ELEN 6889 Final Project
Real-time images detection and hashtags generation for tweets

Zian Wang **zw2776** Qishuo Wang **qw2360** Yunhe Liu **yl4879** Xingyu Wu **xw2813**

I. Problem Definition

A hashtag is used to index keywords or topics on Twitter. People usually use the hashtag symbol (#) before a relevant keyword or phrase to categorize their Tweets and help them search in Twitter easily. However, we need to attach the relevant hashtags manually before we post a tweet, which is very inconvenient. If relevant hashtags can be recommended for users to choose from, it will be more efficient and user-friendly. Therefore, the project aims to predict hashtags of the images attached to the real-time tweets. In the project, we only focus on 12 specific animal classes.

Our project is divided into two part: stream processing and hashtag prediction. Stream processing includes producing Twitter stream and optimizing it by load shedding. Hashtag prediction includes images pre-processing by resizing and normalization, predicting hashtags using well-trained model, sampling stream from results and displaying the images with predicted hashtags. The overall pipeline of the project is shown in Figure 1. For more details about our project, please refer to the following website: https://github.com/cjdsj/Real-time-images-detection-and-hashtags-generation-for-tweets.

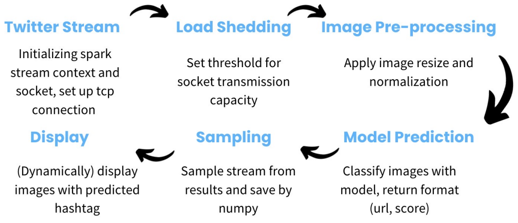
Twitter Stream
Initializing spark stream context and socket, set up tcp connection
Load Shedding
Set threshold for socket transmission capacity
Image Pre-processing
Apply image resize and normalization
Model Prediction
Classify images with model, return format (url, score)
Sampling
Sample stream from results and save by numpy
Display
(Dynamically) display images with predicted hashtag

Fig. 1: Overall Pipeline

II. Challenges

Firstly, our project focuses on 12 specific animal classes. However, in the real-time twitter streams, there are a large amount of images that not belong to these classes. Although we can create a class named others to contain these images, it is hard to extract the features of the class because there are so many different things in the real world. Therefore, one challenge is to process the images not belongs to the 12 classes.

In addition, we want to extract the images under the hashtags to form a dataset at the beginning. However, we found that the images under a specific hashtags are not always related to the hashtag. For example, under the #hens hashtag, it not only has images of hens but also images of eggs, chicks and roosters. We need to denoise the dataset if we want to have a perfect model performance. It is hard to do it manually by our teammates because we need a large amount of images, like ten thousand images per class. To add insult to injury, while completing the project, we found that the dataset we get from Twitter is unbalanced, which means that there are a large amount of images under popular hashtags such like cats and dogs while there are no more than 10000 images under cold hashtags such as spider and hen. Therefore, we need to find a way to denoise these images downloaded from Twitter and make up for the images in the class that has insufficient images..

Next, the inference time of neural network is very long while the throughput of the Twitter stream is usually very high. It is a necessity that the throughput of stream is no more than processing speed. Otherwise, we cannot process all the data in the stream. So it would be a challenge to pick up an optimization method to control the throughput of Twitter stream to fulfill our model processing requirements.

III. Technical Details

A. Dataset

1) Local Dataset

The dataset used in the project is an animal image classification dataset downloaded from Kaggle. It includes 12 classes: butterfly, cats, cow, dogs, elephants, hen, horse, monkey, panda, sheep, spider and squirrel, which is shown in Figure 2. Each class includes 1400-1500 images. However, in each class, some images that don't belong to the class exists and the existence of these images will affect the accuracy of our teacher model.

Because the number of pictures is not a lot, we chose to denoise it manually by ourselves. In addition, a new class named others is created for the images other than the 12 classes of images because Twitter not only have the 12 hashtags and 12 classes of images.

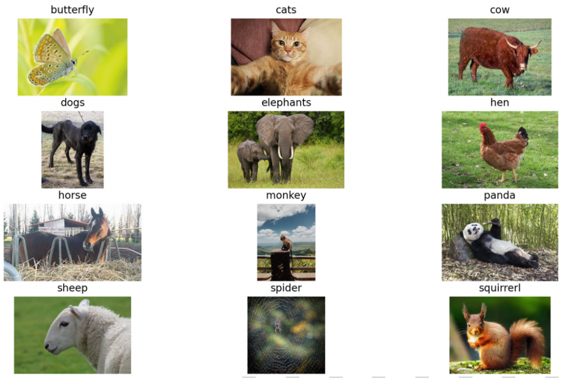

Fig. 2: Dataset

B. Transfer Learning

Transfer learning is a machine learning method that reuses an initial model developed for task A in the process of developing a model for task B. The basic idea of transfer learning is the similarity between two different tasks.

For example, task A is to classify cats and dogs, while task B is to classify tigers and lions. These animals can be distinguished from hair, body size, shape, etc. Therefore, there is lots of shared knowledge between task A and task B. Assuming existing a model trained for task A, we could retrain the existing model for task B based on the knowledge of task A rather than scratch.

In this project, we use *ResNet50* with bottleneck residual blocks as the basic model, which has been pretrained on ImageNet to implement classification. There are two major transfer learning scenarios:

1) Fine-tuning the base model

We initialize the network with the pre-trained *ResNet50* network, instead of random initialization. Here, we aim to reduce training costs.

2) Fixed Feature Extractor

In order to extract the features of the pre-trained *ResNet50* model, we freeze the weights of all the layers in the network, except that of the last fully connected layer. To further improve the classification accuracy, we define a new last fully connected layer and only the layer would be trained.

If we use the pre-trained model as a fixed feature extractor, the training speed would be extremely fast since there is only one fully-connected layer going to be trained. Yet the performance will not be very good considering the features for two datasets are different. So we choose fine-tuning the base model. This method needs more time for training yet performs better.

C. Semi-Supervised Learning

For deep learning, larger dataset means better performance. Since we have Twitter API, we decided to get more data from Twitter to help us train a better model. But the problem of images on Twitter is that there are so many noises. In order to deal with this issue, we introduced semi-supervised learning(1). Our learning pipeline is showed in Figure 3. And it can be described as:

(1) Train a teacher model on local dataset D;
(2) Run the trained model on noised Twitter dataset U and select positive images for each label to construct a new dataset D';
(3) Train a new student model on D';
(4) Fine-tune the trained student on the labeled set D.

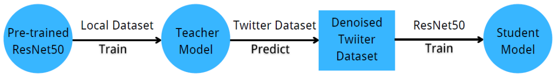

Fig. 3: Learning pipeline of semi-supervised learning

The result of semi-supervised learning is showed in TableI. Although the accuracy of student is a little low at the beginning, it exceed that of teacher model after fine-tuning on the local labeled dataset.

	Test Accuracy
Teacher Model	95.34
Student Model	94.61
Fine-tuned Student	96.07

TABLE I: Result of Semi-Supervised Learning

D. Model Compression

In stream processing, there are several algorithms for data pre-processing to compress the data, like quantization and dimensionality reduction. Since our input data is images, we can not compress the data size, but instead of that, we can compress the model size and reduce model inference time.

There are three methods in model compression: quantization, weight pruning and knowledge distillation. In this project, we implemented weight pruning. Unlike other simple networks, ResNet has the dimensionality dependency between each pair of convolutional layers(2). In Figure 4, we can see the layers in red have dimensionality dependency because their output will be added together at last, so they are un-prunable. And all other layers in green are prunable.

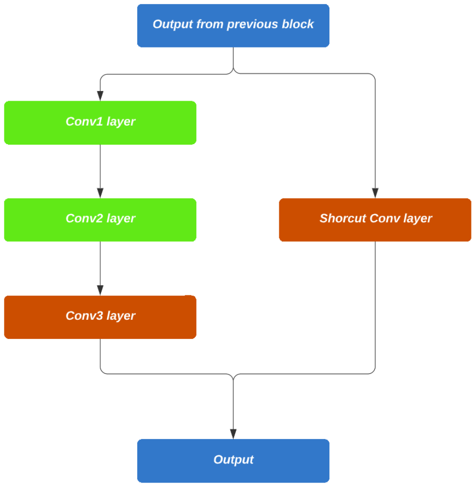

Fig. 4: Prunable and un-prunable layers

In order to measure the importance of each neuron in one layer, we use Average Percentage of Zeros $(APoZ)$(3) as the index.

$$PL_c^{(i)} = \frac{\sum_{k=1}^{N} \sum_{j=1}^{M} f(PL_{cj}^{(i)}(k))}{N \times M}$$

where i is the layer index, c is the channel index, M is the total number of validation samples and N is the dimension of the output feature map. Function f returns 1 if the variable inside equals 0. In conclusion, APoZ measures the percentage of zero in activation output of each neuron for all the samples in the validation-set. The higher APoZ, the more useless this neuron is. If the APoZ of the neuron is larger than the standard deviation of the average APoZs in the same layer, then we can prune this neuron. In order to remove more neurons, we prune it iteratively. In each iteration, we prune a small number of neurons, then fine-tune the networks.

TableII shows our result of iterative pruning. Every time after pruning, accuracy will drop significantly. But after fine-tuning, it will rise back to high. Also, even though we pruned almost 30% neurons in the model, it's accuracy only drops less then 1.2%.

Iteration	Pruned Accuracy	Final Accuracy	Sparsity
1	90.04	96.01	0.0718
2	86.17	95.81	0.1276
3	87.39	95.56	0.1690
4	84.58	95.27	0.2072
5	78.72	95.07	0.2399
6	69.41	94.88	0.2670

TABLE II: Iterative Pruning

IV. Streaming Processing

In order to display specifically about how our model performs, we utilize two stream strategy to show the results. Firstly, we use general search method where we get all tweets with images after searching. And we also use targeted stream rules where we filter the stream that only the tweet containing the keyword in our classes will be accepted. The we set up socket, TCP connection and stream context components to send data to Pyspark with specified load shedding threshold, window size and max sending epochs.

A. Data pre-processing

1) Noised dataset generation

Firstly, we need to use Twitter v2 'recent' endpoint to get the noised dataset for our teacher model to denoise and then learned by our student model. And we have previously talked about that the classes we choose are not all popular in Twitter, so in order to fulfill our noised dataset amount requirement, we firstly applied hashtag searching, and the boundary of hashtag searching can be detected by the parameter *next_token* in Twitter API. Then we use keyword searching if the queried dataset is not enough. Figure 5 shows a brief strategy that we implemented to get the noised dataset:

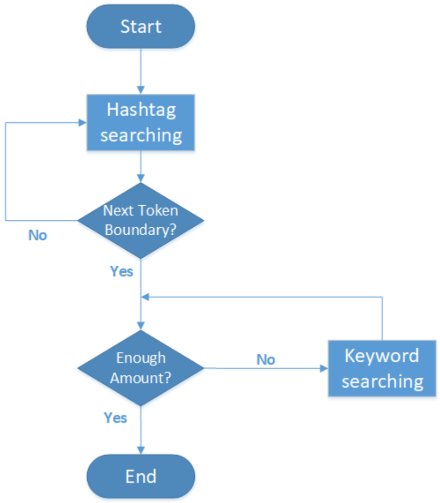

Fig. 5: Dataset Preparation Flow Chart

2) Get stream data from Twitter and send to Pyspark

In order to detect real-time images on Twitter, we should use another endpoint of Twitter named 'stream' with proper stream rule settings. As mentioned before, we use general stream rule and model targeted stream rule to get the stream from Twitter and then send the data to pyspark.

In our sender module, we firstly set up a port with an IP address waiting for connection. Meanwhile, in receiver module, we initialize the stream context and the socket parameters which the sender module listens to. Once we receive the connection from the receiver, we start to get stream from Twitter endpoint and send data to pyspark. At sender part, we set the load shedding strategy as follows:

(1) *window_size* is shared between sender and receiver, and we calculate the approximate time cost of our model based on a specific input flow (here we define as *max_input*)

(2) Increase *max_input* until the time consumption is similar with the window size, then set the threshold as *max_input* at sender part

(3) *max_epoch* is used to set how many times that sender should send data to spark. Since we have threshold at sender, once the amount of sent data reaches the threshold, we will call the sender to sleep for *window_size* and wait for the next epoch sending.

For more details, please refer to (here) for more information. Figure 6 shows the procedures how we link those modules to process the streaming data:

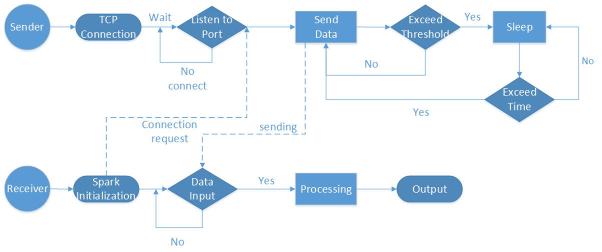

Fig. 6: Procedures of Pyspark and Twitter Stream Connection

B. Spark and torch

1) Inference architecture

In order to speed up classification, we re-partition input data frame into $(2, M, N)$ partitions, where 2 means the number of models in each GPU, m is the number of GPUs per worker, and N is the number of workers per driver. The inference architecture is shown in Figure 7.

2) Broadcast

In spark, some data (model with all the data and configure files here) needs to be transferred into each task in each executor (GPU here). If there are T tasks in an executor, each task needs a copy of the data transferred in from the driver, which will lead to a large amount of duplicate data in an executor. Hence, there is a variable, broadcast, in spark, only sending a copy of data to the executor in order to greatly reduce the duplicate data in each executor. To explain clearly, the principle of Broadcast is shown in Figure 8.

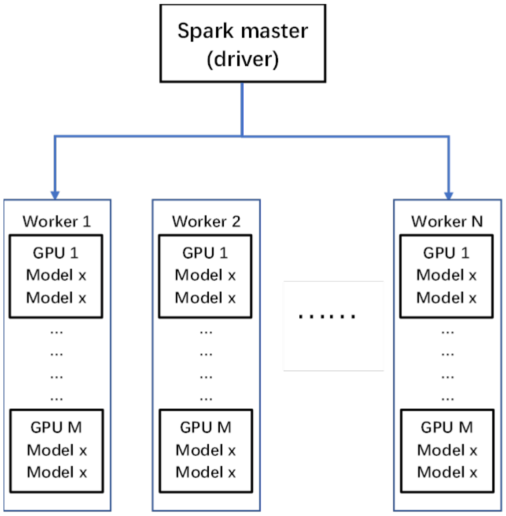

Fig. 7: Inference Architecture

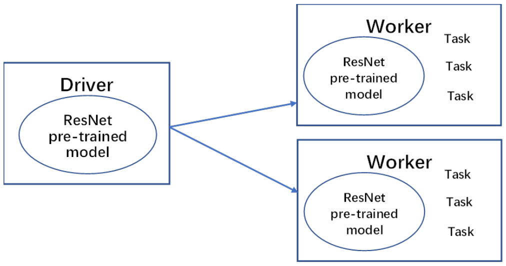

Fig. 8: Broadcast

3) Pandas-UDF

$Pandas\_UDF$ is a new API in PySpark 2.3. Spark transfers data through Arrow and uses Pandas to process data. $Pandas\_UDF$ uses *pandas_udf* to declare a function, whose methods include Scalar (scalar map) and Grouped Map (grouped map). In this project, we wrap prediction function as a $Pandas\_UDF$ to predict the classification of image data. The steps to predict image classification using the Spark and PyTorch are as follows:

a) Import package

(1) Enable Arrow support. Apache Arrow is an in-memory columnar data format used in Spark to efficiently transfer data between JVM and Python. When converting Spark data frames to Pandas data frames, we can use Arrow for optimization;

(2) Import libraries such as pandas, pytorch, and pyspark;

(3) Set Pytorch to define whether to use the GPU.

b) Prepare and broadcast the pre-trained ResNet50 model

Load $ResNet50$ model from dbfs:/FileStore with 13 categories on the Spark Driver node and broadcast the status of the $ResNet50$ model. dbfs:/FileStore is a special folder that provides high-performance I/O for deep learning workloads.

c) Define and load required model

Define $get\_model\_for\_eval$ function to return pretrained ResNet50 model that loads the parameters of ResNet50 model from Spark Broadcast variable. (4)

d) Build a custom Pytorch dataset

We extract images from the URLs in stream and convert them to RGB, saving them as a image data set.

e) Define predictions Function

(1) Augment images with PyTorch, such as resizing, normalization and so on. Take together multiple data transformation steps.
(2) Convert arrays to Tensor data structures.
(3) Call $torch.utils.data.DataLoader$ function to load image data. Each batch contains 500 images.
(4) Get the parameters of the $ResNet50$ pretrained model broadcast by Spark.
(5) Predict classification for images in each batch.

f) Wrap the function as a Pandas_UDF for model prediction

Predict and save the results of prediction in /dbfs/FileStore/animals/ with the above predictions function and custom image data set.

V. Future Work

A. Scaling

Scaling in stream processing is to handle fluctuations in stream demand. For example, there is a Music Festival. With viewers taking pictures and uploading to twitter at the same period, there will suddenly be huge fluctuations in stream demand. It is therefore important to prepare a scaling plan in advance to handle more concurrent stream before the servers become overloaded.

Firstly, we implemented load-shedding which could effectively control the rate of flow. Secondly, in order to accelerate the whole process, we need more and better CPUs instead of GPUs. Our team did an experiment about inference time using NVDIA Tesla V100, it turned out that inference time of model do not limit the throughput of whole pipeline. The really time-consuming part is reading images from URL and transfer data from CPU to GPU. For I/O operations, we can use multi-threads to parallelize. And for transferring data, more CPUs can do multi-processing to accelerate.

B. More hashtags

The result of the project shows that we can added a hashtag for the image within our classes, and the classification accuracy is over 95%. But it would be more user friendly and more realistic if we can provide multiple hashtags for the image. Therefore, two methods might be applied to deal with this scenario.

Firstly, since the score of the image is expressed as a list of values for each class, we can set an appropriate threshold that for each value of the class, if the value exceeds the threshold then we regard it as the recommended hashtag. In this way we can get multiple hashtags to recommend, and the key problem is how to set the threshold properly to get accurate prediction as well as multiple classification results.

Another method is that we can apply multiple binary classification models to predict the image class by class. Therefore we can apply 12 models and each model will only predict whether the image belongs to the current class or not. And here suppose we have 12 classes $\{D_1, D_2, \cdots, D_{12}\}$ and if we want to train the model M_1 for D_1, we will set the images belonging to D_1 as positive samples and all other samples as negative.

C. Model compression

Model compression is to reduce model usage cost. By increasing the inference speed, the amount of model parameters and computation is reduced. Nowadays, the main methods of model compression include pruning, quantization and knowledge distillation. We have used pruning in our project and will continue to compress the model using quantization and knowledge distillation.

1) Qualification

For images, we always use a matrix of pixel values to save them, which is a method that occupy a great deal usage because of using 8 bits that represent 0-255 for each pixel value. However, since adjacent pixel values are contiguous, we don't need to use 8 bits per pixel to store them. (5) For example, we have a 3*3 image shown as below, where the number is pixel value. We could use 3 bits to represent center point 8, and then use 2 bits to represent the offset: -1 means +1, 0 means no offset, -1 means -1. Then there is a great saving in data storage space. The original 256 values, each of which is 8 bits, requires a total of 2048 bytes to record all the data. After using the new method, each value can be expressed as the size of the center point of 3 bits and the offset of 2 bits, then it becomes 5 bits to represent a number, and a total of 1280 bytes are enough. The process is qualification.

7	8	9
7	8	8
9	9	7

2) Knowledge Distillation

Knowledge Distillation means transformation of the predictive power from a complex model to a smaller network. In traditional neural network, the goal of training is to make predicted value as close as possible to true value (Hard-target). In knowledge distillation, it is a training process that uses the class probability of teacher model as a soft-target.(6)

For example, in MNIST data set to recognize the handwritten digit, assuming that an input ”2” is more similar to ”3”, the output probability of ”3” of softmax will be higher than another negative label. Another ”2” is more similar to ”7”, the output probability of ”7” will be higher than others. The values of Hard-target corresponding to these two ”2” are the same, but their Soft-target is different. Therefore, soft-target contains more information than Hard-target.(7)

Soft label modifies the softmax function and increase the temperature coefficient T, which defined as below:

$$q_i = \frac{e^{z_i/T}}{\sum_j e^{z_i/T}}$$

where q_i is the probability of each class output, and z_i is the logits of each output. When $T = 1$, it is the standard Softmax formula. The higher the T, the more attention the training will pay to the negative label. The training algorithm is shown as below:

(1) Training teacher model on the original dataset $\boldsymbol{U}$.
(2) Use to generate soft-target.
(3) Training student model through soft-target, T_{high} and hard-target, $T = 1$.
(4) Set $T = 1$ of student model and implement inference online.

Not only knowledge distillation compresses model, but also enhances the generalization ability of the model.

References

[1] I Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, and Dhruv Mahajan, “Billion-scale semi-supervised learning for image classification,” *arXiv preprint arXiv:1905.00546*, 2019.

[2] Nima Aghli and Eraldo Ribeiro, “Combining weight pruning and knowledge distillation for cnn compression,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3191–3198.

[3] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang, “Network trimming: A data-driven neuron pruning approach towards efficient deep architectures,” *arXiv preprint arXiv:1607.03250*, 2016.

[4] Raajay Viswanathan, Arjun Balasubramanian, and Aditya Akella, “Network-accelerated distributed machine learning for multi-tenant settings,” in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 447–461.

[5] Kurt Demaagd, Anthony Oliver, Nathan Oostendorp, and Katherine Scott, *Practical Computer Vision with SimpleCV: the simple way to make technology see*, ” O’Reilly Media, Inc.”, 2012.

[6] Jian-Hao Luo and Jianxin Wu, “Neural network pruning with residual-connections and limited-data,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1458–1467.

[7] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami, “The limitations of deep learning in adversarial settings,” in *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 2016, pp. 372–387.