

CONVOLUTIONAL NEURAL NETWORK FOR MNIST DATASET

presented by :

1. SHEFALI VERMA (IIT2015031)

INTRODUCTION

In this project, i have made a convolutional neural network which will recognise images from MNIST dataset.

- **WHAT IS COVOLUTION?**

A convolution in CNN is nothing but a element wise multiplication i.e. dot product of the image matrix and the filter. A convolution operation takes place between the image and the filter and the convolved feature is generated. Each filter in a CNN, learns different characteristic of an image.

DATASET

MNIST dataset is a very popular dataset for images of handwritten digits from 0-9.

MNIST is one of the most studied datasets for this purpose.

It has 60,000 grayscale images under the training set and 10,000 grayscale images under the test set.

ARCHITECTURE

➔ We will use the Keras library with Tensorflow backend to classify the images.

- LIBRARIES REQUIRED
 - 1) KERAS API
 - 2) NUMPY
 - 3) TENSORFLOW

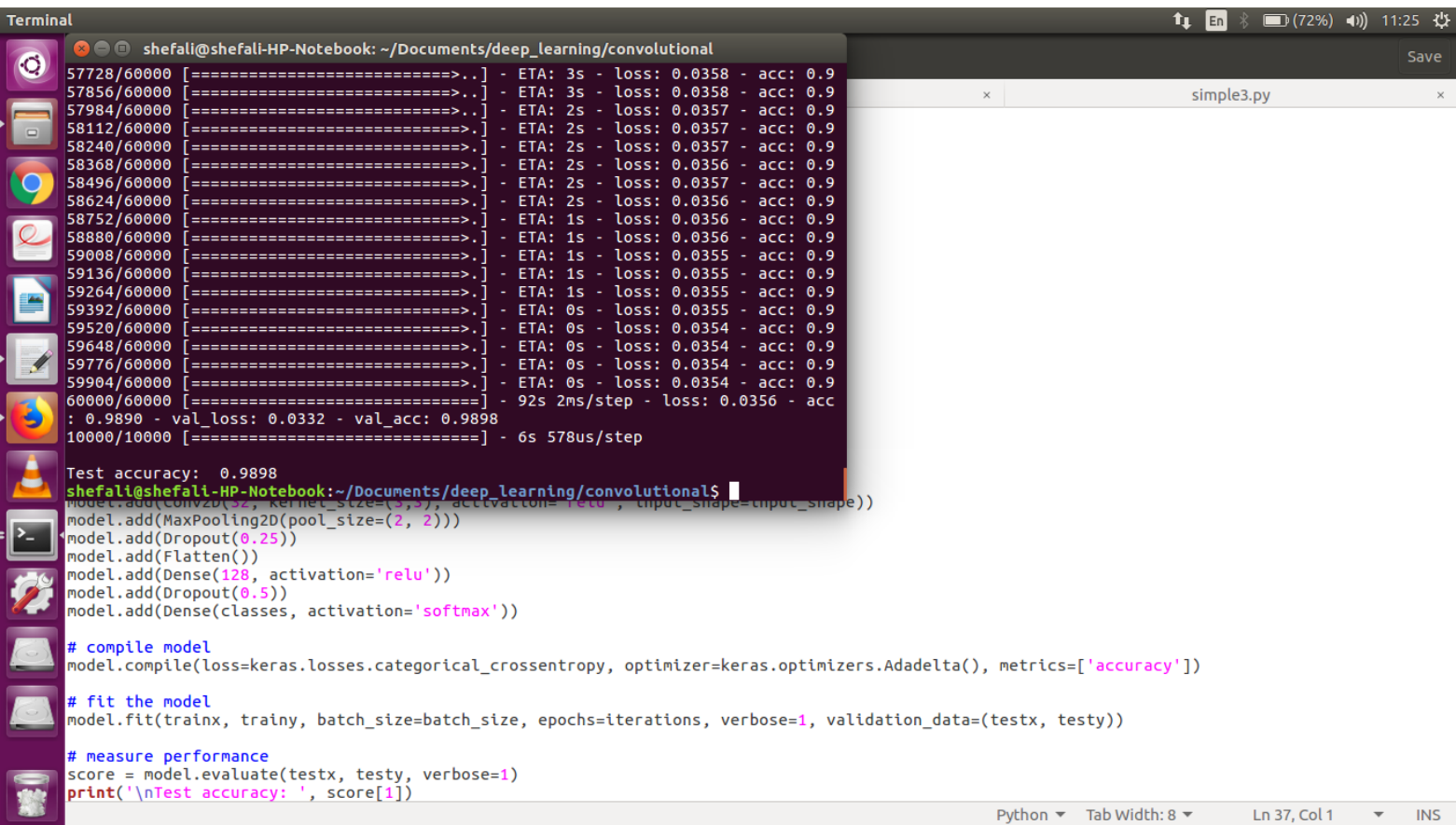
IMPLEMENTATION

- KERAS will provide us MNIST dataset as a mapping of an image with its corresponding digit. We will first load the data in a training set and a testing set.
- We get images in the format of <batchsize,height,width> ,and hence we have to change the format to <batchsize,height,width,no_of_channels> ,because this is the format in tensorflow . No of channels in greyscale images is 1,while in RGB images it is 3. Hence we will pass 1 in no channels as we are using grayscale images.
- We will reshape the data by diving values by 255 ,so that we get images in 0-1 range in place of 0-255 ,and we will transform the corresponding number(of every image) to its one hot-encoding version for the ease of training of data.
- KERAS allows us to use any no of filters with filter size of your choice .We have to give the size of input image ie.28*28 for the first layer. Then for activation ,we chosen ReLu (Rectified Linear Unit) since it reduces training time and does not creaye the problem of vanishing gradients. Relu function is $f(x)=\max(0,x)$.
- We used max-pooling layer since it is better than average pooling. We used pooling size of 2*2. Pooling is better because model will make some assumptions about features and hence avoid overfitting.Also it reduces no of parameters to learn,reduces dimensionality and hence decreases the training time.
- After one layer,we need to do batch normalisation,so that the scale of each dimension remains the same. It results in reduction of training time.
- Now,we will make the data pass through as many no of layers as we want. Afterwards,we need to make the multi-dimensional data to a single-dimensioned vector for feeding it to a fully-connected layer. DENSE layers are the keras's version of fully-connected layers. We will pass data through Dense layer since it will classify the features learned in training.
- Then to avoid over-fitting,we will have dropout of 20% of neurons.Therefore,we will perform dropout and pass 0.2 in argument for 20% dropout.
- Second last Dense layer has to make classes in the range of 0-9,and hence we will pass 10 in the argument for 10 classes.
- In the last layer ,SOFTMAX activation layer,a value of probability for all different 10 classes will be generated,and the class with maximum value of probabability will be generated. And,this will be the final output.
- Now we will compile our model ,i.e., update weights using adadelata,a SGD ,using backpropogation.
- Then we evaluate and test our model.

EXPERIMENTS

1. EXPERIMENT NO 1 : batchsize :128
epochs :24
dropout : .25 and .50
pooling size :2,2

We get accuracy of 98%



```
Terminal
shefali@shefali-HP-Notebook: ~/Documents/deep_learning/convolutional

57728/60000 [=====>..] - ETA: 3s - loss: 0.0358 - acc: 0.9
57856/60000 [=====>..] - ETA: 3s - loss: 0.0358 - acc: 0.9
57984/60000 [=====>..] - ETA: 2s - loss: 0.0357 - acc: 0.9
58112/60000 [=====>..] - ETA: 2s - loss: 0.0357 - acc: 0.9
58240/60000 [=====>..] - ETA: 2s - loss: 0.0357 - acc: 0.9
58368/60000 [=====>..] - ETA: 2s - loss: 0.0356 - acc: 0.9
58496/60000 [=====>..] - ETA: 2s - loss: 0.0357 - acc: 0.9
58624/60000 [=====>..] - ETA: 2s - loss: 0.0356 - acc: 0.9
58752/60000 [=====>..] - ETA: 1s - loss: 0.0356 - acc: 0.9
58880/60000 [=====>..] - ETA: 1s - loss: 0.0356 - acc: 0.9
59008/60000 [=====>..] - ETA: 1s - loss: 0.0355 - acc: 0.9
59136/60000 [=====>..] - ETA: 1s - loss: 0.0355 - acc: 0.9
59264/60000 [=====>..] - ETA: 1s - loss: 0.0355 - acc: 0.9
59392/60000 [=====>..] - ETA: 0s - loss: 0.0355 - acc: 0.9
59520/60000 [=====>..] - ETA: 0s - loss: 0.0354 - acc: 0.9
59648/60000 [=====>..] - ETA: 0s - loss: 0.0354 - acc: 0.9
59776/60000 [=====>..] - ETA: 0s - loss: 0.0354 - acc: 0.9
59904/60000 [=====>..] - ETA: 0s - loss: 0.0354 - acc: 0.9
60000/60000 [=====] - 92s 2ms/step - loss: 0.0356 - acc
: 0.9890 - val_loss: 0.0332 - val_acc: 0.9898
10000/10000 [=====] - 6s 578us/step

Test accuracy: 0.9898
shefali@shefali-HP-Notebook:~/Documents/deep_learning/convolutionals
model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(classes, activation='softmax'))

# compile model
model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])

# fit the model
model.fit(trainx, trainy, batch_size=batch_size, epochs=iterations, verbose=1, validation_data=(testx, testy))

# measure performance
score = model.evaluate(testx, testy, verbose=1)
print('\nTest accuracy: ', score[1])
```

2. EXPERIMENT NO 2 : batchsize :64
epochs :10
dropout :.30 and .45
pooling size :2,3

We get accuracy of 97.76

```
Terminal
Open  Save
ingx, ingy = 28, 28
if kr.image_data_format() == 'channels_first':
    trainx = trainx.reshape(trainx.shape[0], 1, ingy, ingx)
    testx = testx.reshape(testx.shape[0], 1, ingy, ingx)
    input_shape = (1, ingy, ingy)
else:
    trainx = trainx.reshape(trainx.shape[0], ingy, ingy, 1)
    testx = testx.reshape(testx.shape[0], ingy, ingy, 1)
    input_shape = (ingx, ingy, 1)

# preprocessing
trainx = trainx.astype('float32')
testx = testx.astype('float32')
trainx /= 255
testx /= 255

# setup class matrices
trainy = keras.utils.to_categorical(trainy, classes)
testy = keras.utils.to_categorical(testy, classes)

# setup the model, add layers
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.30))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.45))
model.add(Dense(classes, activation='softmax'))

# compile model
model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])

# fit the model
model.fit(trainx, trainy, batch_size=batch_size, epochs=iterations, verbose=1, validation_data=(testx, testy))

# measure performance
score = model.evaluate(testx, testy, verbose=1)
print('\nTest accuracy: ', score[1])

shefali@shefali-HP-Notebook: ~/Documents/deep_learning/convolutional/midproj$
58880/60000 [=====] - ETA: 0s - loss: 0.0543 - acc: 0.9
58944/60000 [=====] - ETA: 0s - loss: 0.0544 - acc: 0.9
59008/60000 [=====] - ETA: 0s - loss: 0.0544 - acc: 0.9
59072/60000 [=====] - ETA: 0s - loss: 0.0544 - acc: 0.9
59136/60000 [=====] - ETA: 0s - loss: 0.0544 - acc: 0.9
59200/60000 [=====] - ETA: 0s - loss: 0.0545 - acc: 0.9
59264/60000 [=====] - ETA: 0s - loss: 0.0545 - acc: 0.9
59328/60000 [=====] - ETA: 0s - loss: 0.0545 - acc: 0.9
59392/60000 [=====] - ETA: 0s - loss: 0.0544 - acc: 0.9
59456/60000 [=====] - ETA: 0s - loss: 0.0544 - acc: 0.9
59520/60000 [=====] - ETA: 0s - loss: 0.0544 - acc: 0.9
59584/60000 [=====] - ETA: 0s - loss: 0.0544 - acc: 0.9
59648/60000 [=====] - ETA: 0s - loss: 0.0545 - acc: 0.9
59712/60000 [=====] - ETA: 0s - loss: 0.0544 - acc: 0.9
59776/60000 [=====] - ETA: 0s - loss: 0.0545 - acc: 0.9
59840/60000 [=====] - ETA: 0s - loss: 0.0545 - acc: 0.9
59904/60000 [=====] - ETA: 0s - loss: 0.0545 - acc: 0.9
59968/60000 [=====] - ETA: 0s - loss: 0.0544 - acc: 0.9
60000/60000 [=====] - ETA: 0s - loss: 0.0545 - acc: 0.9
cc: 0.9837 - val_loss: 0.0379 - val_acc: 0.9876
10000/10000 [=====] - 54s 897us/step - loss: 0.0545 - a
Test accuracy: 0.9876
shefali@shefali-HP-Notebook: ~/Documents/deep_learning/convolutional/midproj$
```

ANALYSIS & CONCLUSION

- We get almost the same results with above changes.
- Running time was almost same even when i changed all values.