DATS 6203 Final Project

Emotional Recognition

By: Mike Labadie and Matt Wilchek

Introduction

Human emotion is complex. We express emotions in a variety of ways. Humans also develop emotional detection models to try to detect the cues they are seeing. These models are built over the entirety of their lifetimes, and as such has become an area of interest to apply deep learning to. Today, computers have been slowly learning to do the same. Emotional Recognition has become a popular research area involving computer vision, machine learning, and behavioral sciences. It can be applied in many business areas such as security, human-computer-interaction (HCI), health care, and advertising. Solutions for emotional recognition are also actively evolving with better methods for data pre-processing and algorithm training. For our final project, we are seeking to build models that can understand human emotional cues delivered through facial expressions while applying some of the core concepts we learned in this class.

About the Dataset

The primary data that we decided to use as our foundation for applying our class concepts to emotional recognition was first collected through Carnegie Mellon University (CMU). In 2000, the Cohn-Kanade (CK) database was first established for the purpose of promoting research into automatically detecting individual facial expressions. Since then, the CK-database has become one of the most widely used test-beds for algorithm development and evaluation (Lucy, 2010). The team sought to build a robust database on which to perform comparative analyses of different predictive approaches. They realized a need to have a training set that spanned the gamut of expressions and emotions. To this end, CMU built an interdisciplinary team across psychology and computer sciences to build such a database through experiments in both 2000 and 2010.

The dataset we used is comprised of 593 image sequences from 123 subjects. Subjects were given instructions to perform a facial display associated with one of seven emotions. These emotions were:

Anger, Contempt, Disgust, Fear, Happy, Sadness and Surprise. Each image sequence started and ended with the subject in a neutral emotion state, with images capturing the subject working to and away from the “peak” emotion image (mostly near the end of the sequence).

Images were sized at 640x490 pixels and mostly came in 8 bit gray-scale. There were a couple image sequences that were included in color. The sequences varied in duration (usually from 10 to 60 frames). Our dataset is taken strictly from frontal images; though original researchers from the university also captured 30 degree angled images. In their papers, CMU often alludes to the difficulty of capturing and analyzing emotion through facial expressions. Sequences were studied by CMU's emotion researchers to validate that images were representative of the requested emotion. Some of the sequences were evaluated by multiple individuals and were asked to hand tag the emotion expressed. In many instances, taggers could not agree on the expressed emotion, and therefore left the labels for those sequence images blank. Over 200 of the sequences in the database did not have a given emotion.

Data Pre-Processing

The goal for our project was to build a predictive tool that can accurately detect emotions from the facial expressions of people in images. As previously mentioned, the dataset we used to build our modeling efforts on contained consecutive sequences of images. CMU tried to model using information per sequence by ultimately using a Support Vector Machine (SVM) model. We, however, focused on just the stills shots. While there likely is powerful predictive information that could be used in the sequences, implementation of such models is beyond what our time would allow.

Before preparing the data to be processed for modeling, a solution needed to be implemented to cover the missing classifications for the 200+ missing labeled sequences. Ultimately, the solution decided was to manually check the non-labeled sequences manually to verify a proper classification. Unfortunately, some of those manual decisions that needed to be made were quite difficult. Some of the emotions for subjects looked similar to how we perceived other emotions. For example, some of the un-labeled sequences could be classified as Fear or Surprise. Some individuals had similar facial features that could be classified to either; primarily features that involve an open mouth. Most subjects who were classified with a surprise emotion had an open mouth, but some also were classified as fear with an open mouth. This left us a possible risk for incorrectly training the model, but was still a process that we still had to ultimately implement instead of dropping and losing so many images.

Upon finishing the manual classifications, we also thought of including only a certain subset of each sequence of images. Another observation we noticed while manually reviewing the images was that most sequences had a number of pictures that displayed no emotional facial features. All of the sequences began their sequence with a neutral facial display, and then slowly evolved into what was considered to be the "peak" emotional state. By including images that displayed no distinct facial features included with a classification could also run the risk of increased error. The number of images per sequence also varied as mentioned earlier, so if a set percentage was taken from each sequence it would be possible important images would be left out that would display subtle, but distinct facial features. However, to accurately know what images to take per sequence would also mean another manual review into each sequence which would have been very time-consuming. Ultimately, a decision was made to use a percentage given the allotted time for this project. Due to the inability of being able regularize the dataset we were going to process, we also decided that during the modeling process we would include Dropout layers and Early Stopping for each epoch.

Once we finally arrived at the correct image sequences we were going to split into training, test, and validation sets, we found an additional step of processing that in the end was a huge key for boosting the performance of our model, called Facial Detection. As mentioned in our introduction, this

research area is ever-evolving. As such, using a Multi-Task Cascaded Convolutional Neural Network (MTCNN) it's possible to quickly detect a face from any image and then extract it (Brownlee, 2019). This removes lots of background noise in the facial emotion images CMU has. By implementing a MTCNN library for this purpose, we were able to pre-process our images to only focus on all facial features available which increased our maximum dimensionality to 11,664 features (Centeno, 2019). Another advantage for extracting just the faces for our images is that it allowed our model to generalize more for any new image which effectively improved our overall validation accuracy.

Modeling and Analysis

Since the CMU team only applied a SVM for their modeling efforts, we knew there could be interesting results from applying a deep learning network architecture that we learned from class. After our experience with the previous exams we instantly thought one of the best models to effectively classify images would be a Convolutional Neural Network (CNN) due to the ability to adjust kernels and strides over images to detect key features among images. Detecting facial features we figured would be critical for an overall accurate model. However, just to be sure we both conducted some initial research that led us to finding other articles describing the effectiveness for using a CNN architecture, including the article that led us to using a MTCNN for data pre-processing mentioned earlier and a couple tutorials from Medium.com that broke down how to setup a CNN for facial recognition (Sharma, 2018).

After implementing the data pre-processing concepts reported earlier, we split the data into training, testing, and validation using the Keras data-frame Image Data Generator. The data generator also provides a handy 'flow_from_dataframe()' function that allows the identification of data sets with simple file locations and class labels. We also followed this data split between images with and without the MTCNN facial detection so that we could properly measure the improvement from using it. Subsequently creating our test split objects from the Image Generator, Keras also provides a 'fit_generator' function that uses a compiled Keras model and fits the data while using the data we split. By using the Keras 'fit_generator' function we were able to effectively implement the network in our chosen framework. The Image Data Generator also allowed some augmentation such as left or right flips, rotations, or brightness adjustments to the images, which we implemented a couple of times to see if any improvement could be measured. We primarily judged the performance of our model by using Categorical Cross Entropy which included loss and accuracy in our train and validation data sets.

Once we had an initial architecture working, we invested a lot of time refining and iterating over the original solution. We found that by using data from our MTCNN results seemed to work better with the identification of important facial features towards each emotion. Below is a depiction of how the model worked as it stepped through each image per layer and extracting smaller and smaller features that weighed more for the classification of the image.

The critical piece that was able to pick up the minute facial features per image were the activation functions that were being executed per layer. As such, our group performed research on alternative activation functions that could be used than the ever-common 'relu' function. In one of our earlier homework assignments that involved creating a classification algorithm and creating circular boundaries, one of the key pieces that really shaped the solution involved using the 'tanh' function which gave the right curved boundaries to create a circular boundary line. By recalling using this activation for neural networks, it lead to the discovery an important Stanford University article that described the importance of different activation functions for CNN architectures for the purpose of facial recognition (Johnson, 2019). With the possibility that our model could improve by implementing other activation functions, we developed a Hyper-parameter tuning script to help accomplish this.

The tuning script developed was primarily executed by using a unique framework called 'Talos.' The script was capable of running hundreds of possible models with different configurations. A dictionary of parameters drives the possible parameter configurations and for us included different learning rates, number of epochs, dropout rates, number of neurons, optimizers, and of course different activation functions. We initially placed the dictionary of possible configurations universally throughout the model we made, but realized this was a mistake since we had five layers of architecture; one input, one output, and three hidden. After revising our tuning script so that the parameters could be different for each respective layer our model was able to greatly improve based on the results which would output a table of possible configurations like below.

model_id	round_epochs	val_loss	val_accuracy	loss	accuracy	activation_1	activation_2	activation_3	activation_4
69	30	1.954847	0.044177	1.944651	0.142953	elu	relu	tanh	tanh
280	6	1.951149	0.056225	1.948308	0.140281	relu	tanh	elu	tanh
242	9	1.963848	0.056225	1.934212	0.215765	elu	elu	elu	tanh
1906	10	2.088284	0.056225	2.180749	0.178357	elu	relu	tanh	tanh
247	7	2.991711	0.056225	3.086211	0.170341	tanh	elu	elu	relu

activation_5	batch_size	dropout	epochs	kernel_initializer	last_activation
relu	256	0.3	30	uniform	softmax
relu	256	0.3	20	uniform	softmax
relu	256	0.2	30	normal	softmax
tanh	512	0.25	30	uniform	softmax
tanh	512	0.4	30	uniform	softmax

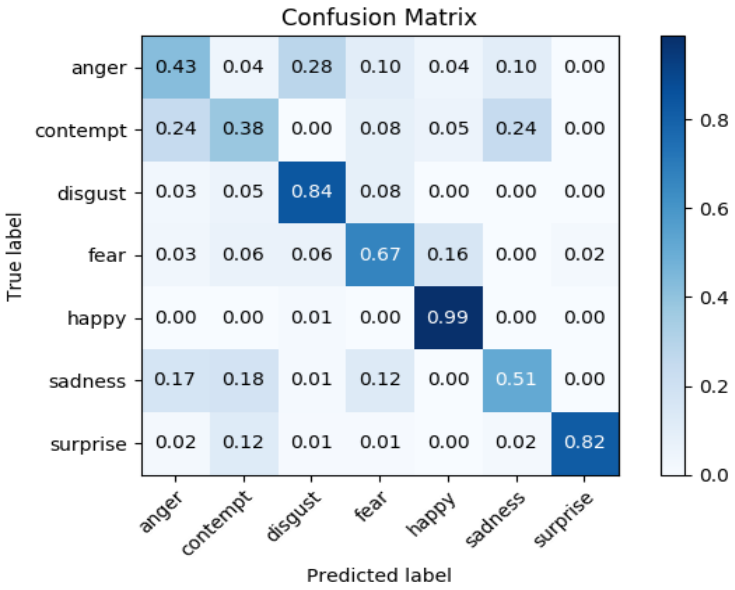
loss	lr	neurons_layer_2	neurons_layer_3	neurons_layer_4	optimizer
categorical_crossentropy	0.0001	64	256	32	keras.optimizers.SGD
categorical_crossentropy	8.00002	64	256	256	keras.optimizers.Adam
categorical_crossentropy	7.00003	64	64	32	keras.optimizers.RMSprop
categorical_crossentropy	2.00008	128	256	32	keras.optimizers.RMSprop
categorical_crossentropy	6.00004	32	128	128	keras.optimizers.Nadam

However, this hyper parameter tuning process had two huge downsides. One, was that each tuning execution would ultimately execute from 2000 to 4000 different Convolutional Neural Networks with different parameters. This in turn, often made the execution of this script take from 10 to 12 hours to complete. Therefore, carefully timing the number of models to perform, number of epochs, and batch sizes were absolutely critical for the script to finish successfully and within time constraints to review. Second, was that as we refined our final model that we presented in our findings often times we would also be implementing a new data-preprocessing concept that would be fed into each tuned model that based on previous tuned results. This caused in adjusting the tuning to start all over again on the different pre-processed dataset. The first tuning restart occurred when we decided to adjust the Keras Image generator to include small augmentations. The second time was after we decided to implement the MTCNN Facial Detection to the original images. In the end, due to the complexity increase of the original dataset each epoch for each model also became more complex often times increasing in execution time. Near the end of this project most models would complete between 15-30seconds per epoch.

Once, we completed hundreds or thousands of tuned model iterations, we then selected the best one based on the highest validation accuracy and loss. We developed a separate model evaluation script that used an additional held out dataset and used a Keras evaluation generator/predict generator to measure how the final model would perform. Per our tuned models, we were able to successfully obtain a validation accuracy of up to 70% for our best model, which is detailed in depth for the following section.

Results

Our first model was based on an example we saw from Keras's own website. It was a CNN model built to perform image recognition on the Cifar10 dataset. This example/baseline model had 21% accuracy. As mentioned, we then searched thousands of models to find parameters with higher performance. The best we found through this tuning process had 48% accuracy. Our final step forward in performance came through training our model on images limited to just faces. The tuned model using face detection had 70% accuracy.

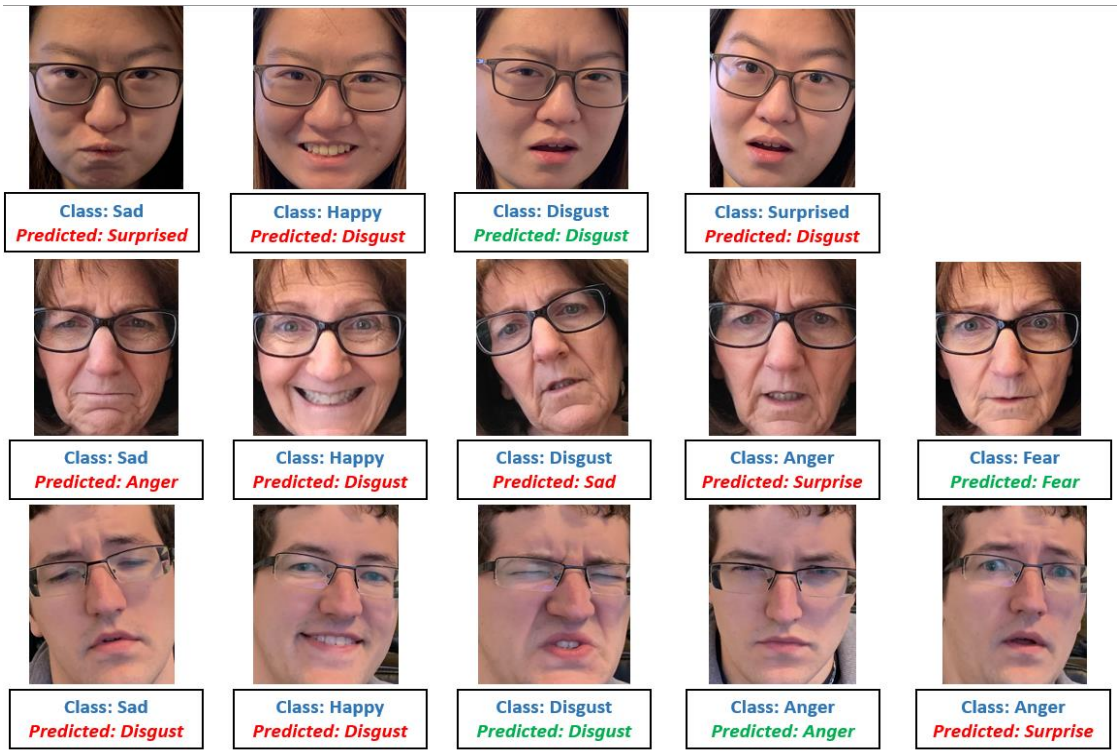


The confusion matrix above proves the accuracy across each class of emotion for the best model. Our best model had poor accuracy with the anger and contempt emotions. We did not find this surprising. We personally tagged over 200 images. We had a great deal of difficulty delineating the two emotions. The potential inaccuracy of our own tagging compounded the difficulty of distinguishing between two emotions. We had very good results identifying happiness and surprise. These two emotions are very relatable to humans. These subjects probably had far less difficulty expressing these emotions. We certainly found it very easy to assign the happy emotion during manual tagging.

Classification Report

	precision	recall	f1-score	support
anger	0.53	0.43	0.47	68
contempt	0.29	0.38	0.33	37
disgust	0.54	0.84	0.66	37
fear	0.65	0.67	0.66	63
happy	0.85	0.99	0.91	84
sadness	0.67	0.51	0.58	72
surprise	0.99	0.82	0.90	100
micro avg	0.69	0.69	0.69	461
macro avg	0.65	0.66	0.64	461
weighted avg	0.71	0.69	0.69	461

The classification report above also details the precision, recall, and f1-score of our best model across each emotion. As stated, the scores for anger and contempt are significantly lower across all measuring categories. Despite the above performance we also attempted to test our model against images from family and friends, the results of which can be seen below.



The test was fun to do, but the results were a bit disappointing as expected. Part of the results may have been from our glasses. Just from a quick glance one can see some of the different facial features we interpret the labeled emotions to be. Some mouths open, some not, some with a furrowed brow, some not, etc. We were happy to at least get one classification correct though for each person, which proves that our model has at least some merit for working as intended. If time allowed, we may have been able to refine the model more, change the architecture or create a better trained dataset to improve those results.

Conclusion

Overall we were very impressed with the effort Carnegie Mellon undertook. This was an extensive project with a noble goal. The robust effort was boosted by a multidisciplinary team they brought together. Due to their data collection, other organizations both academic and private are looking at this research as a baseline to develop more advanced tools for emotional recognition. In their initial research paper, it was suggested that in order to have a strong model there dataset would need to be significantly bigger. After working with just this dataset from them to train a model, we would have to agree to that based on the initial evidence. Upon manually reviewing almost every single image in CMU's dataset, we've realized that people have many different ways to express the same type of emotion we can think of to be common. For example, some subjects would be angry with a frown or grinned teeth, while others, mostly men from research, would show more neutral mouth shapes, but more intense stares and slightly puffed cheeks to show their frustration. To see such small differences in facial features made us realize that this task alone would be more difficult than we initially thought about.

If more time was available to work on this project, especially not during a major holiday, we believe we could have made a better model with a better dataset backing it. Once we discovered the power of MTCNN facial detection, we realized that the training was definitely picking up more important facial features. The 'tanh' activation functions were also very important to obtain the different curves around a mouth or forehead. More data could have easily been collected or scraped from google or we could have asked for more data from our friends, classmates, and family. Carnegie Mellon University also developed another open source facial recognition framework called, OpenFace. The framework leverages an immense database of facial images hosted by Amherst University called "Labeled Faces in the Wild" which could also have been potentially used to strengthen our data (Amos, 2016).

In terms of improvement to the neural network architecture, it would also have been interesting to test other types of networks such as some of the pre-trained models from PyTorch like ResNet or other Recurrent Neural Networks (RNN). Since we didn't draw much insight from the sequence changes over time, a dynamic neural network might make more sense to test with such as a Long Short Term Memory (LSTM) model. After witnessing the model testing on just a few images from people we knew it was fun to see some of the classifications actually working; which proves that the concept is still there and emotional recognition is entirely possible. We were skeptical if the sequence of images was important in order for our model to perform well, but we do think having images with more subtle facial features for certain emotions would be important. As this report mentioned in the beginning, this field of research is constantly growing, and the analysis and implementation of applying our Keras CNN model is just barely touching the possible power and real solution for facial recognition technology. As

such, the experience gained from conducting this project will become invaluable to us for touching the forefront of the Deep Learning field.

References

Amos B., B. Ludwiczuk, M. Satyanarayanan, "Openface: A general-purpose face recognition library with mobile applications," CMU-CS-16-118, CMU School of Computer Science, Tech. Rep., 2016.

Brownlee, Jason, "How to Perform Face Recognition With VGGFace2 in Keras", Machine Learning Mastery. Website. June 5, 2019.

Centeno, Iván de Paz, "MTCNN", GitHub Project Repository. Website. November 14, 2019.

Johnson, Justin, "Commonly Used Activation Functions," Stanford University. Website. Nov 26, 2019.

Lucey, P., J.F. Cohn, T. Kanade, J. Saragih, Z. Ambadar and I. Matthews, "The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression", in the Proceedings of IEEE workshop on CVPR for Human Communicative Behavior Analysis, San Francisco, USA, 2010. Website.

Sharma, Nishank, "How to do Facial Emotion Recognition Using a CNN?" Medium.com. Website. December 24, 2018.

Vijayabhaskar, J., "Tutorial on Keras flow_from_dataframe," Medium.com. Website. September 21, 2018.

Keras Example using CNN. (2019, November 6). Retrieved from https://keras.io/examples/cifar10_cnn/

Keras Modeling Documentation. (2019, November 6). Retrieved from https://keras.io/models/sequential/

Keras Documentation on ImageDataGenerator. (2019, November 6). Retrieved from https://keras.io/preprocessing/image/

Rosebrock, Adrian, "Keras ImageDataGenerator and Data Augmentation," PyImageSearch. Website. July 8, 2019.

OpenCV Documentation. (2019, November 6). Retrieved from https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_image_display/py_image_display.html

Sklearn Documentation. (2019, November 6). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

Code Appendix

This is a description of our coding products for this project. The order represents the logical order needed to replicate our work. All code can be located in our GitHub repository here.

Configuration

This file represents a centralized area for settings (i.e. file paths, naming, etc) used throughout the project.

- **configuration.py**: contains settings (i.e file paths, naming, etc) used throughout the project.

Preprocessing

These scripts were used to create datasets and metadata for training our models.

- **preprocessing_get_sequence_labels.py**: creates initial datasets built by finding all images and their associated class labels.
- **training_split.py**: splits the dataset into a training and testing set.
- **preprocessing_helper_face_detection.py**: creates new images with extracts of just the subject's face.

Modeling

These scripts represent the progression of our modeling efforts.

- **training_model_building.py**: first model, as taken from Keras's website.
- **training_model_tuning.py**: performs hyperparameter tuning.
- **training_model_building_best_model.py**: implementation of the best model found in tuning.
- **training_model_building_best_model_just_faces.py**: implementation of the best model, trained on the face extract images.

Evaluation

These scripts were used to evaluate the quality of our trained models.

- **model_predict.py**: loads saved best model and tests classification on personal friend images
- **model_evaluation.py**: evaluates the three models on the hold out set.
- **model_evaluation_helpers_mlabadie.py**: implements a confusion matrix as given by sklearn.