Spring 2022 UC Berkeley Data Discovery Project: Building a Speech to Text Model For People With Dementia

Authors: Vinh Bui, Lilly Liu, Hazel Heo

Introduction

Speech is essential to humanity; without our primary form of communication and ability to express feelings and thoughts, society would fall apart. Speech differentiates humans from animals, and is one of the keys to self-expression and agency. For those affected by Dementia, it becomes difficult to find the right words, understand what others say, and put together sentences—thereby impacting how patients see themselves and communicate with others. “About 1 in 9 age 65 and older (10.7%) has Alzheimer's” and “more than 6 million Americans are living with Alzheimer's. By 2050, this number is projected to rise to nearly 13 million” as the number of older Americans rises [1]. With such a large and growing population being impacted by Dementia, it is imperative to develop technologies to aid them.

Mentia's signature project DevaWorld is currently the leading technology of digital therapy for those with Dementia or who are Cognitively Impaired. DevaWorld is an interactive virtual world that is accessed through an app and played on an iPad with the assistance of a caregiver. Inside DevaWorld, the user is guided through different options such as playing the piano, watering plants, or eating virtual chocolates with the avatar Julie.

One of the most frustrating parts of Dementia is losing one's sense of autonomy and accomplishment. DevaWorld is a source of therapeutic entertainment that restores these things. Currently, DevaWorld is always played with a caregiver; however, by implementing AI into Julie, DevaWorld will be playable without the assistance of a caregiver so they can focus on more important tasks. Our goal for this project was to create an accurate Speech-to-Text model for Dementia patients, accounting for discrepancies in speech proficiency.

Methods

(i) Accuracy testing: The crucial aspect of finding the most plausible Speech-to-Text mechanism was to assess the accuracy of each model that we utilized. The accuracy testing used in each model puts its background on string comparison, based on two different methods: Levenshtein distance and Cosine similarity. Levenshtein distance method was mainly used in order address typos, which in Speech-to-Text testing was used as a string metric to measure the difference between two sequences given. It essentially provided a minimum number of single-character edits—insertions, deletions, or substitutions—required to change the input into the labeled conversation. Cosine similarity using nltk cosine similarity function was used to print out percent proportion similarity to three decimal places. It functions to return a percent proportion of the input and the labeled data based on the cosine similarity; cosine similarity defines the angle between two vectors in n-dimensional space. It works based on a four step mechanism: 1) The

function removes punctuations from a given string, fostulates the case difference, and removes stop words. 2) The CountVectorizer function transforms each word into a vectorized matrix form. 3) Calculates the cosine similarity and compresses the matrix by reshaping them based on similarity. 4) Take each index and calculate the percent similarity. We mainly implemented the nltk cosine similarity function in most of our trials since it printed out explicit proportional similarity sequences that allowed us to compile the accuracy data.

(ii) Google Speech-to-Text: The first model we used was Google Speech Recognition using the SpeechRecognition Library. We did not create a machine learning model from scratch, this library provides us with convenient wrappers for various well-known public speech recognition APIs (such as Google Cloud Speech API). To build this speech recognition model, we created a function in order to conduct speech recognition of a long audio file. The function split the audio file into smaller chunks, transcribing each file by converting the speech into text using Google Speech Recognition. The final output of the function is a string representing the converted text.

(iii) Amazon AWS: Another model we implemented as Amazon AWS that transcribes files from the S3 storage cloud of the user bucket. We set up the S3 bucket storage and moved all the audio files to the list in order to use the hash function to automate the system. The model was structured for multiple speaker files, in order to build a reliable context model in a setting where a dementia patient's caregiver might be present. The model that we utilized detected up to 10 multiple speakers, which takes two arguments at the same time, which are the audio file name and the max speakers in order to enhance the accuracy of the function. It splits the audio input based on the pauses and returns the boolean value of the inputted transcribe job, which then results in the dictionary data type. Then the function gives you the raw transcription code without the speaker label, which is then outputted as a JSON speaker model file. We added an additional custom list of vocabularies in order to enhance the accuracy, based on the settings that the dementia patients will be experiencing through DevaWorld. Since AWS accepts specific types of vocabulary input through Python, we converted columns of custom vocabularies into a .txt file and uploaded them to the model. Here are the custom vocabularies we used (from DevaWorld prompts): ["Julie", "Robin", "Livingroom", "Bedroom", "Bathroom", "Garden", "Barn", "tackroom", "drink", "tap", "follow", "books", "movies", "sport", "travel", "fire", "light", "records", "door", "records", "painting", "piano", "window", "music", "song", "tune", "chocolates", "fan", "outfit", "closet", "bed", "carpet", "TV", "television", "cat", "couch", "recliner", "seat", "lamp", "pillows", "blankets", "clothing", "hat", "sunglasses", "medication", "pill", "music", "teeth", "brush", "pajamas", "sleep", "freshen", "clean", "toothpaste", "gums", "heater", "bathrobe", "shower", "curtain", "toilet", "washcloth", "cream", "radio", "barn", "stall", "starr", "grooming", "groom", "horse", "work", "dixie", "pasture", "touch", "dirt", "fur", "curry", "comb", "coat", "mane", "shine", "body", "face", "halter", "rope", "strokes", "fred", "dog", "rosie", "goat", "gate"]

(iv) Azure Speech-to-Text Data-Labeling: The dataset for the project is from a previous study. The data has not been labeled yet. So, we have to do the human label to fit the data into the model. The method that we choose is providing the “ground true” for the audio data, then fit a part of the dataset for training, the rest for testing. This is a good approach because the goal of the research is to build a model that can transcribe elderly person talk. There are several ways to improve accuracy of the model (ie vocabulary, human-labeled data), but for Azure we choose to use Audio + human-labeled transcript data for both training and testing. This method is time consuming. However, we still need labeled data to automatically

validate the model, so we choose to test approach on. To validate the accuracy of the model, we use the WER (Word Error Rate) formula which is already integrated into Azure Speech Studio. WER is derived from Levenshtein distance, but working at word level instead of phoneme level. The word error rate can then be computed as:

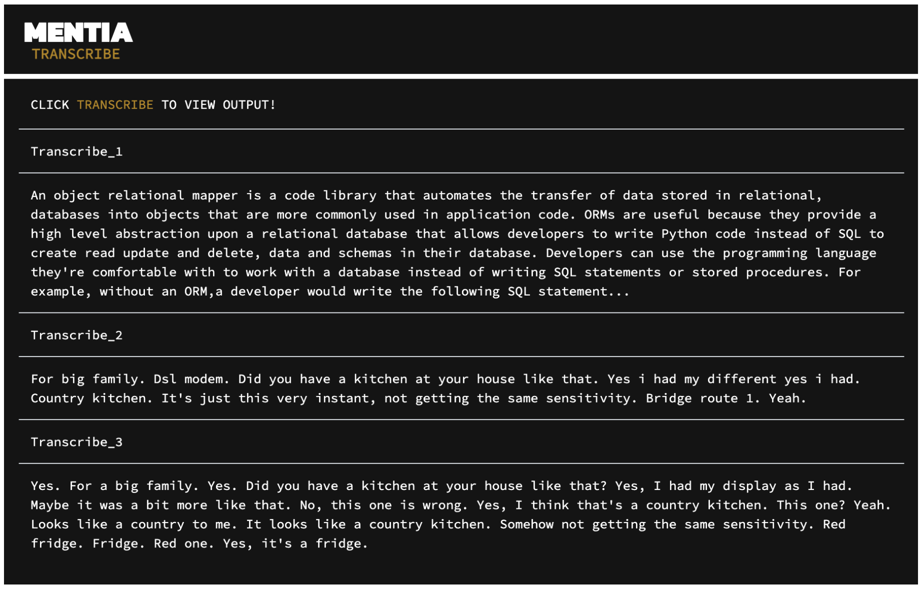
$$WER = \frac{S+D+I}{N} = \frac{S+D+I}{S+D+C}$$

- S is the number of substitutions,
- D is the number deletions,
- I is the number of insertions,
- C is the number of corrected words,
- N is the number of words in the reference (N = S + D + C)

(v) AssemblyAI: AssemblyAI was another high accuracy speech-to-text web application programming interface that we used to transcribe. To do this, we created three files that work together: upload_audio_file.py, which uploads your audio file to a secure place on AssemblyAI's service so it can be access for processing; initiate_transcription.py, which tells the API which file to transcribe and to start immediately; and get_transcription.py, which prints the status of the transcription if it is still processing, or displays the results of the transcription when the process is complete. Within the upload_audio_file.py file, we check that the file passed in exists, then use Request's chunked transfer encoding to stream large files to the AssemblyAI API. The initiate_transcription function essentially just sets up a single HTTP request to the AssemblyAI API to start the transcription process on the audio file at the specific URL passed in. The get_transcription function uses the AssemblyAI API with our API key and the transcription identifier. We retrieve the JSON response and return it.

Results

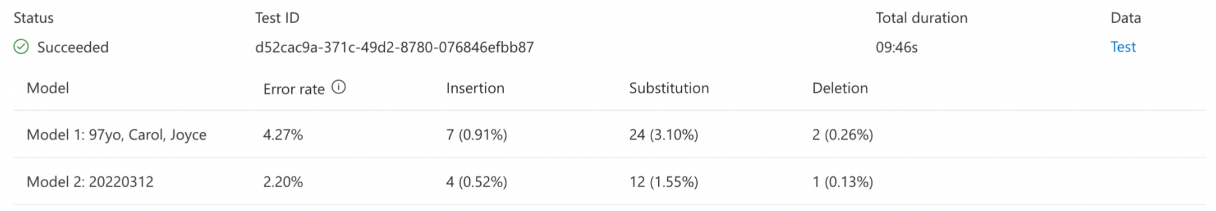
Google Speech-to-Text was inaccurate, producing an accuracy of about 82% using cosine similarity comparison. There were no available ways to boost accuracy available for free to the public, so we decided to move on to other models.
AssemblyAI's base model was more accurate than Google Speech-to-Text, with 85% precision. Using custom vocabulary based on DevaWorld prompts, the model accuracy was improved to an average of 89% on 10 files.
Amazon AWS produced accuracy that is close up to 0.734 based on the nltk accuracy that we developed, before additional custom columns of vocabulary files were assessed. After inputting the additional accuracy mediations, the accuracy increased to 0.921. We inputted around 80-100 new custom variations in order to increase the accuracy of the dementia patient transcribe model. Below is the .html front-end design that was developed that runs the Amazon AWS model when user inputs the video into the S3 bucket. This page allows the users to choose a specific video to transcribe, which is saved in the repository of the AWS server, which is saved as a code in the MongoDB AWS server. The ajax connection between the python and the front-end function allows the model to run, which directs the users to view the transcribe on the webpage.



MENTIA
TRANSCRIBE

```
CLICK TRANSCRIBE TO VIEW OUTPUT!

Transcribe_1

An object relational mapper is a code library that automates the transfer of data stored in relational,
databases into objects that are more commonly used in application code. ORMs are useful because they provide a
high level abstraction upon a relational database that allows developers to write Python code instead of SQL to
create read update and delete, data and schemas in their database. Developers can use the programming language
they're comfortable with to work with a database instead of writing SQL statements or stored procedures. For
example, without an ORM,a developer would write the following SQL statement...

Transcribe_2

For big family. Dsl modem. Did you have a kitchen at your house like that. Yes i had my different yes i had.
Country kitchen. It's just this very instant, not getting the same sensitivity. Bridge route 1. Yeah.

Transcribe_3

Yes. For a big family. Yes. Did you have a kitchen at your house like that? Yes, I had my display as I had.
Maybe it was a bit more like that. No, this one is wrong. Yes, I think that's a country kitchen. This one? Yeah.
Looks like a country to me. It looks like a country kitchen. Somehow not getting the same sensitivity. Red
fridge. Fridge. Red one. Yes, it's a fridge.
```

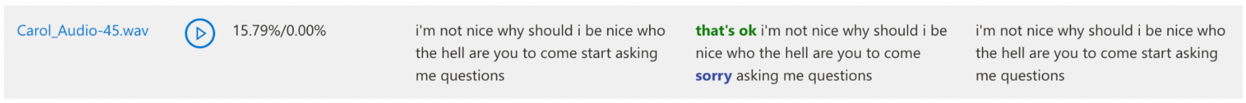
Using WER to assess the accuracy of the model, based on the limited dataset that we have at hand, the general model of Microsoft works well even without the training dataset. The error rate for the base model is 2.2% compared to 4.27% of the custom model. Generally, the base model works better across the board: insertion: 0.52% vs 0.91%, substitution: 1.55% vs 3.1%, deletion: 0.13% vs 0.26%.



Status	Test ID	Total duration	Data
Succeeded	d52cac9a-371c-49d2-8780-076846efbb87	09:46s	Test

Model	Error rate	Insertion	Substitution	Deletion
Model 1: 97yo, Carol, Joyce	4.27%	7 (0.91%)	24 (3.10%)	2 (0.26%)
Model 2: 20220312	2.20%	4 (0.52%)	12 (1.55%)	1 (0.13%)

Accuracy comparison between models



Carol_Audio-45.wav	15.79%/0.00%	i'm not nice why should i be nice who the hell are you to come start asking me questions	**that's ok** i'm not nice why should i be nice who the hell are you to come **sorry** asking me questions	i'm not nice why should i be nice who the hell are you to come start asking me questions

A test case display for visual inspection.

Discussion

For the Azure model, according to Microsoft, they "recommend that you provide word-by-word transcriptions for 1 to 20 hours of audio." However, we are only able to transcript around 10 mins of audio files due to the constraint of time. Therefore, we cannot justify the customized model until we have more labeled data.

References

Alzheimer's Facts and Figures Report. (n.d.). Alzheimer's Association. Retrieved May 3, 2022, from https://www.alz.org/alzheimers-dementia/facts-figures

Core Transcription. (n.d.). AssemblyAI. Retrieved May 3, 2022, from https://docs.assemblyai.com/core-transcription#custom-vocabulary

Makai, M. (2020, August 9). How to Transcribe Speech Recordings into Text with Python. Full Stack Python. Retrieved May 3, 2022, from https://www.fullstackpython.com/blog/transcribe-recordings-speech-text-assemblyai.html

Prepare data for Custom Speech - Speech service - Azure Cognitive Services. (2022, April 27). Microsoft Docs. Retrieved May 3, 2022, from https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-custom-speech-test-and-train

Radečić, D. (2019, October 30). Calculating String Similarity in Python | by Dario Radečić. Towards Data Science. Retrieved May 3, 2022, from https://towardsdatascience.com/calculating-string-similarity-in-python-276e18a7d33a

Word error rate. (n.d.). Wikipedia. Retrieved May 3, 2022, from https://en.wikipedia.org/wiki/Word_error_rate

Facial Emotion Recognition Model For People With Dementia

Authors: Paul Fentress, Tess Tao, Nehal Sindhu

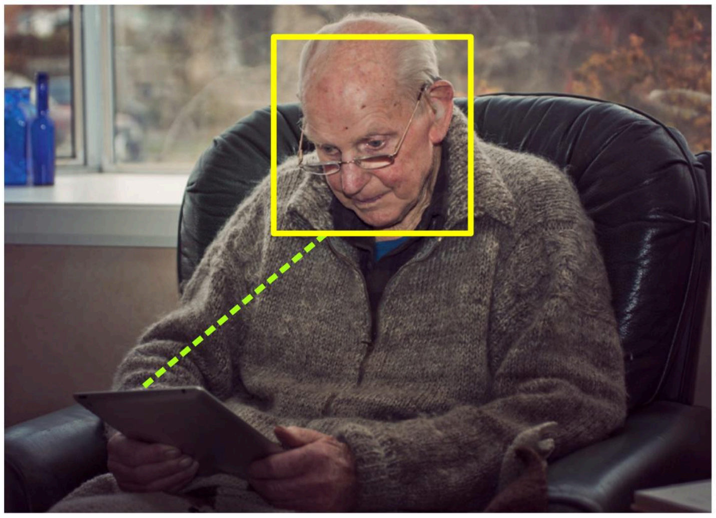
Introduction

In this project, we utilized different Deep Learning techniques including Generative Adversarial Networks (GANs), Convolutional Neural Networks, and Transfer Learning in order to classify the emotional state of Dementia patients, based on images of their faces. The purpose of classifying the emotional state of dementia patients based on images of their face, is to add another dimension of emotional intelligence into Devaworld's in-game avatar named Julie. In fall 2022 Paul Fentress and Chi Hoang built a Speech Emotion Recognition Algorithm (S.E.R.) for Julie. Mentia plans to join the S.E.R. model with this semester's Facial Emotion Recognition model (F.E.R.) and Speech to Text Model, to create a more intelligent and therapeutic dialog between Julie and the user.

Methods

Context
Before building the model, it was necessary to consider how our model would be used in production. The purpose of making this consideration before diving into the project is that if we train our model on data that does not represent how the model will actually be used, when the model is tested in the real world it will perform poorly. The plan for production is to utilize the front facing camera on a tablet/ipad to continuously record the faces of users, and collect data about their emotional state using our F.E.R. model. Then the emotional classification predictions are fed into Julie's Dialog model, which will influence what Julie says next.

Bellow is a diagram of how our model would be used in production:



Machine Learning Project Lifecycle
After considering the context of how this F.E.R. model would be used, we structured our project with the Machine Learning Life Cycle, which includes:

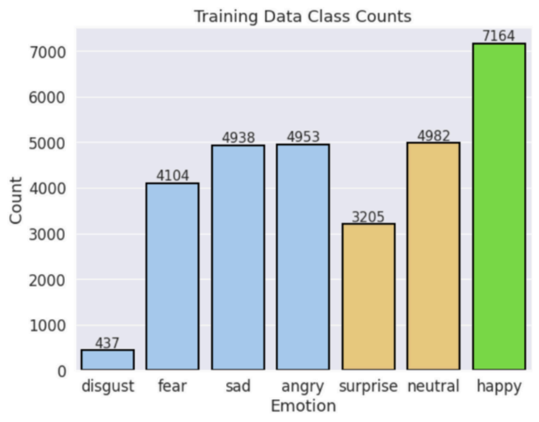
- Defining the Problem
- Data Collection
- Data Processing
- Exploratory Data Analysis
- Model Building
- Model Evaluation
- Model Deployment

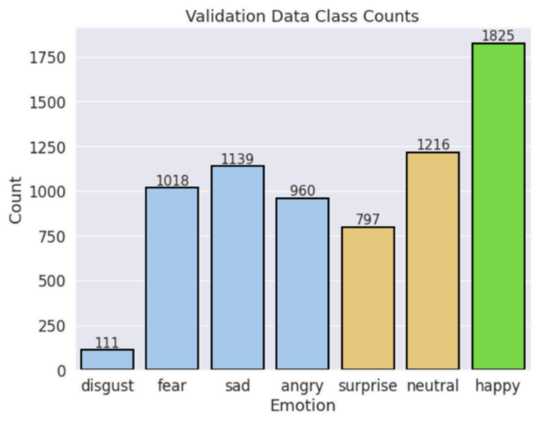
Defining the Problem

The first step in the Machine Learning Lifecycle was to define the problem we are trying to solve. Our question was “Can we build a model to classify the emotional state of Dementia patients based on images of their faces?”

Data Collection

We tried multiple approaches when building our training dataset. We were initially given over 200 videos from Mentia to work with. These videos were recorded interviews with 8 different dementia patients, while they were using the Devaworld app. These video samples were very useful for training and testing the S.E.R. model however, they were not very representative of how the F.E.R. model would be used in production. The reason is that the recordings were of two people sitting at a table, talking to each other, with a camera set up across the room capturing the conversation. The videos recorded in this fashion do not represent the data that would be collected by the front facing camera of the tablet (shown in the diagram above), and also only included 8 different faces. It was for these reasons that we decided to use the Kaggle FER 2019 dataset, which included over 36,000 samples that were more representative of data which would be collected by the front facing camera on a tablet. There were originally 7 classes including disgust, fear, sad, angry, surprise, neutral, and happy. The graphs below show each class's respective number of samples. Due to the disgust class having a small number of samples and very similar expressions as the angry class, we joined those two classes together.





There was a significant drawback of using this Kaggle FER 2019 dataset, which was that this dataset was not the faces of people with Dementia. Building a dataset from scratch with 1000's of labeled images of people with Dementia was not feasible in the time allotted for this project, so we made a compromise. We used a Generative Adversarial Network to "Age" the faces of the Kaggle dataset. This way we could transform all of the faces in our training and validation to appear older using Machine Learning, and better represent the demographics of people with Dementia.

Data Processing

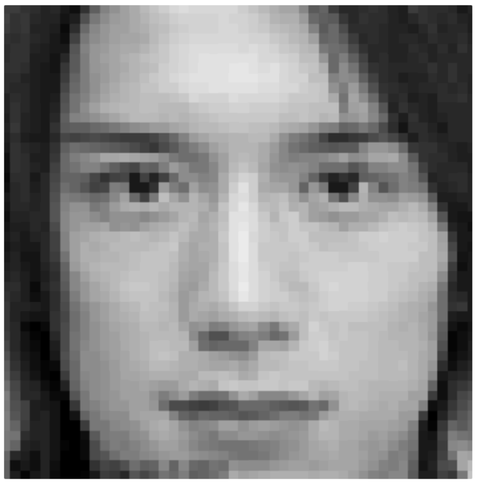
In this project, we used HasnainRaz pre-trained Fast Aging GAN to "Age" the photos in our dataset.

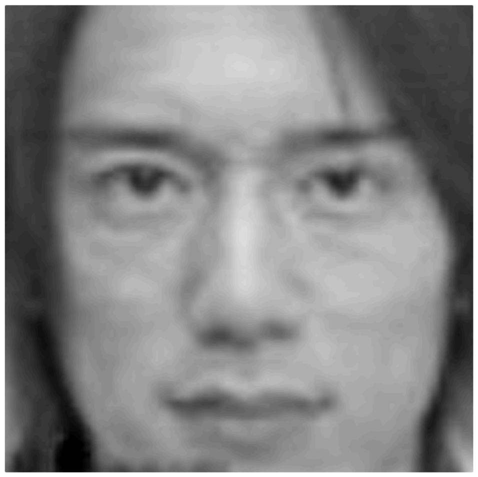
How the GAN works:

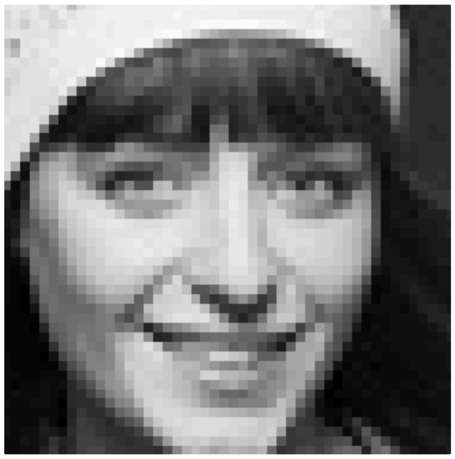
A GAN stands for Generative Adversarial Network, and it is a method of Machine Learning that is often used to create never before seen images or transform existing images. Traditional neural networks can predict the underlying distribution of some data; however, they cannot generate similar but new distributions, and that is what GANs do. GANS use two different networks which are called the **Discriminator** (a classifier), and a **Generator** (another Neural Network) which generates random but similar distributions based on the discriminator.

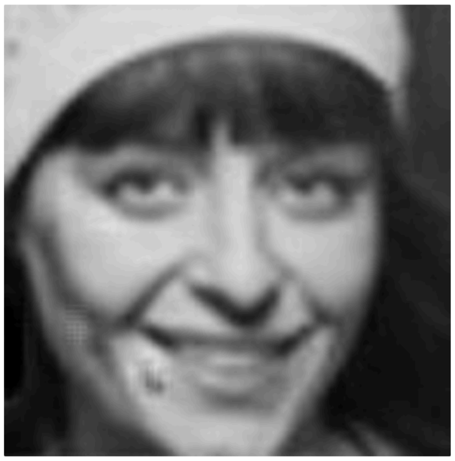
The Discriminator and Generator networks are playing a minimax game against each other where the discriminator wants to minimize loss, while the generator wants to maximize the loss. The discriminator is given an image from the real dataset or the newly generated image from the Generator, and it classifies whether or not it is a real or generated image. The Generator generates random distributions based on the underlying distribution learned from the discriminator, and the Discriminator will classify if the image is real or not. Then the Generator takes this information and learns that it was wrong, and tweaks its weights to try and trick the Discriminator. Eventually, the Discriminator network can no longer tell which images are fake vs. real and the Generator is now generating new random distributions that resemble the data distribution.

For our project, we used a pre-trained Fast Aging GAN to "Age" images, which means the Discriminator was trained to classify faces as "Aged" or "Not Aged", then the Generator added random noise to our dataset of faces based on the underlying distribution learned from Discriminator. This back and forth process was repeated until the Generator "Aged" the images enough to trick the Discriminator into thinking that the person in the photo was indeed elderly.


Original Image


Aged Image


Original Image


Aged Image

In order to use the Fast Aging GAN, we had to build custom functions and tweak the code from the original Github in order to fit our problem. This is the algorithm used in order to iteratively call the Fast Aging GAN:

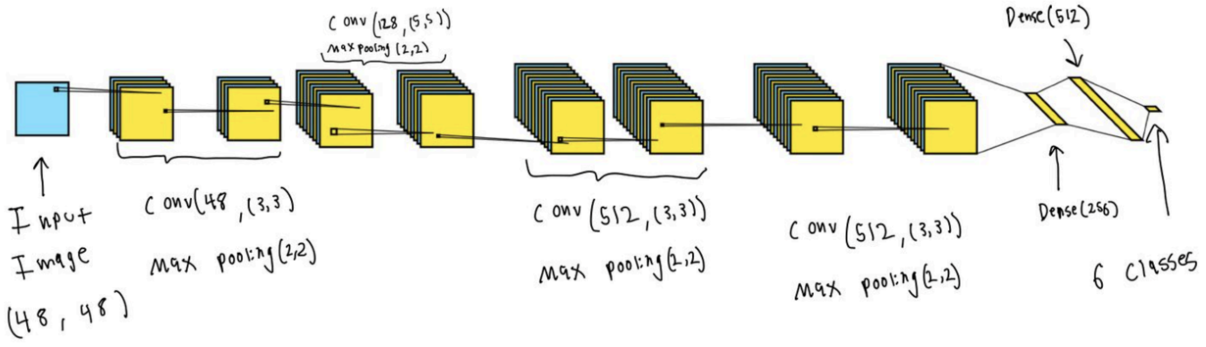
Data Processing Algorithm:

1. Reshape input data from (48, 48) → (1, 512, 512) to fit into the GAN model.
2. Pass the data into the GAN Model to "Age" the image.
3. Crop the output image because it was yielding a 3x3 grid of duplicate images. So we cropped the image from the top corner of the grid. The output image shape was (170, 170, 3) with the aging effect applied.
4. Iterate over every image within a class folder, and apply steps 1-3 above. Then route the new aged images into a new folder using OS.
5. Apply step 4 to every class folder in the training and validation datasets.

After we applied this algorithm to the Kaggle dataset, we now had an "Aged" dataset that was more representative for the problem of classifying the emotions of people with Dementia.

Model Building (In Progress)

Now that we had our data, we started building our image classification models. For the baseline model, we built a Convolution Neural Network using Keras. We chose to use this as a baseline model because it is a simple model that has worked well for image classification problems, due to the presence of the "Convolutional Layers." We used the same CNN architecture that was used by the user "akmadan" Github Repository.



This simple CNN architecture yielded a validation accuracy of 56%, so we decided to use Transfer Learning in order to build a better model.

Applying Transfer Learning

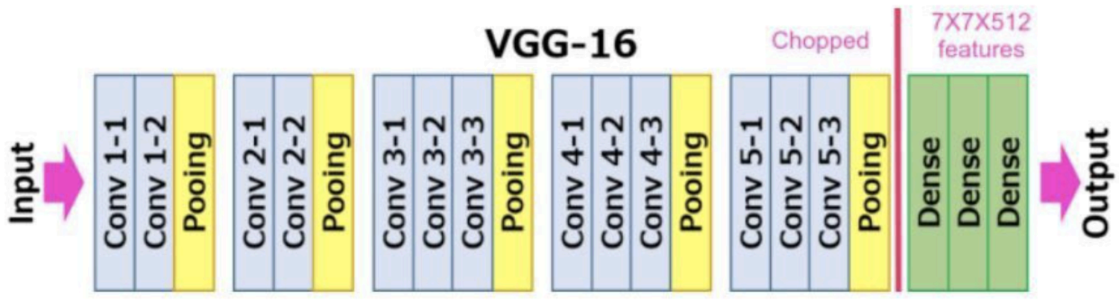
Transfer learning is one of the state-of-the-art techniques in machine learning that has been widely used in image classification. Since our task specifically targets the emotions of people with Dementia, without a huge set of relevant training data, applying transfer learning would enable us to utilize model weights that are previously trained on standard datasets such as ImageNet to improve the efficiency of our task at hand.

Such a process will have two significant advantages over using a custom-made model:

1. Speed: Using transfer learning will enable us to cut short the process of training normal CNNs for days or even weeks.
2. Accuracy: These pre-trained models have already been tuned to detect important features in the images. Although we are specifically interested in the elder population, we believe that transferable knowledge from previous emotion recognition tasks can be applied to our scenario effectively. Generally, a transfer learning model will outperform a customized model by 20%.

VGG Architecture

The VGG net stands for Visual Geometry Group, a standard deep convolutional neural network architecture with multiple layers. From the two models available in VGG, we will be using **VGG-16** to classify our dataset based on its good performance in past research. The **VGG-16** network mainly contains three parts: convolution, pooling and fully-connected layers.



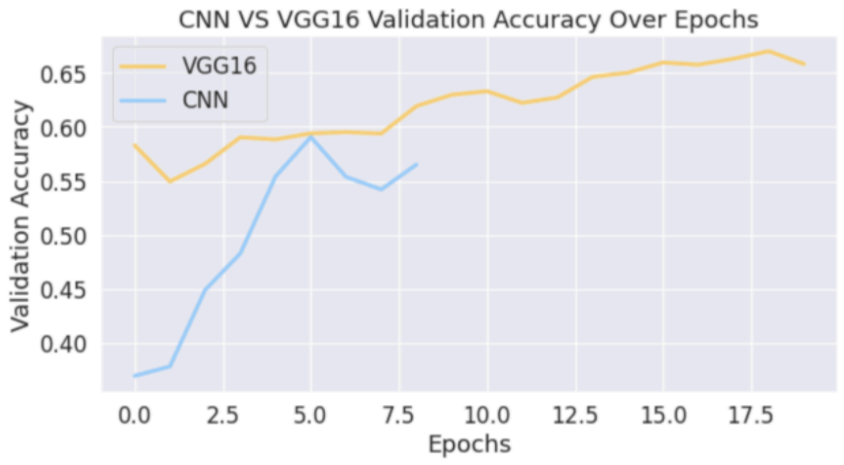
The convolution layer has filters that can extract features from images; the pooling layer reduces the size of parameters and speeds up computation in the network. After pooling, the first 16 layers of the VGG-16 can extract 25,088 features. Using pre-trained weights, we should be able to extract prominent features from the training data after convolution and pooling. The fully connected layer is a simple neural network that can be customized to give classification predictions for the number of classes, which is 6 in our case.

To fit a pre-trained model in PyTorch, we will remove the last fully connected layers and add new layers that will output six classes of emotions that we're interested in detecting. We would then freeze the parameters in the convolution and pooling layers and train the fully connected layers with our collected training data.

Results

Model Evaluation (In Progress)

After training the model for 20 epochs, we were able to obtain a validation accuracy of 67%, which outperforms the baseline CNN by 8%. As the validation accuracy exhibits an increasing trend, we believe that the accuracy could continue to increase if we have more GPU capacity to train more epochs. We plan on running the model for more epochs and testing the models properly as future work.



Model Deployment/Demo

Although our model is still in the development phase, we wanted to create a demo version in order to show what the model would look like while being used in production. Creating a demo version of your Machine Learning projects is one of the best ways to convey the message of what you are trying to accomplish. We created a .h5 model object in order to use in the demo version of the model.

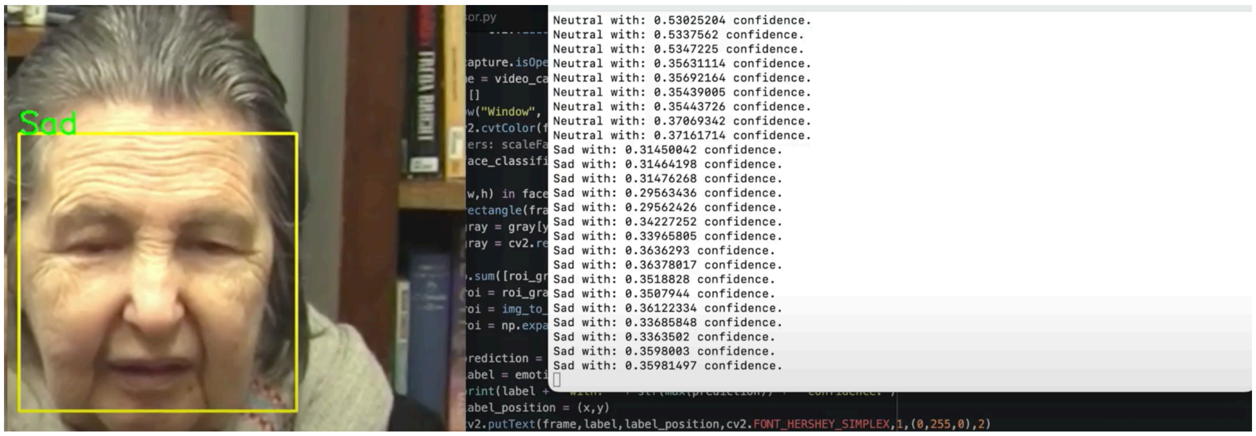
We used the CV2 Cascade classifier to detect faces coming from video data. To train the face detection classifier, the algorithm needs a lot of positive images(images that have faces in them) and negative images(images that do not have faces in them). To find faces of a specific size, we run a window that is roughly the size of the face through the entire image in a raster scan fashion. At each pixel, we will extract features and classify them as face or non face. However, this is a largely time consuming process. Therefore, we used the concept of Cascade of classifiers. Instead of applying all 6000 features on a window, the features are grouped into different stages of classifiers and applied one-by-one. If a window fails the first stage, we discard it. We don't consider the remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region.

We used a simple algorithm in order to find faces, and then make predictions:

Demo Algorithm:

1. Open Video or Webcam
2. Start a forever loop.
    a. Detect faces using pre trained CV2 Cascade face detector.
    b. Draw bounding box around detected face object.
    c. Collect data inside bounding box.
    d. Feed data from bounding box into the FER Model and make prediction.
    e. Write prediction to screen.

Below is a screenshot of our demo in action. You can see we have created a bounding box around the face, then printed onto the screen the prediction from the FER model. On the right is the terminal where we are printing the predictions and their probabilities returned from the VGG Neural Network.



```
Neutral with: 0.53025204 confidence.
Neutral with: 0.5337562 confidence.
Neutral with: 0.5347225 confidence.
Neutral with: 0.35631114 confidence.
Neutral with: 0.35692164 confidence.
Neutral with: 0.35439005 confidence.
Neutral with: 0.35443726 confidence.
Neutral with: 0.37069342 confidence.
Neutral with: 0.37161714 confidence.
Sad with: 0.31450042 confidence.
Sad with: 0.31464198 confidence.
Sad with: 0.31476268 confidence.
Sad with: 0.29563436 confidence.
Sad with: 0.29562426 confidence.
Sad with: 0.34227252 confidence.
Sad with: 0.33965805 confidence.
Sad with: 0.3636293 confidence.
Sad with: 0.36378017 confidence.
Sad with: 0.3518828 confidence.
Sad with: 0.3507944 confidence.
Sad with: 0.36122334 confidence.
Sad with: 0.33685848 confidence.
Sad with: 0.3363502 confidence.
Sad with: 0.3598003 confidence.
Sad with: 0.35981497 confidence.
```

```
abel_position = (x,y)
v2.putText(frame,label,label_position,cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
```

Discussion

After building a baseline model, and deploying the model as a demo version, we found that we can indeed build a F.E.R. model for Devaworld that detects faces and predicts emotions of people with Dementia. However, in order to measure the accuracy for specifically people with Dementia there is future work to be done. In order to truly understand how the model performs specifically for people with Dementia, we need to build and label a diverse dataset of faces from people with Dementia. By building a more representative data set, and improving our model based on this data, we will be able to better measure our performance of our model specifically for people with Dementia.

References

Project Code:
https://github.com/fentresspaul61B/Mentia_FER_Spring_2022

Data Sources:
Title: “Face expression recognition dataset”
Date of Publication: 2019
Author: JONATHAN OHEIX
Distributor: kaggle.com
URL: https://www.kaggle.com/datasets/jonathanoheix/face-expression-recognition-dataset
Version: 1

Papers/Media:
Akshit Madan “Emotion Detection using Convolutional Neural Networks and OpenCV | Keras | Realtime” https://www.youtube.com/watch?v=Bb4Wvl57LIk&feature=emb_title

Facial emotion detection using CNN. Analytics Vidhya. (2021, November 3). Retrieved May 3, 2022, from https://www.analyticsvidhya.com/blog/2021/11/facial-emotion-detection-using-cnn/

Liu, Y., Wang, Z., & Yu, G. (1AD, January 1). *The effectiveness of facial expression recognition in detecting emotional responses to sound interventions in older adults with dementia*. Frontiers. Retrieved May 3, 2022, from https://www.frontiersin.org/articles/10.3389/fpsyg.2021.707809/full

Sreevidya, P., Veni, S. & Ramana Murthy, O.V. Elder emotion classification through multimodal fusion of intermediate layers and cross-modal transfer learning. SIViP (2022). https://doi.org/10.1007/s11760-021-02079-x

GitHub Repositories:
Akshit Madan, Emotion_Detection_CNN (2021), GitHub repository, https://github.com/akmadan/Emotion_Detection_CNN

Hasnain Raza, Fast-AgingGAN (2020), GitHub repository,
https://github.com/HasnainRaz/Fast-AgingGAN