# Facial Expression Classification Using PyTorch & Deep Learning

By Kevin Wong

CS4210 - California Polytechnic State University, Pomona

## ABSTRACT

This project explores the application of deep learning models for automatic facial expression classification. Using the dataset from a Kaggle competition, I developed, trained, and optimized a convolutional neural network (CNN) using PyTorch to classify facial expressions into three categories: Angry, Happy, and Neutral. I experimented with various techniques like data augmentation, Leaky ReLU activation, and dropout. These and more helped me achieve an accuracy and Kaggle public score of **0.82244** on the test set. This poster discusses the model design, training process, and methods that I used and altered to complete this assignment.

## INTRODUCTION

Facial expression classification is a challenging problem with significant uses for human-computer interaction, emotion analysis, and various applications in entertainment and customer service. Accurately recognizing and categorizing human emotions through facial expressions can greatly enhance user experiences and enable more intuitive interfaces for companies to develop and improve upon.

CNNs excel at getting hierarchical features from images, however, achieving high accuracy is still difficult because of the small differences in facial expressions and the risk for overfitting on limited datasets. This project improves a CNN model for facial expression classification by using a wide range of techniques to boost performance and accuracy.

## OBJECTIVES

1. Develop a CNN model for classifying facial expressions into categories: 0-Angry, 1-Happy, and 2-Neutral.
2. Improve model performance through data augmentation and regularization techniques.
3. Evaluate the model's accuracy and generalization on the test set, then continue optimizing until a satisfactory score is reached.

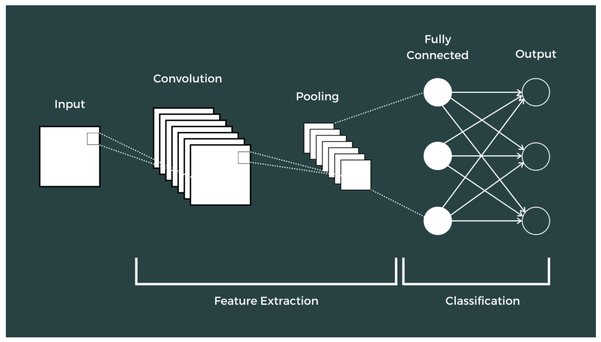
## MATERIALS AND METHODS

For this assignment, we were given a dataset containing 16,175 training examples and 3,965 test examples, each representing a 48x48 image of a facial expression.

The CNN architecture consisted of four convolutional layers followed by batch normalization, Leaky ReLU activation functions, and a dropout layer to prevent overfitting.

Data augmentation techniques such as random rotation and horizontal flip were applied to improve model generalization.

The model was also trained using the Adam optimizer with a learning rate of 0.001.

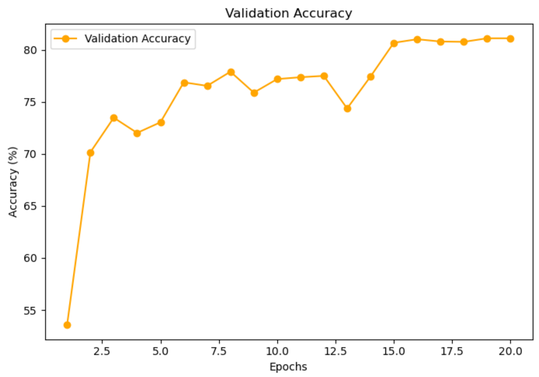



**Improvements from Previous Models:**

- **Leaky ReLU**
  - After doing research about the best activation functions for image classification, I chose ReLU, but my final test accuracy was still under the baseline score of 0.76696. I read that a common error is the "dying ReLU problem", so I decided to implement the Leaky version, I then saw minor improvements the next time I ran my notebook.
- **Transforms**
  - After looking at Week 4's Python Notebook 11 from this course's Canvas page, "11 - Dataset and Transforms", I noticed that I did not include transforming in my own, so I augmented the data with flips and rotations on the training data, and normalizations on both the training and testing data.
- **Gradient Clipping**
  - I researched other common issues with low accuracy on the training model, so after reading a blog from Neptune.ai, I learned that with very deep neural networks, gradients can become too large and "explode" , so gradient clipping caps the gradients at a maximum value.
- **Increased Batch Sizes**
  - Another area I saw that I could improve on was the batch size, because when they're too small, it may cause instability in training, and if they're too large, it could lead to poorer generalization. Firstly, I had a batch size of 64, then changed it to 32, but the accuracy dipped, so I then chose a mini-batch size of 128, and my accuracy increased slightly.
- **Dropout**
  - In Lecture 8 – Neural Networks (IV) from Week 4, we learned about how dropout changes the network's structure, prevents overfitting, and generalizes better to testing data.

## RESULTS

The final iteration of my model achieved a validation accuracy of **81.11%** after **20** epochs. The final test set accuracy I achieved on Kaggle after 10 submissions was a score of **0.82244**. I believe that my choice to include data augmentation, an optimal activation function, and tweaks to my batch sizes significantly improved the model's generalization capabilities and accuracy.





Final Kaggle Score


## CONCLUSIONS

In this assignment, I successfully implemented a CNN-based approach for facial expression classification, achieving a high accuracy that I'm very satisfied with. In future iterations, I'd like to find more ways to speed up the training process and try out pre-trained models. In summary, this project taught me the importance of using the right techniques to prevent overfitting and how the design of the model affects its overall performance. It also showed me that making gradual improvements here and there through testing and tweaking is key for getting the best results.

## REFERENCES

1. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53ref
2. https://saturncloud.io/blog/how-to-do-gradient-clipping-in-pytorch/
3. https://medium.com/@gauravnair/the-spark-your-neural-network-needs-understanding-the-significance-of-activation-functions-6b82d5f27fbf#57ea

## ACKNOWLEDGMENTS

I would like to acknowledge these blogs & posts that helped me implement:

- Gradient clipping
- Dropout
- Leaky ReLU