Designing CNN for Optical Character Recognition

Margo Cabuy
Daria Shumkova
Zeeshan Hussain Khand

What is Optical Character Recognition (OCR)

Popular challenge in Computer Vision

Optical Character Recognition (OCR) is the process of detecting and reading text in images

Handwritten OCR can work with characters captured at various angles and shapes

AIM of the Project

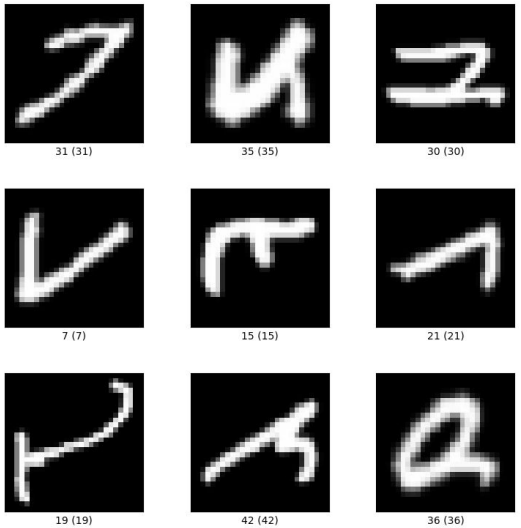
- Build an OCR system by implementing a CNN architecture to distinguish between handwritten letters from the EMNIST Letters dataset.
- Evaluate CNN model on handwritten letters captured by camera.

What is EMNIST Dataset

EMNIST dataset is a set of handwritten character digits derived from the NIST Special Database 19 and converted to a 28x28 pixel image format.

It consists of a set of upper- and lowercase letters, categorized into 26 classes.

The dataset is balanced.



Test datasets

1

6.000 images from EMNIST Letters

(a) Test 1

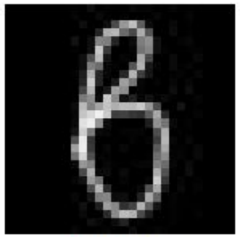
2

52 Handwritten letters captured by camera

(b) Test 2

3

52 letters drawn with the use of Tkinter user interface



(c) Test 3

Loading EMNIST data

Vast amount of data in EMNIST

Download from NIST site as a zipped MATLAB file

Temporarily whole dataset stored

Needed data pulled out, rest out of memory

```
def load_data(path="emnist_matlab.npz", type="letters"):

    # Log about loading
    logging.basicConfig(level=logging.INFO)
    logging.info('Loading dataset = emnist')

    # Load data
    path = get_file(
        path, origin=("http://www.itl.nist.gov/iaui/vip/cs_links/EMNIST/"
                      "matlab.zip")
    )
    with ZipFile(path, "r") as opened_zip:

        # Read file and temporarily store it
        file_name = f"./{type}.mat"
        f = open(file_name, "wb")
        f.write(opened_zip.read(f"matlab/emnist-{type}.mat"))
        f.close()

        # Load data from Matlab file.
        # Source: https://stackoverflow.com/a/53547262
        mat = sio.loadmat(file_name)
        data = mat["dataset"]
        input_train = data["train"][0, 0]["images"][0, 0]
        target_train = data["train"][0, 0]["labels"][0, 0].flatten()
        input_test = data["test"][0, 0]["images"][0, 0]
        target_test = data["test"][0, 0]["labels"][0, 0].flatten()

        # Remove data when loaded
        os.remove(file_name)
```

NIST National Institute of Standards and Technology U.S. Department of Commerce

Preprocessing image data

EMNIST data

loaded from site

Divide in sets

- train
- validation
- test 1

1D into 28x28 2D array

```
data.reshape( (data.shape[0],
28, 28), order='F')
```

Camera data

loaded from drive

Threshold image

empirically determined

Tkinter data

loaded from drive

downscale to 28x28 greyscale

```
Image.open(infilename)
ImageOps.grayscale(img)
img_gray.resize((28, 28), Image.ANTIALIAS)
```

normalization

```
pixels = pixels/ 255.0
```

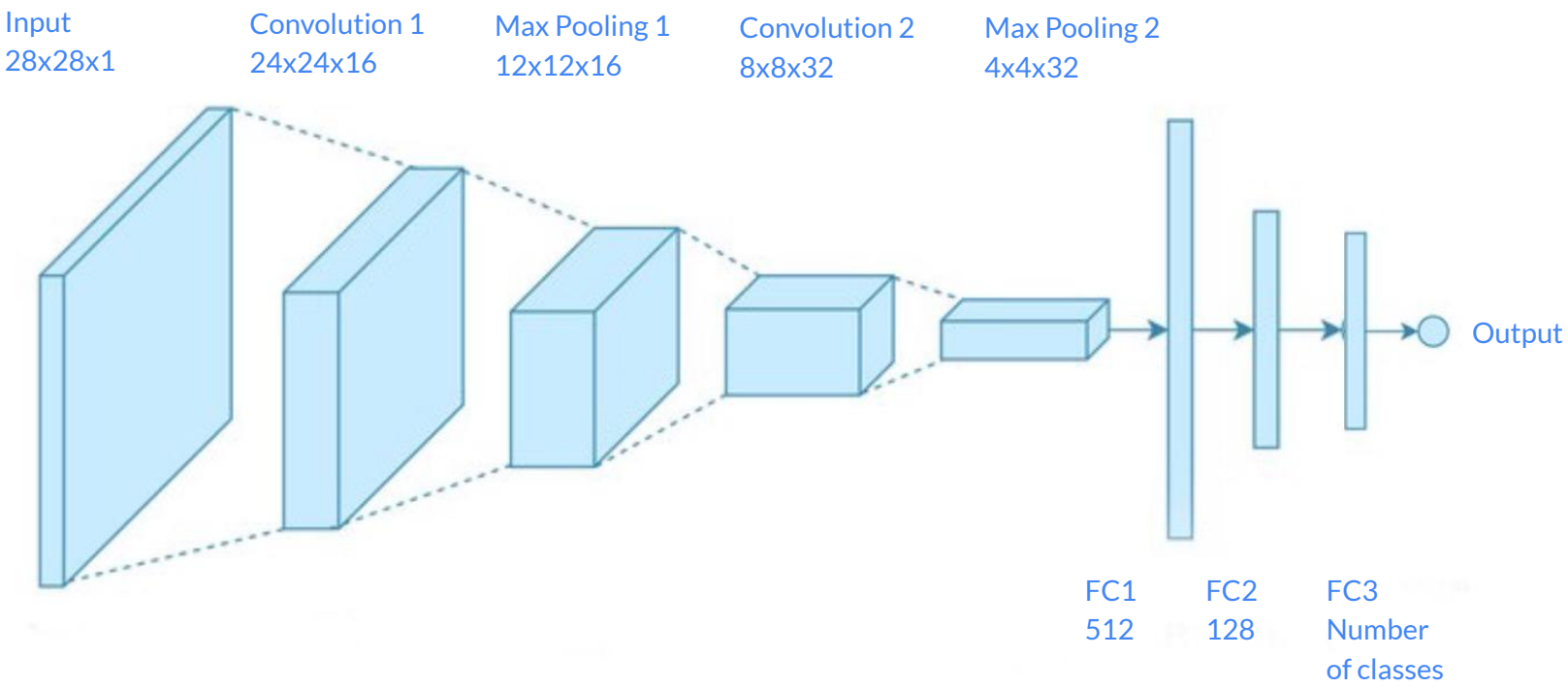
adjust dimensionality of data

```
data.reshape(#images, 1, 28, 28)
```

convert to pytorch tensors

```
torch.from_numpy(data).type(torch.float)
```

Model architecture



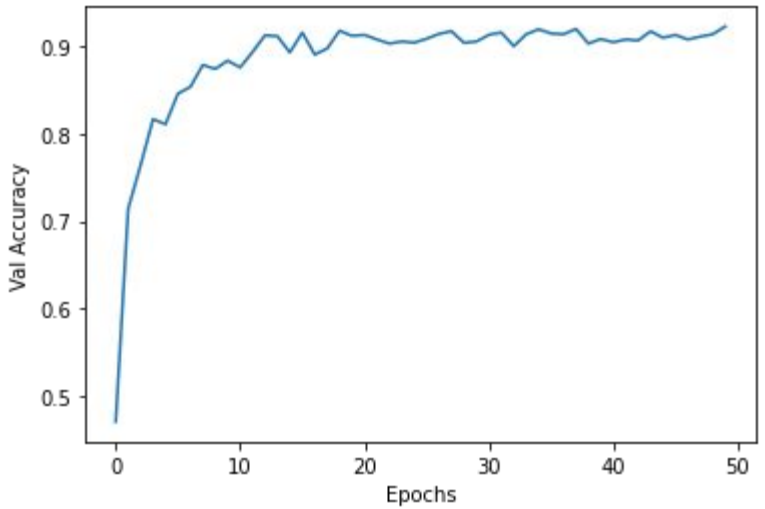
Model architecture

```python
class CNN(nn.Module):

    def __init__(self, num_classes=27):
        super(CNN, self).__init__()
        self.conv = nn.Conv2d(1, 16, 5)
        self.pool = nn.MaxPool2d(2)
        self.conv2 = nn.Conv2d(16, 32, 5)
        self.drop = nn.Dropout(p=0.2)
        self.pool2 = nn.MaxPool2d(2)
        self.fc1 = nn.Linear(32*4*4,512)
        self.fc2 = nn.Linear(512,128)
        self.fc3 = nn.Linear(128,num_classes)

    def forward(self, x):
        x = torch.nn.functional.relu(self.conv(x))
        x = self.pool(x)
        x = torch.nn.functional.relu(self.conv2(x))
        x = self.drop(x)
        x = self.pool2(x)
        x = torch.flatten(x, 1)
        x = torch.nn.functional.relu(self.fc1(x))
        x = torch.nn.functional.relu(self.fc2(x))
        x = self.fc3(x)

        return x
        cnn = CNN()
```

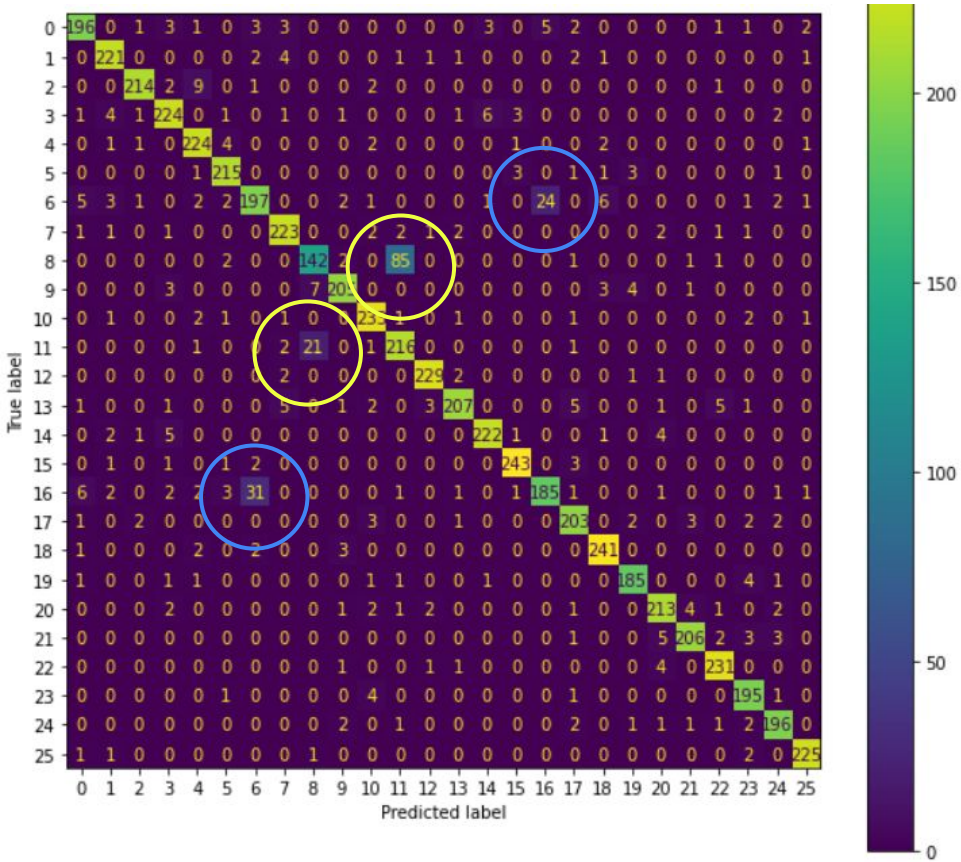


Evaluation of the model using test statistics

Botlab 99.2%
Peng and Yin 88.77%-99.75%

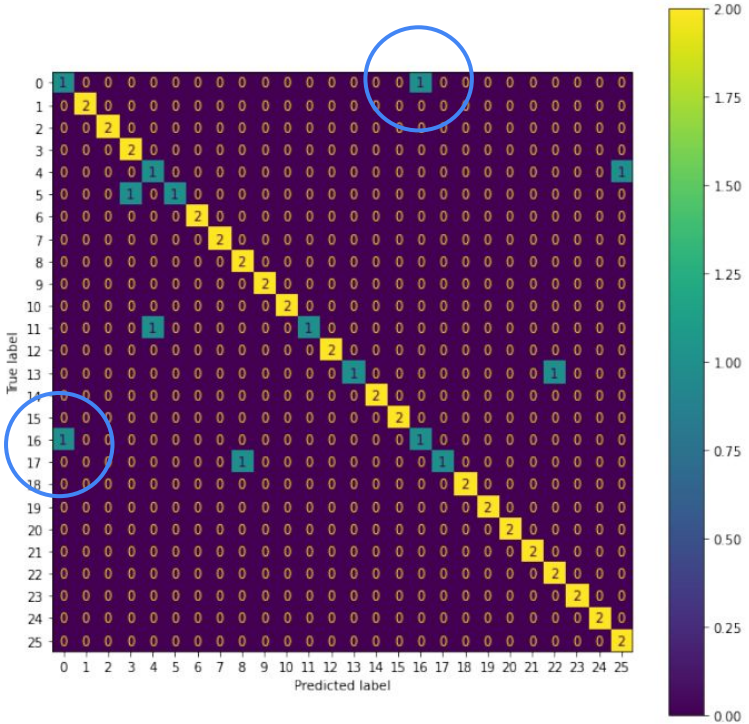
Test set number & sample size	Accuracy	F1 score	Precision score	Recall score
1 (6000) EMNIST letters	0.92	0.91	0.92	0.92
2 (52) Camera captured	0.87	0.86	0.89	0.87
3 (52) Tk UI	0.88	0.88	0.92	0.88

Confusion Matrix Test 1

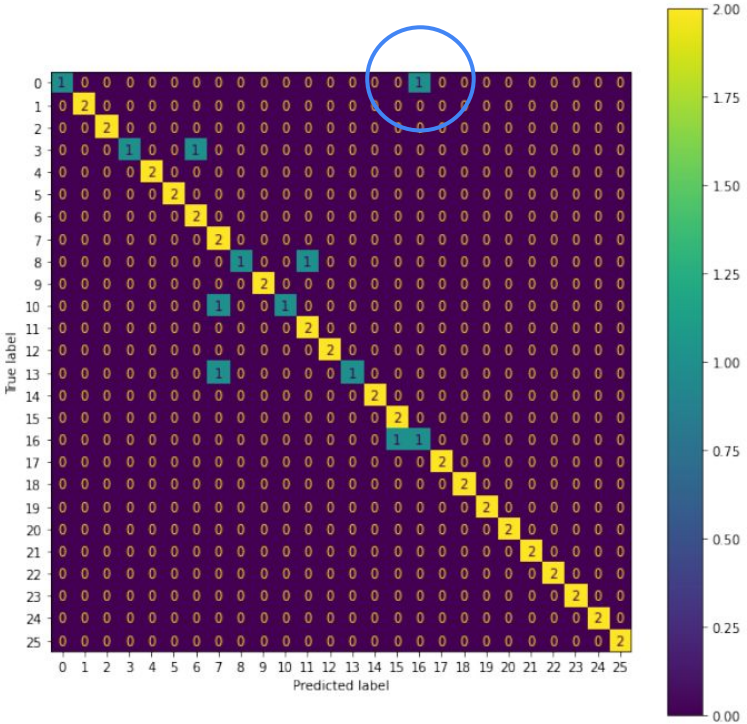


# of mislabeled cases	Letter 1	Letter 2
106	I	L
55	G	Q

Confusion Matrix Test 2 and Test 3



(a) Test 1



(b) Test 2

Mislabeling of A and Q letters together having 3 wrong predictions out of 8

Limitations and Challenges

The EMNIST Letters dataset is too big, therefore, we used only its part. But we still achieved validation accuracy of 92 %

Test 2 images were captured by camera during daylight. But when we ran model on more dark images, model performance was poor

The model generally does not show good performance on some images, because they are very different from EMNIST data

Conclusion

Automatic recognition of handwritten letters, captured by camera, can be implemented using the Pytorch CNN framework and showed an accuracy of 87%