Image Search on Mobile Phones made simpler Final Report

Rajagopaalan Sethuraman
School of Computing, Informatics and Decision Systems Engineering
Arizona State University
Tempe, Arizona 85281
Email: rsethur1@asu.edu

Vishal Srivastava
School of Computing, Informatics and Decision Systems Engineering
Arizona State University
Tempe, Arizona 85281
Email: vsriva10@asu.edu

1. Introduction

In this Project, we are going to implement Face net to recognize images based on similarity in context, location etc. for low power devices like smartphones. To boost the performance of convolutional neural network, we plan to use optimized for inference models with freeze weights using Tensorflow for smartphones.

2. Recent Work

Some Already existing work[1] have been there for Face Recognition implementations with accuracy as high as 94.6%, but with changes brought in by Facenet[2] and Deep Compression Networks[5] it has become possible to computation on Mobile system. Even though advanced techniques exists in implementing face verification and recognition there exists serious challenges to current approaches. At present the existing system Face Net[2] learns a mapping from face images to Euclidean space where distances correspond to measure of similarity between the faces. This space has been produced, and tasks like face recognition, verification can be implemented using techniques with Face Net embedding as feature vectors. This uses convolutional network to optimize the embedding.This approach has much better accuracy. The main challenge in this approach is to implement face net through Tensor flow for low power devices like smartphones.To improve the performance in mobile phones we did plan to implement one of the implementation of FaceNet with the idea of deep compression but due to complexity involved in pruning, quantization and Huffman encoding for the complete model, we are keeping the above for future work. Also we were looking into float16 based model representation but due to lack of hardware resources to train the model using Tensorflow, we are going with float32 with optimized for inference model. Also I believe the Tensorflow inference port lack the support for float16, so we abandoned the idea.

3. Theoretical Background

There had been multiple improvements in Computer Vision applications due to advent of techniques like Convolution Network, Batch Normalization and processing capabilities of embedded devices.

Convolutional neural network has made a breakthrough improvement in the Image classification problems. Due to high correlation among adjacent pixels in the image, it has become possible to extract a smaller condensed representation of image which made it efficient to process large images with relatively less time. We are using multiple convolution layer to harness this feature.

Batch Normalization[7] has also made it easier to train a network with similar images from different sources. It solves the problem of covariance shift that is observed when we look into dataset accumulated from different sources. The difference in pixel intensity and source related divergence make it difficult to train images if they belong from different sources. It effectively normalizes images in a batch makes it easy to learn and thus gives us boost in training performance in even lower no of iterations. We used this approach late in the project to solve the problem with difficult and unstable learning problem.

Our success of the project is also due to recent advancement in computing capabilities of smartphone. There has been a growing trend in embedded computer vision using deep convolution network. Due to which various firms have started looking into porting them in their applications to improve the inference capabilities and explore new domains of applications possible. It is due to increase in processing capabilities of devices that we are able to process deep networks for inference with realtime capabilites. Though the results aren't good but its promising enough to approach in the direction. In support of this, we are using Tensorflow inference model provided by Google to process on the smartphone itself. With some nominal lag we were able to achieve good results on face detection on images.

4. Approach

Our approach is to use a version of FaceNet implemented on Deep Compression CNN using Tensorflow framework for Android. For the current implementation we are aiming to provide tags to photographs based on face recognition and search photographs of family and friends using generated tags and context such as timestamp, event, Geo-spatial context etc. We are using cropped grayscale images from dataset labeled faces in the wild for our training and inference purpose. For improving image search we are using inference tags generated from the image classification. For this project we are using nine classes/people for which we are training and inferring. Based on hits from search string we ordering the images in grid view and presenting the result. We are diverging from our plan of training on device due to unavailability of Tensorflow port and going with inference on a trained model.

A brief discussion on various aspects of the project has been provided below.

4.1. Facenet

We are using Facenet developed by Google. Face Net learns a mapping from face images to a compact Euclidean space where the cosine distances denotes a measure of face similarity. This space has been created, tasks such as face recognition, verification and clustering are easily implemented using techniques with Face Net embedding as feature vectors. The Existing Face Net network consists of a batch input layer and a deep CNN followed by L2 normalization and the triplet loss. Face Net uses a deep convolutional network trained to optimize the performance , rather than an intermediate layer as in earlier deep learning approaches. In order to train, use triplets of aligned matching or non-matching face patches. It learns an embedding ,from an image x into a feature space, such that the Euclidean distance between all faces of the similar identity is very small, but the squared distance between the face images from different identities is significantly large. The benefit of this approach is much greater efficiency. Evaluation of these architectures on a wide range of parameters, leads to different training time and accuracies.

4.2. Android

For our application we are choosing Android OS as its the most widely used open source embedded OS. As Google has been providing ample support of Tensorflow to be ported on Android, we are taking that approach. For now we feel that major challenge will be to provide a real time inference of the photographs collected. As embedded device have really limited computing capabilities, porting something like deep neural networks is a challenging task both in terms of memory footprint and computing power. We are also concerned with the power and energy limitation which are inherent challenges posed by the mobile devices. We are looking into various network ideas and optimization techniques which we can used for this work. A brief discussion on optimization is provided in later sections.

4.3. Context tags

Context tags are used to reference pictures in a database. It can include time, place and image inference data. We are planning to use these tags for generating nouns and based on that generate a search based database. For example nouns such as person's name (Mike, Harry,...), location (Tempe, New York City,...), time(Evening, Morning, 11AM,...) will be used to generate tags which can be used for search. Indirect inference based tags can also be generated and used in the search (like Bob's birthday party) but for current implementation we will use direct inferences. Also Indirect inferences require data from events belonging to personal calendar which gets fairly complicated, so for now its outside the scope of the project.

4.4. String parsing

In this we used string based search to find the image. String will be a sentence or a phrase containing enough nouns to be able to generate a meaningful search result. We are parsing the phrase/sentence and pick out the nouns and then is searched. In the event of failure to narrow down the search result multiple results belonging to different context maybe presented. Search is based on the most hits with tags and so is ordered likewise in the thumbnail view.

4.5. Optimization

Optimization is a crucial section of the project. As we want to have reduction in memory footprint and inference delay by many magnitudes we probably have to sacrifice some amount of accuracy. This tradeoff is something we have looked into and found a optimal point for relatively good realtime inferences. Earlier we planned to use training on device but due to lack of computing power and unavailability of Tensorflow port for Andorid we are only going for inference on device. For future work we will look into methods where device won't be using any external source for computing purpose. We will look into how we will be training the model and specifically when so as to maintain good user experience. For now we are planning for future work to do training on the data only when the mobile device is put on charging. As user won't be using the device during charging and also we can draw more energy during charging, it seems good in both user experience and energy requirement.

5. Implementation

For our implementation we are using Labeled Faces in Wild dataset to get training and inference image data. For our implementation we will be using a subset of the data samples and be using it to train with nine faces. We have designed a Android application for this purpose. Android application is compiled with the Tensorflow inference model and a trained graph to classify the nine faces. At the start the application looks into a certain directory on the device for retrieving images. As it retrieves the images it produces entry to a database which contains data such filename, geographical position tag if available and time of creation and inference tag. Inference tags are generated by passing the image through Tensorflow trained graph on the device and retrieving the result from the graph for classification and adding the class name as the inference tag. This whole process is currently done every time we open the Android application which can make it non responsive for two to five seconds initially. After the inference tags are available for dataset we can process search queries on them. Using the Search bar in the application we can search for a string and based on the no of word hits it refreshes the grid list of images. We can also click on the image in the grid list to know what all tags have been assigned to the images. As the dataset is all cropped images of a single person, we will at best get one inference tag. The accuracy of inference tag depends on the amount of training done for it so it depends on image to image. The idea was to use a image, convert it to grayscale, find faces in the picture and generate down-sampled cropped images from the faces in the picture, passing it through the device and adding labels for names of the people in the image. Figure 1 shows the desired result which we were trying to achieve. We were supposed to proceed with this implementation but due to lack of time we are working on the cropped grayscale images. The actual implementation is Figure 2. Also without much data on a certain class it becomes difficult to train a model for which we would require to generate more images artificially, which could be challenging due to complexity of the dataset.

6. Problems Encountered

Many problems were faced from designing to actual implementation. We will discuss each of them briefly.

6.1. Dataset

Getting relevant dataset was one of the early challenges that we faced while developing our application. First we looked into a rich and latest dataset MS-Celeb-1M for testing and training purpose. First challenge was to download the dataset which is huge (90 GB compressed). As it was not feasible to store that much of data in either of the machine and process it, we then tried to look into cropped version of it. The problem was that, it was unlabeled dataset and needs processing. So we moved onto a simpler dataset Labeled Faces in Wild. We looked into first the normal version of the grayscale images. We decided on the grayscale as they were computationally inexpensive than 3 channel or more based images. Even if the images had faces centrally aligned still we didn't get good results on them. So we moved to the current cropped version of them. The file .pgm was also no

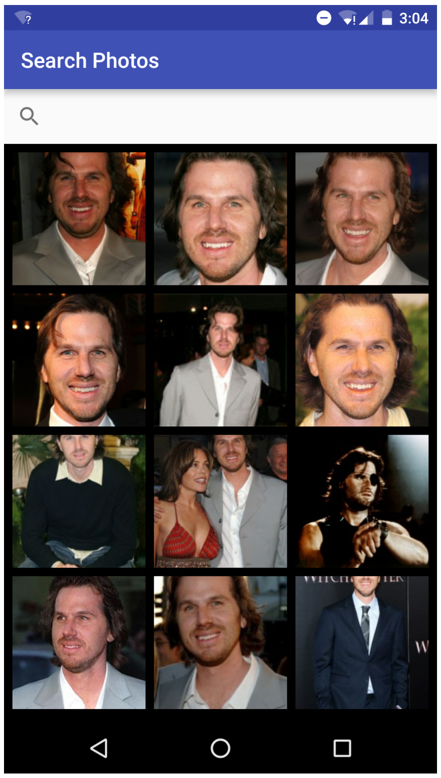


Figure 1. Desired Implementation for the application

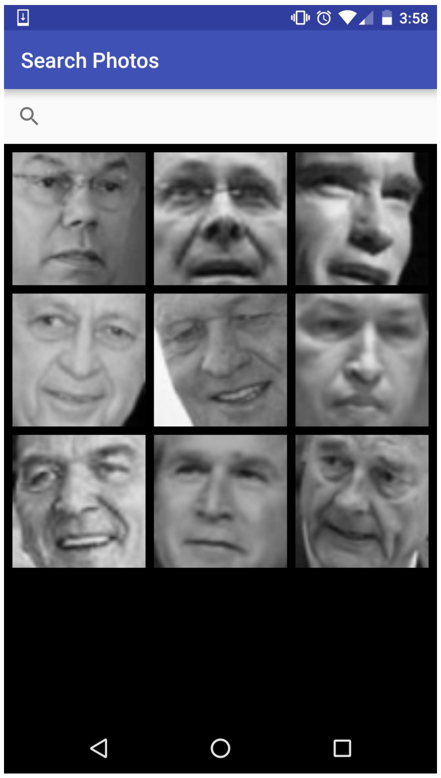


Figure 2. Actual Implementation for the application

less trouble as it is not prevalent form of image file format. The major problem due to was on the Mobile device which we will discuss later. Next we discuss the training problems.

6.2. Training the Model

Next challenge we faced was training the model. The FaceNet NN1 model given in the paper didn't have batch normalization and layer normalization. Earlier we tried training with given model and found that training it was really susceptible to random initialization done in the beginning of training. Sometimes the model was easier to train and went upto 75% training and 40% testing error but sometime after the same no of iterations regressed back to 10% training error 4% testing. This sway in accuracy we believed was due to random initialization. Also the cost function for the model was in the order of 10e8 which was not reasonable. Then we added layer normalization[8] to remove the problem. It did well and gave consistent output to an extend but it only guaranteed close to 60% accuracy on training set. Problem also was that after going through the literature on layer normalization we noticed that it doesn't help with convolution layer which were dominant part of our network. Finally we went ahead and added batch normalization on each layer of convolution. Though it has increased our training time by 1.5x but it has greatly reduced the no. of epochs to get good result. It took 200 epochs to reach an accuracy of 75% on training set. We are still looking into whether it has affected our inference delay or not. But after that we were able to attain a consistent training on the model.

6.3. Training on device

We looked on various methods which can be used to develop framework to train network on the device itself. Two of the approaches were, porting Tensorflow C++ API to Android using Java Native Environment and secondly by using a loosely bundled GO API used in Java for providing Tensorflow support. We looked into Tensorflow C++ API, but due to complexity involved in providing the methods to framework and testing with supported devices would have taken a lot of time. Also we weren't sure how feasible was it to port as the demands of both platforms are severely different. So we looked into GO API based implementation. Here also as the library itself wasn't well supported by the Tensorflow community, chances of making it work diminished. So for now until a port for Android is not here, I would say its relatively difficult to train efficiently on the device. Apart of Tensorflow, ARM is also developing Compute Library for Neural Network but currently it supports very few feature and layers so its impractical to port something complex as FaceNet on it.

6.4. Inference on device

Running computationally expensive operations on mobile device is always a challenge and to alleviate the problem concepts like Cloud computing and Fog computing have emerged successfully. But as the trends are changing, smartphones are becoming more capable with more no of cores and increasing on chip GPU complexity. It has now become more reasonable to do computing on smartphone rather than offloading to cloud resources. We also tried the approach by running Tensorflow inference JNI API provided by Google for Android devices. Though training is far but we can now easily run inferences on the device. Though running deep models are still something to look into. Our network which is about 22 layer deep was giving a variable delay of 150 ms - 300 ms. This variability could be due to process scheduler and other background application and services but still we feel that inference delay wasn't bad. It was able to infer images but mostly incorrect due to poor training but got two of them right in nine samples. I believe it relates to reducing the training error by training it a bit more on the dataset could yield better results.

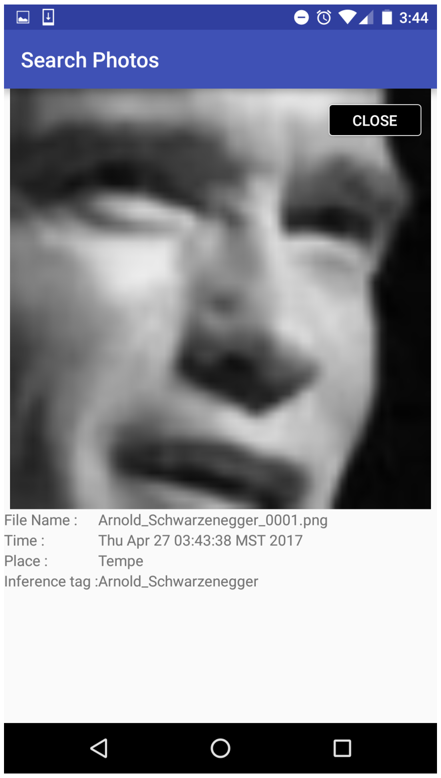
7. Assumption

There are various assumptions which should be taken into consideration because this proposed framework might not work very well in certain circumstances. Due to lack of sophisticated infrastructures, the FaceNet architecture might not be very much accurate. The FaceNet architecture is a deep network which uses 6 convolutional layers ,as deep architectures tend to increase the accuracy. Our current implementation is 22 layer deep. Assumptions are that datasets and folders already exist on the device. The inference delay due to model is acceptable on smartphone. Regarding the dataset we are assuming that the dataset is a 64x64 PNG format grayscale image. Other formats are not supported by the model.

8. Results

The result presented here are based on the FaceNet model trained to 75% training and 39% testing accuracy. As our whole application in an Android application we will use screenshots to explain our results. Firstly, the Figure 2 earlier presented is the main activity of our application. As you can see it has nine images from testing set each from different nine classes. Now after inference is done on the images we get updates on inference tags as the inferred class name. So when you try to open the fullscreen view by clicking on any of then Images in grid view you get Figure 3 & 4. Here in Figure 3 we can see that the Tensorflow graph model inferred it correctly even though the image is at slight angle. Now looking at Figure 4 the model didn't get it correct even though the face is close to center. One reason that was observed was that training set had unequal no of images under each class. It had more no of images under George W Bush but not so many on Arnold Schwarzenegger. So, we can infer that class of George W Bush got over fit compared to Arnold. Apart from that we cannot present with any better explanation on the observation.

Next we discuss our result for search feature. Looking at Figure 5 we can say that it works. The search is based on making hits from nouns in the search bar and generates a sorted list of images based on the inference, place, time & filename tags. This is depicted with Arnold Schwarzenegger at the top of the list with two hits from filename and inference tag. Other Images are there due to hits from Inference tag alloted due to incorrect inferences from the model.



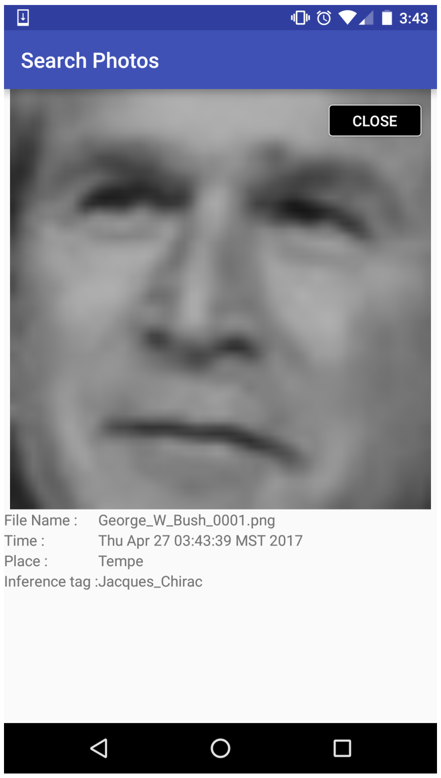
File Name : Arnold_Schwarzenegger_0001.png
Time : Thu Apr 27 03:43:38 MST 2017
Place : Tempe
Inference tag :Arnold_Schwarzenegger

Figure 3. Positive Inference from the application

9. Discussion

We can discuss that inference tags are the result of the training done and model itself. It also seems that training is also correlated to no of training samples and also on division on samples. Classes with more samples tend to overfit whereas classes with less samples tend to underfit. Problem can be fixed by providing equal no of samples for each of the class and by generating more samples.

Earlier we planned to do training and inference on the smartphone device itself but due to challenges posed in porting Tensorflow to mobile device and severe lack of processing capabilities of smartphone we are relieving the idea to only do inference on the device. The training part was done mostly on a PC and Cloud based computing resources with nominal changes to the model. For now as the Tensorflow API for android devices is not available we will stick with this implementation.



File Name : George_W_Bush_0001.png
Time : Thu Apr 27 03:43:39 MST 2017
Place : Tempe
Inference tag :Jacques_Chirac

Figure 4. Negative Inference from the application

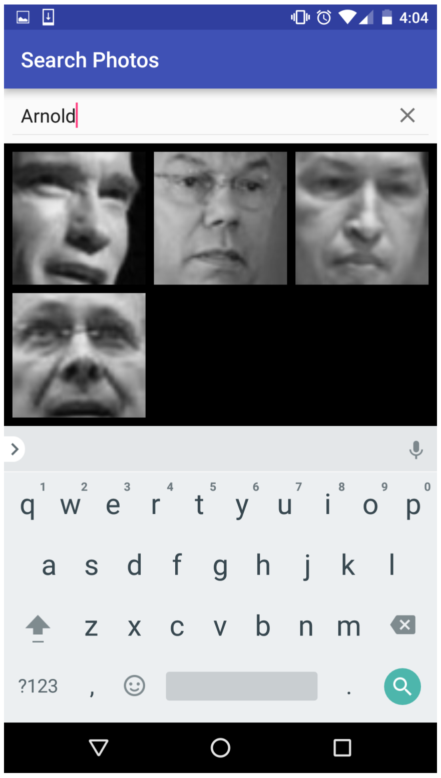


Figure 5. Search in action from the application

Regarding the dataset, we are using cropped grayscale images of size 64x64 for inference purpose. We do understand the actual images for camera are of higher resolution and have more than one channel but due to memory requirements and speeding up the inference we are going with single channel cropped images. We were also looking into generating samples by introduction noises, transforms and generating images from network like GAN. Though creating samples using noises and transform is feasible but due to lack of time we didn't proceed in the direction. Training using GAN generated images would have become a project of its own due to involved complexity, so we refrained to proceed with it. Also as face image data are pretty complex, without a very good implementation of the GAN we cannot generated good dataset.

Also regarding retraining the model for a custom dataset, it can be done by retraining the final layers of the network. Though the dataset is fairly complex but we believe that if there is a significant correlation between the sets than custom images can also be trained though it has not been tested. But all the retraining had to be done on the PC or cloud resources by unfreezing the graph. Also adding more classes or incremental learning could be done but its currently it not supported in Tensorflow so we aren't trying the approach though it was earlier mentioned in our tentative work schedule. Much research has to be done to correctly utilize the approach.

10. Conclusion

We conclude by saying that its fairly possible to port deep neural network to Mobile device for inference related work. The performance on the device will vary with hardware, the complexity of graph and dataset but with nominal delay we can make good prediction using them. We believe that the field of embedded vision is exciting field where many more possibilities and applications have yet to be discovered. It is the most intrusive form of computing discovered and have the potential to be used by billions of customers. With more advancement in technology and deep learning models we can see more empowered ideas emerging.

11. Future Work

As for the future work improving the inference model takes priority as good inference will yield accurate tags. Also we should look into more generic dataset first from normal LFW dataset and then to MS-Celeb-1M dataset by adding cropping and gray scaling pipelines before inference. Also we should look into normal RGB formatted images rather than grayscale and check if it adds useful features to the model.

Other than that the indirect inferences and places should also be added to search in order to get more refined searches.It can add to more contexts to become closer to natural language processing. Though field of it own can be used to integrate voice based searches etc.

We can also look into porting of Tensorflow C++ API, not sure about the feasibility to run it but if running on new version of on chip GPUs, it can give better performance. Though its currently active region of research and development in the embedded industry.

12. Contribution

The following table presents the contributions made to the project.

TABLE 1. Work Division

S.no	Task	Assignee	Status
1	Basic Android App Layout	Vishal	Completed
2	Facenet on Tensorflow	Vishal	Completed
3	Demo test Tensorflow Android	Vishal	Completed
4	Image Feature Extraction	Vishal	Completed
5	Context based Image tagging	Vishal	Completed
6	Improvements to Model	Vishal	Completed
7	Search Algorithm	Vishal	Completed
8	Implement on Android Tensorflow	Vishal	Completed
9	Testing and Validation	Vishal	Completed
10	Squeezenet Implementation	Rajagopaalan	Completed
11	Testing with real dataset	Vishal	Completed
12	Incremental Learning	Rajagopaalan	Abandoned

References

[1] Dave et al., *Face Recognition in Mobile Phones*, Standford University, Jun. 7, 2010.

[2] Dave et al., *FaceNet: A Unified Embedding for Face Recognition and Clustering*, Google Inc, Jun. 17, 2015.

[3] Bengio et al., *BinaryConnect: Training Deep Neural Networks with binary weights during propagations*, arXiv:1602.02830, Mar. 17, 2016.

[4] Rastegari et al., *XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks*, Computer Vision âĂŞ ECCV 2016, 2016

[5] Han et al., *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*, arXiv:1510.00149, Oct. 01, 2015

[6] Iandola et al., *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5MB model size*, arXiv:1602.07360, Feb. 24, 2016

[7] Sergey Ioffe, Christian Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, arXiv:1502.03167, Feb. 11, 2015

[8] Hinton et al., *Layer Normalization*, arXiv:1607.06450, Jul. 21, 2016