Real-time Smile Recognition using Convolutional Neural Networks

Students: Nina Kirakosyan, Lilit Ghandilyan

Supervisor: Artur Kobelyan

A Capstone project report submitted in fulfillment of
the requirements for the degree of
Bachelor of Computer Science

College of Science and Engineering
American University of Armenia

May 30, 2021

Contents

List of Figures

List of Tables

Chapter 1

Acknowledgements

We want to express our sincere gratitude to our supervisor, Artur Kobelyan, for his outstanding support and encouragement throughout the entire work process.

Completing our Capstone project would not have been possible without the resources necessary for the computationally expensive training and extensive data storage that company Krisp generously provided for us. We are most thankful for their contribution and support.

We would also like to thank Rima Shahbazyan, a fellow student and ML enthusiast, for her help during the work on this project.

Chapter 2

Abstract

The smile recognition task has many applications in human-to-human and human-computer interaction analysis. Although many smile recognition methods have been proposed, the real-time smile recognition task for average resource devices remains a relevant research area. With our Capstone project, we propose a combined face detection and smile classification pipeline. We conducted a performance investigation for appropriate face detectors, trained widely used CNN networks using transfer learning, and built and trained a custom network for smile classification. The proposed model shows 92.7-94.5% smile classification accuracy on test data. The final inference pipeline has real-time CPU performance with 28.43 FPS.

Keywords: Smile Recognition, Real-Time, Deep Learning, Convolutional Neural Networks (CNN), Transfer Learning, Face Detection, CNN Classification

Chapter 3

Introduction

The human smile is the most common but, at the same time, essential and communicative facial expression. It is an indicator for a wide range of emotions, such as happiness, joy, content, and satisfaction. Detection and analysis of the human smile from image and video stream data of human-to-human and human-computer interactions have many applications. The smile detection task is relevant in human behavior analysis, photo editing, customer satisfaction analysis, product rating, patient monitoring, and many other areas. Through smile recognition, inference about the subject's emotional state can be made, opening many opportunities in human psychology research, user experience improvement, and many more. Therefore, the detection problem is a specific important direction in facial expression and emotion recognition analysis.

The wide range of its applications has made smile detection a popular yet challenging research topic, which we built upon in the scope of our Capstone project. There is extensive work done among the academic community already. Many approaches to smile detection using geometric feature extraction and, more recently, utilization of deep learning techniques have been presented, with a wide range of accuracy and overall inference time performance. The appropriate data for smile recognition research has also been developed and improved throughout the years. Our project focuses on the smile recognition task using Convolutional Neural Networks (CNNs) specifically, using a combination of smile and emotion labeled datasets.

We present a pipeline made of a face detector and a smile classifier, which covers all the cases of the smile recognition task, such as having no faces or multiple faces in the image. To do so, first an investigation of the performance of various classical and novel face detection methods is presented, justifying the face detector choice for the proposed pipeline. We present the set of experiments and their results of searching the best classifier for the pipeline as well. Transfer learning with different successful CNN architectures, a custom CNN model and their performances are compared based on their inference latency and accuracy. As a final result, we propose a smile recognition pipeline for video stream data that shows real-time performance on an average consumer device under the given parameters.

Chapter 4

Related Work

Investigating academic work in facial expression recognition, we can outline several approaches that have been commonly employed. Early work in this area often relied on recognizing specific facial attributes described by the facial action coding system (FACS) developed by Paul Ekman. It relied on outlined facial action units such as Lip Corner Puller, Cheek Puffer, Dimpler, Cheek Raiser, and others [1]. There is research done in using the extracted FACS-based attributes to classify the facial expression based on action units different settings and positioning [2, 3]. The FACS attribute analysis for expression recognition using SVM classifier and different boosting techniques have been investigated by [3, 4, 5, 6]. Extreme Vector Machines (EVM) on holistic flow-based face registration were introduced for smile detection by [7], with GENKI-4k accuracy of around 88%.

Another studied approach is the application of alternative features such as Gabor Energy Filters (GBE), and Multi-scale Gaussian Derivatives (MGD), which was presented by [5, 6]. Smile detection using GBE and MSGD showed accuracy at 90.78% and 92.97%, respectively, tested on GENKI-4k dataset [6]. Smile detection through boosting pixel intensity differences combined with Adaboost classifier was proposed by [8], with 85-88% accuracy on GENKI-4k.

In recent years, the rapid rise of deep learning applications, reemergence, and evident high efficiency of convolutional neural networks for image classification opened a whole new direction of research in facial expression analysis and smile recognition in particular. Deep learning was used for smile recognition using CNNs with comprehensive architectural parameter selection by [9]. In [10], simultaneous feature learning and smile detection with a deep convolutional neural network combined with SVM were proposed, with around 92% accuracy on GENKI-4k. Modification of deep convolutional network architecture VGG with several data augmentation techniques showed around 95% accuracy on GENKI-4k in [11] and [12]. A combination of face and mouth-based CNNs for smile detection was investigated by [13]. Fully convolutional neural network utilization for face detection and analysis without image preprocessing was discussed by [14]. Several CNN architectures for real-time smile detection were tested, and a tuned model was proposed by [15]. Their proposed model had 93.6% GENKI-4k and 88.5% CelebA accuracy, with combined inference time of the face localization and smile detection of around 40-60 frames per second (FPS) on K40 Desktop GPU.

Although high accuracy models have been proposed for the smile detection task,

and some research has been done for real-time application of CNN-based models, the issue of high accuracy combined with real-time inference ability still poses a challenge and opens space for further investigation, which we tried to contribute to with our work. Most proposed models do not cover real-life cases, such as having no face in the image or having multiple faces.

Chapter 5

Background

5.1 Introduction to Neural Networks

The building blocks of neural networks are single neurons, which perform the operation $z = w^T x + b$, where w is the weight vector, x is the input vector, and b is the bias term. The output z is a scalar. A neural network layer is a stack of such neurons, where each neuron performs the same operation with different weights. The output of these neurons is then fed to the next layer as input. One can stack arbitrarily many such layers, and the output of the network would be the output of the last layer.

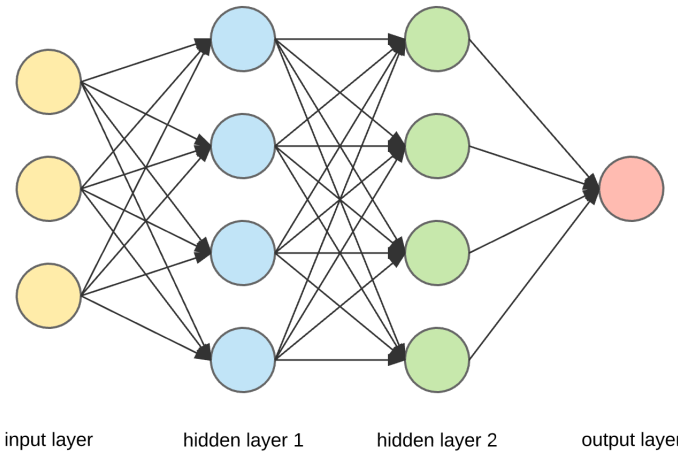


Figure 5.1: Multilayer neural network [16]

However, if $w^T x$ is the only operation that is done, then the whole neural network is simply a linear function. Introducing nonlinearities (activation functions) enables the network to approximate complex functions instead. Some of the most widely used activation functions that have proven to be effective in practice include Sigmoid, Hyperbolic Tangent, and the Rectified Linear Unit (ReLU). ReLU has become the default activation function due to its ability to avoid the vanishing gradient problem [17].

$$\mathrm{ReLU}(x) = \max(x, 0).$$

A neural network can be thought of as a complex function, an approximation of the real-world function that outputs the ground truth for the given problem. Training a neural network is the process of finding the parameters which would approximate

the desired function best. To do so, we define a loss function, which is a measure of how bad the network is performing. Some of the most effective and widely used loss functions are Mean Squared Error for Regression and Cross-Entropy for Classification tasks. As our task comes down to classifying faces as smile and no-smile, we used cross-entropy loss:

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{n} y_i \log \hat{y}_i.$$

We minimize the loss function by changing the weights iteratively using the Gradient Descent optimization algorithm or one of its many variations. The most widely used optimization algorithms include RMSProp, and Adam [18].

Another vital thing to pay attention to when building neural networks is the final networks generalizability. The network can "memorize" the data it has seen during training to improve the loss function while not working well on unseen data (overfitting). To measure the performance of the network and the learned weights on unseen data, we set aside a portion of the data for testing [17].

To improve the generalizability of the network, regularization techniques are applied, such as L2 regularization or Dropout. In L2 regularization, we penalize large weights, which in turn reduces the complexity of the network. Dropout is a method of randomly zeroing the output of some neurons of the layer, resulting in a simpler network which generalizes well during inference [19].

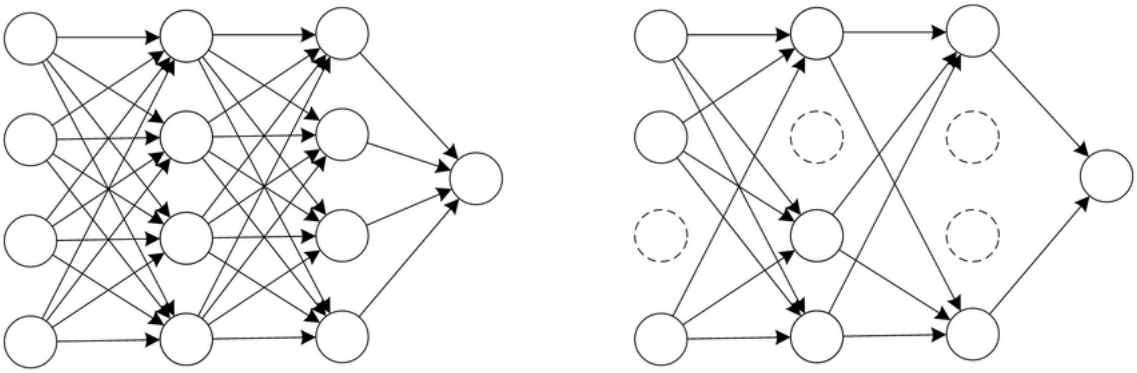


Figure 5.2: A Neural Network before and after Dropout [19]

5.2 Convolutional Neural Networks

For effectively applying deep learning methods to images, special kinds of neural networks, Convolutional Neural Networks (CNNs), have been developed. They preserve the spatial information integral to images, making them the default choice when working with images.

The main building block of CNNs are filters/kernels, which slide over the original image. These filters have the same depth as the original image but can have different sizes (3x3, 5x5 being the most common). As a kernel slides over an image, the corresponding values of the image and the kernel weights are multiplied and summed together. The computed sum, with the added bias term, produces one value in the output activation map. The interval at which a kernel slides over an image is called a

stride. The dimension of the resulting activation map is reduced depending on the chosen stride and kernel sizes. In practice, padding by zero is common to avoid shrinking the size of the outputs [20].

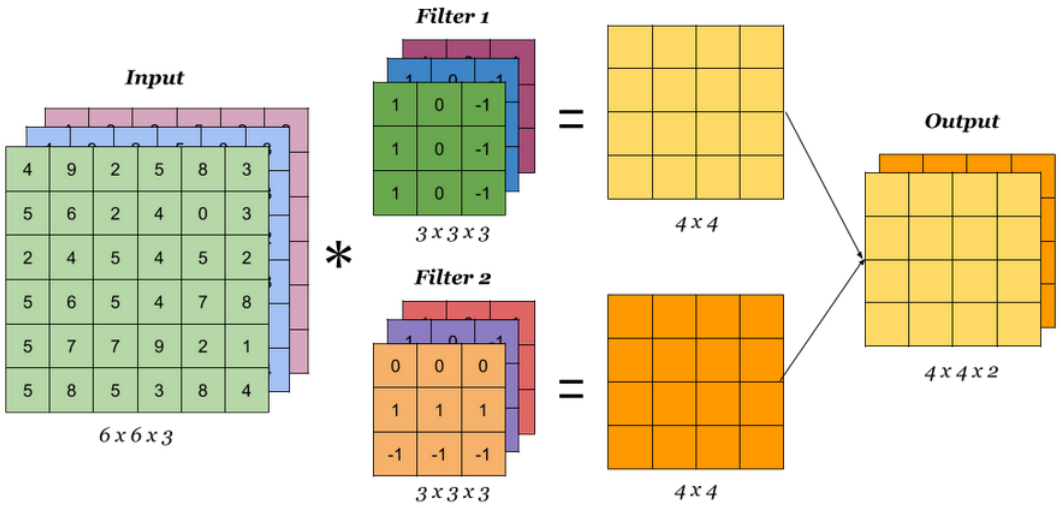


Figure 5.3: Convolutional filters/kernels producing activation maps [21]

A convolutional layer consists of multiple such filters, where each filter can be thought of as looking for a specific pattern within the image. Each filter results in one activation map, and these maps stacked on top of each other form the input for the next layer.

Poolng layers are used to downsample the output width and height. The two most common pooling algorithms are maximum and average pooling. Nowadays, average pooling can be seen rarely, as max-pooling has proven to give the best performance. Pooling is also done by kernels, which slide over the image with a given stride. In max pooling, for example, only the maximum value of the given region is preserved. Pooling kernels have no weights and no depth, so they don't affect the number of weights of the network and the depth of the output.

A Convolutional Neural Network is a sequence of convolutional layers, with activation functions and pooling layers in between. There is a hierarchical structure in these layers, such that the filters at the early layers look for simpler patterns, and the patterns become increasingly complex at the later layers [20].

At the end of the network, we usually have dense layers. The input of the first dense layer is commonly the flattened input of the last convolutional layer. However, the flattened activation maps can have huge dimensions, and the weight matrix of the dense layer that comes after the flattened layer can make the network significantly heavier. For this reason, Global Average Pooling is increasingly commonly used instead of flattening. This performs dimensionality reduction by keeping only the average of each activation map.

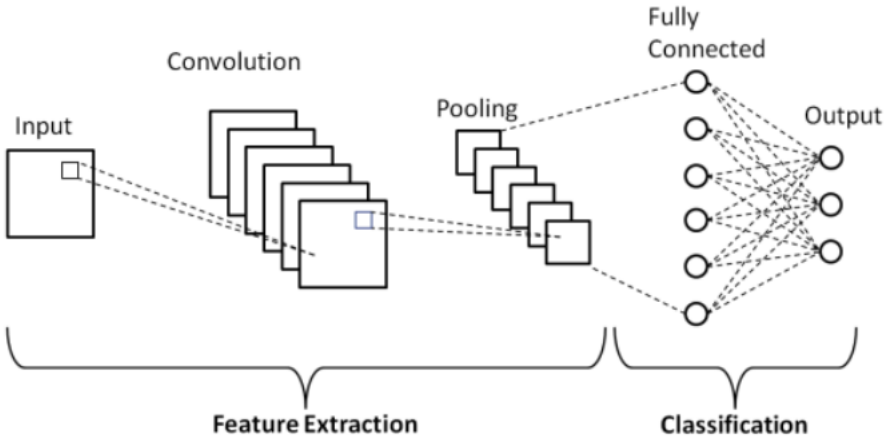


Figure 5.4: Convolutional Neural Network [21]

Precisely like the dense neural networks, convolutional neural networks are trained by gradient descent and its variations, where at each step we try to minimize the chosen loss function. In this case, the parameters we need to optimize are filter weights and biases. At each stage, we do forward propagation, compute the gradient with respect to the parameters and update the parameters by moving towards the negative gradient.

5.3 Transfer Learning and Fine Tuning

Transfer learning refers to the idea of transferring already acquired knowledge from one task to solve a different problem. Transfer learning is widely used in CNN-based computer vision tasks for several reasons. Firstly, training CNNs is computationally expensive, and utilizing already trained networks can cut down this cost. The second reason is the hierarchical feature extraction of CNN-s, where initial layers of the network learn low-level, generalizable features [22]. It is hard to find enough data for many specific problems to avoid overfitting when large networks are used. The networks on which transfer learning technique was tried in the scope of our project are described in the following subsections.

5.4 CNN Architectures

We present a small survey of the most widely used CNN architectures that have been proven to perform well for many problems. The presented networks were important during our experiments both for comparison as well as for providing insights to design the final architecture.

5.4.1 VGG

VGG, with its variants, is one of the most popular CNN architectures. Introduced by Simonyan and Zisserman in [23], it was one of the first very deep convolutional networks used for image classification. VGG is characterized by its architectural simplicity: using only 3x3 kernels, convolutional layers are stacked on top of each other in blocks

connected with the dimension-reducing max-pooling layers and followed by two fully connected layers. The full architecture of the network is shown in Fig 5.5.

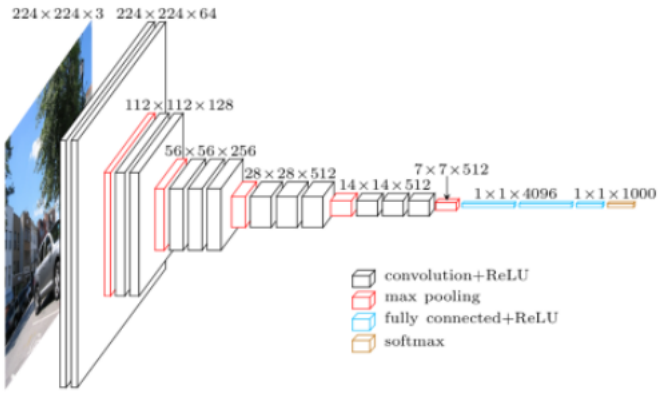


Figure 5.5: VGG16 netowrk architecture [24]

The use of layered 3x3 kernels allowed the network to reach the same receptive field as for larger 5x5 or 7x7 kernels, but with a significant reduction in the number of network parameters. The ReLU activations between layers allowed for more discriminative classifier training. It also showed that the network performance tends to improve with deeper architectures. However, the major drawback of VGG is its very large size and slow training. The 16-layer VGG has around 138 million parameters and performs 15.3 billion FLOPs (floating-point operations), making it not suitable for real-time image classification.

5.4.2 Inception V3

Inception V3 is the third edition of Google's Inception Convolutional Neural Network, introduced initially during the ImageNet Recognition Challenge [25]. The main idea behind it is combining different kernel sizes into one block to have a good-performing model regardless of the size of the object that should be recognized in the image. Inception V3 introduced a more efficient variant of the Inception module with the same receptive field, where 5x5 kernels are replaced with two 3x3 kernels, and the ones with size 3x3 are replaced with a kernel 1x3 followed by 3x1.

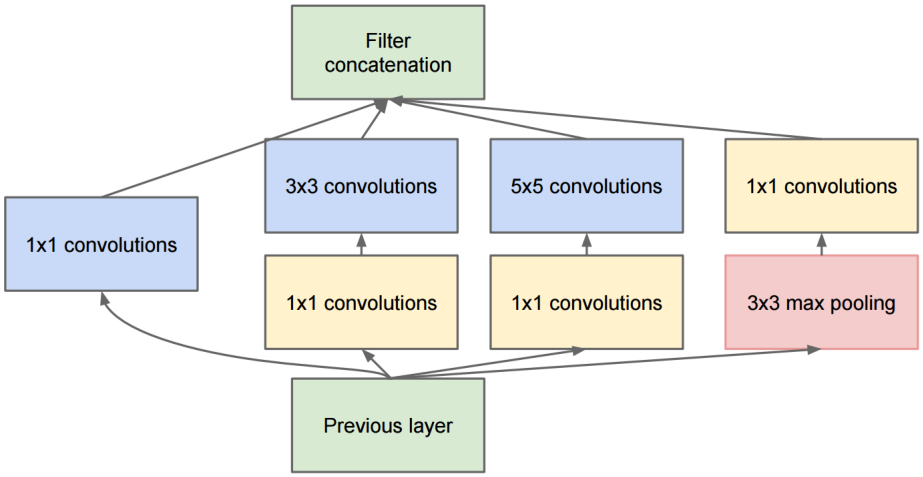


Figure 5.6: Optimized Inception V3 module [25]

5.4.3 EfficientNet

The primary architectural hyperparameters of neural networks that have been shown to affect the performance greatly are the networks depth, width, and the resolution of the input images. Authors of [26] introduced a CNN scaling method for a uniform scaling of those parameters using a compound scaling coefficient.

The authors showed that the effects of scaling of CNN's width, depth, and resolution on performance, in fact, are not independent. They show different performances under different combinations of constraints. As the scaling of any of the said parameters increases the computational resource and memory requirements, an optimization problem was formulated for finding the optimal scaling parameters under constrained resource conditions. Particularly, for the best configuration of the baseline model (taken as a MobileNet inspired model by the authors), allowing the use of 2^N times more computational resources, the scaling of depth by α^N, width by β^N, and image size by γ^N was considered. So, for doubled resources, the optimization problem was formulated as:

$$
\begin{aligned}
depth : d &= \alpha^{\phi} \\
width : w &= \beta^{\phi} \\
resolution : r &= \gamma^{\phi} \\
s.t. \alpha \cdot \beta^2 \cdot \gamma^2 &\approx 2 \\
\alpha \geq 1, \beta \geq 1, \gamma &\geq 1
\end{aligned}
$$

Here ϕ is the compound coefficient controlling the allowed increase in resources. The authors showed that the FLOPS (floating-point operations per second) of the network are proportional to d, w^2, r^2, where d is its depth, w is the width, and r is the resolution. Therefore, the $\alpha * \beta^2 * \gamma^2 \approx 2$ condition constraints increase the number of FLOPS by approximately 2^{ϕ}. The optimal values for α, β, γ were found through grid-search.

5.4.4 MNasNet

Designing a CNN for resource-constrained platforms such as mobile devices is challenging, as both the accuracy and latency should be optimized. To do this, the MNasNet authors [27] use the technique of Network Architecture Search for mobile devices. One of the main ideas introduced by the authors is using a multi-objective reward function for finding the best mobile architecture, a tradeoff between accuracy and inference latency.

Secondly, the authors use the actual inference time of the network instead of using the FLOPS as an approximation. The authors also emphasize that NAS is often used only to find a network block stacked on top of each other. Searching for a block reduces the search space, but layer diversity is lost. An approach to give more flexibility for layer diversity is introduced, which does not increase the search space significantly. MNasNet was shown to have better performance than the MobileNet V2 and comparable performance to Mobilenet V3.

5.4.5 MobileNet

MobileNets are specifically designed to be used in low-resource platforms. Thus, they are comparatively light and latency optimized networks. So far, the authors have introduced three MobileNet versions, which present a host of ideas for optimizing the network size and performance.

The initial MobileNet [28] introduced the depth-wise separable convolutions. Instead of applying a regular convolution, two steps are executed: first, the depthwise convolution is performed, then the pointwise. In the depthwise convolution, we have as many kernels as the number of the current activation maps. On each activation map, one convolutional kernel is applied. The result of each depthwise convolution is then combined by a 1x1 regular (pointwise) convolution. This change reduces the number of operations in each convolutional layer by around nine times.

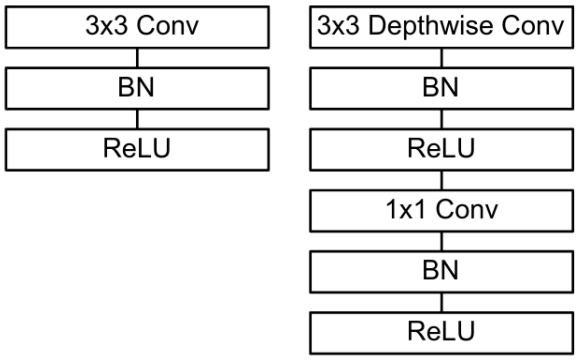


Figure 5.7: Regular 3x3 convolutional block, and MobileNet V1 depthwise separable convolutional block [28]

The second version of MobileNet [29] also uses depth-wise separable convolutions but in a different manner. Here, there are two types of pointwise convolutions: one for expansion and one for projection. In the original MobileNet, a depthwise convolution

was only used to leave the same or increase the number of activation maps. However, the projection layer does the opposite; it reduces the network output to a lower dimension to increase it later with expansion layers. The idea behind the projection and expansion layers is to keep the tensors that are passed along in comparatively small sizes but do the convolutional operations in a higher-dimensional space to improve the power of the network. Residual connections are used between blocks to help with the flow of the gradients.

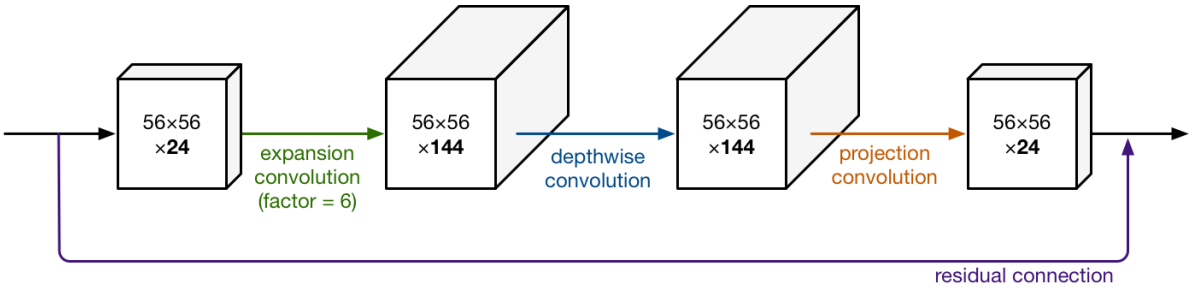


Figure 5.8: MobileNet V2 Inverted Residual block [29]

Mobilenet V3 [30] brings many of the ideas of the previous versions together. The Network Architecture Search technique was used to choose the core building module of the network, building on the ideas of the previous versions. In simple terms, an RNN is trained to find the optimal network architecture for optimizing the reward function, which is a tradeoff between accuracy and latency. Besides the NAS blocks, other changes were done manually in the first and last layers to improve latency. Swish nonlinearity was also introduced, which has been proven to work well experimentally. The authors introduced Mobilenet V3 Small and Mobilenet V3 Large, which only differ in the number of bottleneck blocks (expansion, convolution, projection).

5.4.6 U-Net

The original U-Net Architecture was designed for image segmentation tasks [31], and it produces output which has the same dimension as the input, with corresponding labels for each pixel. The architecture has U shape, where the first path called contraction is made of regular convolutional layers, while the expansive path is made of transposed convolutions.

The architecture is symmetrical in both paths. In the contraction path, there are four blocks, each made up of two Convolutional layers, Max Pooling and Dropout. As our task is classification, only the contraction path was used as an inspiration for the final model.

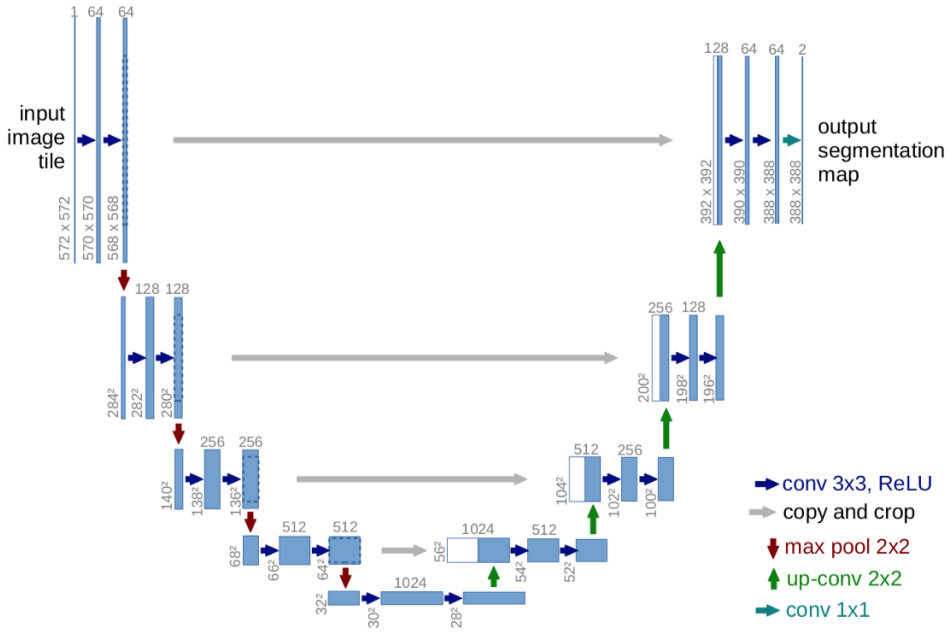


Figure 5.9: U-Net architecture [31]

5.5 Face Detection

We looked into several most popular face detectors to find the best detector for our proposed pipeline - Haar Cascades, HOG, and several CNN-based detectors.

5.5.1 Haar Cascades Classifier

Haar Cascades-based OpenCV Viola-Jones frontal face detection classifier developed by [32] is usually the default choice due to its simplicity and readily available OpenCV API [33]. This detector uses Haar features (Fig. 5.10), which detect lines, edges, and important pixel intensity change patterns in the image.

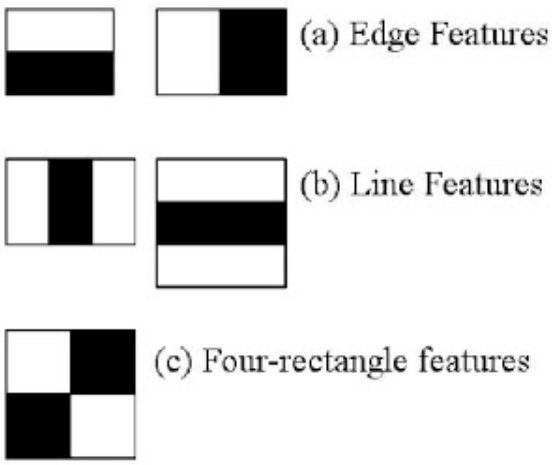


Figure 5.10: Haar features' example [33]

The set of predefined Haar features representing facial attributes are used for face detection. The features are used as kernels that slide over the image, and for each portion of the image, the sum of pixels under the white portion of the kernel is subtracted from the sum of black area pixels. The feature is detected if the calculated difference is close to 1. For computational efficiency, the different features are grouped at hierarchical stages (cascades) and are applied sequentially - the later stage features are applied only if the higher stage features were detected.

5.5.2 Histogram of Oriented Gradients (HOG)

The Python Dlib package was used to utilize HOG-based face detection [34]. The HOG descriptors are the calculated gradients of small blocks of pixels within the image. The gradient vector in this configuration depends on the magnitude and direction of the pixel intensity change within the given block (Fig. 5.11). The extracted features are then passed through a linear SVM classifier to detect faces. Like Haar-Cascades, this classifier, however, works well for frontal face detection only.

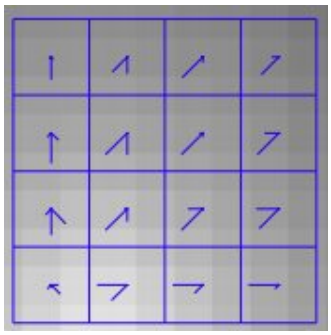


Figure 5.11: Gradients used by the HOG algorithm [35]

5.5.3 MTCNN

Authors of [36] proposed a deep-cascaded convolutional neural network for detecting faces and facial landmarks. The implementation of the proposed algorithm was given in Python, which we tested for our face detection pipeline. The new detector has high accuracy but slow performance.

5.5.4 YOLO

YOLO (You Only Look Once) is a state-of-the-art object detection model, first introduced by [37]. Unlike classical detectors, where the detector is applied at different parts of the image at different scales, the YOLO detector only processes the single image through the neural network once. The network then detects the bounding boxes and probabilities of classification for different regions on the image. Evaluation of a single image offers a state-of-the-art inference time. The Python implementation of the small version of YOLO V3 by [38] was tested for our pipeline.

5.5.5 Google MediaPipe Face Detector

The best performance was shown by Google MediaPipes Face Detector [39]. It is based on a lightweight BlazeFace detector introduced by [40], which uses MobileNet inspired feature extraction with Single Shot MultiBox Detector (SSD), providing accurate facial region detection at state-of-the-art inference time, especially well suited for mobile devices.

Chapter 6

Datasets

Good quality representative data is the key to a good model. There are several open-source, as well as academic research intended datasets that are used for the emotion and smile detection tasks. These include GENKI-4k, CelebA, AffectNet, MTFL, UCF Selfie, and others. The primary datasets used for training and evaluation for our models were CelebA, AffectNet, and GENKI-4k datasets. For our research, CelebA and AffectNet were used for model training, and a portion of CelebA and GENKI-4k were used for testing purposes. The total number of images, as well as the number of smiling and non-smiling facial images in each dataset, is summarized in Table 6.1.

Dataset	Smile	No Smile	Total
GENKI-4k	2,162	1,838	4,000
CelebA	97,669	104,930	202,599
AffectNet	115,385	170,273	285,658

Table 6.1: Summary of datasets

6.1 GENKI-4k Dataset

The GENKI-4k dataset is a popular benchmark dataset, as evident from reported performances in related works. It contains 4K images of faces of a wide range of facial positions, lighting conditions, backgrounds, including people of different ages and ethnicities. The images come labeled with expressions (smile, no-smile) and head-pose labels [41, 42]. Some examples of the GENKI-4k images are illustrated in Fig. 6.1.

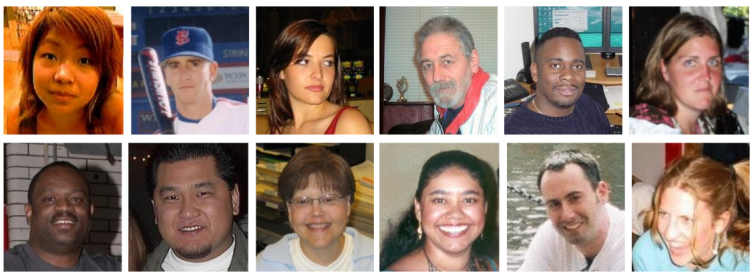


Figure 6.1: Example images from GENKI-4k Dataset

6.2 CelebA Dataset

CelebFaces Attributes Dataset (CelebA) is a large-scale face attributes dataset of celebrity images [43]. It contains more than 200K images, covering over 10K identities, with a wide range of poses and backgrounds. The images have more than 40 total annotations, of which the binary Smiling annotations were utilized for our research purposes. See samples of CelebA images and annotation in Fig. 6.2.

Figure 6.2: Example images from CelebA Dataset

6.3 AffectNet Dataset

AffectNet is a research dataset of facial expressions in the wild [44]. The around 440K images are manually annotated for the presence of seven universal facial expressions and their valence and arousal. The annotated expression categories include happy, sad, surprise, fear, disgust, anger, contempt, and neutral/none. We used the manually annotated images containing faces for our research and classified them into happy/smiling and non-smiling categories.

Figure 6.3: Example images from AffectNet Dataset

Chapter 7

Our Approach

For our problem of real-time smile recognition, we chose the approach of dividing the inference pipeline into two main stages: face localization from the input video-stream data and smile/no smile classification for the detected faces. By applying this approach, we make sure that we are handling cases such as having multiple faces in the photo as well as no faces in the photo. The face detection algorithm would return detected faces, while the CNN trained by us would only focus on classifying the face as smiling or non-smiling.

Thus, we organized our research and experiments through separable sub-problem investigations within the scope of the outlined approach. We investigated applicable face detection methods. To prepare the training data for the classifier, we performed different data augmentation techniques both before and after face detection, as described in the augmentation section. The available data was divided to train-validation-test splits, and the augmentations were only performed on train and validation splits. The test split was only face detected. The smile/no smile classification models were then trained and evaluated.

We used Python, and Tensorflow platform for our project implementation, data augmentation, training the available, and custom CNNs [45, 46]. We also used GitHub for group collaboration purposes.

7.1 Face Detector Selection

As the first stage of video frame processing in our adapted approach is face localization and extraction for further smile classification, the selection of the most accurate and fastest face detector was of great importance. We researched and selected several most applicable face detection methods that could be used in our final inference pipeline. The face detectors presented in Section 3.4 were considered - Haar Cascades, HOG, MTCNN, YOLO, and Media Pipe.

7.1.1 Face Detector Evaluation

The performance comparison was made considering the number of frames processed per second (FPS) by the face detector. The calculated FPS (frames per second) and average SPF (seconds per frame) of the above-listed detectors is presented in Table

7.1. As the goal was a real-time performance on an average consumer CPU, the evaluation was done without GPU utilization, using Intel(R) Core(TM) i7-8565U CPU @ 1.80 GHz 1.99 GHz processor with 8 GB LPDDR3 RAM. The inference times were evaluated for 350x350 RGB images.

Note that the Haar Classifier was configured with a scale factor of 1.3 and min neighbors at 6, and its performance greatly depends on the size of the input image; therefore, the most compatible configuration was identified and chosen for comparison.

Detector	FPS	SPF
Haar	118.292	0.008
Media Pipe	72.830	0.014
YOLO	40.240	0.025
Dlib HOG	12.170	0.082
MTCNN	2.515	0.398

Table 7.1: Face detector performance evaluations

From the fastest performing face-detectors, Haar and Media Pipe, we chose Google's Media Pipe detector as the detector for the final system. The performance of the Haar cascade is highly dependent on the set configurations, making it less flexible, while the Media Pipe detector showed consistently fast and accurate performance. As reported by Media Pipes base model BlazeFace detector paper authors, the detector has around 98.6% average precision [40]. As observed during demo trials, the Media Pipe detectors accuracy was considerably higher than Haar, and in practice, for different test videos with varying sizes of frames, the Media Pipe detector outperformed Haar speed-wise as well.

7.2 CNN classifier selection

7.2.1 Data Preparation

Image augmentation is commonly used to expand the size of the dataset, and to make it more diverse and realistic. Deep learning models are heavily dependent both on the quantity and the quality of the training data, and data augmentation can often lead to better generalization [47]. Even though the datasets were chosen to be most representative, with varying facial poses, ethnic representations, and backgrounds, we further enhanced our training data using different data augmentation techniques.

Unlike data preparation, such as pixel scaling and image resizing, data augmentation is most commonly only applied to the training data but not to the test data. All augmentations were applied randomly with different probabilities. Augmentations both from Tensorflow, as well as more specific augmentations from the Albumentations [48] library were utilized.

The types of data augmentations we used can be divided into three categories: geometric, photometric, and kernel augmentations [47]. We have carefully considered both the type of augmentations and the intensity with which they are applied to ensure that the augmented images are both realistic and the resulting image preserves the label.

Most commonly, augmentations are used parallelly, and the combined augmentation space, instead of individual augmentation ranges needs to be examined. We used geometric augmentations before face detection and the others after face detection to best assimilate real-life data.

A common geometric augmentation type is flipping, which in our case was done only with respect to the vertical axis. Rotation augmentations were also used within the angle range differing from -15 to 15 degrees to the original. When the dataset (such as a face dataset) is perfectly centered, such augmentation also makes the dataset more real-life. Height and width shifts were applied within 0.1 fraction of the original image height and width to avoid positional bias in the data. Shearing was also applied with a 0.1 intensity range (shear angle in the counterclockwise direction in degrees). Zooming with a range of 0.1 was applied as well.

Image color space biases, such as lighting issues, are common problems in Computer Vision. Color space transformations have been shown to be very effective for addressing such issues [47]. We have used random brightness contrast change and hue, saturation value shifts. The color space transforms like the geometric transforms were also carefully chosen to not disregard information that the color carries. Noise injection was also applied, which consists of adding random values to the image drawn from the Gaussian distribution.

Other commonly used data augmentation techniques are blurring and sharpening. We have used motion and median blurring techniques, as well as classical sharpening techniques. Figure 7.1 below illustrates an example of the chosen transformations done over a single image.

Figure 7.1: Image augmentation results for one sample

With data augmentation, we increased the combined training dataset around four-fold. While the combination of CelebA and AffectNet datasets has around 500k images, with data augmentation, we were able to train our models on around two million im-

ages, resulting in more accurate and less prone to overfitting models. Initially, all training was done on the augmented CelebA dataset. However, our experiments showed that the models were prone to overfitting, and the AffectNet dataset was augmented and added as a training dataset.

7.2.2 Transfer Learning

We tried both transfer learning and training a CNN from scratch to find the best smile/no smile classifier in terms of inference latency and accuracy. The models were built and trained using the open-source TensorFlow library, with the Keras module for developing and training deep learning models [45, 46]. The CNN architectures presented in the Background section were used to find the best classifier via transfer learning. Initially, the focus was on finding a well-performing architectures and trying to improve their latency. Later, the focus was on finding the best classifier by using architectures that were designed for low resource inference, such as MNasNet or Mobilenet.

We retrained the pre-trained networks with two steps. First, the feature extraction layer was frozen, and only the newly added custom layers were trained until convergence. This was done to avoid random updates to the feature extraction layers, as the new layers are randomly initialized. Later, layers starting from a specific layer (decided by experimentation) were unfrozen to be trained in the following epochs. The network was trained with a lower learning rate until convergence. This was done to ensure that the convergence was reached, as before unfreezing, significant performance is already achieved, and the found minima are usually close.

For the models which were close to the inference latency goal, such as MobileNet, we also tried layer removal. We removed the Mobilenet layers block by block until the target latency was reached. However, the accuracy loss was significant. None of the models which were trained via transfer learning showed real-time performance combined with the face detector.

7.2.3 Proposed CNN

The final custom CNN was mainly inspired by the U-Net architecture presented in Section 5.4. However, it also incorporates ideas from other networks. The proposed model was found to work best under our task constraints and shows real-time performance as part of the final inference pipeline.

We built a custom network with similar consecutive two-convolutional layers, Max Pooling and Dropout blocks. We experimented with the number of filters in each layer, as well as the filter sizes. We also used Spatial Dropout instead of regular Dropout layers. The convolutions are followed by Global Average Pooling and a Dense layer with 128 nodes. The final architecture was found to work best given its accuracy and inference latency. The detailed final architecture can be found in Figure 7.2 below.

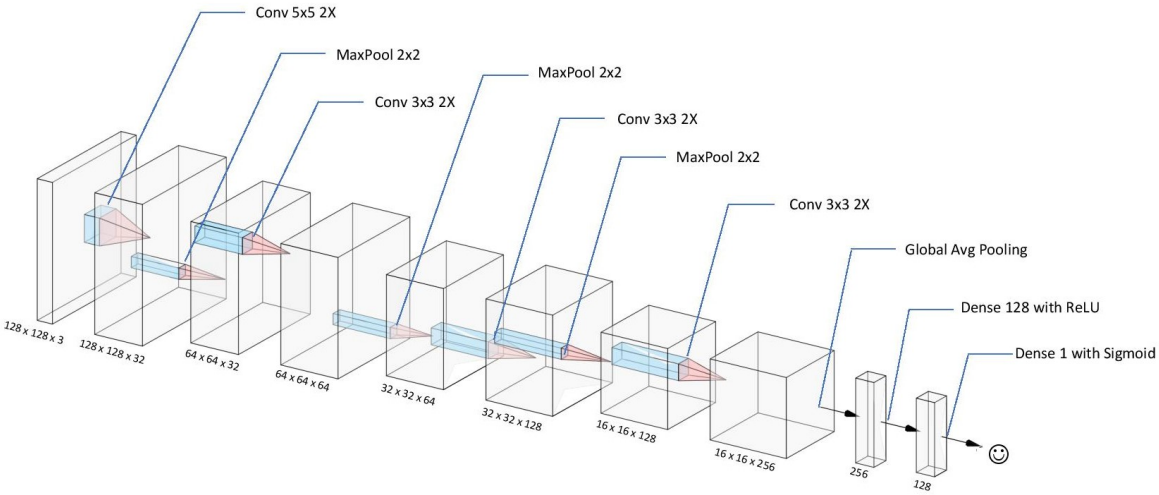


Figure 7.2: Proposed network architecture

As the input image goes through the network, the pixel values from range 0 to 255 are brought to -1 to 1 by basic scaling and shifting. The convolutional layers were initialized with He initialization introduced by [49].

All the convolutional layers were initialized with same padding, and the ReLU activation function was applied after all of them. The network was trained with Adam Optimizer, with an exponential weight decay value of 0.9 for the first moment and 0.999 for the second moment.

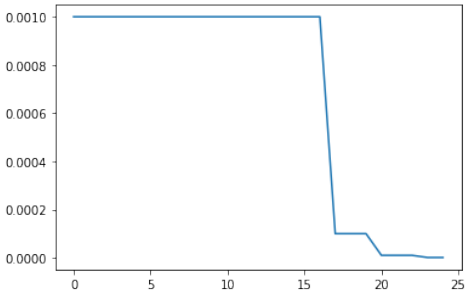
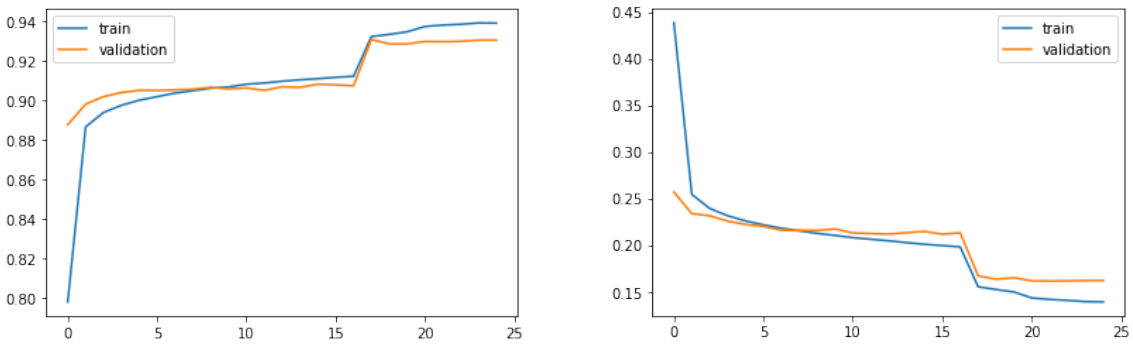


Figure 7.3: Proposed network training learning rate

The learning rate was reduced stepwise from $10e^{-4}$ to $10e^{-7}$ (Fig. 7.3), decreasing after convergence was reached at each step. Initially, the training was done on both AffectNet and CelebA datasets. However, considering the quality difference between the two datasets, the network was trained only on Celeba for the final epochs. The Figure 7.4 shows the accuracy and the loss of the final model during the training.



(a) Accuracy (b) Loss

Figure 7.4: Proposed network training

We found that the key to the performance of the proposed model is two factors: the 5x5 convolution in the first layer and the Spatial Dropout. The 5x5 convolution right after the input layer significantly increases the receptive field of the network in the initial convolutional stages. This is key as reaching the same receptive field through depth would be more expensive, while the initial kernels work on relatively small tensors.

	FLOPs (mln)	Parameters
5x5	39.321	2,400
3x3 consecutive	190.996	10,080

Table 7.2: 5x5 versus consecutive 3x3 convolution architecture comparison

To be more concise, we can imagine two scenarios: reaching a receptive field of 5x5 convolutions in the first layer and reaching the same receptive field via two consecutive 3x3 convolutions. In both cases, the input dimension is 128x128x3, and the desired output dimension is 128x128x32.

In the first scenario, the number of parameters is kernel shape(heightxwidthxdepth) times the number of kernels(5x5x3x32), and the number of FLOPs is kernel parameters times the input size(x128x128). In the second scenario, the number of parameters is (3x3x3x32) in the first layer, while in the second layer, the number of parameters is (3x3x32x32), as the depth of the input tensor is increased. The FLOPs are the number of parameters in each layer times the input height and width (x128x128) (see Table 7.2. It is important to note, that in general two consecutive 3x3 convolutions would decrease the FLOPs and the number of parameters if the input dimension is the same as the output dimension.

We found Spatial Dropout to be another key in the performance of the final model. In Dense neural networks, Dropout is equivalent to dropping a neuron in a single layer, as described in the Background section. Dropout is implemented as zeroing out a column in the weight matrix. However, zeroing out a column in a convolutional weight matrix does not fully eliminate a set of weights, and all weights are still updated. Dropout was shown to still work well on Convolutional neural networks, as it is equivalent to noise injection [50], but there is no clear theoretical reason behind it. Spatial Dropout, however, would do exactly the same intuitive operation as dropping a neuron in

Figure 7.5: Final model feature map visualizations

the Dense network. The Spatial Dropout introduced by [51] randomly drops a kernel and the weights in the dropped kernels would not be updated.

To make sure that Spatial Dropout was beneficial for the network, the final model with the same architecture, but with regular Dropout layers instead of the Spatial ones was also trained and evaluated. It showed around 2% less accuracy on our test data than the proposed model: CelebA test accuracy at 91.23%, and Genki test accuracy at 92.39%.

As a final step, we visualized the layers of the network by applying the final model's feature maps to a sample image (Figure 7.5). We randomly sampled three activation maps from each layer. We can see that the features gradually get more and more complex, giving a somewhat interpretable representation for the types of features the corresponding kernels are looking for. These features are simple lines and edges for the initial layers, while for later layers, the searched features are more complex and less interpretable.

7.2.4 Results

The accuracy of the networks described above is presented in Table 7.3. The classifiers trained via transfer learning are presented without any layer removal or significant modification.

Model	GENKI-4k accuracy	CelebA accuracy
VGG16	0.906	0.895
Inception V3	0.921	0.912
EfficientNet B0	0.926	0.922
MNasNet	0.919	0.921
Mobilenet V3	0.930	0.921
MobileNet Small V3	0.921	0.915
Proposed model	0.945	0.927

Table 7.3: Trained networks' accuracy

The inference times for the described networks were evaluated both on 1) CPU-only system with Intel(R) Core(TM) i7-8565U CPU @ 1.99 GHz processor with 8 GB LPDDR3 RAM, and 2) Intel(R) Xeon(R) Silver 4208 CPU @ 2.10GHz with 251GB RAM, with Nvidia GeForce RTX 2080 Ti 11GB GPU. The results of evaluations on 1000 image frames are presented in Table 7.4 and Table 7.5.

Model	Avg execution time (s)	Max execution time (s)	Std execution time (s)	FPS
VGG16	0.272	0.374	0.015	3.680
Inception V3	0.142	0.312	0.014	7.065
EfficientNet B0	0.124	0.202	0.006	8.057
EfficientNet B3	0.298	0.459	0.019	3.355
MNasNet	0.054	0.090	0.004	18.598
MobileNet V3	0.064	0.109	0.005	15.511
MobileNet Small V3	0.041	0.087	0.003	24.582
Proposed model	0.022	0.044	0.001	44.712

Table 7.4: CPU inference time performance evaluation

Model	Avg execution time (s)	Max execution time (s)	Std execution time (s)	FPS
VGG16	0.022	14.668	0.463	45.106
Inception V3	0.055	24.345	0.769	18.225
EfficientNet B0	0.051	1.096	0.033	19.749
EfficientNet B3	0.091	1.377	0.041	11.008
MNasNet	0.030	1.060	0.033	33.434
MobileNet V3	0.048	1.105	0.033	20.945
MobileNet Small V3	0.041	0.050	0.001	24.346
Proposed model	0.017	12.932	0.409	59.482

Table 7.5: GPU inference time performance evaluation

7.3 Final inference pipeline

Based on the results described in the previous sections, the final proposed inference pipeline, therefore, is presented as follows: input video frame processing through Media Pipe face detector, followed by scaling of pixel values, followed by detected face classification using the proposed U-Net inspired CNN. The overview of the proposed pipeline is illustrated in Figure 7.6.

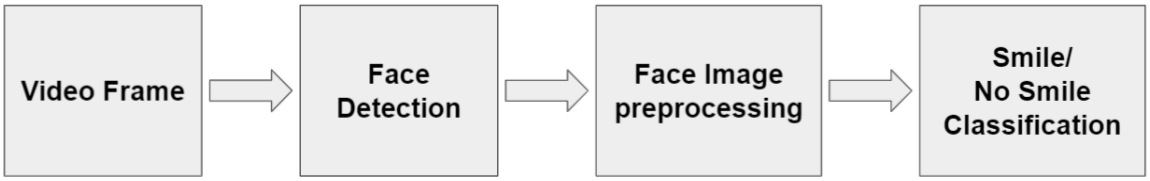


Figure 7.6: Final inference pipeline outline

The overall inference speed was evaluated on the same Core(TM) i7-8565U CPU with 8 GB LPDDR3 RAM for 1000 image frames. A face detector was applied to a 350x350 size image, and then a single smile classification inference was performed

with the proposed model. The pipeline showed real-time performance at 0.035 s average execution time, 0.058 max execution time, and 28.43 FPS. So, the final inference pipeline shows real-time CPU performance for recognizing smiles in one face image.

Chapter 8

Conclusions and future work

The Capstone project was a great opportunity to dive into Deep Learning and Computer Vision for the team members. We incorporated a simple, yet effective approach to solving the problem of smile recognition. In the scope of our Capstone project, as a final result, we were able to propose a high accuracy smile recognition system, which has real-time performance under specified constraints.

Through our experiments and results we showed that the initial focus on finding a best classifier via transfer learning was ineffective for the given task due to the size and inference latency of those networks. Most initial CNNs are optimized only with respect to their accuracy, while the later mobile networks (MNasNet and Mobilenets), which are optimized over both accuracy and inference latency, were found to be performing well only on Mobile platforms and Embedded systems. Designing a custom network and training it from scratch was found to be the most effective solution to solve the task at hand.

The future work would entail improving both the accuracy and the inference latency of the proposed model via the same pipeline or different approaches. Further work could be done to extend the real-time inference for arbitrarily many smile classifications in an image. The selected CNN-based face detector and our classifier are not sharing any layers and information in general, which could be beneficial for the system as a whole. One possible approach would be to design a network with a multiobjective loss - one for face detection, and one for smile classification. Another approach would be to use a fast facial recognition network as a feature extractor and train a small classifier on it. Investigation of the novel, advanced network optimization techniques, such as network compression and acceleration through quantization, could also be the subject of further research.

Bibliography

[1] P. Ekman and W.V. Friesen. *Facial Action Coding System*. v. 1. Consulting Psychologists Press, 1978. URL: https://books.google.am/books?id=08l6wgEACAAJ.

[2] Yingli Tian, Takeo Kanade, and Jeffrey Cohn. "Recognizing Action Units for Facial Expression Analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(2): 97-115". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 23 (Mar. 2001), pp. 97–115. DOI: 10.1109/34.908962.

[3] Irene Kotsia and Ioannis Pitas. "Facial Expression Recognition in Image Sequences Using Geometric Deformation Features and Support Vector Machines". In: *IEEE Transactions on Image Processing* 16.1 (2007), pp. 172–187. DOI: 10.1109/TIP.2006.884954.

[4] M. Valstar, I. Patras, and M. Pantic. "Facial Action Unit Detection using Probabilistic Actively Learned Support Vector Machines on Tracked Facial Point Data". In: *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. Los Alamitos, CA, USA: IEEE Computer Society, June 2005, p. 76. DOI: 10.1109/CVPR.2005.457. URL: https://doi.ieeecomputersociety.org/10.1109/CVPR.2005.457.

[5] M.S. Bartlett et al. "Fully Automatic Facial Action Recognition in Spontaneous Behavior". In: *7th International Conference on Automatic Face and Gesture Recognition (FGR06)*. 2006, pp. 223–230. DOI: 10.1109/FGR.2006.55.

[6] Varun Jain and James Crowley. "Smile Detection Using Multi-scale Gaussian Derivatives". In: (Feb. 2013).

[7] Le An, Songfan Yang, and Bir Bhanu. "Efficient smile detection by Extreme Learning Machine". In: *Neurocomputing* 149 (2015). Advances in neural networks Advances in Extreme Learning Machines, pp. 354–363. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2014.04.072. URL: https://www.sciencedirect.com/science/article/pii/S0925231214011370.

[8] Caifeng Shan. "Smile detection by boosting pixel differences". In: *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society* 21 (July 2011), pp. 431–6. DOI: 10.1109/TIP.2011.2161587.

[9] Patrick Glauner. "Deep Learning For Smile Recognition". In: Aug. 2016, pp. 319–324. DOI: 10.1142/9789813146976_0053.

[10] Junkai Chen et al. "Smile detection in the wild with deep convolutional neural networks". In: *Machine Vision and Applications* 28 (Feb. 2017). DOI: 10.1007/s00138-016-0817-z.

[11] Dinh Sang, Le Cuong, and Do Thuan. "Facial smile detection using convolutional neural networks". In: Oct. 2017, pp. 136–141. DOI: 10.1109/KSE.2017.8119448.

[12] Xin Guo, Luisa Polania, and Kenneth Barner. “Smile detection in the wild based on transfer learning”. In: (Jan. 2018).

[13] Jiongwei Chen et al. “Novel multi-convolutional neural network fusion approach for smile recognition”. In: *Multimedia Tools and Applications* 78 (June 2019). DOI: 10.1007/s11042-018-6945-x.

[14] Youngkyoon Jang, Hatice Gunes, and Ioannis Patras. “Registration-free Face-SSD: Single shot analysis of smiles, facial attributes, and affect in the wild”. In: *Computer Vision and Image Understanding* 182 (Feb. 2019). DOI: 10.1016/j.cviu.2019.01.006.

[15] Pedram Ghazi et al. “Embedded Implementation of a Deep Learning Smile Detector”. In: Nov. 2018, pp. 1–6. DOI: 10.1109/EUVIP.2018.8611783.

[16] Prateek Karkare. *Neural Networks an Intuition*. URL: https://www.kdnuggets.com/2019/02/neural-networks-intuition.html.

[17] Lihi Shiloh-Perl and Raja Giryes. *Introduction to deep learning*. 2020. arXiv: 2003.03253 [cs.LG].

[18] Shiliang Sun et al. *A Survey of Optimization Methods from a Machine Learning Perspective*. 2019.

[19] Amine ben khalifa and Hichem Frigui. “Multiple Instance Fuzzy Inference Neural Networks”. In: (Oct. 2016).

[20] Fei-Fei Li. *Convolutional Neural Networks for Visual Recognition*. 2017. URL: https://cs231n.github.io/.

[21] *Convolutional Neural Network*. URL: https://brilliant.org/wiki/convolutional-neural-network/.

[22] Chuanqi Tan et al. *A Survey on Deep Transfer Learning*. 2018. arXiv: 1808.01974 [cs.LG].

[23] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[24] Manolis Loukadakis, José Cano, and Michael O’Boyle. “Accelerating Deep Neural Networks on Low Power Heterogeneous Architectures”. In: Jan. 2018.

[25] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV].

[26] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].

[27] Mingxing Tan et al. *MnasNet: Platform-Aware Neural Architecture Search for Mobile*. 2019. arXiv: 1807.11626 [cs.CV].

[28] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV].

[29] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV].

[30] Andrew Howard et al. *Searching for MobileNetV3*. 2019. arXiv: 1905.02244 [cs.CV].

[31] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[32] Paul Viola and Michael Jones. "Robust Real-Time Face Detection". In: *International Journal of Computer Vision* 57 (May 2004), pp. 137–154. DOI: 10.1023/B:VISI.0000013087.49260.fb.

[33] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[34] Davis E. King. "Dlib-ml: A Machine Learning Toolkit". In: *Journal of Machine Learning Research* 10 (2009), pp. 1755–1758.

[35] Lourdes Ramirez Cerna. "Face Detection: Histogram of Oriented Gradients and Bag of Feature Method". In: (Oct. 2013).

[36] Kaipeng Zhang et al. "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks". In: *IEEE Signal Processing Letters* 23.10 (2016), pp. 1499–1503. DOI: 10.1109/LSP.2016.2603342.

[37] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[38] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *arXiv* (2018).

[39] *Face Detection*. URL: https://google.github.io/mediapipe/solutions/face_detection.html.

[40] Valentin Bazarevsky et al. *BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs*. 2019. arXiv: 1907.05047 [cs.CV].

[41] http://mplab.ucsd.edu. *The MPLab GENKI Database*.

[42] http://mplab.ucsd.edu. *The MPLab GENKI Database, GENKI-4K Subset*.

[43] Ziwei Liu et al. "Deep Learning Face Attributes in the Wild". In: *Proceedings of International Conference on Computer Vision (ICCV)*. Dec. 2015.

[44] Ali Mollahosseini, Behzad Hasani, and Mohammad H. Mahoor. "AffectNet: A Database for Facial Expression, Valence, and Arousal Computing in the Wild". In: *IEEE Transactions on Affective Computing* 10.1 (2019), pp. 18–31. DOI: 10.1109/TAFFC.2017.2740923.

[45] Martn Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[46] François Chollet et al. *Keras*. https://keras.io. 2015.

[47] Connor Shorten and Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1 (2019). DOI: 10.1186/s40537-019-0197-0.

[48] Alexander Buslaev et al. "Albumentations: Fast and Flexible Image Augmentations". In: *Information* 11.2 (2020). ISSN: 2078-2489. DOI: 10.3390/info11020125. URL: https://www.mdpi.com/2078-2489/11/2/125.

[49] Kaiming He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. arXiv: 1502.01852 [cs.CV].

[50] Yarin Gal and Zoubin Ghahramani. *Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference*. 2016. arXiv: 1506.02158 [stat.ML].

[51] Jonathan Tompson et al. "Efficient object localization using Convolutional Networks". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015). DOI: 10.1109/cvpr.2015.7298664.