Manga Classification

- Introduction

The task is quite simple, we just need to classify the manga using the provided manga pictures. However, "the devil is in the detail". This task is not as simple as other classification tasks. In this report I will describe the uniqueness of this dataset, and present my algorithm that achieved 87.56% on test set.

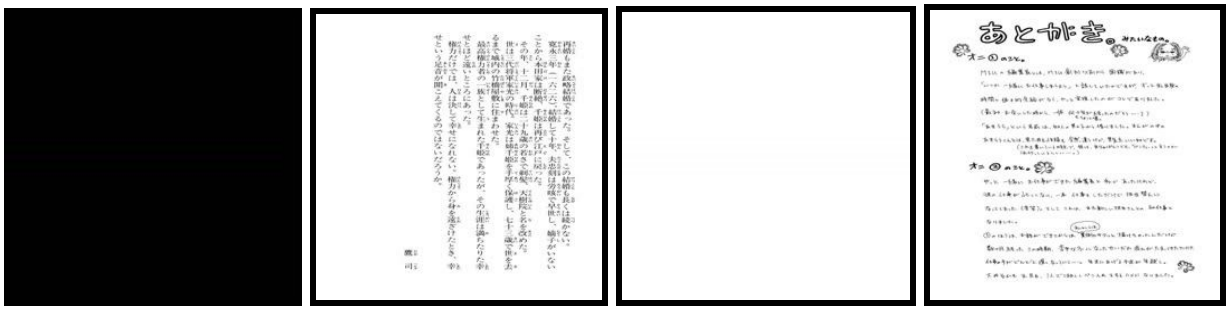
- Dataset

The dataset given for us consists of three folders, ‘train’, ‘valid’ and ‘test’. Training and validation folders have folders for each manga class. Thus, there are 109 folders in training and validation folders. This distribution of images in training and validation folders is very suitable for us. Because we can use ImageFolder [1] from PyTorch library to create dataset.

Training data consists of 12655 images, and validation data consists of 4249 images. Test data, which we should predict, consists of 4334 unlabeled images.

From the dataset we can observe that the classification of manga will be a very difficult task and there are several reasons for that. First reason is that image resolution given in dataset is very small (256,256), and each image contains a lot of information, 5-6 scenes in one image on average. Second reason is that there is no particular structure of manga, as it can be seen from figures 1-4, each manga has its own structure of representing scenes. Problem here is that classification algorithm may try to learn the structure of manga (like horizontal and vertical lines that separate each scene in image), which will result in overfitting, and result in bad generalization capacity. Next problem is that there is a noise in our dataset. For example, there are black and white images, images of manga content and any other images with text only, look to figures 5-8. Pure black and pure white images make it impossible to classify these images. In addition, about every manga has a black or white image. Actually, images with pure text can be used in classification, by identifying and analyzing the meaning of the text. However, we don’t have a lot of data with text, or some pre-trained model on the classification of Japanese text. Last problem is that we have highly undistributed data. For example, there are 222 images of ‘Hamlet’ manga and only 31 images of ‘YouchienBoueigumi’ manga, which makes training very difficult.

Figures 1-4. Sample images from a training data



Figures 5-8. Sample images from a training data that makes training very difficult

● Algorithm and Training

As this task is classification, I turned my attention to the state-of-the-art CNN architectures.I analyzed several papers to choose the best architecture. I considered four CNN architectures, VGGNet[2], DenseNet[3], ResNet[4], ResNext[5]. In all cases, except ResNext architecture, I used pre-trained networks on ImageNet database.

From some simple runs on small dataset I find out that the best performing architecture was ResNet. Then I compared ResNet with different numbers of layers. Results are presented in the next section.

Algorithm

---

1. Load training, validation and test data. For training data loader choose the size of the batch as 32.
2. Download pre-trained ResNet architecture.
3. Change the last fully connected network, from the 1000 units in the final layer to 109 units.
4. Freeze all layers except the fully connected layer.
5. Train the network for about 10 epochs.
6. Unfreeze all layers.
7. Train the network for about 100 epochs.
8. During training in step 7, after some iterations (or epochs) check accuracy on validation data, and if it is bigger than the previous validation accuracy, save the model, and update maximum validation accuracy. This is method is similar to early-stopping technique.
9. Load saved model, and make a prediction on training data.

This is the basic algorithm. However, there have been some tricks (dropout,data augmentation) that I applied for each model individually. They will be described soon.

The goal of training in step 5, is to fine tune weights of last layer with pre-trained feature weights. Since we assume that features already have some representation capacity. In step 5, I used only 10 epochs, because I observed that learning stops after about 10 epochs.

Please note that step 5 is only required if we use a pre-trained network. It makes no sense to do step 5 when we are training the neural network from scratch.

Optimization Technique

By reading some papers on the optimization techniques I find out that there are some equally good optimizers, like Adam[6], SGD, RMSprop[7]. For training in steps 5 and 7 I used Adam optimizer.

Learning Rate

I did not explore extensively about which learning rate to use. I found some learning rates from state-of-the-art object classification papers, and they worked well. For training in step 5 I used learning rate 1*exp(-3), and for training in step 7 I used learning rate 1*exp(-5). The reason behind that was that in the last layer we had not so much parameters to train in comparison to the whole network. In fact, the goal of training in step 5 is to fine tune last layer as much as possible, without caring about overshooting or any other problems related with big learning rates. However, the training in step 7 required us to be very accurate. Thus, I used smaller learning rate.

Dropout

In some models I used dropout technique proposed by Hinton et al [8]. Dropout is a technique that is used to deal with overfitting by preventing units in neural network co-adapting too much. It is key idea is to randomly drop units (along with their connections) from the neural network during training. We should not use dropout during evaluation phase, there are some rules on how to change weights in evaluation phase. However, PyTorch has train() [9] and eval() [10] functions. Thus, we do not really need to care about adjusting weights during evaluation phase.

Data Augmentation

Data Augmentation is another technique to address overfitting. This technique can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

I used 3 augmentation techniques from PyTorch library: RandomResizedCrop [11], RandomHorizontalFlip [12], RandomRotation [13]. I also used Normalize[14] with values from ImageNet dataset, because I used models pre-trained on normalized ImageNet images.

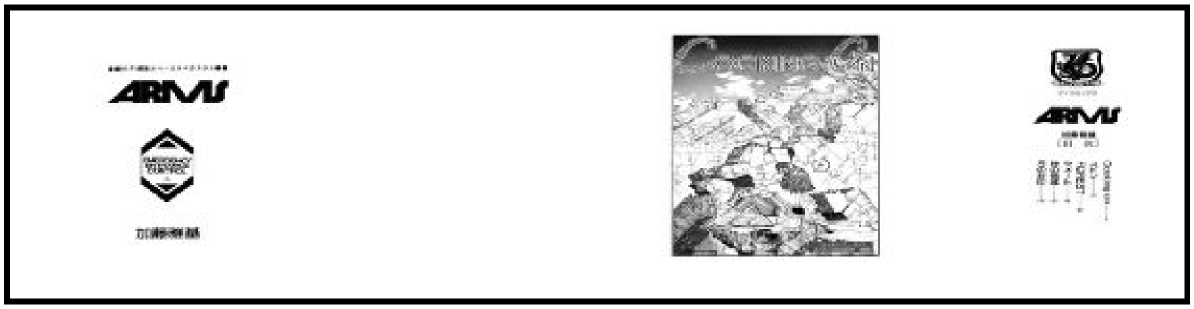


Figure 9. Images in one batch (batch size = 4, used here and in the figure below is used only for illustrative purposes, actual batch size is 32), without applying data augmentation

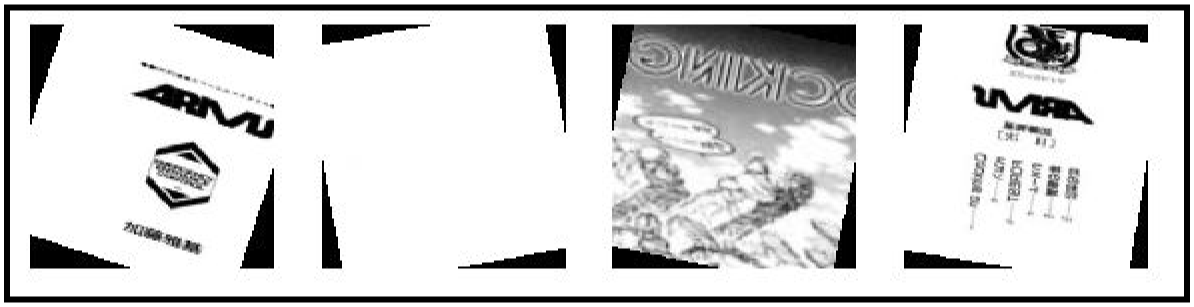


Figure 10. Images in one batch, with data augmentation.

- Results

a. Without dropout and data augmentation

First, I thought to use pure ResNet-18 and ResNet-50 without dropout and data augmentation.Results are shown in the table below.

Model	Training Accuracy	Validation Accuracy	Test Accuracy
ResNet-18	99.12%	70.2%	68.59%
ResNet-50	99.15%	70.0%	69.96%

Table 1. Accuracy when training without dropout and data augmentation

We can observe that models overfit dramatically.

b. With dropout and without data augmentation

I added dropout to the last layer of the network. To be clear, I applied dropout (with probability p = 0.5) before final layer. Results are shown in the table below.

Model	Training Accuracy	Validation Accuracy	Test Accuracy
ResNet-18	97.13%	75.2%	74.77%
ResNet-50	98.25%	76.3%	79.83%

Table 2. Accuracy when training with dropout and without data augmentation

We can observe that dropout increased validation and test accuracies in both models.

c. With dropout and with data augmentation

By using data augmentation we can see that test and validation accuracy increased dramatically. Results are shown in the table below.

Model	Training Accuracy	Validation Accuracy	Test Accuracy
ResNet-18	95.12%	79.99%	80.12%
ResNet-50	97.15%	88.87%	87.01%

Table 3. Accuracy when training with dropout and data augmentation

From the results given in the above tables, it can be clearly seen that the best performing model is ResNet-50 with dropout and data augmentation.

Finally, I merged training data and validation data, and trained network on this dataset. Afterwards, I made a prediction on this model, and achieved 87.56% test accuracy.

- ## Discussion

In Data section I mentioned about noise. Thus, I had an idea of deleting this noise from the dataset and I thought it will lead to the better generalisation. However, test and validation accuracies did not increase, on the contrary, they decreased. Thus, I think that noise served as some kind of regularizer.

In addition, I find out that using pre-trained network is very useful even if the domain of images is very different. When I used pre-trained network, validation accuracy after 10 epochs (in step 5) was 30%. However, without pre-trained network (there is no step 5 when we are not using pre-trained network, only step 7), 30% accuracy was achieved only after 100~120 epochs. Thus, we considerably reduced execution time and power consumption.

To sum up, I think that achieving 87.56% accuracy with a 'dirty' dataset is a very good result. Furthermore, I think that there is room for improvement and we can achieve an accuracy of more than 90%. There are vast amounts of techniques that I did not use due to the time and GPU constraints, which will lead to better performance, like ensemble models, adaptive learning rate, variational dropout, etc.

- ## References

[1] - ImageFolder - A generic data loader from PyTorch.
https://pytorch.org/docs/stable/torchvision/datasets.html#imagefolder
[2] - Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks For Large-Scale Image Recognition.
[3] - Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger. Densely Connected Convolutional Networks.
[4] - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition.
[5] - Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He. Aggregated Residual Transformations for Deep Neural Networks.
[6] - Diederik P. Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization.
[7] - Geoff Hinton. Lecture 6e of Geoff Hinton's Neural Networks Coursera Class.
[8] - Nitish Srivastava, Geoffrey Hinton hinton, Alex Krizhevsky, Ilya Sutskever Ilya, Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting.
[9] - train() - Sets the module in training mode.
https://pytorch.org/docs/stable/nn.html#torch.nn.Module.train
[10] - eval() - Sets the module in evaluation mode.
https://pytorch.org/docs/stable/nn.html?highlight=eval#torch.nn.Module.eval
[11] - RandomResizedCrop - Crop the given PIL Image to random size and aspect ratio.
https://pytorch.org/docs/stable/torchvision/transforms.html#torchvision.transforms.RandomResizedCrop
[12] - RandomHorizontalFlip - Horizontally flip the given PIL Image randomly with a given probability.

https://pytorch.org/docs/stable/torchvision/transforms.html#torchvision.transforms.RandomHorizontalFlip
[13] - RandomRotation - Rotate the image by angle.
https://pytorch.org/docs/stable/torchvision/transforms.html#torchvision.transforms.RandomRotation
[14] - Normalize - Normalize a tensor image with mean and standard deviation.
https://pytorch.org/docs/stable/torchvision/transforms.html#torchvision.transforms.Normalize