Multi-label Classification of Image Data

Xueyang Zhang - 260886286, Wen Xuan Wu - 260929786, Linhui Huang - 260831346

ABSTRACT

In this project, we developed a model to classify image data from a modified MNIST dataset[1], which contains images of 1 to 5 digits. A convolutional neural network (CNN) model trained by automatic differentiation was implemented for this multi-label classification task. Using the model, a 99.714% accuracy was achieved based on the test results from Kaggle.

INTRODUCTION

The dataset used in this project is a modified version of MNIST dataset[1] of 56000 images of size 64x64, each containing 1 to 5 digits. When classifying images, simply vectorizing the input cannot give optimal results because, in an image structure, pixels next to each other are related. To extract and train on 2D image-specific features, we will use a convolutional neural network (CNN) to classify the modified MNIST dataset. CNN's are comprised of convolutional layers, pooling layers, and fully-connected layers[2]. Convolutional layers learn local characteristics of different positions on the feature map padded from the former layer and apply an activation function to the result. Pooling layers extract secondary features and perform downsampling to reduce the dimensionality of the feature map. At last, fully-connected layers use softmax to produce class scores[2].

MODEL SELECTION

Since the convolutional filter will select the feature for us after training, which can be seen as a blur of the image, we do not need to Gaussian blur the image in advance like a convention in computer vision. The size of the convolution filter is usually 3x3. However, an image of size 64x64 seems small because some features may be omitted, especially given that the entire image possibly has 5 digits, with less than one third being useful pixels. Thus, we magnify the images by a factor of 2 into size 128x128 before training.

For individual digits, the test data may seem like a small deformation of the standard digits, composed of rescaling, rotation, and more importantly, shearing. Shifting is not important because using a convolutional filter does not consider shift at all. So we applied those transformations on the training data to let CNN learn and adapt to general characteristics (Appendix 2).

Our model has 5 groups of outputs, each representing a digit in the image, and trains on cross-entropy loss of validation data. The model architecture is inspired by VGG19, which contains convolutional layers using 3x3 filters, max-pooling layers to extract features, and fully connected layers and softmax layers to produce the output[3]. Our model also incorporates Batch Normalization to improve the learning speed. To prevent overfitting, the model performs Dropout, less on convolutional layers and more on dense fully-connected layers which are prone to overfit. To increase the robustness of the CNN model, Gaussian noise is also applied to the input, the output of convolutional layers, i.e. the input of fully-connected layers (Appendix 1).

RESULTS

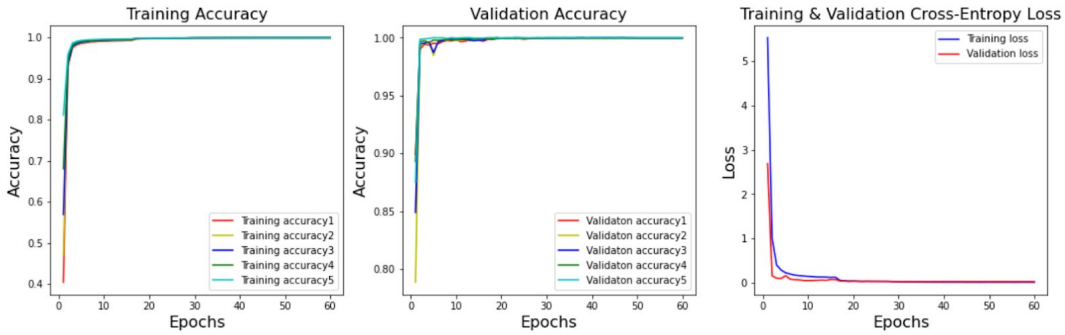

Figure 1: Left: change of training accuracies per epoch, training accuracy 1-5 each represents the accuracy of the 1st - 5th digit. Middle: change of validation accuracies per epoch, validation accuracy 1-5 each represents the accuracy of the 1st - 5th digit. Right: change of cross-entropy loss of training and validation sets per epoch.

As seen from the plots in Figure 1, the validation and training accuracies increase as training epochs increase, and the validation accuracies are always a bit better. The training and validation cross-entropy losses decrease monotonically as training epochs increase. Our model reduces the learning rate when the validation loss has not decreased for 6 epochs, and the early-stopping criteria were that the learning stops when the validation loss does not decrease for 10 epochs. Hence the best model has the lowest validation loss. The predictions by our model on the training set containing 14000 images achieved a public score of 99.714% on Kaggle.

Discussion

VGG is a deep convolutional neural network model architecture and was originally developed specifically for image classification. This is clearly seen through our results. Indeed, inspired by VGG19, our model has obtained a very high accuracy of 99.714%.

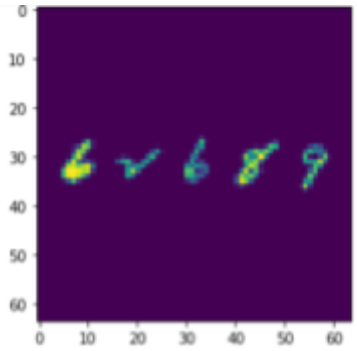

Figure 2: An example of the modified MNIST dataset.

It is very difficult to obtain much better than the current accuracy because as shown in Figure 2, the resolution of the images is not very high (each digit is only 12x12 pixels) and it seems like each of the digits has been drastically scaled down from their original sizes and thus some information has been lost. This can lead to overfitting to poorer observations.

We have also tried a secondary model where we first use an algorithm to partition the entire image into its 1 to 5 digits, i.e. the subcomponents of the image (Appendix 3), then use a similar neural network model also based on VGG to classify each digit individually, and then recombined the digits of the same image together. However, this method has not given us good results. This may be due to the fact that since there are many digits in the same image and if one digit is misclassified then the entire image would be misclassified and thus the errors can accumulate very fast. We have also tried bagging which votes with unequal weights for an average result from models trained on subsamples. This has also led to weaker results.

Some difficulties were encountered in this project. For example, due to the considerable lack of computing power, the learning process takes several hours. Also, the Keras API that we use does not provide much flexibility. Some future improvements that we can make include using more computing power and trying other architectures like ResNet which allows much deeper networks than VGG while keeping low complexity[4].

Conclusion

In conclusion, our CNN model based on VGG performed really well on classifying the modified MNIST dataset, although the performance was limited by the image resolution and computing power. For future research direction, we can use more datasets with images with higher resolutions so that the digits can be more easily recognizable and distinguishable. In addition, an even larger dataset (perhaps with a few million observations) would have given us even more reliable results. Moreover, perhaps we can use more computing power and higher batch size. Trying a ResNet model[4] would also be an interesting idea as it is one of the best models for analyzing images.

Attribution

Xueyang Zhang and Wen Xuan Wu worked on the code. Xueyang Zhang, Wen Xuan Wu, and Linhui Huang worked on the write-up.

Appendix

1. Model Architecture:

Layer (type)	Hyperparameter	Output Shape	Param #	Connected to
input_1 (InputLayer)	----	(None, 128, 128, 1)	0	----
batch_normalization (BatchNormalization)	----	(None, 128, 128, 1)	4	input_1[0][0]
dropout (Dropout)	rate = 0.1	(None, 128, 128, 1)	0	batch_normalization[0][0]
gaussian_noise (GaussianNoise)	stddev = 0.1	(None, 128, 128, 1)	0	dropout[0][0]
conv2d (Conv2D)	32, (3,3)	(None, 128, 128, 32)	320	gaussian_noise[0][0]
conv2d_1 (Conv2D)	32, (3,3)	(None, 128, 128, 32)	9248	conv2d[0][0]
max_pooling2d (MaxPooling2D)	pool size = (2,2)	(None, 64, 64, 32)	0	conv2d_1[0][0]
dropout_1 (Dropout)	rate = 0.25	(None, 64, 64, 32)	0	max_pooling2d[0][0]
batch_normalization_1 (BatchNormalization)	----	(None, 64, 64, 32)	128	dropout_1[0][0]
conv2d_2 (Conv2D)	64, (3,3)	(None, 64, 64, 64)	18496	batch_normalization_1[0][0]
conv2d_3 (Conv2D)	64, (3,3)	(None, 64, 64, 64)	36928	conv2d_2[0][0]
max_pooling2d_1 (MaxPooling2D)	pool size = (2,2)	(None, 32, 32, 64)	0	conv2d_3[0][0]
dropout_2 (Dropout)	rate = 0.25	(None, 32, 32, 64)	0	max_pooling2d_1[0][0]
batch_normalization_2 (BatchNormalization)	----	(None, 32, 32, 64)	256	dropout_2[0][0]
conv2d_4 (Conv2D)	128, (3,3)	(None, 32, 32, 128)	73856	batch_normalization_2[0][0]
conv2d_5 (Conv2D)	128, (3,3)	(None, 32, 32, 128)	147584	conv2d_4[0][0]
conv2d_6 (Conv2D)	128, (3,3)	(None, 32, 32, 128)	147584	conv2d_5[0][0]
conv2d_7 (Conv2D)	128, (3,3)	(None, 32, 32, 128)	147584	conv2d_6[0][0]
max_pooling2d_2 (MaxPooling2D)	pool size = (2,2)	(None, 16, 16, 128)	0	conv2d_7[0][0]
dropout_3 (Dropout)	rate = 0.25	(None, 16, 16, 128)	0	max_pooling2d_2[0][0]
batch_normalization_3 (BatchNormalization)	----	(None, 16, 16, 128)	512	dropout_3[0][0]
conv2d_8 (Conv2D)	256, (3,3)	(None, 16, 16, 256)	295168	batch_normalization_3[0][0]
conv2d_9 (Conv2D)	256, (3,3)	(None, 16, 16, 256)	590080	conv2d_8[0][0]
conv2d_10 (Conv2D)	256, (3,3)	(None, 16, 16, 256)	590080	conv2d_9[0][0]
conv2d_11 (Conv2D)	256, (3,3)	(None, 16, 16, 256)	590080	conv2d_10[0][0]
max_pooling2d_3 (MaxPooling2D)	pool size = (2,2)	(None, 8, 8, 256)	0	conv2d_11[0][0]
dropout_4 (Dropout)	rate = 0.25	(None, 8, 8, 256)	0	max_pooling2d_3[0][0]
gaussian_noise_1 (GaussianNoise)	stddev = 0.05	(None, 8, 8, 256)	0	dropout_4[0][0]
batch_normalization_4 (BatchNormalization)	----	(None, 8, 8, 256)	1024	gaussian_noise_1[0][0]

flatten (Flatten)	----	(None, 16384)	0	batch_normalization_4[0][0]
dense (Dense)	units = 512	(None, 512)	8389120	flatten[0][0]
dropout_5 (Dropout)	rate = 0.5	(None, 512)	0	dense[0][0]
dense_1 (Dense)	units = 512	(None, 512)	262656	dropout_5[0][0]
dropout_6 (Dropout)	rate = 0.5	(None, 512)	0	dense_1[0][0]
gaussian_noise_2 (BatchNormalization)	stddev = 0.05	(None, 512)	0	dropout_6[0][0]
digit_0 (Dense)	units = 11	(None, 11)	5643	gaussian_noise_2[0][0]
digit_1 (Dense)	units = 11	(None, 11)	5643	gaussian_noise_2[0][0]
digit_2 (Dense)	units = 11	(None, 11)	5643	gaussian_noise_2[0][0]
digit_3 (Dense)	units = 11	(None, 11)	5643	gaussian_noise_2[0][0]
digit_4 (Dense)	units = 11	(None, 11)	5643	gaussian_noise_2[0][0]
Total params: 11,328,923 Trainable params: 11,327,961 Non-trainable params: 962				

2. Other hyper-parameters:

Hyper-parameter	Batch Size	Epochs	ReduceLROnPlateau Factor	Minimum Delta	Patience
Value	128	60	0.3	0.00001	6
Hyper-parameter	rotation_range	shear_range	zoom_range	Optimizer	
Value	10	0.1	0.1	ADAM	

3. An algorithm used to partition each input image into its 1 to 5 component digits:

   For each image:
   a. Remove the top 24 rows of pixels and the bottom 24 rows of pixels.
   b. Scan the pixels in each column from left to right
      i. If we find at least one pixel that is non-zero, this means we have found a new digit.
         1. Then save all the columns until we get to a column with all zero pixels again (or less than a certain threshold).
         2. Continue to scan for more digits until we have reached the end of the image or we have found 5 digits.

Bibliography

[1] Modified MNIST Dataset. MNIST_synthetic.h5.
https://drive.google.com/file/d/1LcKqf1d7bctw5lx0YZf31kCUF0zEYOsi/view?usp=sharing

[2] T. Guo, J. Dong, H. Li and Y. Gao, "Simple convolutional neural network on image classification," *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, Beijing, 2017, pp. 721-724, doi: 10.1109/ICBDA.2017.8078730.

[3] Simonyan, K., & Zisserman, A. (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. ICLR 2015. https://arxiv.org/abs/1409.1556

[4] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition.
https://arxiv.org/abs/1512.03385