Emotion Classification Using AffectNet: A Database for Facial Expression Computing in the Wild

Jonathan Giguere

Table of Contents

Introduction

Background

Classifying human emotion in images is something that has been researched for decades. In the 1990's, researchers from the University of California, San Diego wrote a paper titled *Representing Face Images for Emotion Classification* that discusses extracting different features from faces and feeding those features to neural networks to classify the emotion present in the image [4]. It references research and image datasets for this task that date back to the 1970's. Although building classifiers that detect human emotion in images is not a new idea, the advent of deep learning and convolutional neural networks (CNNs) has changed how we approach the problem.

Problem Statement

Detecting a person's emotion by simply looking at them is something humans do every day. This information helps us make informed social interaction decisions. Giving this ability to a machine could improve human-computer interaction and allow for systems to better serve end users.

Aside from improving human-computer interaction, classifying emotion from images or video could have useful applications in the psychology field and beyond.

Project Scope

The scope of this project can be defined as a series of steps and is depicted below:

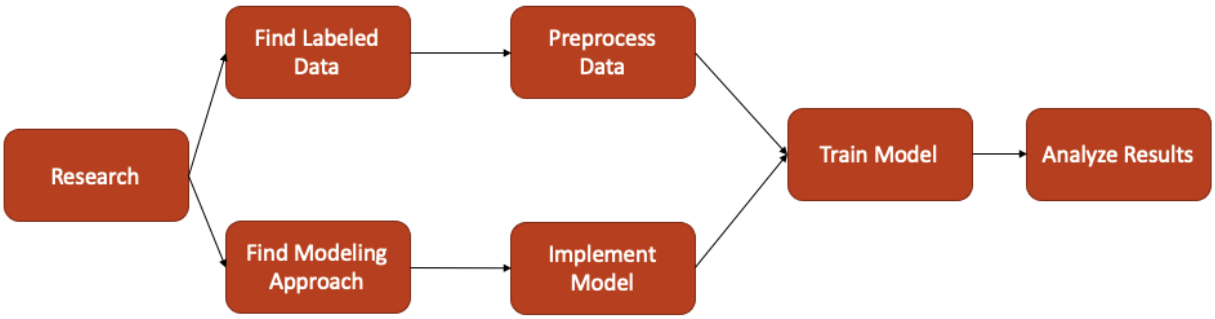

Figure 1. Diagram of Project Scope

I needed to perform research in order to ascertain what data and modelling approach I should use for this task. From there, the data needed to be preprocessed and the model(s) needed to be implemented. Finally, I performed analysis on the modeling results.

Literature Review

Research Summary

As mentioned previously, there is research that dates back to the 1970's around classifying human emotion in images. I chose to focus on three research papers for this project. One has already been mentioned and is titled *Representing Face Images for Emotion Classification* [4]. This paper from the 1990's gave me a good understanding of prior methodologies used for this task before deep learning was the de facto standard for image classification tasks. The other two papers can be applied more directly to my problem as they are from 2015 and 2017 respectively. The authors of both of these papers utilize CNN models.

Transfer Learning Paper

The paper is titled *Deep Learning for Emotion Recognition on Small Datasets Using Transfer Learning* (2015) [3]. Within the paper, researchers discuss using transfer learning with a pre-trained CNN to classify images into categories Angry, Disgusted, Fearful, Happy, Neutral, Sad, and Surprised. Some examples of images from their dataset are given below:

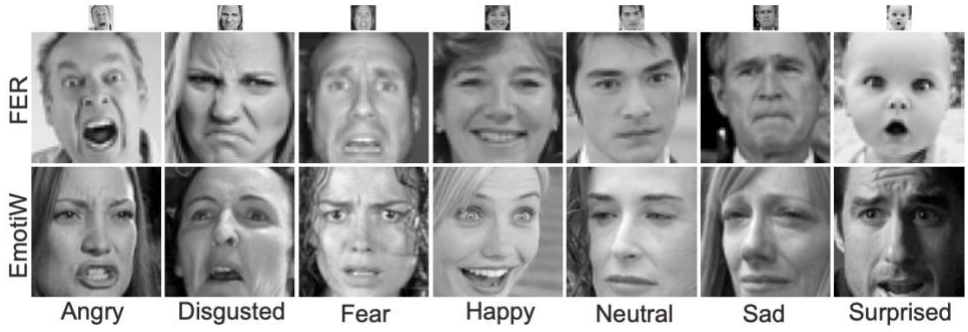

Figure 2. Examples of Image Data Used in this Paper

The researchers actually used two labeled datasets: FER-2013 and EmotiW. The FER-2013 images were captured by searching the internet and the EmotiW images were captured from movie scenes. Combined these datasets have 33,800 labeled gray scale images. Researchers used these images to fine-tune a pre-trained CNN in various ways. In the end, researchers had a model that performed with 55.6% accuracy on a test set.

As we will see later, I found a much larger dataset than what was used for this research. My thinking after reading this paper was to use transfer learning with significantly more data might yield better results.

AffectNet Paper

AffectNet: A Database for facial Expression, Valence, and Arousal Computing in the Wild (2017) [2] describes the creation of a dataset consisting of one million labeled images of human faces. The labels in the dataset include discrete emotion labels as well as valence/arousal labels. The

discrete emotions labeled in the dataset can be seen below alongside the number of samples for each category:

Table 1. Number of Annotated Images in Each Category

Expression	Number
Neutral	80,276
Happy	146,198
Sad	29,487
Surprise	16,288
Fear	8,191
Disgust	5,264
Anger	28,130
Contempt	5,135
None	35,322
Uncertain	13,163
Non-Face	88,895

Looking at valence and arousal is a different way to think about categorizing human emotion. It puts human emotions on a continuum represented by a coordinate plot with two axes: one for valence and one for arousal.

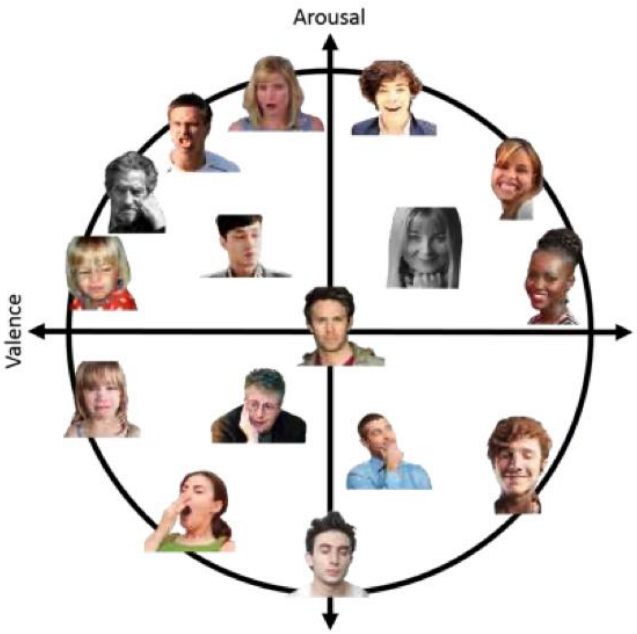

Figure 3. Arousal/Valence Visualized

The image data was collected from popular search engines by searching emotion keywords in multiple languages and 420,299 of the one million images were manually annotated.

The researchers at the University of Denver who put this dataset together also performed some baseline modeling with the popular CNN architecture AlexNet. I directly compare my modeling results to these baselines later in this report.

Methodology

Dataset Description

As mentioned previously, AffectNet consists of one million labeled images. I chose to focus on only the 420,299 manually annotated images. I also simplified the task by limiting the number of emotions the model had to classify. I chose to focus on six emotions: Neutral, Happy, Sad, Surprise, Fear, and Anger.

Table 2. Number of Annotated Images in Each Selected category

Expression	Number
Neutral	80,276
Happy	146,198
Sad	29,487
Surprise	16,288
Fear	8,191
Disgust	5,264
Anger	28,130
Contempt	5,135
None	35,322
Uncertain	13,163
Non-Face	88,895

→

Expression	Number
Neutral	80,276
Happy	**146,198**
Sad	29,487
Surprise	16,288
Fear	8,191
Anger	28,130
Total	308,570

We can see that the Happy images make up 47.4% of my dataset. I will need to deal with this class imbalance in the future. To give an idea of what the images in the dataset look like, some examples are given below:

Figure 4. Examples of Image Data in AffectNet

We can see that most of the images are close up and cropped to fit the human faces. We also observe that there are Non-faces in the data. I ignored this and other categories in my analysis (refer to Table 2 for excluded categories).

Data Preprocessing

Before any preprocessing could occur, I had to request access to the data from Dr. Mohammad H. Mahoor, Ph.D. at the University of Denver. He gave me access to a OneDrive that contained all of the images in various folders. I manually moved the compressed folders containing the images to my personal Google Drive. From there, I wrote scripts to move the data to a virtual machine with a GPU. Within the virtual machine, all images were moved to the same directory and all image filenames were stored in a Python list. This allowed me to create a custom generator to load training batches into memory. The custom generator converts the files in each batch to a matrix representation with shape (224, 224, 3) to be fed to a CNN.

Class Imbalance

As mentioned previously 47.4% of the images I am using are labeled as Happy. The authors of the AffectNet paper dealt with this imbalance in two ways most effectively: they tuned hyperparameters to a perfectly balanced validation set with 500 samples from each emotion category and they implemented a weighted cross entropy loss function. I borrowed both approaches for my modeling task. A look at the train, validation, and test split I used for training my best model is given below:

Table 3. Train, Validation, Test Split for Best Model

	Neutral	Happy	Sad	Surprise	Fear	Anger	Totals
Train	74,070	132,972	25,816	13,939	6,310	24,652	277,092
Validation	500	500	500	500	500	500	3,000
Test	804	1,443	273	151	68	267	3,006

To implement a weighted loss function I discovered a paper titled *Class-Balance Loss Based on Effective Number of Samples* authored by researchers at Cornell and Google. It introduces a hyperparameter ß that can be tuned between the values 0 and 1. If ß = 0, there is no weighting. If ß = 1, there is inverse class frequency weighting. The models for this task all use Cross Entropy as the loss function. The loss function is simply weighted by multiplying the cross entropy loss by a factor.

$$\mathrm{CE}_{\mathrm{softmax}}(\mathbf{z}, y) = -\log\left(\frac{\exp(z_y)}{\sum_{j=1}^{C}\exp(z_j)}\right)$$

$$\downarrow$$

$$\mathrm{CB}_{\mathrm{softmax}}(\mathbf{z}, y) = -\frac{1-\beta}{1-\beta^{n_y}}\log\left(\frac{\exp(z_y)}{\sum_{j=1}^{C}\exp(z_j)}\right)$$

Figure 5. The Difference Between Cross Entropy Loss and Weighted Cross Entropy Loss

Through manually tuning the parameter and reading the paper for best practices, I settled at a value for ß equal to 0.9999. Refer to Appendix B for the weighted loss implementation.

Transfer Learning

My initial strategy was to use transfer learning to classify emotion in images. Transfer learning is a very popular technique used for computer vision tasks. Transfer learning is popular because it reduces training time because you do not have to train an entire network. The intuition around CNNs is that the lower layers in the network learn more basic features that are useful for multiple tasks while the higher layers are more specific to features related to the training data that was used. To perform transfer learning we freeze the weights in the lower layers so they cannot be trained and use data for the new task to train the higher layers.

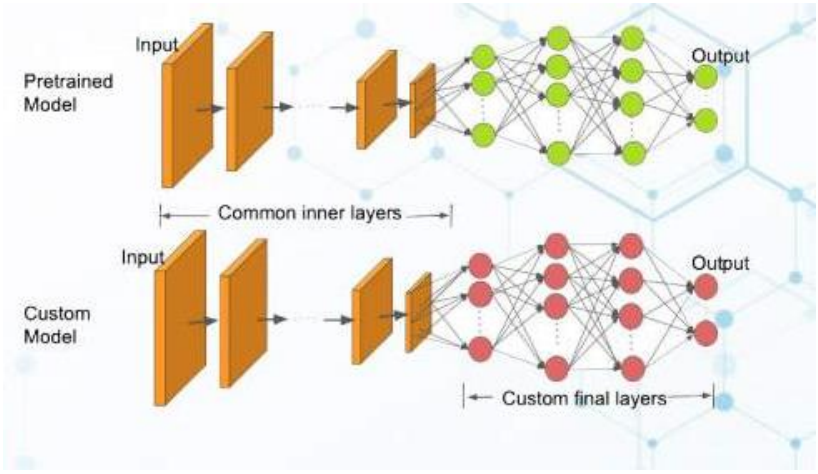

Figure 6. A Diagram Depicting Transfer Learning and Reusing Lower Layers in the Network

For my specific implementation of transfer learning I used the pre-trained ResNet-50 model offered by Keras. This network was pre-trained on ImageNet data which consists of 14 million images used to classify labels in 20,000 different categories. Following the intuition described previously, we can rely on the lower layers of ResNet-50 to be proficient at extracting basic features from our face images given the large amount of training data it has seen.

To adapt the network to my problem, I added a softmax layer with the correct number of outputs, unfroze the last 20% of layers in the network and trained with all of my available data. It is important to mention that my transfer learning model was trained with five of the six emotions I am focusing on. This model only classifies Neutral, Happy, Sad, Surprise, and Anger.

VGG Type CNN

Due to my transfer learning approach yielding lackluster results I also implemented a custom CNN. The CNN has four convolutional layers and is similar to VGG CNN architectures in which small receptive fields are used in increasing numbers as the image passes through the convolutional layers. A diagram depicting VGG-16, a similar architecture, can be seen below:

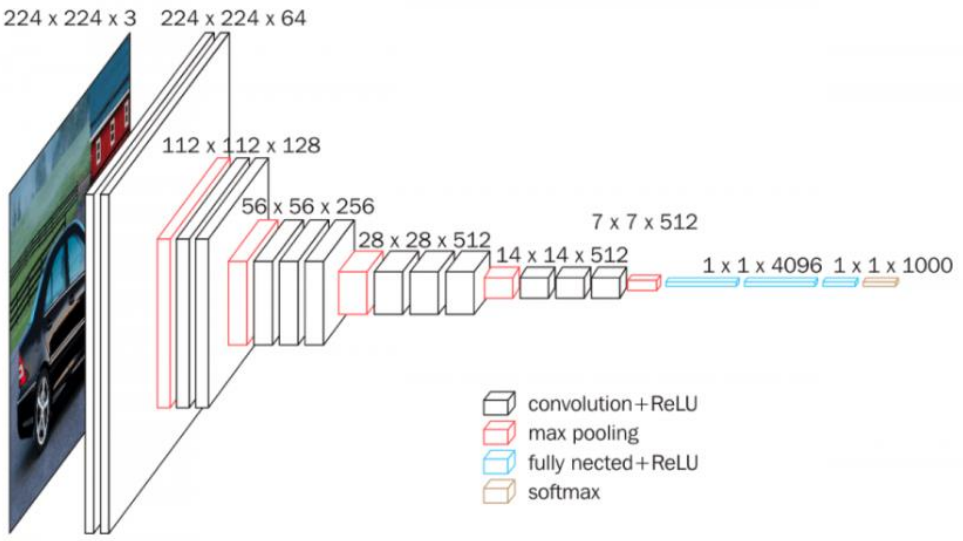

Figure 7. A Diagram Depicting How VGG-16 Works

I trained my implementation of this model with all six of the emotion categories I am focusing on and it performed better than my transfer learning approach. Refer to appendix C for the Python implementation of this model.

Results and Analysis

Transfer Learning

As stated previously, the transfer learning model did not perform as well as my custom CNN.

Table 4. Confusion Matrix for Transfer Learning Model

		Model Predictions				
		Neutral	**Happy**	**Sad**	**Surprise**	**Anger**
Actual Labels	**Neutral**	217	78	175	28	188
	Happy	54	902	118	15	135
	Sad	27	8	134	6	56
	Surprise	15	23	33	27	31
	Anger	25	15	46	3	137

The model obtained an accuracy of 57% and a weighted average F1 score of 59%. These are worse than the baseline numbers the AffectNet researchers obtained with AlexNet.

VGG Type CNN

This model gave me better results with an accuracy of 63% and a weighted average F1 score of 62%. The performance of this model is very similar to what the authors of the AffectNet paper were able to produce.

Table 5. Confusion Matrix for VGG Type CNN

		Model Predictions					
		Neutral	**Happy**	**Sad**	**Surprise**	**Fear**	**Anger**
Actual Labels	**Neutral**	561	130	27	13	10	58
	Happy	266	1,126	15	11	2	23
	Sad	137	46	55	4	8	22
	Surprise	73	24	3	33	11	4
	Fear	30	5	5	8	16	5
	Anger	90	39	19	6	7	108

To take a look at which emotions the model does best and worst at recognizing, I analyzed the recall for each category.

Table 6. Recall for Each Emotion Present

Emotion	Recall
Happy	78%
Neutral	70%
Anger	40%
Fear	23%
Surprise	22%
Sad	20%

We can see that the model correctly identified 78% of Happy faces in the test set and around 20% of Fearful, Surprised, and Sad faces. The model seems to be much better at identifying Happy and Neutral faces which is to be expected because these are the most represented classes. Future work could be done to better balance these numbers.

Conclusion

Comparing to AffectNet Baselines

When comparing results between the AffectNet baseline model and the two models I implemented, we can see that the VGG Type CNN performed best (Refer to Appendix A for full results from AffectNet Paper).

Table 7. Model Performance Comparison

Model	Baseline	Transfer Learning	VGG Type CNN
Accuracy	64%	57%	**63%**
F1-Score	55%	59%	**62%**

The accuracy is slightly lower but the increases in F1-score indicate that the VGG Type CNN might do better on classifying the minority classes. For this reason, I consider it to be a better model. It is important to mention that the baseline model was trained with all eleven of the original labels in the dataset (refer to Table 2). Which means that they had more data and arguably a more complex task.

It is also important to mention that in the AffectNet paper, the authors indicate that the annotators only agreed completely on the emotion present in 60.7% of the images. With this in mind, I was satisfied with achieving higher than 60% accuracy on the task.

Challenges

Working with such a large volume of data was my biggest challenge. The image files were too large to store on my local machine and finding the right tools to process the data required lots of iterating and research.

Training the models took about 30-40 hours for five epochs with all of my training data. This made hyperparameter tuning a very manual process as I only had so many opportunities to fully train models. If I had more computing power, I think I would have been able to iterate faster and get better values for my hyperparameters. I have a hunch that the transfer learning model in particular could get better results if I could have performed more sophisticated hyperparameter tuning.

Future Work

In the future I would like to use a CNN to model the valence/arousal labels that are in the dataset. Looking at emotion this way, on a continuum, seems to be where the field is headed.

References

[1] Cui, Y., Jia, M., Lin, T., Song, Y., & Belongie, S. (2019). Class-Balanced Loss Based on Effective Number of Samples. Retrieved 2020, from https://openaccess.thecvf.com/content_CVPR_2019/papers/Cui_Class-Balanced_Loss_Based_on_Effective_Number_of_Samples_CVPR_2019_paper.pdf

[2] Mollahosseini, A., Hasani, B., & Mahoor, M. H. (2017). AffectNet: A Database for Facial Expression, Valence, and Arousal Computing in the Wild. Retrieved 2020, from http://mohammadmahoor.com/wp-content/uploads/2017/08/AffectNet_oneColumn-2.pdf

[3] Ng, H., Nguyen, V., Vonikakis, V., & Winkler, S. (2015). Deep Learning for Emotion Recognition on Small Datasets Using Transfer Learning. Retrieved 2020, from https://www.researchgate.net/profile/Vassilios_Vonikakis/publication/298281143_Deep_Learning_for_Emotion_Recognition_on_Small_Datasets_Using_Transfer_Learning/links/56e7b18408ae4c354b1bc8d8/Deep-Learning-for-Emotion-Recognition-on-Small-Datasets-Using-Transfer-Learning.pdf

[4] Padgett, C., & Cottrell, G. (1997). Representing Face Images for Emotion Classification. Retrieved 2020, from http://papers.nips.cc/paper/1180-representing-face-images-for-emotion-classification.pdf

Appendix

A: Full AffectNet Baselines

	CNN Baselines								SVM		MS Cognitive	
	Imbalanced		Down-Sampling		Up-Sampling		Weighted-Loss					
	Orig	Norm	Orig	Norm	Orig	Norm	Orig	Norm	Orig	Norm	Orig	Norm
Accuracy	0.72	0.54	0.68	0.58	0.68	0.57	0.64	0.63	0.60	0.37	0.68	0.48
F_1-Score	0.57	0.52	0.56	0.57	0.56	0.55	0.55	0.62	0.37	0.31	0.51	0.45
Kappa	0.53	0.46	0.51	0.51	0.52	0.49	0.5	0.57	0.32	0.25	0.46	0.40
Alpha	0.52	0.45	0.51	0.51	0.51	0.48	0.5	0.57	0.31	0.22	0.46	0.37
AUC	0.85	0.80	0.82	0.85	0.82	0.84	0.86	0.86	0.77	0.70	0.83	0.77
AUCPR	0.56	0.55	0.54	0.57	0.55	0.56	0.58	0.64	0.39	0.37	0.52	0.50

B: Custom Weighted Loss Implementation

	Counts	Proportion	Effective Num	Weight Factor	Mean of Weights	Normalized Weights
Neutral	75374	0.26624702	9994.674197	0.000100053	0.000124674	0.802518525
Happy	134915	0.47656642	9999.986183	0.0001		0.802092228
Sad	25959	0.09169616	9254.309550	0.000108058		0.866721731
Surprise	14590	0.05153692	7675.483319	0.000130285		1.045004055
Fear	6878	0.02429547	4973.365369	0.000201071		1.612773364
Anger	25382	0.089658	9210.015398	0.000108577		0.870890096
Total	283098					
B	0.9999					

```
# Create custom loss function
def weighted_categorical_crossentropy(weights):

    weights = K.variable(weights)

    def loss(y_true, y_pred):
        # scale predictions so that the class probas of each sample sum to 1
        y_pred /= K.sum(y_pred, axis=-1, keepdims=True)
        # clip to prevent NaN's and Inf's
        y_pred = K.clip(y_pred, K.epsilon(), 1 - K.epsilon())
        # calc
        loss = y_true * K.log(y_pred) * weights
        loss = -K.sum(loss, -1)
        return loss
    return loss

# Set Flag for weighted or regular loss
loss_flag = 'weights'

if loss_flag == 'no_weights':
    weights = np.array([1, 1, 1, 1, 1, 1])
elif loss_flag == 'weights':
    weights = np.array([0.802518525169482, 0.802092227896231, 0.866721731456969,
                        1.0450040554419, 1.6127733638543, 0.870890096181114])
```

C: Keras Implementation of Best Model

```
model = Sequential()
model.add(Conv2D(64, (3, 3), input_shape=(256, 256, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3)))
model.add(Dropout(0.3))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Dropout(0.25))

model.add(Dense(64))
model.add(Activation('relu'))
model.add(BatchNormalization())

model.add(Dense(6))
model.add(Activation('softmax'))
```

```
# Compile model in correct mode
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001), loss=weighted_categorical_crossentropy(weights),
              metrics=['accuracy', tf.keras.metrics.Recall(), tf.keras.metrics.Precision()])

# Instantiate generators for feeding in data
batch_size = 64
my_training_batch_generator = JG_Generator(X_train_filenames, y_train, batch_size)
my_validation_batch_generator = JG_Generator(X_valid_filenames, y_valid, batch_size)

# Call the custom generators and train the model
history = model.fit_generator(generator=my_training_batch_generator,
                   steps_per_epoch=int(len(X_train_filenames) // batch_size),
                   epochs=5,
                   verbose=1,
                   validation_data=my_validation_batch_generator,
                   validation_steps=int(3000 // batch_size),
                    callbacks=[checkpoint_cb, early_stopping_cb])
```