Hashtag Generation with CIFAR-10 / CIFAR-100 Image Classification

Weijian Li
CS 557
Emory University

Abstract

This project aimed to develop a software to generate trending hashtags base on input images. CIFAR-10 and CIFAR-100 dataset were used to train machine learning models, which could help conduct objects recognition on the input images. I used a simple CNN as baseline model. An improved deeper CNN, a Residual Network (ResNet) and a VGG16 network were trained to improve the prediction accuracy. Keras with Tensorflow backend was used to implement the networks. A NVIDIA GTX 1060 3G and a NVIDIA Tesla K80 were used during the training process. With class label predicted, the label was then used to query Twitter API to get relevant trending Hashtags.

1. Datasets

Both CIFAR-10 and CIFAR-100 dataset were used in this project. Both datasets contain 60,000 images, among which 50,000 were chosen randomly for training and the remaining 10,000 images were used for testing. Each picture is 32 by 32 pixel in RGB 3 channels. The dataset itself was represented by a numpy array of shape [60000, 32, 32, 3]. The two datasets differ on that the CIFAR-100 contains 20 coarse labels, with each coarse label containing 5 fine labels, which gives 100 classes in total, while the CIFAR-10 dataset only has 10 classes. This difference also caused a higher difficulty for predicting precise labels for CIFAR-100.

2. Data Pre-processing

Before being fed into the models, all the training images were divided by 255 and then subtracted by per-pixel means. The reason for this process was to normalize the value at each pixel, so that the value could be in the same range, and making the fitting process more stable.

Another process is called data augmentation (Figure 1). I randomly adjusted the hue and saturation for each image, randomly added noises, and randomly rotated and flipped them before training. This process was meant to make the model more robust when encountering the same objects with different angles, shades or qualities. Data augmentation was applied by ImageDataGenerator of Keras, which allows processing the image in small batches before each epoch in real time. This real-time process would make the training process faster.

Figure 1: Data Augmentation

3. Models and Performance

3.1 Baseline Model

The baseline Model I tested first was a shallow Convolutional Neural Network. The structure is shown as in Figure 2. One convolutional block contains two Convolution 2D layers and a Max Pooling layer. There were 2 blocks separated by a Drop Out layer, and then a Fully Connected layer with output size 512, and finally a Fully Connected layer with output size of 10 for CIFAR-10 and 100 for CIFAR-100. The kernel size of each convolutional layer was 3 by 3, which is the smallest kernel size that can extract spatial features, such as up, down, corner, straight lines, etc. The filter number was 64 for the first block, and 128 for the second block. ReLU was used as the activation function for every convolutional layer. This structure was proposed by Keras official tutorial [1].

With learning rate of 1e-4 and no decay, this model achieved an accuracy of 47.51% on CIFAR-100 and 81.3% on CIFAR-10 in 250 epochs, and the performance was used as a baseline.

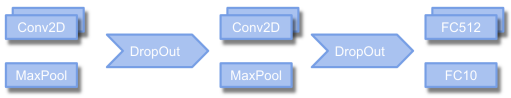

Figure 2: Shallow CNN Configuration

3. Models and Performance

3.2 Improved Convolutional Neural Network

One natural thing to be developed based on the baseline model was to make the network deeper. As shown in Figure 3, for each convolutional block, I increased the number of convolutional layers to 3, and the number of filters to 3 times of the previous model, which gave me 92 filters for each layer in the first block, and 192 layers for the second layer. Instead of Max Pooling, the last convolutional layer for each block was of strides size 2, which deepened the network and down sampled the data at the same time. Finally, a convolutional layer with kernel size 1 by 1 and filter number 10 for CIFAR-10 and 100 for CIFAR-100 was applied to get the corresponding output dimension.

With learning rate of 1e-4 and no decay, this model achieved an accuracy of 51.45% on CIFAR-100 and 87.99% on CIFAR-100. Both accuracies were higher by a considerable amount than the base model. As shown in Figure 4, the learning curve on CIFAR-10 was decent. No overfitting problem occurred, with validation accuracy and training accuracy increasing almost simultaneously. However, the training accuracy was highest at around 0.9, which left a large space for improvement. With the training accuracy apparently started converging, meaning that it was not able to learn the remaining 10%, the models performance just stopped increasing. The similar phenomenon also happened on CIFAR-100 as shown in Figure 5. With that, a more complicated network was needed.

Figure 3: Improved CNN Configuration

3.3 Residual Network

The Residual Network (ResNet) was once the state-of-art model for image classification. It was designed to solve the gradient degradation/ explosion problem for deep networks. Typically, more convolutional layers might not increase the performance, because as gradient propagates through more and more layers, the magnitude could be diverging or converging to 0. To solve this problem, instead of training a mapping H(x): x $\rightarrow$ y, we find the residual F(x) = H(x) - x, and train the new mapping H(x)= F(x)+x [2], which takes the residual into consideration.

The structure of the network naturally contains adding original x to the output of convolution of x. There are 2 types of residual blocks shown in Figure 6. Each block contains two branches, one is the main branch with convolutional layers, batch normalization layers, and activation layers, another is the short-cut layer with x unchanged. They differ on that the second type of block has a convolutional layer in its short-cut layer. I stacked the two types of blocks alternatively: after every 2 type-1 blocks was connected to a type-2 block. Finally, I added a fully connected layer of output size 10 for CIFAR-10 and 100 for CIFAR-100. With learning

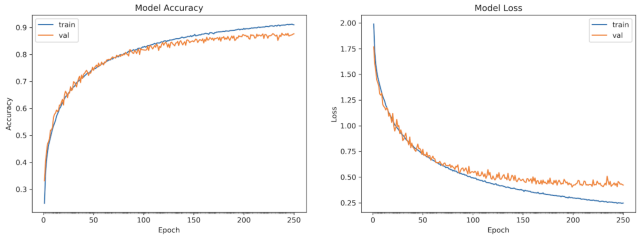

Figure 4: Performance of the Improved CNN on CIFAR-10

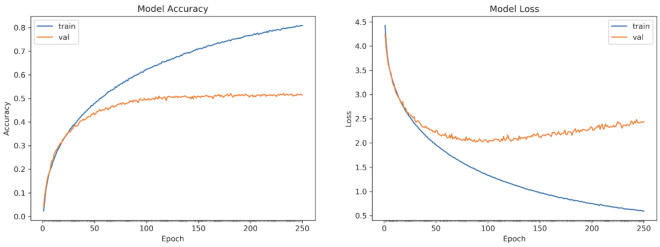

Figure 5: Performance of the Improved CNN on CIFAR-100

rate 1e-3 for epoch 1-7, 1e-4 for 8-16, 7e-5 for 16-30 and 7e-6 for the remaining epochs, I finally achieved an CIFAR-100 accuracy of 63.1% and CIFAR-10 accuracy of 78.1%. Even though the performance on CIFAR-100 was improved by a large amount, 63.1% was still not enough for accurate image classification. Also, the performance on CIFAR-10 was even worse than the previous model. The reason was that this model led to a severe overfitting problem before the validation accuracy arrived at a high level. The hyperplane described by the objective function was far more complex than simpler CNN, which required the control for learning rate and weight decay to be more delicate than simpler model. However, the training accuracy using ResNet actually converged to 1, meaning that on one hand, it was overfitting, but on the other hand, this model learned more from the training set than other models did, so it had more potentials for better performance with properly tuned parameters.

3.4 VGG

Since ResNet was too complex for me to fine tune the parameters, I tried a simpler model called VGG. The structure of VGG16 network is shown as below in Figure 7 [3]. On top of this structure, I added a batch normalization layer after each convolutional layer, added a dropout layer after each

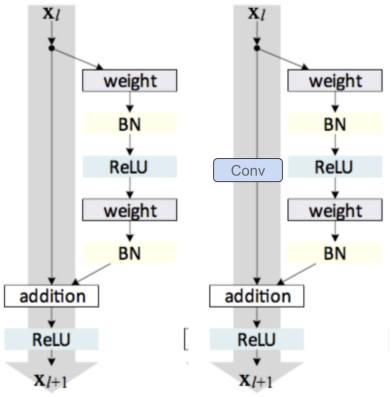

Figure 6: The 2 Types of Residual Blocks

max pooling, reduced the filter number to 16, 32, 64 and 128, and reduced 2 fully connected layer to only 1 with output size 512. This adjustment aimed to fit the smaller size of CIFAR-10 and 100, because the original model was design for ImageNet dataset, and this was inspired by this GitHub repository [4].

This model did a surprisingly well job. With learning rate 1e-3 and timed by factor 0.5 when hitting plateau and weight decay of 0.0005 for every convolutional layer, this model achieved 93.48% accuracy on CIFAR-10. This model is much simpler than ResNet and easier to tune. However, as shown in Figure 8, the model didnt achieve an accuracy close to 1 compared to ResNet, and it suffered from gradient vanishing towards the end of the training.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 7: VGG16 Configuration

4. Hashtag Generation

I finally chose VGG16 and CIFAR-10 as my model and dataset for hashtag generation. The test input images were downloaded from the internet, and then re-sized to 32 by 32 pixels. The resized images were then fed into the model for inference. Pillow and Python-twitter were used to conduct hash tag query. A sample output is shown in Figure 9. In the root directory, just type `python Hashtag.py cat1.jpg` in the terminal to run the script. All test images are store in /data folder.

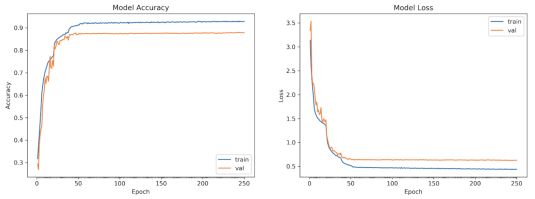

Figure 8: Performance of VGG16 on CIFAR-10

5. Conclusion

Compared to other networks, ResNet was indeed good at avoiding gradient degradation or explosion, however it was harder to tune. With more residual blocks stacked, I expect that the advantage of ResNet will become more and more obvious. ResNet with more and 1000 layers was built because of the flexibility of this model, but the training process will be much longer.

For relatively deeper network like VGG and ResNet, the control of learning rate and weight decay seemed to play a more crucial role during the parameter tuning phase compared to simpler CNN. For the first 2 models, I did not change the learning rate throughout the training and still got decent performance. However, with improper hyperparameters, ResNet and VGG could be worse than simpler CNN.

```
I guess this is a:  cat
#cat  #Cat  #kitten  #Kitty  #cats  #WetNoseWednesday  #catsofinstagram  #Cats  #Kittens  #Kitten  #Pets  #Pet  #Meow  #Moe  #blackcat  #kitty  #pets  #blackcatsofinstagram  #CuteCats  #dog  #meow  #CatsOfTwitter  #PERFECT  #CAT  #blackcats  #pet  #music  #puppy  #catordog  #art  #moma  #newburyport  #pomeroyphoto  #cat_features  #cat_features  #catstagram  #catstagram  #FatCat  #FatCat  #CheddarCuddlesCat  #CheddarCuddlesCat  #CheddarCat  #CheddarCat  #christmascat  #christmascat  #catssleeping  #catssleeping  #eastcoastcats  #eastcoastcats  #caturday  #caturday  #luckyblackcats  #luckyblackcats  #catlove  #catlove  #rincat  #rincat  #catphotography  #catphotography  #tuxedocat  #tuxedocat  #Instacat  #Instacat  #Catlovers  #Catlovers  #itsacat  #itsacat  #iheartcats  #iheartcats  #CuteCat  #CuteCat  #nachocat  #nachocat  #rescuedcat  #rescuedcat
```

Figure 9: Sample Output

References

1. Keras Official Repository
https://github.com/keras-team/keras/blob/master/examples/cifar10_cnn.py

2. Very Deep Convolutional Network for Large-scale Image Recgonition
Karen Simonyan, Andrew Zisserman. 2015. International Conference on Learning Representations 2015

3. Deep Residual Learning for Image Recognition
Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

4. geifmany's GitHub Repository
https://github.com/geifmany/cifar-vgg