Project Report

Face Detection and Recognition for Indian Faces

Akshay Nagpal

AI 0066

Contents

VGG16 Implementation

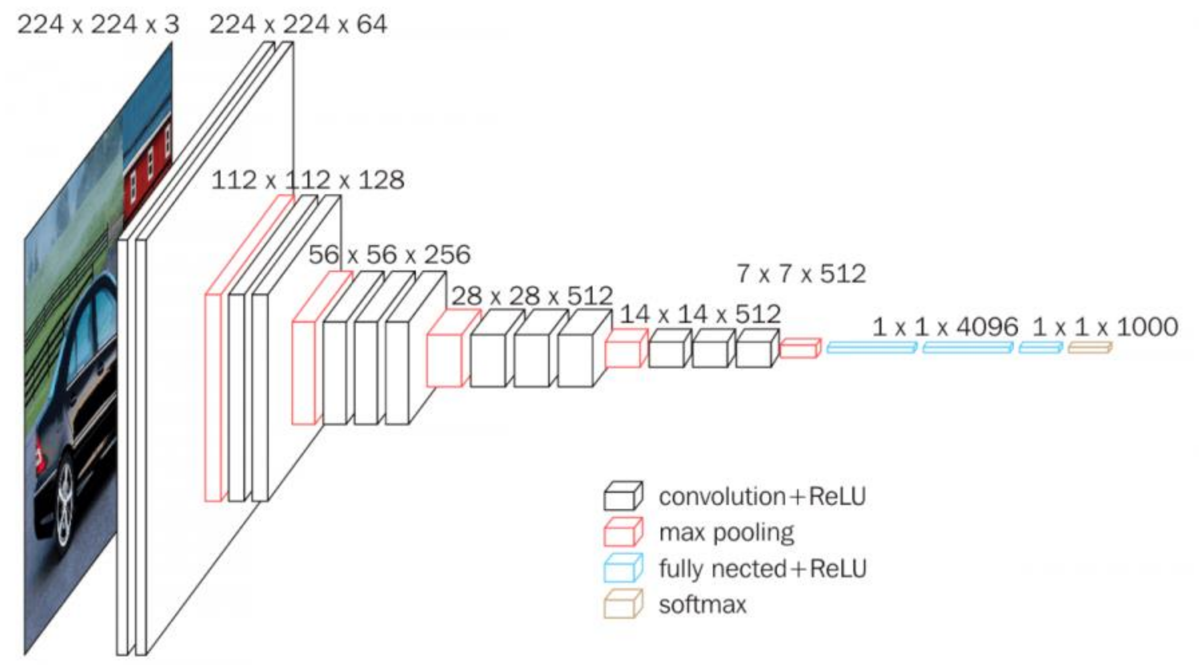
GitHub: https://github.com/aknakshay/VGG16---CIFAR10

The objective was to implement VGG16 in any Deep Learning framework and to edit the architecture to include Dropout layers, Batch Normalization other and small tweaks to perform image classification on CIFAR10 dataset.

VGG16 is a deep learning model which improves the performance by focusing on increasing the depth of the network by stacking many convolution filters of small sizes together.

In VGG16, filters of receptive field 3 × 3 are used, comparatively smaller than the filters used by the previous models. Instead of using a 7 × 7 filter, three 3 × 3 filters are used which decrease the trainable parameters and effectively covers the same receptive field. With a convolution stride of 1 px, the padding used is 'same'. Max Pooling is performed over 2 × 2 px with a stride of 2.

There are 16 weight layers and 5 max pool layers in the original architecture. Input image is of 224 × 224 dimensions and there are 138 million trainable parameters. We can visualize the same in the below given image:



The same was implemented in Keras with tensorflow backend. BatchNormalization and Dropout layers are added after each MaxPooling operation. Dense layers are changed to have 4096,512 and 10 neurons respectively.

ADAM is used as an optimizer with initial learning rate 1e-3. Categorical Crossentropy is used as the loss function and accuracy as a metric.

A learning rate scheduler is added which decreases the learning rate by a factor of 10 if the validation accuracy has not increased compared to it's n-3rd run and if learning rate has not been decreased in the last 3 epochs.

Accuracy: ~ 85% in 21 epochs

MNIST Challenge

GitHub: https://github.com/aknakshay/MNIST-Challenge/

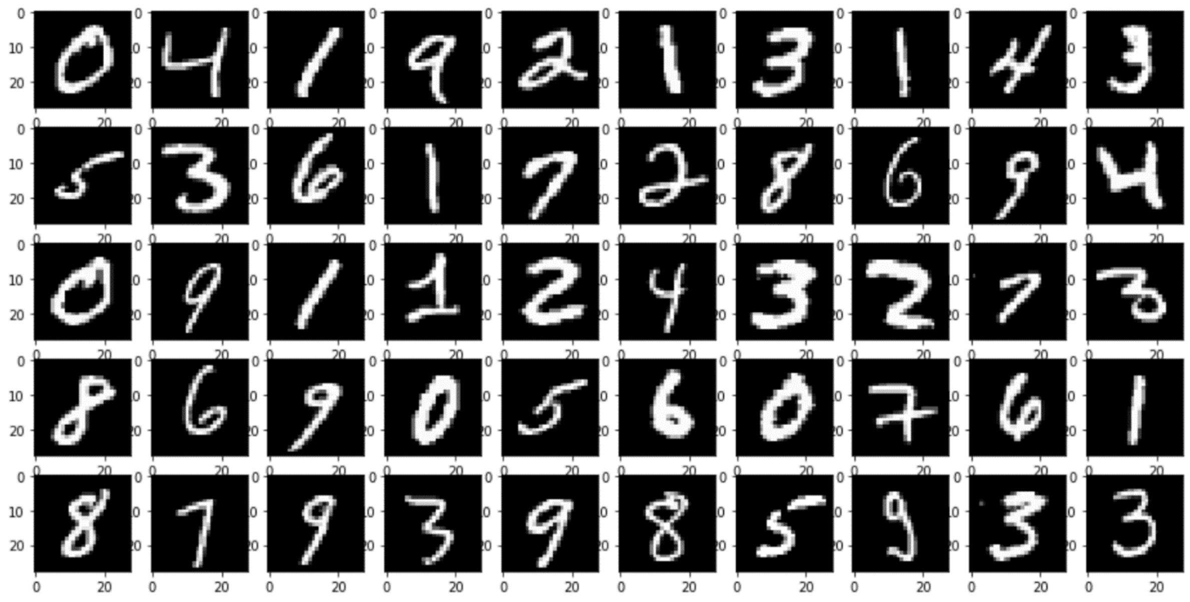
The objective was to implement a model with less than 10000 parameters on MNIST data and get an accuracy of more than 99% in 10 epochs.

The solutions were implemented in Keras with tensorflow backend.

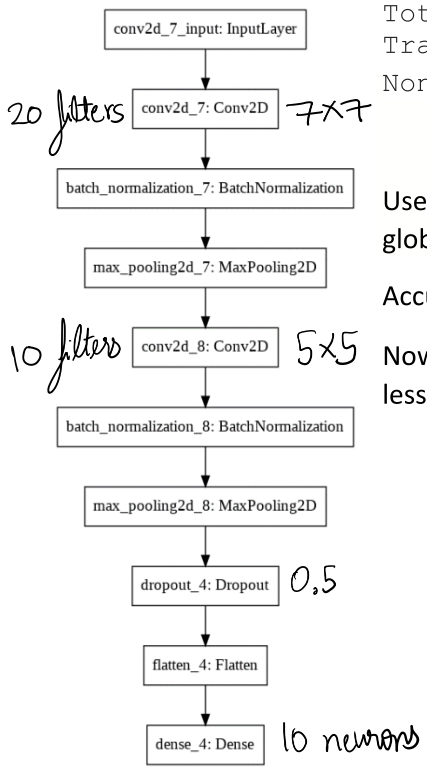
ADAM is used as an optimizer with initial learning rate 1e-3. Categorical Cross entropy is used as the loss function and accuracy as a metric.

A learning rate scheduler is available which decreases the learning rate by a factor which can be customized if the validation accuracy has not increased compared to its n-3rd run and if learning rate has not been decreased in the last 3 epochs. However, here it's not being used as the factor is kept to be 1.

Data:



Solution 1: Convolutions Model


conv2d_7_input: InputLayer
20 filters
conv2d_7: Conv2D
7×7
batch_normalization_7: BatchNormalization
max_pooling2d_7: MaxPooling2D
10 filters
conv2d_8: Conv2D
5×5
batch_normalization_8: BatchNormalization
max_pooling2d_8: MaxPooling2D
dropout_4: Dropout
0,5
flatten_4: Flatten
dense_4: Dense
10 neurons

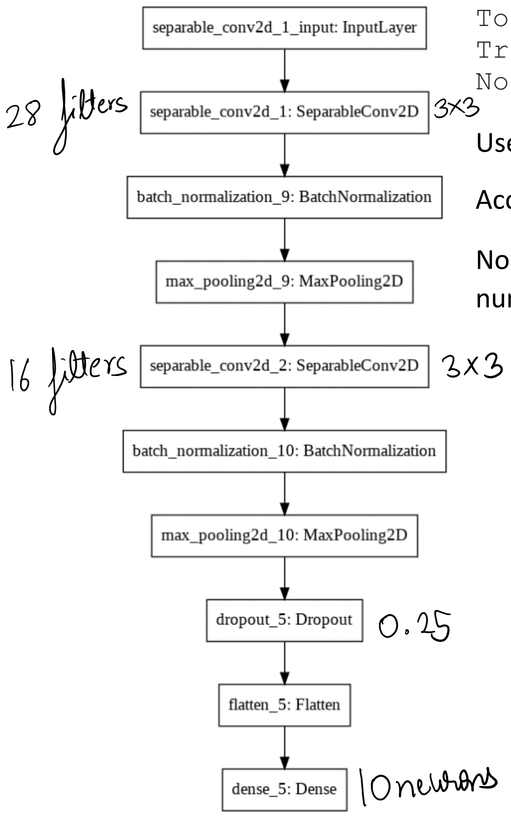
```
Total params: 8,640
Trainable params: 8,580
Non-trainable params: 60
```

Used bigger kernels of size 7 × 7 and 5 × 5 as information is distributed globally in the image and not locally.

Accuracy on Test Set: 99.2%

Now that the objective was achieved, tried reducing the parameters to less than 5k parameters.

Solution 2: Separable Convolution Model


separable_conv2d_1_input: InputLayer
28 filters
separable_conv2d_1: SeparableConv2D
3×3
batch_normalization_9: BatchNormalization
max_pooling2d_9: MaxPooling2D
16 filters
separable_conv2d_2: SeparableConv2D
3×3
batch_normalization_10: BatchNormalization
max_pooling2d_10: MaxPooling2D
dropout_5: Dropout
0.25
flatten_5: Flatten
dense_5: Dense
10 neurons

```
Total params: 4,967
Trainable params: 4,879
Non-trainable params: 88
```

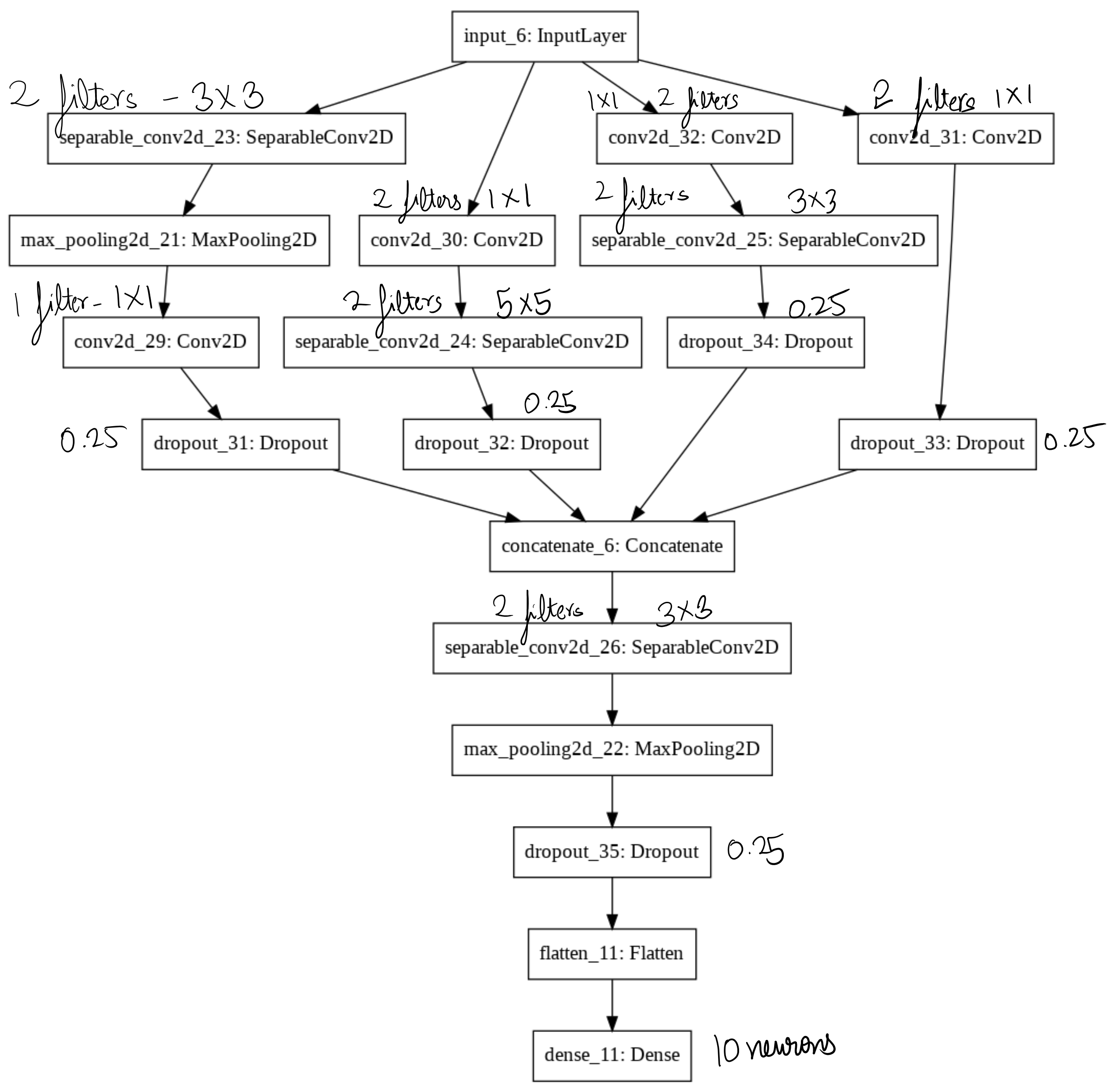
Used smaller kernels of size 3 × 3 but with many channels.

Accuracy on Test Set: 98.42%

Now that the objective was achieved, tried further by reducing the number of trainable parameters even further.

Solution 3: Inception Model

```
Total params: 3,077
Trainable params: 3,077
Non-trainable params: 0
```



Made use of the inception module in the beginning. Used filters of various sizes – 1 × 1, 3 × 3, 5 × 5.
Accuracy on Test Set: 97.36%

Observations

With an efficient architecture, smaller and more efficient models can be built with lesser parameters to achieve comparable performance.

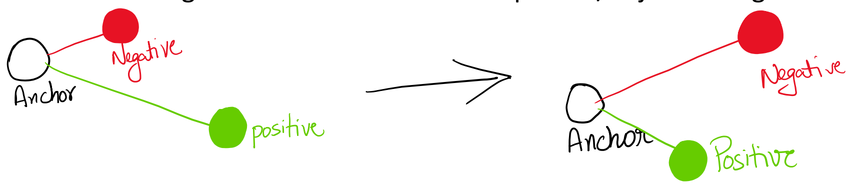
- In the initial solution, model had **8640 parameters** with an accuracy of **99.20%**
- In the second solution, model had **4967 parameters** with an accuracy of **98.42%**. The number of parameters were reduced by 42.5% with only 0.78% decrease in accuracy compared to the first solution.
- In the third solution, model had **3077 parameters** with an accuracy of **97.36 %**. The number of parameters were reduced by 64.38% with only 1.84% decrease in accuracy compared to the first solution.

FaceNet: Face Detection and Recognition

GitHub: https://github.com/aknakshay/Face-Recognition-From-Scratch

FaceNet is a deep learning model which learns mappings from face images to a compact Euclidian space and the distance between two embeddings correspond to the measure of similarity between faces. The embeddings can be used for face verification, clustering faces, etc.

The model uses **triplet loss** as the loss function.

> For image embeddings, triplet loss is great way to create embeddings. In triplet loss, we take an image as an anchor, another image which is of the same person/object as positive example and another image which is not of the same person/object as negative example.
>
> 
>
>
> We try to create embeddings in a way that the distance between the embeddings of the anchor and positive image should be lower and the distance between the embeddings of the anchor and the negative image should be higher.

Triplets are generated online by selecting hard positive/negative examples from within a minibatch. The triplet images are transformed into 128 Dimensional embeddings and the distance between embeddings of positive and anchor is decreased and distance between embeddings of negative and anchor image is increased.

Steps

1. Load Image Dataset
2. Pre-process Dataset
    a. Detect Faces in the image using MTCNN and crop the image accordingly
    b. Affine transformations are applied to straighten the face
    c. Normalize the image
    d. Resize the image to the input size of 224 × 224
3. Training (with Online Triplet Selection)
   For minibatch in data:
    a. Train the network forward and store the 128-D embedding
    b. Calculate the distance between each of the image in the mini batch
    c. From an anchor, find semi hard and hard positives and negatives amongst the minibatch for each identity with alpha (margin) set to 2
    d. Using argmax of positive and argmin of negative, get the triplet to be used to calculate loss
    e. Back Propagate the feedback w.r.t loss
4. Validation: Classification using SVM over embeddings
    a. Pre-Process the validation/test set
    b. Get 128-D embeddings using the facenet model
    c. Divide the dataset into train and test
    d. Train an SVM model using train set
    e. Test the model using test set

Implementation:

Face Aligner Class: Aligns the faces by applying affine transformations

Face Aligner's implementation is taken from PyImageSearch [1]. It has been modified to be used with MTCNN Face Detector.

Loading Image Dataset: Indian Movie Faces Dataset

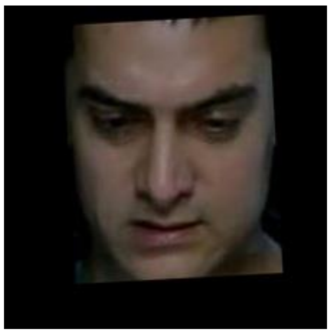
In the dataset – Indian Movies Faces Dataset [2], there are 100 identities, with a total of 34513 images. For an actor, pictures from various movies with different looks over the period are provided. Quantile analysis is done over the dataset to see the approximate image dimensions.

Quantile	Dimensions
0	9 X 9
1	34 X 34
2	45 X 45
3	57 X 57
4	71 X 71
5	87 X 87
6	108 X 108
7	133 X 133
8	168 X 168
9	232 X 232
10	872 X 872

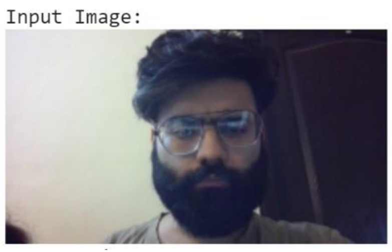
Function to process data and load the locations of files in a list. If transformed data is present in given directory, it just adds the location in the files

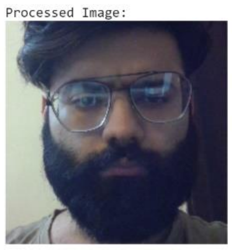
I have used MTCNN face detector and face alignment affine transformation to process the dataset. With this, 23424 images got filtered out as MTCNN could not detect a face in them. The target dimension is also kept 224 X 224, which is the input size that our model accepts.

Input:

Processed Image: 

Using personal picture:





After the initial pre-processing and filtration, there are only 11,089 images left.

Loading the location and label of image in a data frame

The location and label of the images is stored in a data frame by splitting the location.

Checking the dimension of images and doing quantile analysis on it

Shape function returns the number of pixels in an image. It multiplies the height and the width of each image. The same is stored in the dataset corresponding to each image.

Quantile Analysis can be done on the image dataset if pre-processing isn't done. Since pre-processing is done, all the quantiles would show 224 X 224 as the dimension.

Encoding Labels from strings to integers

Labels are encoded into integers from strings.

Removing images having lesser pixels

It is only required if dataset pre-processing is not done. Since I have already done that offline, it is not required.

Separating out last n identities

All the identities are stored in a list. Data from last 10 identities is stored in the validation set and the remaining are put in the training set.

Filtering out identities having less than 10 images

Identities which have less than 10 images are filtered out from the training dataset.

```
Total Images for Training Before Filtration: 9859
Total Images for Training After Filtration: 9850
```

Image Pre-processing

ImageDataGenerator of Keras is used for image pre-processing. The images are normalised using z-score normalisation. The training images are also flipped horizontally randomly. Validation images, however, only goes through z-score normalisation process.

Face Detection and Alignment can also be done on the fly but that is avoided here since the same operations are done in each epoch and leads to redundancy. Moreover, it consumes a lot of memory and per epoch training time increases to 5x. Hence, it's done offline in the very start. However, it can be enabled by uncommenting MTCNN and FaceAligner initialiser and pre-processing function parameter in the ImageDataGenerator initialiser in the code.

FaceNet paper suggests that each identity should be covered in each of the mini batch. And since I am using Triplet loss, 2 images of each identity are required to have a valid triplet of that identity. Because of the memory constraints, I could only have a batch size of 100.

As an experiment, I did arrange the dataset to cover maximum identities in a batch, i.e. 50. But the loss does not converge after it reaches the margin since the dataset is not shuffled per epoch and the model tries to train with the same 100 images set per minibatch.

The above-mentioned can be tried with offline triplet mining but with online triplet mining, the best way to go ahead is to shuffle the data each epoch which ImageDataGenerator helps us do.

9850 images are used for training and 1230 images are used for validation/test.

Training

Triplet Loss Function

Triplet loss implementation is taken from O. Moindrot [3].

I have used hard batching to calculate loss. The implementation has a bug [4] in which the loss gets stuck at margin. The same is resolved by a workaround suggested in the same bug [5].

Batch All can also be used. For the same, a wrapper function is created as the normal one is also returning a metric. But I haven’t tried training with this.

Also, I did try triplet semi-hard loss provided on TensorFlow 1.x [6] but it also has an issue with integration with keras. Though, it has not been used by me for final training, the same issue is fixed at the end of the notebook.

Lambda Layer - L2 Normalisation

L2 normalisation layer is defined using a lambda layer. The input is squared, AveragePooling2D layer is used for average pooling and then square root is done on the tensor.

Model Architecture - NN2 Model

NN2 Model architecture is implemented with 11 inception layers. A total of 7.5 Million parameters are present in the same. Model architecture is attached on next page. However, the implementation of inception layers using MaxPool might not be the same as NN2.

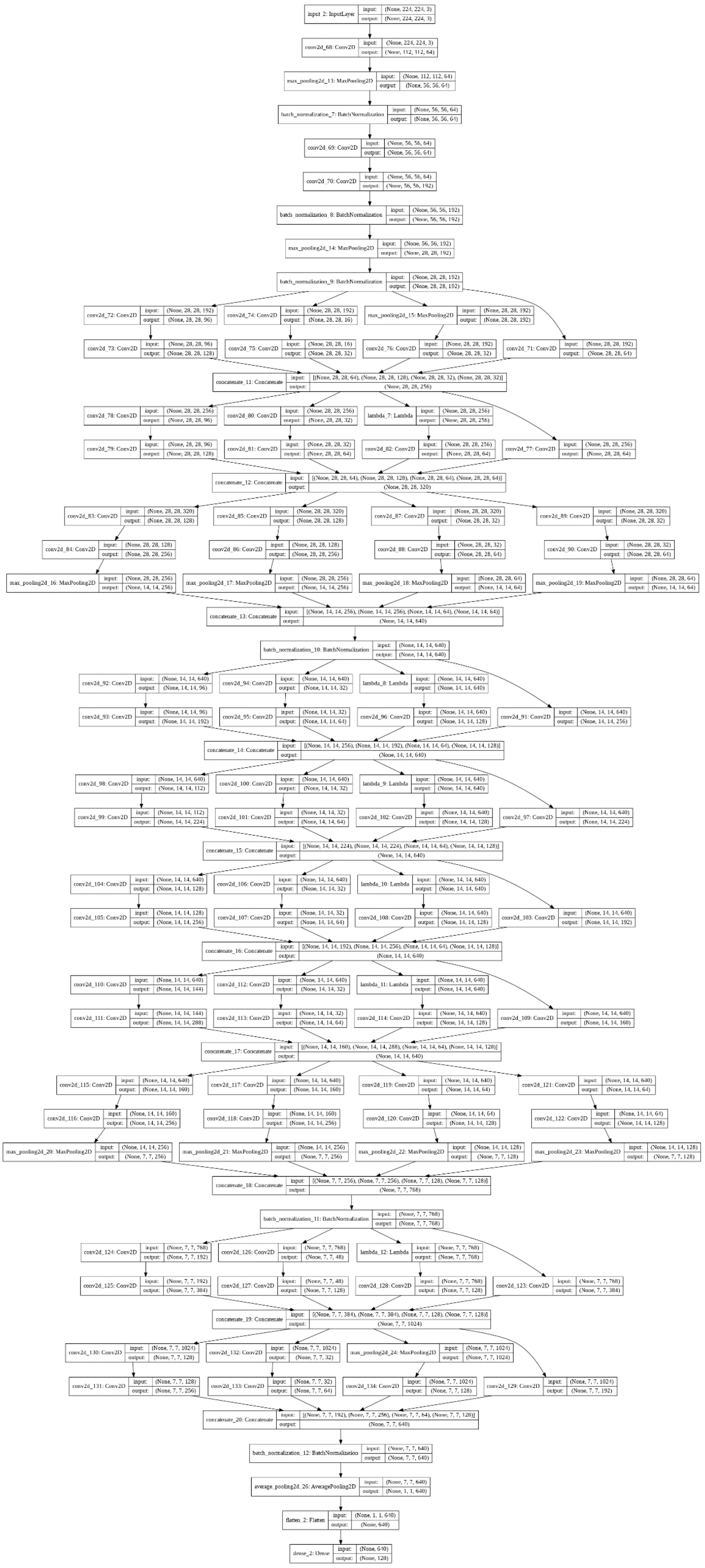
Moreover, while training, Gradient Explosion was seen. Hence, a few Batch Normalization layers are added to resolve the same.

```
Total params: 7,589,792
Trainable params: 7,584,800
Non-trainable params: 4,992
```

Metric: Fraction of Positive Triplets

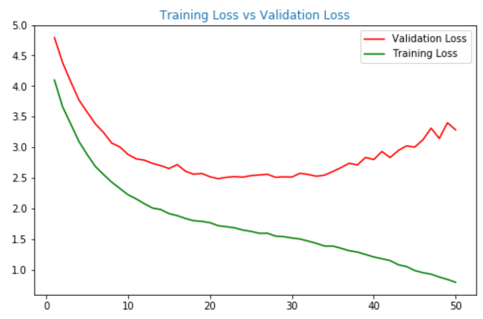
The metric is derived from batch all triplet loss implementation. It calculates the fraction of positive triplets to those of total valid triplets.

Ideally, it should decrease over epochs. It is one indicator that can be used to see if training is going right.


input_2: InputLayer input: (None, 224, 224, 3) output: (None, 224, 224, 3)
conv2d_68: Conv2D input: (None, 224, 224, 3) output: (None, 112, 112, 64)
max_pooling2d_13: MaxPooling2D input: (None, 112, 112, 64) output: (None, 56, 56, 64)
batch_normalization_7: BatchNormalization input: (None, 56, 56, 64) output: (None, 56, 56, 64)
conv2d_69: Conv2D input: (None, 56, 56, 64) output: (None, 56, 56, 64)
conv2d_70: Conv2D input: (None, 56, 56, 64) output: (None, 56, 56, 192)
batch_normalization_8: BatchNormalization input: (None, 56, 56, 192) output: (None, 56, 56, 192)
max_pooling2d_14: MaxPooling2D input: (None, 56, 56, 192) output: (None, 28, 28, 192)
batch_normalization_9: BatchNormalization input: (None, 28, 28, 192) output: (None, 28, 28, 192)
conv2d_72: Conv2D input: (None, 28, 28, 192) output: (None, 28, 28, 96)
conv2d_74: Conv2D input: (None, 28, 28, 192) output: (None, 28, 28, 16)
max_pooling2d_15: MaxPooling2D input: (None, 28, 28, 192) output: (None, 28, 28, 192)
conv2d_73: Conv2D input: (None, 28, 28, 96) output: (None, 28, 28, 128)
conv2d_75: Conv2D input: (None, 28, 28, 16) output: (None, 28, 28, 32)
conv2d_76: Conv2D input: (None, 28, 28, 192) output: (None, 28, 28, 32)
conv2d_71: Conv2D input: (None, 28, 28, 192) output: (None, 28, 28, 64)
concatenate_11: Concatenate input: [(None, 28, 28, 64), (None, 28, 28, 128), (None, 28, 28, 32), (None, 28, 28, 32)] output: (None, 28, 28, 256)
conv2d_78: Conv2D input: (None, 28, 28, 256) output: (None, 28, 28, 96)
conv2d_80: Conv2D input: (None, 28, 28, 256) output: (None, 28, 28, 32)
lambda_7: Lambda input: (None, 28, 28, 256) output: (None, 28, 28, 256)
conv2d_79: Conv2D input: (None, 28, 28, 96) output: (None, 28, 28, 128)
conv2d_81: Conv2D input: (None, 28, 28, 32) output: (None, 28, 28, 64)
conv2d_82: Conv2D input: (None, 28, 28, 256) output: (None, 28, 28, 64)
conv2d_77: Conv2D input: (None, 28, 28, 256) output: (None, 28, 28, 64)
concatenate_12: Concatenate input: [(None, 28, 28, 64), (None, 28, 28, 128), (None, 28, 28, 64), (None, 28, 28, 64)] output: (None, 28, 28, 320)
conv2d_83: Conv2D input: (None, 28, 28, 320) output: (None, 28, 28, 128)
conv2d_85: Conv2D input: (None, 28, 28, 320) output: (None, 28, 28, 128)
conv2d_87: Conv2D input: (None, 28, 28, 320) output: (None, 28, 28, 32)
conv2d_89: Conv2D input: (None, 28, 28, 320) output: (None, 28, 28, 32)
conv2d_84: Conv2D input: (None, 28, 28, 128) output: (None, 28, 28, 256)
conv2d_86: Conv2D input: (None, 28, 28, 128) output: (None, 28, 28, 256)
conv2d_88: Conv2D input: (None, 28, 28, 32) output: (None, 28, 28, 64)
conv2d_90: Conv2D input: (None, 28, 28, 32) output: (None, 28, 28, 64)
max_pooling2d_16: MaxPooling2D input: (None, 28, 28, 256) output: (None, 14, 14, 256)
max_pooling2d_17: MaxPooling2D input: (None, 28, 28, 256) output: (None, 14, 14, 256)
max_pooling2d_18: MaxPooling2D input: (None, 28, 28, 64) output: (None, 14, 14, 64)
max_pooling2d_19: MaxPooling2D input: (None, 28, 28, 64) output: (None, 14, 14, 64)
concatenate_13: Concatenate input: [(None, 14, 14, 256), (None, 14, 14, 256), (None, 14, 14, 64), (None, 14, 14, 64)] output: (None, 14, 14, 640)
batch_normalization_10: BatchNormalization input: (None, 14, 14, 640) output: (None, 14, 14, 640)
conv2d_92: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 96)
conv2d_94: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 32)
lambda_8: Lambda input: (None, 14, 14, 640) output: (None, 14, 14, 640)
conv2d_93: Conv2D input: (None, 14, 14, 96) output: (None, 14, 14, 192)
conv2d_95: Conv2D input: (None, 14, 14, 32) output: (None, 14, 14, 64)
conv2d_96: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 128)
conv2d_91: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 256)
concatenate_14: Concatenate input: [(None, 14, 14, 256), (None, 14, 14, 192), (None, 14, 14, 64), (None, 14, 14, 128)] output: (None, 14, 14, 640)
conv2d_98: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 112)
conv2d_100: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 32)
lambda_9: Lambda input: (None, 14, 14, 640) output: (None, 14, 14, 640)
conv2d_99: Conv2D input: (None, 14, 14, 112) output: (None, 14, 14, 224)
conv2d_101: Conv2D input: (None, 14, 14, 32) output: (None, 14, 14, 64)
conv2d_102: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 128)
conv2d_97: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 224)
concatenate_15: Concatenate input: [(None, 14, 14, 224), (None, 14, 14, 224), (None, 14, 14, 64), (None, 14, 14, 128)] output: (None, 14, 14, 640)
conv2d_104: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 128)
conv2d_106: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 32)
lambda_10: Lambda input: (None, 14, 14, 640) output: (None, 14, 14, 640)
conv2d_105: Conv2D input: (None, 14, 14, 128) output: (None, 14, 14, 256)
conv2d_107: Conv2D input: (None, 14, 14, 32) output: (None, 14, 14, 64)
conv2d_108: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 128)
conv2d_103: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 192)
concatenate_16: Concatenate input: [(None, 14, 14, 192), (None, 14, 14, 256), (None, 14, 14, 64), (None, 14, 14, 128)] output: (None, 14, 14, 640)
conv2d_110: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 144)
conv2d_112: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 32)
lambda_11: Lambda input: (None, 14, 14, 640) output: (None, 14, 14, 640)
conv2d_111: Conv2D input: (None, 14, 14, 144) output: (None, 14, 14, 288)
conv2d_113: Conv2D input: (None, 14, 14, 32) output: (None, 14, 14, 64)
conv2d_114: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 128)
conv2d_109: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 160)
concatenate_17: Concatenate input: [(None, 14, 14, 160), (None, 14, 14, 288), (None, 14, 14, 64), (None, 14, 14, 128)] output: (None, 14, 14, 640)
conv2d_115: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 160)
conv2d_117: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 160)
conv2d_119: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 64)
conv2d_121: Conv2D input: (None, 14, 14, 640) output: (None, 14, 14, 64)
conv2d_116: Conv2D input: (None, 14, 14, 160) output: (None, 14, 14, 256)
conv2d_118: Conv2D input: (None, 14, 14, 160) output: (None, 14, 14, 256)
conv2d_120: Conv2D input: (None, 14, 14, 64) output: (None, 14, 14, 128)
conv2d_122: Conv2D input: (None, 14, 14, 64) output: (None, 14, 14, 128)
max_pooling2d_20: MaxPooling2D input: (None, 14, 14, 256) output: (None, 7, 7, 256)
max_pooling2d_21: MaxPooling2D input: (None, 14, 14, 256) output: (None, 7, 7, 256)
max_pooling2d_22: MaxPooling2D input: (None, 14, 14, 128) output: (None, 7, 7, 128)
max_pooling2d_23: MaxPooling2D input: (None, 14, 14, 128) output: (None, 7, 7, 128)
concatenate_18: Concatenate input: [(None, 7, 7, 256), (None, 7, 7, 256), (None, 7, 7, 128), (None, 7, 7, 128)] output: (None, 7, 7, 768)
batch_normalization_11: BatchNormalization input: (None, 7, 7, 768) output: (None, 7, 7, 768)
conv2d_124: Conv2D input: (None, 7, 7, 768) output: (None, 7, 7, 192)
conv2d_126: Conv2D input: (None, 7, 7, 768) output: (None, 7, 7, 48)
lambda_12: Lambda input: (None, 7, 7, 768) output: (None, 7, 7, 768)
conv2d_125: Conv2D input: (None, 7, 7, 192) output: (None, 7, 7, 384)
conv2d_127: Conv2D input: (None, 7, 7, 48) output: (None, 7, 7, 128)
conv2d_128: Conv2D input: (None, 7, 7, 768) output: (None, 7, 7, 128)
conv2d_123: Conv2D input: (None, 7, 7, 768) output: (None, 7, 7, 384)
concatenate_19: Concatenate input: [(None, 7, 7, 384), (None, 7, 7, 384), (None, 7, 7, 128), (None, 7, 7, 128)] output: (None, 7, 7, 1024)
conv2d_130: Conv2D input: (None, 7, 7, 1024) output: (None, 7, 7, 128)
conv2d_132: Conv2D input: (None, 7, 7, 1024) output: (None, 7, 7, 32)
max_pooling2d_24: MaxPooling2D input: (None, 7, 7, 1024) output: (None, 7, 7, 1024)
conv2d_131: Conv2D input: (None, 7, 7, 128) output: (None, 7, 7, 256)
conv2d_133: Conv2D input: (None, 7, 7, 32) output: (None, 7, 7, 64)
conv2d_134: Conv2D input: (None, 7, 7, 1024) output: (None, 7, 7, 128)
conv2d_129: Conv2D input: (None, 7, 7, 1024) output: (None, 7, 7, 192)
concatenate_20: Concatenate input: [(None, 7, 7, 192), (None, 7, 7, 256), (None, 7, 7, 64), (None, 7, 7, 128)] output: (None, 7, 7, 640)
batch_normalization_12: BatchNormalization input: (None, 7, 7, 640) output: (None, 7, 7, 640)
average_pooling2d_26: AveragePooling2D input: (None, 7, 7, 640) output: (None, 1, 1, 640)
flatten_2: Flatten input: (None, 1, 1, 640) output: (None, 640)
dense_2: Dense input: (None, 640) output: (None, 128)

Adam is used as an optimizer with learning rate of 1e-4. Lower learning rate is important for loss to converge in this model.

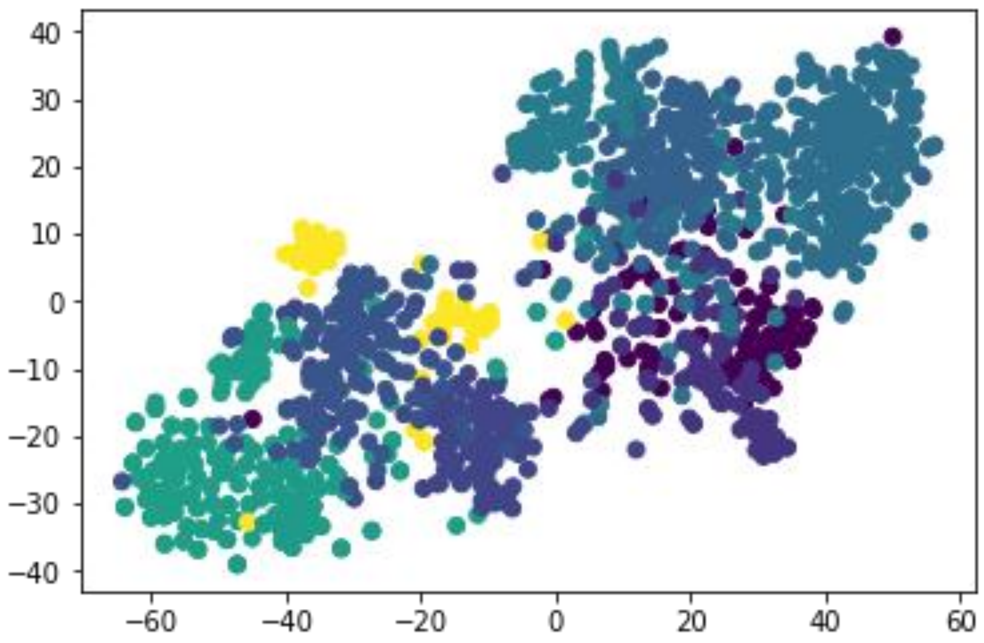
Model is trained over 50 epochs. It starts overfitting around 25th epoch. Due to the time constraint, I could not retrain it with lesser epochs.



Since custom functions are used, while loading the model, objects of the same needs to be provided as done in the notebook.

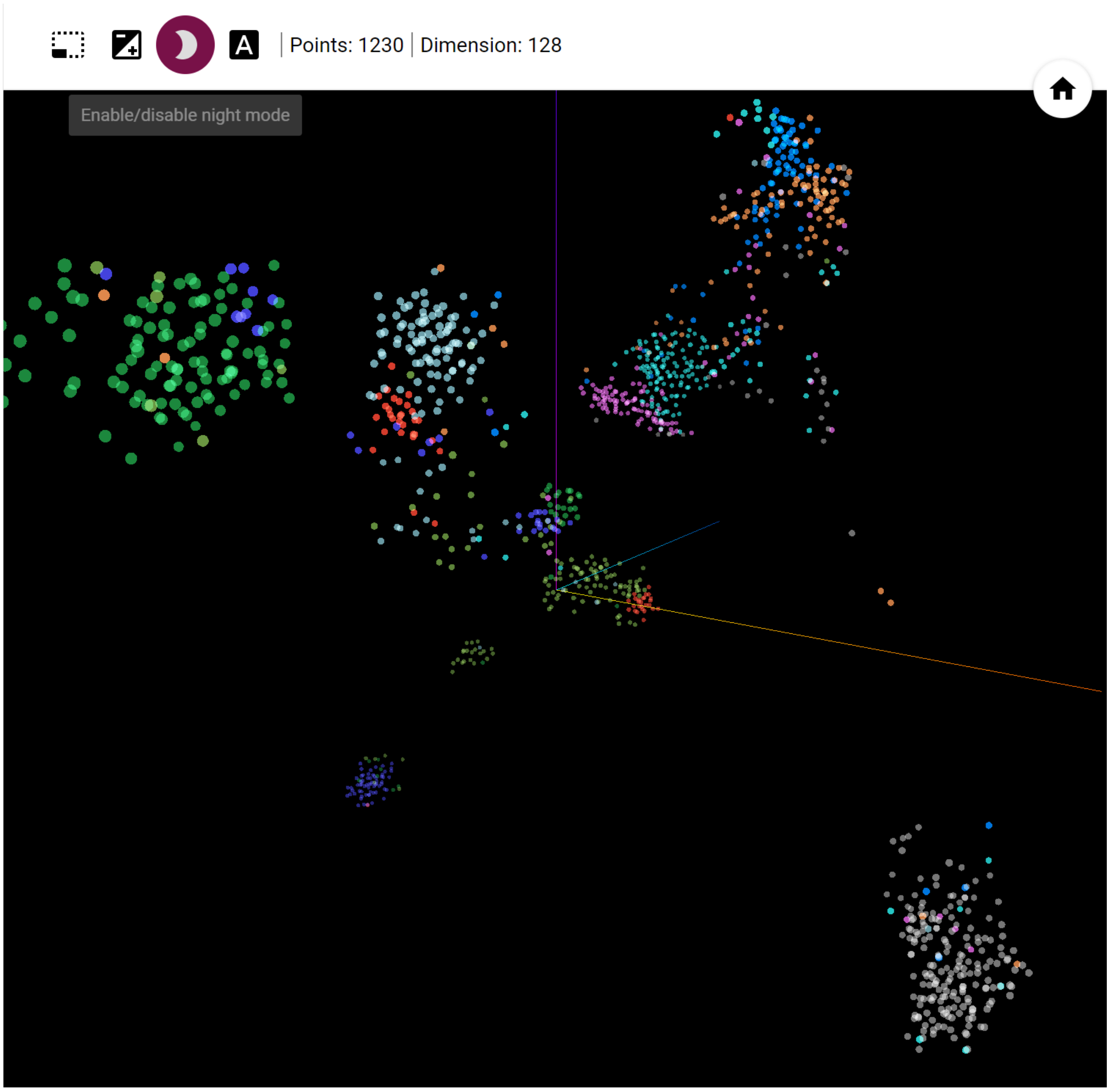
Embeddings

Embeddings of the validation set are visualised using T-SNE. As we can see, clusters are formed for each class of images. The margin between clusters is very less at boundaries and at someplaces, it's overlapping. The same can be solved by increasing the number of images for training.



The embeddings can be visualised on TensorFlow Projector using the link: https://projector.tensorflow.org/?config=https://raw.githubusercontent.com/aknakshay/Face-Recognition-From-Scratch/master/tensor_projector_config.json

I am attaching a screenshot of the same.



T-SNE with Perplexity=46, Learning Rate=10 and 904 iterations

Face Recognition: Classification using SVM over embeddings

ImageDataGenerator of Keras is used for image pre-processing. The images are normalised using z-score normalisation. A total of 1230 images are there as the test set.

Using the predict_generator method of keras, I am getting the embeddings for each of those 1230 images.

I am splitting the dataset into train and test using train_test_split method of sklearn in a ratio of 80:20.

SVM without any hyperparameter tuning is trained using train data. On the test data, I am getting an accuracy of **82.52%.**

Classification Report:

```
    Classes     precision    recall  f1-score   support

          2          0.56      0.56      0.56        18
         16          0.73      0.79      0.76        14
         21          0.79      1.00      0.88        19
         26          0.86      0.97      0.91        32
         30          0.73      0.91      0.81        33
         35          0.93      0.93      0.93        43
         40          0.71      0.43      0.54        23
         52          0.86      0.67      0.75        18
         53          0.97      0.94      0.96        34
         93          1.00      0.67      0.80        12

    accuracy                            0.83       246
   macro avg         0.81      0.79      0.79       246
weighted avg         0.83      0.83      0.82       246
```

References

[1] A. Rosebrock, "Face Alignment with OpenCV and Python," [Online]. Available: https://www.pyimagesearch.com/2017/05/22/face-alignment-with-opencv-and-python/.

[2] M. H. P. B. J. G. M. K. R. V. V. H. J. C. K. R. R. R. V. K. a. C. V. J. Shankar Setty, "Indian Movie Face Database: A Benchmark for Face Recognition Under Wide Variations," in *National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG)*, 2013.

[3] O. Moindrot, "Triplet Loss Implementation," [Online]. Available: https://github.com/omoindrot/tensorflow-triplet-loss/blob/master/model/triplet_loss.py.

[4] I. #18, "problem about loss value," [Online]. Available: https://github.com/omoindrot/tensorflow-triplet-loss/issues/18.

[5] ChristieLin, "Workaround for triplet loss implementation Issue #18," [Online]. Available: https://github.com/omoindrot/tensorflow-triplet-loss/issues/18#issuecomment-455031523.

[6] TensorFlow, "TensorFlow Core r1.14 API," [Online]. Available: https://www.tensorflow.org/versions/r1.14/api_docs/python/tf/contrib/losses/metric_learning/triplet_semihard_loss.