COMP.SGN.240 Advanced Signal Processing Laboratory Image Recognition Report

Fareeda Mohammad, Nardos Estifanos

November 2023

1 Introduction

In this image recognition assignment, we took a deep learning approach towards binary classification to recognize presence or lack of smile from image/video input.

2 Data processing

The dataset used in to obtain the model was GENKI-4k[1] dataset, which contains 4000 images, 2162 smiling, the rest not. It shows good enough class distribution.

The data processing pipeline will iterate over the image folder, read individual item along with corresponding label from provided annotation file (taking only first entry 1/0 - smile/non-smile). Then, image is resized to 64x64x3, and normalized by 255.0 which helps during training, and finally converted to tensor following expanding dimension on axis 0.

Following this, processed data is shuffled and split to train/test set with 80/20 setup. Then, both are converted to tensorflow dataset, which makes operating on the dataset easy among other benefits. The class distribution for both sets is shown below.

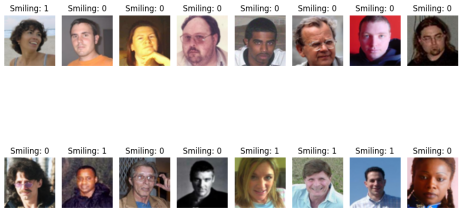
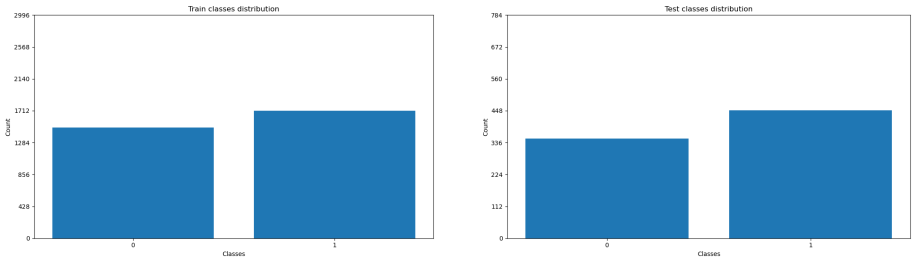


Figure 1: Sample images in training dataset



(a) Train dataset class distribution (b) Train dataset class distribution

Figure 2: Class distributions

3 Model architecture

The binary classifier network was provided as shown below. It makes use of CNN layers to learn complex features about the images of the two classes, with RELU activation to learn complex boundaries, and max pooling layers to focus on dominant features. Finally, this complex feature representation is fed to the dense neural network to predict corresponding class.

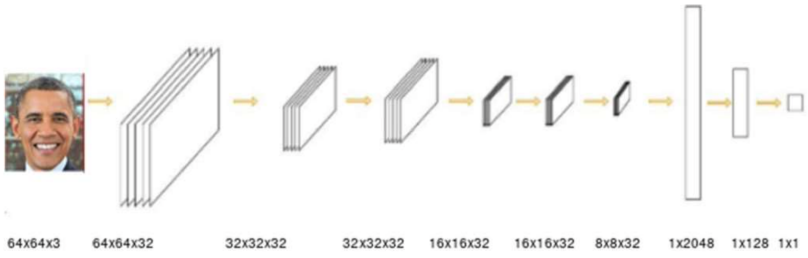


Figure 3: Topology of example network (from instructions)

4 Training and Results

The network shown above was compiled, initially, with ADAM optimizer, binary cross entropy loss, and accuracy as metrics. Training was done on google colab with T4 GPU, a batch of 64 was used. Although the network was learning fast (< 50 epoches), it suffers from overfitting as the performance on the test significantly lower than the training metric. Later, NADAM optimizer is used as it leads to improved performance.

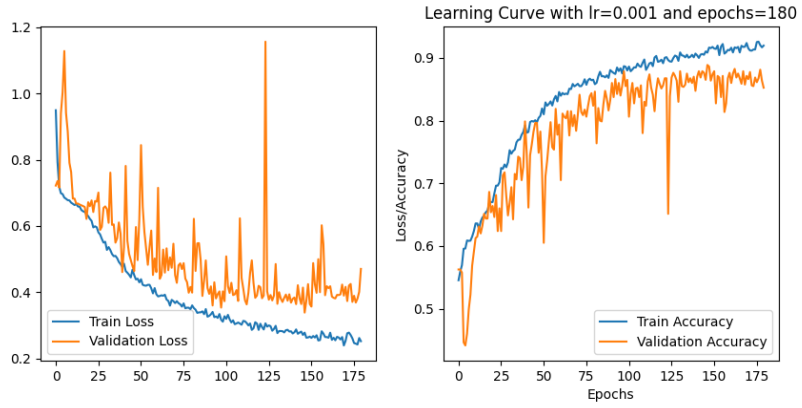


Figure 4: Training vs Test loss and accuracy (to investigate overfitting, and overall training)

4.1 Dealing with overfitting

4.1.1 L2 Regularization and Dropout

The first approach was applying L2-Regularization to penalize by $\lambda = 1e-4$, so that the model doesn't memorize the training data, and falsely seem to achieve high accuracy. And the dropout layer randomly turns off paths so that the model gets discouraged to memorize. This helped with keeping the diverging effect in control, and keep training and validation (test) learning curves move similarly. However, overall accuracy dropped and the model couldn't learn.

4.1.2 Data Augmentation

Even though the dataset was diverse in terms of contrast, rotation, brightness, and more, there wasn't enough collection for each case, hence, to make the model robust in classifying under different conditions and to help make CNN models become shift invariant, augmenting training data was neccessary. During training, the tensorflow sequential data augmentation layer was used, however, due to experimental nature of this implementation, this layer based method is used in google colab, implemented with a function in local machine case. Adding augmentation helped the model to gain significant accuracy, along with regularization. However, this time 50 epoches was not enough to reach 85% accuracy. And learning/convergence took longer.

4.1.3 Batch Normalization

To help the modified model learn faster, batch normalization layer is added in between CNN blocks. This helps the model stabilize and accelerate its learning.

4.1.4 Train longer with dynamic learning rate

After this to reach the target accuracy, epoch was set to higher number (180) and since it was loosing its momentum at the last rounds, dynamic learning rate with exponential decaying was used to help it train faster longer.

4.2 Results

With this modification, and training setup, $> 87\%$ accuracy was achieved in test set. Our implementation allows you to train, evaluate, and infer by passing arguments mentioned in the README.md file. The infer option allows image, video file, and webcam inputs. The inference time is around 50ms per image/frame. Here are some of the results and classifications of our model.

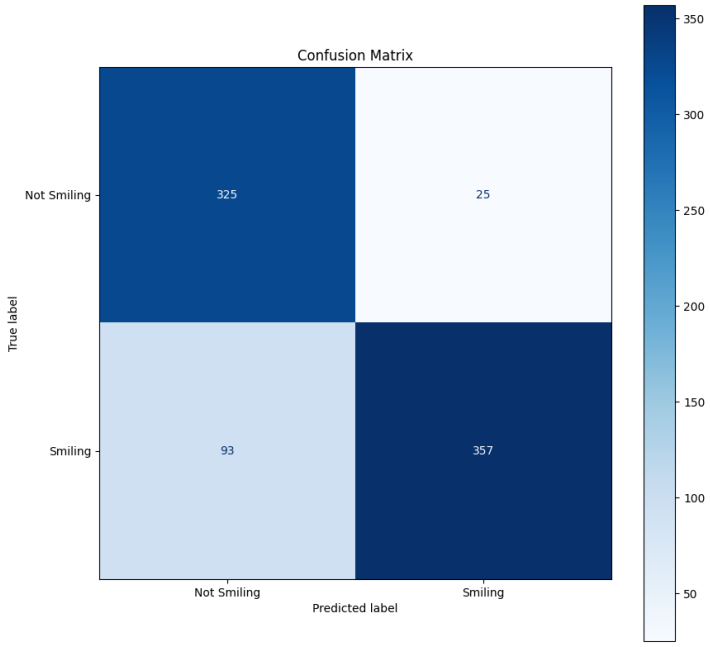


Figure 5: Confusion Matrix (diagonal darker shades indicate good model or less confusion)

Figure 6: Model Performance on one batch from test set

5 Summary

To summarize, the binary classifier was implemented making use of deep learning design. Different training and tuning experiments helped reach target accuracy. And to improve in video/webcam based feeds, HAAR cascade face detector feature was added to extract wide area around the user face to minimize loss of performance by irrelevant context. After detecting faces, the area is expanded so that it matches the dataset content.

References

[1] The MPLab GENKI Database, GENKI-4K Subset. http://mplab.ucsd.edu. Accessed: 2023.