CPSC 542: Assignment #1
Implement a Convolutional Neural Network

LEWIS, T.
TYLEWIS@CHAPMAN.EDU
ID# 002366930
02/27/24

Problem statement

How can we enable computers to understand pictures containing text? This task encompasses a range of processes, from detecting individual characters to also understanding the syntactical and semantic structures of language.

One relevant application of this technology is in the digitization of historical texts, which often exhibit a wide variety of stylistic variations. These variations can include differences in handwriting, obscure characters, and even physical degradation of the documents over time. Such texts can be challenging to decipher even for trained human eyes, making the task for computers notably complex.

Incorporating a convolutional neural network (CNN) as a deep NN solution for this task is a promising approach to address this problem. The convolutional model excels in identifying key patterns in images, regardless of their coordinate location. By training a CNN on examples of specific characters of interest, we can employ a system to recognize and interpret handwritten stylistic variations with high accuracy.

Identify a vision dataset to work with for classification tasks:

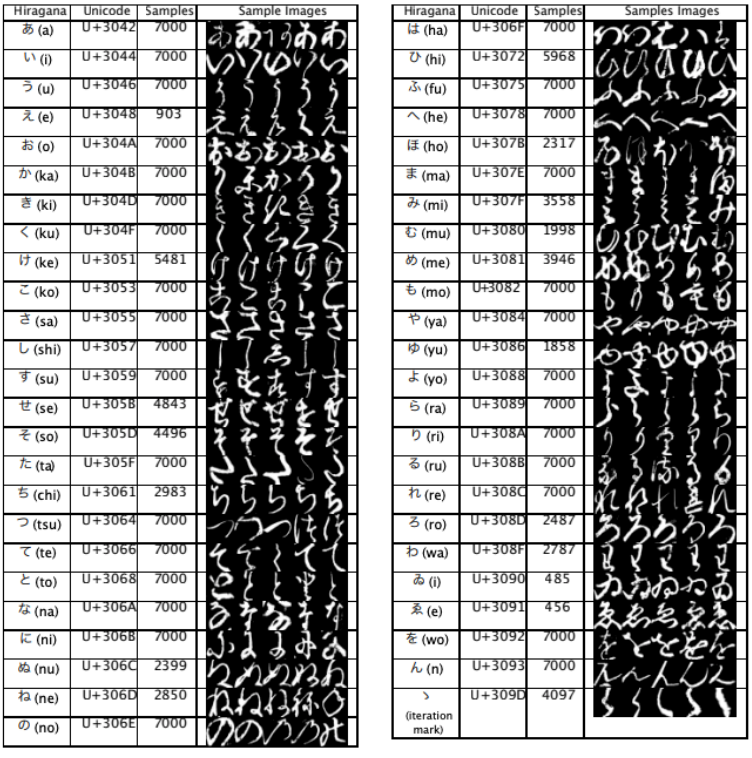
For assignment completion, I opted to use the Kuzushiji-49 data.

The Kuzushiji-49 database contains 70,000 examples of handwritten Kuzushiji characters - Japanese cursive-styled characters. Containing 49 different classes including one punctuation symbol and 48 Hiragana characters, (Fig. 1).

The computer vision exposure I have prior to this course is an assignment to use Euclidean Distance K-nearest neighbors to predict labels from a similar dataset, except containing handwritten Arabic numerals (1, 2, 3, etc). This yielded around ~.8 accuracy as I remember.

Initially I used the 'KMNIST-10' dataset, with just 10 classes of characters, but during the week extension, transitioned to this set which includes 5x more classes.

Each image is a vector/tensor of shape (28,28,1) to represent a 28x28px greyscale image



Hiragana	Unicode	Samples
あ (a)	U+3042	7000
い (i)	U+3044	7000
う (u)	U+3046	7000
え (e)	U+3048	903
お (o)	U+304A	7000
か (ka)	U+304B	7000
き (ki)	U+304D	7000
く (ku)	U+304F	7000
け (ke)	U+3051	5481
こ (ko)	U+3053	7000
さ (sa)	U+3055	7000
し (shi)	U+3057	7000
す (su)	U+3059	7000
せ (se)	U+305B	4843
そ (so)	U+305D	4496
た (ta)	U+305F	7000
ち (chi)	U+3061	2983
つ (tsu)	U+3064	7000
て (te)	U+3066	7000
と (to)	U+3068	7000
な (na)	U+306A	7000
に (ni)	U+306B	7000
ぬ (nu)	U+306C	2399
ね (ne)	U+306D	2850
の (no)	U+306E	7000

Hiragana	Unicode	Samples
は (ha)	U+306F	7000
ひ (hi)	U+3072	5968
ふ (fu)	U+3075	7000
へ (he)	U+3078	7000
ほ (ho)	U+307B	2317
ま (ma)	U+307E	7000
み (mi)	U+307F	3558
む (mu)	U+3080	1998
め (me)	U+3081	3946
も (mo)	U+3082	7000
や (ya)	U+3084	7000
ゆ (yu)	U+3086	1858
よ (yo)	U+3088	7000
ら (ra)	U+3089	7000
り (ri)	U+308A	7000
る (ru)	U+308B	7000
れ (re)	U+308C	7000
ろ (ro)	U+308D	2487
わ (wa)	U+308F	2787
ゐ (i)	U+3090	485
ゑ (e)	U+3091	456
を (wo)	U+3092	7000
ん (n)	U+3093	7000
ゝ (iteration mark)	U+309D	4097

Fig. 1: Characters contained in Kuzushiji-49 dataset

CPSC 542: Assignment #1
Implement a Convolutional Neural Network

LEWIS, T.
TYLEWIS@CHAPMAN.EDU
ID# 002366930
02/27/24

Methods

The Kuzushiji-49 data is not included with native PyTorch. This required implementation to fetch the datasets, I used LLM to generate a driver for this functionality. The images are already normalized and did not require any preprocessing or augmentation steps. I applied a dataset split allocating 75% of the images for training and 25% for validation.

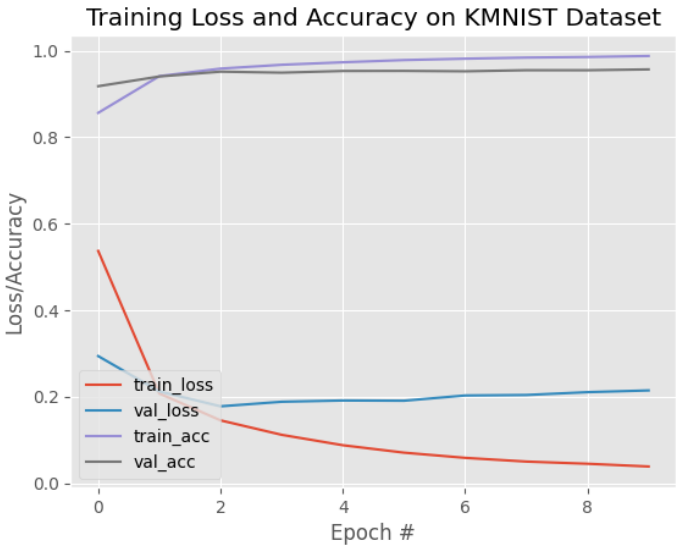
My model employs a LeNet architecture built with the PyTorch library, and features a sequential arrangement of layers designed to process the input images:

- **Convolutional Layers:** Two convolutional layers are included, where the first layer applies 20 filters of size 5x5, and the second layer uses 50 filters of the same size. These layers are responsible for extracting features from the input images.
- **Activation Functions:** Each convolutional layer is followed by a ReLU activation function to introduce non-linearity, enhancing the model's learning capability.
- **Pooling Layers:** Max-pooling layers follow each activation function, reducing the spatial size of the representation and hence the number of parameters and computation in the network.
- **Fully Connected Layers:** After flattening the output from the convolutional layers, it is passed through a fully connected layer with 500 units, followed by a ReLU activation, culminating in a soft-max classifier that outputs the probability distribution over the character classes.

Results

Training Configuration:

The network is trained over 10 epochs with a batch size of 64, employing the Adam optimizer and a learning rate of 1e-3.



Performance Monitoring: A history of training and validation loss and accuracy is maintained throughout the training epochs, serving as a metric for evaluating the model's performance.

Evaluation: Upon completion of training, the model is assessed on a separate test dataset, and a classification report is generated. This report details the model's precision, recall, and F1-scores for each digit class, offering a comprehensive evaluation of its classification efficacy.

CPSC 542: Assignment #1
Implement a Convolutional Neural Network

LEWIS, T.
TYLEWIS@CHAPMAN.EDU
ID# 002366930
02/27/24

```
[INFO] training the CNN...
[INFO] EPOCH: 1/10
Train loss: 0.536884, Train accuracy: 0.8561
Val loss: 0.293800, Val accuracy: 0.9176

[INFO] EPOCH: 2/10
Train loss: 0.207312, Train accuracy: 0.9410
Val loss: 0.212111, Val accuracy: 0.9402

[INFO] EPOCH: 3/10
Train loss: 0.145194, Train accuracy: 0.9584
Val loss: 0.177643, Val accuracy: 0.9513

[INFO] EPOCH: 4/10
Train loss: 0.111795, Train accuracy: 0.9672
Val loss: 0.188212, Val accuracy: 0.9490

[INFO] EPOCH: 5/10
Train loss: 0.087725, Train accuracy: 0.9730
Val loss: 0.191091, Val accuracy: 0.9529

[INFO] EPOCH: 6/10
Train loss: 0.070553, Train accuracy: 0.9781
Val loss: 0.190573, Val accuracy: 0.9531

[INFO] EPOCH: 7/10
Train loss: 0.058527, Train accuracy: 0.9813
Val loss: 0.202740, Val accuracy: 0.9521

[INFO] EPOCH: 8/10
Train loss: 0.049918, Train accuracy: 0.9838
Val loss: 0.203872, Val accuracy: 0.9546

[INFO] EPOCH: 9/10
Train loss: 0.044887, Train accuracy: 0.9853
Val loss: 0.210383, Val accuracy: 0.9546

[INFO] EPOCH: 10/10
Train loss: 0.038549, Train accuracy: 0.9875
Val loss: 0.214400, Val accuracy: 0.9566

[INFO] Completed training.
[INFO] total time taken to train the model: 1619.54s
```

```
[INFO] evaluating network...
              precision    recall  f1-score   support

     Class 0       0.93      0.94      0.94      1000
     Class 1       0.97      0.94      0.96      1000
     Class 2       0.94      0.95      0.95      1000
     Class 3       0.83      0.90      0.86       126
     Class 4       0.94      0.91      0.93      1000
     Class 5       0.89      0.88      0.88      1000
     Class 6       0.94      0.90      0.92      1000
     Class 7       0.86      0.93      0.90      1000
     Class 8       0.82      0.93      0.87       767
     Class 9       0.95      0.90      0.93      1000
    Class 10       0.93      0.95      0.94      1000
    Class 11       0.95      0.90      0.93      1000
    Class 12       0.91      0.89      0.90      1000
    Class 13       0.91      0.87      0.89       678
    Class 14       0.87      0.86      0.86       629
    Class 15       0.95      0.93      0.94      1000
    Class 16       0.98      0.94      0.96       418
    Class 17       0.93      0.94      0.93      1000
    Class 18       0.96      0.91      0.94      1000
    Class 19       0.95      0.96      0.95      1000
    Class 20       0.89      0.91      0.90      1000
    Class 21       0.90      0.94      0.92      1000
    Class 22       0.85      0.90      0.88       336
    Class 23       0.90      0.93      0.91       399
    Class 24       0.95      0.92      0.93      1000
    Class 25       0.94      0.89      0.91      1000
    Class 26       0.94      0.95      0.95       836
    Class 27       0.96      0.88      0.92      1000
    Class 28       0.96      0.92      0.94      1000
    Class 29       0.82      0.92      0.87       324
    Class 30       0.88      0.96      0.92      1000
    Class 31       0.96      0.90      0.93       498
    Class 32       0.96      0.87      0.91       280
    Class 33       0.96      0.90      0.93       552
    Class 34       0.92      0.96      0.94      1000
    Class 35       0.89      0.92      0.91      1000
    Class 36       0.85      0.94      0.89       260
    Class 37       0.97      0.97      0.97      1000
    Class 38       0.93      0.93      0.93      1000
    Class 39       0.91      0.92      0.91      1000
    Class 40       0.90      0.90      0.90      1000
    Class 41       0.92      0.96      0.94      1000
    Class 42       0.91      0.96      0.93       348
    Class 43       0.93      0.90      0.92       390
    Class 44       0.89      0.72      0.80        68
    Class 45       0.88      0.83      0.85        64
    Class 46       0.92      0.95      0.93      1000
    Class 47       0.94      0.98      0.96      1000
    Class 48       0.88      0.81      0.85       574

    accuracy                           0.92     38547
   macro avg       0.92      0.91      0.91     38547
weighted avg       0.92      0.92      0.92     38547
```

Discussion

As shown above, the model achieved a 0.92 accuracy. The model reached its best performance during the third epoch. The accuracy seems high to me, but it could likely be improved further, which would be ideal for a consumer tool. I played around with filter and step sizes and determined that my current settings produced the highest accuracy results.

I additionally compared this performance against the KMNIST-10 dataset (results in readme), containing only ten Japanese characters — the increasing the size of the class pool led significantly higher levels of validation-loss. I anticipate this such model may be used in part as a transfer functioning layer within a picture-to-text system.