COMPUTER VISION - PROJECT 2

Experiment with convolutional Neural Networks in TensorFlow

Dataset:

- 60,000 training example, 10,000 testing examples
- In this project we consider only 10% for training data to fit the model (6,000 training examples)
- The dimensions of the input image has been reduced from 28x28 to 7x7 by using a maxpool(4x4) layer as the first layer of the CNN

Ways that cannot be Explored:

- Cannot increase filter size from 3x3
    - The input image is only 7x7, so cannot use filters like 5x5 and 7x7 on it
- Cannot do strided convolution
    - As we already have very less data. We cannot afford to lose it. Moreover, these methods are used to reduce the computational complexity while working with high resolution Images
- Padding is used as ‘same’ and not ‘valid’ to not lose information during conv operations

How to run the code:

- The program takes 2 arguments : train_x file name and train_y file name
- Train files should be in 'small' folder
- Test files should be in 'big' folder

Example : python proj2.py "train_x_1_10.csv" "train_y_1_10.csv"

Basic structure of the model & training parameters:

We will introduce variations to the basic structure, and we will add then to our model it improves the testing accuracy. This process will be repeated multiple times to find the best CNN architecture for the given data

Input → maxpool(4x4) → conv(3x3) → conv(3x3) → flatten → Dense → Dense → Output

Code:

```
model = keras.Sequential([
    layers.MaxPool2D(4, 4, input_shape=(28,28,1)),
    layers.Conv2D(7, (3,3), padding='same', activation='relu'),
    layers.Conv2D(16, (3,3), padding='same', activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')])
```

Training Parameters:

- optimizer = Adam
- loss = sparse_categorical_crossentropy
- No of epochs = 10

Preprocessing:

- Converted data type from unit8 to float32 for faster computation
- Normalized range of input values from 0-255 to 0-1 (also reduces computational cost)

Data:

Exploring with data generated by : python3 fraction_xy.py big/train_x.csv big/train_y.csv 0.1 1

Finalized model will then be tested for different subsets of the training data

Exploration:

1. No of filters in conv layers:

# layers in conv1	# layers in conv2	Train Acc	Test Acc
7	16	0.7970	0.7742
16	32	**0.8201**	**0.7833**
32	64	0.8376	0.7879

Choosing : 16, 32 (No big difference in testing accuracy after this)

As the number of filters increases the gap between the train Accuracy and test Accuracy increases. The model is overfitting.

2. L2 regularization along with conv

Lambda Value	Train Acc	Test Acc
0.02	0.7715	0.7444
0.01 (Default)	**0.7843**	**0.7630**
0.005	0.7954	0.7699
0.001	0.8207	0.7835

Choosing : 0.01

(Increasing lambda beyond this value only reduces both the accuracies, but not the gap between them)

3. No of filters in Dense layers:

   No of units in the final dense layer is always No of output categories (10, in this case)

# units in dense1	Train Acc	Test Acc
64	0.7843	0.7630
128	0.8024	0.7719
256	**0.8145**	**0.7827**
512	0.8241	0.7830

Choosing : 256 (No big difference in testing accuracy after this)

4. L2 regularization along with Dense

Lambda Value	Train Acc	Test Acc
0.02	0.7174	0.6957
0.01 (Default)	0.7301	0.7169
0.005	0.7485	0.7357
0.001	**0.7798**	**0.7589**
0.0005	0.7915	0.7673

Choosing : 0.001 (reduce reasonable gap between the accuracies without lot of drop in Test Acc)

5. Introducing BatchNormalization() after each conv layer

	Train Acc	Test Acc
Before Batchnorm	0.7798	0.7589
After Batchnorm	**0.8991**	**0.7772**

Choosing : Include batch norm after each conv layer

This improved Test accuracy but also causes overfitting. Need to introduce Dropout regularization

6. Adding Maxpool(2x2) before flatten()

   Not used after each conv layer. As the input size is already 7x7, doing 2x2 maxpooling will reduce it to 3x3. we can use it only once.

	Train Acc	Test Acc
Before Maxpool	**0.8991**	**0.7772**
After Maxpool	0.8847	0.7771

Choosing : not including maxpool, no improvement

7. Dropout Regularization after flatten()

Dropout prob	Train Acc	Test Acc
No Dropout	0.8991	0.7772
0.2	0.8752	0.7785
0.3	0.8743	0.7812
0.5	**0.8499**	**0.7858**
0.6	0.8167	0.7781

Choosing : 0.5

8. Dropout regularization after Dense1

Dropout prob	Train Acc	Test Acc
No Dropout	0.8499	0.7858
0.2	**0.8216**	**0.7915**
0.3	0.8062	0.7769

Choosing : 0.2

9. Explore optimizers

Optimizers	Train Acc	Test Acc
Adam	**0.8216**	**0.7915**
RMSprop	0.8037	0.7702
SGD	0.7549	0.7581
Adamax	0.7995	0.7861
Adagrad	0.6973	0.7362

10. Adding more dense layers

Adding more dense layers with or without regularization did not improve the model accuracy

11. Exploring activation functions:

ReLU id better than tanh and Leaky ReLU

12. Exploring for different subset of training data

The final model is trained using different small datasets created using seed values from 0 – 9 in the program : fraction_xy.py

Dataset	Train Acc	Test Acc
train_x_0_10.csv	0.8045	0.7803
train_x_1_10.csv	0.8216	0.7915
train_x_2_10.csv	0.8171	0.7673
train_x_3_10.csv	0.8123	0.7725
train_x_4_10.csv	0.8025	0.7828
train_x_5_10.csv	0.8074	0.7740
train_x_6_10.csv	0.8129	0.7690
train_x_7_10.csv	0.8010	0.7659
train_x_8_10.csv	0.8146	0.7796
train_x_9_10.csv	0.7969	0.7743

Train Acc Range = 0.7969 - 0.8216
Test Acc Range = 0.7659 - 0.7915