# CS5131 Project Report

# Detecting Clickbait in YouTube Videos

Ashley Khoo and Daniel Toh

M21503

# Problem

YouTube is a website where people can post videos on a public space for others to watch. Some people make video-making their jobs, that means they earn their living off making videos on YouTube, generally these people are called YouTubers. Since YouTubers earn more money the more viewership they have, several YouTubers have resorted to “clickbait” in which previews (title, thumbnail) are often exaggerated (or sometimes even completely different) from the actual content of the video.

Objective

To design a model that is able to determine whether a given video is clickbait or not given its metadata (title, thumbnail, description, likes, dislikes, views).

# Dataset

There are no publicly available datasets for clickbait videos. To obtain a dataset of videos which is labelled requires a human annotator to watch a video and determine whether it is clickbait or not. However, this is an infeasible strategy to label a few thousand videos.

Therefore, we will collect videos in 2 ways. The first set, called alpha, will be collected by labelling channels instead of videos. The second, called beta, will be collected by labelling videos. You can find the videos in [4].

To collect channels for set alpha, we came up with a list of 50 channels which we believe consistently produce clickbait / non-clickbait videos and marked them, then scraped the lastest 200 videos from each channel using Selenium. Although this method allows us to easily get a large dataset, it is very "noisy" data.

For set beta, human annotators would randomly select videos to label. Since there are significantly more non-clickbait videos than clickbait videos, we had to intentionally find clickbait videos. Furthermore, videos selected were only chosen by 2 people, therefore there is quite a lot of bias. To mitigate this, we chose 8 broad categories of YouTube videos and tried to balance the distribution of videos across categories.

Set beta was further split 50:50 into sets beta 0 and beta 1. Alpha and beta 0 will be used in training while beta 1 will be used in testing.

Here is a summary of counts:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Alpha | Beta0 | Beta1 |
| Non-clickbait | 5477 | 128 | 124 |
| Clickbait | 4264 | 42 | 46 |

## Video Hydration

A simple web scraper was used for this task. For each video, the html source was downloaded, which contains a json file with metadata such as title, description, likes, dislikes, views. The thumbnail was also collected by finding the image links of video thumbnails. The thumbnail is a 480 by 360 image.

Some fields of certain videos are missing. There are several reasons for this. Likes and dislikes can be hidden by a video’s publisher, (possibly to hide the fact that their video is heavily disliked by the community). Premium YouTube videos and age-restricted YouTube videos posed another problem as a single html download would not be sufficient to collect data from these types of videos due to them being blocked to a normal user, so some fields for such videos are blank. However, they make up such a tiny proportion of our dataset that we decided to leave them blank.

We had to set a pause time between each time we scraped thumbnails as YouTube would block us when no pause time was set (possibly to deter people from scraping their site).

# Thumbnails (Image Processing)

As thumbnails are image data, we used CNNs to process them. So we built a CNN to classify them.

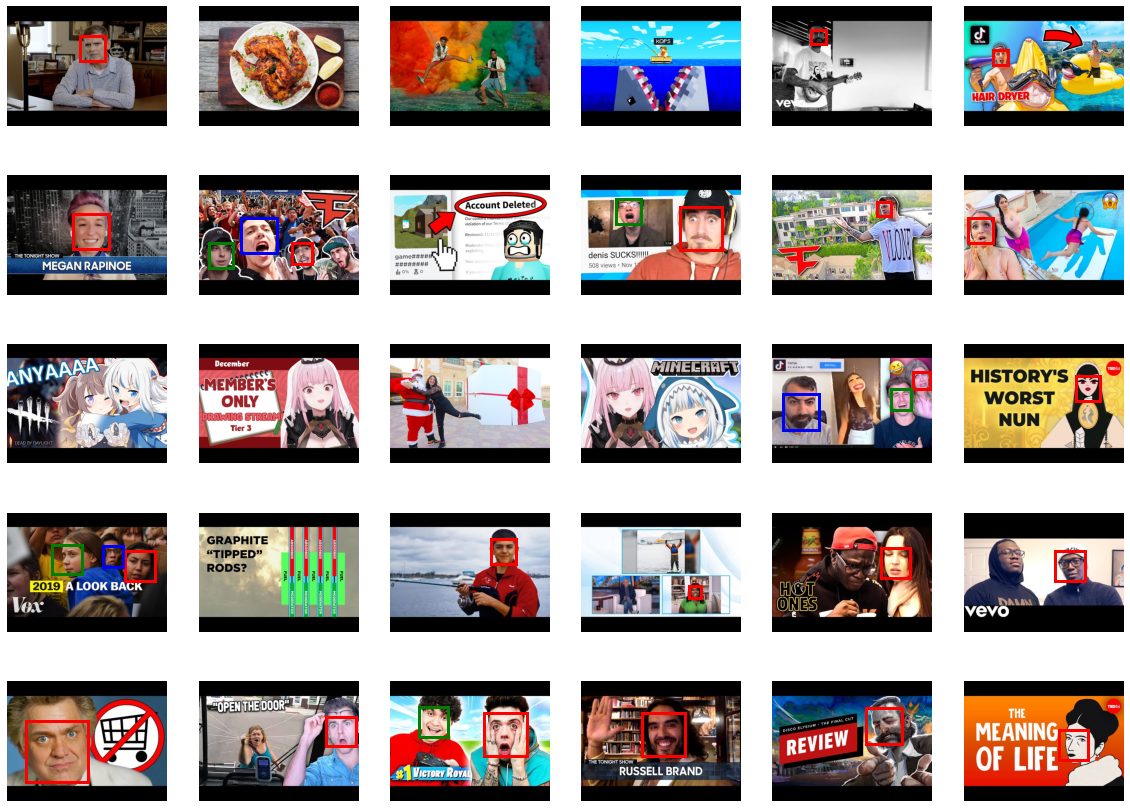
This model gets a validation F1 score of 0.7739 on training set alpha.

From observations, certain facial expressions appear frequently in the thumbnails of clickbait videos.

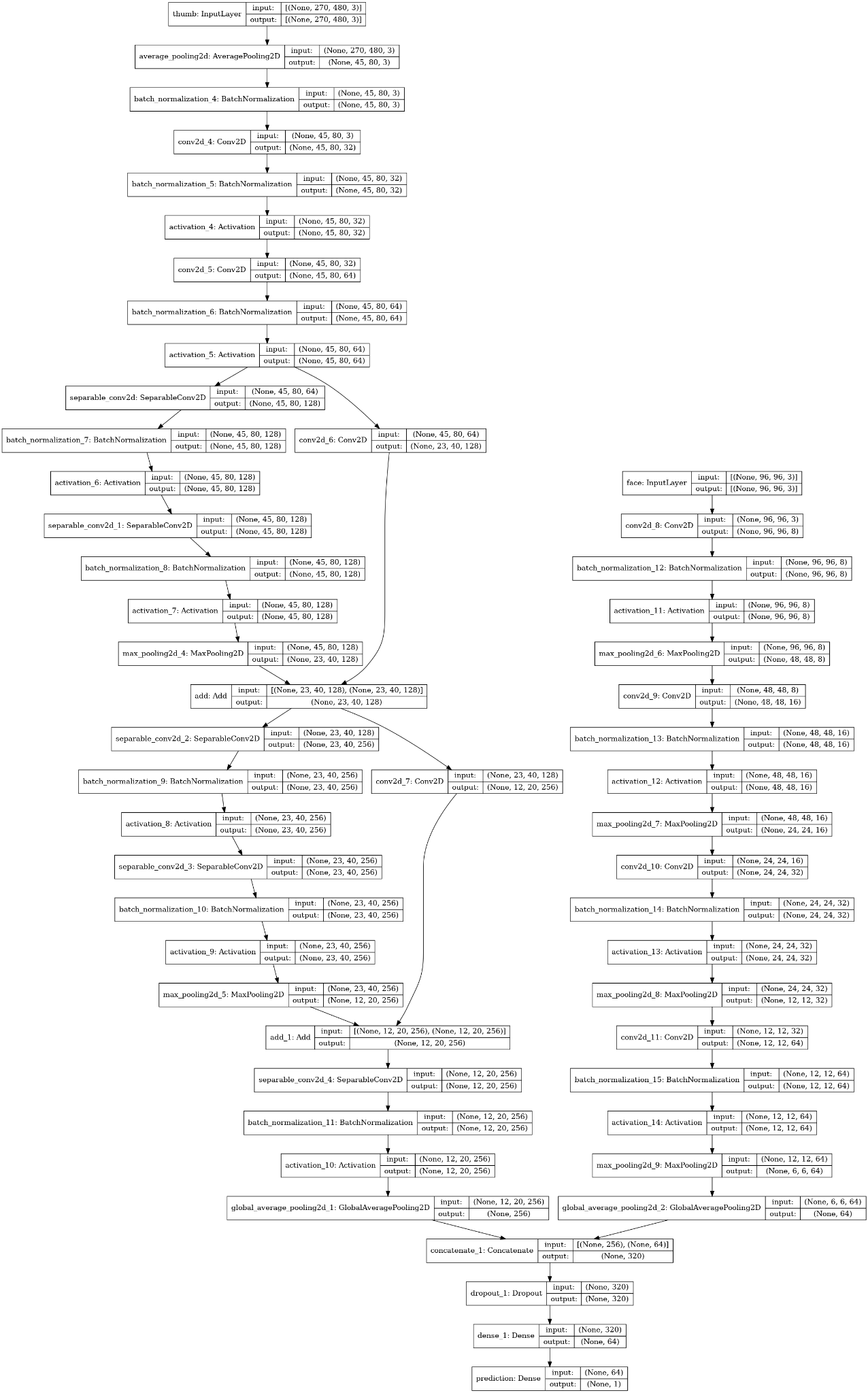


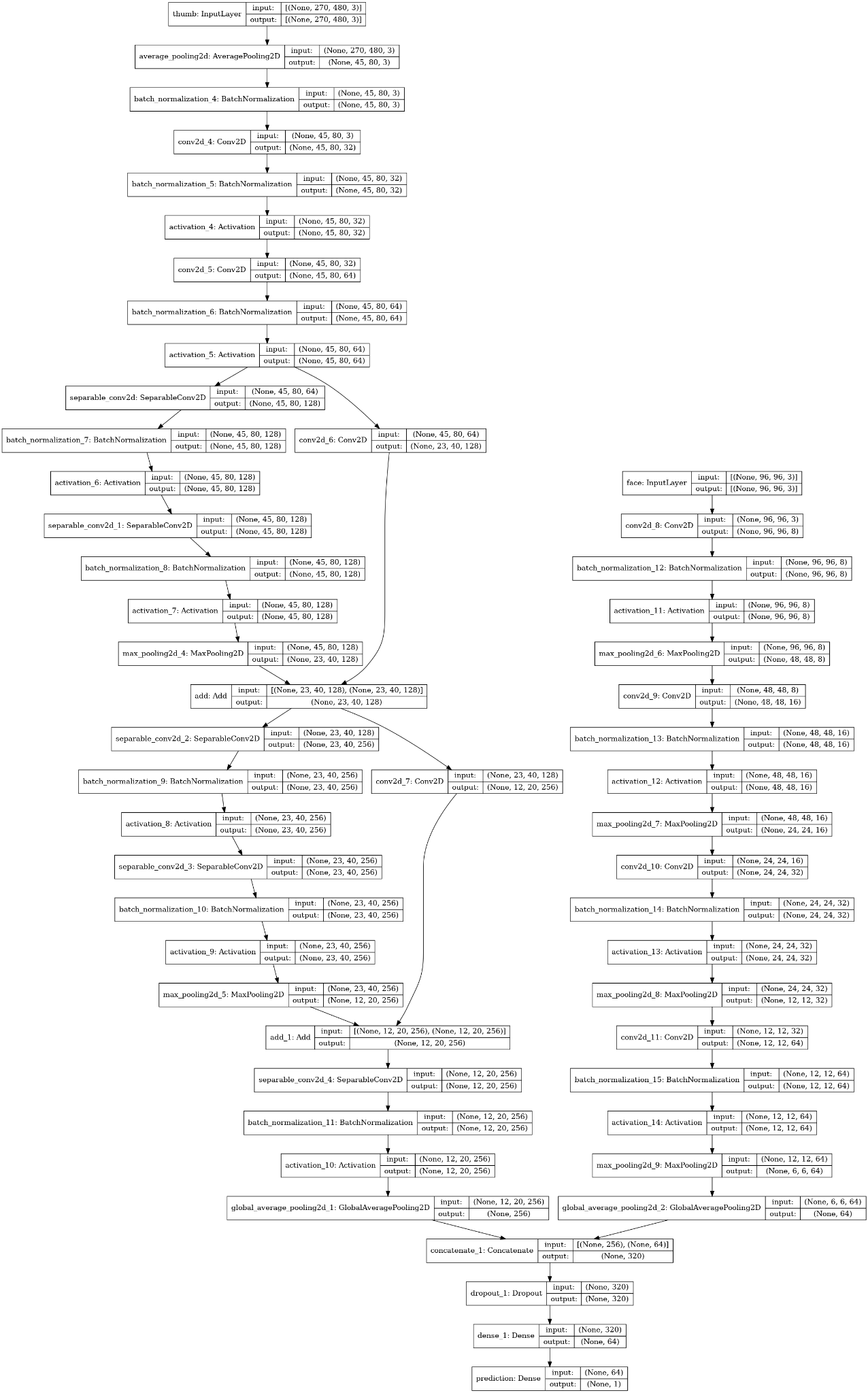
These facial expressions are usually exaggerated, with a telltale sign of these being the open mouth and raised eyebrows (also known as the soyboy face [2]).

To help our model detect such facial expressions, we used Face Recognition[1] to detect the faces then feed them into our model. Although it is possible for our CNN to learn to recognize facial expressions, we believe that this allows the model to pay more attention to the facial expression. We will choose 1 random face and feed it into the model



Interestingly, face recognition recognizes the faces of cartoons from TED videos but not anime faces.

This model gets a validation F1 score of 0.7851 on training set alpha. However, this is only an increase from the previous F1 score by about 0.01.

However this did not give any significant performance improvements compared to not using the faces. We decided to use autoencoders to extract the important parts of the face. And so that the model will not overfit on random details such as hair color.

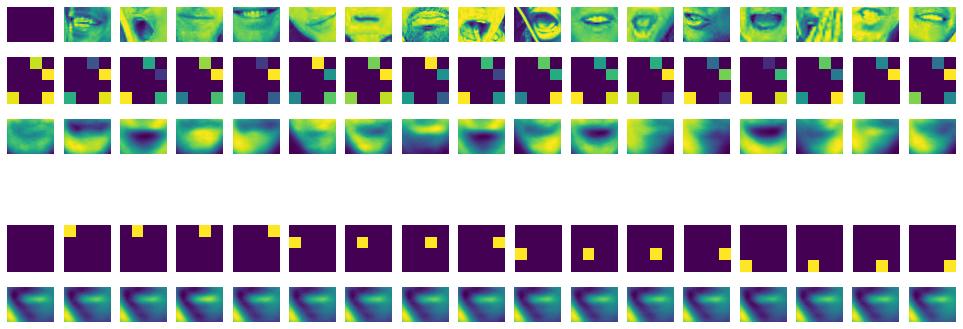
## Autoencoders

Facial expressions usually only involve the eyes and the mouth, and thus humans usually can tell facial expressions looking at these parts only.

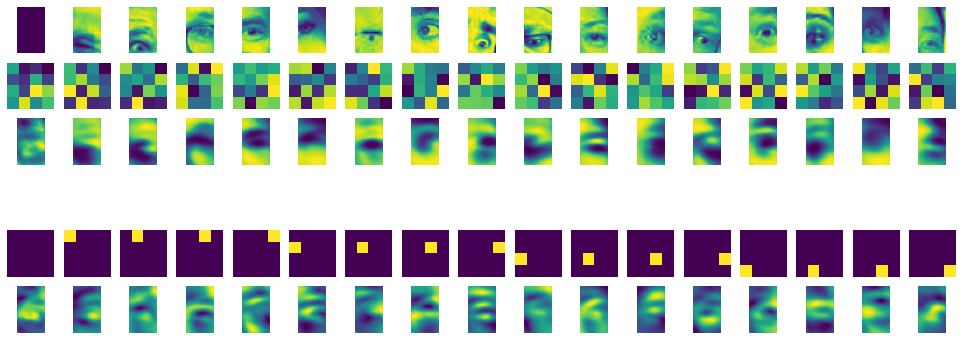
As such, we used auto encoders to encode a person’s eyes and mouth. Face Recognition usually crops faces such the position of the eyes and mouth are at roughly the same place. We trained autoencoders on both a single eye and the mouth. We only used the red channel as human flesh (especially lips) have high amounts of red color. Images were shifted to have a mean of 0.5 and a standard deviation of ⅓, which mostly fits in the range [0,1].

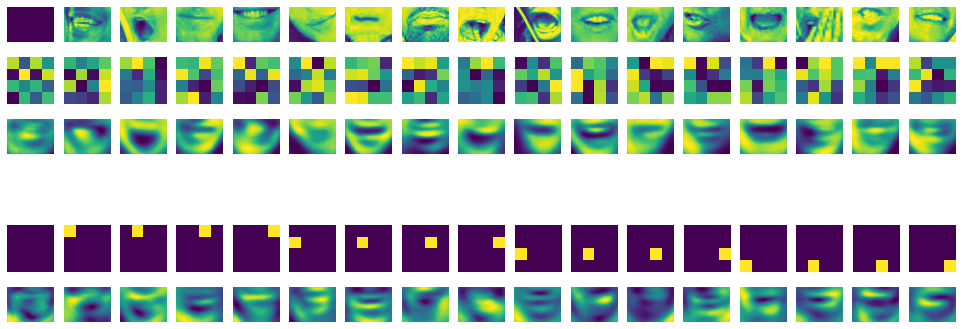
We used a latent space of 16. With both the encoding and decoding layers being fully connected with sigmoid activation.

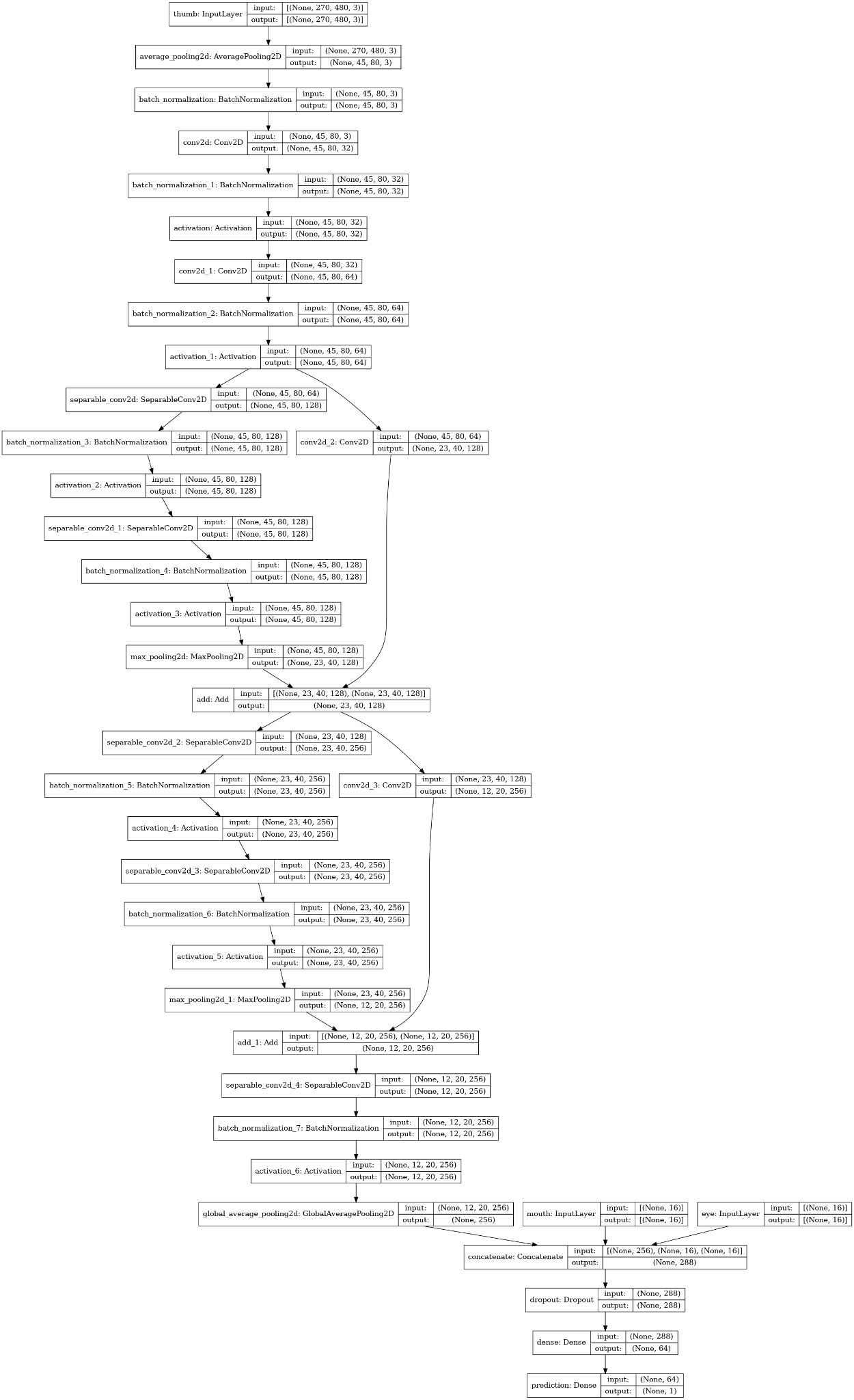
Sigmoid activation was used as ReLU activation in the encoding layer often caused the encoded value to be 0. Sigmoid activation is used at the decoding layer as the outputs should be in the range [0,1].



Here is the result for autoencoding the eyes and mouth. The encoding for the default image (when there is no face) and the outputs of various encoding bits are shown.







Visually, the autoencoder seems to be able to represent the image quite well, dropping away the finer details and only retaining the rough shape of the various facial parts, as we intended.

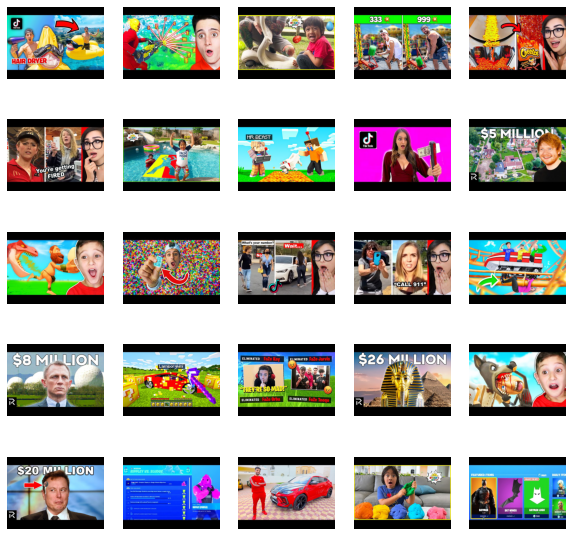
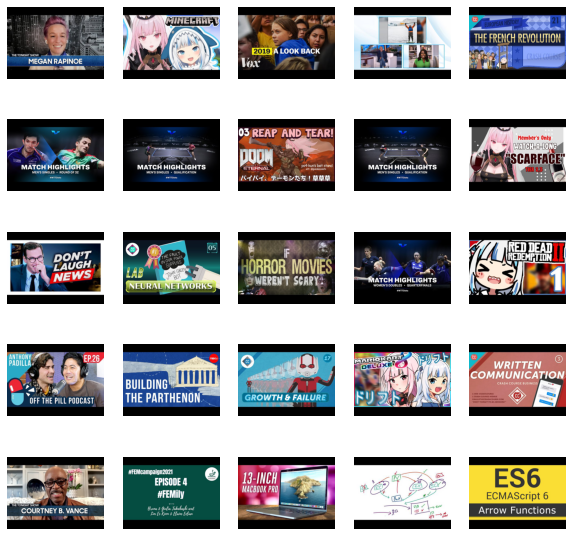
This model gets a validation F1 score of 0.7968 on training set alpha. However, this is only an increase from the previous F1 score by about 0.02.

Despite the fact that the improvement is only 0.02, we observe that using autoencoders of only the mouth and eyes was able to beat the model that was given the original face as an input, which shows that the use of autoencoders was successful here.

## Super Clickbait and Super Non-Clickbait

We can look at what kind of images our model finds as clickbait identifiers and non-clickbait identifiers. We define super clickbait as videos whose output in the model is greater than 0.99 and super non-clickbait as output is lesser than 0.01.

Below is shown some super clickbait and non-clickbait videos, the first 5 columns are super non-clickbait and the next 5 are super clickbait.



For non-clickbait, the model seems to have learnt that large text (possibly accompanied with a human face) is usually non-clickbait.

For clickbait, the model seems to have learnt that red arrows correspond to clickbait, we also see that videos are usually very brightly colored, possibly to appeal to children, and display the soyboy face mentioned above.

# Title (Natural Language Processing)

Much information can be gained from the title of the video.

We will convert the text data of the title into numerical data for the model to use using TextVectorizer from keras.

Before that, we will do some preprocessing of the title.

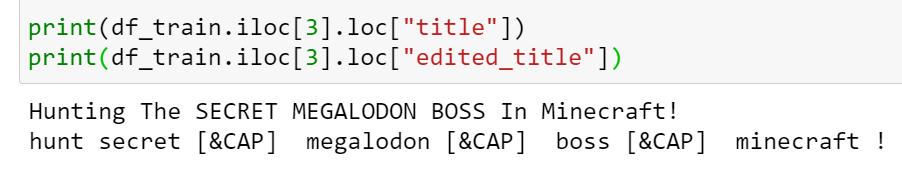
Firstly, we will remove non-printable characters (non-English) from the title.

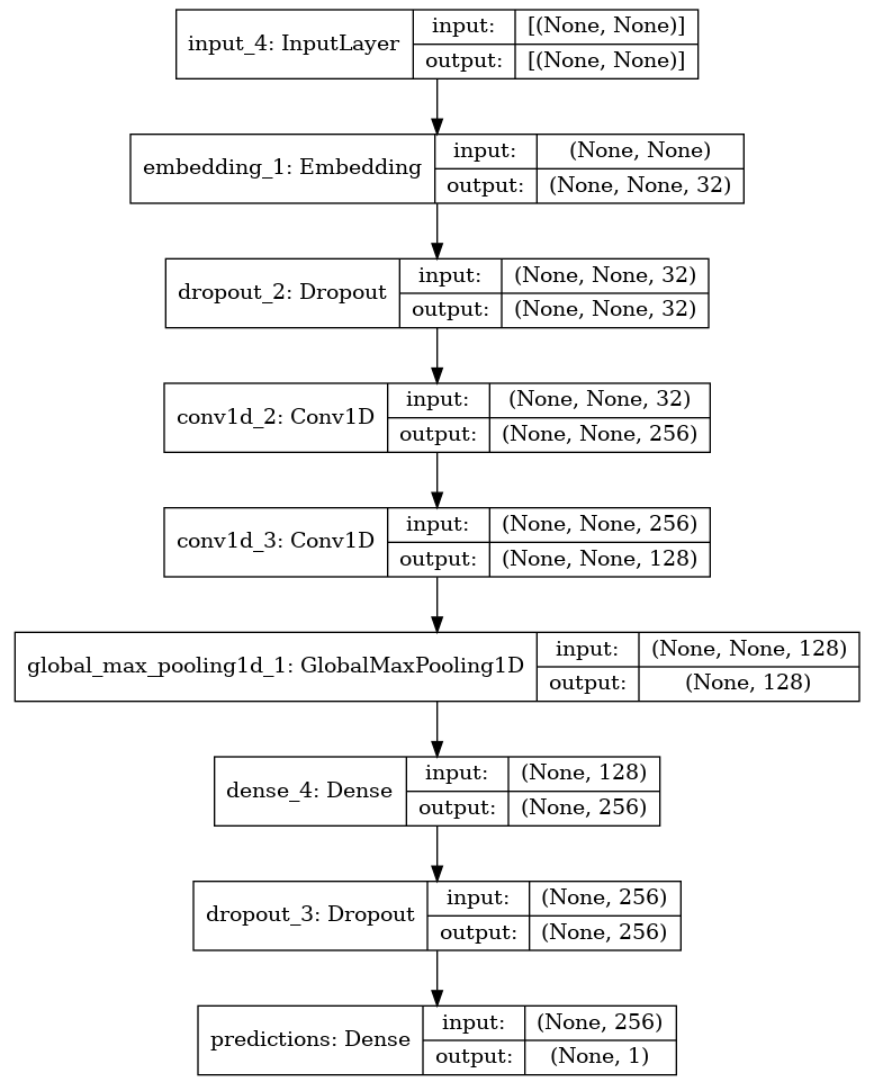
Then, we will use the TweetTokenizer from the nltk library to tokenize the title. This is because YouTube video titles may not follow traditional writing styles and thus informal online writing styles such as those used in tweets is more similar.

For each token, we will convert it to lowercase. As fully capitalizing a word in a title makes it stand out, the capitalization of words in the title is important info and thus simply converting the tokens to lowercase will result in a loss of information. As such, we will add a marker to indicate that the word was fully capitalized in the final string.

We then remove stop words from the title, which are vecommon. The list of stopwords is taken from the nltk library.

Lastly, we lemmatize the words using WordNetLemmatizer from nltk. We also feed the lemmatizer the predicted part of speech of the token for more accurate lemmatization [3].



Here is an example of our preprocessing.

We can see that capitalized words are converted, stop words such as “the” and “in” are removed, and words like “hunting” are lemmatized to “hunt”. Punctuation is left in the edited title as extensive use of punctuation such as exclamation marks and question marks draw attention to the title.

Now we use these edited titles to put into our keras model.

We will use a TextVectorization layer to convert our text into a vector for the model to process. We will fit the vocabulary of the layer to our training dataset, as in practice we do not have the titles of the videos we will test on.

After we vectorize the text, we will feed the vectors into our model.

We embed the text vectors into a smaller dimensional space.

We will use convolutional layers with dropout as the positions of the words relative to each other is important, especially since the capitalization markers are placed next to the words.

# Combining Neural Networks

To obtain a better model, we simply combined both the model for classifying based on thumbnails with the one classifying based on titles.

We did not have enough time to include other metadata such as views, likes or dislikes, but we believe that adding that as additional input to the model would not affect that much.

# Results (Dataset Alpha):

|  |  |
| --- | --- |
| Accuracy | 0.9374 |
| Precision | 0.9410 |
| Recall | 0.9116 |
| F1 | 0.9154 |

# We obtain a high accuracy of 93.7% and a high F1 score of 0.915.Our precision is significantly higher than our recall.

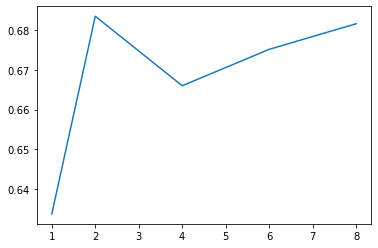
# Extension

Now we will try to use the model above to predict videos in dataset Beta, note that some of the videos in dataset Beta are intentionally selected to be confusing for the model, so we expect quite a low F1 score.

We will use dataset Alpha and Beta0 to predict dataset Beta1. Since the data in beta0 was more “quality” than the data in alpha, we want our model to treat the data in beta0 with more weight, to accomplish this, we made a training set with 1 copy of alpha and K copies of beta0, where K was a tunable parameter. We tried values of 1,2,4,6,8 for K.

Here are the results of the F1 score on dataset Beta1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 4 | 6 | 8 |
| 0.634 | 0.683 | 0.666 | 0.675 | 0.681 |



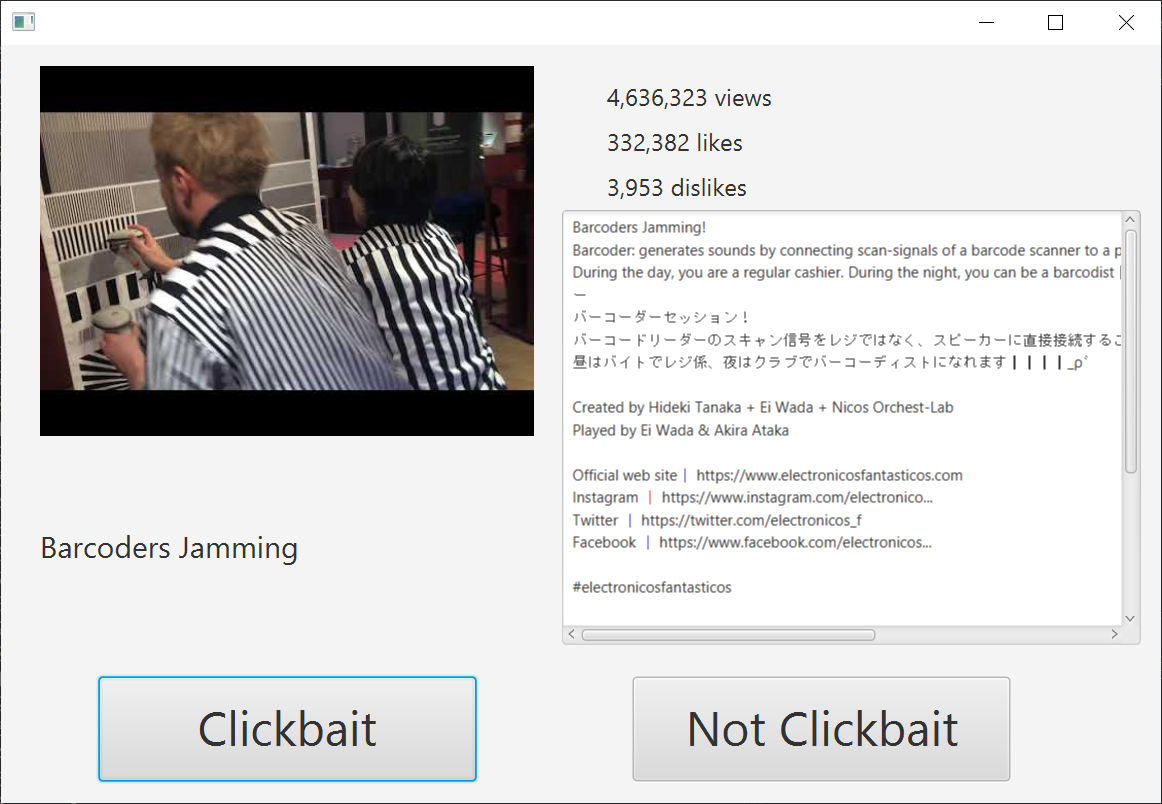
Not much can be said about how K affects the F1 score as the graph is very noisy as we only took 1 run for each K. But it is clear that K=1 performs very badly.

Our model gets the highest F1 score of 0.683 at K=2. This will be the F1 score of our model on dataset Beta1.

## Human Accuracy

A simple JavaFX program was made for humans to predict whether a video was clickbait or not. Human volunteers would be told what we defined clickbait and would try to classify videos in dataset Beta1 as clickbait or non-clickbait.

Here is an example of a video shown in our JavaFX program.



In focus group discussion with human volunteers, we found that most of the volunteers only looked at the picture and thumbnail to determine whether a video was clickbait.

# 

# 

# Extension Results

|  |  |
| --- | --- |
| Evaluator | F1 Score |
| Human 1 | 0.779 |
| Human 2 | 0.767 |
| Human 3 | 0.716 |
| Human 4 | 0.689 |
| Model | 0.683 |
| Human 5 | 0.672 |
| Human 6 | 0.638 |
| Human 7 | 0.557 |

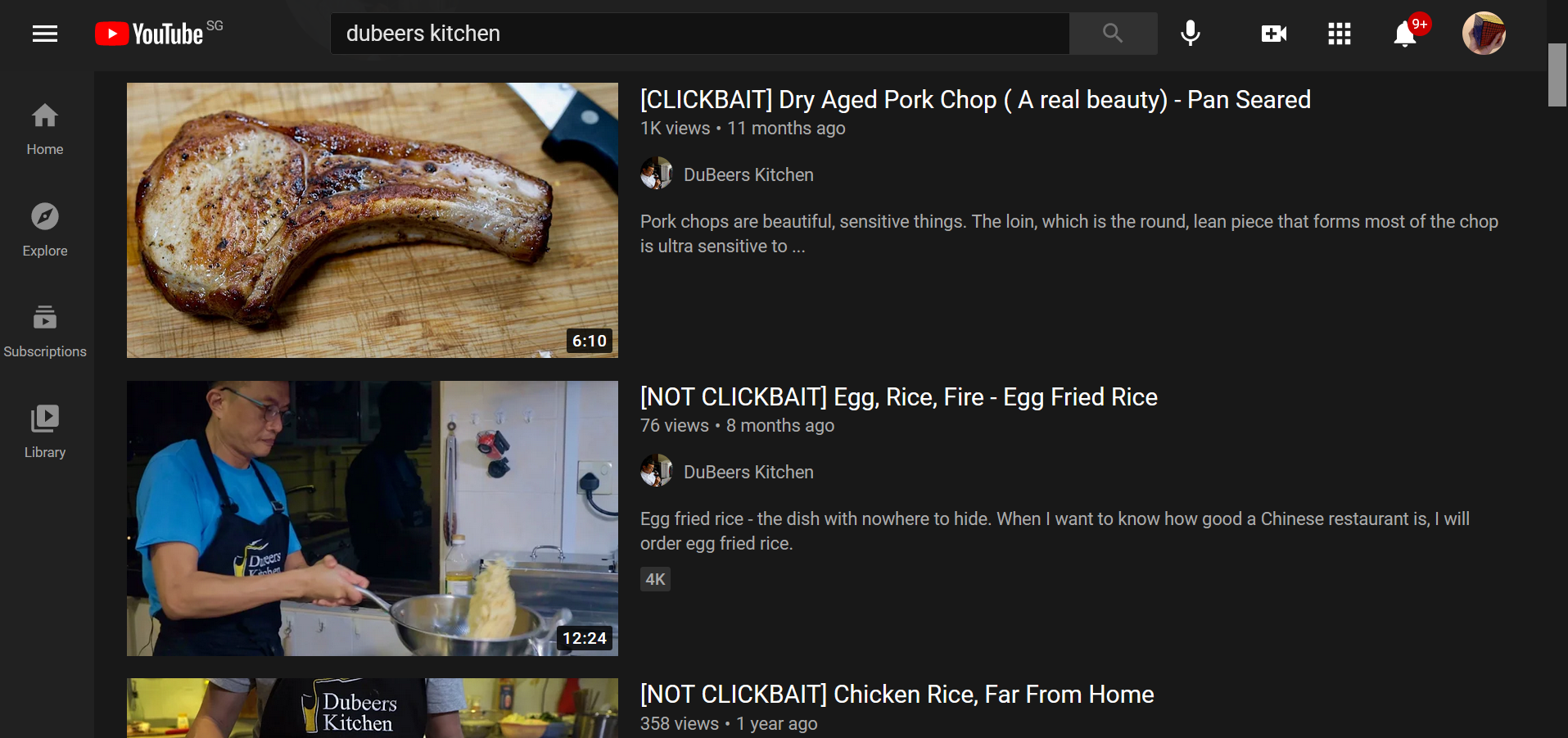
# Our model performs decently, near the average F1 score of the humans. No further analysis could be done as our data on human performance is biased as participation is voluntarily.

Although this may seem as a failure for our model, we think that this result is quite good, even though it only has seen 50 channels+170 extra videos, it is able to perform similarly to a human with a few years of “training” on the internet watching YouTube videos.

# GUI

We initially planned to use a tampermonkey script to edit the titles of the YouTube videos, but ran into some issues with integrating the python script into javascript.

Here is a prototype of how we want our userscript to work. When the user searches for videos in YouTube, they can tell whether a video is clickbait or not. (Currently, the userscript just randomly decides whether a video is clickbait or not).



Thus, we decided to use tkinter to implement a simple GUI for our model. The user inputs a YouTube url and receives the verdict (clickbait or not clickbait) as well as the score the model gives. The GUI also displays the title and thumbnail of the YouTube video to verify that the user requested the correct url.

The models used for the GUI are loaded using pickle.

|  |  |
| --- | --- |
|  |  |

Conclusion

Machine Learning can be used for identifying clickbait videos given a large enough and varied dataset, and our model achieves a decent accuracy and F1 score.

Limitations

Our data collection could have been improved. This is because labelled videos based on channels are not very accurate and could differ from hand-labelled videos as the model may end up associating unrelated channel characteristics (YouTuber faces, YouTube series titles) to clickbait characteristics instead of videos. This is evident from the difference in accuracies and F1 scores between the validation set and test set.

Furthermore, the size of the hand labeled videos was very small. We believe that we could have gotten a better accuracy on the hand labeled videos if we had more quantity and variety in our hand labeled dataset. We could potentially ask for other humans to help and collect data while casually watching YouTube. This also allows for videos used in training to more accurately reflect what people are watching normally.

This brings up another problem about bias in our dataset, as the videos (and channels) are chosen by only 2 people, the dataset does not have enough variance and are mostly videos and teenage boys would watch, despite trying to mitigate that by trying spread our videos across 8 categories, it is possible that the 8 categories we chose does not represent what the the general population would consume.

Another issue is that while clickbait title or clickbait thumbnail would result in a clickbait video, a clickbait video may not necessarily have both a clickbait title and a clickbait thumbnail, hence our labelling is slightly inaccurate.

Despite the objective description, being “clickbait” is still open to interpretation, as shown by the low accuracy and F1 scores across human volunteers.

On the hardware side, our CNN takes very long to train, and thus we could not tune the model perfectly due to time constraints. We could explore using online computing platforms (AWS, Google) to speed up computation.

[1] <https://github.com/ageitgey/face_recognition>

[2] <https://knowyourmeme.com/memes/soy-boy-face-soyjak>

[3] <https://www.machinelearningplus.com/nlp/lemmatization-examples-python/>

[4] <https://docs.google.com/spreadsheets/d/1aFLYyHhLeXBAFCSs1p2bCcdlKG6eCjGAmbxWhH17-_c/edit?usp=sharing>