

Data Science YouTube channels Video Metadata

MSIS 2508, Machine Learning
Project

Anson Hsu, Aidan Nguyen, Ryan Wong,
James Baguley

Table of Content

- 1: Project Goal
2. Data Description
- 3: Data clean-up
- 4: EDA/ Data preparation
5. Comprehensive Information Approach
- 6: Thumbnail-based Approach
7. Text-based Approach
8. Conclusion
9. Appendix

1: Project Goal

For this research endeavor, our objective is to explore alternative methodologies distinct from conventional machine learning approaches, particularly focusing on the utilization of numerical and categorical datasets to forecast the view count of YouTube videos. The dataset sourced from Kaggle encompasses not only numerical and categorical data but also includes image data in the form of thumbnail image links, as well as text data comprising video titles and descriptions.

Although models built solely on image or text data may not exhibit comparable performance to those leveraging comprehensive datasets, this study aims to present alternative methodologies by considering datasets that are text-like or image-like in nature, thereby enabling predictions regarding YouTube video view counts. Additionally, the thumbnail-based and text-based models are expected to offer valuable insights into how thumbnail images, video titles, and textual descriptions influence the view count of YouTube videos.

2. Data Description

This dataset contains metadata of around 60 Data Science YouTube channel videos and 44261 video metadata. Following are the useful features to our models from the datasets.

Numerical Features:

- viewCount: The number of view of video (This is the target variable)
- durationSec: The duration of the YouTube video in seconds.
- likeCount: The number of likes received by the video.
- dislikeCount: The number of dislikes received by the video.
- commentCount: The number of comments posted on the video.
- days_since_published: The number of days since the video was published.

Categorical Features:

- channelTitle: The title of the YouTube channel that uploaded the video.

- videoCategoryLabel: The category label assigned to the video (e.g., Technology, Education, Entertainment).
- definition: Indicates whether the video is available in high definition (HD) or standard definition (SD).
- caption: Indicates whether the video has captions or not.
- licensedContent: Indicates if the video content is licensed.
- thumbnail maxres: the web link format of the thumbnail image

Regarding missing values in the thumbnail links, it's worth noting that $34192/44261 = 77.25\%$ of the data has successfully extracted thumbnail links, resulting in a dataset with a majority of videos having associated thumbnail images for analysis.

3: Data clean-up

In our machine learning and deep learning project in data science, we encountered missing values in several features, categorized into three distinct types. Our approach involves addressing these missing values individually for each type which is shown in [Figure1](#).

Firstly, a group of features including "videoCategoryID," "videoCategoryLabel," "duration," "durationSec," "dimension," "definition," and "caption" exhibit missing values for the same video, as illustrated in [Figure2](#). Given the consistency of missing values within this group, removing the corresponding row is deemed acceptable without significant impact on the dataset. Additionally, three missing values in the "viewCount" feature necessitate the removal of these rows, as imputing missing values in the target feature is not considered appropriate.

Secondly, upon examination of videos with missing values in "likeCount," "dislikeCount," and "commentCount," we observed that these videos were either erroneous or deactivated by the YouTube content creators (shown in [Figure3](#)). Consequently, treating these missing values as zeros or disregarding them as missing would be inaccurate. In addition, the likeCount and dislikeCount have missing value simultaneously which is shown at [Figure4](#). Our proposed solution involves a two-step process: firstly, filling the

missing values in "likeCount," "dislikeCount," and "commentCount" columns with -1 to denote missing data; secondly, creating two binary columns, namely "like_dislike_error" and "comment_error," where 0 signifies the absence of missing values and 1 indicates the presence of missing values. The creation of these binary columns is depicted in [Figure5](#).

Thirdly, the missing values in the "licensedContent" feature do not require imputation, as they signify the absence of a licensee for the corresponding videos. This aspect will be addressed using the SimpleImputer method when constructing the scikit-learn pipeline.

4: EDA/ Data preparation

After cleaning up the data, exploratory data analysis (EDA) was conducted to gain insights into the dataset through visualizations. The analysis revealed that the duration of videos has little impact on view counts, as indicated by [Figure5](#), where most videos exhibit similar view counts regardless of duration, with some outliers present. Similarly, [Figure6](#) illustrates that the publication date of videos does not significantly affect view counts, further emphasizing the importance of content in attracting viewers.

Examining the number of videos posted versus view counts by channel title ([Figure7](#)) and category ([Figure8](#)) highlighted that content quality plays a crucial role in viewer engagement. Channels with varying posting frequencies displayed similar view counts, suggesting that content quality outweighs quantity in attracting viewers. Additionally, certain video categories, such as "Comedy," garnered significantly higher view counts compared to others, indicating viewer preferences.

The combined analysis of video attributes like definition, captions, and licensed content ([Figure9](#)) indicated that higher-quality videos tend to attract more viewers. Videos with higher resolution, subtitles, and licensed content generally achieved higher view counts, emphasizing the importance of video quality alongside content.

Further investigation into optimal posting times ([Figures 10,11,12,13](#)) revealed that posting videos during weekends and specific times of the day, such as late evening or late afternoon, results in higher view counts. These findings provide valuable insights for content creators aiming to maximize viewer engagement.

We decided that for better compatibility with the classification models we were planning on using and for overall interpretability of results we should divide our target, view count, into 5 different categories representing very low views, low views, medium views, high views and very high views. This was done by separating view counts into 5 bins, 1 bin for every 20th percentile up to 100, then inserting this as a new categorical variable back into the dataframe.

Performing EDA grouped by the new view category variable, we were able to determine that the 100th percentile of all videos had a far higher average views than the lower percentiles ([Figure 14](#)). By compiling counts of likes, dislikes and comments into a new variable “Engagement,” we were also able to determine that these top percentile videos also commanded the vast majority of consumer engagement, though subject to more variance than videos in lower view categories ([Figure 15](#)). We also looked at the like/dislike ratio by view category, again by creating a new variable that divided like count by dislike count, which revealed that a more diverse range of opinions are shared on videos in the middle percentiles compared to those in the bottom and top percentiles ([Figure 16](#)). Last we chose to observe mean view count (logged to provide a more interpretable visualization) by the most popular genres and channels, which resulted in a return of channels mostly focused on the “Education” and “People and Blogs” genres with a few outliers ([Figure 17](#)).

5. Comprehensive Information Approach

Our Comprehensive approach involved using all of the available columns in our data in order to predict the view category of a video. We tested several different classification models to see which could do this the most accurately. Due to the relatively large size of the data, we used a split of 0.9 train to 0.1 test for all of the models.

5.1: Decision Tree

The first model we tested was a decision tree classifier. Preprocessing for this model involved first splitting the data into numerical columns and categorical columns, applying Simple Imputer to both of them to remove any NA values, and then applying One Hot Encoder to the categorical features to transform into a format acceptable by the model. Both types of columns were then fed into the decision tree classifier ([Figure 18](#)).

We then applied GridsearchCV to find the depth for this decision tree that would provide the best accuracy, experimenting with different ranges. We attained the best result by setting the max depth range from 1 to 10, which returned a best depth of 9 with a training and test score of 0.777915 and 0.747661. Investigating the feature importance of this model revealed that the most significant feature by far was like count, followed by days since publication, dislike count, and duration ([Figure 19](#)).

5.2: Random Forest

We followed a similar process for the random forest model, using the same preprocessing pipeline as the decision tree and employing GridsearchCV to find the optimal number of features and the depth that would give us the best testing score. In this case it gave us a max depth of 14 and max number of features of 20, which resulted in a training and test score of 0.879728 and 0.766959, higher than the decision tree model but less well-fitted ([Figure 20](#)). Feature importance has also changed slightly, listing like count, dislike count, and comment count as the 3 most important features in closer proportions than the decision tree ([Figure 21](#)).

5.3 Gradient Boosting

For gradient boosting, the next model in our analysis, we changed the preprocessing pipeline by applying standard scaler to the numerical columns only. As gradient boosting relies on numerical features being similarly scaled compared to a decision tree or random forest model, this seemed a necessary step. After this we again tuned the model using GridsearchCV, this time searching for the optimal number of estimators and learning rate. This returned an optimal learning rate of 0.1 and an optimal number of

estimators of 100, which provided a training and test score of 0.867347 and 0.780409, the highest among the models we tested ([Figure 22](#)).

5.4 Logistic Regression

We prepared the data by applying preprocessing steps to handle both categorical and numerical columns. This involved using a Simple Imputer for missing values and applying One-Hot Encoding for categorical features and Standard Scaler for numerical ones. These steps were encapsulated in a preprocessing variable. Subsequently, we constructed a pipeline incorporating the preprocessing steps and logistic regression model. We further refined the model by tuning the hyperparameter C using GridSearchCV, exploring values such as 0.01, 0.1, 1.0, 10, and 100. The GridSearchCV process determined that the optimal C value was 100, achieving an accuracy of 0.71 on the training set and 0.7 on the testing set ([Figure23](#)).

5.5 Kernelized SVM

Next model we tried was Kernellized SVM. We applied the same preprocessing steps as we did for Logistic Regression to ensure our data was properly prepared. Using a pipeline with two stages, we first processed the data and then trained an SVC model using the standardized data.

Following this, we employed GridSearchCV to optimize the hyperparameters C and gamma. For C, we constructed an array with seven logarithmically spaced values ranging from 10^{-3} to 10^3 . Similarly, for gamma, we generated seven values logarithmically spaced between 10^{-3} and 10^3 , dividing these values by the number of samples in the training set.

After performing the grid search, we determined that the optimal values for C and gamma were 100 and 0.032, respectively. These parameters yielded an accuracy of 0.64 on the training set and 0.59 on the testing set ([Figure24](#)).

5.6 MLP Classifier

In our final modeling approach, we employed the MLP classifier. Like with previous models, we applied the same preprocessing steps using our established pipeline. Utilizing grid search cross-validation, we explored various configurations of the model's hidden layers. These configurations ranged from single-layer networks with differing numbers of neurons (10, 50, and 100) to multi-layer networks, including combinations

like (10, 10) and (50, 50). Ultimately, the best accuracy was achieved with a single-layer network comprising 10 neurons. On the training set, this yielded an accuracy score of 0.742, while on the testing set, it resulted in a score of 0.714 ([Figure25](#)).

5.7 Summary

In conclusion, among the six models evaluated, Random Forest emerged as the top performer with an impressive accuracy score of 0.77 on the testing set, closely followed by Gradient Boosting with a score of 0.76 ([Figure26](#)). The superior performance of these models can be attributed to their decision tree-based approach, which excels at capturing intricate data relationships while mitigating overfitting. Decision tree algorithms are known for their ability to handle non-linear relationships and interactions between features, making them particularly effective for complex datasets like ours. This adaptability, combined with ensemble techniques used in Random Forest and Gradient Boosting, further enhances their predictive power, resulting in higher accuracy compared to other models.

6: Thumbnail-based Approach

We are focusing on the implementation of Convolutional Neural Network (CNN) models specifically tailored for processing image data, such as the Thumbnail images sourced through web scraping. These images are resized to 128 x 128 pixels, as depicted in [Figure27](#).

The rationale behind employing CNN models lies in their ability to effectively capture and learn intricate patterns within images. By utilizing convolutional layers, CNNs excel at discerning visual features and structures that are crucial for attracting viewers' attention and encouraging them to engage with the content. For instance, in the context of a data science YouTube channel, elements like the presence of an Excel icon or other visually appealing features can significantly impact viewer interest and engagement.

CNN models undergo a process of pattern recognition and classification, where they learn from training data to identify key components and patterns within images. This learned knowledge is then utilized during testing to classify the viewCategory associated with the videos. This approach leverages the strengths of CNN architectures, which are specifically designed to handle image data and extract

meaningful features for analysis and prediction purposes.

Regular CNN Model

Data Preparation:

To prepare the data for our machine learning and deep learning model, we initially transform the image-type data into numpy arrays, resulting in an input matrix X with dimensions (34192, 128, 128, 3). Additionally, we encode the target variable "viewCategory" using one-hot encoding as follows:

- [1,0,0,0,0] represents viewCount group 1
- [0,1,0,0,0] represents viewCount group 2
- [0,0,1,0,0] represents viewCount group 3
- [0,0,0,1,0] represents viewCount group 4
- [0,0,0,0,1] represents viewCount group 5

This transformation is essential for handling the multi-class classification problem before inputting the data into our model. Finally, we split the dataset into training and testing sets with a ratio of 9:1.

Model Architecture:

As depicted in [Figure28](#), our model architecture comprises two convolutional layers with a kernel size of 3x3, a stride of 1, and 32 and 64 filters, respectively. Adjacent to each convolutional layer are two max-pooling layers with a pool size of 2 and strides of 2. Following these layers, we incorporate a Global Pooling layer, which aggregates the features learned by the convolutional layers across the entire image, providing a condensed representation of the image features. This is then followed by a Flatten layer and two Fully Connected neural network layers equipped with batch normalization and dropout mechanisms. The output layer utilizes the softmax function with a dimension of 5, aligning with our multi-classification problem with five bins.

Hyperparameters and Training Process:

For training, we set the learning rate to 1×10^{-3} , the batch size to 256, and conduct 30 epochs, as outlined in [Figure29](#). These hyperparameters and training iterations were determined based on empirical experimentation and optimization.

Training and Testing Results:

The testing accuracy, as shown in [Figure30](#), is recorded at 0.47398. Additionally, the loss and accuracy metrics for both the training and testing sets, displayed in [Figure31](#), indicate no signs of overfitting but rather convergence after the tenth epoch. These results validate the efficacy of our model in accurately classifying the target variable across multiple classes.

Image Augmentation CNN Model

We conducted an experiment involving image augmentation techniques, including horizontal and vertical reflection, rotation at various degrees, and zooming in and out. These augmentation methods are visually demonstrated in [Figure31](#) showcasing the original images and [Figure32](#) exhibiting the augmented images.

We employed the same model architecture as a regular CNN and utilized identical hyperparameters and training processes. However, in this experiment, we introduced different augmented images solely into the training set to augment the training dataset, thereby enhancing the variety of patterns learned by the model. This augmentation strategy is particularly beneficial as variations in video thumbnail images, such as size and the degree of patterns like Excel icons, may manifest differently.

During training, we maintained the same batch size and epoch configuration, but each epoch consisted of a varied set of augmented images. The testing accuracy, documented in [Figure33](#), was recorded at 0.44064, indicating a slightly lower performance compared to the regular CNN model. Nonetheless, the loss and accuracy metrics for both the training and testing sets, as displayed in [Figure34](#), reveal no signs of overfitting but rather convergence after the tenth epoch. These findings affirm the efficacy of our model in accurately classifying the target variable across multiple classes despite the incorporation of image augmentation techniques.

7. Text-based Approach

To facilitate the training of our model for viewCategory classification, we leverage the Recursive Neural Network (RNN), a specialized type of artificial neural network tailored for processing sequential data by maintaining an internal memory or state. Unlike conventional feedforward neural networks, RNNs are structured with cyclic connections, enabling them to exhibit temporal dynamics and capture dependencies within sequences. This inherent capability allows RNNs to handle variable-length inputs and produce variable-length outputs, rendering them versatile for a wide array of sequential data processing tasks.

Data Preparation:

Our feature matrix X encompasses string variables such as video title and video description which is shown in [Figure35](#), while y represents the five bins of viewCategory. However, due to the challenge of feeding unequal-length string variables into the neural network model, we employ a transformation process to convert the texts into numerical datasets. This transformation not only represents the semantic meaning of text through mathematical transformations but also standardizes the length of the text data.

We adopt a tokenization method to convert the text into a numerical dataset, breaking down the text into individual words or subwords based on their frequency order. This approach facilitates granular analysis and processing of the text. As depicted in [Figure36](#), the tokenization process generates a dictionary for our X dataset, with the most frequent word, "http," assigned as the 1st word due to its high frequency in the datasets. Furthermore, we utilize the nltk package to clean stop words from the text data.

To standardize the text length, we employ a mathematical equation to determine the optimal length that covers the majority of the text. The equation $\text{Mean}(\text{num_tokens}) + 2 * \text{Std}(\text{num_tokens}) = 520$ is used to calculate the desired text length. Approximately 93% of the text data comprises less than 520 tokens, which are left unchanged. For texts longer than 520 tokens, we truncate the words to reach the desired length.

Conversely, for texts with fewer than 520 tokens, we apply the Padding method to fill the remaining space with zeros, as illustrated in [Figure38](#). Finally, we split the dataset into training and testing sets with a ratio of 9:1.

Regular LSTM model and GRU model

Model Architecture:

We have developed two distinct models as depicted in [Figure39](#). The first model architecture begins with an Embedding layer with a dimension of 512. This layer is responsible for transforming the input text data into dense numerical vectors of fixed dimensions, effectively capturing semantic relationships between words in the text data. Following the Embedding layer are three LSTM layers, designed to address the vanishing gradients or long-term dependence issues inherent in RNNs. Subsequently, the model incorporates three Fully Connected neural network layers equipped with batch normalization and dropout mechanisms. The output layer utilizes the softmax function with a dimension of 5, aligning with our multi-classification problem involving five bins.

As for the second model depicted in [Figure40](#), it shares a similar structure with the LSTM model described above but replaces the LSTM layers with GRU layers. GRU, being less complex and requiring fewer parameters to train compared to LSTM, presents a viable alternative in our model architecture.

Hyperparameters and Training Process:

We set the learning rate to 1×10^{-3} , the batch size to 256, and conduct 20 epochs as outlined. These hyperparameters and training iterations were determined through empirical experimentation and optimization.

Training and Testing Results:

The Regular LSTM model achieved a testing accuracy of 0.51842 as shown in [Figure41](#). However, the loss and accuracy metrics for both the training and testing sets displayed in [Figure43](#) indicate overfitting occurring after the fifth epoch, suggesting the

potential need for an early stop method to mitigate this issue. On the other hand, the regular GRU model outperformed the LSTM model with a testing accuracy of 0.54152 as depicted in [Figure42](#), but it also exhibits overfitting problems, as evidenced by the data in [Figure44](#).

Bi-direction GRU model and Bi-direction GRU with Attention model

We are conducting experiments to enhance our models by incorporating more sophisticated techniques into the GRU architecture, which has demonstrated superiority over LSTM. The bi-directional approach in GRU, as commonly utilized, involves integrating an additional backward RNN model. This strategy allows us to consider not only the relationships from left to right but also from right to left, thereby capturing bidirectional dependencies in the sequential data more comprehensively.

Furthermore, we are exploring the implementation of attention mechanisms in our GRU model. Attention mechanisms are designed to dynamically weigh the importance of different parts of the input sequence during model training. This method allows the model to focus more on relevant information while de-emphasizing less significant elements, potentially leading to improved performance and accuracy in sequence modeling tasks. The model architecture is shown in [Figure45](#) and [Figure46](#)

Training and Testing Results:

The Bi-direction GRU model achieved a testing accuracy of 0.55088 as shown in [Figure47](#). However, the loss and accuracy metrics for both the training and testing sets displayed in [Figure49](#) indicate overfitting occurring after the fifth epoch, suggesting the potential need for an early stop method to mitigate this issue. On the other hand, the Bi-direction GRU with Attention model underperformed the Bi-direction GRU model with a testing accuracy of 0.54649 as depicted in [Figure48](#), but it also exhibits overfitting problems after the seventh epoch, as evidenced by the data in [Figure50](#).

8. Conclusion

From the analysis of three distinct approaches, it is evident that comprehensive models

such as SVM and Random Forest exhibit superior test accuracy (0.767 for Random Forest) compared to image-based (0.474 for CNN) and text-based (0.551 for Bi-directional GRU) approaches. This observation is rationalized by the significant impact of key features like likeCount, dislikeCount, commentCount, and daysSincePublished, as identified by the Random Forest model, on the viewCount prediction. Despite the lower test scores relative to the Random Forest Model's accuracy of 0.767, the CNN, CNN AUG, LSTM, GRU, Bi-directional GRU, and Bi-directional GRU with attention models are analyzed further through confusion matrices (From [Figure51](#) to [Figure56](#)).

Upon inspecting these confusion matrices, it becomes apparent that misclassifications often occur within grids adjacent to the diagonal, indicating that misclassified labels are typically only one step away from the true label. This nuanced observation suggests that the models do not perform poorly but rather exhibit minor misclassifications. For instance, in [Figure51](#), for a true label of 1, the regular CNN model predicts 286 images as label 0 and 154 images as label 2. Similarly, 44 images are predicted as label 3, and 11 images as label 4. By assuming that these misclassified labels within one step from the diagonal are correct, both the image-based and text-based approaches could achieve accuracies close to 80%.

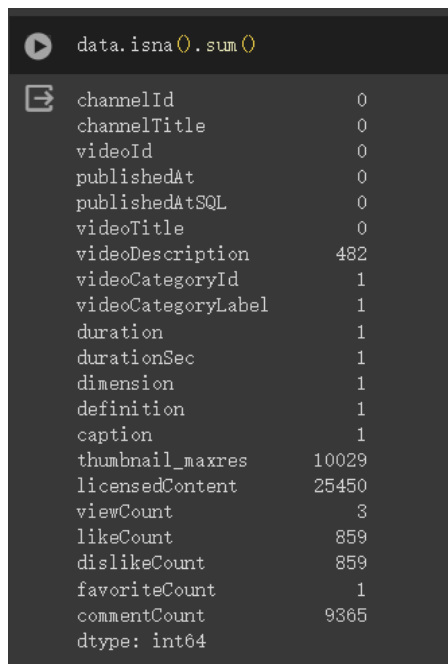
This analysis underscores the importance of considering the proximity of misclassifications to the true labels, highlighting the potential for improved accuracy by refining the classification criteria to encompass such nuances.

From a business perspective, these findings underscore the importance of leveraging diverse data sources and advanced modeling techniques for more accurate predictions in digital content viewership. Incorporating features related to viewer engagement and content characteristics, as identified by the Random Forest model, can significantly enhance the accuracy of predictive models. Furthermore, understanding the nuances of misclassifications, particularly in ordinal variables like viewCategory, can lead to more nuanced and accurate decision-making processes. Overall, our analysis provides valuable insights for content creators, digital marketers, and business analysts seeking

to optimize viewer engagement and maximize the impact of their digital content strategies.

9. Appendix

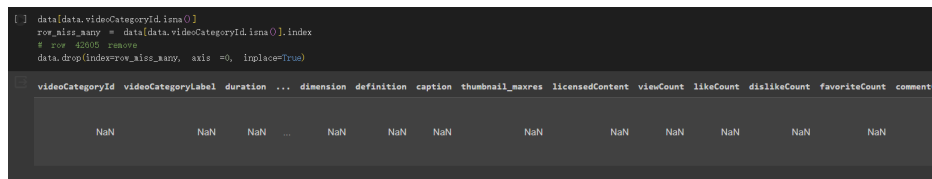
3: Data clean-up/preparation



```
data.isna().sum()
```

channelId	0
channelTitle	0
videoId	0
publishedAt	0
publishedAtSQL	0
videoTitle	0
videoDescription	482
videoCategoryId	1
videoCategoryLabel	1
duration	1
durationSec	1
dimension	1
definition	1
caption	1
thumbnail_maxres	10029
licensedContent	25450
viewCount	3
likeCount	859
dislikeCount	859
favoriteCount	1
commentCount	9365
dtype: int64	

Figure1



```
[ ] data[data.videoCategoryId.isna()]
row_missing = data[data.videoCategoryId.isna()].index
# row 42005 remove
data.drop(index=row_missing, axis =0, inplace=True)
```

videoCategoryId	videoCategoryLabel	duration	...	dimension	definition	caption	thumbnail_maxres	licensedContent	viewCount	likeCount	dislikeCount	favoriteCount	commentCount
NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure2

0:00 / 1:35

Seattle Snow Feb 8 2014. at Night.

ExcelsiFun
975K subscribers

25 | Share | Download

5.1K views · 10 years ago
Seattle Snow on Feb 8, 2014. Isaac goes out into the snow at night, then considers what tomorrow's huge snow drifts will bring, then throws a snow ball at the camera. ...more

Shop the ExcelsiFun store

Power Query is fun! Unisex Classic Pullover Hoodie
\$38.99

You come to excelsifun to learn Fun, Free & Efficient Power Query tips. Now you can show off this Power Query Data Transforming Fun with t-shirts and mugs! Check out the excelsifun store here at Teespring & YouTube :) Custom designed... Spring |

Comments are turned off. [Learn more](#)

Figure3

likeCount	dislikeCount
NaN	NaN
NaN	NaN
NaN	NaN
NaN	NaN

Figure4

4: EDA

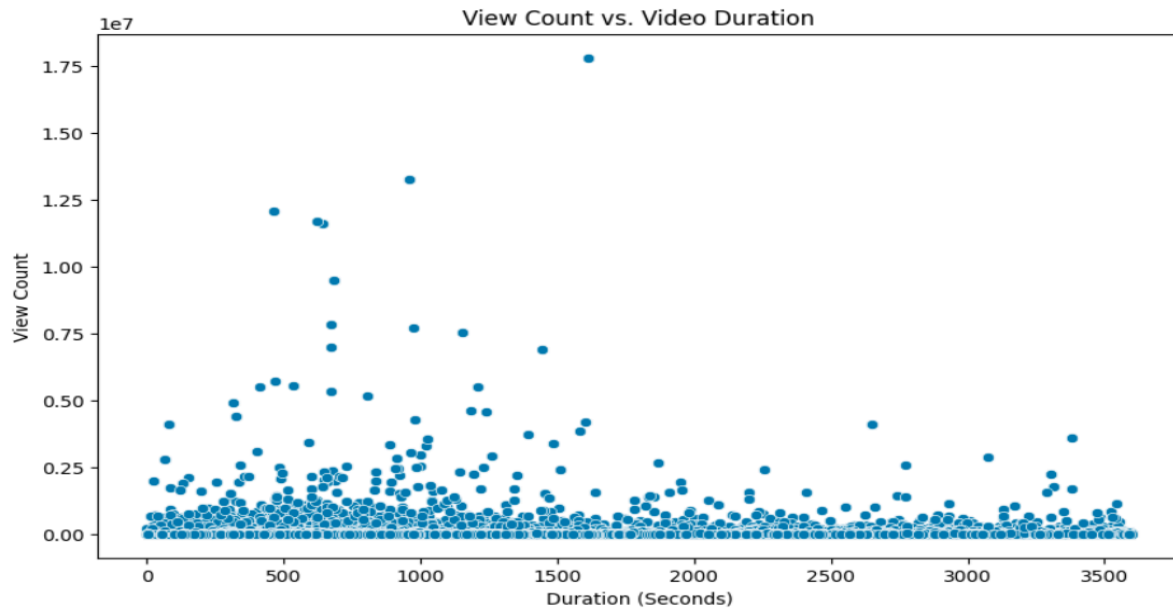


Figure 5

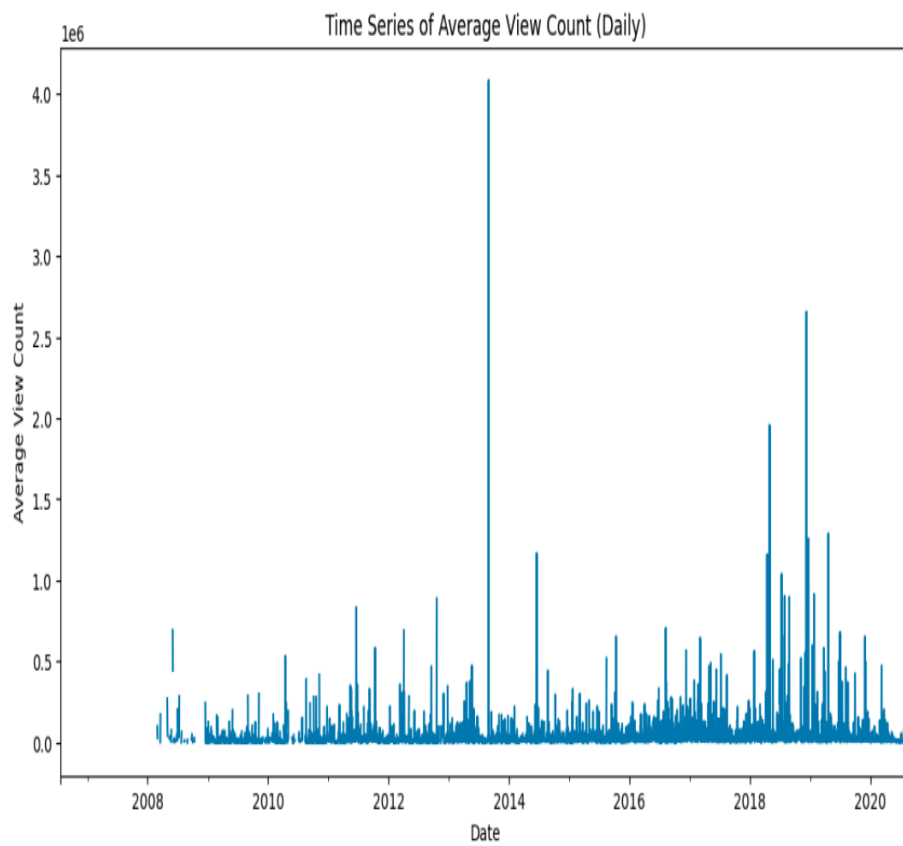


Figure 6

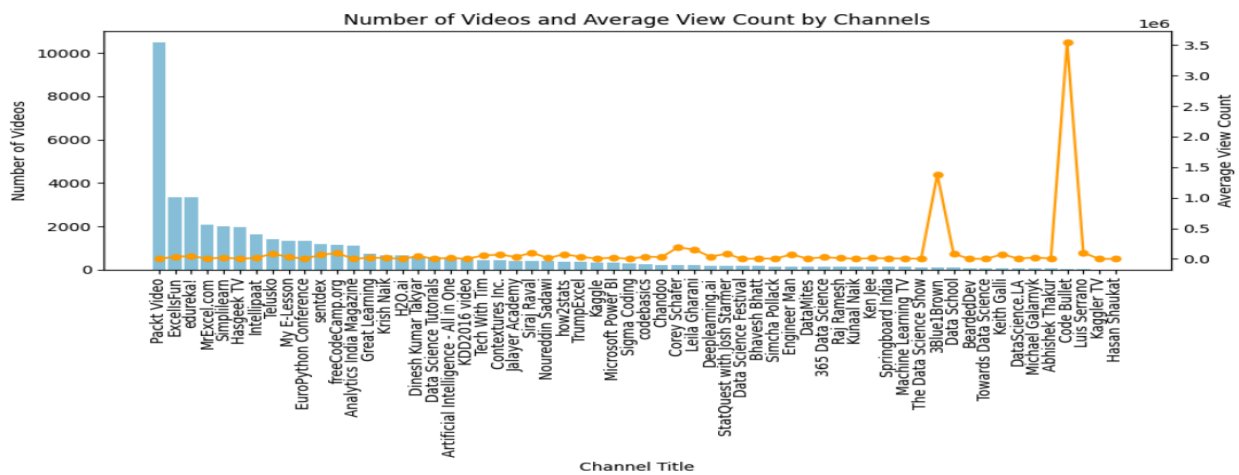


Figure 7

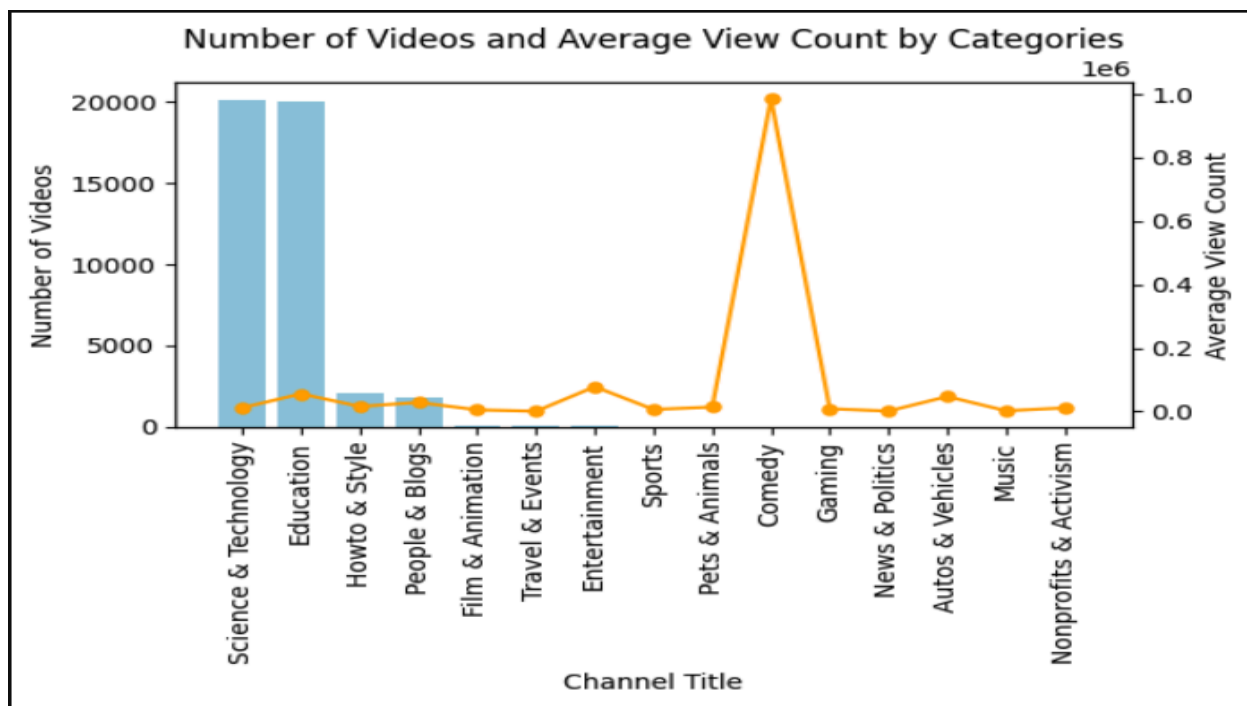


Figure 8

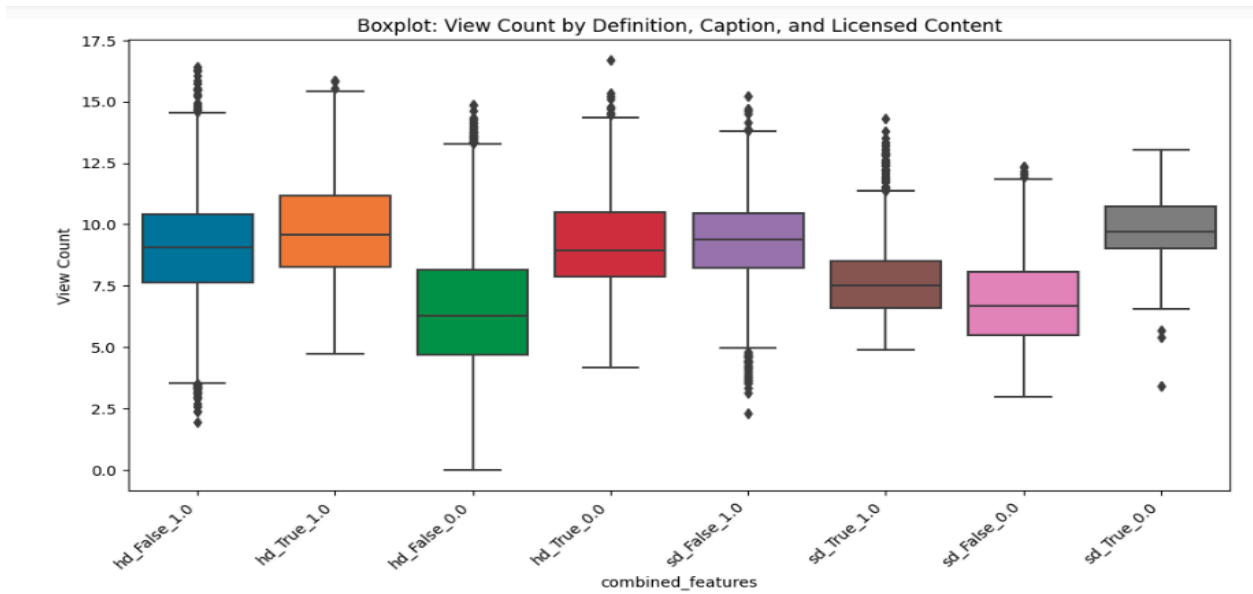


Figure 9

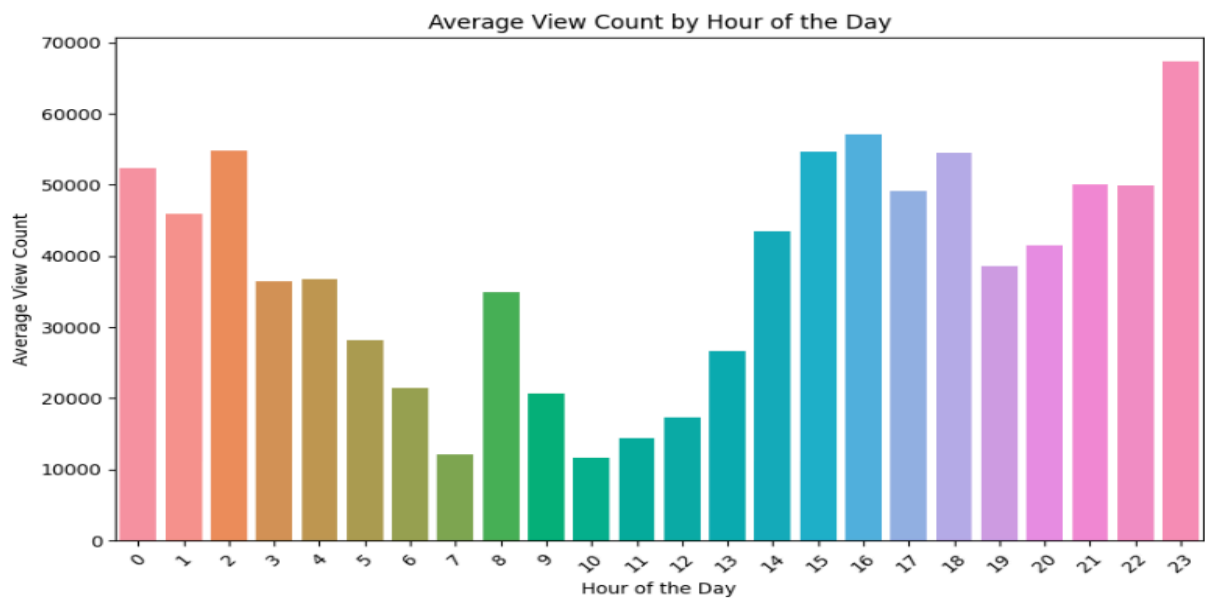


Figure 10

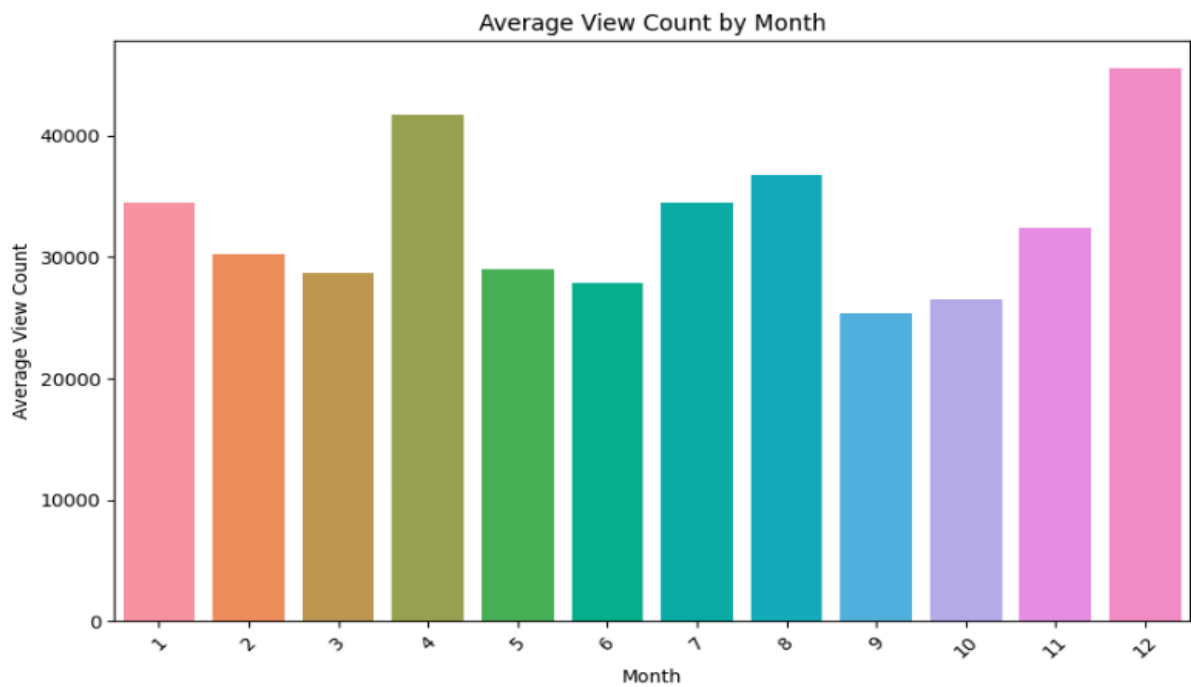


Figure 11

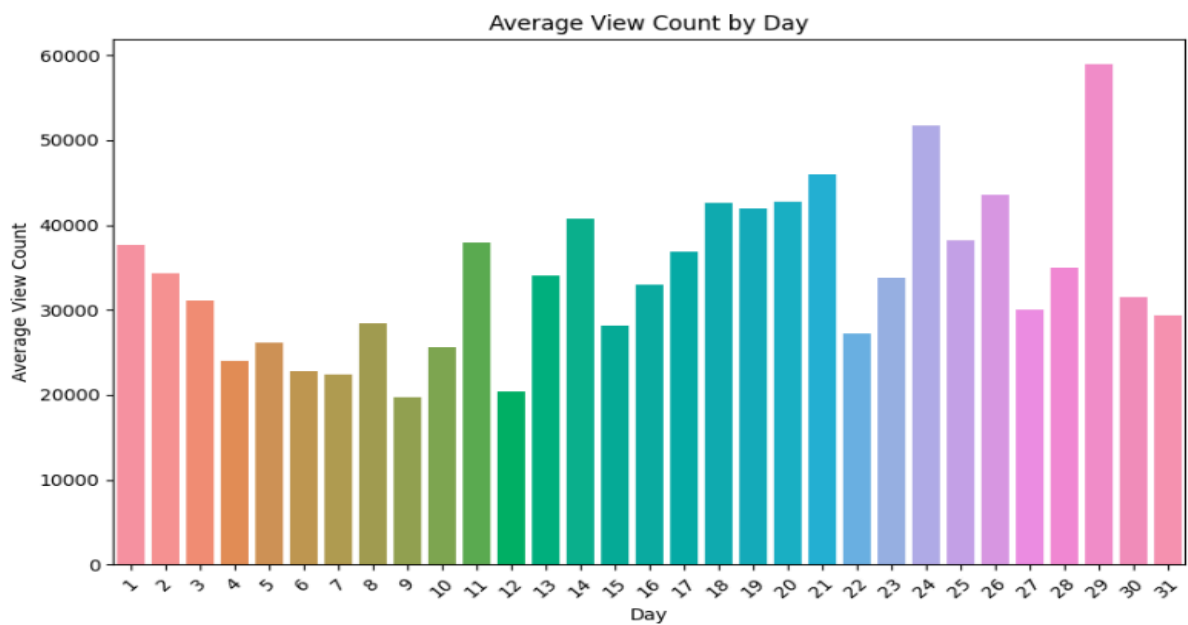


Figure 12

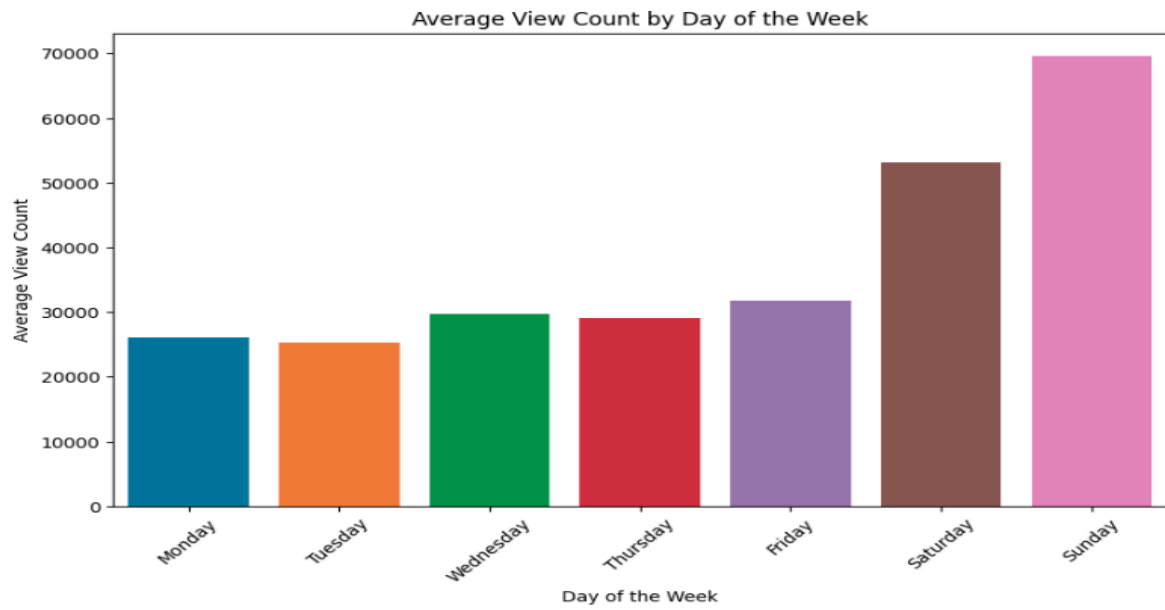


Figure 13

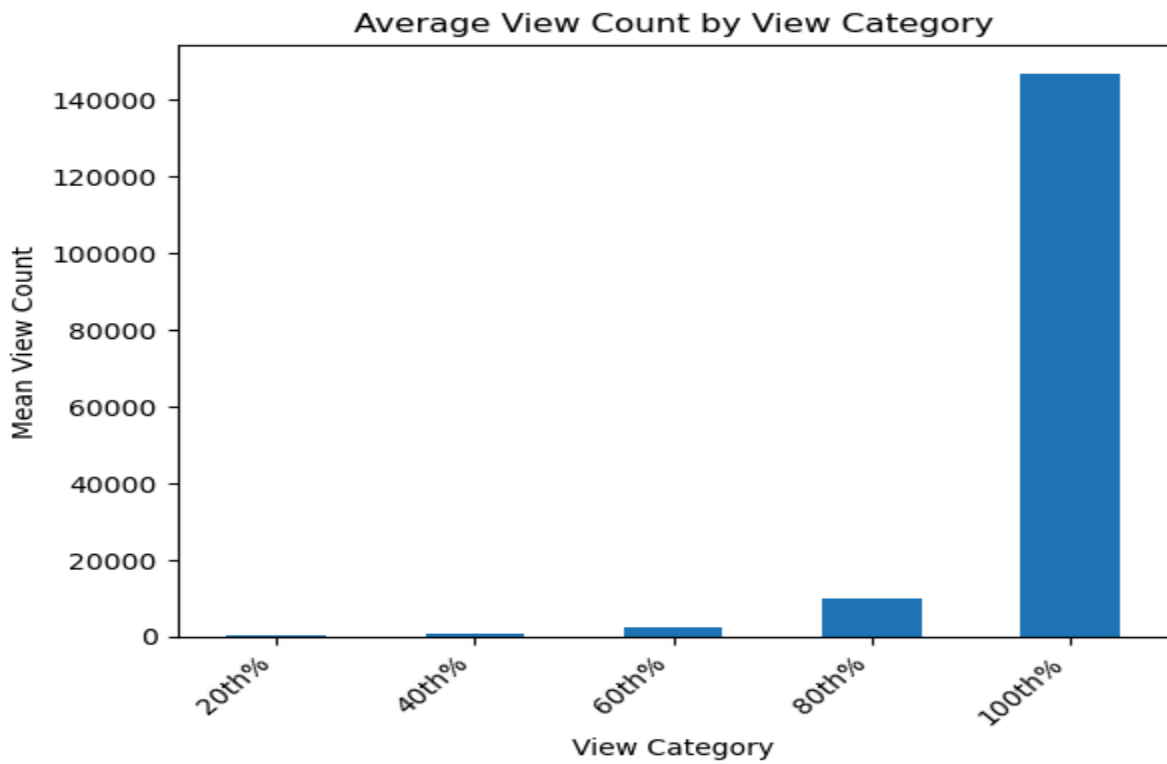


Figure 14

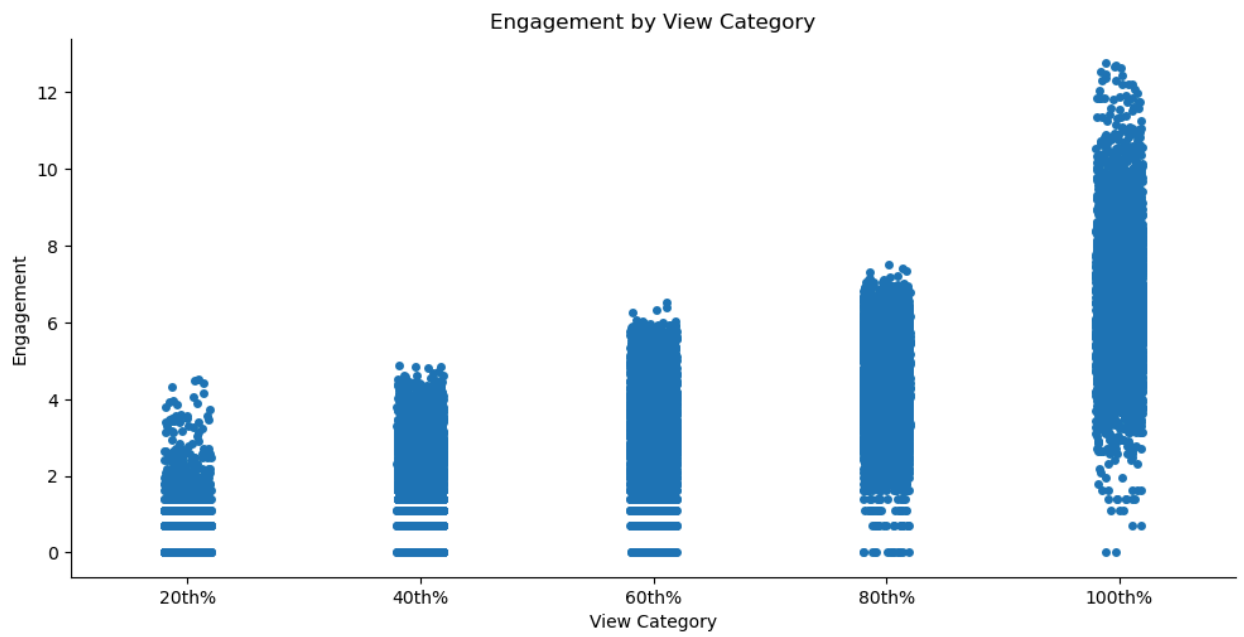


Figure 15

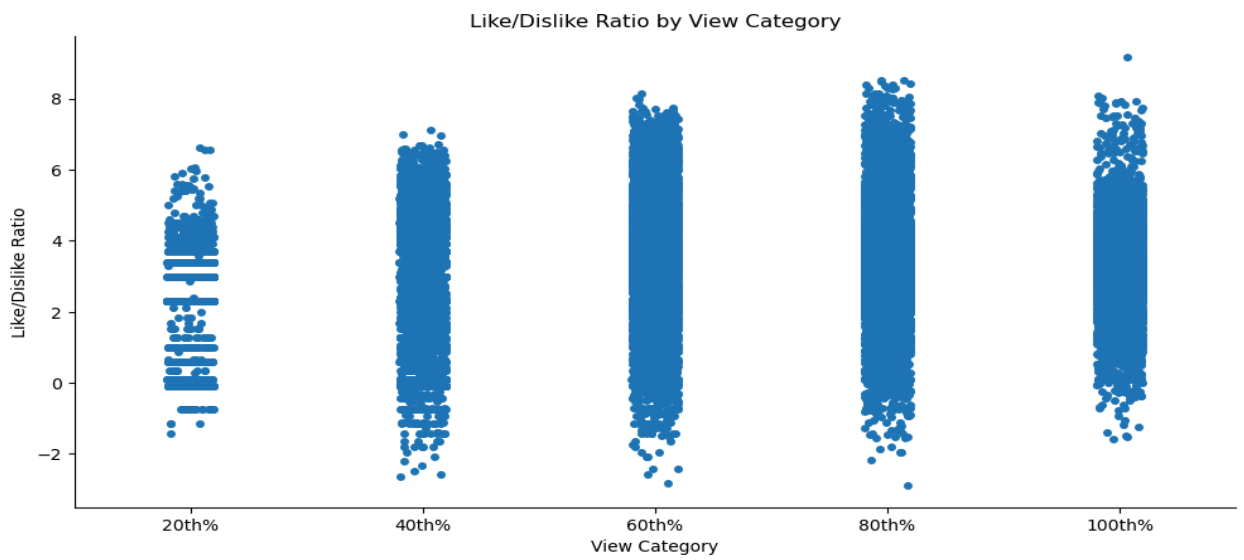


Figure 16

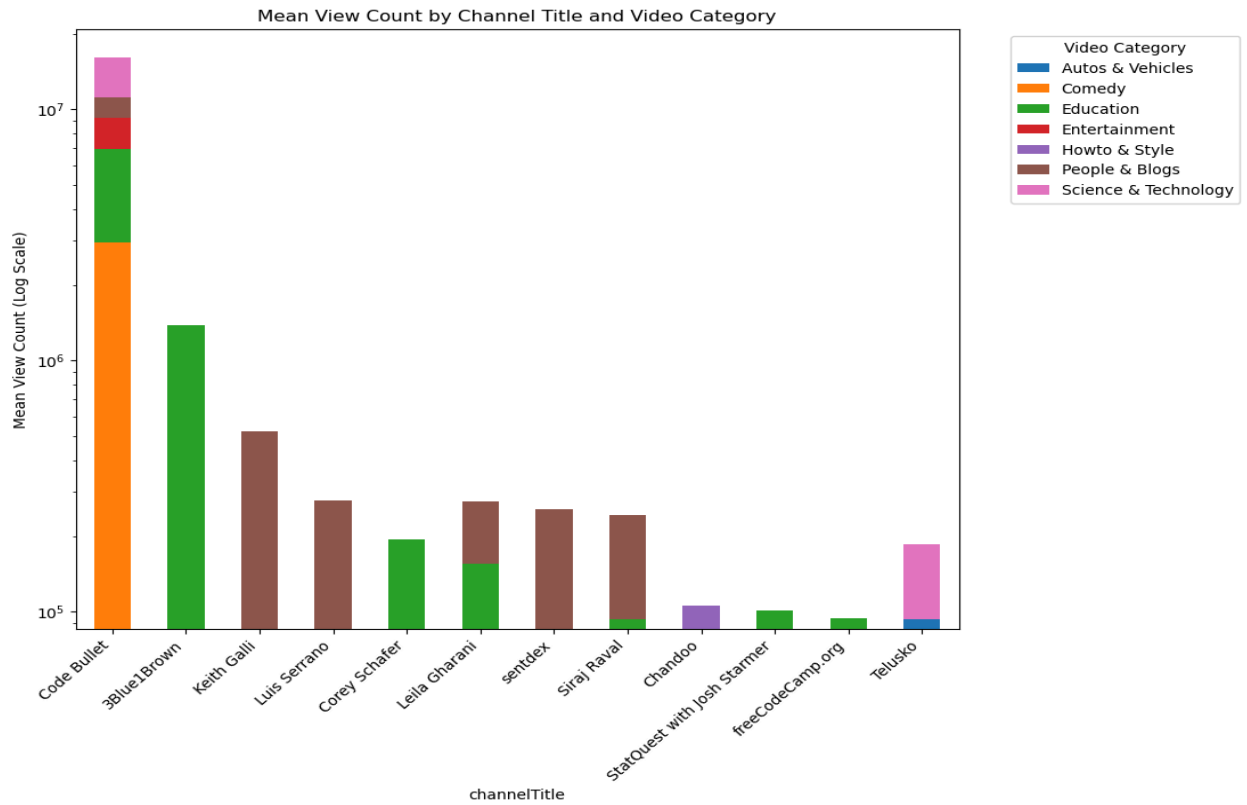


Figure 17

5. Comprehensive Information Approach

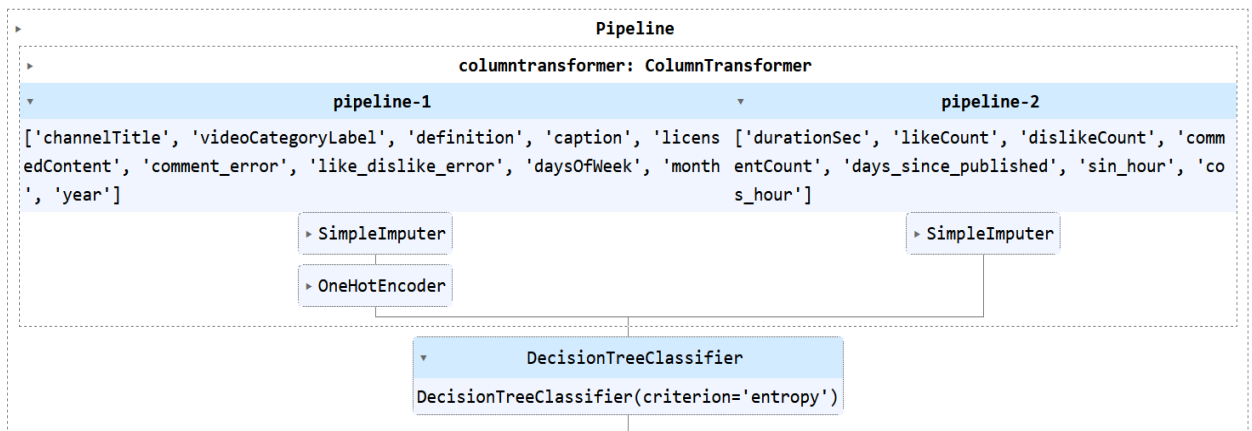


Figure 18

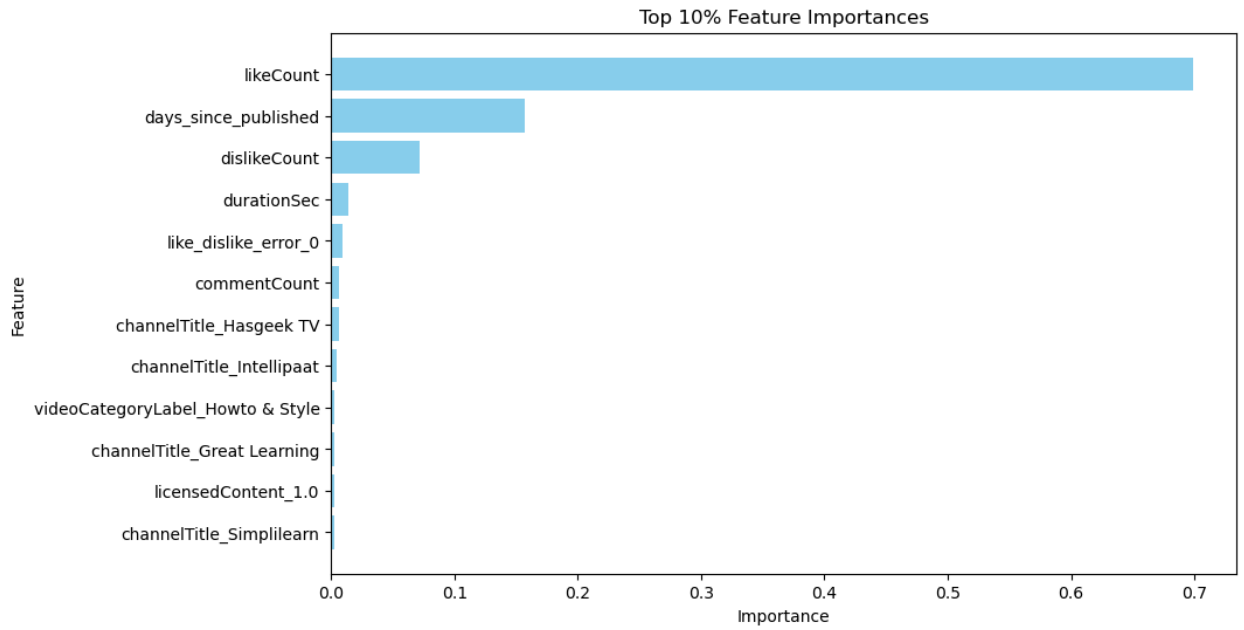


Figure 19

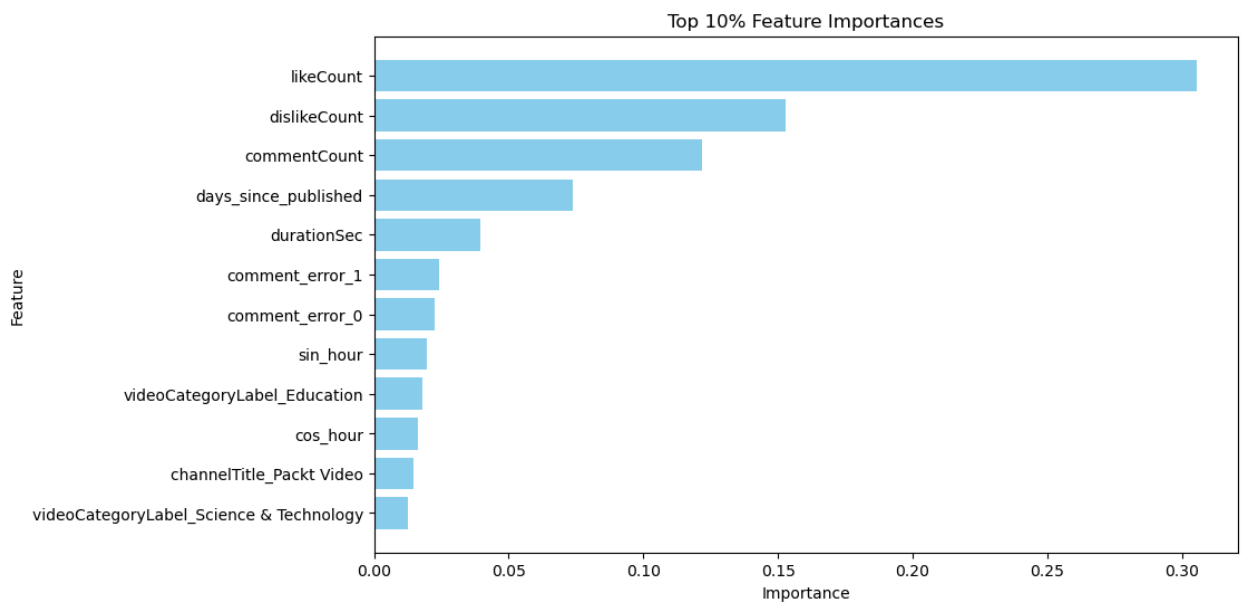


Figure 20

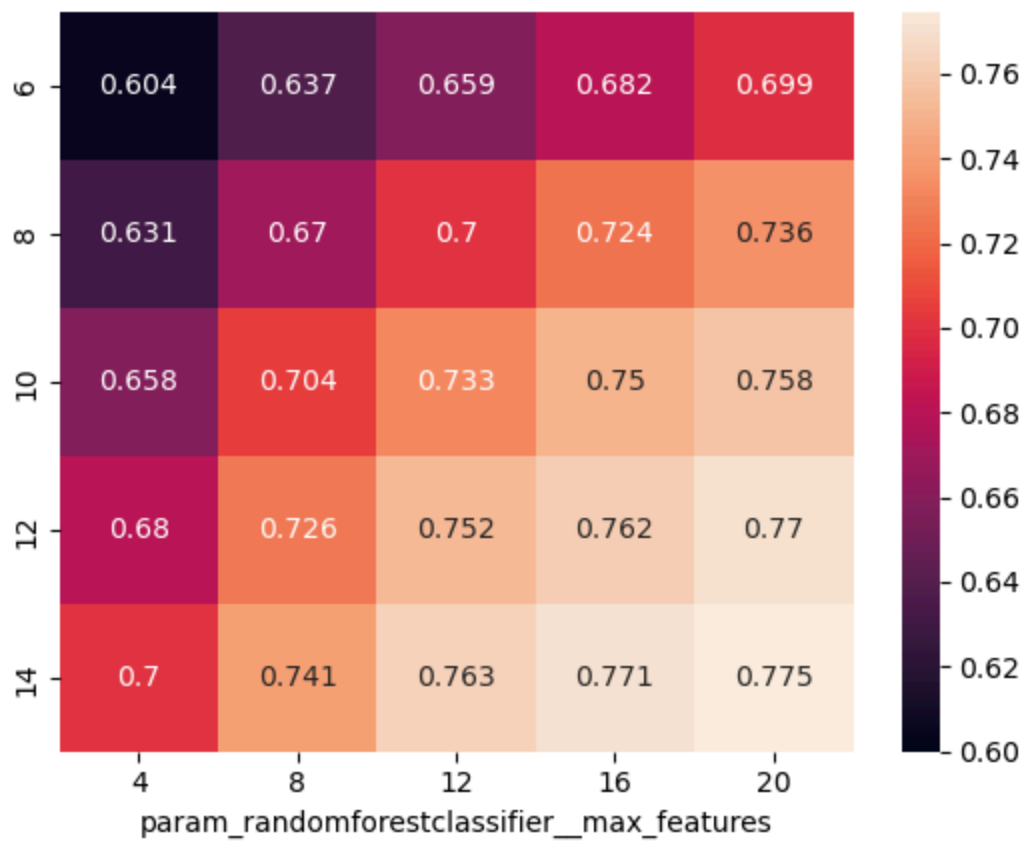


Figure 21

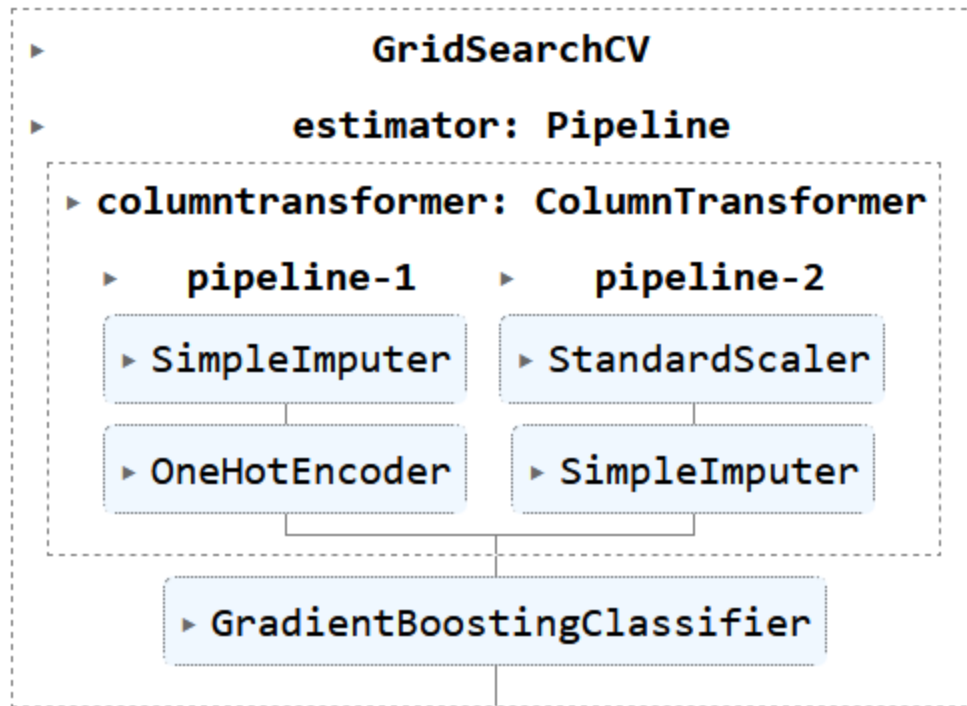


Figure 22

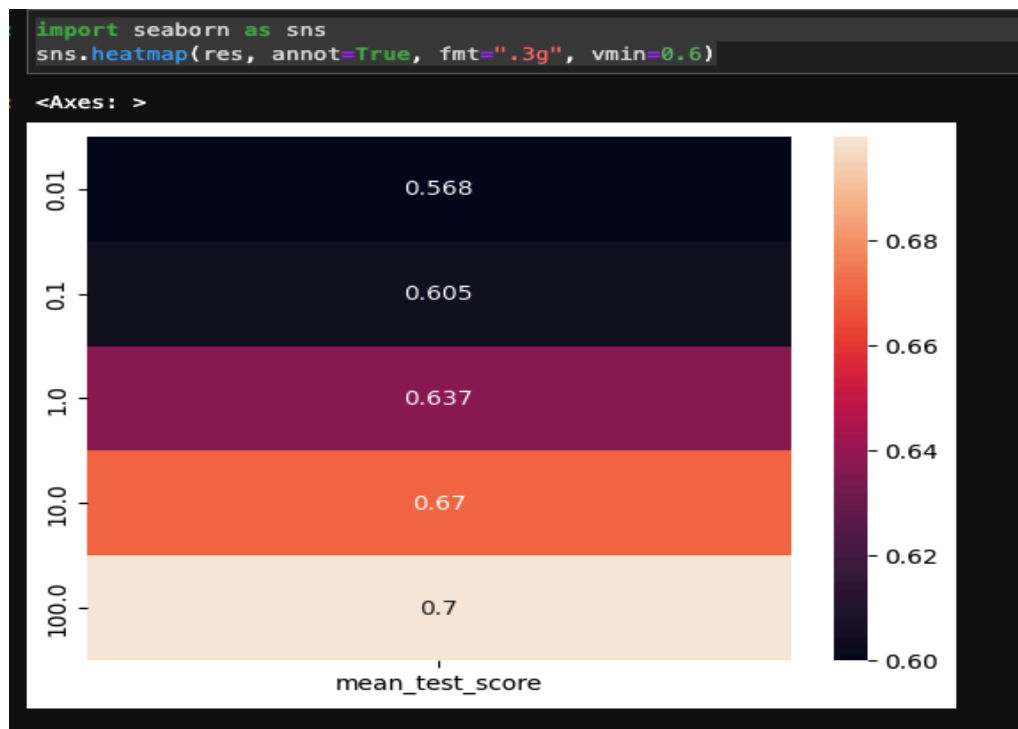


Figure 23

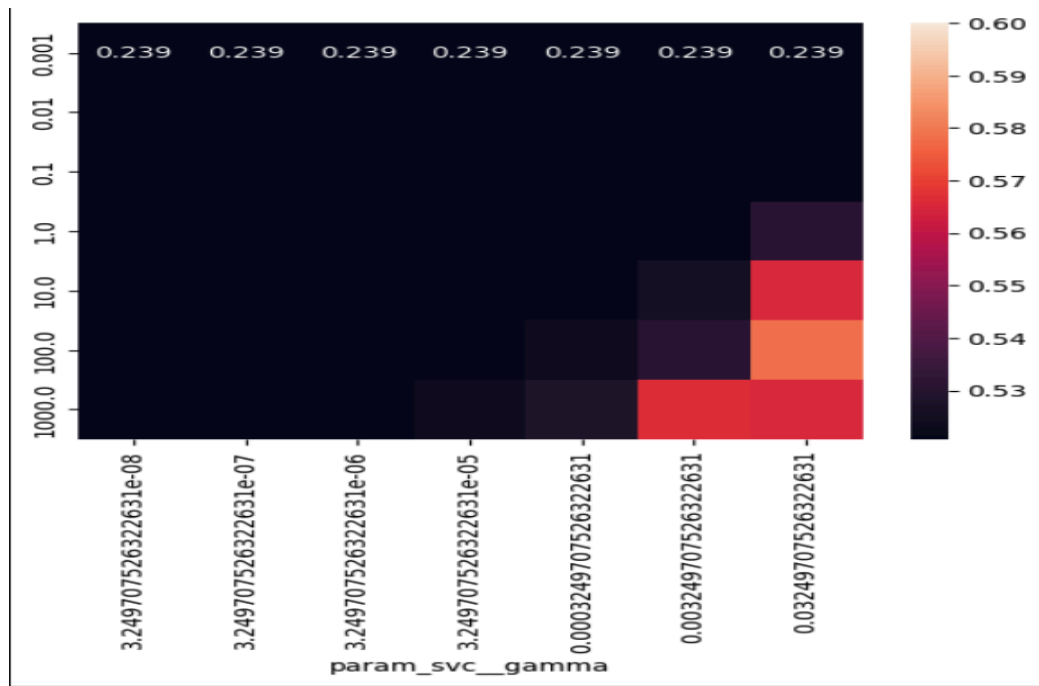


Figure 24

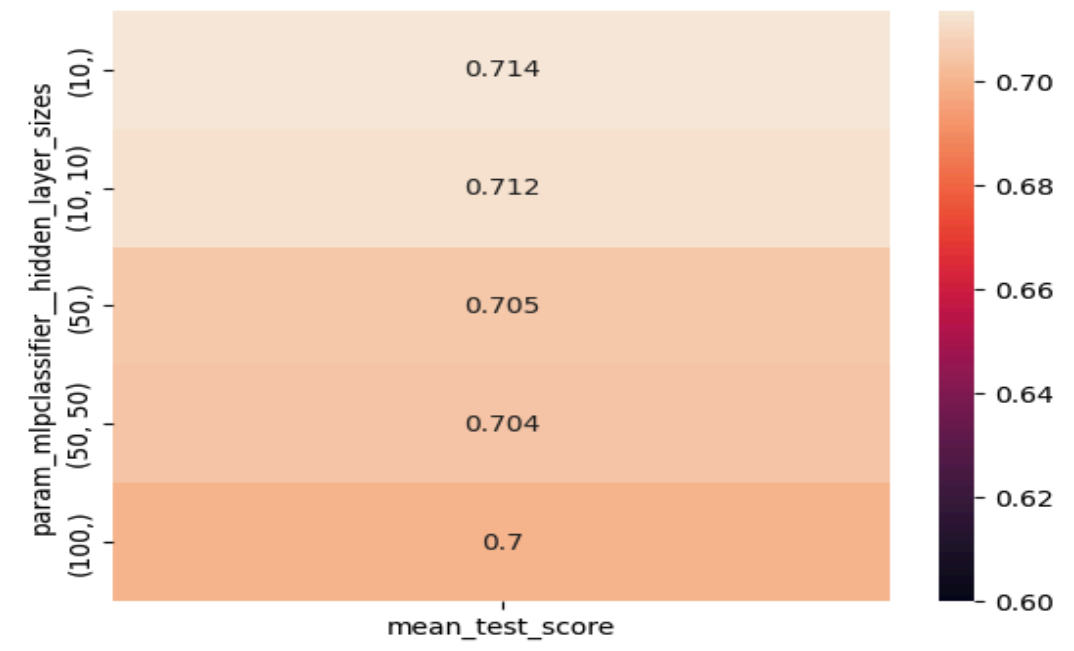


Figure 25

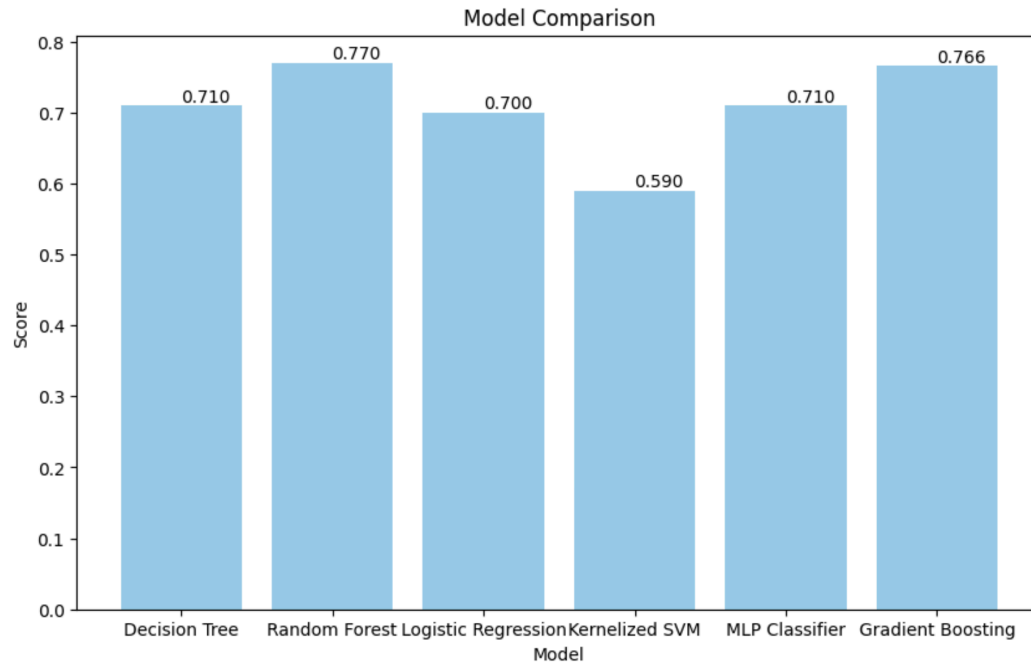


Figure 26

6: Thumbnail-based Approach



Figure 27

Model: "sequential_48"

Layer (type)	Output Shape	Param #
layer_conv1 (Conv2D)	(None, 128, 128, 32)	896
pooling1 (MaxPooling2D)	(None, 64, 64, 32)	0
layer_conv2 (Conv2D)	(None, 64, 64, 64)	18496
pooling2 (MaxPooling2D)	(None, 32, 32, 64)	0
GlobalPooling (GlobalMaxPooling2D)	(None, 64)	0
Flatten (Flatten)	(None, 64)	0
fully-connected1 (Dense)	(None, 256)	16640
batch_normalization_61 (BatchNormalization)	(None, 256)	1024
dropout_60 (Dropout)	(None, 256)	0
fully-connected2 (Dense)	(None, 128)	32896
batch_normalization_62 (BatchNormalization)	(None, 128)	512
dropout_61 (Dropout)	(None, 128)	0
output (Dense)	(None, 5)	645

=====
Total params: 71109 (277.77 KB)
Trainable params: 70341 (274.77 KB)
Non-trainable params: 768 (3.00 KB)
=====

Figure 28

```
learning_rate=1e-3
batch_size=256
epochs=30
|
optimizer = keras.optimizers.Adam(learning_rate=learning_rate, weight_decay=1e-5)
model_cnn.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics='accuracy')

train_model_cnn = model_cnn.fit(X_train, y_train, validation_split=0.1, epochs=epochs, batch_size=batch_size)
```

Figure 29

```
107/107 [=====] - 0s 4ms/step - loss: 1.2832 - accuracy: 0.4740
[1.283213496208191, 0.47397661209106445]
```

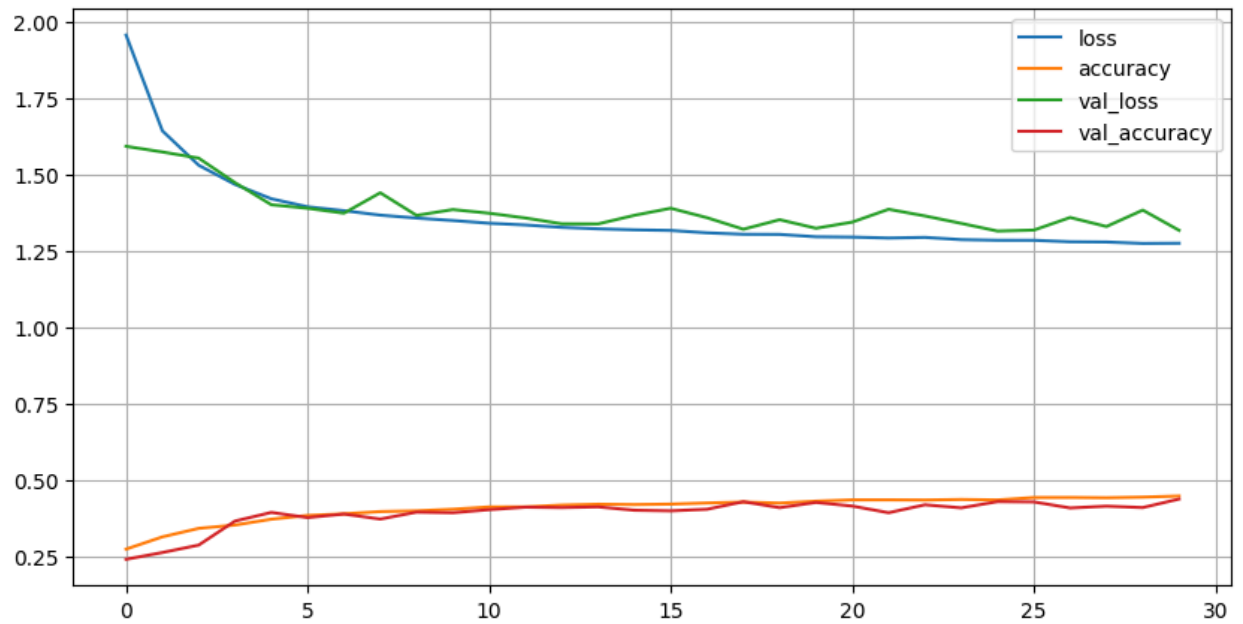


Figure 30

Original Images

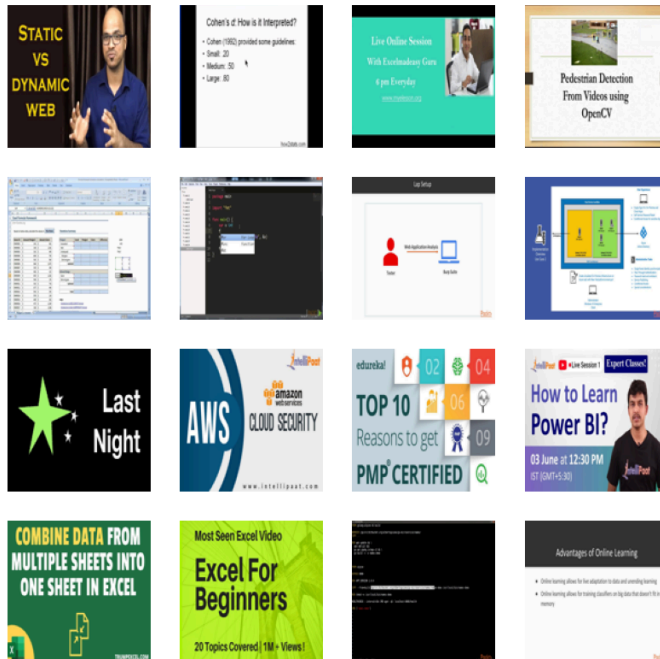


Figure 31

Augmented Images

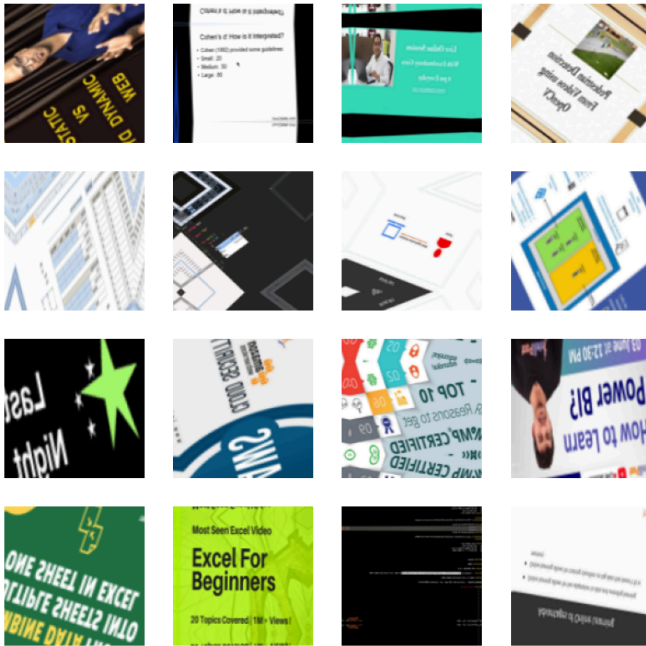


Figure 32

```
model_cnn_aug.evaluate(test_dataset)

14/14 [=====] - 0s 31ms/step - loss: 1.3086 - accuracy: 0.4406
[1.3085730075836182, 0.4406432807445526]
```

Figure 33

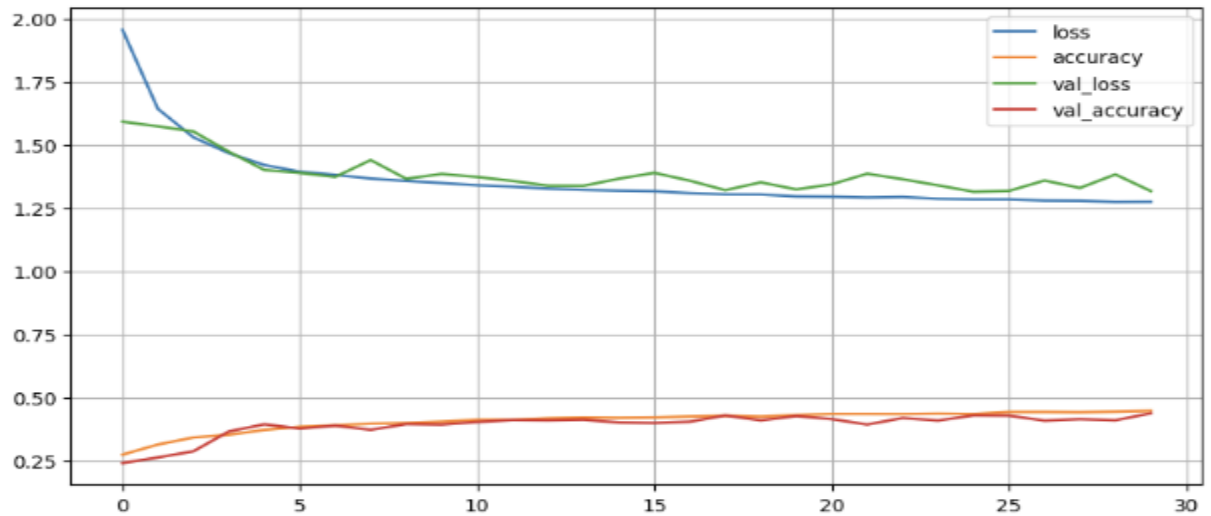


Figure 34

7. Text-based Approach

```

7      Power Query & DAX Formulas for Advanced Data M...
16     Double OR Logical Tests in SUMIFS Function usi...
18     Combine Two Excel Files Into PivotTable Report...
38     Create a Dynamic Lookup Table using INDEX & LO...
43     OR Logical Test Average Excel Formula. DAVERAG...
...
44256  KDD2016 paper 1202Title: Sampling of Attribute...
44257  KDD2016 paper 1227Title: Identifying Earmarks ...
44258  KDD2016 paper 1236Title: Online Feature Select...
44259  KDD2016 paper 958Title: Scalable Betweenness C...
44260  KDD2016 paper 799Title: Identifying Earmarks i...
Length: 34192, dtype: object
7      3
16     3
18     3
38     3
43     2
...
44256  1
44257  0
44258  1
44259  2
44260  0
Name: viewCategory, Length: 34192, dtype: category
Categories (5, int64): [0 < 1 < 2 < 3 < 4]

```

Figure 35

```
['https': 1, 'com': 2, 'www': 3, 'data': 4, 'http': 5, 'video': 6, 'facebook': 7, 'twitter': 8, 'course': 9, 'us': 10, 'learn': 11, 'excel': 12, 'youtube': 13, 'learning': 14, 'training'
```

Figure 36

```
MrExcel's Learn Excel #648 - Changing Numbers GraphicallySteve asks :  
[1136, 11, 12, 1326, 614, 2231, 1569, 6654, 3886, 150, 3490, 12, 874,
```

Figure 37

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 0, 0, 0, 1136, 11, 12, 1326,  
614, 2231, 1569, 6654, 3886, 150, 3490, 12, 874, 5331, 190,  
3886, 431, 614, 619, 6712, 328, 2957, 2162, 351, 499, 249,  
686, 249, 3886, 38, 328, 147, 69, 960, 2151, 1044, 12,  
76, 834, 541, 389, 541, 107, 650, 582, 6, 240, 718,  
76, 582, 210, 11, 12, 1052, 322, 130, 328, 85, 191,  
548, 6, 232, 748, 76, 11, 47, 1044, 582, 2131, 152,  
1052, 582, 210], dtype=int32)
```

Figure 38

Model: "sequential_35"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 520, 512)	5120000
LSTM_1 (LSTM)	(None, 520, 128)	328192
LSTM_2 (LSTM)	(None, 520, 64)	49408
LSTM_3 (LSTM)	(None, 32)	12416
FC1 (Dense)	(None, 32)	1056
batch_normalization_42 (BatchNormalization)	(None, 32)	128
dropout_41 (Dropout)	(None, 32)	0
FC2 (Dense)	(None, 16)	528
batch_normalization_43 (BatchNormalization)	(None, 16)	64
dropout_42 (Dropout)	(None, 16)	0
output (Dense)	(None, 5)	85

=====
Total params: 5511877 (21.03 MB)
Trainable params: 5511781 (21.03 MB)
Non-trainable params: 96 (384.00 Byte)
=====

Figure 39

Model: "sequential_25"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 520, 512)	5120000
gru_1 (GRU)	(None, 520, 128)	246528
gru_2 (GRU)	(None, 520, 64)	37248
gru_3 (GRU)	(None, 32)	9408
FC1 (Dense)	(None, 32)	1056
batch_normalization_44 (Batch Normalization)	(None, 32)	128
dropout_44 (Dropout)	(None, 32)	0
FC2 (Dense)	(None, 16)	528
batch_normalization_45 (Batch Normalization)	(None, 16)	64
dropout_45 (Dropout)	(None, 16)	0
output (Dense)	(None, 5)	85

=====
Total params: 5415045 (20.66 MB)
Trainable params: 5414949 (20.66 MB)
Non-trainable params: 96 (384.00 Byte)
=====

Figure 40

```
107/107 [=====] - 2s 21ms/step - loss: 1.2698 - accuracy: 0.5184
[1.2697726488113403, 0.5184210538864136]
```

Figure 41

```
107/107 [=====] - 2s 20ms/step - loss: 1.3337 - accuracy: 0.5415
[1.3337229490280151, 0.5415204763412476]
```

Figure 42

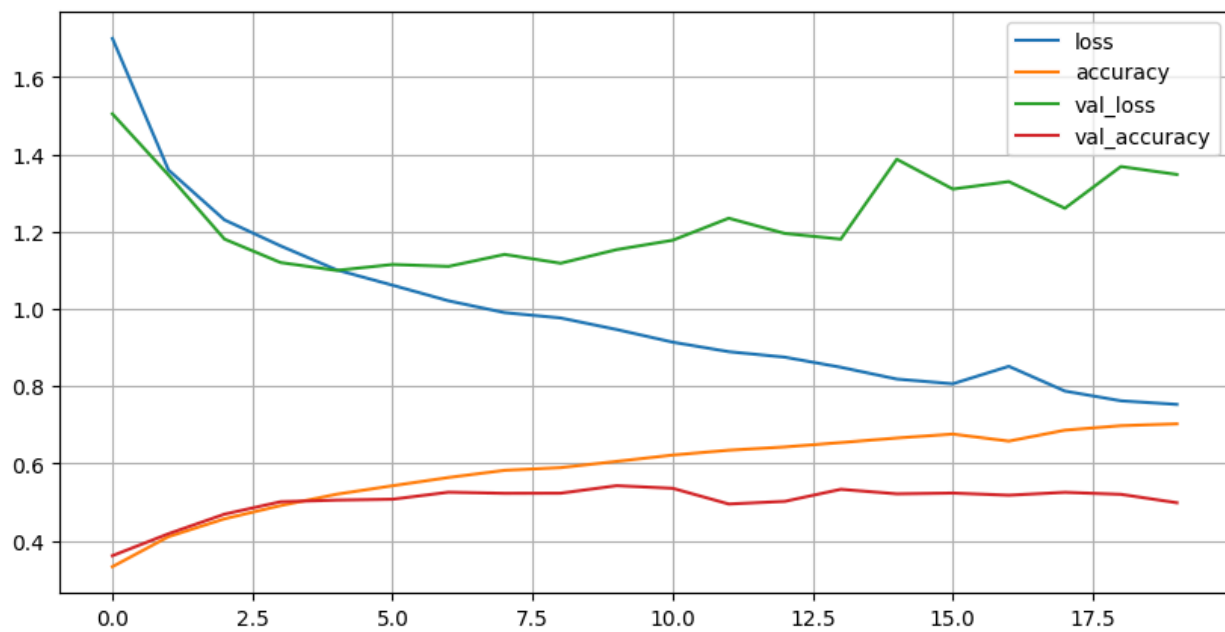


Figure 43

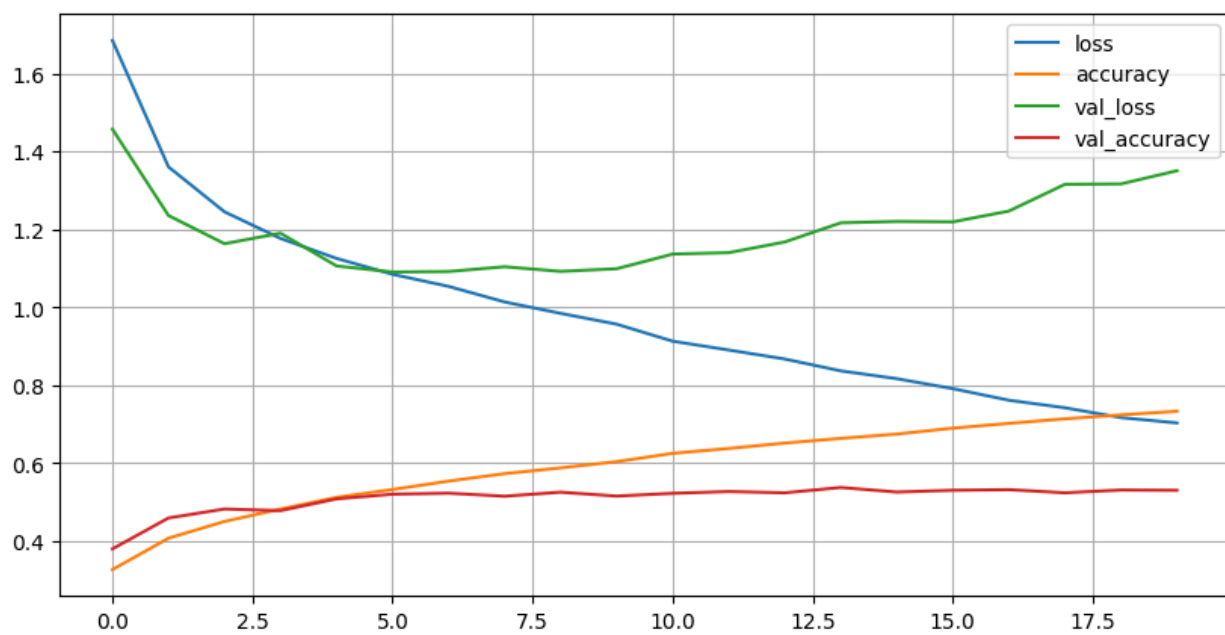


Figure 44

Model: "sequential_26"		
Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 520, 512)	5120000
bidirectional_20 (Bidirectional)	(None, 520, 256)	493056
bidirectional_21 (Bidirectional)	(None, 520, 128)	123648
bidirectional_22 (Bidirectional)	(None, 64)	31104
FC1 (Dense)	(None, 64)	4160
batch_normalization_46 (Batch Normalization)	(None, 64)	256
dropout_46 (Dropout)	(None, 64)	0
FC2 (Dense)	(None, 32)	2080
batch_normalization_47 (Batch Normalization)	(None, 32)	128
dropout_47 (Dropout)	(None, 32)	0
FC3 (Dense)	(None, 16)	528
batch_normalization_48 (Batch Normalization)	(None, 16)	64
dropout_48 (Dropout)	(None, 16)	0
output (Dense)	(None, 5)	85
Total params: 5775109 (22.03 MB)		
Trainable params: 5774885 (22.03 MB)		
Non-trainable params: 224 (896.00 Byte)		

Figure 45

Model: "sequential_36"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 520, 512)	5120000
bidirectional (Bidirectional)	(None, 520, 256)	493056
tf_attention_model_24 (TFAAttentionModel)	(None, 520, 256)	1051904
bidirectional_1 (Bidirectional)	(None, 520, 128)	123648
bidirectional_2 (Bidirectional)	(None, 64)	31104
FC1 (Dense)	(None, 64)	4160
batch_normalization_44 (BatchNormalization)	(None, 64)	256
dropout_43 (Dropout)	(None, 64)	0
FC2 (Dense)	(None, 32)	2080
batch_normalization_45 (BatchNormalization)	(None, 32)	128
dropout_44 (Dropout)	(None, 32)	0
FC3 (Dense)	(None, 16)	528
batch_normalization_46 (BatchNormalization)	(None, 16)	64
dropout_45 (Dropout)	(None, 16)	0
output (Dense)	(None, 5)	85

Total params: 6827013 (26.04 MB)
Trainable params: 6826789 (26.04 MB)
Non-trainable params: 224 (896.00 Byte)

Figure 46

```
107/107 [=====] - 4s 40ms/step - loss: 1.2241 - accuracy: 0.5509
[1.2241483926773071, 0.5508772134780884]
```

Figure 47

```
107/107 [=====] - 5s 46ms/step - loss: 1.3373 - accuracy: 0.5465  
[1.33726167678833, 0.5464912056922913]
```

Figure 48

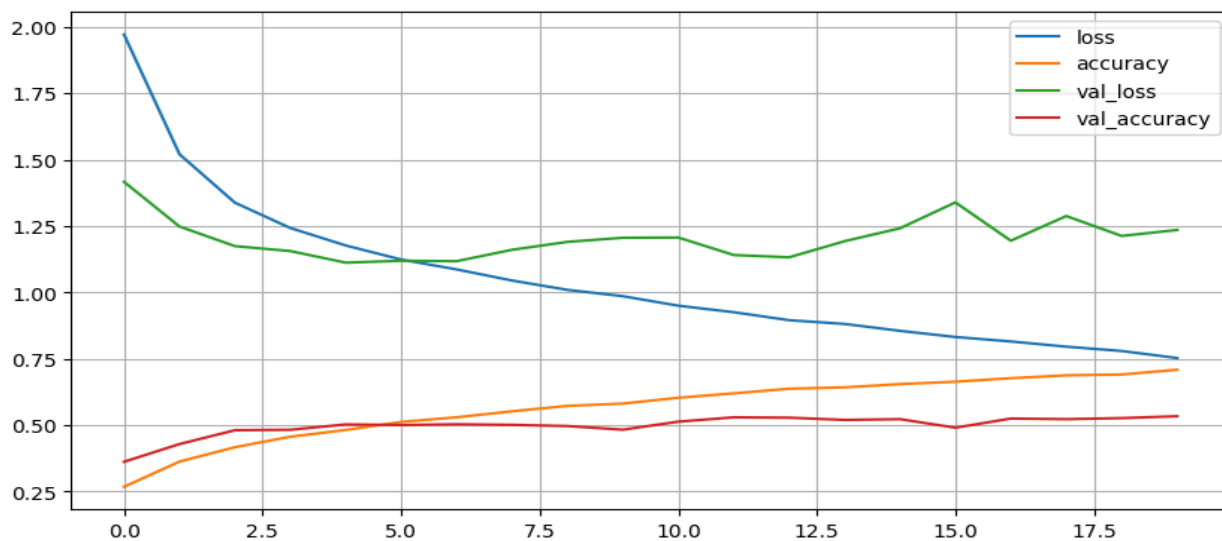


Figure 49

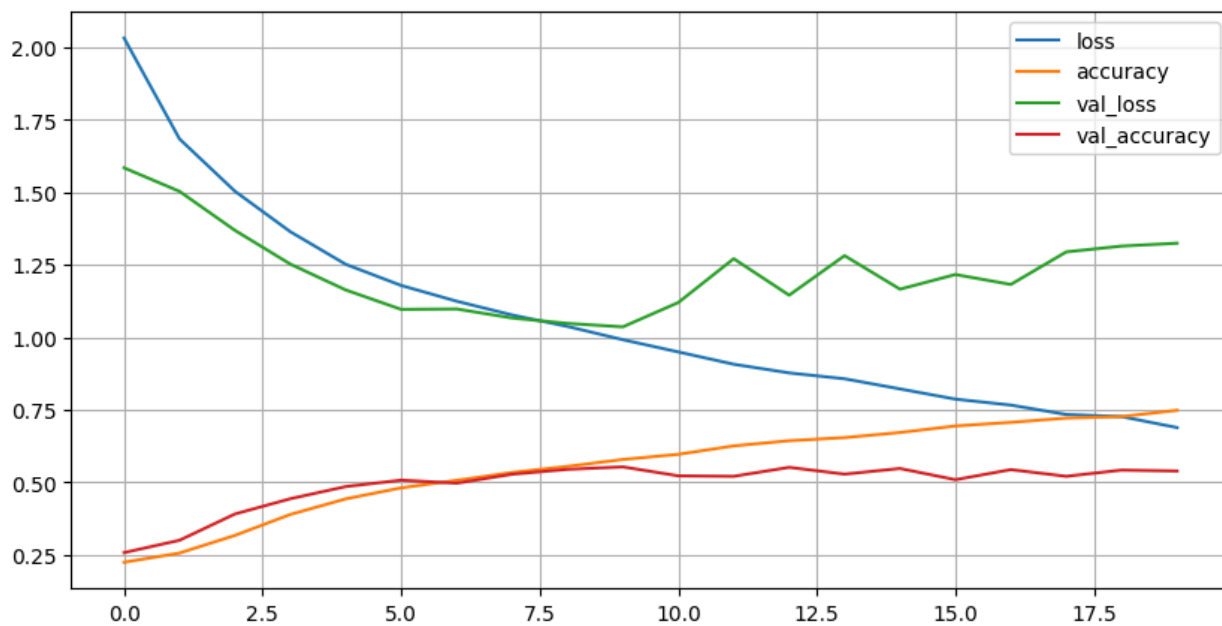


Figure 50

8: conclusion

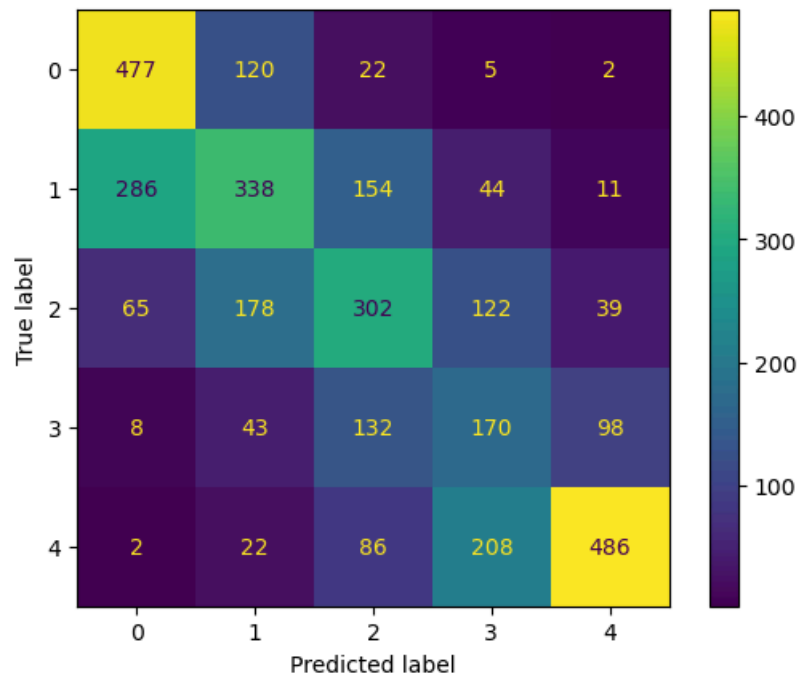


Figure 51

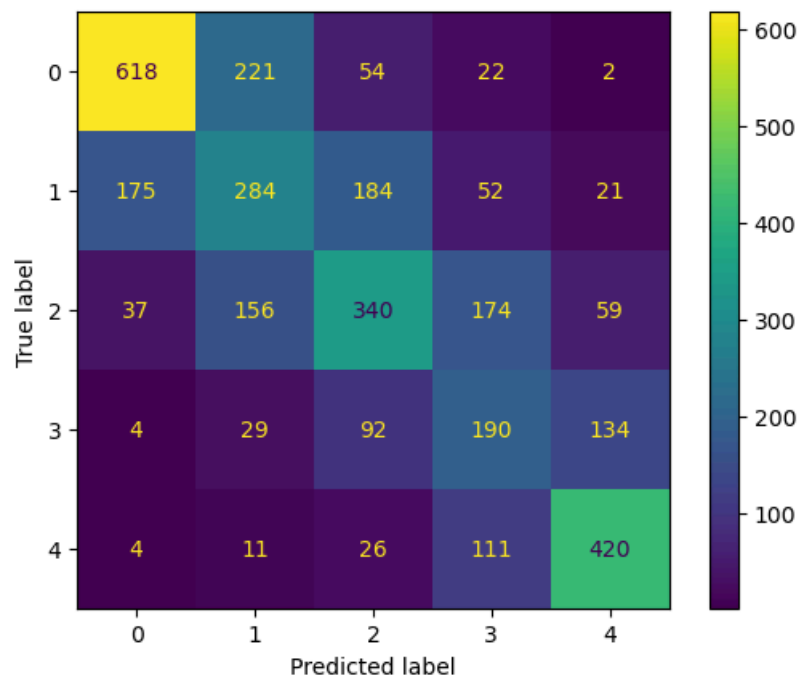


Figure 52

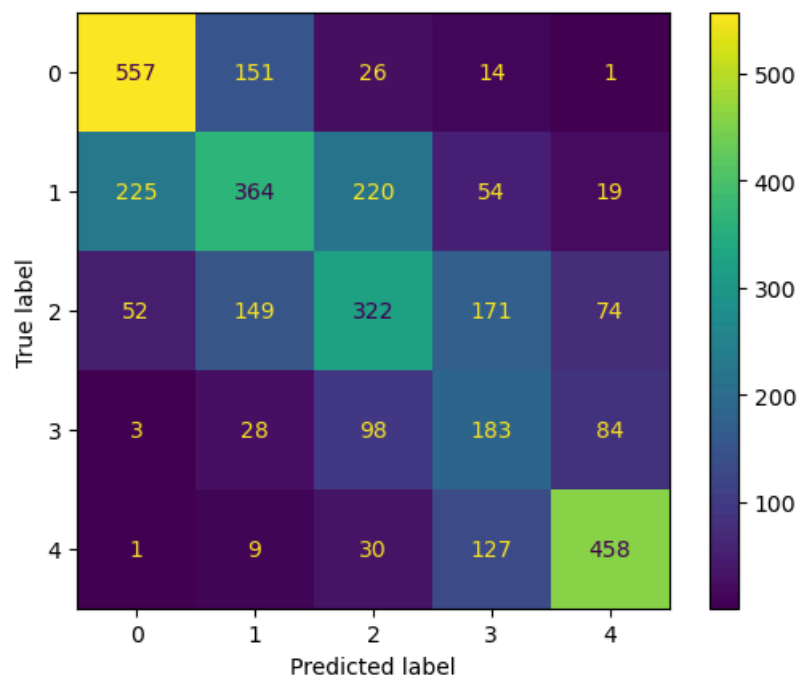


Figure 53

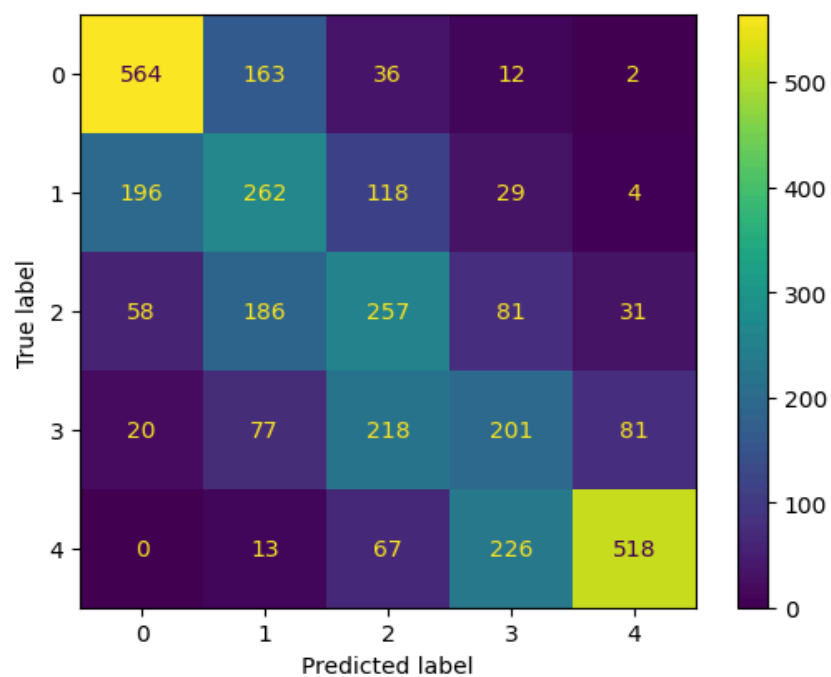


Figure 54

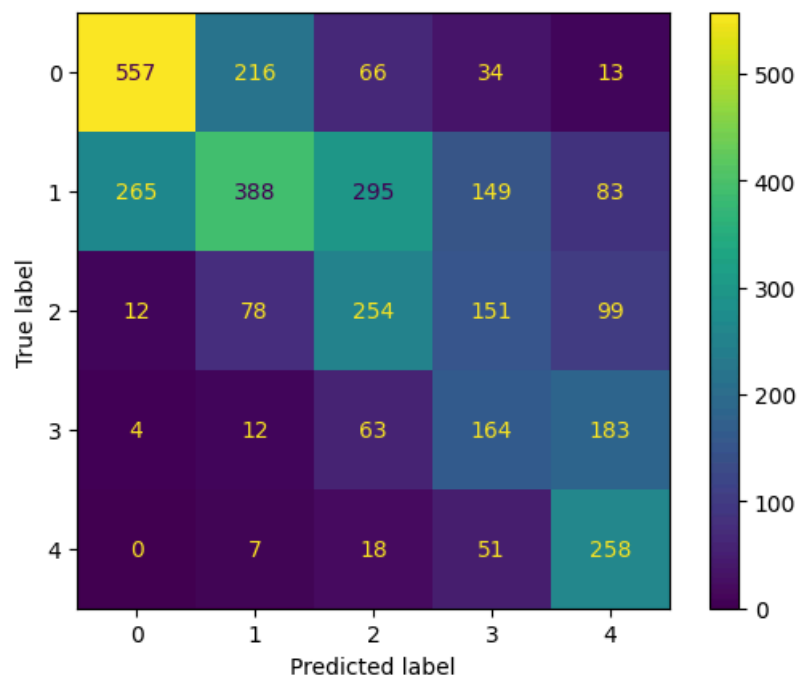


Figure 55

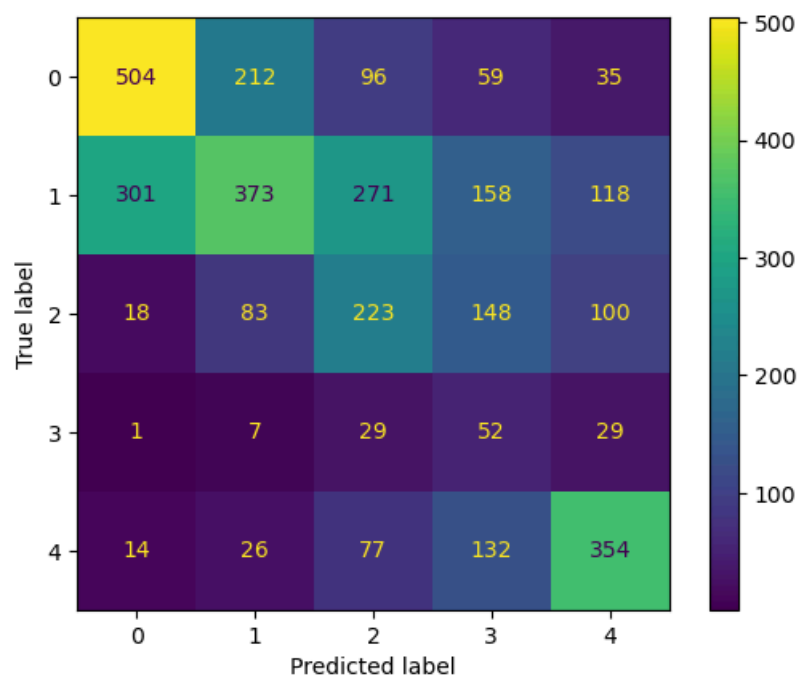


Figure 56