The source code is at the end of this document

1 Classification: Convolutional Neural Networks

1.1 Design and Implementation Choices of your Model

Inspired by VGG and ResNet, almost all convolution layers are 3×3 and almost all pooling layers are 2×2.

The following CNN models are tested.

Model 1

CNN1 is a sample CNN with only one convolutional layer. Dropout layer is used, which is the easiest way to prevent overfitting.

Model 2

CNN2 is similar to the network in the link given in the assignment. It has four convolutional layers, each followed by a batch normalization layer, which outputs the previous layer by substracting the batch mean and dividing it by batch standard deviation. Batch normalizatoin could speed up training and increase the stability of the network.

Model 3

CNN3 is just CNN2 with an extra block(a Conv2D, a Batch normalization and a Dropout layer), because I wanted to check if a deeper network could do a better job.

Model 4

CNN4 is a VGG like network (two or three Convolutional layers followed by a pooling layer) given in this link. Batch normalizatoin layer was also applied to speed up computation.

VGG19

VGG19 with imagenet pretrained weight. VGG19 contains 16 convolutional layers and 3 fully connected layers. VGG has a relatively simple network structure and it performs well in the image classification. Before using this model, I had to transform the input data into desired form. The origin input format of VGG19 is 224×224×3, and the minimum input dimension for keras VGG19 model is 32×32×3. I stacked the data three times and reshaped it to 150×150, because a too small input would affect the performance of the network and 150×150 is almost the largest size I could get within my computer's capability.

Its performance was really bad(accuracy 65.53%) so I gave it up. Maybe because our labels were twisted and it was not a conventional image classification problem so we should not use the weights directly.

Inception V3

Inception network stacks the result of multiple filters together, therefore, the network can learn small details, middle sized features and almost whole images at the same time. So it will usually achieve a good result.

Since using the pretrained weight directly was not a good idea, I tried transfer learning on Inception V3. We had to adjust the input format to 150×150×3 like VGG19, and last 10 layers of the model were retrained.

The performance was bad as well(accuracy 40.03%), so I did not try any further improvements.

Inception

The Inception V3 model is too big for us to train from scratch. Following this blog, I built a smaller Inception network. But its accuracy was only 81.12%.

Resnet50

When the network becomes deeper and deeper, the model is prone to overfitting the data. Also, when performing the back-propagation, the gradient may becomes infinitively small or really big, making it harder to make progress. By skipping one or more layers, that is connecting one layer directly to a layer several depth apart, the model becomes more robust. This shortcut connection makes it possible for us to build deeper networks.

The Resnet50 used here was implemented based on Coursera CNN week 2 programming assignments.

Data Augmentation

ImageDataGenerator was used to generate transformed images for neural networks. Each image was zoomed, rotated, flipped or sheared and then fed to the network. But it did not perform well as expected. The validation accuracy decreased gradually from around 90 percent to 82 percent as more and more data being generated. Hence, data augmentation was not used later.

Hyperparameters, regularization and optimizers will be discussed later.

1.2 Implementation of your Design Choices

1.3 Kaggle Competition Score

1.4 Results Analysis

1.4.1 Runtime performance for training and testing

1.4.2 Comparison of the different algorithms and parameters you tried

Model	**Accuracy**
1(3 layers)	79.10%-91.36%
2(7 layers)	85.68%-90.15%
3(8 layers)	87.42%-90.47%
4(7 layers)	89.68%-90.15%
Inception	81.12%
Resnet50	88.77%-89.56%

As we can see from the table, generally, the deeper a network, the more stable its performance. Model 1 only has 3 layers and its accuracy on validation set fluctuate a lot. 79.10% and 91.36% are both very rare cases, they occur only once or twice in hundreds of runs. The accuracy is about 88% on average. Also, we cannot say that model 1 is better than others because its best performance is 91.36%. Because it is the easiest to train(it took only 3s to train a batch with batchsize = 128), I could train it hundreds of times but for other models, it takes tens of minutes to hours to train. So I could only train them a few times. Model 2 and 4 are both 7 layers but model 4 is more stable than model 2. This may due to its VGG like structure.

The performance of Inception and Resnet50 are disappointing. It took 10 times longer to train

them but the accuracy is worse than a simple VGG like model. A possible explanation for this is that the training set is not large enough. For the original Inception and Resnet, they are trained by millions of 224×224 colored images. However, we only have 60,000 28×28 grey scale images. The lack of input information made it difficult to build an accurate model.

1.4.3 Explanation of your model

The structure of each model have been shown in section 1.1. This section will mainly focus on hyperparameters, regularization and optimizers.

ReduceLROnPlateau was used as callback function on every model to reduce learning rate when the accuracy has stopped improving.

Only dropout layer as regularisation was used because it is the easiest one.

The epochs was set as 40 initially and later I would check if there was sign of overfitting. If there was, the number would be decreased.

The batch size was set as 128 or 256. These two numbers led to similar performance.

Activation function: softmax for output layer and ReLU for others.

The following hyperparameters are tuned using hyperas and hyperopt following this guide:

- Dropout rate: a random number between 0 and 1.
- Number of units in a Dense layer: 128, 256 or 512.
- Optimizer: Adam, SGD or RMSProp, each with learning rate of 0.001, 0.01 or 0.1.

However, I found hyperas not so efficient and sometimes it generated a set of parameters that led to a really mediocre result after 30-50 evaluation.

1.4.4 Plots

1.4.5 Evaluate your code with other metrics on the training data and argue for the benefit of you approach.