Youtube Clickbait Classifier Report

Jack Wolfgramm

September 2022

Abstract

Clickbait poses a major problem for the YouTube platform. The need to keep viewer watch time high has created an environment that incentives creators to produce content that gathers more clicks. This type of behavior can result can result in long-term dissatisfaction with the platform and, ironically, lower total watch time. To address this issue, developing a preemptive clickbait detection method could be worthwhile. This project explores the possibility of creating a model that can detect clickbait based on thumbnails alone. For this project I relied heavily on the work done by Xie, Le, and Lee in their paper CHECKER: Detecting Clickbait Thumbnails with Weak Supervision and Co-teaching [11]. They details how they generated a small handmade dataset by using Amazon Mechanical Turk, and generated labels to new thumbnails computationally to expand the dataset (this technique is called *weakly-supervised learning*). Finally, a better model could be created by using a co-teaching framework [4]. The paper used both titles and image thumbnails while my project only focuses on image thumbnails. The course of my project followed a similar (but much simpler) trajectory, using transfer learning and semi-supervised learning to try to create the best model possible. However, my results were completely contradictory to the findings of the paper. In particular, I found that training the model on automatically labeled made the model worse, not better. The creator of this project is strongly technically limited compared to the researchers who created the Checker paper and the results of the project have to be understood in light of this.

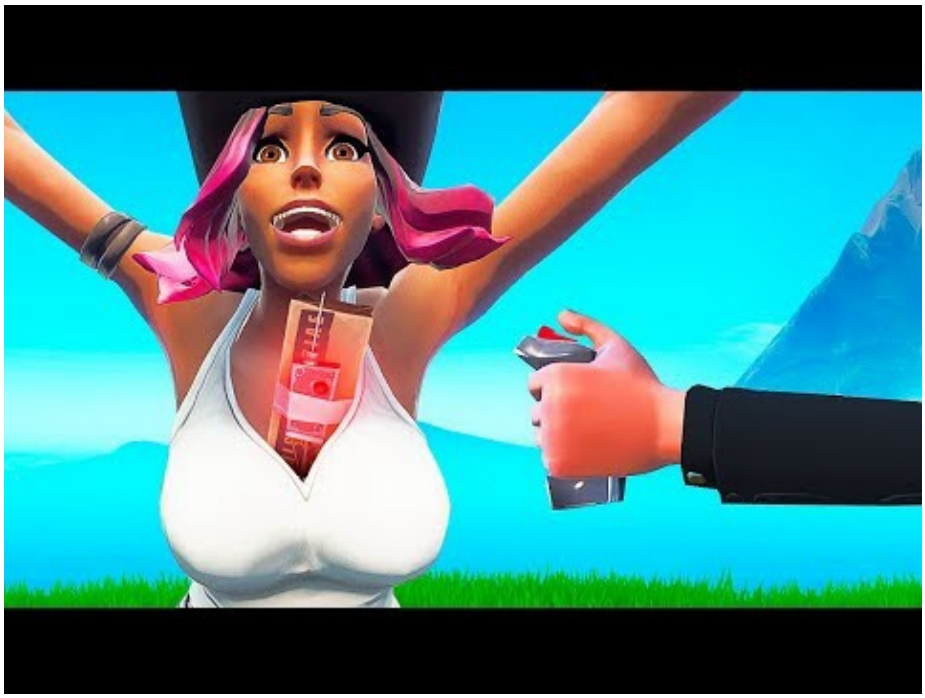

Figure 1: YouTube Thumbnail that was detected as possibly clickbait. Typical features of YouTube Clickbait are apparent.

Contents

1 The problem with clickbait

Clickbait has exploded in the last decade, with the rise of modern social media over more traditional forms of entertainment and news. The problem has perhaps been most strongly felt in the realm of journalism, where "fake news" and the decline of subscription services has created an ecosystem where attention grabbing titles meant to spark outrage are becoming the norm. This has led to the creation of models that detect clickbait news articles [9] [1] [3], an example of applied NLP.

Though not as important on a societal level, clickbait on YouTube creates a serious problem for both the platform and consumers. Dissatisfaction may drive viewers away and creators may become disillusioned with the platform if they feel they must use clickbait to generate their own communities. This led the author to the current project. Manual verification of YouTube videos would be far to timely and costly. If YouTube were able to auto-detect clickbait, then perhaps methods could be implemented to stop their spread. For example, videos flagged as possible clickbait could be put through a 24 hour probation period where the video is shown fewer times to viewers, creating incentive to improve thumbnails. Since only the title, description, and thumbnail can be easily parsed by current methods before upload, this represents a key opportunity to use computer vision methods. Data-centric methods also come to the fore since obtaining large, high quality data is very difficult.

2 CHECKER paper and Data Set

Of course, the idea to detect clickbait with machine learning is not new. Various research in the literature shows that my project was not as original as I thought it was going to be. On the other hand, this provides an opportunity to work with higher quality data and to compare my results with that of others. One of the primary difficulties that I had when starting the project was finding a dataset to work with. Researchers often do not provide their code and dataset to work with. This practice has been shifting somewhat in recent years due to the "machine learning replicability crisis" [5], and, from what the author can gather, it is currently accepted to be best practice to include both data and code. Still, finding a dataset to work with was not necessarily easy. For that, I greatly appreciate the authors of the CHECKER paper. Much of the discussion and methods in the paper are too technical for data science bootcamp project, but it is highly recommended.

Based on previous work done in [12], they were able to crawl 8,987 videos from a variety of clickbait heavy non-clickbait heavy channels. This data includes various video meta-data like the like count, the description, and the 10 most liked comments. They then used Amazon's Mechanical Turk service to manually "crowdlabel" 787 videos. They then used the human labelled portion of the dataset to automatically generate labels on much of the remaining dataset.Label functions were created and Snorkel,Epoxy [2], and a simple majority voter were

compared to find the best label generator.
Though the details escape the author, models which simultaneously take both image and text were used for classification, for example VisualBERT [7]. This differs from the approach taken in my project were I used transfer learning based on an image only model (in particular, VGC19 [10]). The final results are included in figure 2. A co-teaching framework was used to train the models. The

	w/o generated labels		w/ generated labels	
Method	AUC-ROC	F1 score	AUC-ROC	F1 score
SVM	0.7149	0.3830	0.7355	0.4000
Logistic Regression	0.7144	0.4912	0.7629	0.5986
VisualBERT [16]	-	-	0.8460	0.6722
LXMERT [25]	-	-	0.8458	0.6640
UNITER [8]	-	-	0.8196	0.6554
Ours + Block [4]	0.8452	0.6538	0.8644	0.6831
Ours + Mutan [3]	**0.8659**	0.6415	**0.8666**	**0.6933**
Ours + MFH [27]	0.8603	**0.6585**	0.8603	0.6884

Figure 2: Results from the CHECKER paper comparing various models and techniques. Note that F1 and AUC-ROC scores were generally better when the generated labels were used, which is different from my own results in this project.Table taken from [3]

author does not understand what this means exactly but the intuition behind many methods used to help neural nets train on noisy data is actually quite simple. Despite the fact that there are enough parameters that eventually the model can learn even completely random noise, the model will train much faster on data that is not noisy. That is, data that contains "real patterns" will result in much faster training, and hence the loss on each of the data points can be used as a guide to whether the label is noisy or not. Thus, by finding the right balance of how many of the high-loss data points to drop, we can train a model only on the non-noisy labels. While the model will still fail to correctly predict the noisy data in the test set, it will be more accurate on the non-noisy labels. This principle was used in my own project.

3 Project Results and Discussion

The results from my personal project are highly ambiguous. I will note the three largest discrepancies between my experience working with the dataset and what is recorded in the CHECKER paper, in the order of most to least surprising.

- The base model that I created (using transfer learning with VGG19 as a base model) performed notably better than all models created with all

techniques in the paper. It isn't clear what has happened here. Common sense seems to suggest that their is some basic problem with my methods, but the author has no idea how this could have happened. For instance, the method that I used was heavily based on an official Keras tutorial for transfer learning based on image data. My best model (using no generated labels) had an F1 score of 0.8232, while the paper seems to indicate the absolute best model had an F1 score of 0.7153. The ambiguity here cannot be resolved by my unlearned eye, and I would highly appreciate if an expert would review what happened. I will pay you 100 dollars if someone messages me with a good response.

- The method that I used to generate labels seems to perform better than the methods used in the paper. My label generation model achieves an F1 score of 0.678, while in the paper the best score was 0.667. Since frameworks like Snorkel are supposed to work the best, my crude labelling method should, naively, be worse than the methods using labelling functions that are used in the paper.

- Finally, using the generated labels makes my model perform much worse. The F1 score tanks down to around 0.5-0.6. Using data-centric techniques does not seem to help the model trained on generated data significantly (in fact it worsens the matter), but it does seem to help the model trained on hand-labelled data. As a side note, this indicates that the crowdworked labels may be somewhat noisy.

These peculiarities aside, table 1 has the results from my investigation. Iterative trimmed loss minimization is the data-centric method used to try to train over noisy labels.

Method	F1 Score	
	W/out Generated Data	W/ Generated data
VGG19 w/ extra layers (base model)	0.8149	0.5784
+Iterative Trimmed Loss Minimization	**0.8306**	0.3575

Table 1: F1 score with various approaches, bold is the best result.

4 Technical Implementation

4.1 VGG19

It is very hard to train computer vision algorithms. In fact, large competitions like imagenet were created to motivate further research. The imagenet dataset currently contains millions of images in thousands of categories. VGG19 is just one model to come out of the imagenet competition. It can categorize images into 1000 different classes. Obviously VGG19 would work better than something

that could be cooked up in the home lab for the original purpose of classifying imagenet pictures, but what about the current application of classifying YouTube thumbnails? This is where transfer learning can come in handy! The key idea is to use the first 19 layers (where the "19" in VGG19 comes from) and leave off the final prediction layer, and then to put an extra layer that finally feeds to a final binary prediction. The high-level features of the model are already well trained (we can think of this as being able to identity whether there is a man, a cartoon, or whatever VGG is seeing in the image) and we simply train the model to find which combinations of the high level features are indicative of clickbait. This allows us to use the training over millions of images that was done in the model creation, while only having a few examples of the specific kind of categorization that we are searching for.
Concretely, I added a fully connected layer between two dropout layers (to prevent overfitting, which was a problem without them). Finally, a binary prediction was made by outputting a single probability of being in the non-clickbait class. I used the Adam optimizer. Working with images can be extremely memory intensive, so I used the image dataset from directory function that Keras provides to feed the data in batches. Twenty percent of the data was held seperate to run validation. The base layers were frozen and ten epochs were run with the loss and F1 score for each epoch shown in figure 3. Typically, after an

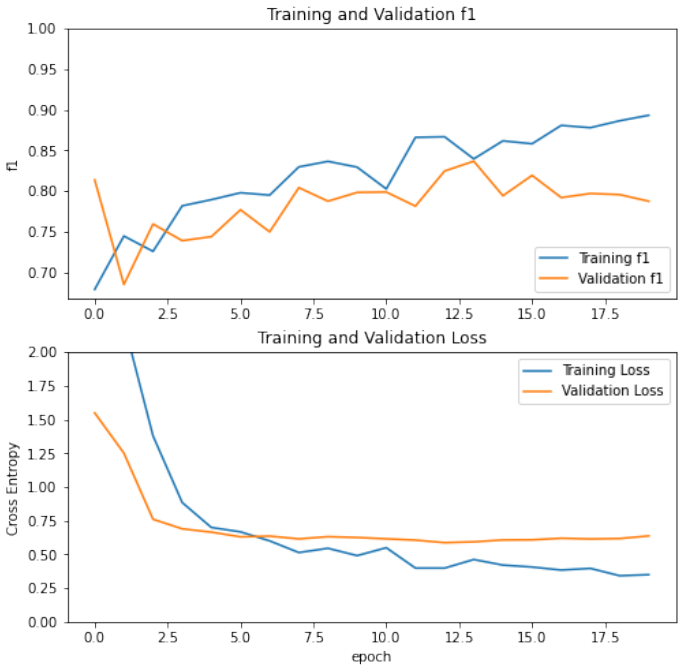

Figure 3: F1 and loss over the ten epochs of training. The model seems to be overfitting towards the final epochs.

initial run over the new layers is done, the best epoch is chosen and a fine-tuning run is performed. This is done by unfreezing the original layers of the model and letting them vary slowly so that the parameters can be better fit to the problem at hand. Training is done this way to prevent the hard won knowledge contained in the model from being lost too early, and instead to try to improve the original model only when decent predictions are being made. Unfortunately, fine tuning did not result in the model improving in any notable way, as shown in figure 4 Overall, the model with no extra generated data and no data-centric

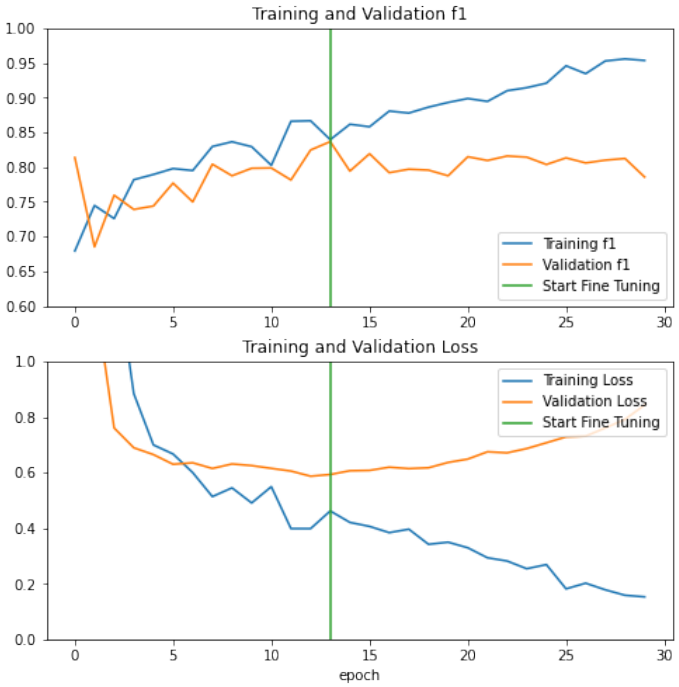

Figure 4: F1 and loss over the ten epochs of fine tuning. It seems that the training all parameters does not help the model, and instead only results in overfitting.

approaches achieved an F1 score of about 0.815. Again, this seems too high because the CHECKER models were not able to achieve this score!

4.2 Label Generation

To explore the possibilities of data-centric machine learning, I also tried to expand the dataset, as in the original work. The hope of label generation is that automatically generated labels will still contain a large enough signal-to-noise ratio to be beneficial for training. To that end, it is very important to generate labels as accurately as possible. In practice, when attempting to

generate labels according to both F1 and precision metrics, it was found that the models performed much better when trained on labels optimized for F1 score (perhaps the non-clickbait dataset became too noisy, or the small amount of extra precision gained was not worth the recall trade-off).
Included with the thumbnails is various metadata about the video. The author found that the most important predictors of clickbait were the title, description, and the most popular comments of the video. The text data was simply run through a TFIDF Vectorizer and then fed with the rest of the data to the model. XGboost was found to give the best performance (versus random forests or logistic regression. The flow of data can be pictured in figure 5. Since the

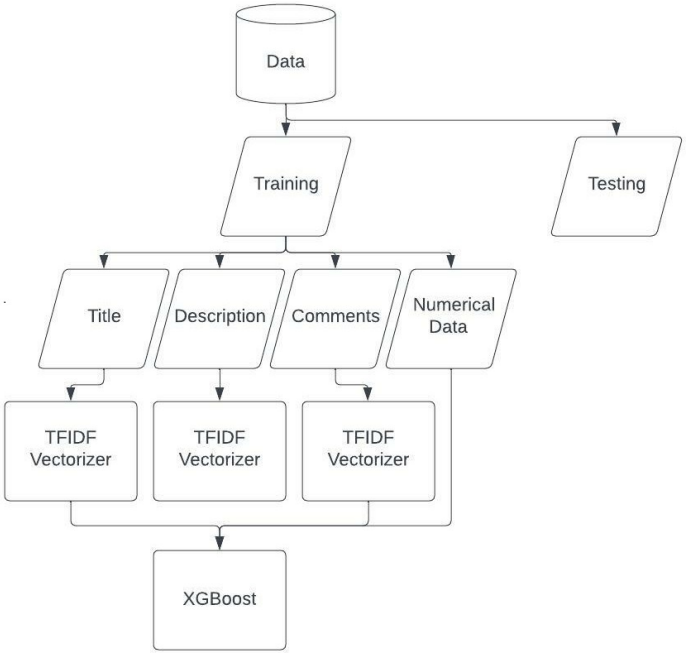

Figure 5: Pipeline created to generate the labels. The method seems crude compared to using labelling functions and finding their correlation, overlap, etc. However, the model seems to work well.

overall quality of the labels should kept as high as possible, hyperparameter tuning was done using Optuna, a hyperparameter optimization framework that tends to work much faster than both grid-search and random-search. Since the optimization is done so quickly, only 200 runs were done, and there was only marginally better improvement after the 50'th run. The final results and comparison can be seen in table 2. As noted, the F1 score using this method seems to improve on the F1 score in the paper. Labelling functions were not used in this project. Labelling functions are functions that return a best guess based on a single criteria. Individually they are very weak, but taken together

	CHECKER Paper			This Project
	Majority Voter	Epoxy	Snorkel	XGBoost
F1 Score	0.635	0.637	0.667	0.678

Table 2: F1 score of my own approach vs. the methods taken in the paper. The paper used labelling functions.

they can be quite accurate.
Using the model created, I was able to generate labels for the remaining 8,200

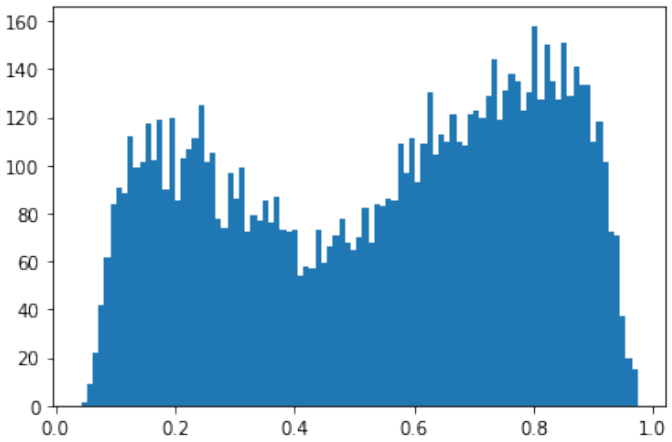

Figure 6: Histogram of the predicted probability a given data point is not clickbait. Despite generally falling in to two categories, many points are quite uncertain, indicating that the model did not see similar data during training.

thumbnails (increasing the size of our dataset by 10-fold). However, it should be noted that the model seems very unsure about where to put the majority of the thumbnails, see figure 6.

4.3 Iterative Trimmed Loss Minimization

Finally, we have to train our model using the new data! A portion of the hand labelled data was set aside for validation. This is important because we should not do our model testing on generated data points. Again, starting with a fresh model (VGG19 + extra layers) we ran trained the model over ten epochs. As can be seen from 1, doing this made the model much worse. To that end, the author tried to use data-centric methods to try and improve the training over noisy data.
The original plan was to use cleanlab. Cleanlab takes any model (as long as it can fit in to a scikit learn wrapper), and automatically finds and cleans noisy labels. It does this by training the model over some portion of the dataset,

and finding the data points with significant loss on the remaining fresh data. Thus, iteratively, it finds the data that is likely mislabelled throughout the whole dataset. However, cleanlab will not work with the current framework! The problem has to do with the way that data is fed in batches into the model. In particular, since scikit learn does not support batch data, there is no way (as far as the author can determine) to use cleanlab. Perhaps Pytorch could be utilized in conjunction with cleanlab.
The CHECKER paper uses a co-teaching framework with a similar idea (that high loss datum should not be trained over), but coding co-teaching in tensorflow/Keras is beyond the technical acumen of the author. Again, Pytorch would probably work much better in this case. Luckily, tensorflow does support enough easy customization to be able to use Iterative Trimmed Loss Minimization (ITLM) [8]. The central idea is like the other methods: find which datum cause the highest loss, and don't train over them. This introduces another hyperparameter, alpha, which is the percent of data in each batch that should be trained over. Many, many thanks Irene Kim [6], whose medium article on how to practically implement this method was invaluable as TensorFlow is a black box to me!
ITLM was tried to improve results with non-generated and generated data. Surprisingly, using ITML failed to improve the model when used in conjunction with noisy data. In fact, any significant alpha made the model much worse. On the other hand, it did improve when training over the human data. The hyperparameter tuning can be seen in figure 7. Of course, this result has to be

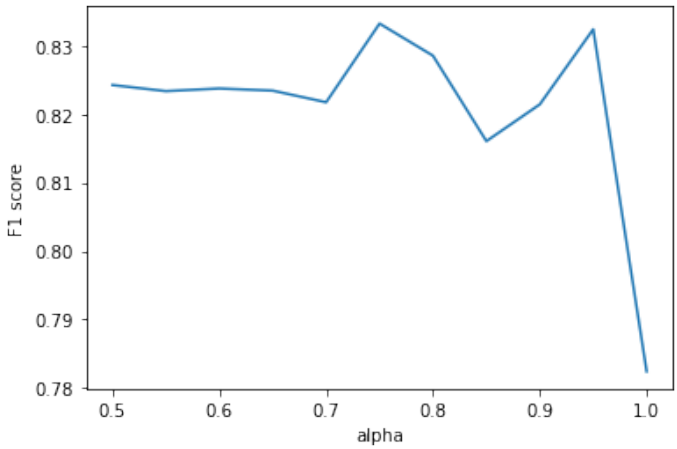

Figure 7: Plot showing alpha vs best F1 score over ten epochs. The pattern is unclear but seems to indicate that some of the data is noizy.

taken with a grain of salt for two major reasons. First, ITML seems to make the model worse exactly when we expect it to make it better. Second, it isn't exactly clear how this changes the model. ITML remains an interesting approach,

but unless more work is done to show that it works like theory expects, results should be taken with caution.

5 Conclusion

Personally, I have learned a great deal from this project. Primarily among them is that I have very little idea what neural nets are or how they work. They do some sort of training over the data and make some sort of prediction (which seems better than random guessing), but what exactly is happening and what constraints should be placed on the predictions remains very unclear. Trying to stay on a positive note, transfer learning and data-centric approaches seems like a very rich and promising area for future learning. Finally, I believe that learning Pytorch would be a very good investment.

Practically speaking, if I were to continue to work on this project, I would try the following approach: first train a model on the hand-labelled data, and only then try to train the model on the generated data. It is possible, though unlikely, that using this method with a combination of ITML would lead to better results. Due to time constraints I did not have time to try this implementation.

Finally, a brief remark on the business value of the models discussed. Frankly the author does not believe that any model produced so far would be valuable for YouTube. Even if a model could be created that was fairly accurate (on the current data), it isn't clear that the model would generalize to other data. For example, most of the thumbnails were collected from only a few different channels. It generally thought that neural nets can learn almost any pattern - including accidental ones. This could cause the model to learn what the style for particular channels in the dataset - rather than learning more generally what clickbait thumbnails look like. Due to this and many other reasons, the chase to create both good data, and a good model for the problem of YouTube clickbait will continue for the forseeable future.

References

[1] Abhijnan Chakraborty, Bhargavi Paranjape, Sourya Kakarla, and Niloy Ganguly. Stop clickbait: Detecting and preventing clickbaits in online news media, 2016.

[2] Mayee F. Chen, Daniel Y. Fu, Frederic Sala, Sen Wu, Ravi Teja Mullapudi, Fait Poms, Kayvon Fatahalian, and Christopher Ré. Train and you'll miss it: Interactive model iteration with weak supervision and pre-trained embeddings, 2020.

[3] Yimin Chen, Nadia Conroy, and Victoria Rubin. Misleading online content: Recognizing clickbait as "false news". 11 2015.

[4] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. Co-teaching: Robust training of deep neural networks with extremely noisy labels, 2018.

[5] Sayash Kapoor and Arvind Narayanan. Leakage and the reproducibility crisis in ml-based science, 2022.

[6] Irene Kim. Handling noisy label data with deep learning, Jul 2022.

[7] Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. Visualbert: A simple and performant baseline for vision and language, 2019.

[8] Yanyao Shen and Sujay Sanghavi. Learning with bad training data via iterative trimmed loss minimization, 2018.

[9] Kai Shu, Suhang Wang, Thai Le, Dongwon Lee, and Huan Liu. Deep headline generation for clickbait detection. 08 2018.

[10] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[11] Tianyi Xie, Thai Le, and Dongwon Lee. Checker: Detecting clickbait thumbnails with weak supervision and co-teaching. In Yuxiao Dong, Nicolas Kourtellis, Barbara Hammer, and Jose A. Lozano, editors, *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track*, pages 415–430, Cham, 2021. Springer International Publishing.

[12] Savvas Zannettou, Sotirios Chatzis, Kostantinos Papadamou, and Michael Sirivianos. The good, the bad and the bait: Detecting and characterizing clickbait on youtube. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 63–69, 2018.