A Battery-Powered Friend: Towards Embedded Emotion Recognition
(CS231N Project Report)

Paul Shved*
Stanford CS 231n (Spring 2019)
pshved@stanford.edu

Abstract

In this project, we set out to build a smiling robot: an embedded, battery-powered device that "smiles back" when a human subject in front of it smiles. We define a problem of building a neural system that would power the smiling robot. We prepare the dataset, apply augmentations, and use MTurk to label data for this project specifically . We set out to find the neural architecture that, once deployed, consumes the least power, but achieves the desired quality defined as 80% of the Area Under ROC Curve compared to state-of-the-art models. We use the number of operations during the evaluation pass as an estimate of the power consumption. The AUC performance targets are set by a CNN similar to a 6-layer AlexNet that achieves state-of-the-art performance on our dataset. Based on the validation set performance, we have found a 4-layer convolutional network that achieves the desired outcome, and theoretically runs on SparkFun Appollo 3 in 3 seconds; based on the test set evaluation, only architectures that required 6 seconds perform well. Due to time constraints, we leave the deployment of the resultant CNN onto an actual embedded device to future work. We additionally find, by inspecting saliency maps, that our model tends to identify the smile by locating muscle contraction in the cheeks rather than by the shape of the lips.

1. Introduction

Humans communicate with one another and the world around them via multiple means. In addition to speech and gestures, emotions displayed through facial expressions is one of the most powerful ways humans (and other animals) use to communicate with each other. [6]. While some emotions are learned, many emotions, such as joy or anger, are innate. The facial expressions humans use to display these emotions also tend to be innate [17]. Thus the use of emotions in human-machine interactions will require no special training as most humans already know how to display basic emotions such as joy.

Recognizing emotions displayed through facial expression requires the device with which the human operator interacts to run visual recognition software. The challenge with this is twofold. On one hand, human response to emotions tends to be immediate: failure to recognize frustration quickly might lead to more frustration. The other challenge is that robots and embedded devices have limited hardware and limited computational power. While performing inference on device has tight resource constraints, training and data augmentation and preparation will be performed using a more powerful system.

1.1. Problem Statement

In this project, we make progress towards developing and deploying a neural system that performs the following task. Given a still image in visible light, the system provides a "yes or no" answer to the following query:

"Is there a smiling person in this picture?" (1)

We aim to build a model that can be deployed onto an embedded system. If the answer to this question is "yes", the system will light up a pattern of lights simulating a human smile.

Due to time constraints (mostly, in order to put emphasis on Visual Recognition tasks rather than on working with and optimizing embedded hardware) we will be implementing emotion recognition from single-frame images.

1.2. Embedded System Constraints

Deployment onto embedded system imposes an upper bound on computational requirement of a model. We aim to achieve a response from the system **within 3 seconds** when running on a SparkFun Apollo3 development board.

*The author thanks Anna Smagina for the advice on dataset collection methods for visual recognition and for suggesting to use augmentations.

The board chip's clock speed is 48 Mhz, so one might assume that the chip is capable of doing 48 million of integer additions per second. However, there are two considerations:

- CNN-based models use floating-point rather than integer operations (without special tuning);
- other operations used in CNN-based models include multiplication and division (e.g. in normalization layers), which are more expensive;
- loading and storing data in memory requires multiple clock cycles.

Thus in this work we use a floating-point load-add-store as an approximation of a computational cost of one average operation, which takes 6.5 microseconds on Arduino Uno M3 [2]. That board is three times as slow as our target device. Extrapolating, our upper bound on the number of floating point operations is **0.46 million operations per second**; 3 seconds yield **1.38 million operations per inference** as the upper bound we use in this project.

2. Related Work

Large body of prior work on emotion recognition exists. The literature references non-neural systems features [14], purely neural systems [19], and combinations of those [11].

2.1. Non-neural systems

Non-neural systems for emotion recognition such as [14] rely on Haar-like features and Viola-Jones object detector [24]. This object detection uses simple convolution that detects horizontal and vertical edges, as well as brightness change in a certain direction (e.g. the bottom of the cheeks on the human face will likely be darker than the top).

The kernels in these convolutions are not trained on the data. The training happens on the features produced by them, e.g. by using AdaBoost algorithm [9]. A drawback of that this approach would be that the resultant system would not be robust to changes in face orientation and lighting conditions.

An advantage of such systems would be that they require little computational power. E.g. one of the modern such systems [14] was training as well as running inference on hardware equivalent to a modern cell phone.

2.2. Neural systems

State of the art facial emotion recognition using deep learning have been the state of the art in the recent years due to their better accuracy [15].

An advantage of CNN-based architecture to non-neural systems surveyed in section 2.1 is its robustness to scale, pose changes, and position changes, which was established in [8]. CNNs tens to even learn the Facial Action Unit features by themselves [12].

Consider a network architecture for emotion recognition in [7]. The architecture first uses deep CNN to extract features from each still image in a video stream; the architecture is based on a pre-trained deep network with 5 CNN layers, two max-pooling layers (AlexNet [13]), and two fully-connected layers which were re-trained on emotion recognition tasks. This produces the single-image features. The video features are produced using eigenvectors, covariance matrices, and multidimensional mean, and variance of each feature. PArtial Least Square regression [21] is applied instead of more traditional SVM or Logistic regressions. The model achieves state-of-the-art performance on FER2013 dataset. The advantages of this approach is that it does not rely on handcrafted feature or complex preprocessing. Other papers such as [19] use a similar approach.

Many emotion recognition systems employ preprocessing before using CNNs to extract features. Almost all use Viola-Jones face detection [24] to find the region that contains a face in the larger frame. Other approaches to face detection use Fast R-CNN [10]. In this approach, the whole image is passed through a CNN architecture, which proposes multiple regions. These regions are considered as faces for further feature extraction using an emotion-specific, different CNN.

Once the face region was identified, some papers such as [11] use "face alignment" [3] prior to CNN-based feature extraction. In this approach, landmark features (location of the eyes, nose, and mouth) are detected using non-neural methods, and a spatial transformation is used to normalize the position of the face so that it fills the entire frame.

Other state-of-the-art facial emotion recognition systems employ more complex pipelines and ensembles of neural networks [15], but exploring facial recognition in this much detail was out of scope of this project.

We note that state-of-the art emotion recognition systems evaluated on large test sets achieve 71-75% accuracy for 6 classes evaluated on 3589 images and 85-89 on 834 test set images. [15].

2.3. Deep learning and Embedded devices

Recent work by Pete Warden [20] demonstrated that online speech recognition can run on a coin battery powered device with only 100kb of RAM. The speech recognition algorithm in [20] used a multi-layer Convolutional Neural Network, and the model size used was only 80Mb.

We found that Tensorflow machine learning framework was ported to FPGA architecture [18]. This motivated our choice of framework in our implementation.

However, we were not able to find documented literature on successful deployment of modern emotion recognition methods on embedded hardware. Some pre-neural methods

used laptops [14] used laptops to achieve portability; others use embedded hardware as a frontend to a cloud-based emotion recognition service [5].

3. Dataset and Data Preparation

We used a dataset of 6000 training images and 1000 test images of size 224x224 (center-cropped and downscaled). The images were based on [23], and we obtained labels from Mechanical Turk specifically for this project. 1000 of test set images was withheld for validation; no cross-validation was used. We used augmentations and face detection, but no face alignment.

3.1. Obtaining the dataset

Facial recognition datasets are difficult to obtain for independent research. Most datasets we surveyed require collaboration with a full-time employee of an academic institution. The dataset we were able to obtain quickly (we used JACFFE [4]) contain very few examples of "happy" facial expressions (50 and 6 respectively). Early evaluations demonstrated this dataset is insufficient. Thus, we have performed our own, project-specific labeling.

In our data preparation, we start with the "Google facial expression comparison dataset" [23]. The dataset contains images of people and groups of people "in the wild" (in natural, non-laboratory conditions) that visibly display emotions. It contains 156,000 images, with about 10% withheld for the test set. The images were also annotated with bounding boxed for faces produced by a Viola-Jones detector [24]. Sadly, the images of the dataset were not labeled with actual emotions.

We have performed a study on Mechanical Turk. The raters were asked a question "Is the person smiling in this picture?", with possible answers "Yes", "No / Unsure", "Picture invalid (not a person, multiple people, etc)". We've labeled 8000 images from the training set, and 1000 images from the test set. The study took about 1 hour to complete.

To save on study cost, we have asked only 1 rater to evaluate 1 picture. On manual evaluation of a random sample of 64 pictures, we found that 9 of them were labeled incorrectly, which estimates that the dataset contains 14% mislabeled data. Asking 3 raters to rate every picture, and taking a majority vote would decrease this rate to a more acceptable $0.14^2 \approx 2\%$.

We have found that 31% of the dataset included people who are smiling. See more how we tackled class imbalance in section 3.3.

3.2. Augmentation

A popular technique to boost the size of the dataset in visual recognition tasks is data augmentation. In this project we use the generic augmentation techniques: occlusion, small random affine transformations (rotation, flipping), gaussian blur, noise, and embossing. Every augmentation was applied with small probability of 20%. The intended device camera may be used in low-light environments, so we applied weak motion blur (limit=5) with 85% probability, and strong (limit=40) with 10%. Examples of the augmentations can be found on Figure 1.

Figure 1. Left column: images from the training set with the augmentation performed; right column: source images of the same subject (for comparison).

Note that we did not perform augmentations on either test or validation sets.

3.3. Compensating for class imbalance

We found that 31% of training data was labeled with positive labels (the person was smiling). Using Corss-Entropy loss requires approximately equal number of examples. However, since the class imbalance was not significant, we simply repeated random 2 out of each 3 images from the "smiling" class to achieve the balance instead of using a more sophisticated technique like Focal Loss [16].

4. Methods

The challenge of this project is not just to build a network that achieves strong performance on the metrics, but also the one that performs inference in few enough arithmetic operations as was described in section 1.2. In section 4.1 we first describe our meta-learning strategy and in 4.2 we describe the architecture of the resultant CNN-based network.

4.1. Building an embeddable network

Our meta-learning algorithm is parametrized with a target reduction in the baseline model performance: by how much can our reduced model perform worse on validation set than the baseline model while still considering viable for deployment. As viability for deployment we use a target maximum number of operations F. We chose the **Area Under ROC curve minus** 0.5 as our target metric and call target reduction β.

ROC curve ("receiver operating characteristic") demonstrates how a binary classifier performs compared to random chance. Given a model and an evaluation set, the curve is a 2D plot of points:

$$(\text{ False Positive Rate, True Positive Rate}) = \tag{2}$$

$$\left(\frac{\Sigma \text{ False Positive}}{\Sigma \text{ Actual Negative}}, \frac{\Sigma \text{ True Positive}}{\Sigma \text{ Actual Positive}}\right) \tag{3}$$

Examples can be found in the table 5. The area under this curve is referred to "AOC" and is considered the primary metric for comparison of different architectures in this project.

As our meta-learning algorithm, we essentially perform a depth-first search on the reductions of the architecture of the baseline model until we find a model that performs well enough on validation set.

Algorithm 1: An A*-like meta-learning algorithm

Data: A baseline model m_0; function T that trains and evaluates the AUC on validation set $T : m \to t$
Result: Final model m or `nil` if not found
Function `Recurse` *(m, β, s)* **is**
  Data: Model m, AUC target β, FLOPS target s
  Result: Model m' or `nil`
  if $T(m) < \beta$
    return `nil`
  elif *Size*$(m_{best}) < s$
    return m
  for $m' \in$ *Reductions*(m)
    $m_{best} \leftarrow$ `Recurse`(m',β);
    if $m_{best} \neq$ `nil`
      return m_{best}
  return `nil`
$\beta_0 \leftarrow 80\% \cdot T(m_0)$;
return `Recurse` *(m_0, β_0)*

4.2. Network architecture

At the core of the used methods is a Convolutional Neural Network. Setting image-specific tasks aside, a single neural network unit for classification task usually consists of two parts: feature extraction and classification. The latter is usually a fully-connected layer (or multiple) that produces output $z = Wx + b$ where x is the feature vectors produced by the feature extraction. The result of the final fully-connected layer is fed to loss function during training, or to a related classifier during testing. In this project we used Cross-Entropy loss L and Softmax classifier defined as (for a binary classifier):

Layer	Depth	stride	kernel	activation
Conv2D	96	4	11	relu
MaxPool		2	3	
BatchNorm				
Conv2D	256	1	5	relu
MaxPool		2	3	
BatchNorm				
Conv2D	384	1	3	relu
Conv2D	384	1	3	relu
Conv2D	256	1	3	relu
MaxPool		2	3	
FC	4096			softmax
FC	2			softmax

Table 1. Simplified AlexNet architecture that we used as a baseline model m_0. The final layer produces class scores for classes "Not Smiling" and "Smiling".

$$L = -\left(y \log y' + (1-y)\log(1-y')\right) \tag{4}$$

$$y' = \frac{e^{z_1}}{\sum_{i=1}^{2} e^{z_i}} \tag{5}$$

Feature extraction can be performed using non-neural features (edge detection, histogram of oriented gradients, etc), or using neural feature extractor such as CNN that can be trained on the input data. A 2D convolution is an operation to apply a kernel $k(i,j)$ of size $a \times b$ to the input image $a(i,j)$ to produce the output image $b(i,j)$

$$b(x,y) = \sum_{m=-a}^{a} \sum_{l=-b}^{b} k(m,l)a(x-m, y-m) \tag{6}$$

The values of b become features for the next layer of the network. The "weights" $k(i,j)$ are updated during training to minimize the loss given by equation (4).

During inference, the value of y' from (5) is compared with **threshold** T to determine if the class.

4.2.1 Baseline architecture

Many facial emotion recognition systems surveyed [7] [10] use AlexNet [13] and derived / fine-tuned architectures for visual emotion recognition. We start with m_0 as the simplified form of AlexNet presented in the cs231n Spring 2019 course lectures (See Table 4.2.1).

Note that since we need to reduce (rather than extend) the architecture of the network, we did not consider using pretrained embeddings.

Initially, we aimed to tune our learning rate in order achieve state of the art emotion recognition performance (accuracy of 70+%) on our dataset using the AlexNet described in Table 4.2.1. We tuned learning rate and optimizer

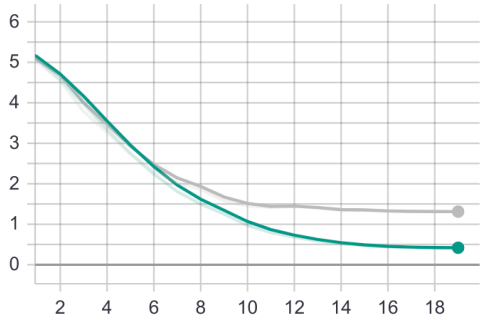

Figure 2. Loss for training and validation sets on the baseline AlexNet model. Y axis: cross-entropy loss value, X axis: epoch number. Loss graphs for all other networks were similar.

parameters until we were able to achieve stable, converging training depicted in Figure 2. Max Learning Rate is 0.05 with linear increase to this value until epoch 10 and cosine decay given by $\nu = \nu_0 \frac{1+cos(\frac{epoch}{N})}{2}$ after. We trained thus for 20 epochs: training for more epoch started to decrease the performance on the validation set universally. As optimizer, we started with using Adam, however, the model was not converging in most runs, so we tried Stochastic Gradient Descent with Nesterov momentum ($\mu = 0.8$ worked best).

Nesterov Momentum update replaces SGD's $x' = x_0 - \nu dx$ with multi-step update. We track both the current position x, but also "velocity" v, which are updated using the following rules:

$$v' = \mu v - \nu dx \tag{7}$$
$$x' = x - \mu v + (1 + \mu)v' \tag{8}$$

We used batch size of 64 to utilize the resources of one nVidia RTX 2080Ti GPU for training and evaluation. Total training time for each run was 28 seconds per epoch.

Since we achieved the desired 71% accuracy, the area under curve of that model on the test set 0.70 was used to calculate 0.67 as the performance target for the reduced model.

4.2.2 Simplifications used

As the potential simplifications for Algorithm 4.1 we considered:

- **Reducing number of filters**: AlexNet was designed to distinguish many classes whereas in the current project we only have two classes. Reducing the number of filters to some extent might not have effect.
- **Increasing stride of convolutions**: larger stride means that the kernel will be applied fewer times for the same image, albeit the layer will produce fewer features.

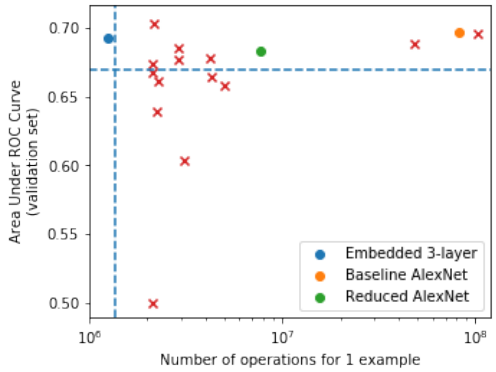

Figure 3. AUC of validation set evaluation of every model tried during the evaluation of the meta-algorithm. The horizontal line is the quality cutoff: since the Baseline AlexNet model achieved 71.5% AUC, and the value of AUC of a completely random classifier is $0,5$, the cutoff $\beta = 0.8$ yields $0.5 + (0.71 - 0.5)\beta = 0.67$ cutoff for AUC of the considered models. The vertical axis is the operation count cutoff. The successful target models must land in the top left quadrant.

- **Reducing the size of the fully-connected layer** (same reasoning as for reducing the number of filters)
- **Removing middle layers**.

We also employed methods that improve error without adding computation to inference (more computation requires more energy). An example of that is Dropout regularization [22]: during training evert activation is omitted with probability p; this simulates averaging over large ensemble of models and reduces overfitting. During testing and inference, all connections are active. This improves performance without increasing the number of operations (but we must be careful to measure the operations during testing only).

4.3. Baseline "simple" method

We've also employed a baseline fully-connected two-layer network, which we already implemented for Assignment 2. A non-neural baseline that was simple to implement (count white pixels) performed poorly even in the milestone assessment, and was not expanded in the final part of the project.

5. Experiment results

After running the meta-algorithm presented in section 4.1 on the baseline 4.2.1, we have arrived to the following three-layer model presented in Table 5. During the evaluation of meta-algorithm, multiple models were trained and evaluated, and the scatter plot of their validation set performance against the number of operations is presented in figure 3.

Model	Set	AUC	AUC drop	Accuracy	Precision	Recall	FLOPS	Est. runtime
Baseline AlexNet	test	0.70	0	0.72	0.53	0.52	82,700,000	3 min 1s
Reduced AlexNet	val	0.67	-15%	0.68	0.45	0.41	5,100,000	-
Reduced AlexNet	test	0.66	-20%	0.69	0.47	0.44	5,100,000	11s
Embedded 3-layer	val	0.69	-5%	0.70	0.48	0.49	1,300,000	-
Embedded 3-layer	test	0.62	-40%	0.64	0.39	0.39	1,300,000	2.9s

Table 2. Evaluation of the baseline AlexNet, the 3-layer Embedded model and the baseline Two-layer fully connected network. Additionally, the "Reduced AlexNet" described in section 5. The threshold for Precision and Recall metrics is set to 0.5 for the purpose of this table

Layer	Depth	stride	kernel	activation
Conv2D	12	8	11	relu
MaxPool		2	3	
Conv2D	32	1	5	relu
MaxPool		2	3	
BatchNorm				
Conv2D	256	1	3	relu
MaxPool		2	3	
Dropout	p=0.5			
FC	1024			softmax
FC	2			softmax

Table 3. Simplified AlexNet architecture that we used as a baseline model m_0. The final layer produces class scores for classes "Not Smiling" and "Smiling".

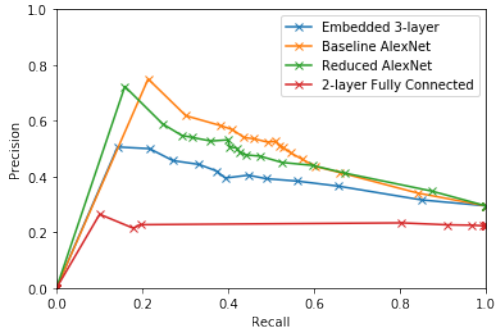

Figure 4. Precision-recall curve as evaluated on the test set.

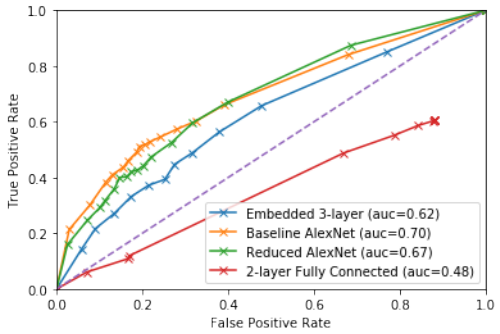

Figure 5. ROC curve as evaluated on the test set. Dotted line denotes the expected performance of a completely random classifier.

We noted during our experiments that removing Batch Normalization layers do not decrease the number of operations significantly, but doing so reduces performance on the validation set.

The results of evaluation of the models are presented in Table 5. In addition to the two baseline and the final network, it contains another net, **"Reduced AlexNet"**. During evaluation on the test we found that the Embedded 3-layer model didn't perform as well on test set as it did on the validation set. Out of the database of training runs, we have identified the minimal model that does perform within the acceptable range on the test set as well as on validation set.

This model is simply the baseline AlexNet displayed in Figure 4.2.1 with two simplifications: (a) every convolutional layer has 4 times as few filters, (b) the fully-connected layer only has 1024 hidden units, and (c) the stride on the first convolutional layer is increased to 8. However, this model would take 6 seconds to run instead of the 3 seconds we were targeting.

When selecting the value for the classification threshold, we need to consider the Precision-recall and ROC curves that are presented in figures 4 and 5 respectively. These curves provide a valuable insight into the relative performance of our models. For example, we can see that the Baseline and Reduced AlexNet models perform almost indistinguishably. Another insight is that precision is unelastic for the embedded 3-layer model, an we can achieve higher values of recall compared to the best precision for this model.

5.1. Feature visualisation

As a way to gain insight on how our model decides whether the person is smiling or not, we have used saliency maps. A saliency map is a visualization of how much every individual pixel contributes to the final score. We have computed it using gradient ascent of the cross-entropy loss (4) over the pixels of the image itself. The resultant heatmap of the gradients was visualized in Figure 6.

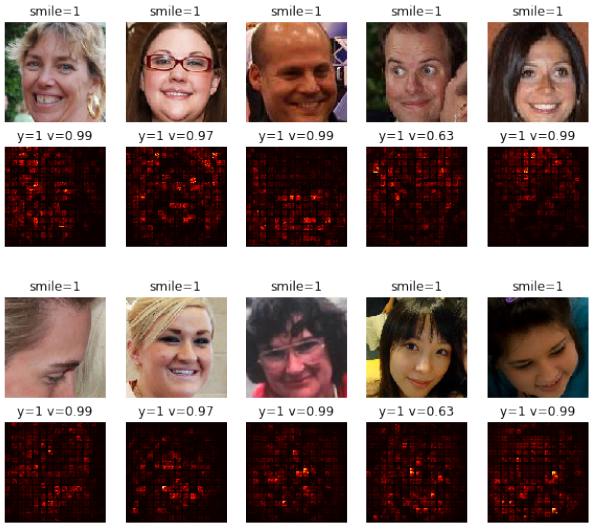

Figure 6. Saliency maps of the 3-layer embedded model (top) and "Reduced AlexNet" model (bottom). "Red" means low or no activations, and "Yellow" means higher activation. Note that the the models pay moderate attention to the mouth, but most of the attention goes towards the shape of the cheek (e.g. subject #2 in the top row).

Figure 7. Examples of "False Positives" of the 3-layber Embedded model: the model predicted "smiling", while the ground truth label was "not smiling". The value of v denotes the score of the "smiling" class.

Look at the saliency maps of the Reduced AlexNet model (Figure 6). We noticed, especially while observing subject #2 in Figure 6 in the bottom (but also all other subjects in the bottom) and especially subject #3 at the top that the pixels at the mouth itself have little effect. The most pronounced effect was the deformities in the checks.

The model even correctly identified. based on these features, that the subject #1 in bottom row is smiling, although the actual smile was cropped out during preprocessing (verified by the original photograph).

5.2. Error Analysis

Examples in figure 7 feature several errors for the 3-layer embedded model. One very prevalent example was mislabeled ground truth data (subject #4 in both figures). Subject #3 in 7 has features in the cheek similar to what saliency maps associate with smiling expression albeit the subject is not smiling. The inverse is true for subject #2 bottom row: although the subject is smiling, the cheek features are not as pronounced due to dim lighting. However, the model has predicted a substantial score of 0.27, so tuning threshold might also help classify this example.

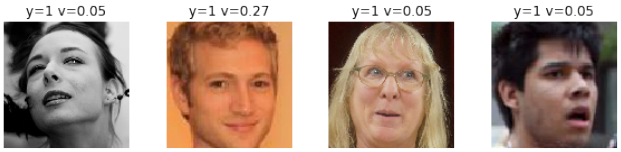

Figure 8. Examples of "False Negatives" of the 3-layber Embedded model: the model predicted "not smiling", while the ground truth data contained "smiling". The value of v denotes the score of the "smiling" class.

6. Conclusions

In this project, we have prepared a dataset and replicated performance approaching state-of-the-art on this dataset using AlexNet [13] for emotion recognition, similar to [19] and [7]. We've defined a framework for evaluating the fitness of a model for deployment on hardware with low clock speed. We've proposed and executed a meta-learning algorithm that can iteratively achieve the desired portability. However, the algorithm might have overused the validation set, as the model produced by it performed worse on validation set. Despite that, a model of acceptable quality (no more than 20% worse than the AlexNet-based detector as measured by $roc - 0.5$) that theoretically can be executed within 11 seconds on an embedded system was found (Table 5).

7. Future work

Due to time constraints we were not able to actually deploy the model on an embedded system, so this is left for future work. Attempting to deploy on SparkFun Apollo3 would provide the necessary validation of the assumptions made about performance of neural systems on embedded hardware.

The meta-algorithm was only executed manually, and automating it and improving it (e.g. not stopping after the first success which might not fare well on the test set) is left to future work. Obtaining a higher-quality dataset would also be necessary for high-quality results. Additionally, we would like to explore if removing connections and features from a network pre-trained on a more generic task achieves better performance than training a smaller network on the existing data for the task.

References

[1] E. K. V. I. I. A. Buslaev, A. Parinov and A. A. Kalinin. Albumentations: fast and flexible image augmentations. *ArXiv e-prints*, 2018.

[2] Arduino Community. Speed of math operations (particularly division) on arduino, 2016. https://forum.arduino.cc/index.php?topic=92684#msg2733723, Last accessed on 2019-04-25.

[3] A. Asthana, S. Zafeiriou, S. Cheng, and M. Pantic. Incremental face alignment in the wild. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1859–1866, June 2014.

[4] M. Biehl, D. Matsumoto, P. Ekman, V. Hearn, K. Heider, T. Kudoh, and V. Ton. Matsumoto and ekman's japanese and caucasian facial expressions of emotion (jacfee): Reliability data and cross-national differences. *Journal of Nonverbal Behavior*, 21(1):3–21, Mar 1997.

[5] Coolest Project Fair. How we built our facial recognition ferris wheel, 2018. https://create.arduino.cc/projecthub/Spivey/how-we-built-our-facial-recognition-ferris-wheel-dfa150, Last accessed on 2019-04-25.

[6] C. Darwin. *The expression of the emotions in man and animals*. New York ;D. Appleton and Co.,, 1916. https://www.biodiversitylibrary.org/bibliography/4820 — Includes index.

[7] W. Ding, M. Xu, D. Huang, W. Lin, M. Dong, X. Yu, and H. Li. Audio and face video emotion recognition in the wild using deep neural networks and small datasets. In *Proceedings of the 18th ACM International Conference on Multimodal Interaction*, ICMI '16, pages 506–513, New York, NY, USA, 2016. ACM.

[8] B. Fasel. Robust face analysis using convolutional neural networks. volume 2, pages 40 – 43 vol.2, 02 2002.

[9] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, Aug. 1997.

[10] R. B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[11] D. Hoe Kim, W. Baddar, J. Jang, and Y. Man Ro. Multi-objective based spatio-temporal feature representation learning robust to expression intensity variations for facial expression recognition. *IEEE Transactions on Affective Computing*, PP:1–1, 04 2017.

[12] P. Khorrami, T. L. Paine, and T. S. Huang. Do deep neural networks learn facial action units when doing expression recognition? *CoRR*, abs/1510.02969, 2015.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[14] B. T. Lau. Portable real time emotion detection system for the disabled. *Expert Syst. Appl.*, 37(9):6561–6566, Sept. 2010.

[15] S. Li and W. Deng. Deep facial expression recognition: A survey. *CoRR*, abs/1804.08348, 2018.

[16] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollr. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, Oct 2017.

[17] D. Matsumoto and B. Willingham. Spontaneous facial expressions of emotion of congenitally and noncongenitally blind individuals. *Journal of personality and social psychology*, 96:1–10, 02 2009.

[18] D. H. Noronha, B. Salehpour, and S. J. E. Wilton. Leflow: Enabling flexible FPGA high-level synthesis of tensorflow deep neural networks. *CoRR*, abs/1807.05317, 2018.

[19] S. Ouellet. Real-time emotion recognition for gaming using deep convolutional network features. *CoRR*, abs/1408.3750, 2014.

[20] Pete Warden. Scaling machine learning models to embedded devices, 2019. https://petewarden.com/2019/03/27/scaling-machine-learning-models-to-embedded-devices/ Last accessed on 2019-04-25.

[21] R. Rosipal and N. Krämer. Overview and recent advances in partial least squares. In C. Saunders, M. Grobelnik, S. Gunn, and J. Shawe-Taylor, editors, *Subspace, Latent Structure and Feature Selection*, pages 34–51, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[23] R. Vemulapalli and A. Agarwala. A compact embedding for facial expression similarity. *CoRR*, abs/1811.11283, 2018.

[24] P. Viola and M. Jones. Robust real-time object detection. In *International Journal of Computer Vision*, 2001.

A. Contributions and Acknowledgements

In this project, almost all code, including dataset management, downloading images, sampling data for Mechanical Turk survey, merging Ground Truths, experimental framework, and training loop, and management of evaluation results was written by the author from scratch, with the following exceptions:

- The code to visualize Saliency maps was taken from the authors own submission to cs231N based on the Jupyter Notebook supplied with the class.
- the initial version of the training loop was also taken from Assignment 2; however as the code became more complex, all this code was effectively removed and rewritten using TensorFlow API

The following open-source packages were used:

- TensorFlow 2.0 for model training, evaluation and management.
- Albumentations [1] for performing image augmentations on the fly.

The author thanks Anna Smagina for invaluable advice on dataset preparation, augmentation techniques, and for supplying examples of prepared datasets, and for supplying references for ”handcrafted” feature techniques.