Flower Classification using CNN
MSBD5001 Group 11

LIU Yingjie *20711100*
XIE Chongshan *20755209*
HUANG Luyu *20721052*
GAO Wanqi *20733691*
WANG Ruoyu *20733823*

Abstract

In this project, we aim to tackle the problem of flower image classification. We learn to build tensorflow CNN models and explore multiple strategies including data augmentation, transfer learning, dropout layer and ensemble learning. We show their effectiveness in both independent and combined experimental results.

1 Background Introduction

Nowadays, there are over 5,000 species of mammals, 10,000 species of birds, 30,000 species of fish and astonishingly, over 400,000 different types of flowers. It is difficult to fathom just how vast and diverse our natural world is. When we find some beautiful flowers in the nature, we can hardly recognize the exact species or sub-species of these flowers.

Images can be represented as a matrix in the computer system. If we want to extract the features in the image, it will take lots of time because of the complex computation. However, with the development of the computers and computational techniques, it will be possible for us to make large scales of computation of hundreds and thousands of pixels in the matrix. Advanced machine learning techniques also makes it possible for us to deal with the problem only the botanists can solve.

2 Dataset

The dataset we chose is called Petals to the Metal: Flower Classification on TPU. It is a dataset from Kaggle competition. Link: https://www.kaggle.com/c/tpu-getting-started/overview

We need to classify 104 types of flowers based on their images drawn from five different public datasets. Some classes are very narrow, containing only a particular sub-type of flower (e.g. pink primroses) while other classes contain many sub-types (e.g. wild roses).

The dataset has three parts. training samples, pre-split training samples (also called validation sample) and testing samples (without labels). There will be 12753 training samples, 3712 validation samples and 7382 testing sample.

Figure 1 shows some pictures in the dataset.

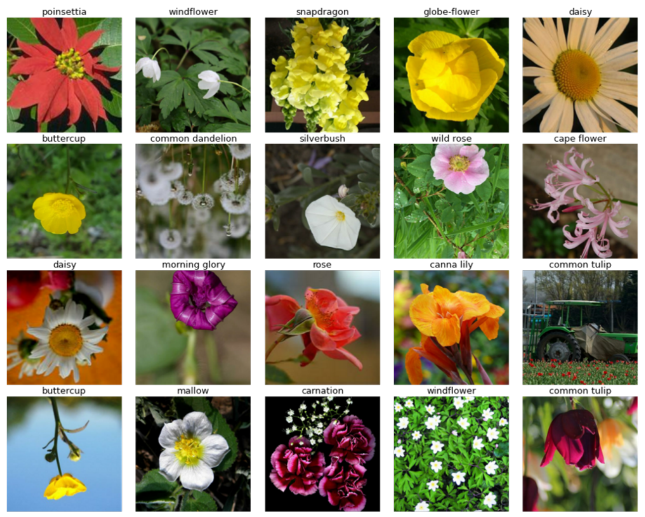

Figure 1: Example pictures

We also notice that the dataset is in .tfrec format, a binary file format which can be processed efficiently by Tensorflow, so we will learn and use Tensor Processing Units (TPUs) to deal with the data. It's a powerful hardware accelerator specialized in deep learning tasks provided by Google Colab, which can usually be used to process large image databases.

3 Data Preprocessing

Because the original data set is not large enough, we can not use enough data to get better training effect, and it is difficult to find new supplementary pictures. Therefore, we use data augment technology to expand the training dataset. However, in fact, not all data augment operations will achieve an ideal results. We have done a lot of experiments to enhance the effect though different combination.

3.1 Introduction of Data Augmentation

There are two common patterns of data augmentation: offline augmentation and online augmentation. The former is suitable for smaller data sets. We will eventually increase the dataset by a certain multiple, which is equal to the number of conversions made before. The latter is more suitable for larger datasets because it is difficult to afford the explosive increase in scale. In addition, programmers will do small batch conversion before feeding the model. Some machine learning frameworks support online enhancement, which can be accelerated on GPU.

Since the original data set is not very large, we can directly choose offline augmentation.

3.2 Different Data Augmentation methods

The methods we tried include random flipping, random chop and resize and random blockout.

We did these transformations independently on the original dataset, and fed the data to CNN model for training. Figure 2 shows the experimental results.

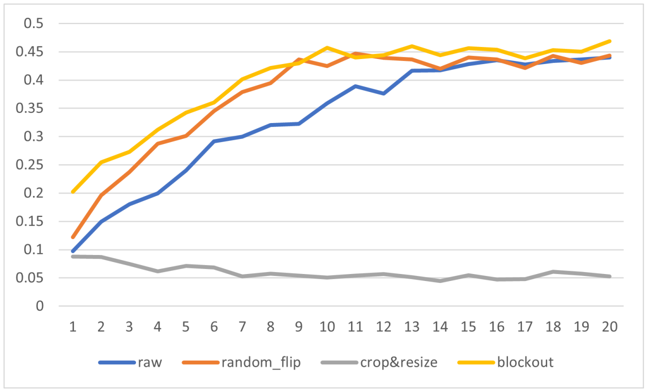

Figure 2: Data augmentation experiment

From these results, we conclude the following:

- Random block works best for our data.
- Random flipping introduces a small improvement in the beginning, but in the long run it doesn't do much.
- Random cropping & resizing makes the data basically untrainable.

We think that the main reason why cropping and resizing is harmful to the model is that the relations among neighbouring pixels are largely distorted by the resize operation, so the hidden features changed dramatically, which caused confusion for the model.

The intuition of using random blockout is that since most flower images have property of central symmetry, while random background objects, such as vehicle or people, don't have this property, we may preserve more information about flowers than other object when we block out a portion of the picture. Figure 3 shows some example pictures after random blockout.

Because color is an important feature of flowers, we did not use any image transformation functions that change the color of the image.

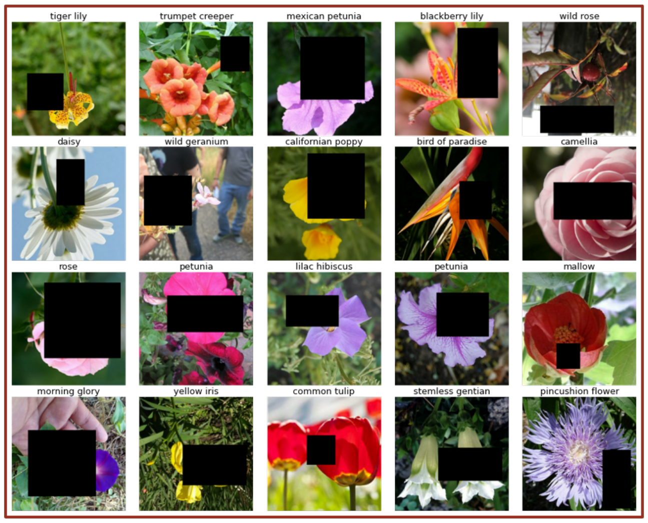

Figure 3: Random blockout examples

4 CNN model

CNN [1] can learn a large number of mapping relationships between inputs and outputs, and the result of its output is the specific feature space of each image. When processing the image classification task, we will use the output feature space as the input of the fully connected layer, and use the fully connected layer to complete the mapping from the input image to the label set to complete the classification task. In this project, we build a CNN model using layers provided by tensorflow.

4.1 Layers

Convolutional layer: The convolutional layer is the result of the convolution kernel calculated by sliding windows one by one on the upper input layer. Generally, a convolution kernel is used for feature extraction and feature mapping.

Pooling layer [2]: Pooling is to shrink the input image, which can reduce pixel information, only retain important information, and reduce the amount of data calculations. Generally, for example, the pooling area is 2*2 in size, and then converted into the corresponding value according to certain rules, and this value is used as the result pixel value.

The next layer in the traditional CNN model is the fully connected layer, but in our CNN model we use Global pooling layer, which converts a multi-dimensional output from previous layers to a one-dimensional output. Because the conversion between the feature map and the final classification is more simple and natural; on the other hand, unlike the FC layer that requires a lot of training and tuning parameters, Global pooling layer reducing the spatial parameters will make the model more robust.

Figure 4 shows a typical CNN model structure.

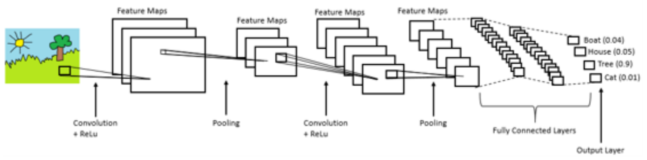

Figure 4: CNN model

4.2 Model tuning

We tried different configurations of CNN networks with various numbers of layers, filters, and two different activation functions.

Relu activation function will make the output of some neurons to be 0, which causes the sparsity of the network, and reduces the interdependence of parameters, and can also alleviate the occurrence of overfitting problems.

The Swish activation function is a new activation function proposed by Google in October 2017. The definition of Swish function is $f(x) = x \cdot sigmoid(\beta x)$, where β is a constant or trainable parameter.

Fig 5 and 6 shows the graphs of relu and swish.

The result of our model tuning experiment is shown in Table 1. We do 12 training epochs for each configuration and compared model accuracy. We concluded the best configuration which is indicated in the table by underline.

number of parameters/ network structure	16	32	64	128
2Conv+1Pool+relu	0.2985	0.2901	0.4278	0.4418
1Conv+1Pool+relu	0.2449	0.3454	0.3925	0.4825
2Conv+1Pool+swish	0.2578	0.3322	0.4230	**0.4968**
1Conv+1Pool+swish	0.2332	0.2955	0.4086	0.4434

Table 1: CNN model tuning result

5 Avoid Overfitting

In the model we use, if there are too many parameters and too few training samples of the model, the trained model is prone to overfitting. If overfitting occurs, it will show that the model has a smaller loss function on the training data and a higher

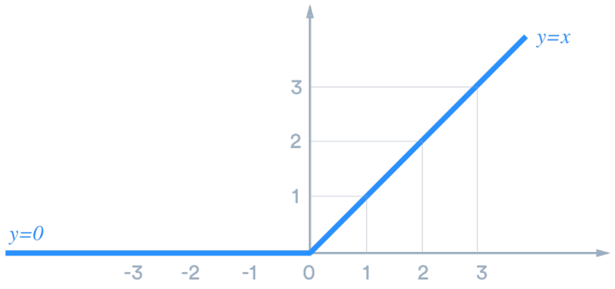

Figure 5: relu

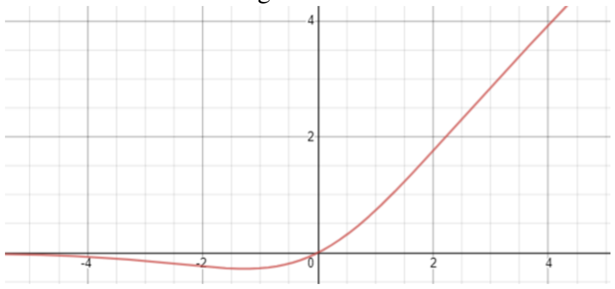

Figure 6: swish

model accuracy rate, but on the test data, the loss function is larger and the model accuracy rate is lower.

The Dropout layer [3] [4] can effectively alleviate the occurrence of over-fitting and achieve the effect of regularization to a certain extent. The dropout layer is to let the activation value of a certain neuron stop working with a certain probability, which can make the model more generalized because it will not rely too much on some local features.

It is worth noting that dropout layer is only effective during the training process, and is automatically disabled by Tensorflow during validation and testing.

6 Transfer learning

Transfer learning is a powerful tool for image classification and other machine learning tasks. We imported 3 different existing models pre-trained on ImageNet, send their output to a dense layer, and trained it to fit our dataset.

6.1 Xception

Xception [5] is an improvement of Inception v3, it uses depthwise separable convolution to replace the convolution operation in the original Inception v3. We remove avg pool layer from original Inception v3 and use only one 1×1 convolution kernel connecting 3×3 convolution layer. The depthwise separable convolution in Xception is similar to the 'extreme' version of Inception v3, but position of 1×1 convolution kernel and 3×3 convolution layer exchange.

6.2 VGG16

VGG16 [6] makes the improvement over AlexNet by replacing large kernel-sized filters with multiple 3×3 kernel-sized filters one after another. The idea is the increase in the number of network layers will not bring about an explosion in the amounts of parameters, because the amounts of parameters is mainly concentrated in the last three fully connected layers. So VGG16 builds the deep convolution neural network with 16 layers by repeatedly stacking 3×3 small convolution kernels and 2×2 MAX pooling layers, it increases the complexity of the model and reduces the training error.

6.3 DenseNet

DenseNet [7] connect all layers in the network in pairs can maximize the information flow between all layers in the network, so that each layer in the network accepts the features of all layers before it as input. Due to the large number of dense connections in the network, so it called DenseNet. DenseNet has two characteristic:1. To a certain extent, reduce the problem of gradient dissipation during training. 2. Since a large number of features are reused, a large number of features can be generated using a small number of convolution kernels, and the size of the final model is also relatively small. We use DenseNet201 in our experiment.

6.4 Experiments

After 12 epochs training for each model, DenseNet has the highest accuracy. However, we noticed that all the training results seem to be overfitting. So we add one dropout layer after each pre-trained model, which successfully decreased overfitting. The results are shown in Table 2.

	train	val	train_drop	val_drop
Xception	0.8849	0.833	0.888	0.8341
VGG16	0.9251	0.8397	0.92	0.851
DenseNet	0.9311	0.8718	0.9265	0.8728

Table 2: Transfer learning experimental results

7 Ensemble learning

Ensemble learning [8] is a technique of combining the prediction results of multiple week models to get an assembled model that may have more accuracy than any of the constituent model.

Traditionally, people usually do ensemble learning using models with similar internal structures. For example, Bayes optimal classifier and random forest algorithm are both successful models for prediction. When we do ensemble learning with similar models, each model may use only a portion of available features of the data, or only a portion of the data itself.

There is also another way called stacking. Stacking means we use several models that are ideally different in nature and use a combiner algorithm to get a final result from the results produced by each model.

In our project, we have built 4 models with different internal structures, so we choose the stacking approach, and use simple majority voting as the combiner algorithm, as shown in Figure 7.

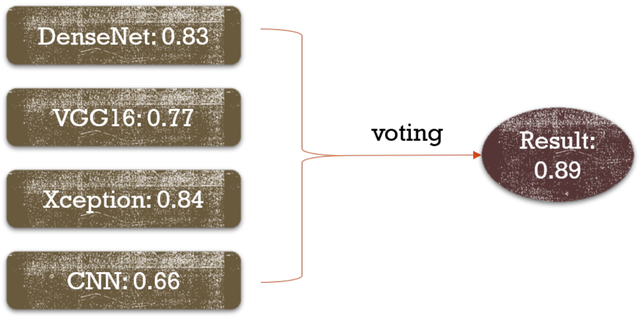

Figure 7: Ensemble learning

From our experiments we can see that the voted result is better than any single model participated in the voting.

8 Combined experiment

We combined all the results we obtained from previous experiments, and trained a final model. We got an accuracy of 89 percent, f1 score of 0.865 on the validation set. Figure 8 shows some of the classification examples, and Figure 9 shows the confusion matrix. We can see that most pictures are correctly classified.

9 Future work

There are two things we think we may improve, if time is abundant.

In this project, we rely on data augmentation to reduce the effect of background objects. If we use another classifier on a more general dataset, such as imagenet, we may be able to use that classifier to do a pre-classification to our dataset, and filter out the images that are recognized as objects other than flower.

Although 104 classes is not to the point of intractable, we may still try some techniques to pre-split the classes into coarse classes, and do a two round coarse-fine classification. One way of doing this is to use the confusion matrix as a distance matrix between the classes, and use a clustering algorithm to divide the 104 classes into clusters.

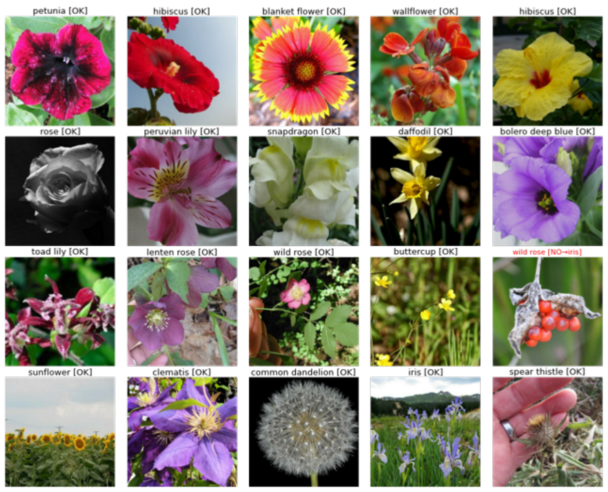

Figure 8: Classification examples

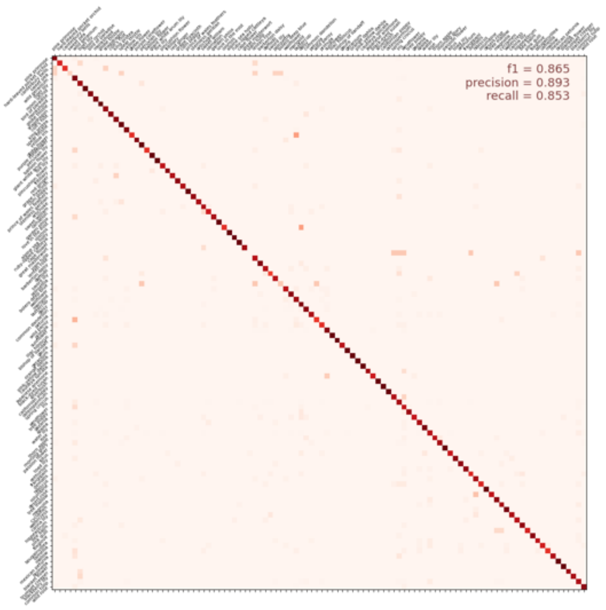

Figure 9: Confusion matrix

10 Acknowledgements

Contributions of each member are as follows:

Liu Yingjie: Planning and coordination for most of the experiments; built initial CNN model; introduced dropout layer for model experiments; implemented ensemble learning.

Xie Chongshan: Implemented basic dataset operation and training/validation output; introduced basic transfer learning programming guide.

Huang Luyu: Data augmentation experiments with multiple methods.

Gao Wanqi: Transfer learning experiments with multiple models and dropout.

Wang Ruoyu: CNN model tuning experiments with different parameters, activation functions and dropout.

References

[1] R. Venkatesan and B. Li, *Convolutional neural networks in visual computing: a concise guide*. CRC Press, 2017.

[2] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International conference on artificial neural networks*, pp. 92–101, Springer, 2010.

[3] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[5] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2017.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[7] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018.

[8] G. I. Webb and Z. Zheng, "Multistrategy ensemble learning: Reducing error by combining ensemble learning techniques," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 8, pp. 980–991, 2004.