Google 3D: YouTube Viral Video Forecasting

AI Studio Final Presentation

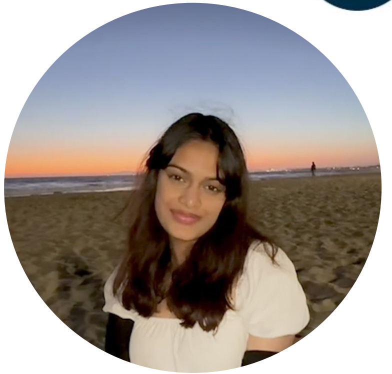
Break Through Tech Virtual Program @ Cornell Tech
12/06/2024

Introductions

Meet Our Team!

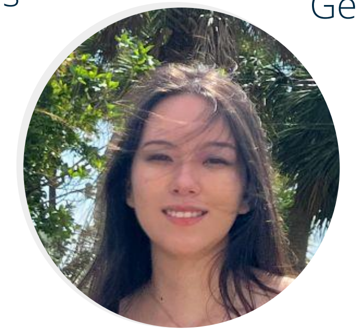
Heta Patel
Stevens Institute of Technology

Krithika Subramanian
University of Texas at Austin



Rishita Dhalbisoi
Georgia Institute of Technology

Lauren Hu
Rice University



Veronica Zhao
New York University

Our AI Studio TA and Challenge Advisors

Helenna Yin
AI Studio TA

Nasser Qadri
Challenge Advisor

AI Studio Project Overview

Build a machine learning model to predict which YouTube videos are likely to become viral or trending. The model should consider early engagement metrics, video metadata, and potentially external factors like news events or social media trends

Business Impact

- **User Experience**: Providing users with trending or likely-to-be-viral content enhances their experience, making them more likely to return to the platform. This improves user retention and satisfaction.
- **Advertising Revenue**: Viral videos attract a large number of viewers in a short period, leading to higher engagement and more ad impressions. Predicting which videos will become viral allows Google to optimize ad placements and maximize revenue.
- **Content Promotion**: By identifying potential viral content early, Google can promote these videos through recommendations, trending lists, and search results. This increases user engagement on platforms like YouTube, keeping viewers on the site longer.

Our Approach & Key Findings

- We performed exploratory data analysis and found that the nature of many trending videos depends upon when it was *previously* trending
    - A sequential aspect of the dataset was introduced
- We tested many models, some did not take into account related videos, others did
    - Overall results show that the arbitrary models and our neural network work very well, but we can still tune our time series and LSTM neural network to improve their accuracies

Data Understanding & Data Preparation

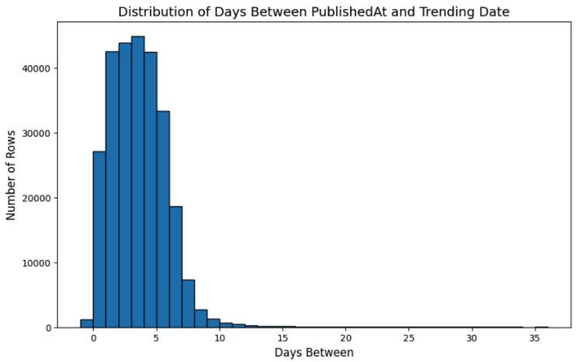
Data Overview and Preparation

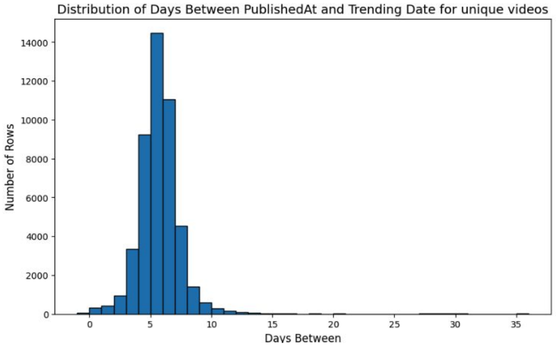
- Data Set: https://www.kaggle.com/datasets/rsrishav/youtube-trending-video-dataset
    - Data set of numerical data, strings, and time stamps
    - Data set includes with 47,142 entries and 16 columns
    - Stored in csv file on Google Drive
- Data preprocessing steps that we took are data cleaning and ensuring that any missing information and outliers are properly managed. We also feature engineered to ensure that the features we use are optimal for our model rather than having possible redundancy or irrelevant data.

Exploratory Data Analysis

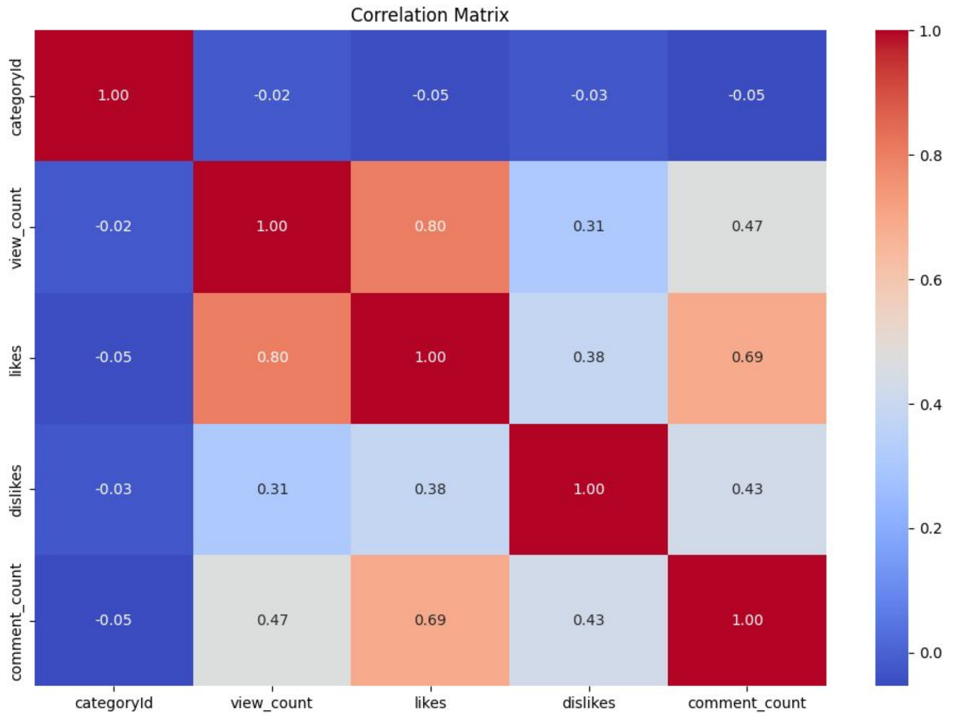
- Unique videos:
  - The same video would appear in the data multiple times, at different trending dates
  - Leveraged this by making a new dataframe called DfUnique to consider the dataframe with only unique videos with the highest view count kept — used for EDA and data familiarization
  - Leveraged the fact that the same video was in the dataset multiple times, allowed to make future predicts and remodel research question
- Time
  - Given publishedAt and trendingDate fields in the data, able to use this to see the time of onset of virality
  - Utilized pandas to_datetime to turn these fields into proper dates we could work with– useful for model data splitting, analysis, etc.
  - Split into month and year, seasons for timeframes

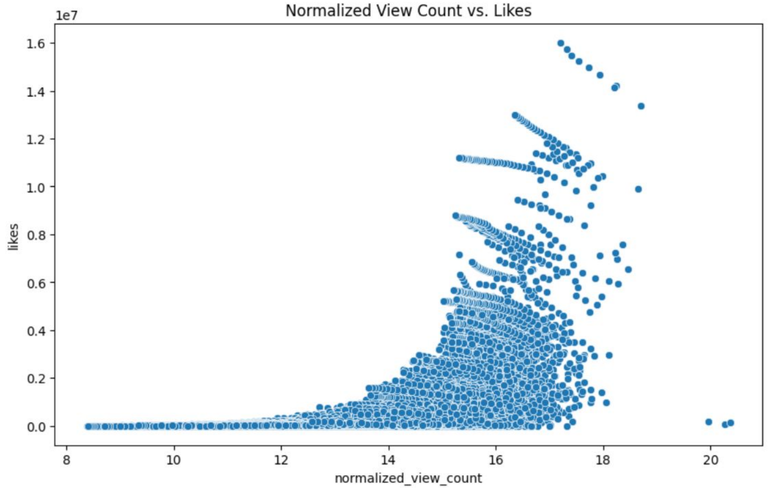
```
idx = df.groupby('video_id')['view_count'].idxmax()
dfUnique = df.loc[idx].reset_index(drop=True)
```



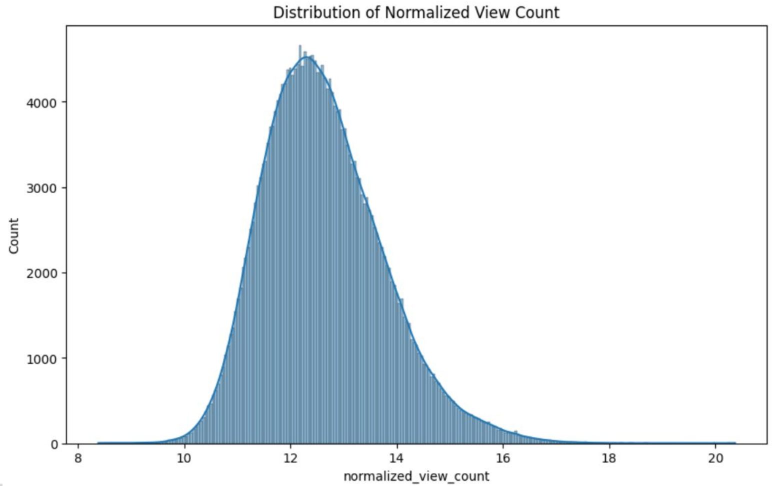


Exploratory Data Analysis





Where the normalized view count is the log of the view count divided by the days since published.



Exploratory Data Analysis

To define a video as “viral”, we will use a target variable of normalized view count.

Based on the exploratory data analysis, the correlation matrix, and our own intuition, we expect a few features to be strongly correlated with this target variable:

- Likes and comments
- The *growth rate* of likes and comments
- If the video was published on a weekend
- What season the video was published in

Data Preprocessing

- Tags:
  - Tags are essentially keywords associated with a video, which was leveraged to identify the impact of sociopolitical factors and current media trends
  - Converted into a format for easier analysis
  - Generated tags for videos without them using NLP keyword extraction on titles using Spacy library
- Feature Set:
  - Used Pearson correlation and Spearman correlation to help aid determine which features should be in the feature set
  - Engineered new features, such as like and comment ratios,, to identify and eradicate outliers and measure engagement
  - Dropped if correlation of feature was < 0.1

```python
## define a function for keyword extraction on the text columns of our dataset
def get_keywords(input_text):
    top_phrases = [] ## default empty for nan
    if isinstance(input_text, str): ## only run if non-nan since sub will fail
        keyworded = spacy_nlp(input_text)
        top_phrases = [phrase.text for phrase in keyworded._.phrases[:10]]
    return top_phrases
```

```python
## above funciton took way too long to run, trying to optimize by leveraging spacy's batch processing
## to process multiple rows
# Disable unnecessary components for faster processing
#spacy_nlp = spacy.load("en_core_web_sm", disable=['ner', 'parser', 'tagger'])

def get_keywords_batch(texts):
    docs = list(spacy_nlp.pipe(texts, batch_size=32))
    keywords_list = []
    for doc in docs:
        keywords = [phrase.text for phrase in doc._.phrases[:10]]
        keywords_list.append(keywords)
    return keywords_list
```

Pearson Correlation:

	like_ratio	comment_ratio	comment_count	likes	view_count
like_ratio	1.000000	0.397553	0.075805	0.210867	-0.021805
comment_ratio	0.397553	1.000000	0.238413	0.102149	-0.019629
comment_count	0.075805	0.238413	1.000000	0.685399	0.402247
likes	0.210867	0.102149	0.685399	1.000000	0.627792
view_count	-0.021805	-0.019629	0.402247	0.627792	1.000000

Spearman Correlation:

	like_ratio	comment_ratio	comment_count	likes	view_count
like_ratio	1.000000	0.475750	0.312076	0.557670	-0.020350
comment_ratio	0.475750	1.000000	0.519042	0.095660	-0.226291
comment_count	0.312076	0.519042	1.000000	0.727022	0.651364
likes	0.557670	0.095660	0.727022	1.000000	0.771007
view_count	-0.020350	-0.226291	0.651364	0.771007	1.000000

Data Preprocessing

- Categories
  - One-hot encoded video category
- Channels:
  - Wanted to identify channels that were commonly trending or averaged high view counts
- Google Trends
  - pytrends
    - Essentially wanted to pass in the top relevant keywords from the tags list, given the timeframe of the same month and the google property filtered to Youtube
    - Cached these results
  - Google Cloud
    - Using BigQuery API to write SQL commands into the top_trends schema in Google's trends database

```python
def get_trends_for_video_bq(tags, start_date, end_date, dma=None):
    # Construct the SQL query with the tags and date range
    tag_list = ', '.join([f"'{tag}'" for tag in tags[:5]])  # Limit to 5 tags
    query = f"""
        SELECT
            week,
            term,
            score,
            dma_name,
            dma_id
        FROM
            `bigquery-public-data.google_trends.top_terms`
        WHERE
            term IN ({tag_list})
            AND DATE(week) BETWEEN '{start_date}' AND '{end_date}'
    """

    # Add DMA filter if specified
    if dma:
        query += f" AND dma_name = '{dma}'"

    query += " ORDER BY score DESC"

    # Run the query and convert the result to a DataFrame
    query_job = client.query(query)
    results_df = query_job.to_dataframe()

    return results_df
```

	week	term	score
0	2020-08-02	Warriors	79
1	2020-08-16	Warriors	46
2	2020-08-16	Warriors	35
3	2020-08-16	Warriors	33
4	2020-08-16	Warriors	33
...	...	...	...
6295	2020-08-23	Warriors	<NA>
6296	2020-08-02	Warriors	<NA>
6297	2020-08-09	Warriors	<NA>
6298	2020-08-23	Warriors	<NA>
6299	2020-08-30	Warriors	<NA>

Modeling & Evaluation

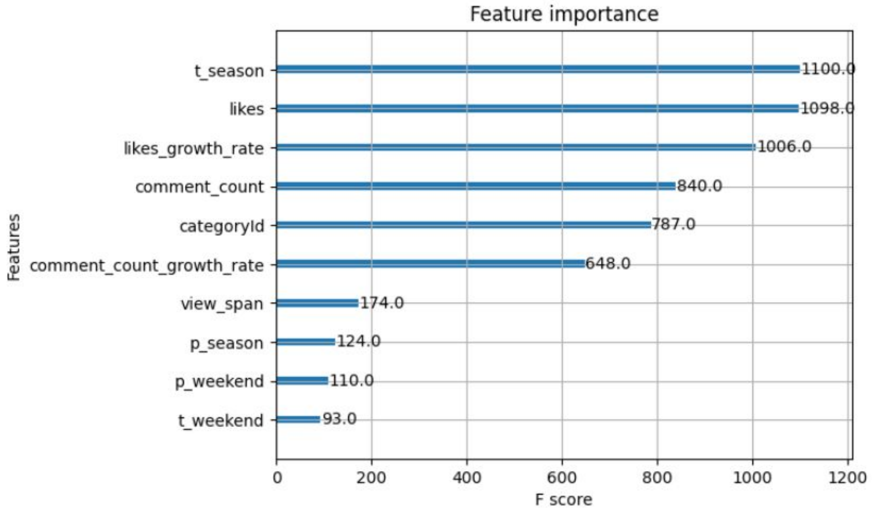
Models Considered

Model Name	**Hypotheses**
Random Forest	Chosen for its ability to handle nonlinear relationships and robustness to overfitting through ensemble learning. Struggles with extrapolating beyond range of training data.
Gradient Boosting Regressor	Can iteratively learn from residual errors, effective for capturing nuances in features like likes_growth_rate and comment_count_growth_rate.
Time Series	Very good for taking advantage of temporal features like trending dates, published dates, and chronological tracking over time.
Neural Networks	Attempt to model complex and intricate relationships across given features and engineered features, but has a tendency to lack interpretability.

Model Training:
Random Forest and Gradient Boosting

- Feature Engineering
  - Features for Normal Models: ['categoryId', 'likes', 'comment_count', "view_span","p_season","p_weekend", "t_season","t_weekend", 'likes_growth_rate', 'comment_count_growth_rate']
    - Attempt to take into account previous trending videos by calculating the growth rates of likes, dislikes, comment count, etc.
    - These values were then used to predict the normalized view count to see how many views the video is predicted to get
    - Used feature importance libraries to understand which features is best for which model (ex. scikit-learn feature_importances_)
- Hyperparameter Tuning
  - Random Forest Model: Manipulation of n_estimators, max_features, and max_depth. Baseline model with minimal transformation.
  - Gradient Boosting Model: Manipulation of depth and n_estimators to get best value (Grid Search CV)

Model Training:
Gradient Boosting Regressor Model



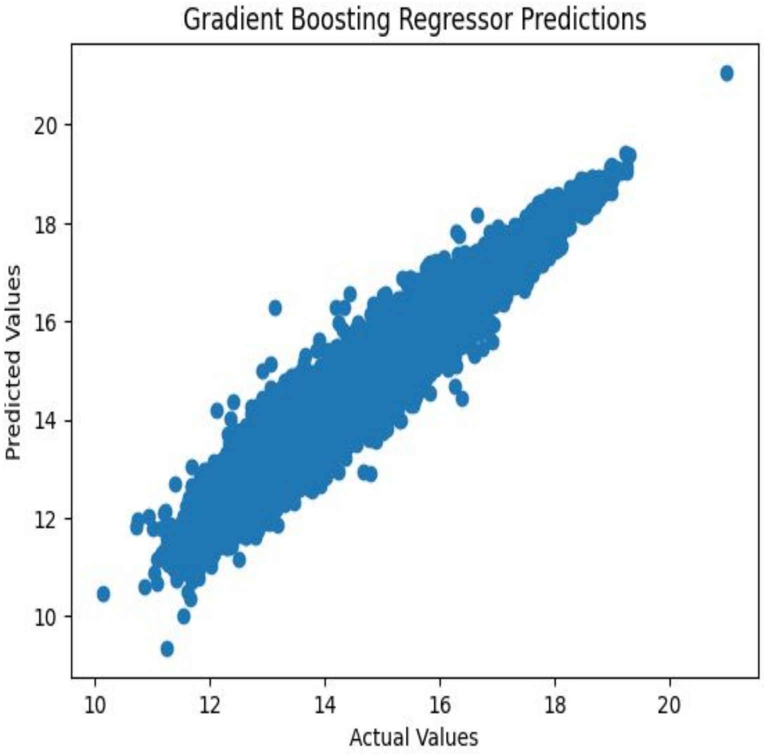
Feature Importance XGBoost for GBR Model

Key Takeaways:
Trending videos are a result of many external factors and therefore the season, likes, comment count, etc. all affect a video going trending rather than a user searching for a video themselves. For example, the season at which a video is produced ultimately affects the trend rate as well (in the summer, videos are more likely to trend due to larger viewership).

Model Evaluation:
Gradient Boosting Regressor Model

```
y_GBDT_pred = gbdt_model.predict(X_test)
# 2. Compute the RMSE
gbdt_rmse = mean_squared_error(y_test, y_GBDT_pred)
# 3. Compute the R2 score
gbdt_r2 = r2_score(y_test, y_GBDT_pred)
print('[GBDT] Root Mean Squared Error: {0}'.format(gbdt_rmse))
print('[GBDT] R2: {0}'.format(gbdt_r2))
```

```
[GBDT] Root Mean Squared Error: 0.08654072803269446
[GBDT] R2: 0.9341704744573249
```



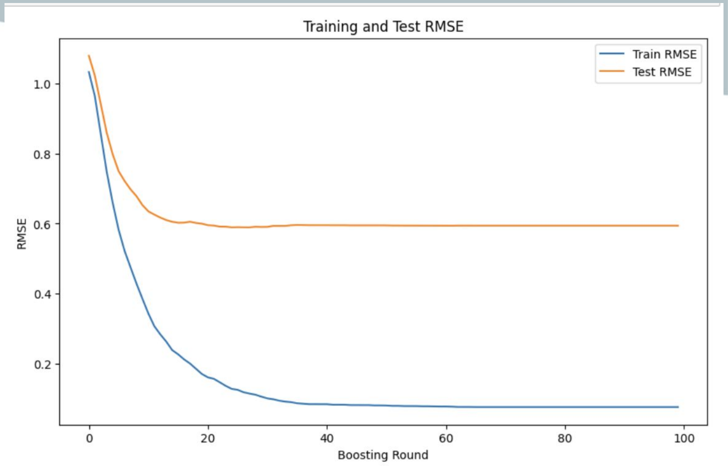
Model Evaluation:
Time Series

- Feature Engineering
  - X Columns
    - Hot-encoded all the categories
    - Hot-encoded the seasons and years
    - Likes, Comment counts, Tag counts, and channel counts on leaderboard
    - View count on Day 1
  - Target Variable
    - Log of Highest view count (dataframe condensed to only unique video IDs)
  - Standardized the variables to ensure balanced scale for data
    - Used StandardScaler()
  - Transformation of dataframes
    - Used Dmatrix, an internal data structure used by XGBoost

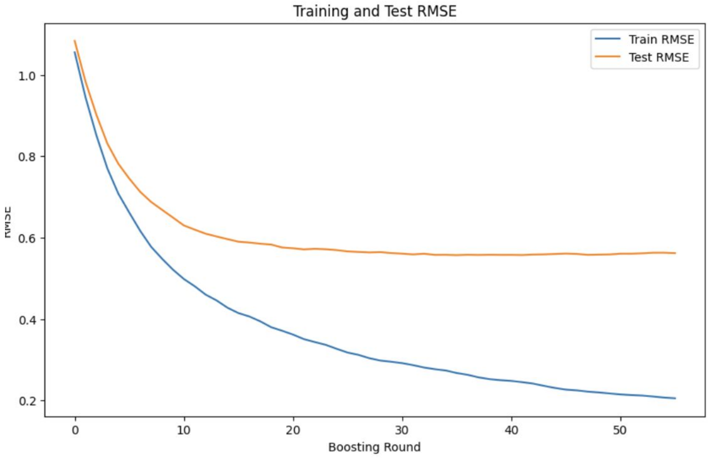
Model Evaluation:
Time Series

- Overfitting
  - The main problem was overfitting:
    - Training MSE = 0.049, Testing MSE = 0.31
    - Graphs to visualize
  - Solution 1:
    - Hyperparameter tuning
    - Used GridSearchCV on parameters
    - Specifically paid attention to:
      - L1 and L2 regularization
      - Max depth for model complexity
      - Learning Rate for convergence

```
[91]    train-rmse:0.07589      test-rmse:0.59440
[92]    train-rmse:0.07589      test-rmse:0.59440
[93]    train-rmse:0.07589      test-rmse:0.59440
[94]    train-rmse:0.07589      test-rmse:0.59440
[95]    train-rmse:0.07589      test-rmse:0.59440
[96]    train-rmse:0.07589      test-rmse:0.59440
[97]    train-rmse:0.07589      test-rmse:0.59440
[98]    train-rmse:0.07589      test-rmse:0.59440
[99]    train-rmse:0.07589      test-rmse:0.59440
```




```
params = {
        'min_child_weight': [1, 5, 10, 20],
        'gamma': [0.5, 1, 1.5, 2, 5, 7.5, 10],
        'subsample': [0.5, 0.6, 0.8, 1.0],
        'colsample_bytree': [0.5, 0.6, 0.8, 1.0],
        'max_depth': [3, 4, 5, 8, 10, 12],
        'learning_rate':[0.01, 0.02, 0.05, 0.1, 0.15, 0.2, 0.25]
        }
```

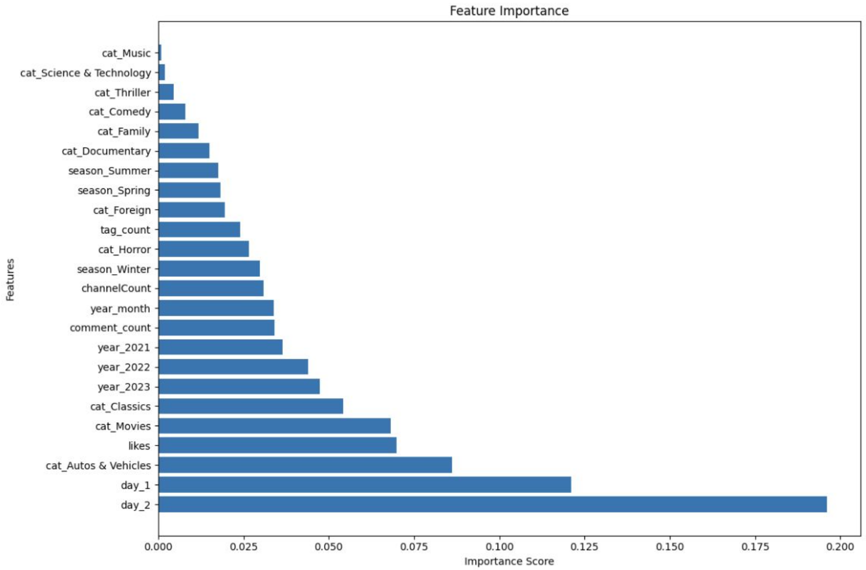


Model Evaluation:
Time Series

- Overfitting
  - K-Fold cross Validation
    - Uses different portions of dataset for training and testing, making it generalizable to unseen data
  - Feature Importance
    - Visualized Important Features
    - Dropped irrelevant features
  - Got testing MSE down to 0.1

```
In [297]: from sklearn.model_selection import KFold, cross_val_score
model = XGBRegressor(
    objective='reg:squarederror',
    eval_metric='rmse',
    learning_rate=0.15,
    max_depth=10,
    reg_alpha=2,
    reg_lambda=10,
    gamma=0.5,                  # Fixed parameter
    subsample=0.8,              # Fixed parameter
    colsample_bytree=0.8,       # Fixed parameter
    min_child_weight=5
)
kf = KFold(n_splits=5, shuffle=True, random_state=1234)
cv_scores = cross_val_score(model, X_scaled, y2, cv=kf, scoring='neg_root_mean_squared_e

# Print Cross-Validation RMSE
print("Cross-Validation MSE:", -cv_scores**2)
print("Mean CV MSE:", np.mean(cv_scores)**2)
```

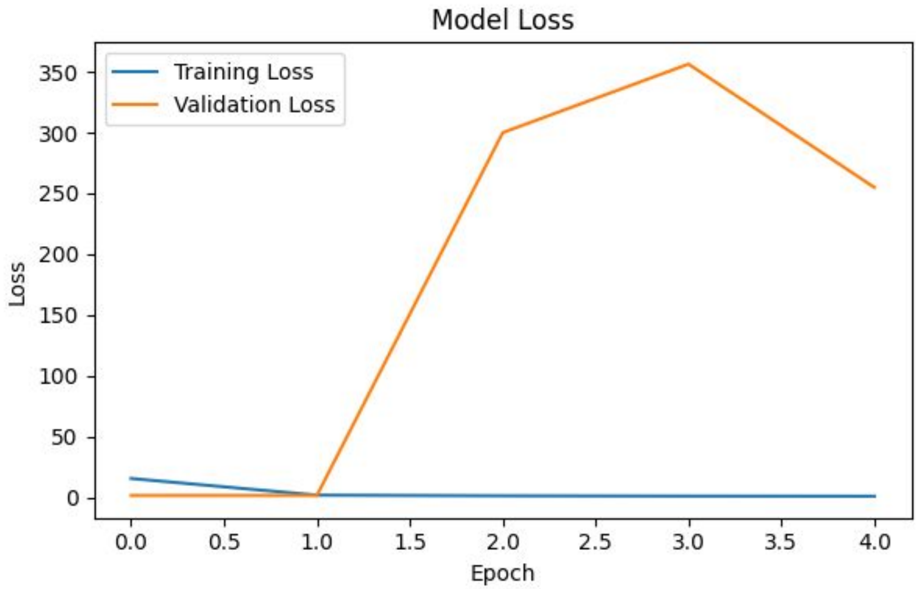


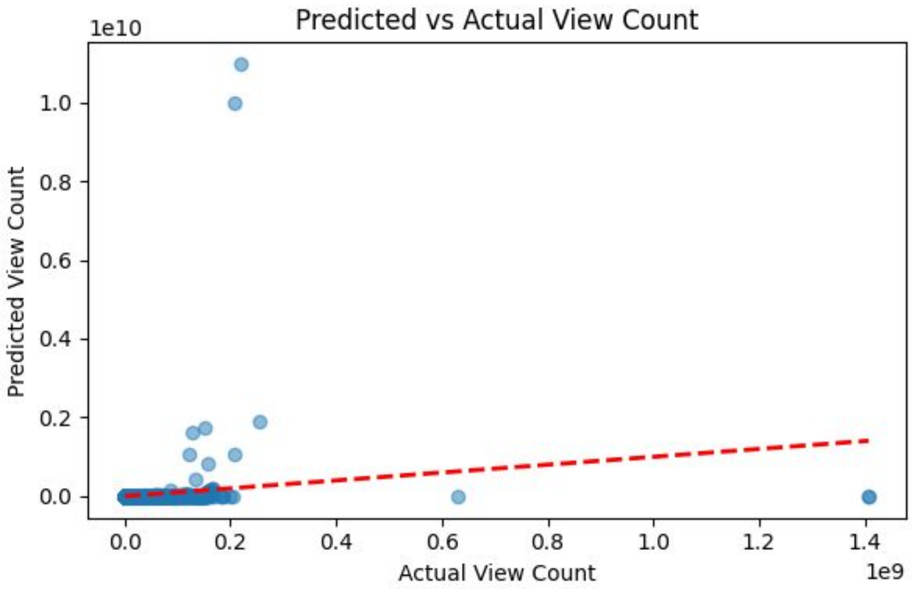
Model Training:
Neural Network

- Feature Engineering
  - Features: ['days_since_published', 'comment_count', 'likes', 'title_length', 'tags_count', 'description_length', 'publishedAt_dayofweek', 'publishedAt_hour', 'categoryId', 'days_tags_interaction', 'likes_growth_rate', 'comment_count_growth_rate', 'publishedAt_year', 'publishedAt_month', 'trendingDate_year', 'trendingDate_month']
    - Attempt to take into account previous trending videos by calculating the growth rates of likes, dislikes, comment count, etc.
- Hyperparameter Tuning
  - Neural Network: Manipulation of num_epochs, batch_size, optimizer learning_step_size, and number of layers in neural network + layer customization (number of nodes, activation, dropout, etc.)
- Model Architecture
  - For a neural network, feed features into input layer of and use Keras Sequential NN model to predict view counts

Neural Network - First Trial

Baseline Neural Network Model - Fully Connected Neural Network





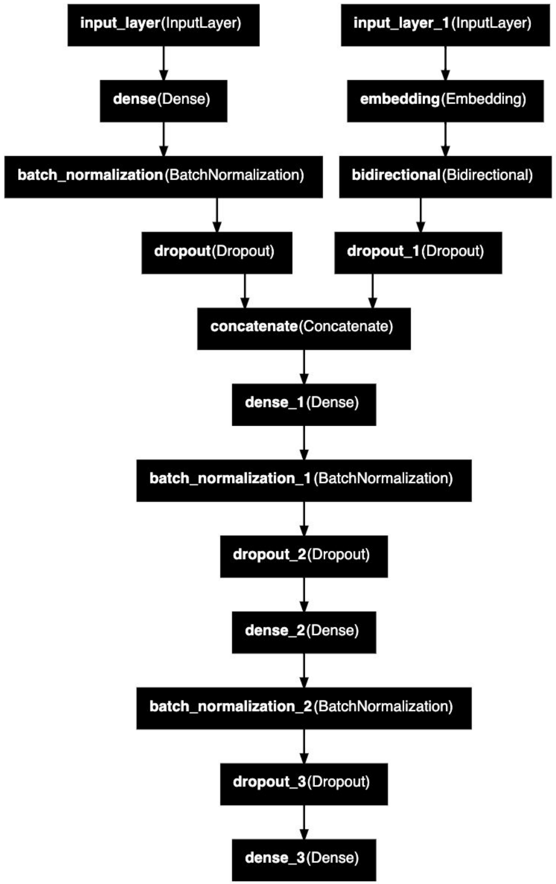
Observations and Issues: The model might be starting to overfit the training data, especially for epoch 3, 4, and 5 we can see that the validation loss is increasing while the training loss is decreasing. There could be issues with the validation data, like outliers or mislabels, that are causing the validation loss to spike. Or, the learning rate might be too high, causing the model to overshoot optimal weights and causing large fluctuations in the loss.

Neural Network - Second Trial
Model Architecture

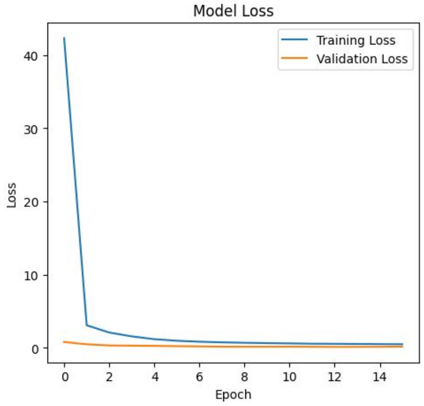
Model: "functional"

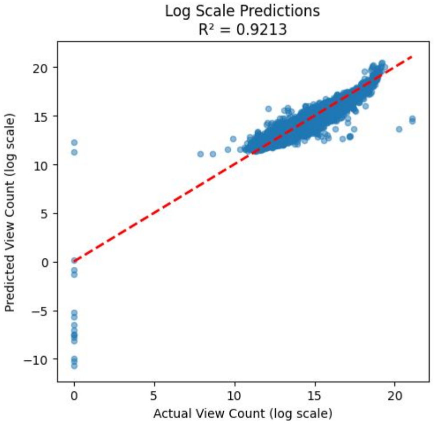
Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 24)	0	-
input_layer_1 (InputLayer)	(None, 50)	0	-
dense (Dense)	(None, 256)	6,400	input_layer[0][0]
embedding (Embedding)	(None, 50, 128)	1,280,000	input_layer_1[0][0]
batch_normalization (BatchNormalization)	(None, 256)	1,024	dense[0][0]
bidirectional (Bidirectional)	(None, 128)	98,816	embedding[0][0]
dropout (Dropout)	(None, 256)	0	batch_normalization[0…
dropout_1 (Dropout)	(None, 128)	0	bidirectional[0][0]
concatenate (Concatenate)	(None, 384)	0	dropout[0][0], dropout_1[0][0]
dense_1 (Dense)	(None, 256)	98,560	concatenate[0][0]
batch_normalization_1 (BatchNormalization)	(None, 256)	1,024	dense_1[0][0]
dropout_2 (Dropout)	(None, 256)	0	batch_normalization_1…
dense_2 (Dense)	(None, 128)	32,896	dropout_2[0][0]
batch_normalization_2 (BatchNormalization)	(None, 128)	512	dense_2[0][0]
dropout_3 (Dropout)	(None, 128)	0	batch_normalization_2…
dense_3 (Dense)	(None, 1)	129	dropout_3[0][0]

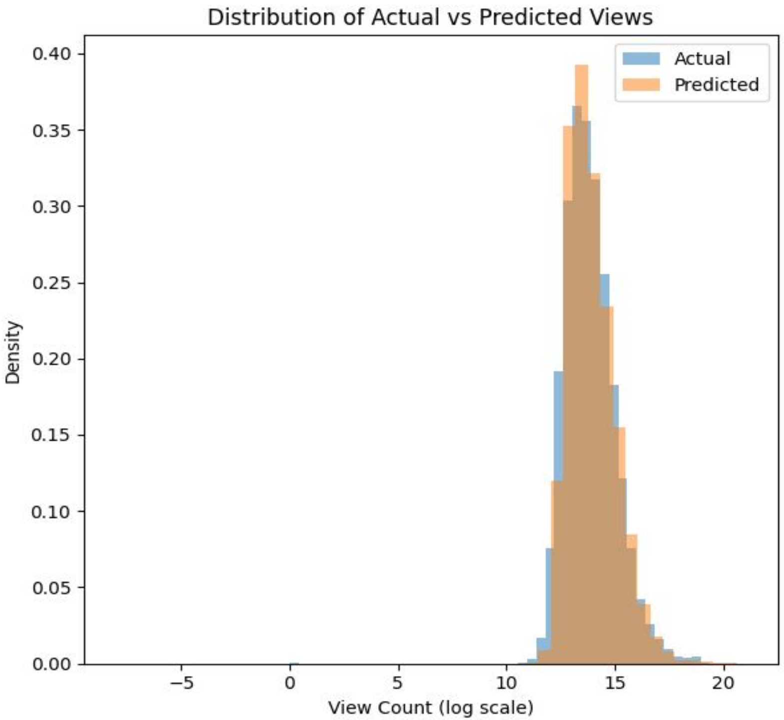
Total params: 4,555,525 (17.38 MB)
Trainable params: 1,518,081 (5.79 MB)
Non-trainable params: 1,280 (5.00 KB)
Optimizer params: 3,036,164 (11.58 MB)

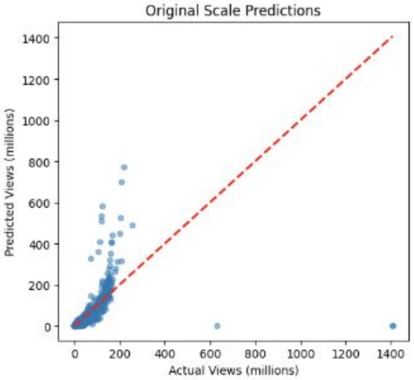


Neural Network - Second Trial
Model Performance










```
Final Model Performance:
MSE: 0.1032
R2: 0.921
```

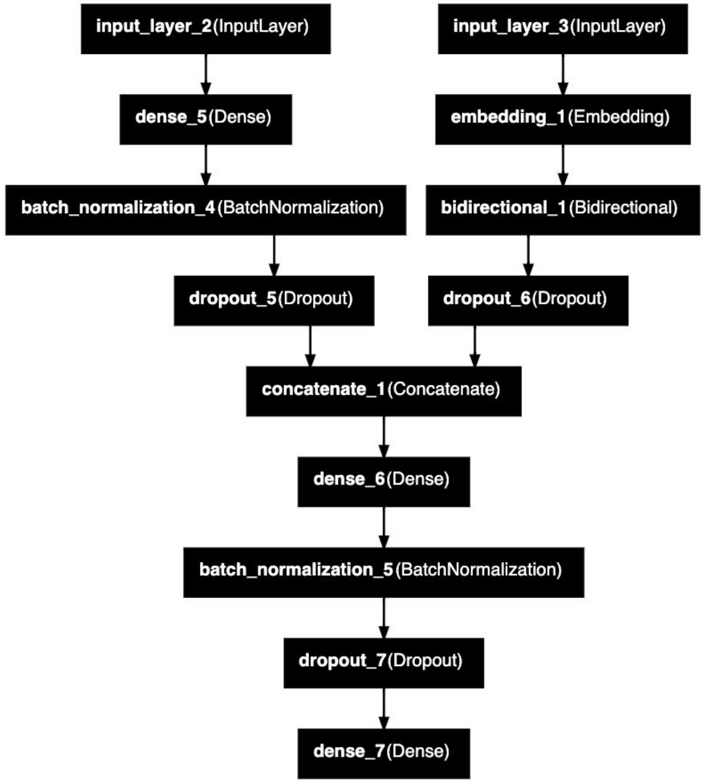
Neural Network - **Final Model**:

Model Architecture

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, 24)	0	-
input_layer_3 (InputLayer)	(None, 50)	0	-
dense_5 (Dense)	(None, 320)	8,000	input_layer_2[0][0]
embedding_1 (Embedding)	(None, 50, 160)	1,600,000	input_layer_3[0][0]
batch_normalization_4 (BatchNormalization)	(None, 320)	1,280	dense_5[0][0]
bidirectional_1 (Bidirectional)	(None, 192)	197,376	embedding_1[0][0]
dropout_5 (Dropout)	(None, 320)	0	batch_normalization_4…
dropout_6 (Dropout)	(None, 192)	0	bidirectional_1[0][0]
concatenate_1 (Concatenate)	(None, 512)	0	dropout_5[0][0], dropout_6[0][0]
dense_6 (Dense)	(None, 160)	82,080	concatenate_1[0][0]
batch_normalization_5 (BatchNormalization)	(None, 160)	640	dense_6[0][0]
dropout_7 (Dropout)	(None, 160)	0	batch_normalization_5…
dense_7 (Dense)	(None, 1)	161	dropout_7[0][0]

Total params: 5,666,693 (21.62 MB)
Trainable params: 1,888,577 (7.20 MB)
Non-trainable params: 960 (3.75 KB)
Optimizer params: 3,777,156 (14.41 MB)



Neural Network - Final Model:

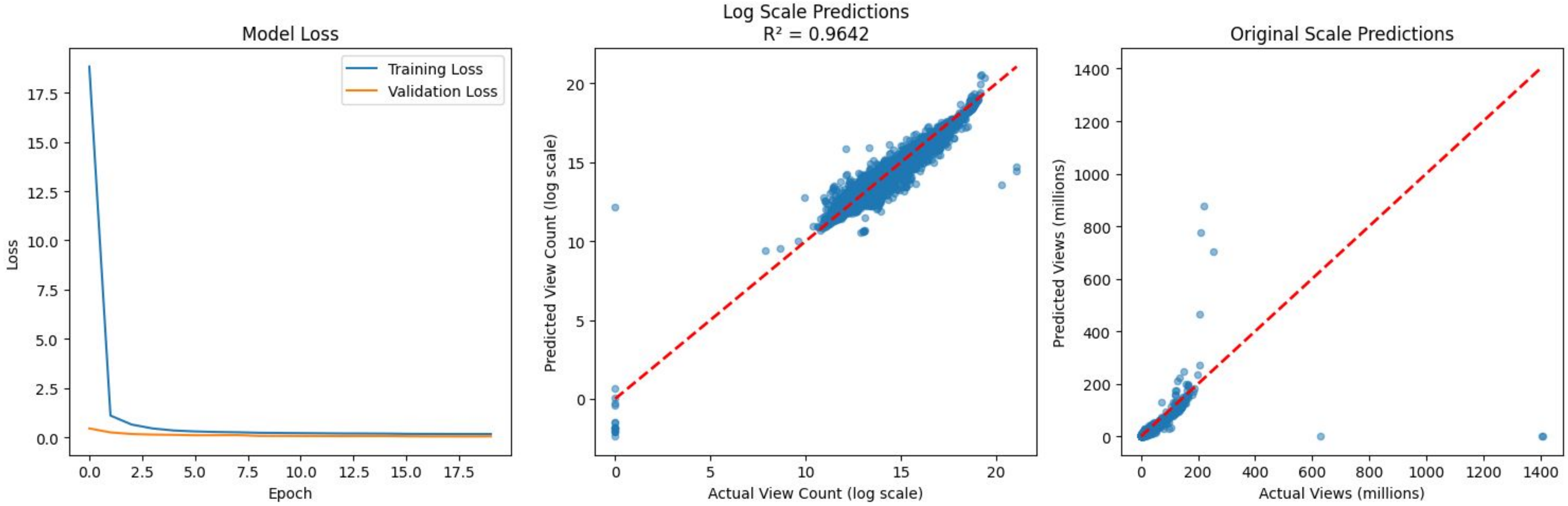
Model Architecture

```
Best Hyperparameters:
num_dense_units: 320
num_dropout: 0.4
embedding_dim: 160
lstm_units: 96
text_dropout: 0.1
num_dense_layers: 1
dense_0_units: 160
dense_0_dropout: 0.1
learning_rate: 0.0021395496476O784
dense_1_units: 160
dense_1_dropout: 0.1
tuner/epochs: 20
tuner/initial_epoch: 7
tuner/bracket: 2
tuner/round: 2
tuner/trial_id: 0012
```

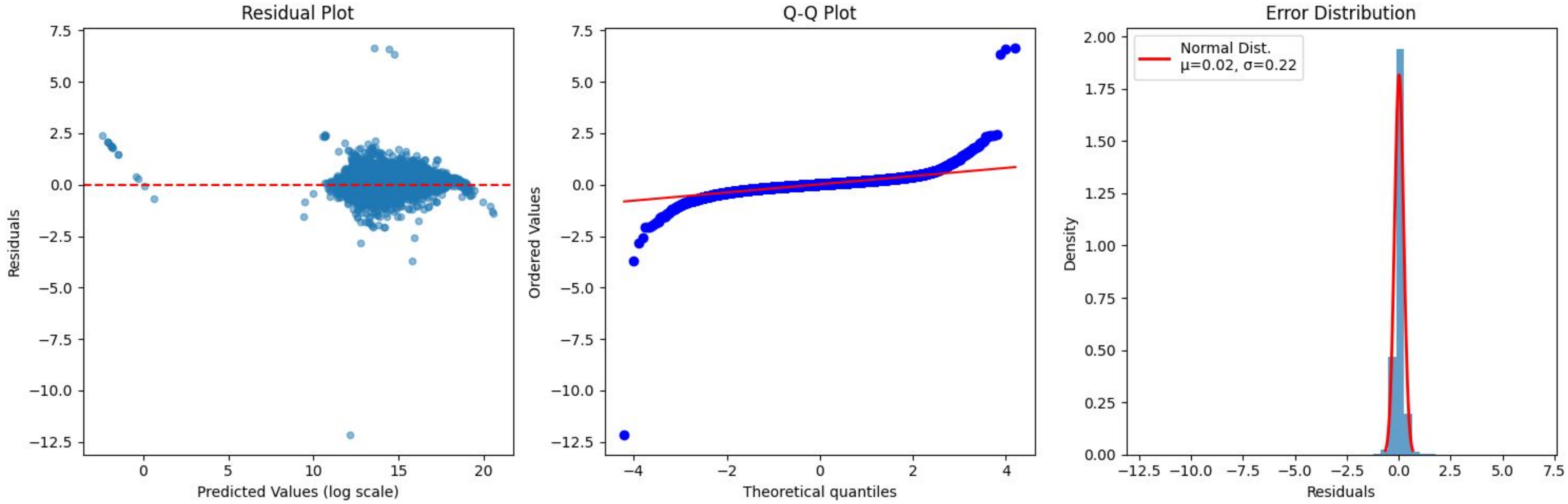
Neural Network - Final Model:
Prediction Results

```
Final Model Performance:
MSE: 0.048
R2: 0.964
```

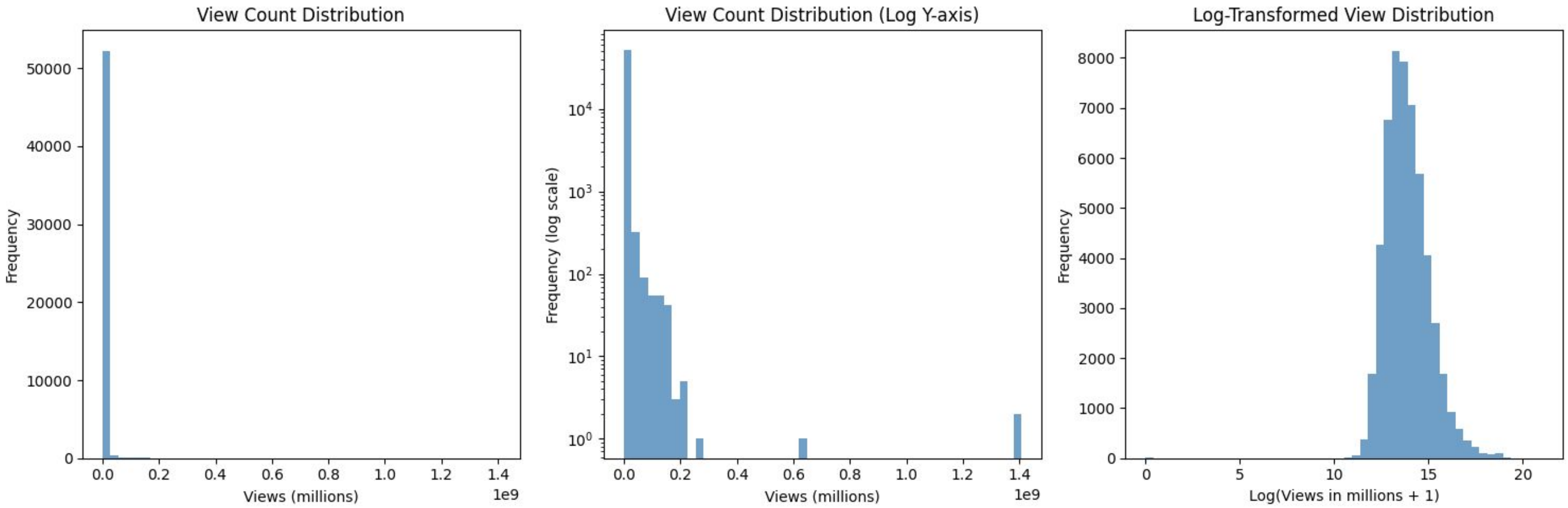
Additional Observation: Validation loss lower than training loss
Hypothesis: The model uses several dropout layers. During training, dropout is active, which adds noise and makes the training harder. During validation, dropout is disabled, which could lead to better performance. Additionally, Batch Normalization could contribute to this.



Neural Network - Error Analysis



Neural Network - Error Analysis




```
View Count Statistics:
Mean views: 2,722,972
Median views: 936,461
Std deviation: 12,563,539

Percentiles:
5th percentile: 215,808 views
25th percentile: 470,785 views
50th percentile: 936,461 views
75th percentile: 2,094,536 views
95th percentile: 8,249,342 views
99th percentile: 30,200,296 views
```

compare to

```
MAE: 520,105 views

Error Distribution (views):
25th percentile: 33,802
Median error: 91,586
75th percentile: 248,850
90th percentile: 674,618
```

Analysis: Our deep learning model demonstrates robust performance in YouTube view prediction across a diverse dataset (220K-30M views). With an MAE of 520K views and median error of 91.5K views, the model handles 75% of predictions within 249K views deviation. The error represents only 19% relative to mean views (2.72M) and scales impressively with video popularity, achieving ~6% error for viral videos (>8M views). Despite high data variance (σ=12.56M), the model maintains consistent accuracy across different view scales, making it particularly valuable for high-stakes view prediction tasks.

Neural Network - Future Work
Two-Stage Cascade Prediction Pipeline

Stage 1: Model Training

1. Train Base Model:
    - Use entire dataset (220K-30M views)
    - Current architecture with full range prediction capability
2. Train Range-Specific Models:
    - Low-Range Model: Specialized for <1M views
    - Mid-Range Model: Optimized for 1M-5M views
    - High-Range Model: Focused on >5M views
    - Each trained on subset of data matching its range

Stage 2: Prediction Flow

1. Input: New video features
2. Base Model Prediction
    - Determines approximate view range
3. Range Selection
    - Route to appropriate range-specific model
4. Final Prediction
    - Selected model provides refined prediction

Model Comparison

Model Name	Description	Results	Pros	Cons
Random Forest	Grows and combines multiple decision trees	MSE: .237 R2: .820	Can learn non-linear patterns	Does not take into account that videos may be related to each other
Gradient Boosting Regressor	Creates multiple decision trees and corrects values as it predicts previous ones	MSE: .0875 R2: .934	Flexibility and accuracy	Large compile time efforts
Time-Series Model & Gradient Boosting	Predicting future view count by combining predictions of multiple decision trees	MSE: 0.166 R2: .818	Takes into account early engagement	Overfitting
Neural Network	**Learns from data propagated through layers of nodes**	**MSE: 0.048** **R2: .964**	**Captures complexity**	**Cost and poor interpretability**

What We Learned

- Different considerations that must be implemented within machine model making
    - Efficiency and time
    - How to feature engineer to obtain the most optimal prediction
- Research and Feature Exploration
    - Completed research on the best methodologies for our project and analyzed engagement metrics and user preferences to understand our data better
- Complex architectures, like neural networks, require far more effort to tune for certain problems

Next Steps/Future Advancements

- Incorporate the relevance scores for keywords
- Figuring out a way to normalize this data

```python
def get_keywords_batch(texts):
    docs = list(spacy_nlp.pipe(texts, batch_size=32))
    keywords_list = []
    for doc in docs:
        keywords = [phrase.text for phrase in doc._.phrases[:10]]
        keywords_list.append(keywords)
    return keywords_list
```

```python
def get_trend_score(keywords):
    """
    Fetch Pytrends interest scores for a list of keywords for the past week on YouTube. Cache results.
    :param keywords: List of strings (keywords/tags)
    :return: Dictionary of keyword scores
    """
    global trends_cache

    # Join keywords into a single query
    query = ','.join(keywords)

    # Check if cached
    if query in trends_cache:
        return trends_cache[query]

    # Fetch trends data
    try:
        pytrends.build_payload(keywords, timeframe='now 7-d', gprop='youtube')
        data = pytrends.interest_over_time()
        if not data.empty:
            weighted_score = (
                (data > 90).sum(axis=0).sum() * 3 +  # High weight for >90
                ((data > 50) & (data <= 90)).sum(axis=0).sum() * 2 +  # Moderate weight
                (data <= 50).sum(axis=0).sum() * 1  # Low weight
            )
            return weighted_score
        else:
            return 0
    except Exception as e:
        print(f"Error fetching trends data for {keywords}: {e}")
        return 0

    # Cache the result
    trends_cache[query] = score
    save_cache()

    return score
```

Final Thoughts

We were able to develop and compare different models to predict the video virality of YouTube videos to some degree of accuracy, with the highest R^2 being 0.964 with the Neural Network model.

The ability to predict video virality aligns closely with several business goals mentioned earlier:

- The results of this model can be used to improve the YouTube recommendation system by prioritizing content with a high likelihood of attracting significant views
    - Improving metrics like watch time, user satisfaction, and user retention
- Creators and advertisers can take advantage of this model by targeting their promotional resources towards possibly trending videos
    - Ad revenue maximization
- It can also help with internal insights for trends and growth: YouTube can identify emerging trends and capitalize on them by increasing visibility of such content

Overall, the models we developed will drive revenue, help improve user and creator satisfaction, and maintain a competitive advantage over other video-sharing platforms.

Questions?