- weighted avg 0.95 0.94 0.94 120

Face-recognition-grid-search

Figure-14: sample representation

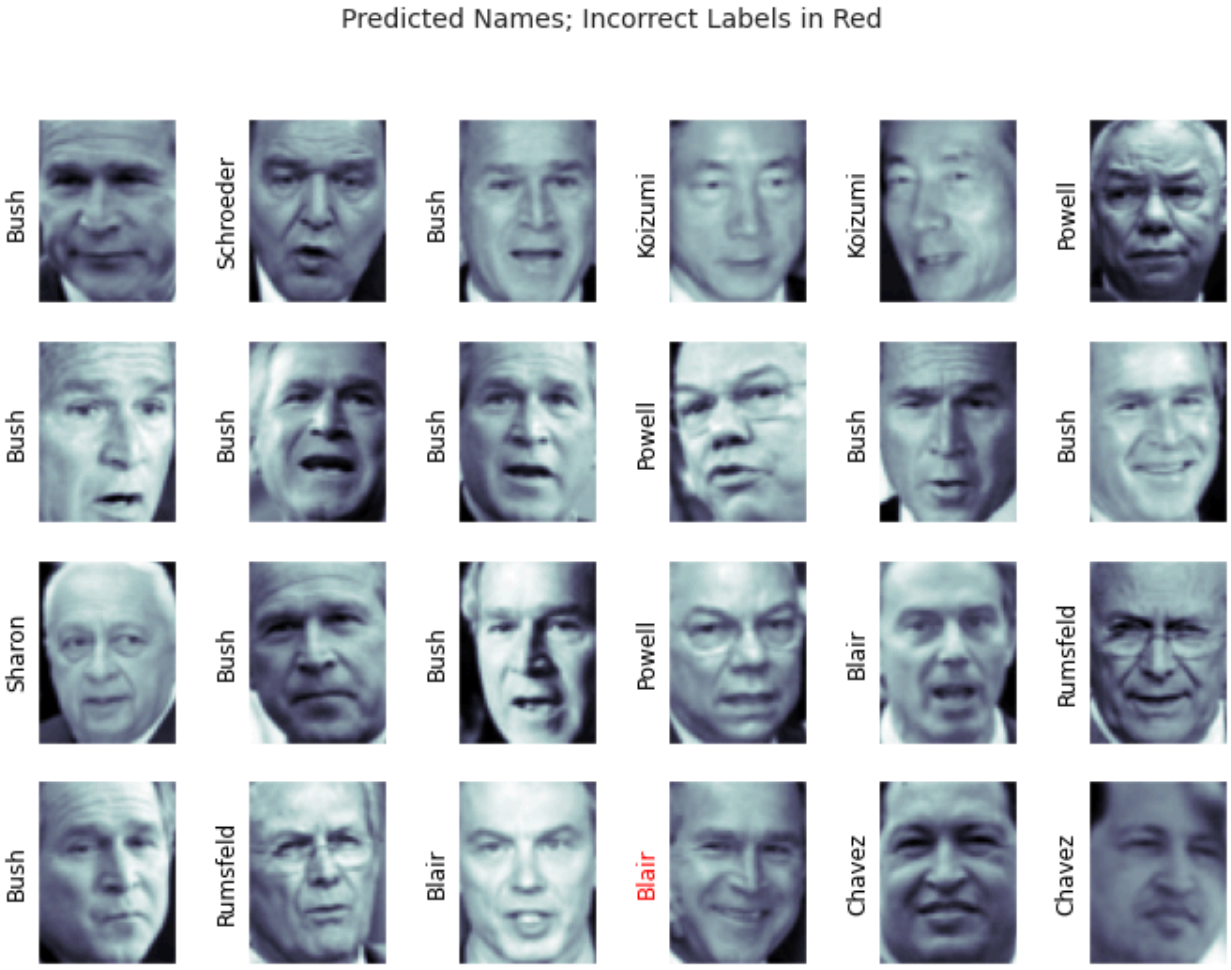


Figure-15: Predicted Results

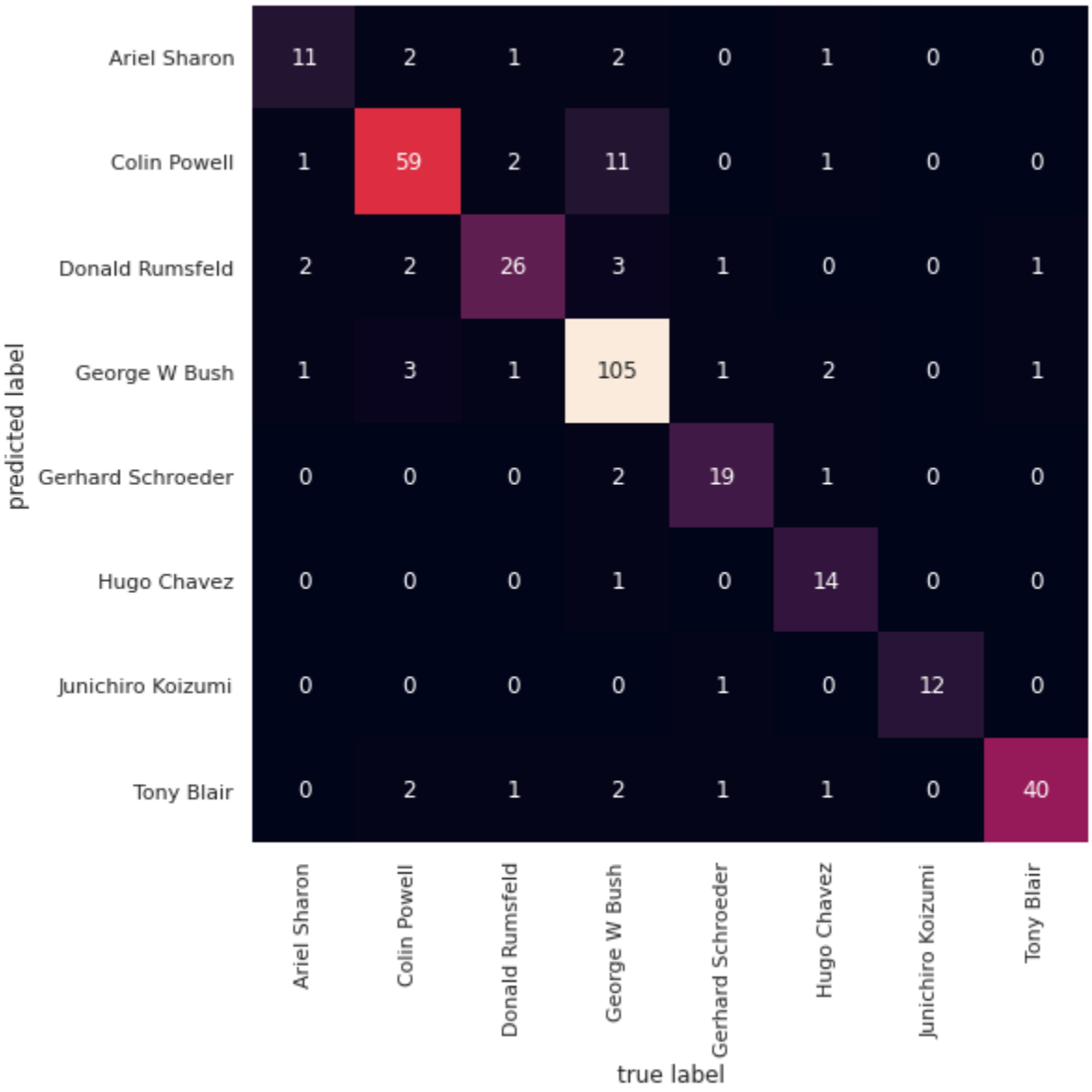


Figure-16: Heatmap confusion matrix

Question-2

U-net

It's a convolutional neural network (CNN), and its primary application in computer vision and image processing is semantic segmentation. With a contracting path (encoder) to collect context and an expanding path (decoder) to rebuild high-resolution segmentation masks, it has a U-shaped structure. Because skip links between similar layers make information flow throughout the network easier, U-Net can generate precise pixel-wise predictions. This makes it especially useful for jobs requiring precise spatial information, such biological picture segmentation.

Dataset : pets dataset
Length: 63219 (62K) [image/jpeg]

Figure-17: Original image, mask and mask+image

Testing Results :
1/1 [==============================] - 3s 3s/step

Figure-18: Original image, mask and mask+image at testing time

DeepLabv3

Semantic segmentation tasks are another use for it. In order to effectively collect multi-scale context information, it uses atrous convolutions. DeepLabv3 combines local context information with global context information via spatial pyramid pooling and dilated convolutions. It is used in many different domains, including medical image analysis, object segmentation, and scene parsing.

Dataset : Chest X-ray

Results : Original image vs segmented image

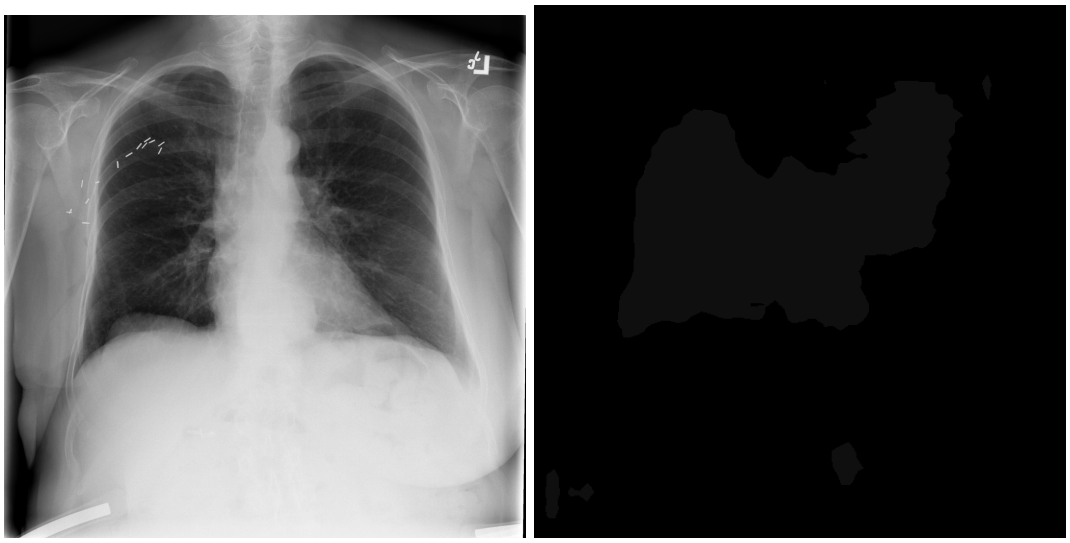


Figure-19: Chest X-ray original image and segmented image

Inference Time: 12.769865274429321 seconds/image
Memory Allocated: 187.0849609375 MB

Question-3

Person Re-Identification

HR-net

This deep learning network is intended for jobs involving high-resolution images. By keeping up high-resolution representations across the network, it is able to efficiently capture fine-grained features. HRNet improves both local and global context awareness by integrating characteristics from several resolution levels through the use of a multi-resolution fusion technique.
HRNet is well-known for its efficiency in tasks like as object identification, semantic segmentation, and human posture estimation. It also performs very well when processing high-resolution inputs and generating precise predictions.

Code : code and implementation details are here Github

Dataset : CrowdPose
No. of Images : 20,000

Training Implementation Details :

BEGIN_EPOCH: 0
CHECKPOINT:
END_EPOCH: 300
GAMMA1: 0.99
GAMMA2: 0.0
IMAGES_PER_GPU: 12
LR: 0.001
LR_FACTOR: 0.1
LR_STEP: [200, 260]
MOMENTUM: 0.9
NESTEROV: False
OPTIMIZER: adam
RESUME: False
SHUFFLE: True
WD: 0.0001

Total Parameters: 28,641,962

Total Multiply Adds (For Convolution and Linear Layers only): 47,653,584,896

Number of Layers
Conv2d : 302 layers BatchNorm2d : 301 layers ReLU : 270 layers Bottleneck : 4 layers
BasicBlock : 108 layers Upsample : 28 layers HighResolutionModule : 8 layers
ConvTranspose2d : 1 layers
2024-05-12 22:56:38,847 => loading model from
models/pytorch/imagenet/hrnet_w32-36af842e.pth
2024-05-12 22:56:51,181 => classes: ['__background__', 'person']

Testing Results :

Total Parameters: 28,641,962

Total Multiply Adds (For Convolution and Linear Layers only): 47,653,584,896

Number of Layers
Conv2d : 302 layers BatchNorm2d : 301 layers ReLU : 270 layers Bottleneck : 4 layers
BasicBlock : 108 layers Upsample : 28 layers HighResolutionModule : 8 layers
ConvTranspose2d : 1 layers
=> loading model from models/pytorch/imagenet/hrnet_w32-36af842e.pth
loading annotations into memory...
Done **(t=3.93s)**
creating index...
index created!
=> classes: ['__background__', 'person']

Question-4

Activity Recognition :

The practice of automatically recognizing an individual's actions using information from sensors such as gyroscopes and accelerometers is known as activity recognition. It gathers information, extracts characteristics, trains models, and then predicts actions in a variety of fields, including security, healthcare, and fitness monitoring.

Dataset : HAR dataset
Length: 529341 (517K)

Results :

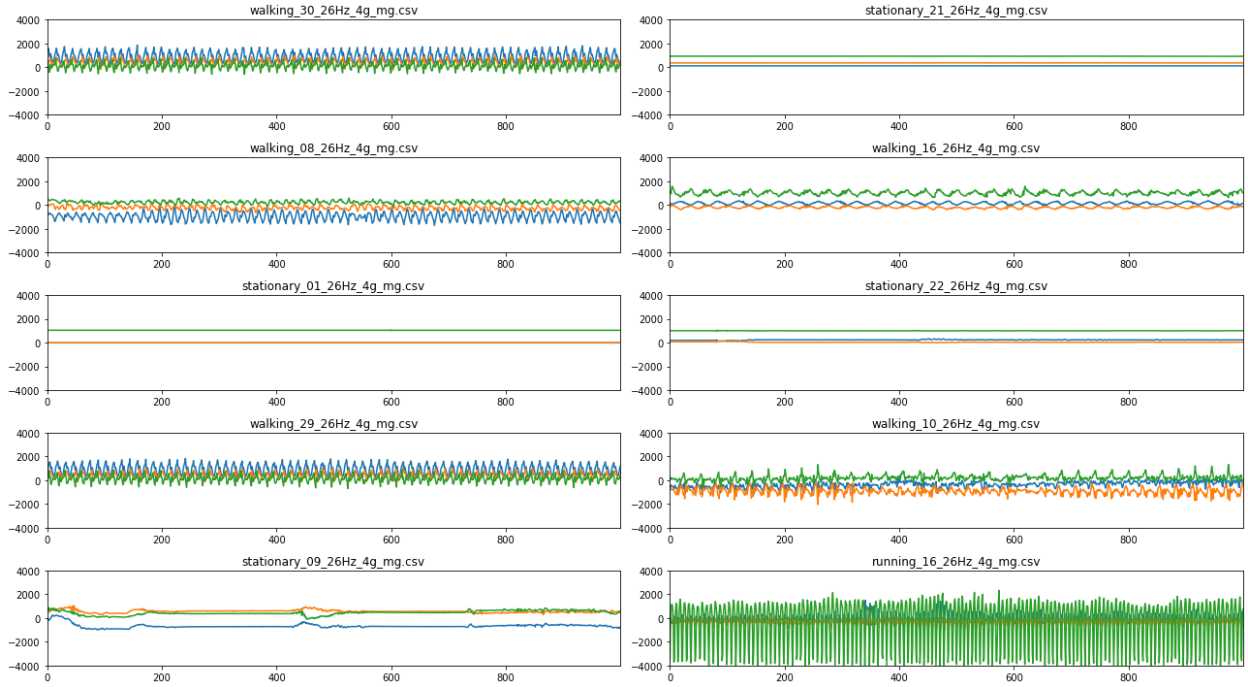


Figure-20: Human activity recognition recording at walking, stationary and running time

(92, 75, 26, 3)
(6900, 26, 3)
(6900,)
['stationary', 'walking', 'running']

Trainning samples: (5175, 26, 3)
Testing samples: (1725, 26, 3)

Epoch 30/30
162/162 [==============================] - 1s 3ms/step - loss: 0.0365 - accuracy: 0.9876
54/54 - 0s - loss: 0.0302 - accuracy: 0.9907 - 268ms/epoch - 5ms/step
Test loss: 0.030187616124749184
Test acc: 0.9907246232032776
Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 24, 16)	160
conv1d_1 (Conv1D)	(None, 22, 8)	392
dropout (Dropout)	(None, 22, 8)	0
flatten (Flatten)	(None, 176)	0
dense (Dense)	(None, 64)	11328
dense_1 (Dense)	(None, 3)	195

Total params: 12075 (47.17 KB)
Trainable params: 12075 (47.17 KB)
Non-trainable params: 0 (0.00 Byte)

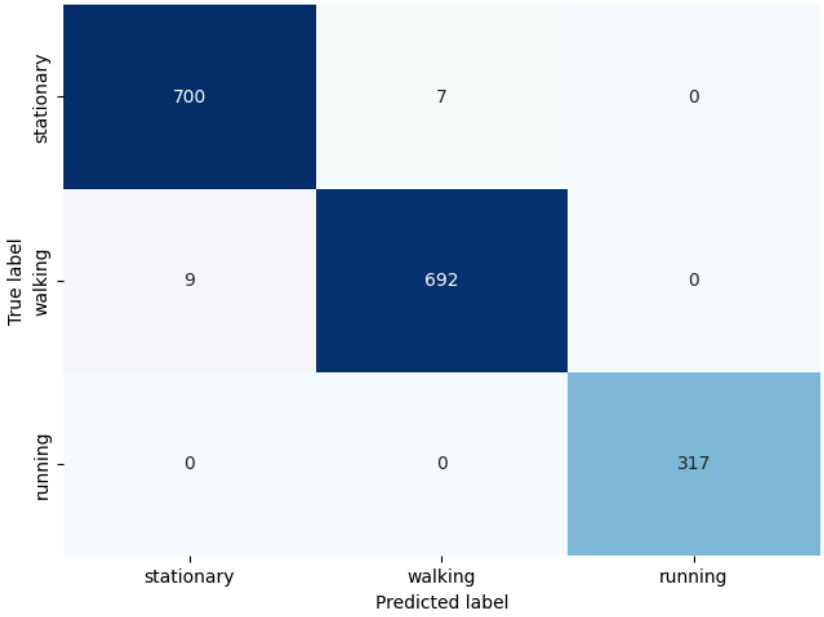


Figure-21: Confusion Matrix for activity recognition

Question-5

GAN

A generator and a discriminator neural network that have been trained simultaneously make up a generative adversarial network. While the discriminator learns to discern between actual and created samples, the generator learns to produce realistic data samples, such as photographs. GANs are commonly utilized for tasks like picture creation, style transfer, and data augmentation in diverse domains including computer vision and natural language processing. This is because they become increasingly realistic outputs via adversarial training.

Dataset : MNIST

Training Implementation details :

```
num_epochs = 5
batch_size = 128
learning_rate = 2e-4
```

Number of parameters for generator: 12656257 and discriminator: 11033985

Results :

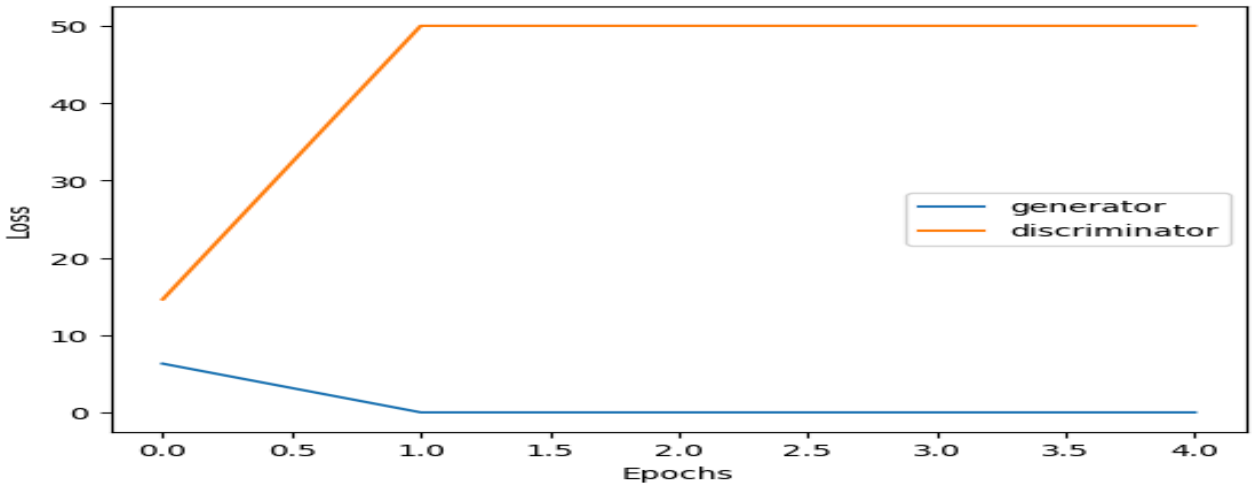


Figure-22: generator vs discriminator graph for GAN

VAE

Neural networks called variational autoencoders (VAEs) are made to understand the underlying structure of data and produce new samples that resemble the training set. By compressing the input data, they are able to create new data points and represent them efficiently in a low-dimensional latent space. In comparison to conventional autoencoders, VAEs generate data with greater robustness and flexibility because they employ probabilistic methodologies to capture the underlying uncertainty in the data. Among other things, they find use in feature learning, data compression, and picture creation.

Dataset : MNIST
Implementation Details :

```
num_epochs = 5
batch_size = 16
capacity = 64
learning_rate = 1e-3
variational_beta = 1
```

average reconstruction error: 2458.309588

Results :

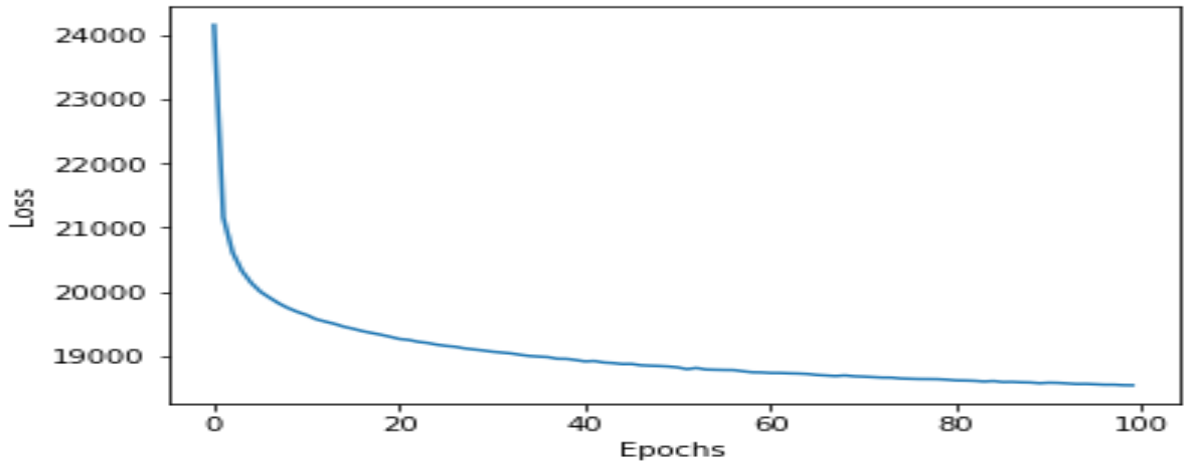


Figure-23: reconstruction loss graph for VAE

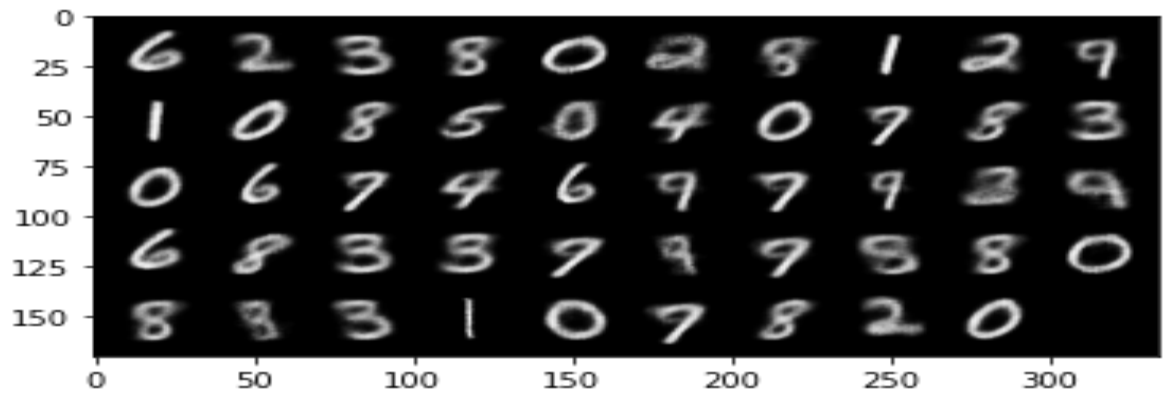


Figure-24: VAE reconstruction image

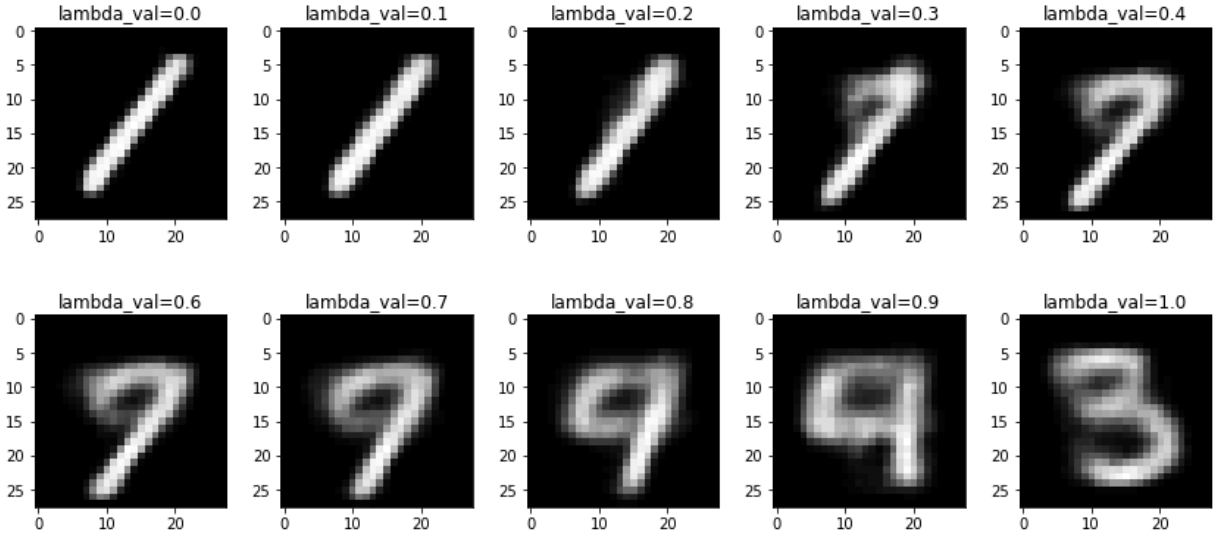


Figure-25: Latent space output for different lambda values

Question-6

- A library called **GraKeL** offers implementations of a number of popular graph kernels. These kernels are combined by the library into a single framework. Moreover, it implements a few frameworks that utilize graph kernels as their foundation. In particular, GraKeL has two frameworks and sixteen kernels.
- The library is compatible with the scikit-learn pipeline allowing easy and fast integration inside machine learning algorithms.
- **Code:** Code and implementation details can be found on GitHub.
- **Datasets:** Links to datasets used in the paper:
- Digit Classification (MNIST): The dataset is contained in scikit-learn and we can load it using the *load_digits()* function
- Text Classification: [Dataset]

Digit Classification:

Figure-26: sample representation

Output

- Computing kernel matrices done in 37.890s

- Classifying digits Classification accuracy: 0.85

Figure-27: Predicted samples

Text Categorization

Vocabulary size: 7186

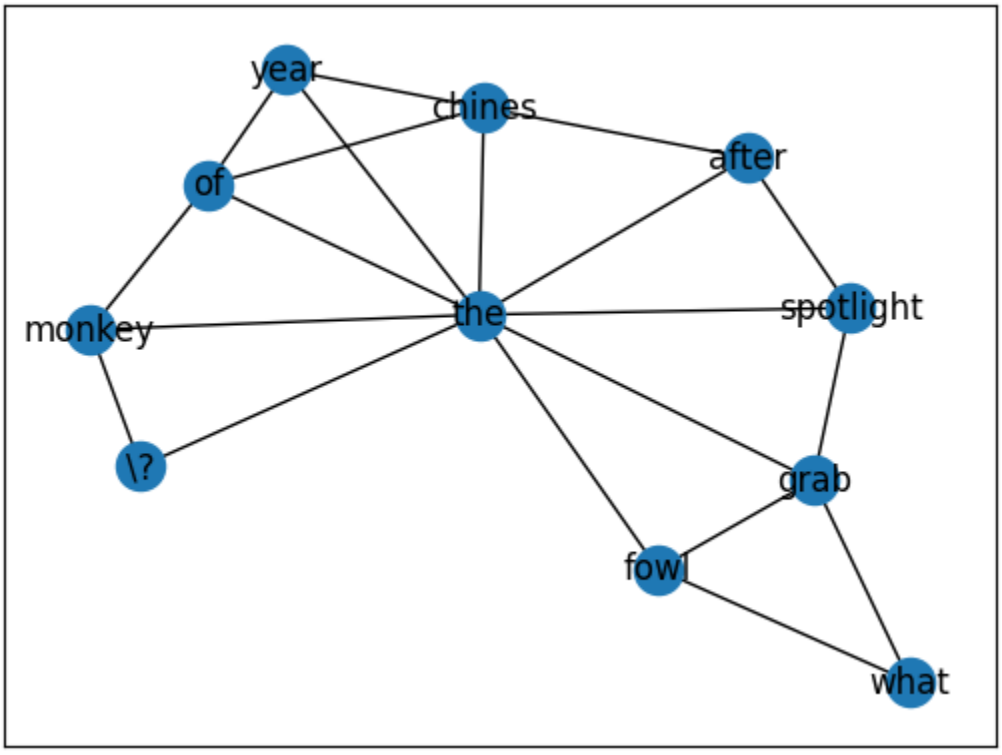


Figure-28: Vocabulary connectivity representation

Output-> Accuracy: 0.858

Extract SIFT features from the MNIST dataset outputs

- Number of nodes: 98
- Number of edges: 338
- Running time 7.18 ms

Question-7

Graph Attention Networks

- One example of a spatial (convolutional) GNN is **GAT**. Because CNNs were so successful in computer vision, researchers wanted to extend the technology to graphs.
- **Code:** Code and implementation details can be found on GitHub.
- **Datasets:** Links to datasets used in the paper:
- Protein-Protein Interactions [Source] [Preprocessed]
- Cora [Preprocessed]

Cora Dataset

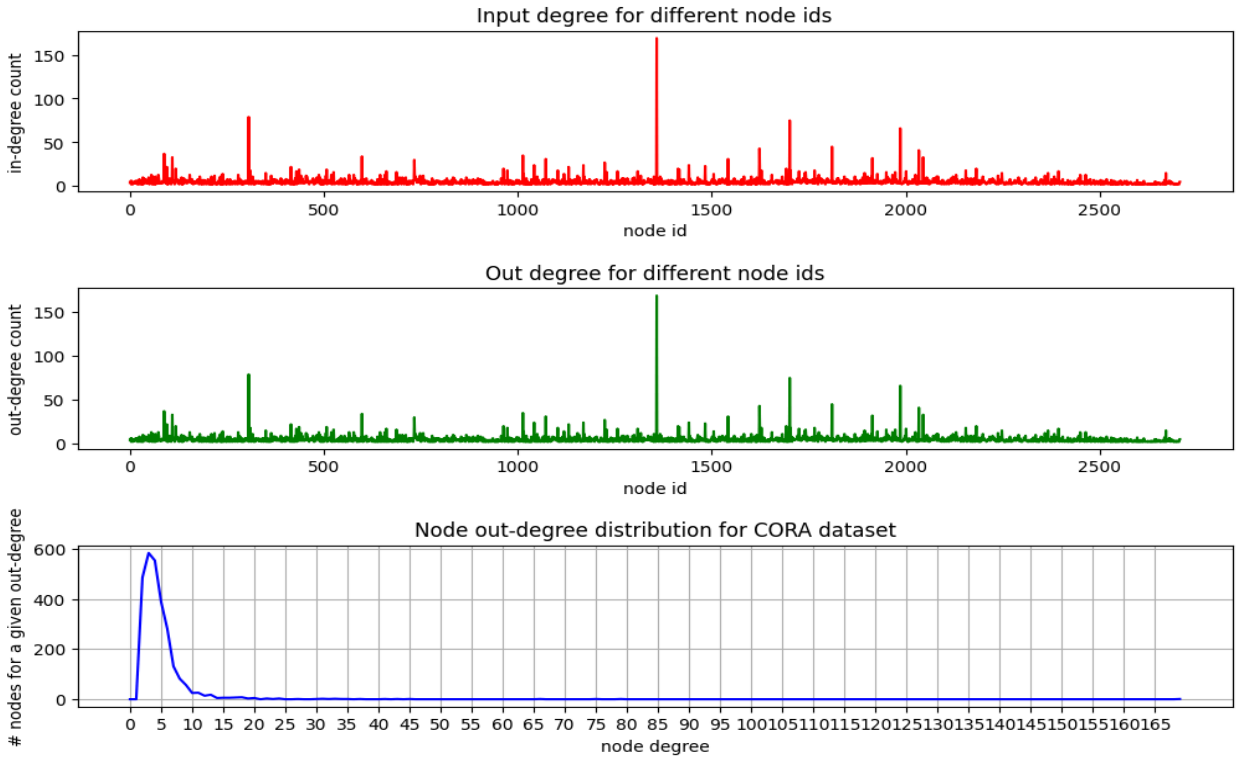


Figure-29: visualizes Cora's degree distributions.

GAT training: time elapsed= 11.85 [s] | epoch=1301 | val acc=0.794

Test accuracy = 0.827

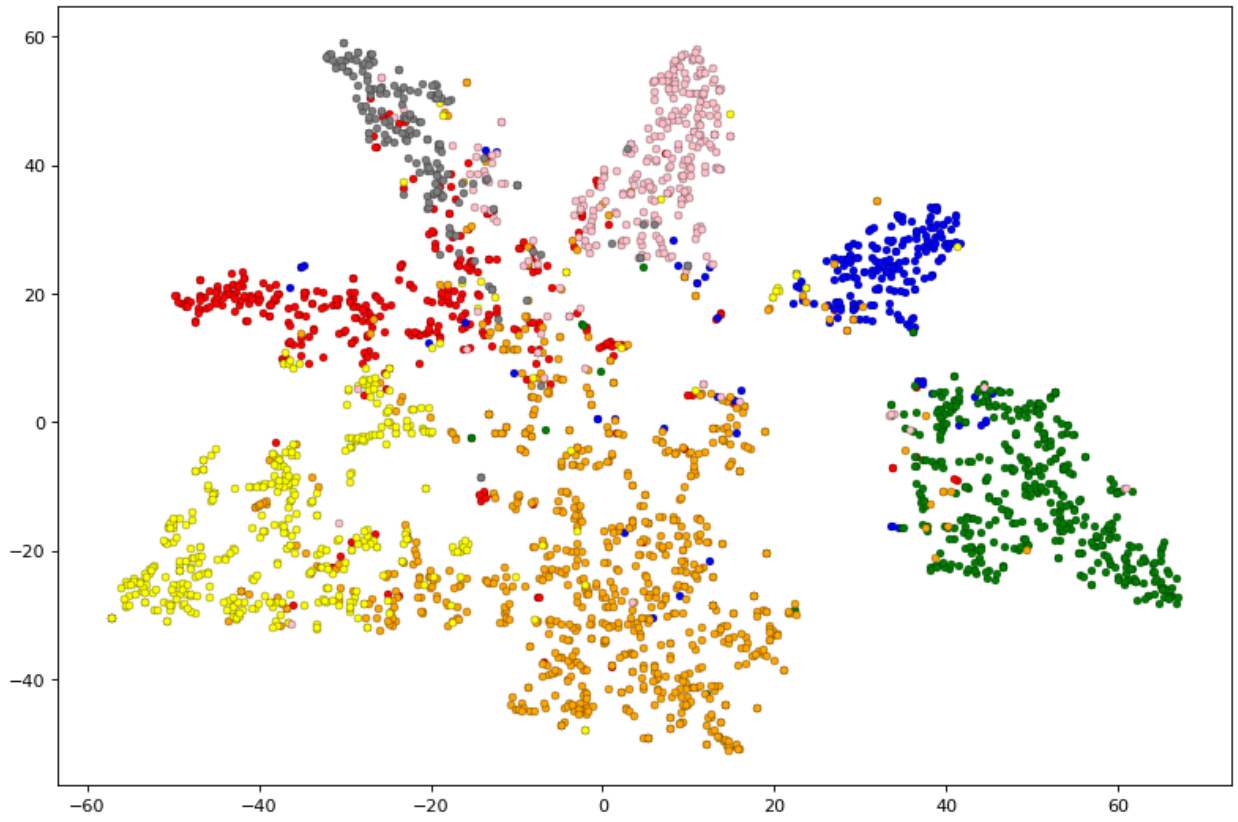


Figure-30: Visualizing GAT's embeddings using t-SNE

Output

***** Model training metadata: *****
commit_hash: 91fb864b8f9ddefd401bf5399cea779bd3c0a63b
dataset_name: CORA
num_of_epochs: 10000
test_acc: 0.822
num_of_layers: 2
num_heads_per_layer: [8, 1]
num_features_per_layer: [1433, 8, 7]
add_skip_connection: False
bias: True
dropout: 0.6
layer_type: IMP3

****

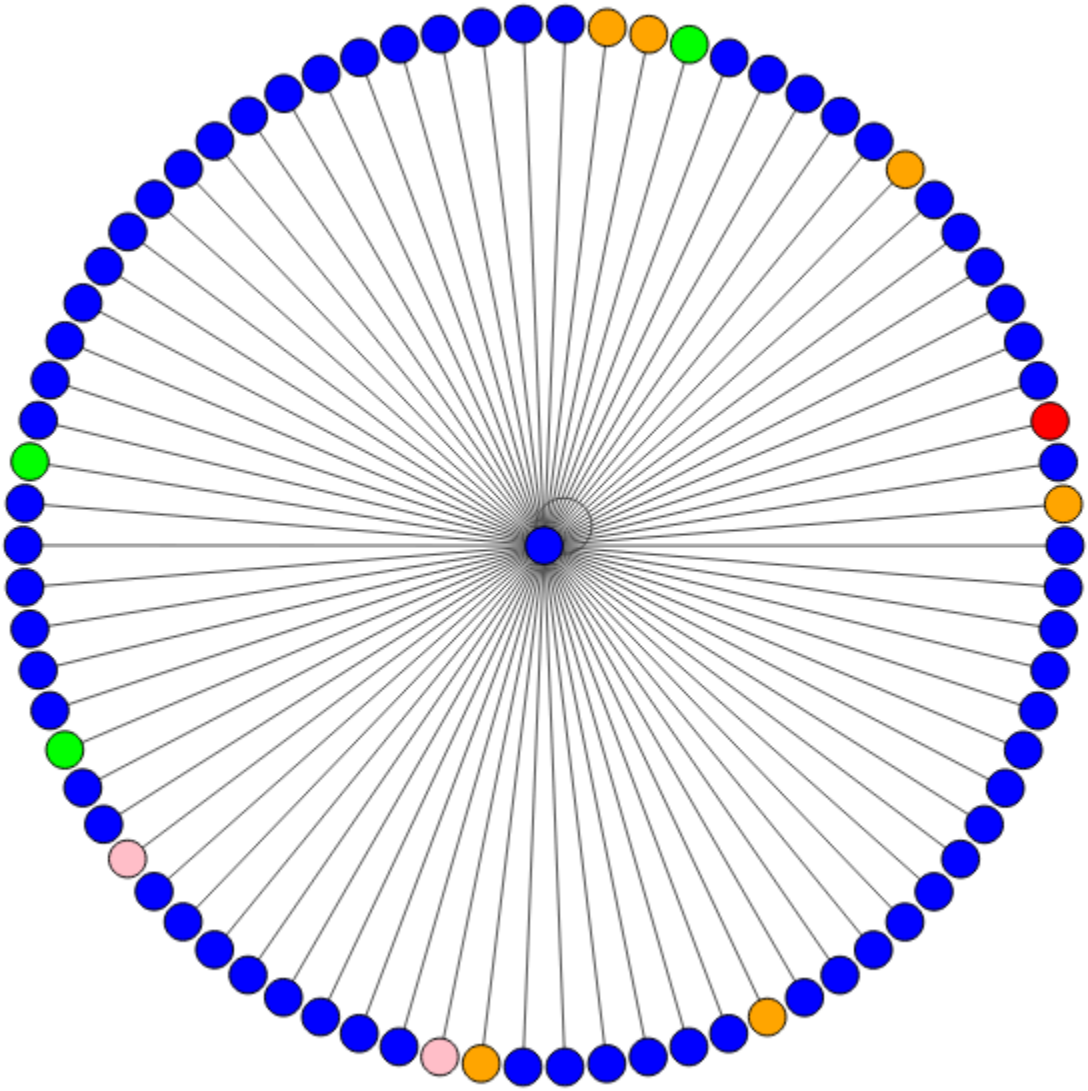
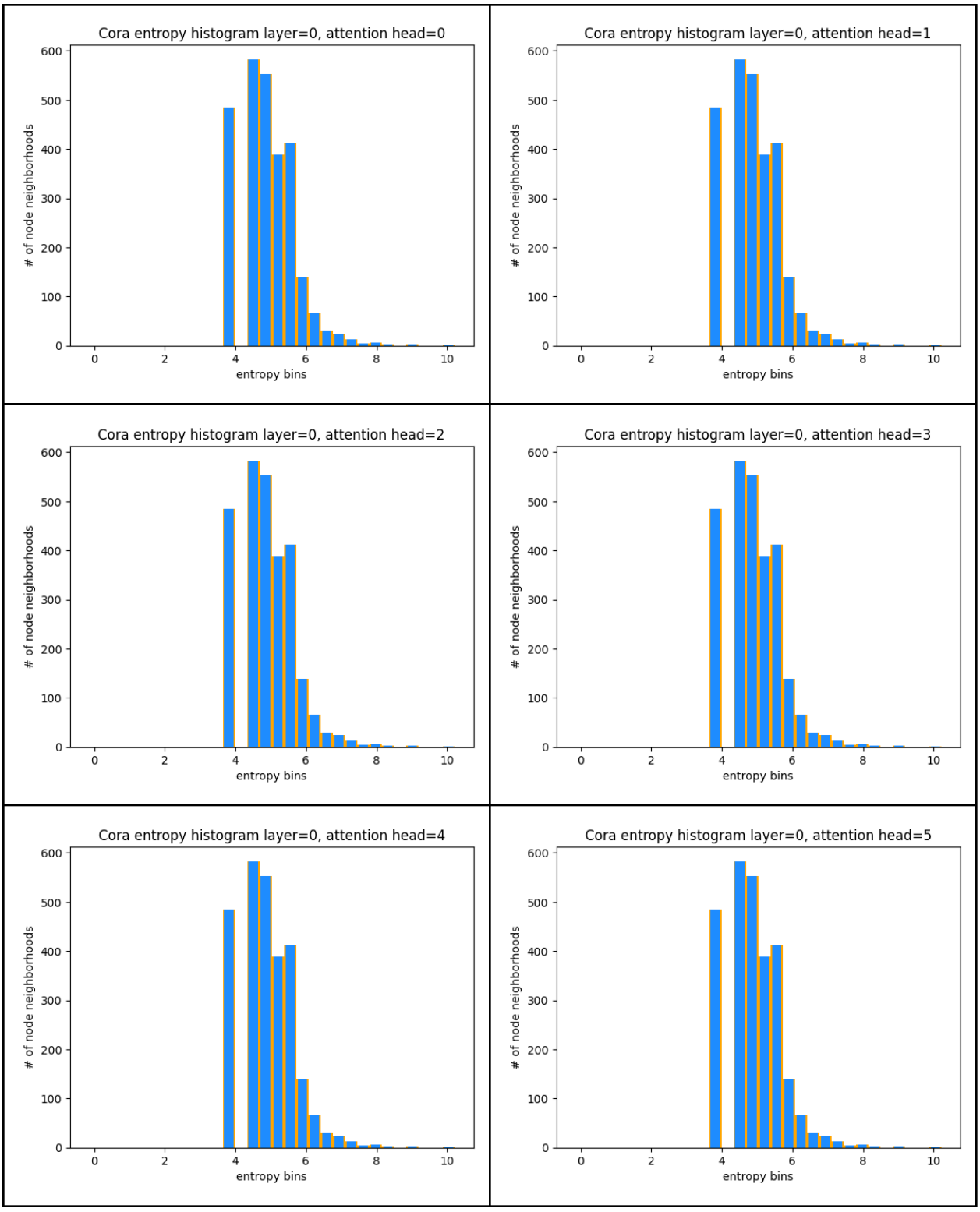


Figure-31: Visualizing neighborhood attention

Output

- Highest degree nodes = [1986 1701 306 1358]
- Max attention weight = 0.012915872037410736 and min = 0.012394174933433533


Cora entropy histogram layer=0, attention head=0
Cora entropy histogram layer=0, attention head=1
Cora entropy histogram layer=0, attention head=2
Cora entropy histogram layer=0, attention head=3
Cora entropy histogram layer=0, attention head=4
Cora entropy histogram layer=0, attention head=5
# of node neighborhoods
entropy bins
0
100
200
300
400
500
600
0
2
4
6
8
10

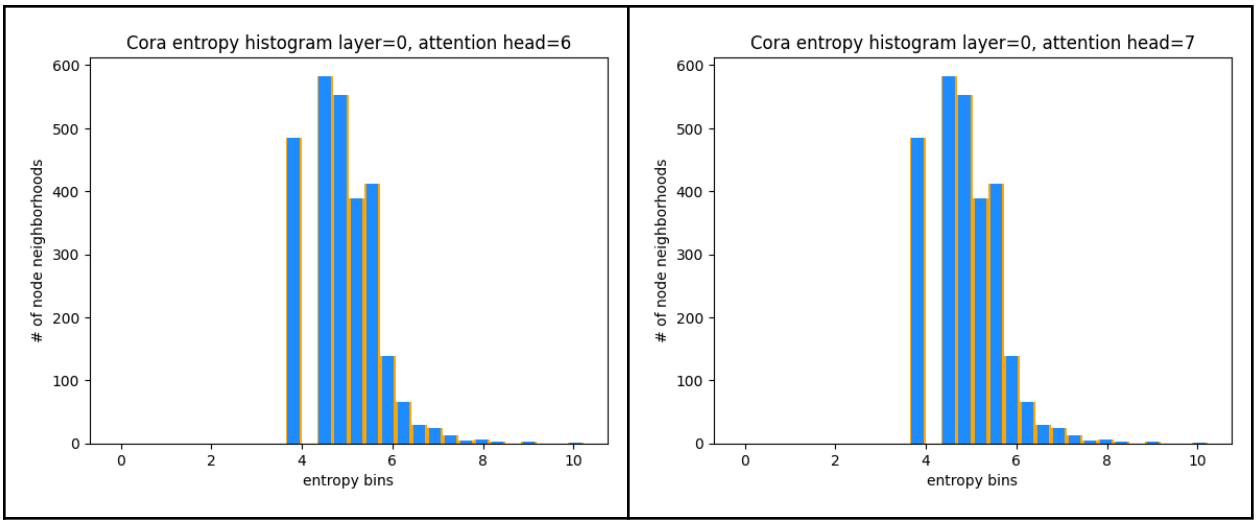


Figure-32:Visualizing entropy histograms

PPI Dataset

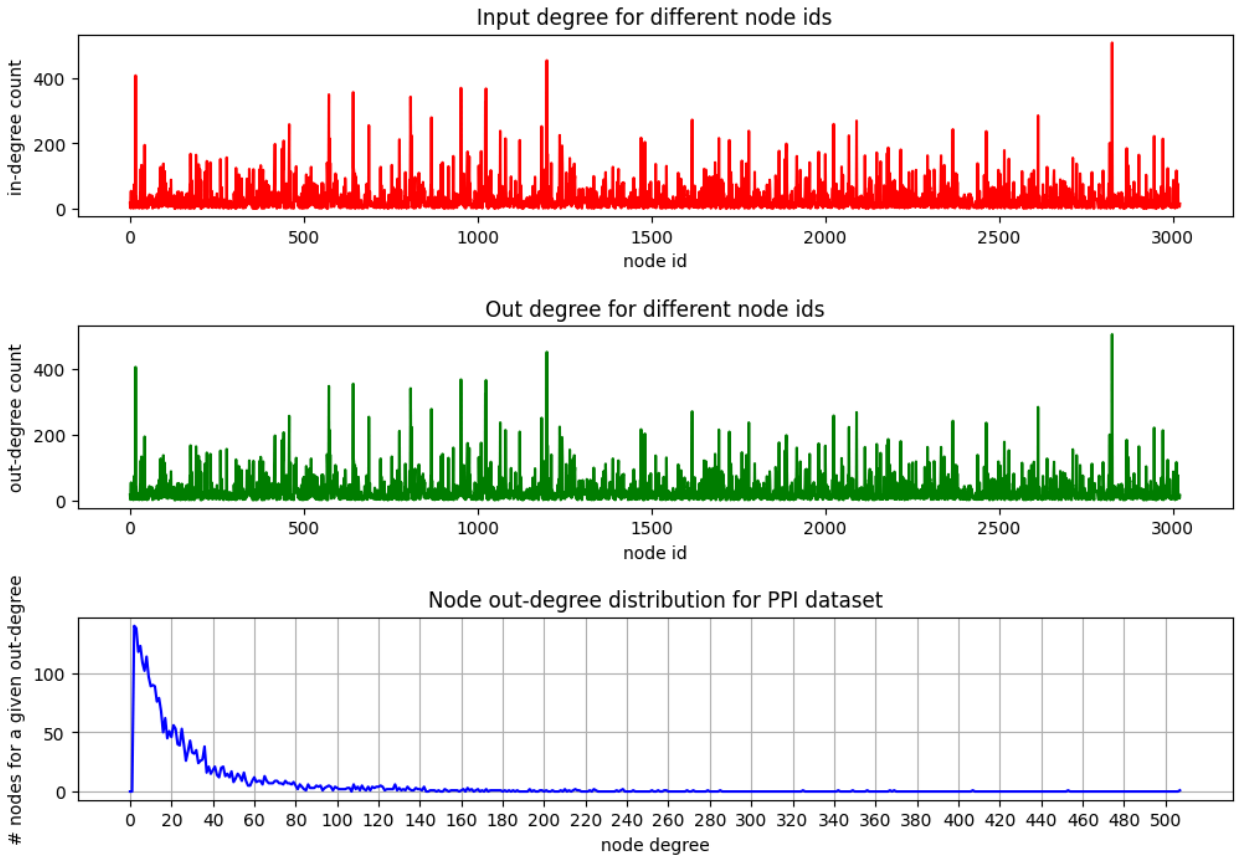


Figure-33: visualizes Cora's degree distributions.

Output

```
GAT training: time elapsed= 383.24 [s] | epoch=191 | batch=1 | train micro-F1=0.9850262881177707.
GAT validation: time elapsed= 384.86 [s] | epoch=191 | batch=1 | val micro-F1=0.9621166280557382
Test micro-F1 = 0.9762566962831557
***** Model training metadata: *****
commit_hash: 39c8f0ee634477033e8b1a6e9a6da3c7ed71bbd1
dataset_name: PPI
num_of_epochs: 200
test_perf: 0.9762566962831557
num_of_layers: 3
num_heads_per_layer: [4, 4, 6]
num_features_per_layer: [50, 64, 64, 121]
add_skip_connection: True
bias: True
dropout: 0.0
**************************************

tensor([False, False, False,  ..., False, False, False], device='cuda:0')
Max attention weight = 0.3447507619857788 and min = 0.0011031113099306822
```

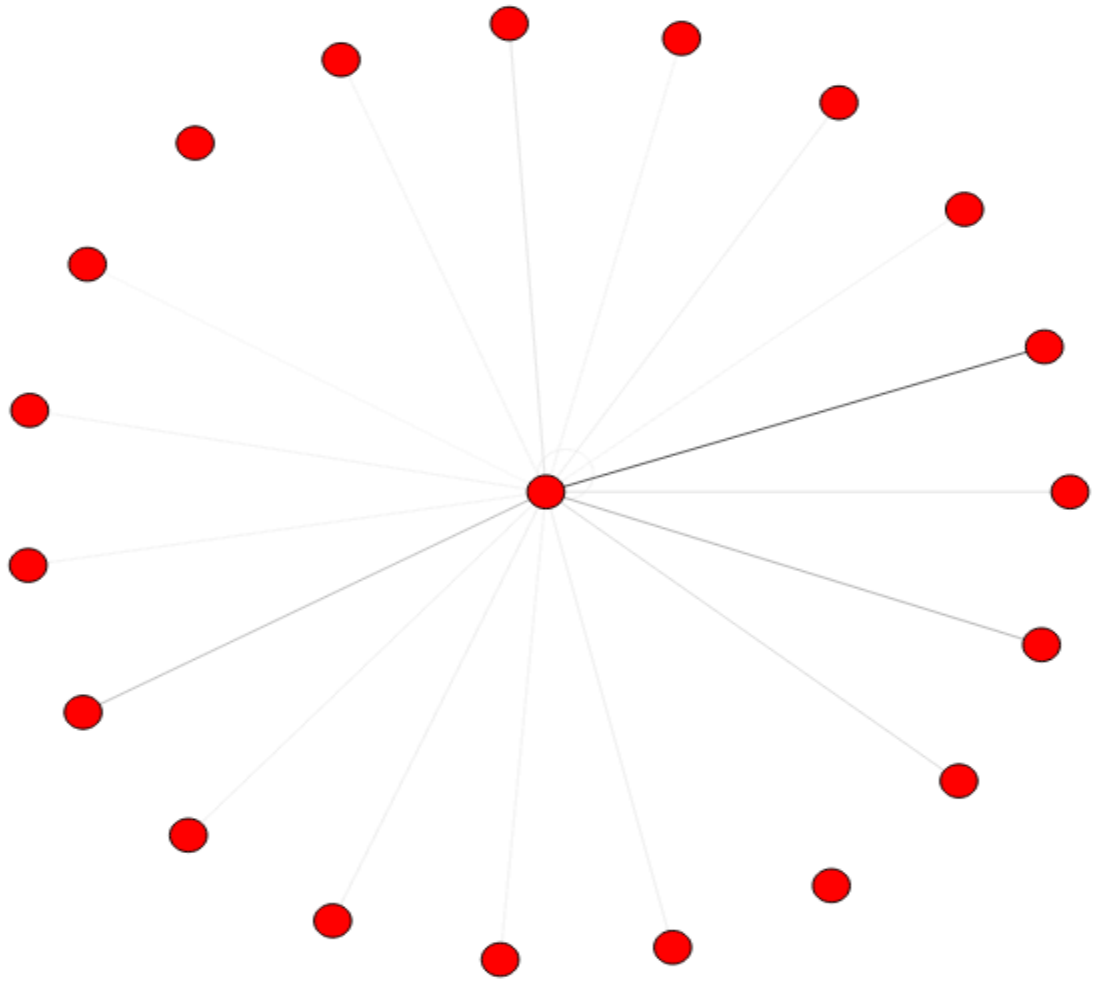


Figure-34: Visualizing neighborhood attention

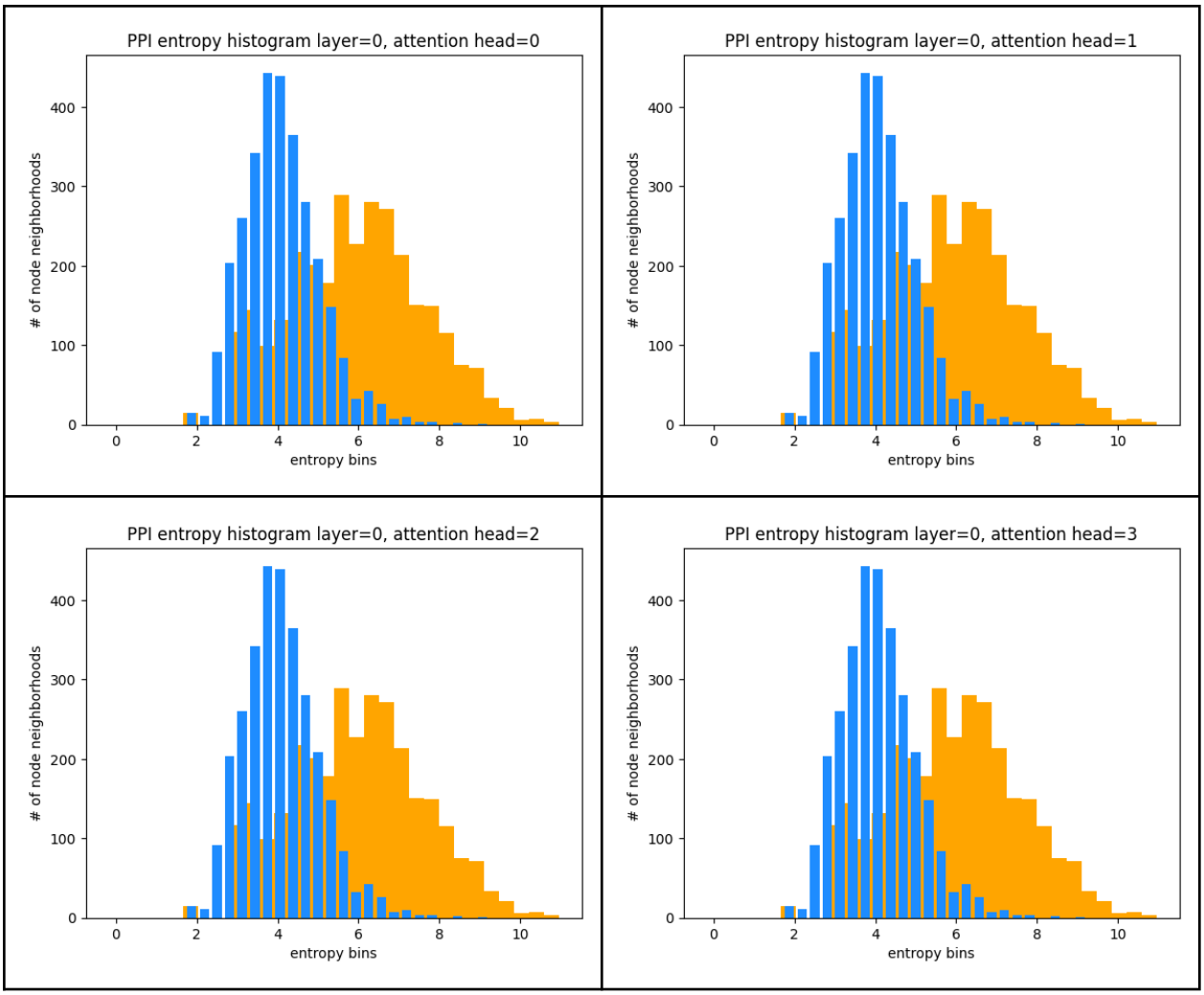


Figure-35: Visualizing entropy histograms

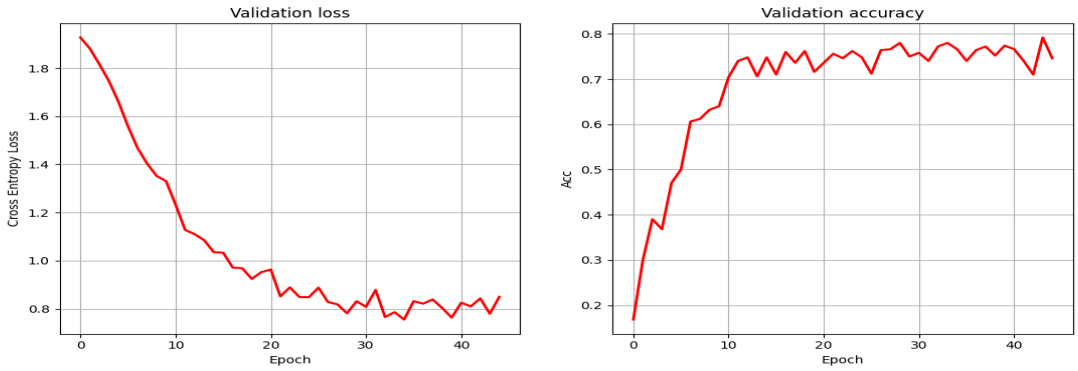


Figure-36: plots of loss and accuracy of the Validation.

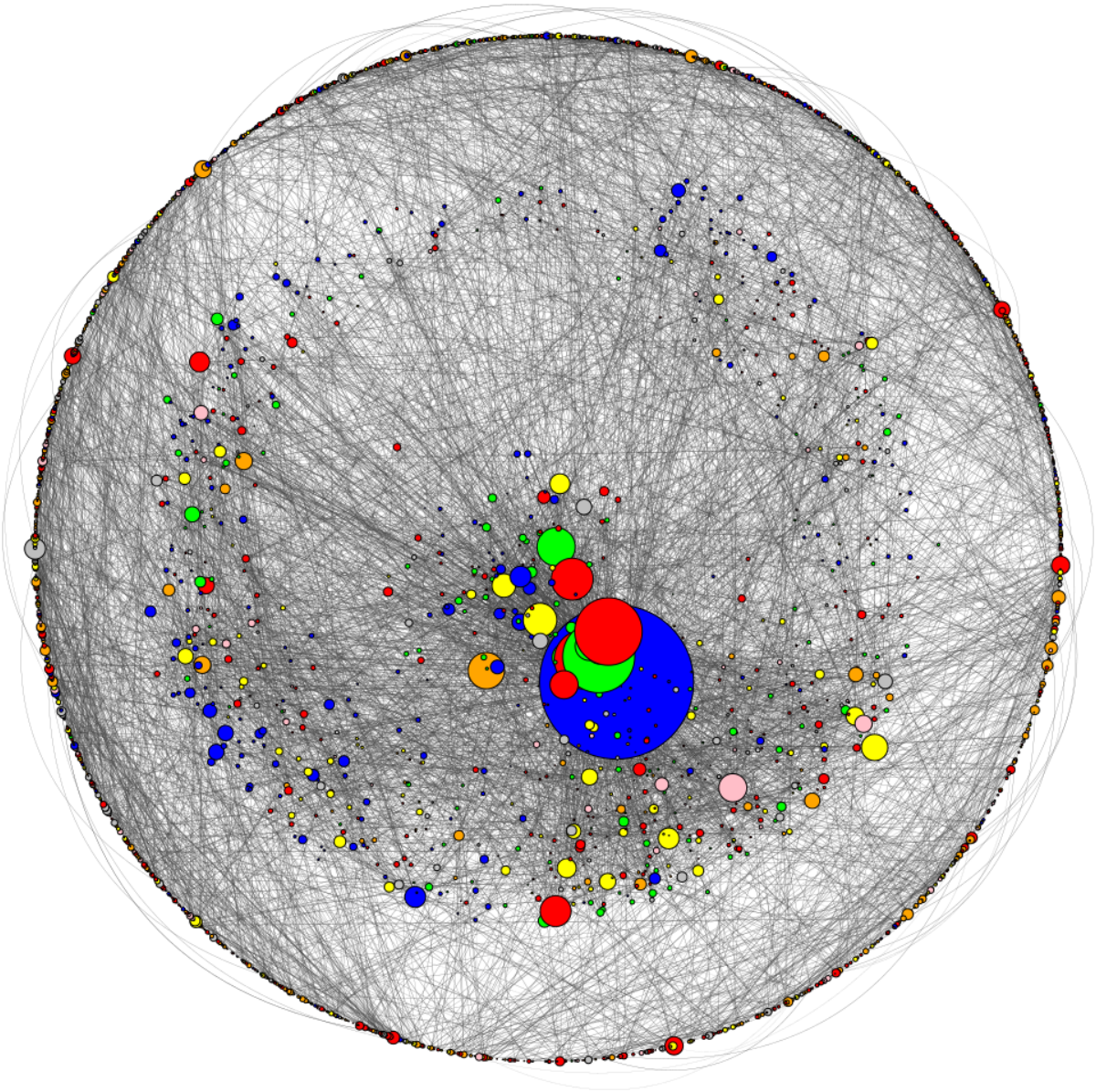


Figure-37: Cora dataset visualization

DeepWalk and node2vec

Paper:

- DeepWalk: Online Learning of Social Representation
- node2vec: Scalable Feature Learning for Networks

Code:

- node2vec doc
- node2vec code
- Example on clustering

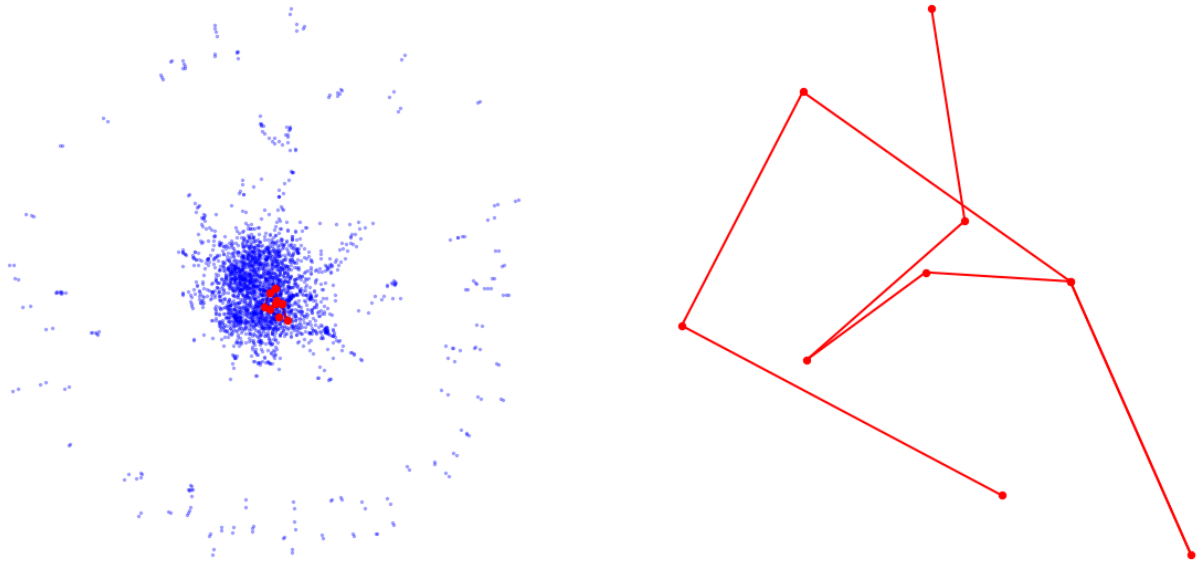


Figure-38: Node2vec representation

GraphSAGE: Cora Dataset

- Build a GNN-based link prediction model.
- Train and evaluate the model on a small DGL-provided dataset.

Output

```
NumNodes: 2708
NumEdges: 10556
NumFeats: 1433
NumClasses: 7
NumTrainingSamples: 140
NumValidationSamples: 500
NumTestSamples: 1000

In epoch 95, loss: 0.09879310429096222
AUC 0.873445789627367
```

Link_Prediction_GCNConv_GAE: Cora Dataset

Output

Epoch: 100, Loss: 0.4417, Val: 0.9026, Test: 0.9119

```
tensor([[   0,    0,    0,  ..., 2707, 2707, 2707],
        [   0,    2,    4,  ..., 2704, 2706, 2707]])
```

Link_Prediction_Cora_stellargraph_GCN

Output

- Train Set Metrics of the trained model: loss: 0.1409, acc: 0.9641
- Test Set Metrics of the trained model: loss: 0.8198, acc: 0.7620

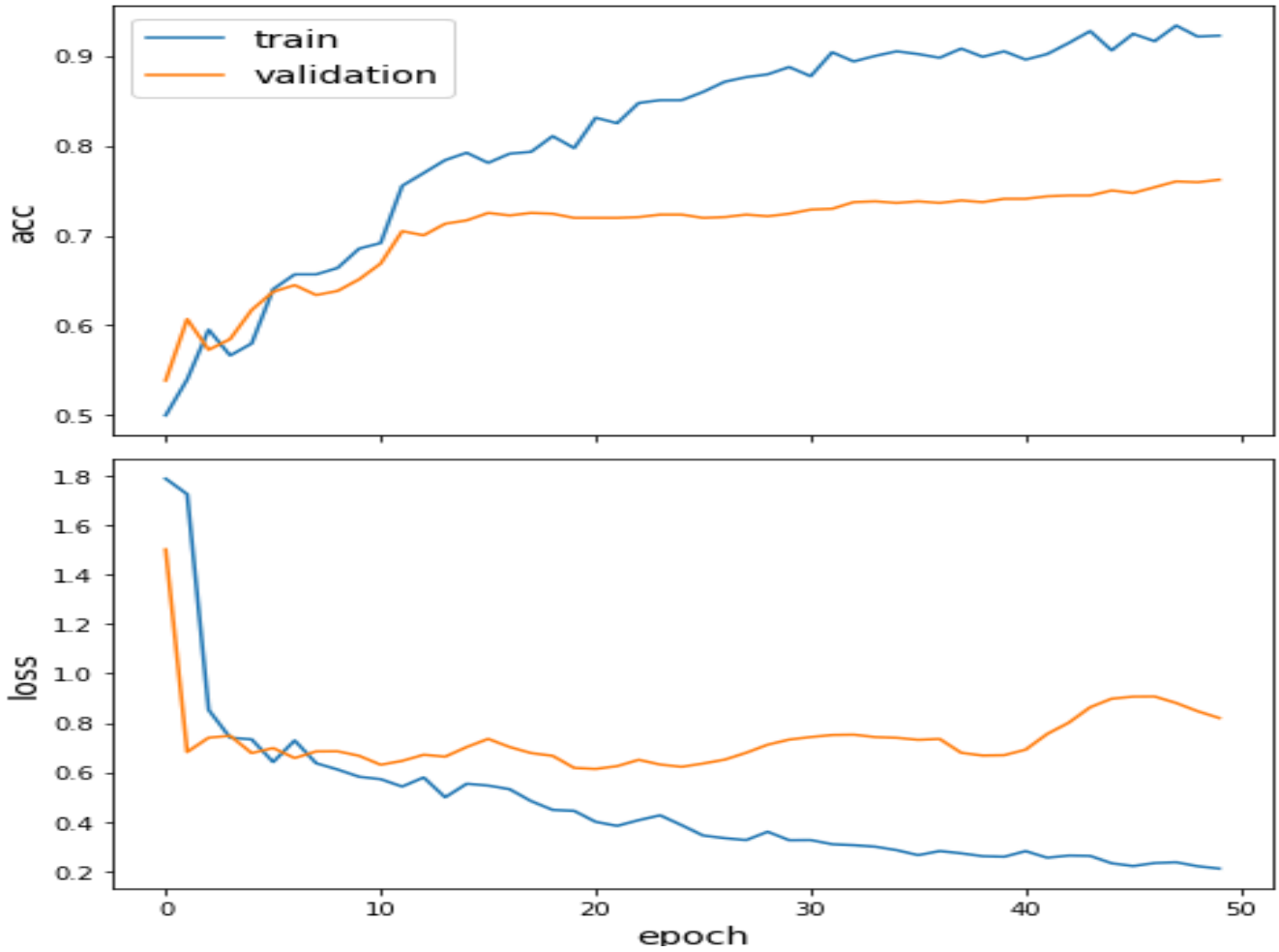


Figure-39: plots of loss and accuracy of train and validation.

Node_Classification_MLP_GCNConv_Planetoid

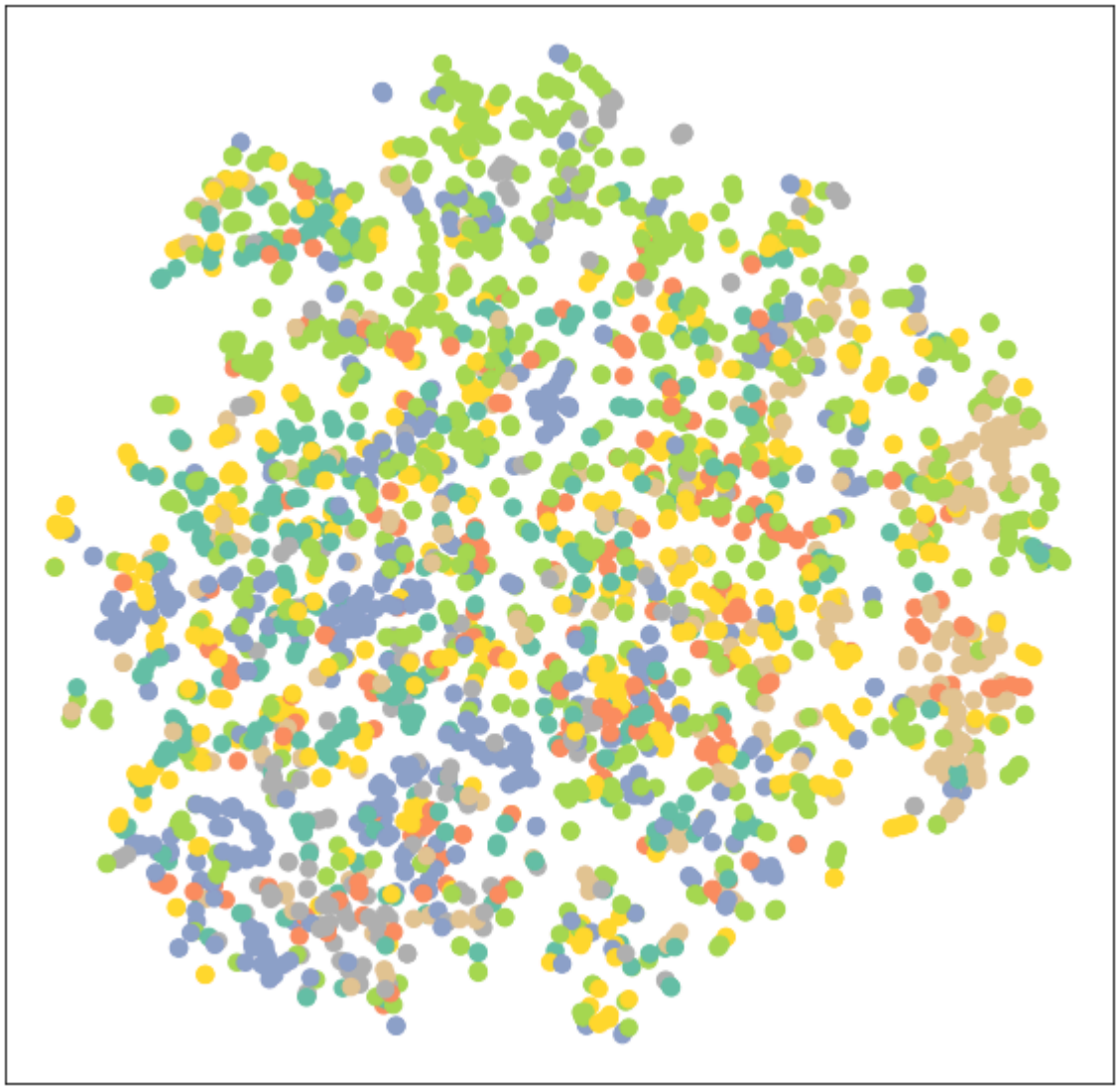


Figure-40: visualize the node embeddings of **untrained** GCN network

Output
Test Accuracy: 0.8140, Train loss: 0.30006

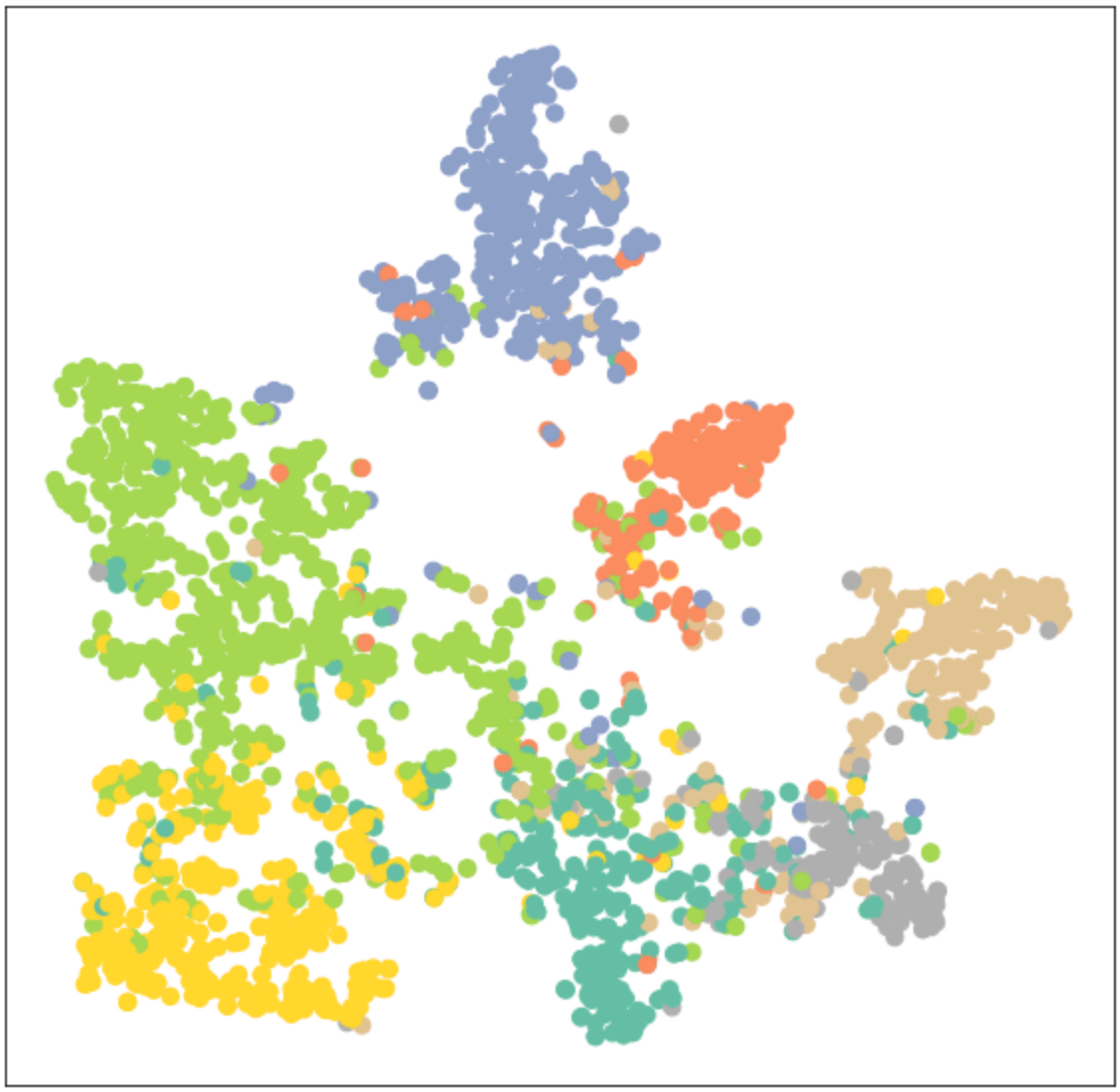


Figure-41:Output embeddings of our **trained** model

Graph_Classification_RDKit_GCNN

- Pytorch Geometric => Build Graph Neural Network
- RDKit => Handle Molecule Data

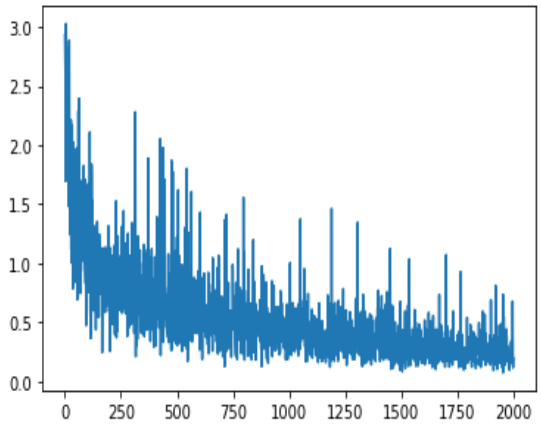


Figure-42: Visualize learning (training loss)

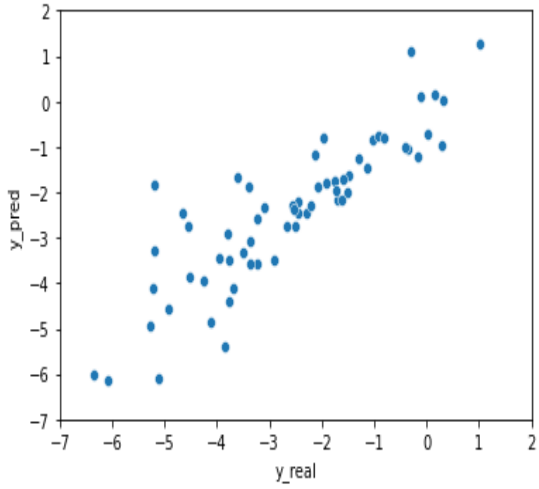


Figure-43: Scatterplot real vs. predicted

Table-1: 5 sample output related real vs. predicted

	y_real	y_pred
0	-3.240	-3.567490
1	-8.400	-7.892982
2	-0.364	-1.043183
3	-2.460	-2.185588
4	-1.710	-2.169898