Implement hand digit recognition in Keras using CNN

Sumbited by :- Jaypratap singh Chauhan

Organization :- AI TECH SYSTEM

Email id :- Jayjpsc125@gmail.com

[1] Introduction

This is a 5 layers Sequential Convolutional Neural Network for digits recognition trained on MNIST dataset. I choosed to build it with keras API (Tensorflow backend) which is very intuitive. Firstly, I will prepare the data (handwritten digits images) then focus on CNN.

I achieved 99.12% of accuracy with this CNN trained using GPU computing.

```
Epoch 15/15
 - 17s - loss: 0.0264 - acc: 0.9920 - val_loss: 0.0295 - val_acc: 0.9912
```

[2] Data preparation

A. Load data

```
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

B. Check for null and missing values

```
X_train.isnull().any().describe()
test.isnull().any().describe()
```

C. Normalization

```
X_train = X_train / 255.0
test = test / 255.0
```

D. Reshape

```
X_train = X_train.values.reshape(-1,28,28,1)
test = test.values.reshape(-1,28,28,1)
```

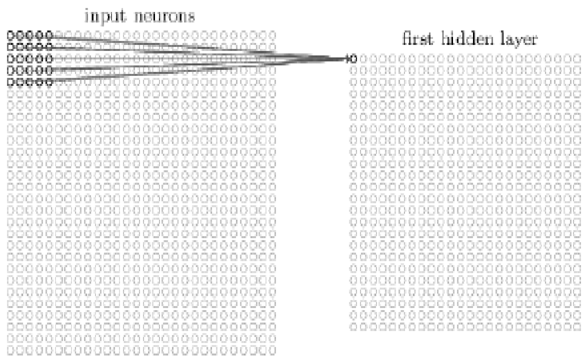
E. Label encoding

```
Y_train = to_categorical(Y_train, num_classes = 10)
```

F. Split training and validation set

```
random_seed = 2
X_train, X_val, Y_train, Y_val = train_test_split
(X_train, Y_train, test_size = 0.1, random_state=random_seed)
```

[3] CNN



I used the Keras Sequential API, where you have just to add one layer at a time, starting from the input.

The first is the convolutional (Conv2D) layer. It is like a set of learnable filters. I choosed to set 32 filters for the two firsts conv2D layers and 64 filters for the two last ones. Each filter transforms a part of the image (defined by the kernel size) using the kernel filter. The kernel filter matrix is applied on the whole image. Filters can be seen as a transformation of the image.

The CNN can isolate features that are useful everywhere from these transformed images (feature maps).

The second important layer in CNN is the pooling (MaxPool2D) layer. This layer simply acts as a downsampling filter. It looks at the 2 neighboring pixels and picks the maximal value. These are used to reduce computational cost, and to some extent also reduce overfitting. We have to choose the pooling size (i.e the area size pooled each time) more the pooling dimension is high, more the downsampling is important.

Combining convolutional and pooling layers, CNN are able to combine local features and learn more global features of the image.

Dropout is a regularization method, where a proportion of nodes in the layer are randomly ignored (setting their wieghts to zero) for each training sample. This drops randomly a propotion of the network and forces the network to learn features in a distributed way. This technique also improves generalization and reduces the overfitting.

'relu' is the rectifier (activation function max(0,x). The rectifier activation function is used to add non linearity to the network.

The Flatten layer is use to convert the final feature maps into a one single 1D vector. This flattening step is needed so that you can make use of fully connected layers after some convolutional/maxpool layers. It combines all the found local features of the previous convolutional layers.

In the end i used the features in two fully-connected (Dense) layers which is just artificial an neural networks (ANN) classifier. In the last layer(Dense(10,activation="softmax")) the net outputs distribution of probability of each class.

[4] Setting the optimizer and annealer

Once our layers are added to the model, we need to set up a score function, a loss function and an optimisation algorithm.

```
optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
```

```
model.compile(optimizer=optimizer,loss="categorical_crossentropy",metrics=["accuracy"])
```

We define the loss function to measure how poorly our model performs on images with known labels. It is the error rate between the oberved labels and the predicted ones. We use a specific form for categorical classifications (>2 classes) called the "categorical_crossentropy".

The most important function is the optimizer. This function will iteratively improve parameters (filters kernel values, weights and bias of neurons ...) in order to minimise the loss.

I choosed RMSprop (with default values), it is a very effective optimizer. The RMSProp update adjusts the Adagrad method in a very simple way in an attempt to reduce its aggressive, monotonically decreasing learning rate. We could also have used Stochastic Gradient Descent ('sgd') optimizer, but it is slower than RMSprop.

The metric function "accuracy" is used is to evaluate the performance our model. This metric function is similar to the loss function, except that the results from the metric evaluation are not used when training the model (only for evaluation).

In order to make the optimizer converge faster and closest to the global minimum of the loss function, i used an annealing method of the learning rate (LR).

The LR is the step by which the optimizer walks through the 'loss landscape'. The higher LR, the bigger are the steps and the quicker is the convergence. However the sampling is very poor with an high LR and the optimizer could probably fall into a local minima.

Its better to have a decreasing learning rate during the training to reach efficiently the global minimum of the loss function.

To keep the advantage of the fast computation time with a high LR, i decreased the LR dynamically every X steps (epochs) depending if it is necessary (when accuracy is not improved).

With the ReduceLROnPlateau function from Keras.callbacks, i choose to reduce the LR by half if the accuracy is not improved after 3 epochs.

And last is the learning process, I used epochs 15 and batch size 50 to get a good accuracy result.

```
learning_rate_reduction=ReduceLROnPlateau(monitor='val_acc',
                                          patience=3,
                                          verbose=1,
                                          factor=0.5,
                                          min_lr=0.00001)
```

```
epochs = 15
batch_size = 50
history=model.fit(X_train,Y_train,batch_size=batch_size,epochs=epochs,
          validation_data = (X_val, Y_val), verbose = 2)
```

There are other process to make the learning process more better like data augmentation which further increasing the efficiency and accuracy of the model and clear the errors with more margin.

[5] REFERENCES

[1] Yann LeCun, Leon Bottou,Yoshua Bengio and Patrick Haffner , paper on “Gradient Based Learning Applied to Document Recognition” , Proc of the IEEE , NOVEMBER 1998

[2]Y.LeCun,L.Jackel,L.bottom,A.brunot,C.Cortes,J .Denker,H.Drucker,I.guyon,U.muller paper on “Comparision of Learning Algorithms for handwritten digit recognition”