# Instagram Bot Finder Final Report

**Presentation:**
https://cornell.zoom.us/rec/share/s9bu44QKUfkDVoY15kDXoA30MOIOgVFI8k6CzGCkv39sRosr5ujrgqKpAu9aF-kS.CjbmnDTLEJCvoa_c?startTime=1653073427000

**Team:**
Emily Hur (esh76)
Megan Jung (mj374)
Eric Zhang (ewz4)

**AI Keywords:**
Neural Networks, Image Classification, Computer Vision, Natural Language Processing/Text Classification, Logistic Regression, Machine Learning

**Application setting:**
Instagram accounts

**Project Description**

Our initial vision for the project we proposed was to use language processing and computer vision techniques to build a classifier that can distinguish between Instagram accounts and comments that belong to spam accounts (bots) versus humans. Using logical reasoning, it's easy for us to deduce which accounts are real based on context clues. Unlike real accounts, bot accounts tend to be centered around adult content, false advertisement, or scams involving money. They tend to flood the comment section of celebrity and sports accounts and negatively impact the Instagram experience. Some users even comment about the large ratio of bots to real people commenting under posts. Bots can even be hazardous, either imparting malware viruses on the user or scamming them of large quantities of money. With this in mind, there are an estimated one billion active participants making Instagram one of the most popular social media networks worldwide, especially among our generation. Yet, fake users remain rampant and may represent up to 10% of accounts, according to Ghost Data in 2018 (Williams, 2018). It's most likely that the percentage has increased since 2018. Therefore, we seek to leverage artificial intelligence to address the aforementioned issue, creating a binary classifier that can identify these spam accounts with the purpose of eventually eliminating them.

Rather than consult publicly available data resources, we took the initiative to create our own dataset in order to control the variables included. Since bots are most concentrated under the comment section of popular Instagram accounts, we focused on recent posts from ESPN, SportsCenter, and Barstool. From this sample, we sifted through the comments and manually selected public "human" and "bot" accounts at random, collecting data on the same features for each. In order to prevent violating privacy rules and minimize confounding variables, we did not

take any information from private accounts. We selected equal numbers of each type of account so we wouldn't need to upsample or downsample the dataset in the future. The features we chose to focus on fell under three categories: numerical (follower count, following count, follower/following ratio, number of posts, number of likes on comment, written (username, comment, biography), and visual (profile picture). Lastly, we included a column classifying each account, assigning it a '1' if it corresponded to a bot and a '0' otherwise.

One can see that fake accounts fall under a wide range, whether it's investment promoters, pyramid scheme recruiters, or people posing as attractive young women. Similarly, comments vary from "I need bf" to "For a start I deposited $15,000 to test the waters, after 9 days I got a return of $100,900." Since our goal was to develop a binary classifier, we wanted a product that would recognize all types of bot behavior.

**Constraints**

A few constraints emerged as we began our data collection. Firstly, we had hypothesized that spam accounts would have low follower/following ratios. This proved to vary a lot, most likely due to bots being followed by other fake accounts. Therefore, the importance of this feature proved to be more unreliable than anticipated. Additionally, there were some accounts that didn't follow any accounts at all, which forced us to set the ratio to a default value of 0 to prevent a division-by-zero error. Another caveat was that some accounts had posted a story, which means a pink and yellow ring appears around their profile picture. This may have impacted the accuracy of our image classifier, as it converts pixels into arrays. Lastly, a major alteration we made to the project was the size of the dataset. We'd initially proposed to collect data for 600 accounts total, but ended up with 314. Since the process was taking longer than expected and we wanted to get started with training our model, we ultimately decided to risk going with a smaller dataset. This means any outliers may have a larger impact on the classifier. Additionally, while we tried to remain objective, the accounts were individually selected based on comments which could've introduced bias. We attempted to counteract these unexpected obstacles by creating several models with different combinations of account characteristics to maximize accuracy despite the smaller dataset.

Overall, however, we didn't stray far from our proposed procedure. As we began conducting research and training models, we fleshed out our plan and outlined exactly how we wanted to train multiple Bot Finders. Although there were some conflicts and delays in workflow, we were still able to successfully code multiple bot classifiers with high accuracy.

**Key AI Concepts.**

Since much of our numerical data lacked trends, the artificial intelligence concepts central to our project were natural language processing and computer vision. We had to train computers to recognize and interpret text and images. From here, the main branch of AI that we focused on was machine learning, a form of supervised learning in which observed values are

used to categorize inputs (usually into two classes). The main algorithm used was logistic regression, implemented both as a singular neuron and with a neural network.

**Overall Process**

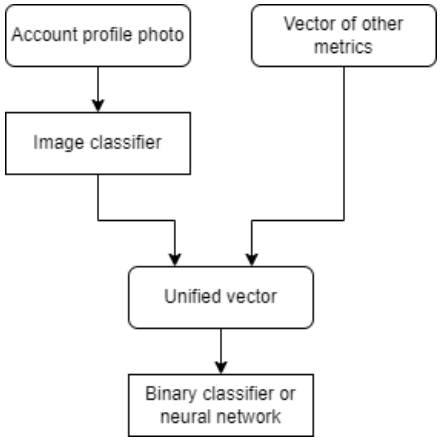


Figure 1: The overall process of our program.

Each account has a profile photo and a vector of other metrics. The vector of other metrics includes account values like the follower to following ratio and the words present in its comment and biography. The profile photo is a separate file, though both can be matched by the account's username.

The process of our classifier is shown in Figure 1. The account's profile photo is first run through the image classifier to determine if the picture is characteristic of a bot. The output is 1 if yes, 0 if no. This value is then combined with the vector of other metrics, and this unified vector is then fed into a binary classifier. This final binary classifier, either using logistic regression or a trained neural network, will consider all the characteristics of the account and determine if it is a bot or a real person.

**Natural Language Processing-Baseline Model**

First, we tackled text classification by creating a baseline model, hoping to use the results to optimize future models. We noticed that many comments and biographies contained emojis that would interfere with the natural language processing. Thus, our initial approach was to eliminate emojis from any inputs containing text using the clean-text package. Other standardized text-preprocessing included converting the text to lowercase, removing \n characters, and expanding contractions. Following this, we used an 80/20 split to separate our data into training and test sets.


Figure 2: Example of bag-of-words algorithm

The first approach we used was a bag-of-words representation which creates a "vocabulary" based on all the unique words present in a column, assigning each of these words an index. CountVectorizer from the sci-kit learn library can transform strings in numerical arrays of fixed length, with each entry in the array corresponding to how many times that word appears in the input. An example of this can be seen in Figure 2. We used this strategy twice, once for comments and once for biographies. We decided not to do this on usernames because those are unique to every row and the resulting matrices would've been too sparse. Seen in Figure 3, this generated 2,124 columns, which produced a data frame comprised mostly of zero values.

| | 000 | 05 | 0598 | 064acf | 09 | 10 | 100 | 100k | 10k | 12 | ... | youtu | youtube | yt | yuhs | zainab | zainabzilahi | zeldon | zilahi | zooming | zu |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **246** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **247** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **248** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 3. Data frame of numerical columns

Next, we normalized the numerical columns to ensure convergence of the classification algorithms, scaling all features to the range [0,1]. We concatenated the data frame of numerical columns with the data frame containing the transformed text for 2,130 columns total. We tested the accuracy of five different classification algorithms on our test set: logistic regression, naive bayes, k-nearest neighbors (n=5), decision trees, and support vector machine. The respective accuracy percentages based on whether the predictions were correct were 92%, 89%, 83%, 89%, and 92%.

**Natural Language Processing-Other Models**

Despite the baseline model performing well, there were a few changes we wanted to implement before evaluating the validity of these models. Firstly, an emoji dictionary was used

to convert all emojis to text. For instance, if a comment used a red heart emoji (❤️), it would be replaced by the word “heart” (Mahto, 2021). Initially, we had concerns about this method altering the meaning of the comments or leading them to be grammatically unclear/incorrect. However, we later saw that this didn’t affect the accuracy of our results and it seemed that the meaning behind the comments and biographies persisted.

We also noticed that a few of the columns (such as “your”) were repeated in the original model because separate vocabularies were constructed for the comments and biographies. To address this, the two text inputs were concatenated into a single feature and the bag-of-words algorithm was only applied once. The number of columns increased slightly from the baseline to 2,320, which can be attributed to the introduction of emojis as text.

We attempted to further improve the model through feature engineering. First, a low variance filter was used with a threshold of 1% to weed out words with low frequency. This resulted in a data frame of 620 columns. We used recursive feature elimination to select the fifty words most relevant to determining the target, a number we arbitrarily selected based on trying to balance information with efficiency. After combining these 50 columns with the original numerical ones (follower/following ratio, number of likes on comment, etc), we re-fit logistic regression (as it had the highest accuracy score with the baseline model). After applying the model to our test set, we got an accuracy score of 92% again. We noted that very few of the 50 words kept corresponded to emojis (except for money_bag), suggesting that the message behind the biographies and comments would be the same even if they were removed. The similar accuracy score also supports the efficiency of this model despite the elimination of over 2000 features. The five most important features based on the magnitudes of their coefficients were ['money_bag', 'ly', 'alive', 'bf', 'none'], which make sense in the context of the types of comments the spam accounts leave. It’s also likely that ‘ly’ is related to the hyperlinks that many of these accounts have in their bios.

The bag-of-words model is often cited as the simplest way to convert text into numbers. Since language is so integral to this model, we also tried a second method of text classification, TF-IDF. In contrast to the bag-of-words technique, this algorithm generates a number by weighing the frequency of a term against how rarely it’s used in the document. In the world of AI, TF-IDF tends to perform better. However, when applied to our dataset, the accuracy was the same at 92%, likely due to our limited sample size. Still, in accordance with outside findings, we decided to move forward with the TF-IDF algorithm despite the similarities in predictive ability. All five of the most important features were completely different: ['my', 'me', 'app', 'netlify', 'bio'], suggesting that there might be multiple words that are indicative of bots. This also supports the statement that the numerical columns are less important than words. This is congruent with our original hypothesis based on the lack of trends in follower/following ratio, for example, when collecting data.

Lastly, we attempted to incorporate the output of text classification as a feature itself. We removed the numerical columns completely and generated predictions based on the comments and biographies after fitting a logistic regression model. We then appended this output to a data

frame containing the normalized numerical columns. After applying a new logistic regression model, the accuracy was around 89%. Since we'd noticed in the other classifiers that the numerical columns weren't very relevant in the final model, we decided not to go with this approach although the accuracy was relatively high.

**Evaluating the Model**

We had five main questions pertaining to the assessment of our project:

- Which natural language processing model was the best?
- How well can our model predict whether or not an account is a bot?
- How can we fine-tune our model?
- What evidence is there to support the validity of our model and its predictions?
- Is the neural network worth implementing?

**Final Model**

In summary, after training multiple bot classifiers, we decided on the following methodology: combining the comments and biographies into a single feature, applying TF-IDF, filtering for the 50 most important features, and using the logistic regression algorithm. This was based on findings from online resources, accuracy percentages, and feature engineering. This finalized model was saved using joblib so it can be loaded in the future.

**Accuracy of the Model**

For a better metric of accuracy, we decided to use K-fold cross-validation, which is a popular technique used to minimize the bias of a model, especially for smaller datasets. The dataset was randomly shuffled and split into five groups. The following process is repeated five times: each subset is removed and treated as a test set, and a logistic regression model is fitted on the remaining four groups and evaluated on the allocated test set. The mean of the five evaluation scores is then reported. Since the usual train/test split method relies heavily on the way the data is divided, this is often cited as a more accurate way to determine which classifier is best. We used K-fold cross-validation on the three best-performing classifiers of the baseline model: logistic regression, naive bayes, and SVC. The corresponding accuracies were now 94.4%, 92.0%, and 94.0%, so we selected logistic regression for hyperparameter tuning. This corroborates that the features included are relevant to a variety of classification models and that through feature engineering, we created a dataset with all the relevant information allowing multiple algorithms to detect spam accounts.

**Fine-tuning the Model**

The parameters of logistic regression that can be set by the user include penalty, C, and solver. Using a randomized grid search, we test 90 different combinations of these parameters in order to determine the ones that yielded the highest accuracy score. Ultimately, the final logistic

regression model selected was {'C': 2.782559402207126, 'penalty': 'l2', 'solver': 'newton-cg'} and the corresponding score was 96.4%.

**Further Assessing the Model**

To better visualize the predictive ability of this final model, we generated a confusion matrix and classification report after running this final model on the test set which can be seen in Figure 4 (Mayer, 2021).

```
[[33  4]
 [ 1 25]]
              precision    recall  f1-score   support

           0       0.97      0.89      0.93        37
           1       0.86      0.96      0.91        26

    accuracy                           0.92        63
   macro avg       0.92      0.93      0.92        63
weighted avg       0.93      0.92      0.92        63
```

Figure 4: Confusion matrix and classification report

The test set contained 63 entries, 34 of which were real accounts and 29 of which were bots. The binary classifier correctly predicted 32 of the 34 real accounts and 26 of the 29 bots, which is 97% and 86% respectively. The global accuracy was 92%. Generally, the threshold for a good classifier is conventionally thought to be 70%, which ours far exceeded.

Finally, we assessed our model using ROC curves and P-R curves as seen in Figure 5 (Hinton, 2020). Precision is defined as the number of true positives divided by the number of true positives + false positives. Recall is calculated by dividing the number of true positives by the total number of true values (true positives + false negatives). Ideally, we wanted to remove as many false positives (increasing precision) as possible without decreasing recall too much. A high-performing classifier that strikes a balance between precision and recall despite their inverse relationship should hug the upper right corner of the graph. With an average precision score of .95, we can conclude that our classifier labels most of the positive samples (bots) correctly.

The ROC (receiver operating characteristic) curve plots the true positive rate against the false-positive rate. The AUC value assesses the performance of the model at different classification thresholds by taking the area under the blue line pictured below. The AUC value returned was 92%, which means the probability of the model distinguishing between the two classes is roughly 92%. In comparison, the area under the “random” classifier in red would be around 50% which is equivalent to random guessing.

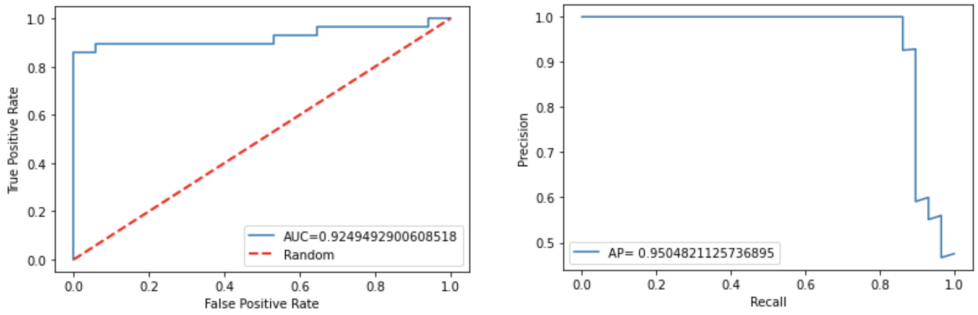


Figure 5: ROC/AOC (left) and PR curve (right)

Under both measures of average precision and AUC, our model proved to be successful. Ultimately, we can confidently conclude that we were not only able to classify convoluted language inputs successfully, but also use those results to train a model with consistently high accuracy.

**Image Classification**

The image classification portion of the algorithm is meant to determine if an account's profile picture is characteristic of a bot account or a real account. The image classifier takes in a picture input and will output a 0 if the picture is not bot-like and 1 if it is. This output value will then be inserted into the vector containing all of the other metrics for an account. Finally, this vector will be passed to a neural network which will evaluate an account on all of the metrics.

The image classifier is a neural network built using the TensorFlow library. The model has a simple sequential architecture, meaning that an input will flow through the network one layer at a time, with each layer taking in a single input and producing a single output. We created two models, a basic model and a model with features to avoid overfitting.

The first layer of the basic model is a rescaling layer, which scales the RGB values of an input image down to between 0 and 1. This is because TensorFlow neural networks perform better on smaller values. Next in the network are alternating layers of convolution and max-pooling. While the convolution layers alter the image, the max-pooling layers downscales the image so that features can be detected more easily. Finally, the image is flattened into a vector and inserted into a fully-connected, or dense, layer, so that every pixel can be taken into account. The final output layer produces a vector that holds the likelihood score of each classification. In our case, the classifications would either be 'bots' or 'people.'

The model with overfitting avoidance is essentially the same as the basic one, only now it has a data augmentation step and a dropout step. Data augmentation will actually produce more samples from the existing training dataset by altering them slightly. We are left with more believable-looking examples that allow the model to see more distinguishable features. The dropout step layer will ignore a certain percentage of the training data. This limits the effects of overfitting by leaving out the occasional sample.

Both networks were trained on the screenshots of profile pictures collected. The results of the training are below. With more epochs, the accuracy of both models generally increased. However, the basic model had slightly increasing loss for both the validation and the test set the more it was trained. On the other hand, the model with overfitting avoidance had decreasing loss for the training set, and slightly increasing loss for the validation set. The training progression is shown in Figure 6.

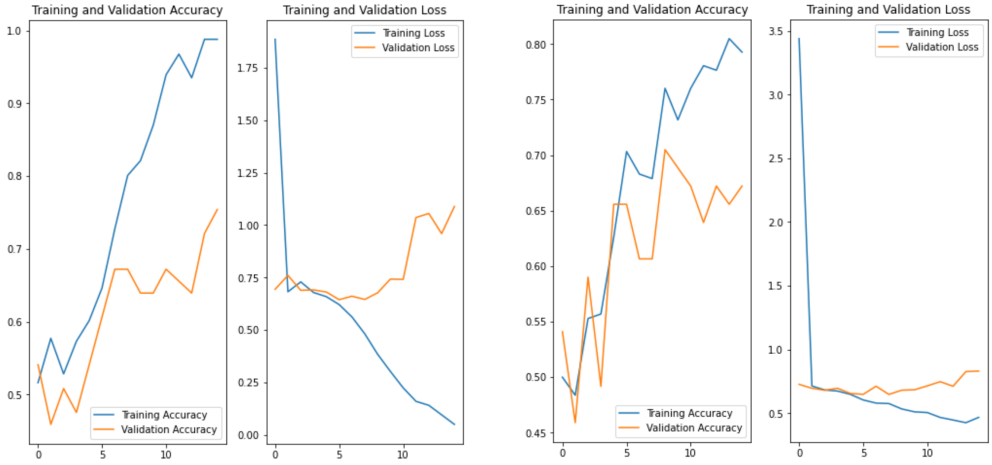


Figure 6: Training performance for the basic model (left) and the model with overfitting avoidance features (right) with respect to the number of epochs

**Neural Network**

Although our logistic regression model proved to be successful, we wanted to take things one step further and implement a neural network. We adapted skeleton code for a basic neural network in Python available online so we could pass our own dataset as an input (Brownlee, 2016). This process involved a lot of debugging, and we almost decided to forego neural networks completely given the success of our logistic regression model. However, neural networks are a type of supervised learning common to many machine algorithms, and we agreed it was worth trying to implement as we sought to further improve the performance of our classifier beyond even 92%.

Thanks to our extensive model testing outlined above, we applied a similar methodology here to standardize the input. We used text classification to parse the biographies and comments, and the 50 most relevant features were selected. The only difference is that we replaced TF-IDF with the bag-of-words algorithm because all the features are normalized, which may not fare as well with the low decimal values outputted by TF-IDF. This was combined with the numerical columns and exported as a .csv file.

The neural network reads in a .csv file and normalizes all the features. The network consisted of an input layer (with 56 inputs), five hidden layers, and an output layer with two neurons, one for each class. The network was first initialized with randomized weights. First, forward propagation was used to generate predictions that would eventually be modified. Backpropagation was used to train weights based on the difference between the system output

and the expected classification (gradient descent). The number of epochs was set to 500 and the learning rate was .1, both determined by convention. After training the network on the training set for 500 iterations for 5 folds, we were finally able to generate predictions using the test set. The performance was evaluated based on whether the accounts in the test set were correctly classified, and the final accuracy percentage was 98.4%.

Logistic regression is comparable to a neural network with a single layer. Given the added complexity of hidden neurons and the iterative nature of this model, it was expected that the accuracy would be higher. The tradeoff is that neural networks can take longer to train and be susceptible to overfitting, which we tried to counteract by splitting the dataset. However, given that these models are applicable to a social media platform with over a billion users, we believe any increase in accuracy is worth looking into. Considering the context of this project, we thus suggest going forward with the neural network model as opposed to the basic logistic regression outlined above, although both were successful.

**Combining Image and Text Classification**

After evaluating the initial performance of the neural network and concluding that it was successful, we decided to move ahead with our program by adding image classification as an input to the network. The image classifier model was used to generate a corresponding output for each account, either a 0 or a 1, depending on the algorithm's assessment of the profile picture. So, the input to this final neural network had 57 inputs, 5 hidden layers, and two outputs. We again trained the network with 500 iterations and five folds to make predictions on the test set. This time, the final accuracy percentage was 100%. This network may be subject to overfitting because the neural network for the image classifier was trained on the same dataset. We tried to minimize this with a train/test split, but this model should be further tested with publicly available resources or new unseen data.

**Conclusion**

In summary, we produced five final models: two trained image classifiers, one logistic regression model, and two neural networks (one with image classification as a feature and one without). All five of these had high accuracy rates and were evaluated on metrics ranging from predictive ability to loss and precision. Going forward, we believe the logistic regression model and neural network integrating text, image, and numerical features should be used as the final products. The former is relatively simple, yields high accuracy percentages, and scores high on a multitude of performance metrics. The latter is more intricate and time-intensive and has some constraints/sources of error to consider, but seems promising and could perform well with more training.

Resources:

Bansal, Shivam5992. “A Comprehensive Guide to Understand and Implement Text Classification in Python.” *Analytics Vidhya*, 26 July 2019, www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python.

Brownlee, Jason. “How to Code a Neural Network with Backpropagation In Python (from Scratch).” *Machine Learning Mastery*, 21 Oct. 2021, machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python.

Hinton, Samuel. “PR vs ROC Curves - Which to Use?” *Samuel Hinton*, 15 July 2020, cosmiccoding.com.au/tutorials/pr_vs_roc_curves.

Mayer, Adrian. “Using K-Fold Cross-Validation to Evaluate the Performance of Logistic Regression.” *Medium*, 6 Jan. 2022, python.plainenglish.io/using-k-fold-cross-validation-to-evaluate-the-performance-of-logistic-regression-4439215f24c4.

Williams, Robert. “Instagram May Have 95M Bot Accounts, The Information Reports.” *Marketing Dive*, 19 July 2018, www.marketingdive.com/news/instagram-may-have-95m-bot-accounts-the-information-reports/528141.

TensorFlow Image Classification: https://www.tensorflow.org/tutorials/images/classification

TensorFlow Saving Models: https://www.tensorflow.org/tutorials/keras/save_and_load