REPORT

Members: - Shubham Sontakke (BT19CSE107)

Topic: - Face Recognition using GoogLeNet

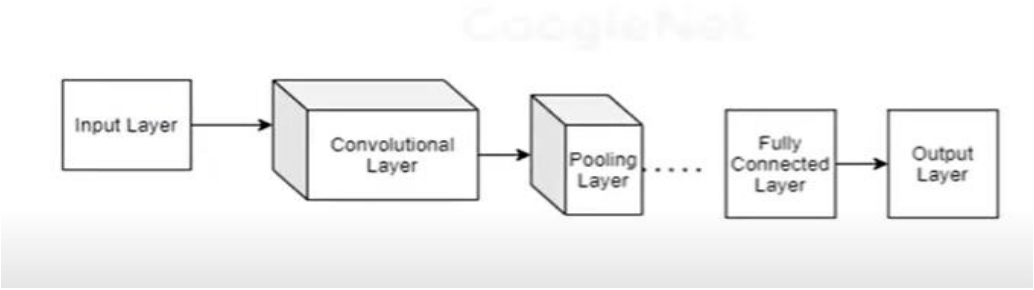
GoogLeNet:

In this assignment we have implemented Face Recognition using GoogLeNet CNN. It has been pre-trained with thousands of images and can classify thousands of Objects. It can classify images into multiple categories with very high accuracy.

GoogLeNet is a convolutional neural network that is 22 layers deep. You can load a pretrained version of the network trained on either the ImageNet or Places365 data sets. The network trained on ImageNet classifies images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. The network trained on Places365 is similar to the network trained on ImageNet, but classifies images into 365 different place categories, such as field, park, runway, and lobby. These networks have learned different feature representations for a wide range of images. The pretrained networks both have an image input size of 224-by-224.

Today GoogLeNet is used for other computer vision tasks such as face detection, adversarial training etc.

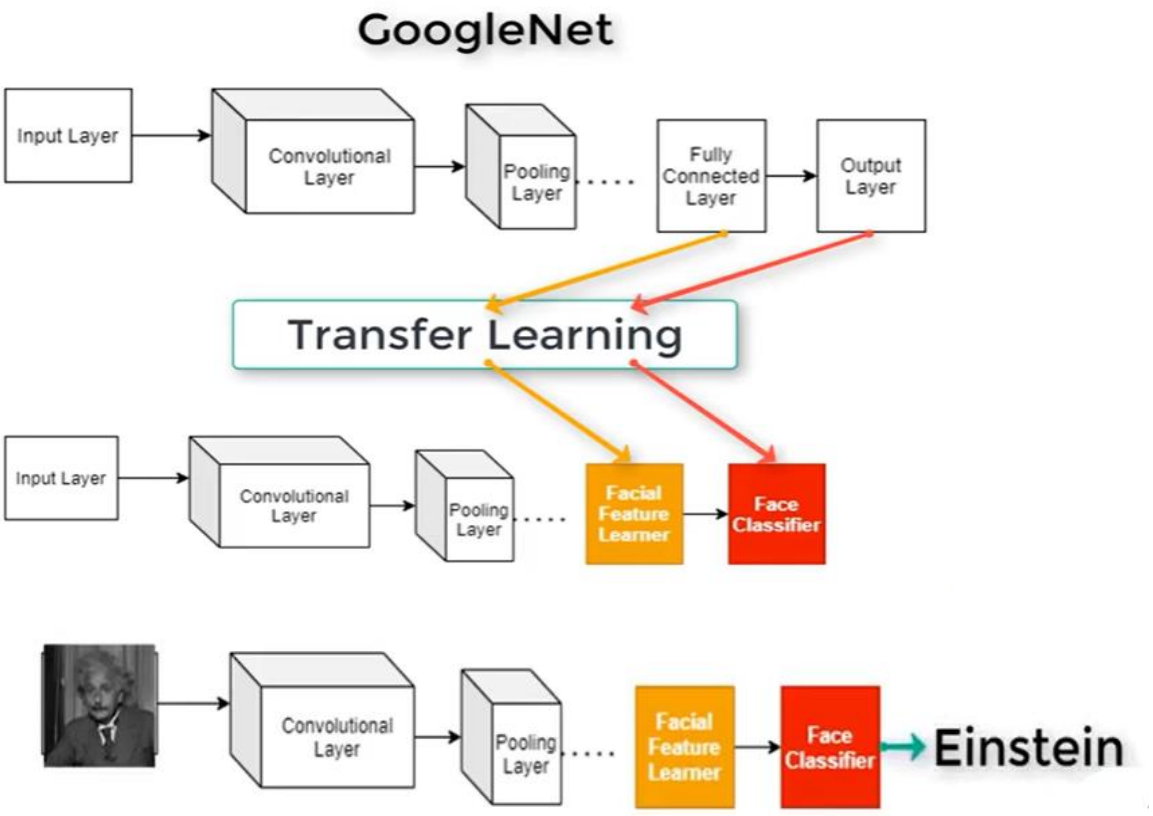
Basic Layer diagram of GoogLeNet:



You can retrain a GoogLeNet network to perform a new task using transfer learning. The transfer learning is a method in deep learning used to use an existing network as a starting point to learn new task. When performing transfer learning, the most common approach is to use networks pretrained on the ImageNet data set. If the new task is similar to classifying scenes, then using the network trained on Places-365 can give higher accuracies.

How does our facial recognition work?

We are appling concept of Transfer Learning to replace the last layers of GoogLeNet with our self-defined layers, so that we can train it to recognize faces.



GooLeNet has multiple layers. Main two are– feature learning layer and task specific layers/ Output layer. The feature learning layers learn the low-level features such as colors, blobs and edges. On the other hand, the task specific layers learn the task specific features. For example – if the task of the network is to classify vehicle images from non-vehicle images, then the task specific layers will learn the features of the vehicles. Normally these task specific layers are the last layers of a network.

In transfer learning, we are going to remove this layer from the existing network and new layers are added so that they can be trained to learn new features for some new tasks. Then these new layers are trained with new dataset, validated and tested. If the network receives training in a proper way with effective dataset, then the network become capable to classify the newly learned objects.

Our model process is divided into 6 steps. These steps are:

1. **Load dataset**
2. **Load the pre-trained network,**
3. **Modify required Layers**
4. **Modify Image Size**
5. **Training the network and**
6. **Testing the network.**

Code:

Model is created on MATLAB & is divided into two files:

1. train.m:

This file is for training Model. It is divided into 5 steps:

i) Load Dataset:

- *Create & Label Dataset*: - Using imageDatastore(). This function requires the location of the image folder. We can include the subfolders and specify the labels of the images using this function.
  In following code, Dataset is path of image folder, Dataset has subfolders so next property is set true & label of images are their folder name.
- *Define Training & Validation Dataset*: - Here we assign dataset for training & validation to a variable.

The *Dataset* contains cropped face images of 12 people. There are 10 images of each person, making total of 120 images. We will train the GoogLeNet model using the images in the dataset folder and then test the model by using the any image to check if the model is able to recognize the face name of the images or not.

```
%#------1.  Load Dataset------#%

% Create & label Dataset
Dataset = imageDatastore('Dataset', 'IncludeSubfolders', true, 'LabelSource', 'foldernames');

% Define Training & Validation Dataset
Training_Dataset=Dataset;
Validation_Dataset=Dataset;

```

ii) Load Network:

- *Set Network*: - Set network as GoogLeNet & use analyzeNetwork(net) to see layer diagram and their value.
- *Store Input size of layer 1*: - The input size is 224×224 pixels. And it accepts images with three channels for example – RGB image.
- *Store whole architecture of network (GoogLeNet)*: - To change few layers after modification

```
%#------2.  Load Network------#%

% Set Network
net = googlenet;
analyzeNetwork(net)

% Store Input size of layer 1
Input_Layer_Size = net.Layers(1).InputSize;

% Store whole architecture of network(Googlenet)
Layer_Graph = layerGraph(net);

```

iii) Modify Required Layers:

- *Store layer 142 & 144*: - Here layer 142 is trained with task specific features. The name of this layer is 'loss3-classifier'. And layer 144 is classification layer.
- *Store number of classes in Dataset*: - Here the 'categories()' function returns the number of categories of data. Finally, the 'numel()' function returns the number of categories we have in training data.
- *Create new fullyConnectedLayer(142) & classificationLayer(144)*: Specify new name for layer & Learning rate. Here, learning rate and the bias values are weighted values not the actual values.
- *Replace layer 142 & 144*: - Using replaceLayer(), replace the existing layers by our redefined layers. The function requires the layer-graph, name of the existing layer and new layer as arguments. We have used the stored layer 142 & 144, then replace it with new one.

```
%#------3.  Modify Requied Layers ------#%

% Store layer 142 & 144
Feature_Learner = net.Layers(142);
Output_Classifier = net.Layers(144);

% Store number of classes in Dataset
Number_of_Classes = numel(categories(Training_Dataset.Labels));

% Create new fullyConnectedLayer(142) & classificationLayer(144)
New_Feature_Learner = fullyConnectedLayer(Number_of_Classes, ...
    'Name', 'Facial Feature Learner', ...
    'WeightLearnRateFactor', 10, ...
    'BiasLearnRateFactor', 10);
New_Classifier_Layer = classificationLayer('Name', 'Face Classifier');

% Replace layer 142 & 144
Layer_Graph = replaceLayer(Layer_Graph, Feature_Learner.Name, New_Feature_Learner);
Layer_Graph = replaceLayer(Layer_Graph, Output_Classifier.Name, New_Classifier_Layer);
analyzeNetwork(Layer_Graph)

```

iv) Modify Images

- *Define range of modification*: - We have randomly translated the images up to 30 pixels. Then on the second line, we defined scale range. We will scale the images up to 10%.
- *Now modify images*: - Using 'imageDataAugmenter()'. The arguments of this function come in {'Name', Value} pair.
- *Resize image for layer 1 of GoogLeNet*: - Using augmentedImageDatastore(). The first argument is 'Input_Layer_Size(1:2)'. This is the required resolution of the GoogLeNet input layer which is 224 x 224 pixels. The next argument is the image where we want to apply the augmentation.

```
%#------4.  Modify Images ------#%

% Define range of modification
Pixel_Range = [-30 30];
Scale_Range = [0.9 1.1];

% Now modify images
Image_Augmenter = imageDataAugmenter(...
    'RandXReflection', true, ...
    'RandXTranslation', Pixel_Range, ...
    'RandYTranslation', Pixel_Range,...
     'RandXScale', Scale_Range, ...
     'RandYScale', Scale_Range);

 % Resize image for layer 1 of Googlenet
Augmented_Training_Image = augmentedImageDatastore(Input_Layer_Size(1:2), Training_Dataset, ...
    'DataAugmentation', Image_Augmenter);

Augmented_Validation_Image = augmentedImageDatastore(Input_Layer_Size(1:2),Validation_Dataset);
```

v) *Train Network*

- *Specify training Option*: - Using '<u>trainingOptions()</u>'. We have specified Batch size, Epochs, Learn rate, Validation Frequency, etc. here.
- *Start training*: - Now the '<u>trainNetwork()</u>' function will train the layers we have newly added with the dataset we have provided.

```

% Specify training Option
Size_of_Minibatch = 5;
Validation_Frequency = floor(numel(Augmented_Training_Image.Files)/Size_of_Minibatch);
Training_Options = trainingOptions('sgdm',...
    'MiniBatchSize', Size_of_Minibatch, ...
    'MaxEpochs', 10,...
    'InitialLearnRate', 3e-4,...
    'Shuffle', 'every-epoch', ...
    'ValidationData', Augmented_Validation_Image, ...
    'ValidationFrequency', Validation_Frequency, ...
    'Verbose', false, ...
    'Plots', 'training-progress');

% Start training
net = trainNetwork(Augmented_Training_Image, Layer_Graph, Training_Options);

```

2. test.m:

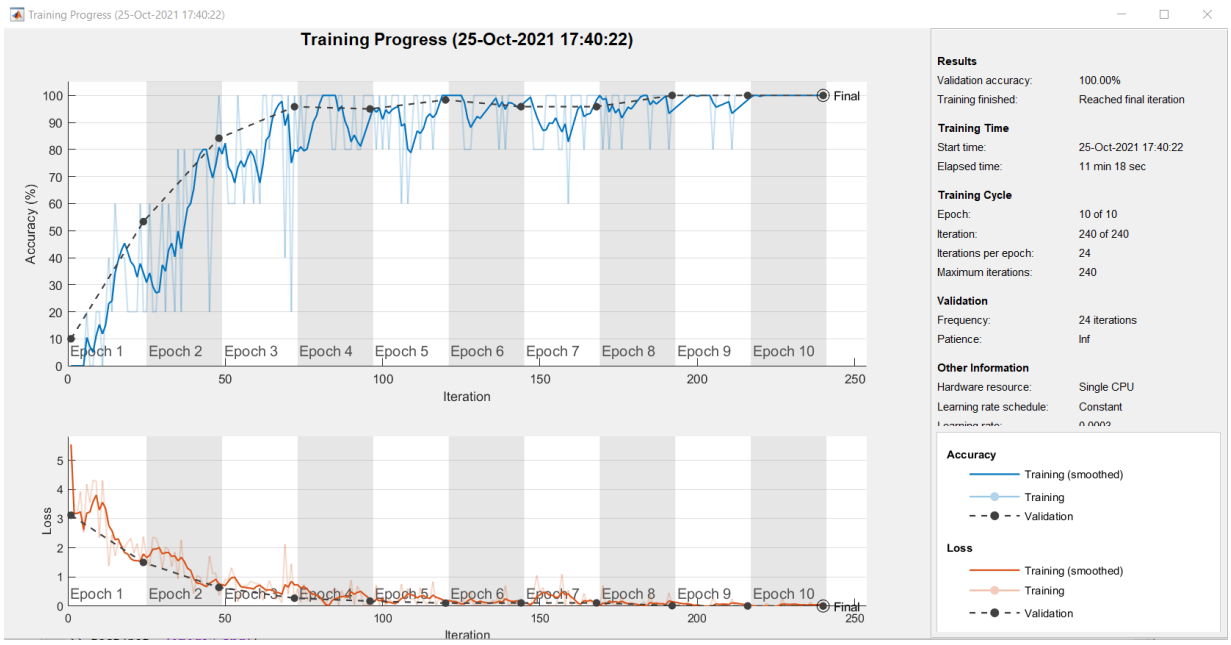
After network has been trained. Now we can pass images to this newly trained network and classify the images. In order to test the network, I have created a function named '<u>test_network(net, image)</u>'. Inside function,

- *Read Image*: - '<u>imread()</u>' function is use to read the image and store it in 'Img'.
- *Resize image*: - Then we resized the image using '<u>imresize()</u>' function with 224*224 size.
- Transfer image to network: - Then we used '<u>classify()</u>' function to classify images using the network we have trained. It takes network & resized image, then return label & probability score.
- *Now show image, label & probability*: - Then we have initiated a 'figure' object to show the image on the figure.

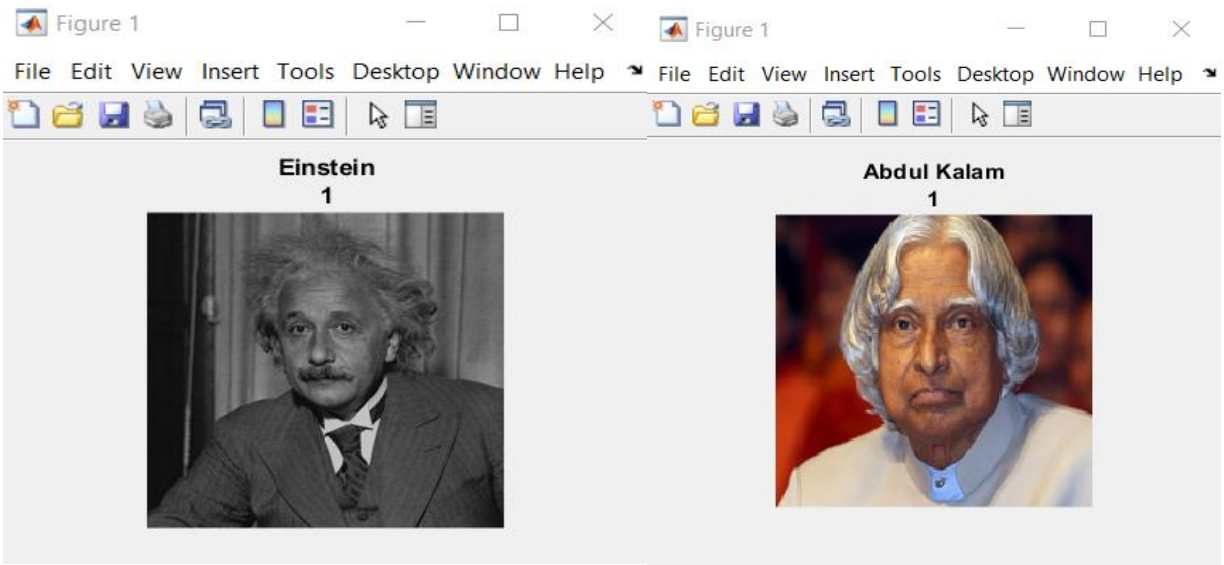
We used ‘imshow()’ function to display the resized image stored in ‘Resize’
At last, we used the title() function to display the predicted label and probability score on top of the image. Then end the function.

```
%#------  Test trained network ------#%

% Function to test network
function test(net, image)

% Read Image
Img = imread(image);

% Resize image
Resize = imresize(Img, [224, 224]);

% Tranfer image to network
[Label, Prob] = classify(net,Resize);

% Now show image, label & probability
figure;
imshow(Resize);
title({char(Label), num2str(max(Prob),2)});
end

% Use " test(net, image1) " to see result
```

Result:-



- This image is of training progress.
- In above image, we can see training accuracy close to 100% and loss close to 0 in end.
- We can also see final validation accuracy 100%



- This image is of testing process.
- This is done using ‘test(net, 'image1.jpg')’ and ‘test(net, 'image3.jpg')’.
- We have given 2 test images; Model has predicted them correctly with accuracy 100%.

Conclusion: -

In this project we have done a lot of research on related algorithms for face detection using GoogLeNet before we actually started to implement. We also look on FaceNet & VGGface for this assignment, but landed on GoogLeNet at last, since few libraries needed were not available or our PC does not support them. We chose to extract some good aspects of these well-developed algorithms and made our own. We use MATLAB because it has more graphical info on process & layers of network. And at last, we have achieved more accuracy than assignment-1. Highest accuracy as shown is 100%.