A NOVEL METHOD FOR HANDWRITTEN DIGIT RECOGNITION SYSTEM

TEAM ID : PNT2022TMID04292
PROJECT MEMBERS : Sneha S
Thivasini M
Dhanya T
Kishore R

MODEL BUILDING

Add CNN Layers :

The convolution layer is the core building block of the CNN. It carries the main portion of the network's computational load. This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image but is more in-depth. This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels.The Sequential model is a linear stack of layers. You can create a Sequential model by passing a list of layer instances to the constructor.

```
[ ]  model = Sequential()
     model.add(Conv2D(64, (3,3),input_shape=(28,28,1),activation='relu'))
     model.add(Flatten())
     model.add(Dense(number_of_classes, activation='softmax'))
```

Compiling The Model

With both the training data defined and model defined, now configure the learning process. This is accomplished with a call to the compile () method of the Sequential model class. Compilation requires 3 arguments: an optimizer, a loss function, and a list of metrics. In our project, we have 2 classes in the output, so the loss is binary cross entropy.

If there are more than two classes in output put “loss = categorical_cross entropy”.

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

Train the Model

Arguments:

steps_per_epoch : it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.

Epochs: an integer and number of epochs we want to train our model for.

Validation_data :

- an inputs and targets list
- a generator

- inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.

validation_steps: only if the validation_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

```
model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=5, batch_size=32)
```

```
Epoch 1/5
1875/1875 [==============================] - 54s 29ms/step - loss: 0.6850 - accuracy: 0.9443 - val_loss: 0.1015 - val_accuracy: 0.9727
Epoch 2/5
1875/1875 [==============================] - 44s 23ms/step - loss: 0.0748 - accuracy: 0.9767 - val_loss: 0.1141 - val_accuracy: 0.9691
Epoch 3/5
1875/1875 [==============================] - 45s 24ms/step - loss: 0.0608 - accuracy: 0.9815 - val_loss: 0.1333 - val_accuracy: 0.9661
Epoch 4/5
1875/1875 [==============================] - 44s 23ms/step - loss: 0.0511 - accuracy: 0.9847 - val_loss: 0.1508 - val_accuracy: 0.9689
Epoch 5/5
1875/1875 [==============================] - 45s 24ms/step - loss: 0.0422 - accuracy: 0.9877 - val_loss: 0.1732 - val_accuracy: 0.9679
<keras.callbacks.History at 0x7f83867bc1d0>Epoch 1/5
1875/1875 [==============================] - 45s 24ms/step - loss: 0.7106 - accuracy: 0.9440 - val_loss: 0.1169 - val_accuracy: 0.9673
Epoch 2/5
1875/1875 [==============================] - 43s 23ms/step - loss: 0.0735 - accuracy: 0.9775 - val_loss: 0.1087 - val_accuracy: 0.9716
Epoch 3/5
1875/1875 [==============================] - 44s 23ms/step - loss: 0.0627 - accuracy: 0.9810 - val_loss: 0.1282 - val_accuracy: 0.9656
Epoch 4/5
1875/1875 [==============================] - 43s 23ms/step - loss: 0.0503 - accuracy: 0.9847 - val_loss: 0.1473 - val_accuracy: 0.9676
Epoch 5/5
1875/1875 [==============================] - 44s 23ms/step - loss: 0.0385 - accuracy: 0.9890 - val_loss: 0.1888 - val_accuracy: 0.9673
<keras.callbacks.History at 0x7f8381de9b10>
```

Observing The Metrics:

Printing the metrics which lists out the Test loss and Test accuracy

- Loss value implies how poorly or well a model behaves after each iteration of optimization.
- An accuracy metric is used to measure the algorithm's performance in an interpretable way.

```
[ ]  metrics=model.evaluate(x_test, y_test, verbose=0)
     print("Metrics(Test  loss & Test Accuracy): ")
     print(metrics)
```

```
Metrics(Test  loss & Test Accuracy):
[0.18875671923160553, 0.9672999978065491]
```

Testing the model

Firstly we are slicing the x_test data until the first four images. In the next step we are printing the predicted output.

```
[ ]  prediction = model.predict(x_test[:4])
     print(prediction)
```

```
1/1 [==============================] - 0s 91ms/step
[[2.9003530e-10 3.5339316e-17 5.8822086e-10 9.9681392e-07 8.1262446e-16
  1.4898324e-11 2.2605819e-17 9.9999905e-01 3.2381652e-13 5.7502453e-10]
 [6.4138507e-14 1.1420456e-13 9.9999988e-01 1.9053617e-12 3.0449039e-20
  1.5983823e-19 6.6109344e-08 2.1400335e-24 7.5008691e-14 1.1501987e-22]
 [3.9195513e-07 9.9999964e-01 4.2406631e-10 1.6127368e-13 1.9968665e-08
```

We already predicted the input from the x_test. According to that, by using argmax function here we are printing the labels with high prediction values.

```
import numpy as np
print(np.argmax(prediction,axis=1))
print(y_test[:4])
```

```
[7 2 1 0]
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

SAVE THE MODEL

The model is saved with .h5 extension . An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

```
[ ]  model.save('models/mnistCNN.h5')
```

TEST WITH SAVED MODEL

Firstly we are loading the model which was built. Then we are applying for a loop for the first four images and converting the image to the required format. Then we are resizing the input image, converting the image as per the CNN model and we are reshaping it according to the requirement. At last, we are predicting the result.

```
from tensorflow.keras.models import load_model
model = load_model(r'models/mnistCNN.h5')
from PIL import Image
import numpy as np
for index in range(4):
  img = Image.open('data/'+ str(index) +'.png').convert("L")
  img = img.resize((28,28))
  im2arr = np.array(img)
  im2arr = im2arr.reshape(1,28,28,1)
  y_pred = model.predict(im2arr)
  print(y_pred)
```

```
1/1 [==============================] - 0s 68ms/step
[[4.4064937e-04 9.9234515e-01 6.1588908e-05 1.2934327e-04 2.0030871e-05
  8.5728883e-04 6.1208014e-03 1.8415057e-06 2.1813921e-05 1.4430716e-06]]
1/1 [==============================] - 0s 20ms/step
[[3.7982823e-05 6.4035934e-01 2.1324331e-06 2.8924141e-02 3.2363313e-01
  4.3717464e-03 3.6930996e-07 2.6595478e-03 9.4047255e-06 2.2602360e-06]]
1/1 [==============================] - 0s 20ms/step
[[4.3565833e-06 1.6701877e-03 2.0151566e-04 9.7906780e-01 8.1160861e-06
  8.0342521e-04 1.3193951e-05 1.6774451e-02 1.4114690e-06 1.4555111e-03]]
1/1 [==============================] - 0s 20ms/step
[[8.4519103e-05 7.0903701e-04 8.0753549e-04 2.8623709e-02 4.2507158e-06
  3.2120671e-03 6.8504387e-06 1.4510438e-04 3.0150916e-03 9.6339184e-01]]
```