CPSC448

Project Proposal

https://docs.google.com/document/d/1puxoi1hOgQyhx2ozLxz7JmSdaY13pf-dHCr5eZlqh5Q/edit?usp=sharing

Description

I am currently taking CPSC 491 working on a project with CDM to build an interactive toy robot to help children with Autism. For this directed studies course, I will be taking on an extension to this project, building on concepts covered in CPSC 340 and CPSC 425. This project extension would be to implement a Chrome Extension that processes face emotions in youtube videos being watched in real time. This feature will be integrated with the toy robot we are building in the project in order to highlight emotions in the video. The goal is for this to be used as an aid to help teach emotion interpretation and engagement at a young age, and in a very inexpensive and accessible way. This is partially inspired from the Autism Glass Project (http://autismglass.stanford.edu/), hopefully providing another aid for emotion interpretation development in an environment that is both comfortable and engaging for the children. For this directed studies course, I will solely be focusing on the Facial and Emotion Recognition portion of this application as this portion is out of the scope of my teams 491 project.

Learning Goals

- Understand the high + mid level intuition behind the emotion recognition models that would be used in this application
- Gain hands on application experience using open source models for facial recognition and emotion recognition
- Learn to fine tune and improve available tools to solve a specific, real world problem
- Gain experience using tensorflow
- Understand workings behind more advanced models used in industry such as Long Short Term Memory
- Understand intuition behind implementation of models used for emotion and facial recognition
- Explore potential options for improvement for this application

Table of Contents/Breakdown

This course was broken down into two main parts in order to maintain a balance of both implementation and reading/learning. The first part stems from the 491 course, implementing and improving the chrome extension outlined above. Here I was able to apply and build upon skills covered in CPSC322 + CPSC 422 to integrate spatial locality into the emotion recognition system using Markov Models. The second was to dive deeper into reading papers to get a better idea for how the open source ML tools used are actually working. This allowed me to strengthen my knowledge of Neural Networks and CNNs covered briefly in CPSC340 and 425. Because the readings and papers have a lot of overlap in terms of material and learning goals I have provided references to these overlaps throughout both sections. For high level layout purposes I have chosen to keep them seperated.

Youtube Chrome Extension

- Demo (https://github.com/ryanknauer/CPSC448/tree/master/YoutubeExtension#demo)
- Outline of Implementation (https://github.com/ryanknauer/CPSC448/tree/master/YoutubeExtension#progress)
    - Setting Up Chrome Extension (https://github.com/ryanknauer/CPSC448/tree/master/YoutubeExtension#1-setting-up-chrome-extension)
    - Capturing Frames (https://github.com/ryanknauer/CPSC448/tree/master/YoutubeExtension#2-capturing-frames)
    - Face Recognition (https://github.com/ryanknauer/CPSC448/tree/master/YoutubeExtension#3-face-recognition)
    - Emotion Recognition (https://github.com/ryanknauer/CPSC448/tree/master/YoutubeExtension#4-emotion-recognition)
- Improving Emotion Recognition Using Markov Models (https://github.com/ryanknauer/CPSC448/blob/master/YoutubeExtension/Markov.md)
    - First Attempt Markov Model (https://github.com/ryanknauer/CPSC448/blob/master/YoutubeExtension/Markov attempt)
        - Results (https://github.com/ryanknauer/CPSC448/blob/master/YoutubeExtensior
        - Next Steps (https://github.com/ryanknauer/CPSC448/blob/master/YoutubeExtensior steps)
    - Hidden Markov Models (https://github.com/ryanknauer/CPSC448/blob/master/YoutubeExtension/Markov markov-models)
        - Transition Model (https://github.com/ryanknauer/CPSC448/blob/master/YoutubeExtensior model)
        - Observation Model (https://github.com/ryanknauer/CPSC448/blob/master/YoutubeExtensior model)
        - Single Value Observation (https://github.com/ryanknauer/CPSC448/blob/master/YoutubeExtensior valued-observation)
        - Vectorized Observations (https://github.com/ryanknauer/CPSC448/blob/master/HMM.ipynb)
        - Results (https://github.com/ryanknauer/CPSC448/blob/master/YoutubeExtensior 1)
    - Resources (https://www.youtube.com/watch?v=9yl4XGp5OEg)
        - CPSC 322 + 422

(https://github.com/ryanknauer/CPSC448/blob/master/YoutubeExtensior
322--422)
    - Bert Huang - Virginia Tech HMM lecture
(https://github.com/ryanknauer/CPSC448/blob/master/YoutubeExtensior
huang---virginia-tech-hmm-lecture)

Paper Readings

- Reading #1 - A Brief Review of Facial Emotion Recognition Based on Visual Information (https://github.com/ryanknauer/CPSC448/blob/master/Readings1.md)
- Reading #2 - Neural Networks, Convolutional Neural Networks, ImageNet (https://github.com/ryanknauer/CPSC448/blob/master/Reading2.md)
- Reading #3 - Recurrent Neural Networks, Long Short Term Memory (https://github.com/ryanknauer/CPSC448/blob/master/Readings3.md)

Future Learning

- Further Implementations (https://github.com/ryanknauer/CPSC448/blob/master/NextSteps.md#implementation-of-emotion-recognition)
- Loose Ends (https://github.com/ryanknauer/CPSC448/blob/master/NextSteps.md#loose-ends)
- Further Explorations (https://github.com/ryanknauer/CPSC448/blob/master/NextSteps.md#explorations)

Chrome Extension Using Open Source Frameworks:

For the first portion of this project, I created a demo Chrome Extension that recognizes a face in a Youtube video and overlays an emoji indicating the emotion of the face. This iteration utilizes two open source frameworks for Facial and Emotion recognition, Face-API.js (https://github.com/justadudewhohacks/face-api.js?files=1) and FrontEnd-EmotionDetection (https://github.com/kevinisbest/FrontEnd-EmotionDetection) respectively. A video demo can be seen here (https://github.com/ryanknauer/CPSC448/blob/master/Images/Emotion%20Recognition%20Demo and instructions for downloading the extension are here (https://github.com/ryanknauer/CPSC448#chrome-extension-demo).

Demo

1. Download or Fork this repo.
2. Go to chrome://extensions (chrome://extensions) in chrome.
3. Turn on developer mode in the top right corner.
4. Click "load unpacked"
5. Open the "YoutubeExtension" file from this repo
6. Go to any Youtube Video and click the 'Watch With Me' button in the bottom right of the video player.
   Note: You may need to refresh the page once for the button to appear

Progress

1. Setting Up Chrome Extension

The first task was setting up the Chrome Extension to allow for content script injections. This allows the extension to interact with the web page anytime a Youtube Video is loaded. From here I injected a button into the Youtube Player that would indicate whent to start looking for emotions in the video as seen below:

2. Capturing Frames

The next step was to explore frame grabbing from a Youtube video. At first I thought this would need to be done using an api such as TabCapture (https://developers.chrome.com/extensions/tabCapture), which was proving to be slow and require a low rate of frames to be processed. After some more research, I discovered that the frames could be captured directly from the HTML Video element by drawing each frame onto a seperate canvas element.

3. Face Recognition

With access to the video frames, the next step is to locate a Face in each frame. As there are many open source packages available in Python, I first looked into NativeMessaging (https://developer.chrome.com/apps/nativeMessaging) which would allow the chrome extension to commmunicate directly with a local python application doing the image processing. The downside of this would be an additional required app to be downloaded by the end user. In the end, I was able to find FaceApi.js (https://github.com/justadudewhohacks/face-api.js?files=1) which ended up being a fairly thorough implementation that didn't require too much implementation overhead. One substational problem I noticed right off the bat was the low frame rate of many of the models available, with only one that seemed to work well enough for real-time video recognition called TinyFaceDetector (https://github.com/justadudewhohacks/face-api.js?files=1#tiny-face-detector). This effeciency is one area I would like to explore further as I start to research the recongition models more.

4. Emotion Recognition

With the face detection giving a bounding box for the face itself, I was able to grab the image of the face alone in order to perform an Emotion Recognition function (https://github.com/ryanknauer/CPSC448/blob/6d8590f970eb55345c6eb4c7e3eb5426009df0e5/ . While there was much less sources for Emotion Recognition, I found another open source project called FrontEnd-EmotionDetection (https://github.com/kevinisbest/FrontEnd-EmotionDetection) with a pre-trained model for TensorFlow.js. Once the emotion detection was working, I then added an overlay of an emoji in the top left corner indicating the emotion of the recognized face as seen below:

5. Tuning

Tuning will be an ongoing process for this project and one that I will look to explore in my readings. One very big issue with the inital implementation was the model jumping between emotions on non-discernible faces, which is very common in a normal youtube video. As an initial step to help lower the overwhelming nuber of false negatives I implemented a fairly simple thresholding method (https://github.com/ryanknauer/CPSC448/blob/6d8590f970eb55345c6eb4c7e3eb5426009df0e5/ Because the model returns percentages for each possible emotion, I created a minimum threshold for any non-nuetral emotion. This causes only strong deviations from nuetral to be picked up, which helped slightly with the staggering predictions on nuetral faces.

The next large portion of the project involves further tuning this model by integrating Markov Models. To keep clutter down I have seperated that to a seperate markdown file here (https://github.com/ryanknauer/CPSC448/blob/master/YoutubeExtension/Markov.md#markov-chains-for-video-smoothing).

Tuning with Markov Chains

One large issue identified with the first iteration of the recognition application was the choppiness of false-positive in a video. This is because we were checking each still image seperately with no context for the previous frames. This would result in constant flashings of incorrect emotions which would be fairly detrimental to the experience for an end user. To solve this, we want to incorperate information from previous frames into our emotion prediction model. For example, if we see 10 frames heavily predicted to be a sad face followed by a single frame predicted to be happy, we most likely don't want our model to be predicting happy until we have seen more consistent happy predictions in a row. One idea mentioned in my first reading (https://github.com/ryanknauer/CPSC448/blob/master/Readings1.md) is the use of RNNs and LSTMs, which I explore in more depth during my third reading (https://github.com/ryanknauer/CPSC448/blob/master/Readings3.md). These models can be extremely complexand would require a large overhead both in knowledge and development. A great idea Steve came up with was to utilize markov models and incorperate them with our CNN acting as an observation sensor as I will outline in this document. Markov Models have a much lower overhead while still encompassing the same ideas of RNN + LSTMs on a much smaller scale.

First Attempt

For the first attempt, I want to try using a binary transition model, which only indicates if the emotion stays the same or not. While their are likely different probabilities of moving from say happy -> nuetral vs happy -> sad, these should be very small relative to staying in the same state(e.g. happy -> happy) from frame to frame. To do this I created a transition array with each index correlating to an emotion, and each value indicating a weight. I then set the value of the previous emotion state to a preselected weight. Finally, performing an element-wise array multiplication with the predicted emotion values of the current frame(held in a tensor 'z'):

```
let transition = [1,1,1,1,1,1,1]
transition[most_recent_emotion] = markovWeight
return z.mul(transition)
```

In order to easily compare and test different weighting values, I also implemented a URL-parameter grabber to allow for these values to be set without reloading the extension. These can be set via url parameters such as:

https://www.youtube.com/watch?v=qTLrjhReNtg&markovWeight=9&useMarkov=1

Results

Original w/o markov chain

Using markov chain w/ markovWeight 9 (normalized to 0.6 for same state and 0.066 for all other transitions)

As you can see above, this did a successful job smoothing over quick jumps in emotions, however this comes with a bad tradeoff seen below.

Original w/o markov chain

Using markov chain w/ markovWeight 9 (normalized to 0.6 for same state and 0.066 for all other transitions)

I've found it very difficult testing different weights for 2 reasons.

1. The neutral expression is very overpowering. This would likely need to be treated with a different set of rules compared to other emotions.
2. Due to performance issues not every frame is being captured. This will provide vastly different results depending on the computer and video.

Next Steps

My next step will be to try using a hidden Markov Model, as our previous state is actually a set of probabilities not just a single correct emotion. This will help when our previous state is not 'certain' on the best emotion and overinfluencing the following states.

Hidden Markov Models

In order to utilize more than just the previous frame using a Hidden Markov Model would likely be much more effective. We can model our our hidden state as seen below where Emotional State is the actual emotional state shown in the video, our Observation is the representing the value observed for that frame, and each t represents and individual frame.

Transition Model

Our transition Model will represent the probability of transitioning from one state to another(including staying in the same state). This can be modeled as a Square Matrix where an index i,j represents the probability of moving from state i to state j. Because the initial goal is to

smooth over sporadic jumps between emotional states I will start with an abritrarily high weight for each diagnoal representing remaining in the same state.

Observation Model

The observation model describes that if we know the current state of an individual frame, what is the probability of the CNN model to predict each state. Again this can be represented as Square Matrix with a value X at index i,j that in the given state i, state j has an X probability of being observed.

Single Valued Observation

Most common HMM assume a single valued observation representing the exact state observed. So, even though our CNN observation provides a probabilistic observation for each state, we will start by providing only the highest probable state as the single observed state. Of course this looses any differentiation between more or less confident guesses by the CNN observation model.

Vectorized Observation

LaTeX Doesn't Display In Markdown so this is in a seperate file here (https://github.com/ryanknauer/CPSC448/blob/master/HMM.ipynb)

Results

Below I have added a side by side comparison between our original CNN model(Right) and the updated HMM model using vectorized observations(right):

As you can see, the original model flashes between many different emotional states very quickly due to weaknesses in the observation model and a lack of context from previous frames. By adding in a HMM that preserves information from previous frames we get a much smoother transition between emotional states.

Resources

CPSC 322 + 422

CPSC 422 helped lay the groundwork for understanding these models and algorithms, especially when dealing with filtering HMMs. While it had been a long time since taking these courses, and I needed to refresh my understanding using tools listed below - the deeper underlying concepts came back to me much quicker this time around. When taking 422 one of the biggest challenges I faced was mapping the model's usecases to real world problems. This project did a fantastic job showing me exactly that.

Bert Huang - Virginia Tech HMM lecture

https://www.youtube.com/watch?v=9yl4XGp5OEg
While the more conceptual aspects of HMM's came back to me fairly quickly from 422, I definitely struggled more with the math side of implementing filtering. This lecture had the most concise

explanation and visuals describing the math needed to implement filtering, which helped lead me to the end result.

Reading #1

I started out by reading "A Brief Review of Facial Emotion Recognition Based on Visual Information' (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5856145/pdf/sensors-18-00401.pdf) to get an overview of the different high level steps involved in Emotion Recognition. As I can already tell that I need to strengthen my knowledge in many areas in order to understand the deeper technical explanations, I will plan to come back to this paper later on and giving a more technical summary. The article outlines two main methods for Facial Emotion Recognition(FER). The original method was broken down into many steps:

1. Face detection - locating a face in an image allowing for only important data to be used in following steps
2. Landmark Detection - locating potential features (e.g. SIFT) not yet specific to facial features; used for #1 as well
3. Feature Extraction - locating features such as noses, eyes by clustering Landmarks from step #2
4. Facial Expression Classification - Making a Facial Expression Classification based on features from #3

The second and newer method utilizes Deep Learning and Neural Networks with very effective results. This method allows for less dependence on "face-physics based models", which I believe indicates ...

In addition, this paper also classifies two seperate FER problems split between static and video input. This indicates methods that would be useful for interpreting based not just on a current frame, but previous frames as well - a method that would be very useful to explore for the Youtube Recognition Project being explored.

Video Feature Extraction

For training on video sequences, the paper outlines a method of feature extraction using displacement between matching features in sequential frames as an additional dynamic feature. The issue that this method would present in the Youtube Extension Project is that the frame rate being captured is very limited and unpredictable depending on the local computer's specs. These sequential video models would be trained with a specific(and very high) framerate in mind, likely causing all sorts of problems attempting to make predictions with inconsistent, and much larger gaps between frames. This leads me to believe that continuing on the path of smoothing over static image models, e.g. using Markov Models (https://github.com/ryanknauer/CPSC448/blob/master/YoutubeExtension/Markov.md), may be the most effective approach for this project.

Knowledge Gaps

The rest of the paper dives into deeper concepts of deep learning which I currently am lacking enough knowledge in to fully understand. At this point I will pause reading this paper in order to read up on CNN's and work my way back to this reading with a stronger knowledge base.

Revisiting

After completing my 2nd and 3rd readings on NN,CNNs,RNNs, and LSTMS I was able to come back and finish the remainder of this paper. One of the interesting discoveries I made in doing this was that it outlines the potential usefulness of RNNs and LSTMs for emotion recognition in video sequences. The paper outlines a number of different attempts for using both of these models, all of which seemed to have favorable results. The majority of these actually used a CNN in conjunction with the sequencing models in order to first gather higher level observations from the image, which were then used as inputs to the RNN/LSTM. One of these studies also discovered the impact that a bidirectional sequencing model had over a unidirectional one(e.g. looking at frames both before and after the current one). This would be trickier in real time as we would need to constantly buffer ahead in the video, however it is an interesting route to observe in the future especially if a preprocessing method was implemented over real-time.

Readings 2

Overview

In my first reading I hit a wall when the paper got deeper into the use of Nueral Networks and CNNs, so for my second set of readings I wanted to further my understanding of CNN's.

Materials + Overview

First, I went through 3Blue1Brown's mini-course on Neural Networks (https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi). These videos give an extraordinary overview of the intuition behind how and why Neural Networks actually work.

Next, as the core of emotion recognition would be built on CNNs, I went through Deep Lizard's Series on CNNs (https://www.youtube.com/watch?v=YRhxdVk_sIs&list=PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU&index=21).

Finally, looking into Emotion Recognition tutorials lead me to read the paper: ImageNet Classification with Deep Convolutional Neural Networks (https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf), which outlines an extremely successful image recogntion CNN model that is often used as a starting point for FER models. Going into this paper, one of my biggest points of confusion was how the hyperparameters Nueral Networks are decided upon. Nueral Nets come with a large amount of variability in this area, and while the high level workings are explained very well, there seems to be little intuition on how to actually choose these parameters. While part of the answer seems to be that there really is no 'right way' or concrete intuition behind setting up a CNN/NN, this paper did provide alot more understanding behind the actually process.

Questions/Key Findings

3B1B's video gives an example of a NN recognizing a number from an image. Originally I believed CNN's were reducing the input of an image to a NN. Now I understand the structure differences behind both, but why are CNN's more useful than pure NN's?
Answer:

- A NN with each pixel being one input node loses any spatial locality between pixels. This means that if you rearrange the pixels in your training and test set, it would have the same result. A CNN using convolution filters as each node in a hidden layer includes

spatial context up to the size of the filter.
- This comes with the added benefit of reducing the input space significantly
References: quora post (https://www.quora.com/Why-are-convolutional-neural-networks-better-than-other-neural-networks-in-processing-data-such-as-images-and-video), 3B1B videos, and DeepLizard videos

What are some of the design choices used when building a CNN?

- First, one of the main answers to this question is "nobody really knows" - there is currently no right way to design/build a CNN just ways that seem to work the best. Imagenet is an example of this, and one of the most widely used examples of this for image classification. Imagenet is commonly used as a baseline model for solving similar problems and fine tuned (https://www.youtube.com/watch?v=5T-iXNNiwIs) to the specific problem scope.
- GPU limitations seem to be the next biggest impact of design choices, as current GPU's are only able to train complex models with large amounts of optimizations implemented to speed training up. For example with the ImageNet paper, the majority of the paper outlines optimizations such as:
    - Using Relu Activation functions over sigmoid/tanh functions as ReLu results in significantly more 0 or "inactive" responses saving computation time on backpropagation(outlined in 3B1B video here (https://www.youtube.com/watch?v=aircAruvnKk&t=460s))
    - Dropout - ensambling(combining multiple models) is a very effective tool to reduce overfitting, however with complex models like Imagenet ensambling would take far too long. Dropout sets a probability of setting any neuron to 0, again saving time on backpropagation due to increased sparsity as with Relu. This works effectively with ensambling only increasing training time by a factor of 2.

Readings 3

Materials

Brandon Rohrer - "Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM)" (https://www.youtube.com/watch?v=WCUNPb-5EYI&t=713s)
This video gives a high level overview of both RNNs and LSTMs. I felt this did a great job of covering the intuition behind why RNN and LSTMs are useful which I find to be a large learning barrier with many of this more dense conceptual models.

Understanding LSTM Networks (https://colah.github.io/posts/2015-08-Understanding-LSTMs/)
This write up on LSTMs contains the best examples and visuals I have found and really helped me solidify the understanding of each gate. Additionally, it did a great job outline the transition from RNNs to LSTMs.

Illustrated Guide to LSTM's and GRU's: A step by step explanation (https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21)
This walkthrough covered similar topics as the above two, however I found it very useful in understanding the input fields for each gate in an LSTM due to the animated graphics showing the information flow of an LSTM. Additionally, this walked along each step referencing a real world example which I find to be very helpful for conceptualizing the process.

Recurrent Neural Networks

Interestingly enough, the application phase of this project using Markov Models (https://github.com/ryanknauer/CPSC448/blob/master/YoutubeExtension/Markov.md) really helped my transition into understanding the high level behind RNN and LSTM as these solve the same problem we were trying to solve using Markov Models. A RNN is a Neural Network that feeds in the prediction of the previous state as inputs into the Neural Network. This is similar to our use of HMM using a transition probability from a previous state to the next one. The weakness of this, is that the context does not maintain predictions more than one step back. This becomes a problem in something like language translation, where words can be heavily reliant on one another even if they are spaced multiple words apart. This is perfectly outlined from Understanding LSTM Networks (https://colah.github.io/posts/2015-08-Understanding-LSTMs/) using the sentence "I grew up in France... I speak fluent ____." The ability to predict the blank word is highly dependent on the word "France" used in the previous sentence.

Long Short Term Memory

As discussed above, the weakness of RNN's is that they only preserve the prediction of the previous state. A LSTM provides a NN(actually a collection of NNs) architecture that allows for memory of previous predictions to be stored, decides how long to store the predictions, and which predictions to ignore at certain states. These different tasks are implemented through gates, each powered by it's on Neural Network and regulate the flow of information throught the model. This control of information is vital as the complexity of the network would grow exponentially for every prediction that is stored into memory if the model was not concurrently filtering out the most 'useful' pieces of information. The most common layout of theses gates consists of:

- Cell State - This isn't actually a gate, but it is the core pathway for information within LSTMs. The cell state carries information throughout the sequence of states, the rest of the actual gates then decide when to add and remove information to/from the cell state
- Forget Gate - This gate decides which Cell State values to forget based on the current sequence state inputs and the previous state predictions. This outputs a vector passed through a activation function(sigmoid) indicating the weight of importance for keeping or forgetting.
- Input Gate - This gate decides what new information to add to the Cell state using both a tan + sigmoid function to create a update vector to apply to the existing cells state. This takes the same inputs as the Forget State: current sequence state inputs and the previous state predictions.
- Output Gate - The last gate instead also takes the cell state and creates a filtered version based on our two other inputs(current state + previous predictions).

Application to Miigo

Note: My intuition on LSTM effectiveness below is likely incorrect after revisiting Reading 1 (https://github.com/ryanknauer/CPSC448/blob/master/Readings1.md#revisiting). LSTMs actually have been used quite successfully in this context when combined with CNNs for deeper feature extraction.

For our application, RNNs might actually be a better fit than LSTM models as the biggest challenge we face is smoothing over jumps in emotions. Generally, emotions will not have very complex patterns over time like that of speech or writing. After further researching these models, I also believe that our implementation (or at least direction of implementation) of Markov Models was a

very viable solution for the problem we faced. One of the biggest reasons for this is the unpredictability of computing time leading to dropped frames. Since we are running the predictions in real time, we often drop dozens of frames based on computing power. Since an RNN would be trained based off a linear sequence of constant time intervals, these dropped frames would add an additional layer of complexity. With that said, it seems that RNN would still be able to handle this by recursively predicting each of the dropped frames before predicting the current one. Unfortunately this would then likely add an even larger time burden.

Future Learning

Implementation of Emotion Recognition

If time permitted, my next step for my own personal learning goals would be to attempt training my own emotion recogntion model. From my research, I discovered the common practice of fine tuning (https://www.youtube.com/watch?v=5T-iXNNiwIs) an existing model. This consists of taking the first X hidden layers of an already implemented and successful NN/CNN throwing away the fully connected prediction layers, and retraining the a new connected layer model towards your new problem statement. In this case I would consider using ImageNet discussed in reading #2, and adapting the fully connected layers for emotion classification.

Loose Ends

One of the main loose ends remaining for the Markov Model portion of this project is deciding on and quantifying effective observation and transition models for the HMMs. Currently I am using diagonalized matrices and setting the diagonal through trial and error. Ideally, I would be able to do a run through on test data to provide a potential estimate of these actual probabilities. Another great idea Steve mentioned was to effectively shift the transition model based on the time elapsed between frames. This would help lower the weight of transitions in cases where we drop a large number of frames between observations.

Researching faster emotion recognition models. There was a very limited number of Emotion Recognition models available that could be easily used in Javascript. This was by far the slowest computation, thus finding a faster model would help tremendously with the frame dropping issues discussed above and in the RNN/LSTM reading seciton (https://github.com/ryanknauer/CPSC448/blob/master/Readings3.md#application-to-miigo)

Looking for a faster way to load the models into content scripts. Currently, every time you want to run the emotion recognition chrome extention on a youtube video the applicaition must load the FER models from the background chrome extension to the foreground content scripts. This is extremely slow and really hurts the experience both as a user and developer. Here I could possibly run the Emotion Recognition in the background chrome extension scripts, removing the need for transfering the models altogether.

Explorations

Lastly, I would like to continue exploring the potential usefulness of RNNs and LSTMs in the context of this application. I outlined one potential issue due to dropped frames here (https://github.com/ryanknauer/CPSC448/blob/master/Readings3.md#application-to-miigo), however there are still many potential solutions for this. First, as outlined RNNs could still handle dropped frames, however the usefuless of this is needs further research. Second, dropped frames

could be eliminated by either buffering the videos or preprocessing the entire video on a backend server before watching. Without the issue of dropped frames, RNN (and potentially LSTM) models may make significant improvements on the overall emotion recognition system. A snippet from my first reading indicates the succesfulness of RNNs in the context of Emotion Recognition in video sequences:

"Kahou et al. [11] proposed a hybrid RNN-CNN framework for propagating information over a sequence using a continuously valued hidden-layer representation. In this work, the authors presenteda complete system for the 2015 Emotion Recognition in the Wild (EmotiW) Challenge [52], and provedthat a hybrid CNN-RNN architecture for a facial expression analysis can outperform a previouslyapplied CNN approach using temporal averaging for aggregation." Byoung Chul Ko (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5856145/pdf/sensors-18-00401.pdf)

Additional, my first reading (https://github.com/ryanknauer/CPSC448/blob/master/Readings1.md#revisiting) indicated an impact on accuracy when implementing a bidirecitonal model over a unidirectional one(e.g. looking at both previous and future frames). While this would be difficult in real time without a self-implemented buffering system, it would be an interesting path to explore using a preprocessing method. This could be adapted both for our HMM model implementation as well as any future RNN or LSTM models.