# Machine Learning Engineer Nanodegree

## Capstone Project

## Image Detection for Facial Attribute - Smiling

**Jyothi Mudur**

**January 7, 2019**

## I. Definition

## Project Overview

Machine Learning has varied uses. One such is in the field of Image Recognition to detect facial attributes in the images of people. When it comes to images, Convolutional Neural Network (CNN) models are preferred to Multi Layer Perceptrons (MLPs). After the basic preprocessing of the data (if required) is done, the images can be directly fed into the CNNs. This is however, not the case with MLPs wherein it is required to convert the multi-dimensional array (images) to a single dimensional vector. Also, the number of parameters to train would be enormously huge in the case of MLPs. Off late, many researches have been done in the field of smile detection considering various other models[2] and CNNs[3]. In this project, I would be creating a model with acceptable accuracy (as compared to the benchmark model) to detect if the person in the given image is smiling or not. CelibA [1] dataset is used here which contains over 200,000 celebrity images, each with 40 attribute annotations - 'Smiling' being one such attribute. The dataset used in this project is downloaded from Kaggle [4].

I could do this project by referring to the details mentioned in the Reference section, the knowledge acquired during the Machine Learning Engineer Nanodegree course and the lessons therein.

## Some notes on the CelibA dataset:

CelebFaces Attributes Dataset (CelebA) is a large scale face attributes dataset with more than 200,000 celebrity images, each with 40 attribute annotations. The images here cover large pose variations and background clutter. This data was originally collected by the researchers at MMLAB, The Chinese University of Hong Kong. Description of the dataset from Kaggle: https://www.kaggle.com/jessicali9530/celeba-dataset

Content

- 202,599 number of face images of various celebrities

- 10,177 unique identities, but names of identities are not given
- 40 binary attribute annotations per image
- 5 landmark locations

Data Files

- img_align_celeba.zip: All the face images, 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
- list_bbox_celeba.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
- list_attr_celeba.csv: Attribute labels for each image. There are 40 attributes. "1" represents positive while "-1" represents negative

## Problem Statement

The goal here is to implement a Convolutional Neural Network (CNN) model to be used on a subset of the CelibA dataset and successfully identify (with acceptable accuracy) if the person in the image is Smiling or not. Initially a simple CNN model would be created from scratch (this will be considered as a benchmark model). The goal would then be to solve the problem using a better model with improved accuracy. This would be achieved by either using pre-trained models (Transfer Learning) and/or augmentation techniques.

## Metrics

Accuracy of the model is considered as a good evaluation metric to quantify the performance of both the benchmark model and the final solution model.

**Why Accuracy?**

Accuracy is a good metric when classifiers are considered where the target distribution have equal contribution of positive and negative outcome and where True Positives and True Negatives are more important.

This is defined as:

Accuracy = (Correctly classified points) / Total Number of Points i.e.

Accuracy = (Number of True Positives + Number of True Negatives) / (Total number of pictures)

Another alternative is the F1 score which is defined as below. This is the harmonic mean of Precision and Recall. It is also one of the evaluation metrics that can be used in machine learning for classification problem. For skewed distribution like credit card fraud or disease detection, F1 score is preferred over accuracy. Also, this is mostly relevant when True Negatives and False Negatives are crucial [5].

F1 score = ( 2 * Precision * Recall ) / ( Precision + Recall )

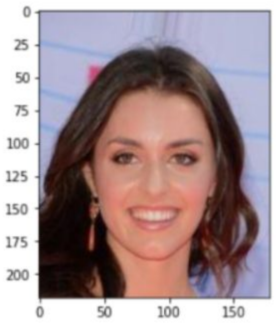
As can be seen in the Exploratory Visualization, the distribution in the initial/original dataset has almost equal number of positive and negative outcomes. The data is clearly not skewed towards either one of the outcomes. I have ensured that the number of positive and negative outcomes is the same in the subset of the data being considered in this project. Considering all of the above, **Accuracy** is chosen as the evaluation metric in this project.

## II. Analysis

### Data Exploration

In this project, a subset of the CelebA dataset which includes 200,000+ colored images of 218 x 178 pixels is being used. All these images are jpg images cropped and aligned. Each image has 40 attributes like Smiling, Attractive, Bald, etc associated with it. All these details are available in the 'list_attr_celeba.csv ' as a part of the dataset. In this project, the attribute under consideration is 'Smiling'.

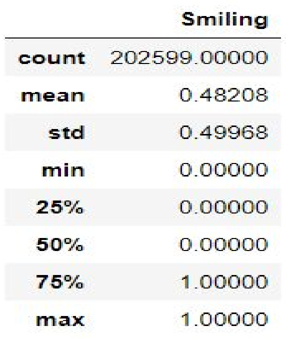
A sample image from the list is considered and the corresponding 'Smiling' attribute is extracted for observation.



Filename: 000043.jpg
Smiling

### Exploratory Visualization

To begin with, let us consider a subset of the dataframe (the attributes are read from the csv file as a dataframe - `'list_attr'`) - `'list_attr_smile'` - which consists of the details of image id with only the `'Smiling'` attribute. A description and a plot of this can be seen as below:



| | Smiling |
|---|---|
| **count** | 202599.00000 |
| **mean** | 0.48208 |
| **std** | 0.49968 |
| **min** | 0.00000 |
| **25%** | 0.00000 |
| **50%** | 0.00000 |
| **75%** | 1.00000 |
| **max** | 1.00000 |

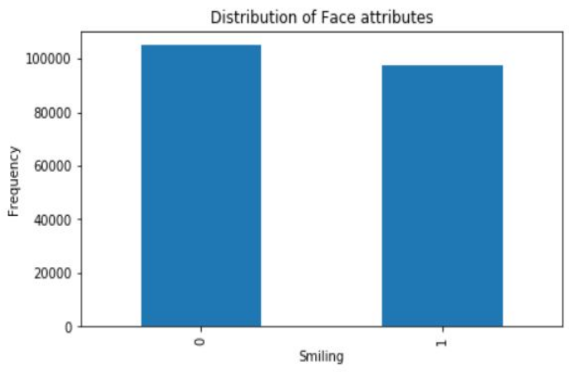
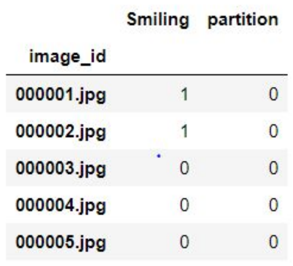


*Fig.1: Summary of dataframe - list_smile (left), Distribution of Face (Smile) attributes (right)*

From the above, we see that the target distribution has almost equal contribution of positive and negative outcome. The recommended details for partition (Training, Validation and Testing) is given in '`list_eval_partition.csv`'. Reading this as a dataframe, we have:

```
0    162770
1     19867
2     19962
Name: partition, dtype: int64
```

where 0,1,2 indicates Training, Validation and Testing respectively. Both the data frames are merged per the image_id. A snapshot of the first five items in this dataframe is as shown below.



| | Smiling | partition |
|---|---|---|
| **image_id** | | |
| **000001.jpg** | 1 | 0 |
| **000002.jpg** | 1 | 0 |
| **000003.jpg** | 0 | 0 |
| **000004.jpg** | 0 | 0 |
| **000005.jpg** | 0 | 0 |

*Fig.2: First five items in the combined dataframe*

**Jupyter Notebook**: *data_prep.ipynb*

## Benchmark

A simple CNN model was developed with some layers of Conv2D alternating with MaxPooling layers and dropout layers. Dropout layers with values (0.25) was added to avoid overfitting of the data. This model was trained on the same subset of data that was chosen to train the final model. As detailed in the metric section, accuracy is considered as an evaluation metric. The simple model was able to predict smiles with an accuracy of 81%. Hence, I consider this as a good benchmark.

## Algorithms and Techniques

The accuracy achieved with the benchmark model is 81%. The goal here is to develop a model with different techniques to achieve an accuracy better than this. Convolutional Neural Network (CNN) is the classifier used in this project. New classifiers are built based on the pre-trained InceptionV3 and VGG-19 trained on the ImageNet dataset[6].

1. **Reduction in the number of samples:** Due to computing resource limitations to ease the work, a subset of the CelibA dataset is used for this project. The number of samples considered:

| | |
|---|---|
| Training | : 10,000 |
| Validation: | 2,500 |
| Test | : 2,500 |

A python function is written to consider the above number of samples as per the recommended partition and based on the attribute so as to get equal distribution of positive and negative outcomes.

2. **Data Augmentation**: Data Augmentation is used as one of the methods to improve the model [7].In real life scenarios, the attribute / feature to be considered need not be in the center of the image as with the case of our dataset that is considered. Hence, we need to ensure that our data is robust against all these scenarios. Hence, the data is augmented/expanded artificially by creating modified versions of the images (by shifting, flipping, zooming, changing the brightness,..) in the dataset. This is achieved in Keras deep learning neural network by the use of *ImageDataGenerator* class[8]. This generates batches of tensor image data with real time data augmentation. The data will be looped over in batches. In this project, subsets of the initial datasets are created and copied to directories as per the categories. Hence, considering .flow_from_directory(directory) function to create training, validation and testing data generators.

2. **Transfer learning**: Improvements are made in the model to increase accuracy by considering pre-trained networks like InceptionV3 and VGG19. These are trained on ImageNet and features are extracted accordingly. Here the considered dataset and final expectation is different. Hence, dense layers to suit our cases-like output of 2 categories are added at the end. The initial pretrained network layers are frozen and only the newly added layers are trained accordingly.

3. **Tuning the hyper parameters**: The hyperparameters are tuned based on several trials and finally fine-tuned to get the improved model. For optimizers, 'rmsprop', Stochastic Gradient Descent (SGD) are tried with a very low learning rate and momentum. Epochs of 5,20,.. were tried. Batch size of 32 is considered.

4. **Using bottleneck features:** When pretrained models are tried initially, random weights are used. Also, the model was trained with ImageNet dataset. Hence, in this trial, the pre-trained model VGG19 is run once with the considered CelibA dataset and the weights are slightly adjusted. This is then saved as bottleneck features. To this, we add a couple of few custom layers to build a composite model. This is then trained accordingly also considering the saved weights for the pre-trained model. This is then compiled and used to predict the required attribute ('Smiling').

Combination of the above is also considered to achieve the improved model - Bottleneck features with data augmentation along with fine tuning the model for the weights and hyper parameters.

## III. Methodology

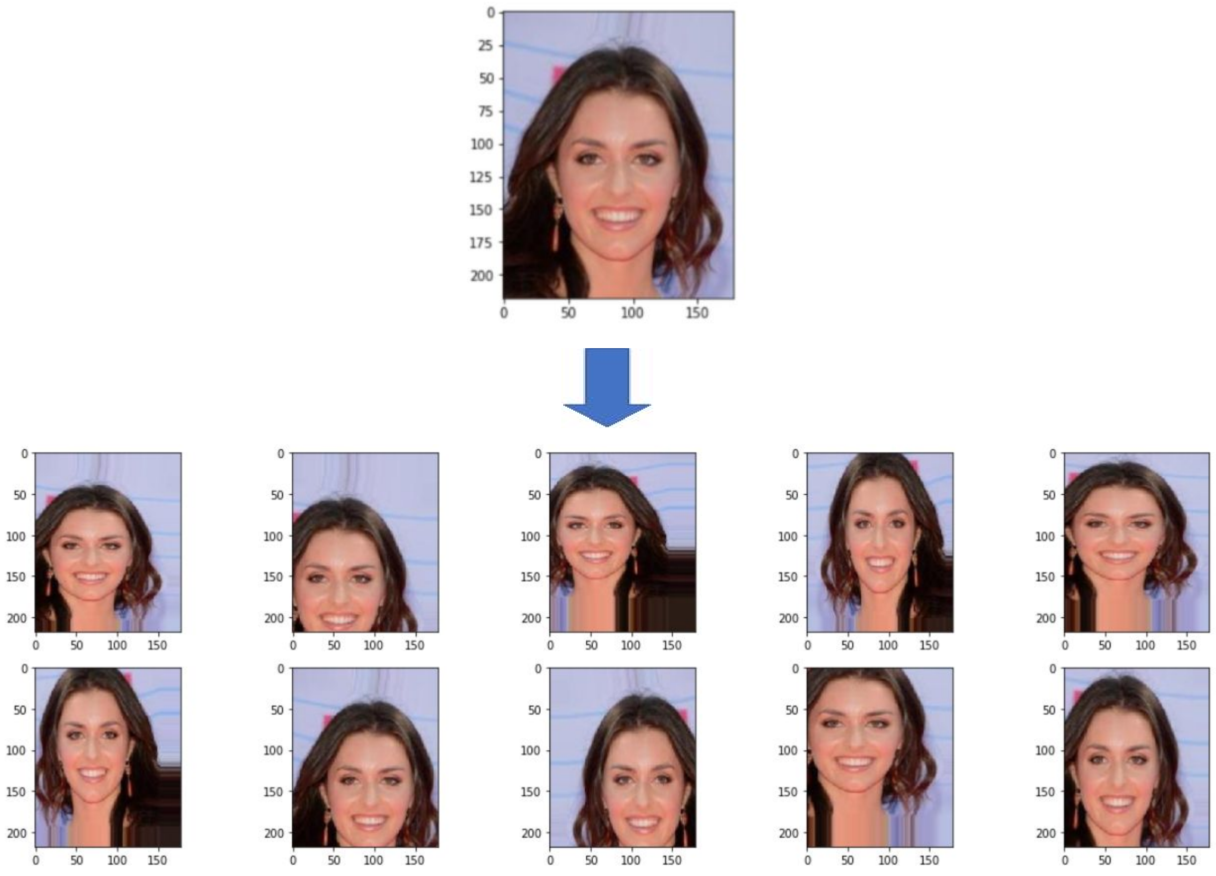
## Data Preprocessing

**Data Augmentation to increase variety in the data set:**
As described in the Algorithms and Techniques section, Data Augmentation is considered to increase the robustness of the model. It will add variety that may be present in the real world data; but absent in the training data. The output of the data augmentation exercise is used as input to train the model.

I have used the following transformations to create augmented images:

1. Width_shift_range = 0.2
2. Height_shift_range = 0.2
3. zoom_range = 0.2
4. horizontal_flip = True

The ImageGenerator used for this purpose will use random combination of the above transformations to any given input and hence the number of combinations are actually much higher than just 4.



**Pre processing images:**

There are three kinds of pre-processing done on the images

- Loading images into multi dimensional arrays
- Transforming images to be useful for pre-trained models
- Transforming images to reduce exposure to outliers

Images are loaded into memory as multi-dimensional arrays. In the CelebA dataset, all the images have height of 218 pixels and width of 178 pixels. They are color images hence each pixel is loaded as combination of values of red (R), green (G) and blue (B). Hence the images are loaded into a multidimensional array of shape 218 x 178 x 3. This is done by the pillow library. When using TensorFlow as backend (for pre-trained networks), Keras CNNs require a 4D array. Hence an additional dimension is added using numpy to make it 1 x 218 x 178 x 3

However each pre-trained model requires its input image to be coded in the format that it prefers. Some models restrict the range of values the data can take between 0 and 1, while others do it -1 to 1. This difference is normalised by each pre-trained model having its own preprocess_input method[9]. Care must be taken to make sure that I use the right preprocess_input method corresponding to the pre-trained model. It should also be ensured that these preprocessing should be done on the images only once.

When not using any pre-trained model, it is good to keep all the pixels in the same range of values in order to limit the exposure to outliers. This can be done by dividing every pixel in every image by 255. This method of pre-processing was used in the CNN implemented as the benchmark model.

## Implementation

Here are the steps I followed in the implementation of model for smile classification:

1. Implementing benchmark model using a simple CNN
2. Choosing a good pretrained model as base model
3. Creating a top-level model that utilizes the features determined by the base model and does the classification
4. Creating a composite model by stitching the base and the top-level model
5. Fine tuning the composite model
6. Testing the final model

**Implementing benchmark model:**

I created a simple CNN model with these layers:

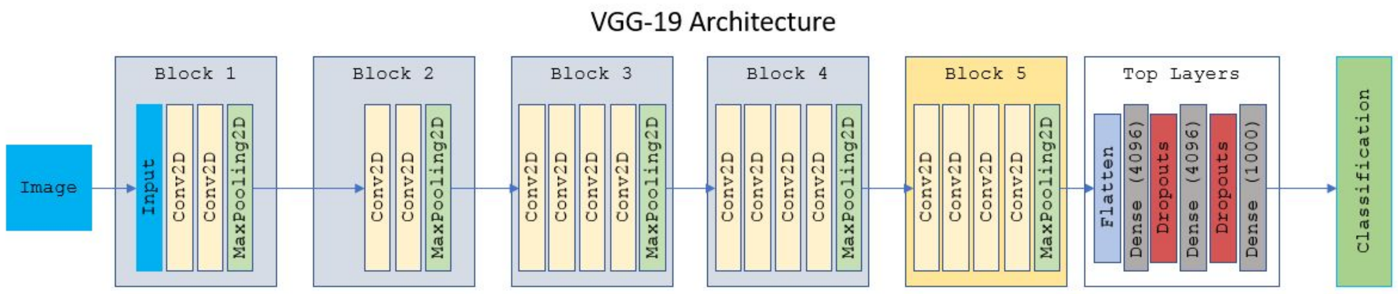
1. Four `Conv2D` layers with 16, 32, 64 and 128 filters with `'relu'` activation
2. Four `MaxPooling2D` layers after each `Conv2D` layers
3. Two `Dropout` layers of 0.25
4. One `GlobalAveragePooling2D` just before the final layer.
5. Final `Dense` layer of 2 nodes with `'softmax'` activation

Since this was a CNN being trained from scratch and sufficiently deep, I compiled the model with the rmsprop optimizer and accuracy as the metric to optimize.

For fitting the model, I ran 20 epochs of model against 10000 training and 2500 validation images; saving the model with the least validation loss. This model was able to achieve 81.04% accuracy on the testing dataset consisting of another 2500 images from the CelebA data set.

This CNN model was able to achieve reasonable accuracy without any complex techniques like data augmentation or transfer learning. Hence, we can consider this as a good benchmark to compare the final model against.

**Choosing a good pre-trained model as base model:**
For an efficient image classification solution, I had to take advantage of already trained models. These models have deep convolutional layers that generalise features of any given dataset. I have to make use of these generalised layers and build the final classification layer that is specialised for my application - smile detection. Towards this, I have chosen to use `VGG-19` as the base model.



`VGG-19` was also trained on imagenet and hence should be able to detect the features in a human face. It has five blocks of convolution layers followed by a series of dense layers for classification. I am not interested in the classification and hence, will throw away the dense layers.

`VGG-19` model takes input as an image with dimensions 224 x 224 x 3 by default. I have to therefore, instantiate the model with dimensions appropriate for our data set - 218 x 178 x 3. Also, the weights need to be from imagenet so that it will be useful for our classification.

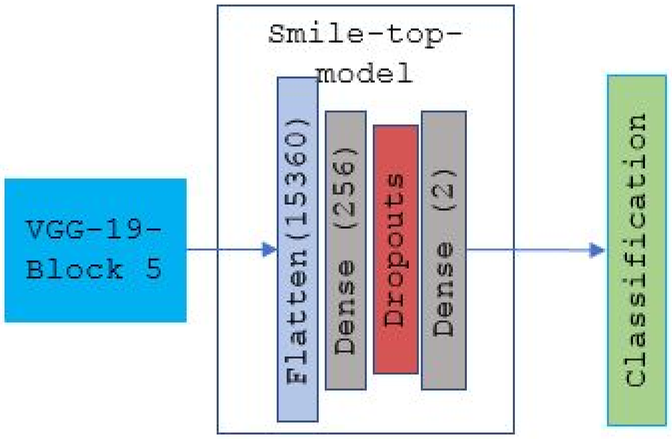
In the dog-project, I had learnt about bottleneck features and how they can be extracted from a pre-trained model. This was used to significantly reduce the time taken to train a model from scratch. The bottle neck features were extracted for the training and validation data sets.

source: https://github.com/keras-team/keras-applications/blob/master/keras_applications/vgg19.py

**Creating the Top-Level model:**
Once the top layers of `VGG-19` is removed, I have to replace it with layers that will do the classification that we desire - smiling or not-smiling. For this I have used MLP (multi-layer-perceptron) with just one Dense layer of 256 nodes and a dropout layer of 0.5. Final classification is done by a Dense layer of 2 nodes with softmax activation. This Top-Level

model takes as input the output of VGG-19 model which is a ( 6 x 5 x 512 ) matrix. As we can see, this is a very deep matrix compared to the 218 x 178 x 3 input. For the MLP, we need to flatten this input into a single dimension array of 15360.

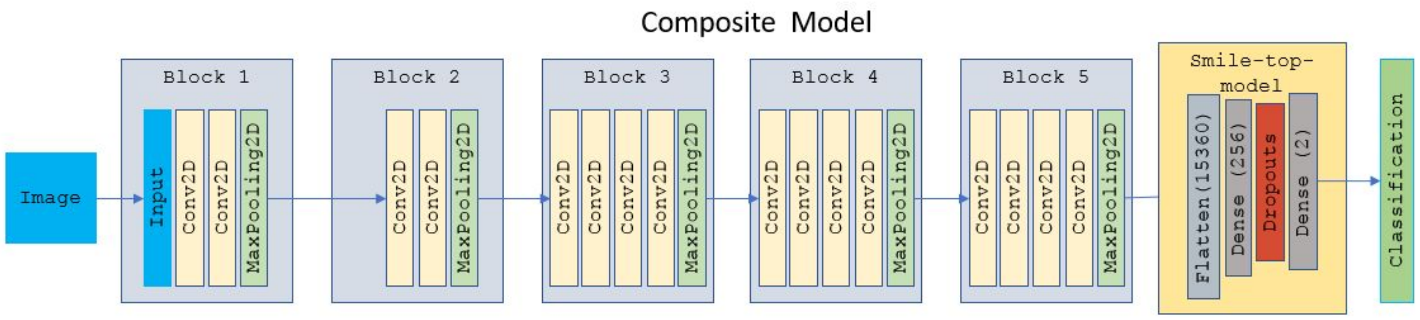
Hence the top-level MLP looks like this:



Once the MLP was created, it was trained on the bottleneck features extracted by `VGG-19` from the 10000 training images. I was able to run 50 epochs of this MLP which achieved close to 83% validation accuracy. I saved the weights of the top model as smile-weights to be used later.

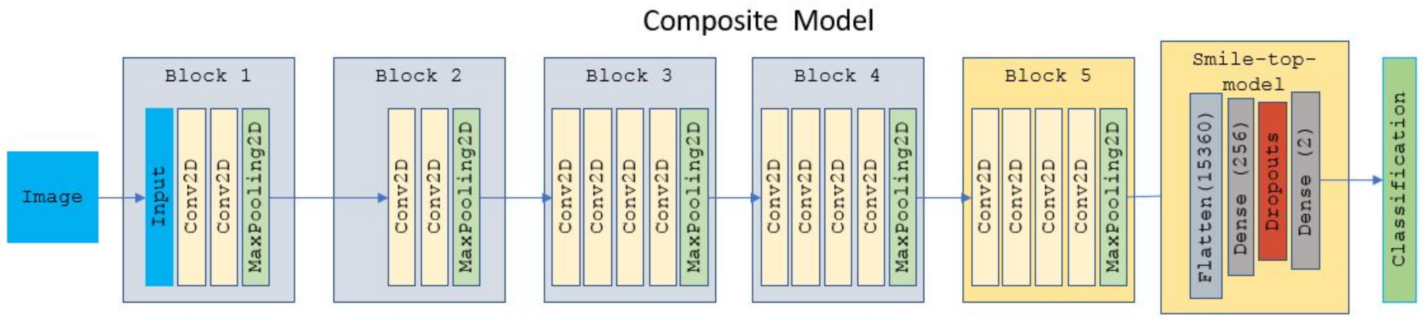
**Creating a composite model:**
Now we have two component models - `VGG-19` without top levels and custom top level model specialised just for smile detection. We need to create a composite model that can take an image of dimensions 218x178x3 as input and produce a categorical distribution output of - not-smiling (0) and smiling (1). This is done by using sequentially adding these models. The weights were loaded - ‘imagenet’ for VGG19 and ‘smile-weights’ for top-model.

Hence the composite model now looks like this:



This model was tested on the test data set and achieved 81.04% accuracy. This was surprisingly exactly the same as the benchmark model.

**Fine tuning the composite model:**
Once the smile-weights are loaded for the top level classification layers, I could then attempt to unfreeze the last convolution block of `VGG-19` and re-train then entire model.



After a few failed attempts, I realised that detailed thought has to be given towards choosing the right optimizer for the training. I finally choose Stochastic Gradient Descent (SGD) [10] optimiser with a very slow learning rate (0.0001) and leaving other parameters at default. This model has 13,372,162 trainable parameters. Hence it takes a lot of time to train. I ran this for only 5 epochs and ended it with a 90+% of validation accuracy.

## Refinement

1. The first attempt to improve on benchmark was done by just adding data augmentation to the benchmark model. In this model, I learnt about the requirement of pre-processing an image.
2. I built the next CNN right on top of frozen layers of VGG-19 and training the entire model with these layers:
    - `GlobalAveragePooling2D`
    - `Dense` with 1024 nodes with `relu` activation
    - `Dropout` = 0.25
    - `Dense` with 512 nodes with `relu` activation
    - `Dropout` = 0.25
    - `Dense` with 2 nodes with `softmax` activation

I used RMSprop as the optimiser. When doing this, it was clear that learning rate was too much causing the training loss and validation loss to diverge significantly right after the initial few epochs. Hence this was abandoned.

3. I realised that RMSprop was the cause for this huge jumps in weights of parameters resulting in widely diverging loss values. I learnt that using SGD with very slow learning rate (0.001) is better suited for these conditions. Hence I tried with that setting.

4. Finally, I read about the technique of transfer learning using bottleneck features from this article[11]: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

5. Using data augmentation for fine tuning the last block of convolution layers in the base model

6. Finally, the solution had all these components
    a. VGG-19 base model without top layers
    b. Transfer learning using bottleneck features
    c. Fine tuning of Last convolution block of VGG-19
    d. Data Augmentation for variety in training data
    e. Using SGD with a very slow learning rate.

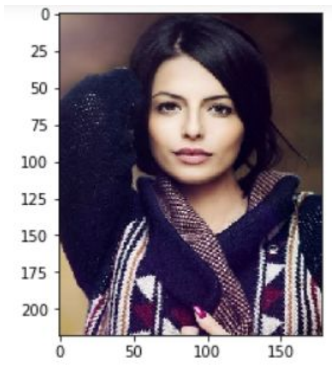
## IV. Results

This table summarises the results of various models that were built on the path to the final model

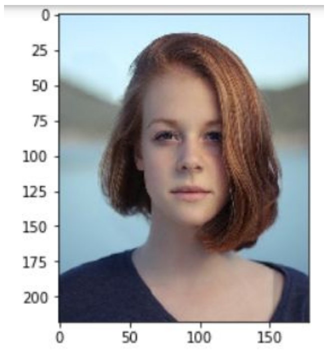
| Model | Accuracy | Jupyter Notebook / Notes |
|---|---|---|
| Benchmark | 81.04% | *benchmark_model.ipynb*<br>Conv2D, Dropouts and Dense layers<br>Optimizer used = RMSProp; 20 epochs |
| Benchmark + Data Augmentation | 82.12% | *benchmark_data_aug.ipynb*<br>Conv2D, Dropouts and Dense layers<br>Optimizer used = RMSProp; 20 epochs |
| Transfer learning by simple concatenation of models | 74.28% | *transfer_learning_VGG19-SGD.ipynb*<br>Freeze all convolution layers of VGG-19.<br>Train the composite model starting with random weights.<br>Optimizer used -> SGD with learning rate 0.0001; 20 epochs |
| Transfer learning using bottleneck features | 81.04% | *VGG19-Transfer-BottleNeck-Aug.ipynb*<br>Store bottleneck features from VGG-19 model<br>Train the top-level model using bottle neck features.<br>Freeze all layers in VGG19.<br>Replace the top level classification layers with smile classification<br>Optimizer used -> SGD with learning rate 0.0001; 50 epochs<br>This was the model that learnt fastest. |
| **Final Composite model** | **90.64%** | *VGG19-Transfer-BottleNeck-Aug.ipynb*<br>Store bottleneck features from VGG-19 model<br>Train the top-level model using bottleneck features.<br>Unfreeze the final convolution layer in VGG-19<br>Retrain - the last convolution layer + smile-top-layer with starting weights from previous run.<br>Data Augmentation with 4 variations was used to train the model.<br>Optimizer used -> SGD with learning rate 0.0001; 5 epochs |

## Model Evaluation and Validation

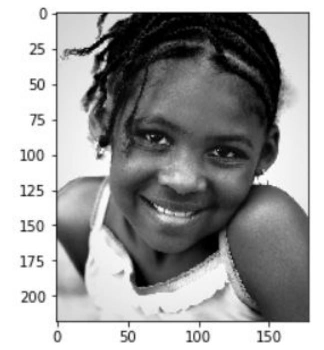
**How does the model perform on real world data?**
I used about 50 images of people downloaded from public sources to test the model with images that were not present in the dataset.
Examples of the images that were predicted correctly:



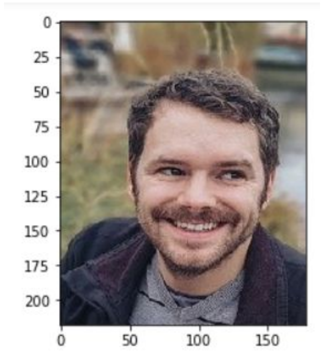
```
Path:   images/0/00000.jpg
Prediction:      Not Smiling
Result:             Correct
```



```
Path:   images/0/00005.jpg
Prediction:      Not Smiling
Result:             Correct
```

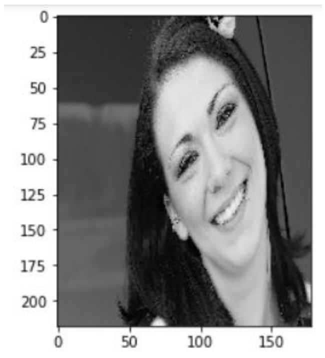


```
Path:   images/1/00006.jpg
Prediction:          Smiling
Result:             Correct
```

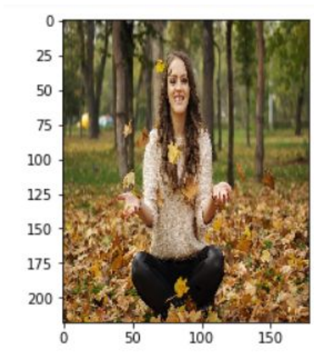


```
Path:   images/1/00017.jpg
Prediction:          Smiling
Result:             Correct
```

1. Model was able to classify correctly most of the images that had frontal face occupying a significant portion of the image.
2. Whether the image was grayscale or color did not seem to matter to the model.
3. The dimensions / aspect ratio of the images also did not impact much. Smiles and faces were recognised even when the aspect ratio was slightly distorted to fit the image to a 218 x 178 dimension
4. However, some images (images/1/00005.jpg) were wrongly classified in spite of having full face visibility. This revealed that the model has some weakness in classifying faces that are tilted i.e. not upright. Even though I had used data augmentation as a technique to generalise the model, it did not help since I did not have the rotation option enabled. This can be pursued as one of the improvements that can be made to model to achieve higher accuracy.



```
Path:   images/1/00005.jpg
Prediction:      Not Smiling
Result:          Incorrect
```



```
Path:   images/1/00010.jpg
Prediction:      Not Smiling
Result:          Incorrect
```

5. When the face in the photo was small, detection of smile was inaccurate. (Ex: images/1/00010.jpg above). There is a lot of non-face features in the photo that may have come in the way of correctly identifying the facial features. We may have to pass the image through a face detection layer like haarcascades, and then pass only the image in the bounding box to the smile classifier.
6. The model did not have any false positives. No smile was detected in any image that did not have a smile.

**How does the model perform against the rest of CelibA data set?**
In order to ensure that these were the only shortcomings in the model, I ran them against the entire testing data of CelibA data set of 19962 images. The model was able to predict the classification with **90.43%** accuracy. This is in-line with the accuracy obtained when the model was tested on the subset testing data of 2500 images. This further strengthens the decision to run the training / validation / testing on subset rather than the whole data set. If the dataset is sufficiently generic, then a non-trivial subset can achieve the desired accuracy much faster than complete dataset.

**Jupyter Notebook:** *visualizations.ipynb*

## Justification

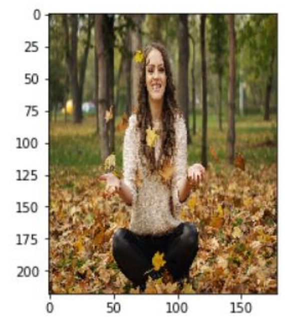
The final model performs better than the benchmark model on the chosen metric - **accuracy**.
The final model however, took a lot longer to train / validate / test on the subset of the CelibA dataset. The prediction time was also marginally longer.

## V. Conclusion

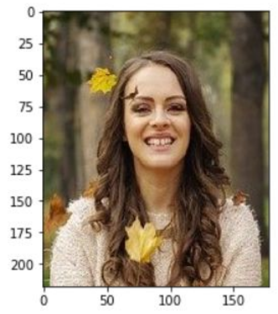
## Free-Form Visualization

**Impact of zoom on the ability of model to detect facial features - smile.**

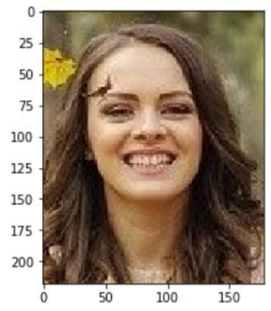
It appears that the model is sensitive to the relative size of the facial features compared to the rest of the image. As can be seen in the below two examples, the model fails to detect the smile in the first two images in which the face is occupying less than approximately 20% of the image size. Once the face gets larger than that, the smile is detected with probability just short of certainty. It would be interesting to analyse what is the minimum size of the face that should be present for smile to be detected.



```
Path: images/1/00010.jpg
Prediction: Not Smiling 93.28%
Result: Incorrect
```



```
Path: images/1/00019.jpg
Prediction: Not Smiling 73.16%
Result: Incorrect
```



```
Path: images/1/00023.jpg
Prediction: Smiling 99.94%
Result: Correct
```

In the above example, the model was fairly certain that there was no smile in the unzoomed image - 93% certain. However, with intermediate zoom, the probability reduced to 73%. When the face occupied ~30% of the image, the model switched prediction with very high confidence - 99.94%
**Jupyter Notebook:** *visualizations.ipynb*

## Reflection

This project has given me an opportunity to put all the theory learnt into good practice and there-by solidify the understanding. This project had XXXXX distinct phases

1. **Data Analysis**
   The CelibA data set was a really large data set. I had to run the entire project on a CPU and hence would not be able to use the full dataset for training. Hence, had to come up with a way to select a reasonable subset of the data without compromising on the generality of the features.

2. **Benchmark creation**
   I was surprised at the accuracy of the initial benchmark model. The straight-out-of-text-book solution of slapping together a few Conv2D models with dropout layers for feature extraction and a couple of Dense layers for classification worked pretty well to give 81% accuracy. Very soon, I realised that incremental improvements on top of that initial benchmark is hard. Even after data augmentation and re-training the model by consuming 2x the time, I was able to improve only to 82%. I learnt that here is where a simple model will hit the ceiling. To make it more accurate, we would need to harness the power of already trained models.

3. **Transfer Learning**
   There is no guidance on the internet forums on which base model to choose for this application. All image classification models like VGG-16, VGG-19, InceptionV3, ResNet-50, etc are all equally popular. Although, I started with a model with InceptionV3 as base - I ran into some incompatibilities (which I later realised to be simple coding errors). I found a very good tutorial on-line to help me learn about transfer learning using VGG-16[11]. I was able to replicate the same steps with necessary modifications to suit VGG-19 and Keras 2.0.

   I assume any base model is sufficiently trained with state of the art techniques. If I am going to unfreeze some blocks of base model, then those layers and the the top-layers on top of them should start off with reasonably accurate weights. If we start off with random weights, then there is a chance that model will settle in a less optimal state. This is especially important since I am running the training on a CPU and do not have time / compute engines that can bring the model back to the same level of accuracy. Hence the two pass approach to build the composite model was a great learning.

4. **Fine tuning**
   There are many parameters that are configurable and do not have clear guidance on what they should be.
   I used the divergence between validation loss and training loss to be an indication of over-fitting. When that happened, I increased the Dropout parameter.
   When the validation loss was going up and down wildly, I realised that it could be because of using an incorrect optimiser for this purpose.
   When doing data-augmentation, I experimented with doing augmentation for validation data as well. However, later, I removed it because the random changes may influence the choice of the best model saved. I am still not convinced one way or another and it requires more iteration and experiments to know the influence of this parameter on the overall accuracy of the model.

5. **Evaluation and Validation**
   The model was evaluated on the entire testing set of 19652 images, in addition to the 2500 initially chosen to test. The accuracy was roughly the same - 90.43%. This shows that as long as the images are resized and cropped to show the full frontal features, the model will be able to predict the smile.
   However, when the model was tested on random real world images, the accuracy was much less - ~75%. As expected, the model did not have any false positives. It was interesting to learn the reasons for missed detection. I found two reasons - too small face, rotated face. Both these can be fixed as mentioned in the Improvement section.

## Improvement

1. This project has provided me a very good template on how to use transfer learning. I could not find any comparative study available which discusses the relative accuracies of various base models. I plan to generalise the above project to take the base-model as input and generate accuracy matrix for all of them.
2. Implement a face detection preprocessor and provide only the face to the smile detection. This will considerably increase accuracy for predicting smiles in real world images where the face itself occupies a small area relative to the whole image. An attempt at implementing this is done in visualizations.ipynb. This technique was able to improve the accuracy of the model against randomly chosen real-world images from 66% to 90%. However, this additional step is time consuming and it took almost 1s per image.
3. The current model expects only one face in the image. If there are more faces, it does not recognize the smiles. I can use the above technique to detect all faces using the pre-processing step and feed each one to the smile detector.
4. Smile Detection has various applications. In some ways, it gives out the “mood” of the person in the image. It is also however the most “faked” facial feature. I will try to see if there is any way to detect a real smile v/s a fake one. It may require a high resolution image that shows other features like pupil dilations, cheek contours.
5. The model can be improved by including images that only have partial faces - non-frontal-face images. In the real world images, it is natural for the subject to be looking elsewhere.

## References

[1] http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

[2] https://www.csie.ntu.edu.tw/~fuh/personal/FaceDetectionandSmileDetection.pdf

[3] https://arxiv.org/pdf/1802.02185.pdf

[4] https://www.kaggle.com/jessicali9530/celeba-dataset

[5] https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2

[6] http://www.image-net.org/

[7] https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

[8] https://keras.io/preprocessing/image/

[9] https://github.com/keras-team/keras/blob/master/keras/applications/imagenet_utils.py

[10] https://keras.io/optimizers/

[11] https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

https://datacarpentry.org/python-ecology-lesson/03-index-slice-subset/ (used in data_prep.ipynb)

https://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html(used in data_prep.ipynb)

https://www.kaggle.com/bmarcos/image-recognition-gender-detection-inceptionv3

https://pixabay.com/ for real world images which are downloaded royalty free.