Hiragana Symbol Classification

By Jonathan Williams

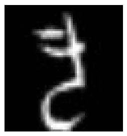
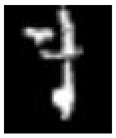
NetLin Model

Results

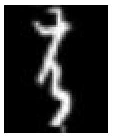
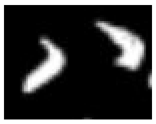
```
[[763.   7.    8.    2.    60.   8.    4.    16.   9.   10.]
 [  5.  669.   64.   36.   54.   28.   20.   30.  39.   53.]
 [  9.  1 10.  689.  60.   77.  125.  150.  27.  93.   85.]
 [ 15.   19.   26.  760.  20.   17.   10.   11.  44.    3.]
 [ 30.   28.   26.   15.  627.  19.   24.   86.   8.   52.]
 [ 64.   22.   22.   57.   18.  725.  25.   16.  31.   29.]
 [  2.   58.   47.   13.   33.   28.  721.   55.  43.   19.]
 [ 62.   12.   35.   17.   36.   7.    20.  623.  7.    30.]
 [ 31.   24.   46.   28.   20.   33.   11.   90.  705   40.]
 [ 19.   51.   37.   12.   55.   10.   15.   46.  21.  679.]]
```

Average loss: 1.0096, Accuracy: 6961/10000 (70%)

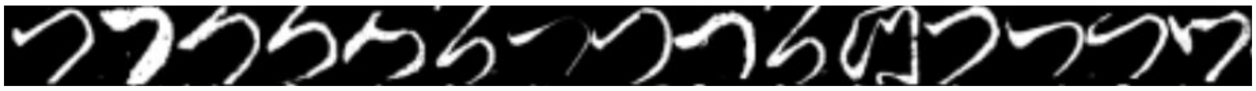
NetLin is simply a perceptron model, hence it can only accurately classify linearly separable data. It performs poorly in the KMNIST classification task because there is plenty of overlap in the features of the handwritten old-style hiragana script.
For example the following symbol for a 1 looks very similar to a symbol for a 6

1:	6:
	

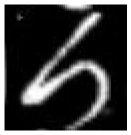
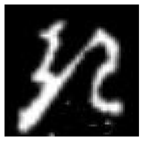
Furthermore, none of the symbols are written in one single unique way. For any number, there doesn't appear to be any features that are present in all its representations. For example, the following symbols clearly exhibit little to no correlation, yet they are meant to both represent a 5:

The model performed best on the numbers that have less variation amongst their own kind, and that are fairly unique compared to other symbols. A good example of this is the symbol for 3:



NetLin performed notably better with 3s, classifying them correctly 760 times and incorrectly a total of 2+36+60+15+57+13+17+28+12 = 240 times. The model often confused 3s with 2s (60 times) and with 5s (57 times).

A symbol for 3: 	A symbol for 2: 

NetLin also performed notably better with 0s, 5s and 6s, numbers which have distinct and less varied features. However, it performed poorly with 4s, 7s and 9s.
Clearly it is very difficult to classify images using only a single layered network with no activation function because it is unable to identify non-linearly separable features.

NetFull Model

Results
Number of hidden nodes used: 90

`[[847.   7.   5.   3.  60.   6.   4.  13.  10.   6.]`
`[  3. 783.  19.  22.  37.  19.  36.  17.  38.  35.]`
`[  1.  32. 792.  29.  14.  75.  57.  19.  12.  49.]`
`[  8.   5.  48. 894.   5.  16.  10.   8.  41.   4.]`
`[ 28.  28.  16.   6. 775.  17.  32.  53.   7.  36.]`
`[ 29.   8.  18.  10.   9. 797.   8.   4.  11.   7.]`
`[  4.  75.  33.   7.  25.  28. 821.  35.  29.  20.]`
`[ 39.   7.  25.   6.  23.   4.  10. 741.  11.  28.]`
`[ 31.  25.  28.  12.  25.  26.   6.  53. 836.  24.]`
`[ 10.  30.  16.  11.  27.  12.  16.  57.   5. 791.]]`

Average loss: 2.6059, Accuracy: 8077/10000 (81%)

NetFull performed considerably better because it was able to make use of a hidden layer along with the tanh activation function. With a hidden layer, this model can classify non-linearly separable data by mapping inputs to linearly separable points on a plane of higher dimension.

Much like NetLin, this model also performed better when classifying 3s and 6s as well as 8s. Similarly, the model still performed poorly on 4s, 7s and 9s. This indicates that fully connected networks still have issues with image classification tasks.

NetConv Model

Results

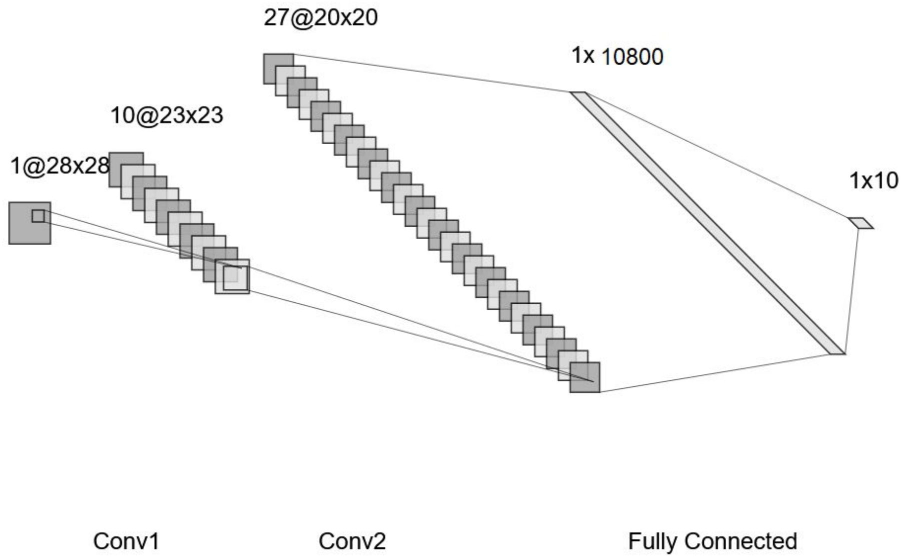
```
[[943.   4.    7.    0.   19.    4.    4.   16.    8.    6.]
 [  1. 925.    7.    1.    8.   13.    3.    5.    7.   11.]
 [  1.  14.  902.   27.    9.   53.   28.    9.   13.    5.]
 [  2.   1.   26.  939.    3.    2.    4.    1.    5.    2.]
 [ 33.  10.   11.    5.  925.    7.   10.    8.    6.   11.]
 [  5.   0.    6.   10.    4.  902.    6.    3.   10.    3.]
 [  2.  28.   25.    7.   13.   12.  940.   21.    3.    2.]
 [  5.   4.    5.    2.    6.    1.    2.  908.    2.    4.]
 [  6.   5.    5.    5.   10.    4.    0.    7.  944.    6.]
 [  2.   9.    6.    4.    3.    2.    3.   22.    2.  950.]]
```

Average loss: 0.3240, Accuracy: 9278/10000 (93%)

Convolutional neural networks are ideal for image classification tasks because they are able to convolve layers into feature maps where regular and more complex features can be identified. NetConv performed much better than the previous models.

The other models struggled to accurately classify 9s, whereas NetConv classified 9s notably well. This makes sense, considering that 9s have very complex, yet distinct features:

The following diagram shows my implementation of NetConv:



I used an 8x8 filter for the first and second convolution. Additionally, I used a padding of size 1 for the first convolutional layer and padding of size 2 for the second convolutional layer. Hence we have the following sizes:

- (28+1) **-** 8 + **2** = 23
- (23+1) **-** 8 + **2*2** = 20

Out of the many variations of NetConv that I tested, this particular model achieved 93% accuracy. Through my experimentation with the network properties I was able to discover why it performed the best.

Changing the number of channels used in the convolutional layers affected my results significantly. Increasing the number of channels would always increase the performance of the model. However, it also increased the training time and temperature of my laptop. Assigning more channels forces the network to compute more gradients which allows the network to retain more information pertaining to a particular feature. This is important since each layer gets smaller in size. I found that increasing the number of channels past 25 would only marginally increase the performance.

Pooling is a technique used to improve spatial variance on feature maps, enabling features to be detected anywhere on the image. Adding a max pooling layer after convolution didn't improve the performance of my model. This makes sense considering that the hiragana symbol is the only thing captured in each image and it is always located at the centre of the image. Classifying KMNIST images only requires the network to look for a specific feature in a specific location, hence max pooling has very little effect on performance. However, I did notice that using a max pooling layer decreased training time, most likely because it reduces the size of the feature map.

I discovered that padding the image and feature map improved my results. Padding prevents the network from ignoring features nearing the edge of the image. This way, more features were able to be identified.