Facial Emotion Detection via Convolutional Neural Network for Embedded Platforms

Kevin Worsley, Computer Engineering
Mentor: Dr. Mohamed El-Hadedy (Aly)
Kellogg Honors College Capstone Project

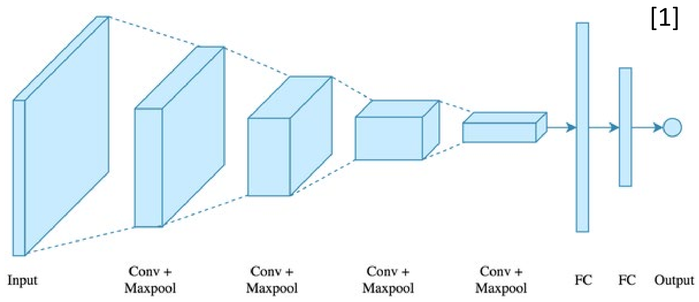
Objective

The objective of this project was to design a Convolutional Neural Network (CNN) to detect a variety of facial expressions and provide a signal to a PIC microcontroller indicating its decision. This network would run on an embedded device for portability in its enclosure.

Convolutional Neural Networks

There are a wide variety of neural network architectures used in modern computing, but a CNN is best suited to identify patterns in samples, and extrapolating those patterns to make classifications. This led to its implementation in the project as an image classifier.

[1]



A CNN will take in a predefined input, usually an image, and pass it into a **convolutional layer**. The layer will iterate over the sample, filtering groups of pixels and constructing a new sample based on these filter results. The filter is generated randomly at first, but becomes more accurate as the model trains. It then passes through an **activation**, which serves to remove any negative values that may have been generated in the convolution. Negative values indicate no pattern, so they are of no use going forward. There may be a **batch normalization**, which reduces the variance of the sample, which serves to simplify training and reduce noise. The sample then passes through a **max pooling** layer, where the generated feature map is shrunk down, reducing its dimensions. This can help reduce noise, and ensure the sample going forward contains the most relevant pattern information. This cycle is then repeated, often many times, during **training**, changing parameters to experimentally increase accuracy.

Once convolution completes, the model is **flattened**, which reduces its patterns to a **fully connected (dense)** format. This allows the model to begin training as the layers become smaller and smaller. The output layer will have units equal to the amount of classifications, and a **softmax** activation will be used. The final units represent each possibility the model is designed to detect, and softmax activation returns the most confident result.

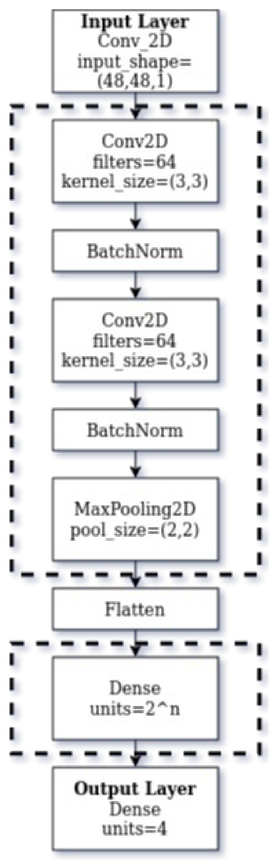
Experimental Models

For this application, 25 different models were constructed, using TensorFlow as the underlying structure, and Keras to construct the model's layers. Each model was trained and evaluated for its **loss value**, a measure of how poorly the model performed. **Accuracy** was also measured in the **training** and **validation** phases of the model. Training and validation are important to separate, as in training the model sees the same data over and over. Validation is a brand new data set, and tests the model on data it's never seen before, which gives a better idea of the model's performance in a real world situation. Model 19 is a more common CNN architecture, while Model 20 is a modified version of the Xception model, implemented with **separable convolution** [2], a divide-and-conquer approach to convolution operations.

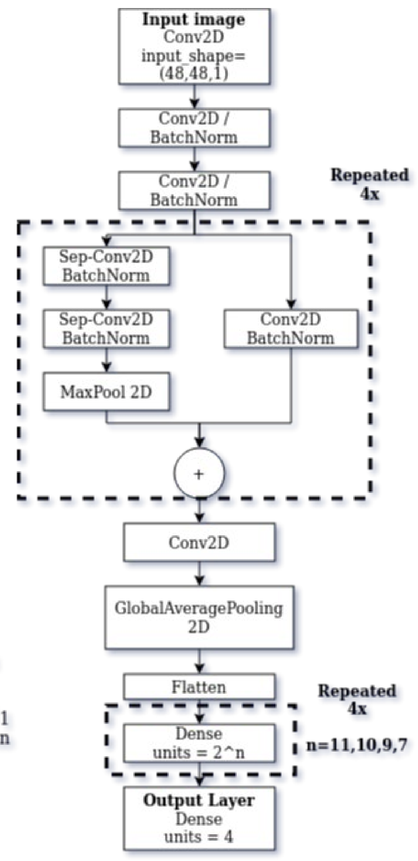
Model Results and Structure

Model Number	Train Accuracy	Validation Accuracy	Validation Loss
17	0.6704	0.7038	0.7588
19	0.7100	0.7295	0.6985
20	0.6462	0.9146	0.6328

Model 19



Model 20 [2]



Classification Results

Grey images are what the model receives, in the 48x48x1 greyscale format. Model 19 was used for these results.

Implementation on Embedded Device

The model, once trained, was then converted to the TensorFlow Lite format. The model's weights are compressed and converted to a lightweight format, which enables the model to be run with far less computing resources needed than its standard format, without loss of accuracy. Model 19 was chosen as the most accurate model, and was deployed on the device.

This model was implemented on the Raspberry Pi Zero W, a small embedded computing platform. The device features an ARM11 processor clocked at 1 GHz, with 512 MB of LPDDR2 SDRAM. [3] The device's small size and low power draw made it an appealing choice, as well as its full integration with a Python environment for ease of development.

The system uses OpenCV to handle camera inputs, as well as its implementation of a **Haar Cascade** face detector [4] to identify the user's face in the captured image. The resulting Region of Interest (ROI) is then preprocessed by scaling it to the appropriate size, as well as changing its color space to greyscale. The image is then invoked in the TensorFlow Lite interpreter, and a classification is generated. The classification code is then sent over to the PIC via the GPIO output pins.

Performance

The most relevant performance metric is **wallclock time**, how long it takes the Raspberry Pi to perform specific tasks with respect to the entire system, not just CPU time.

Operation	Time Elapsed (avg)
Import libraries	39.45 s
Load cascade detector	660 ms
Load TF interpreter	75 ms
Capture image	500 ms
Locate face, preprocess	1.42 s
Classification	3.68 s

We can see the most time consuming operation was the TensorFlow Lite interpreter, which achieved an admirable 3.68 seconds for the classification, given the heavy hardware constraints and complexity of its neural network.

Conclusion

The system performs with sufficient accuracy to accomplish its objective in a real world environment. It functions as a key module of a larger project, and provides a low-cost, performance-conscious solution to a common application of computer vision and machine learning implementation.

References

[1] A. Dertat, "Applied Deep Learning- Part 4: Convolutional Neural Networks," *TowardsDataScience.com*, 08-Nov-2017. [Online]. Available: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.

[2] O. Arriaga, P. G. Plöger, and M. Valdenegro, "Real-time Convolutional Neural Networks for Emotion and Gender Classification," *ICRA 2018*.

[3] M. Hawkins, "Introducing the Raspberry Pi Zero W," *Raspberry Pi Spy*. [Online]. Available: https://www.raspberrypi-spy.co.uk/2017/02/introducing-the-raspberry-pi-zero-w/.

[4] "Cascade Classifier," OpenCV. [Online]. Available: https://docs.opencv.org/3.4/db/d28/tutorial cascade classifier.html.