

Optical Character Recognition

Prepared for: Eduardo Blanco

Class: CSE 598, Intro to Deep Learning [Fall'21]

Prepared by: Prakhar Bhartiya

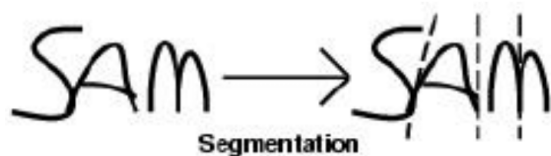
Github Link : [https://github.com/Prakhar-Bhartiya/EMNIST_evaluation_CSE598_Project]

Problem:

The world is rapidly becoming more digital. However, as humans, we still have a large number of papers that are handwritten. Almost all of a library's content may now be stored on a single hard disk. However, transferring handwritten/non-digital material to digital format is difficult. We employ the "Optical Character Recognition(OCR)" approach to solve these issues.


Many computer vision applications need character recognition, such as automatic number plate identification, robot navigation, product search, and image-based translation. The character recognition problem is separated into two categories in the literature: optical character recognition (OCR) and character recognition in natural imagery. EMNIST is the most well-known and biggest dataset for research and benchmarking in optical character recognition.

Main aim of this research based project is to get optimal result using minimal CNN model.



Input : Isolated English letter from handwritten document.

Output : Classified class label.

Eg. Input : 

Output: S

Dataset :

EMNIST Dataset

The EMNIST dataset is a set of handwritten character digits derived from the [NIST Special Database 19](#) and converted to a 28x28 pixel image format and dataset structure that directly matches the [MNIST Dataset](#).

As the original dataset contains the data set in binary format. Hence I am using PYPI python package manager EMNIST implementation.

Dataset Summary

There are six different splits provided in this dataset. A short summary of the dataset is provided below:

- EMNIST ByClass: 814,255 characters. 62 unbalanced classes.
- EMNIST ByMerge: 814,255 characters. 47 unbalanced classes.
- EMNIST Balanced: 131,600 characters. 47 balanced classes.
- EMNIST Letters: 145,600 characters. 26 balanced classes.
- EMNIST Digits: 280,000 characters. 10 balanced classes.
- EMNIST MNIST: 70,000 characters. 10 balanced classes.

Resource :

Citation : Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from <http://arxiv.org/abs/1702.05373>

NIST dataset guide : https://s3.amazonaws.com/nist-srd/SD19/sd19_users_guide_edition_2.pdf

NIST dataset : <https://www.nist.gov/services-resources/software/public-domain-ocr>

PyPI Implementation : <https://pypi.org/project/emnist/>

EMNIST ByClass

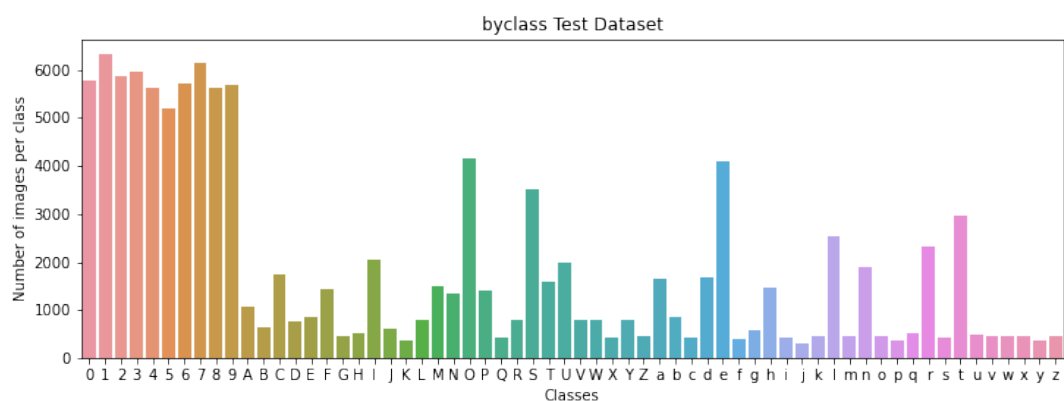
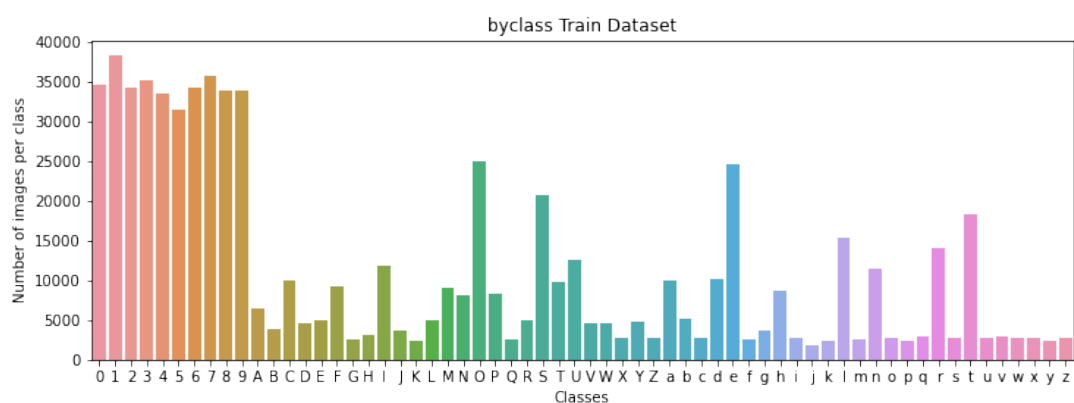
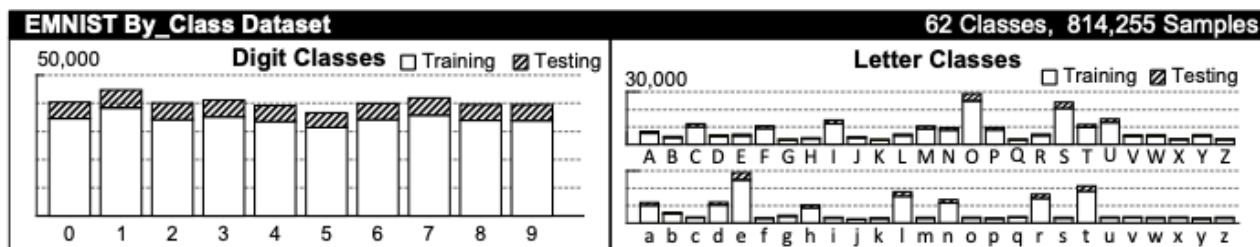
62 classes = 26 lowercase alphabets. [a - z] + 26 uppercase alphabets. [A - Z] + 10 digits [0-9]

Uppercase and lowercase letter classes are separate.

Classes are **not balanced**.

Each image is 28x28 pixel, in png format.

Dataset consists of handwritten letters from various writers.



EMNIST ByMerge

47 classes = 11 lowercase alphabets. [a - z] + 26 uppercase alphabets. [A - Z] + 10 digits [0-9]

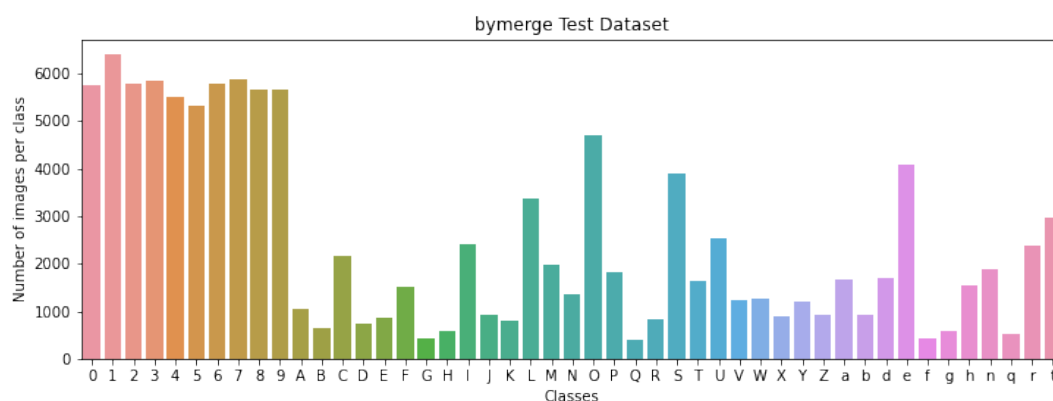
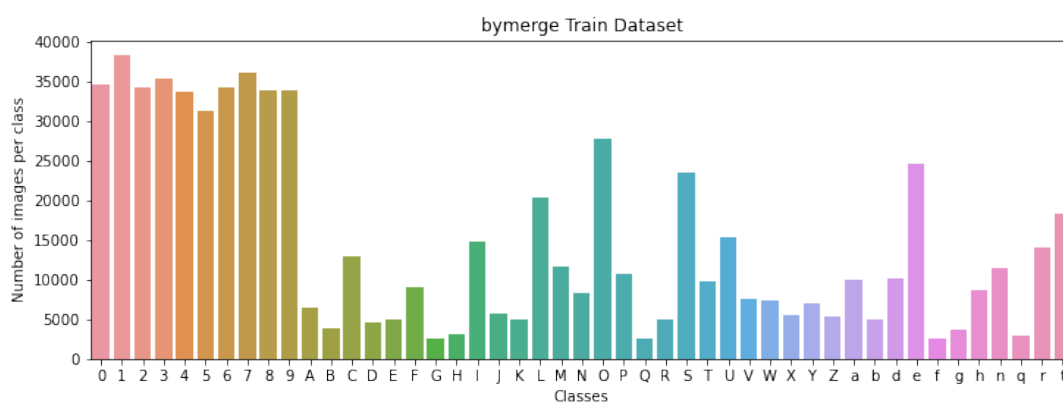
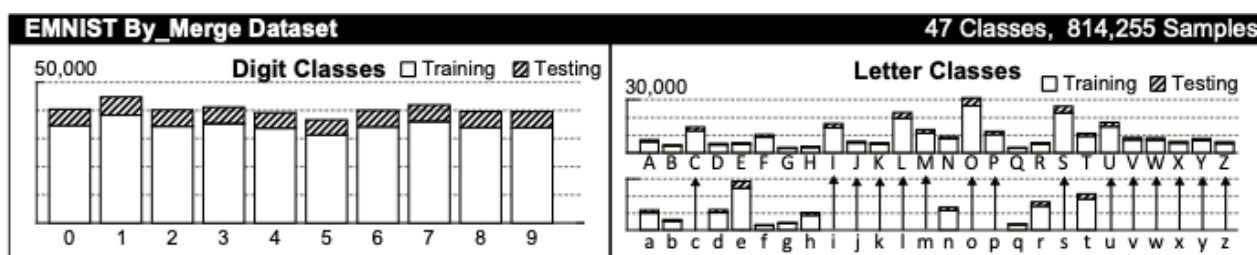
Uppercase and lowercase letter classes are separate. Except for these lowercase alphabets

['c', 'i', 'j', 'k', 'l', 'm', 'o', 'p', 's', 'u', 'v', 'w', 'x', 'y', 'z'].

Classes are **not balanced**.

Each image is 28x28 pixel, in png format.

Dataset consists of handwritten letters from various writers.



EMNIST Balanced

47 classes = 11 lowercase alphabets. [a - z] + 26 uppercase alphabets. [A - Z] + 10 digits [0-9]

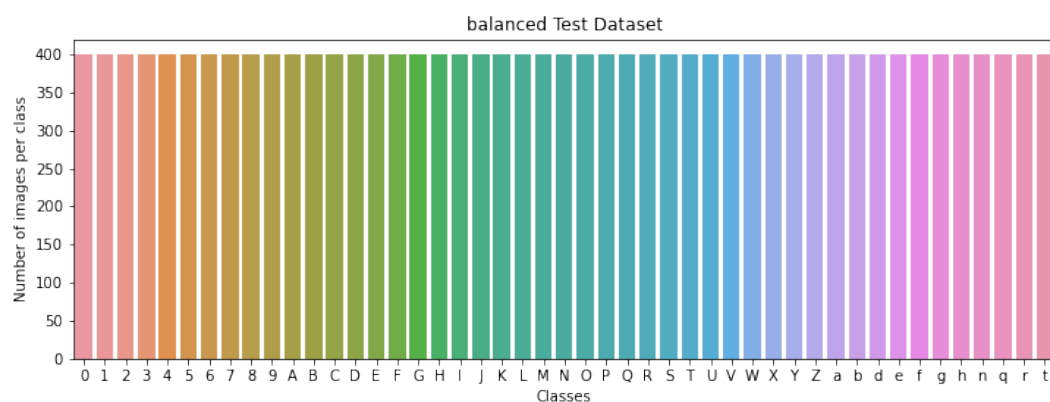
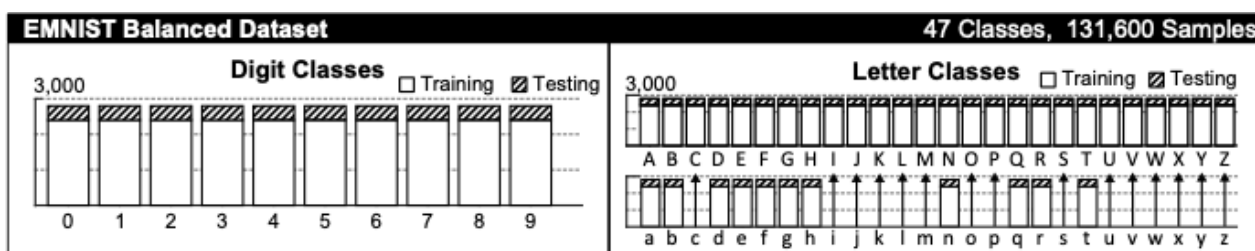
Uppercase and lowercase letter classes are separate. Except for these lowercase alphabets

['c', 'i', 'j', 'k', 'l', 'm', 'o', 'p', 's', 'u', 'v', 'w', 'x', 'y', 'z'].

Classes are **balanced**.

Each image is 28x28 pixel, in png format.

Dataset consists of handwritten letters from various writers.



EMNIST Letters

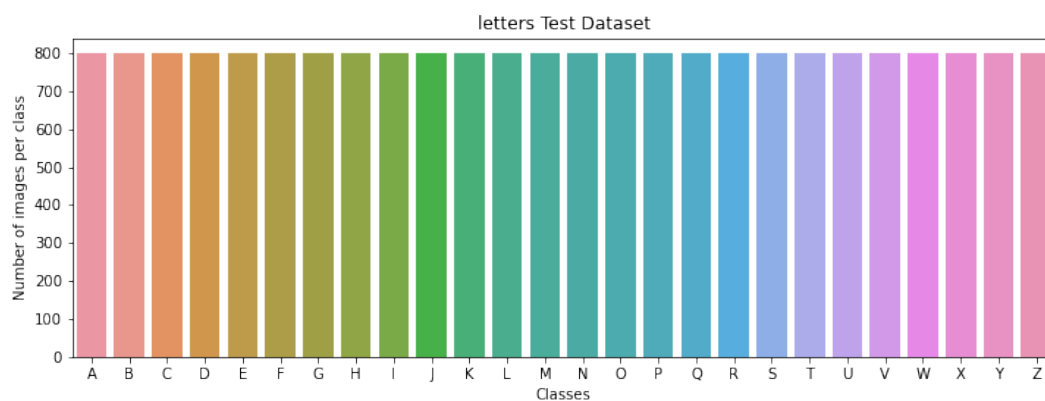
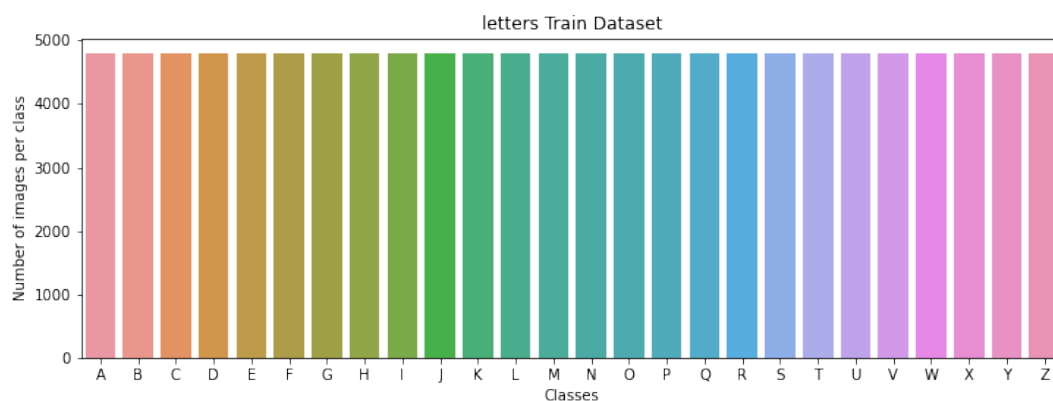
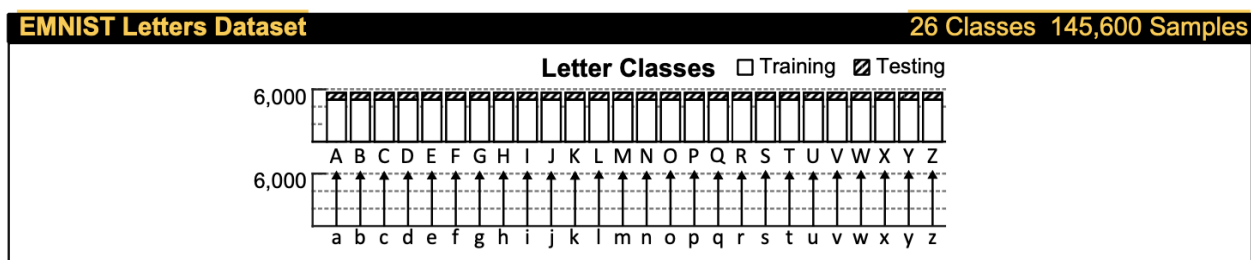
26 classes = 26 alphabets. [a/A - z/Z]

Uppercase and lowercase letter classes are merged.

Classes are **balanced**.

Each image is 28x28 pixel, in png format.

Dataset consists of handwritten letters from various writers.



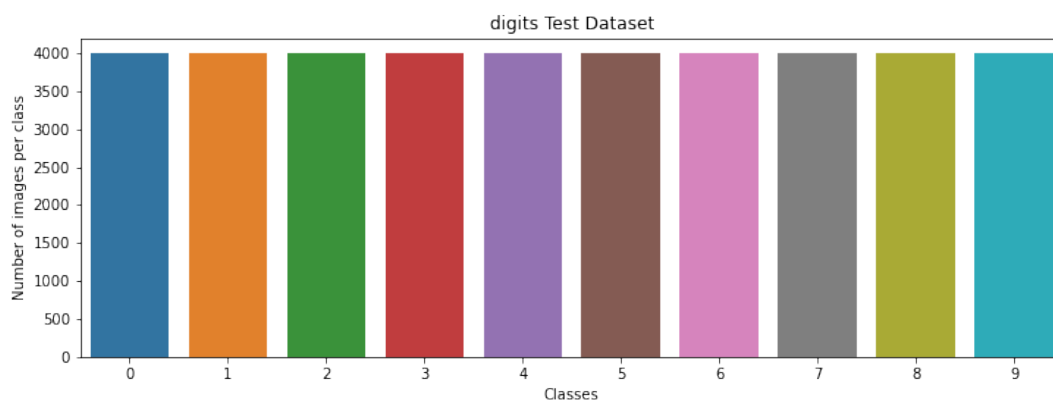
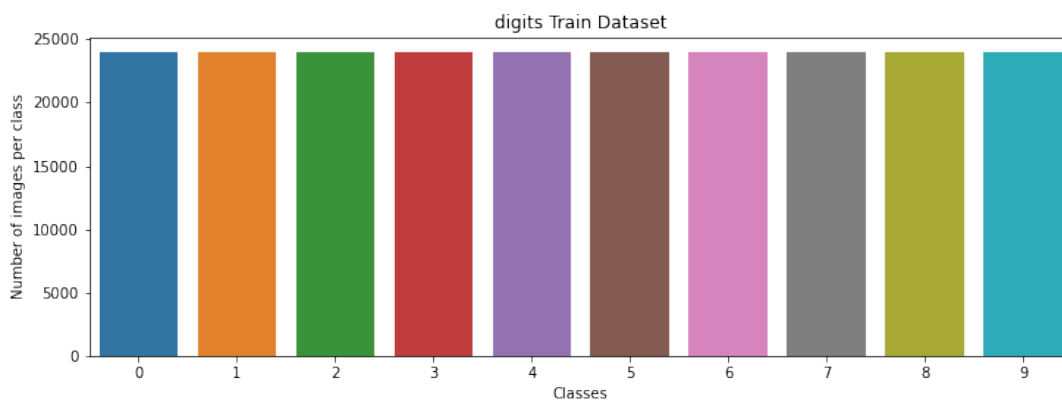
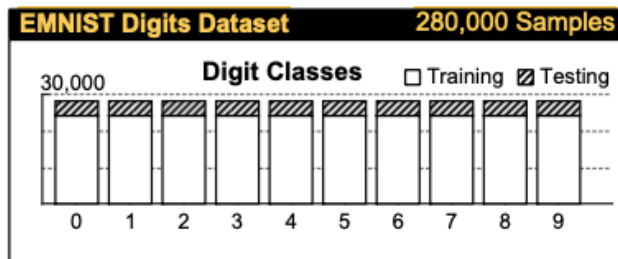
EMNIST Digits

10 classes = 10 digits [0-9]

Classes are **balanced**.

Each image is 28x28 pixel, in png format.

Dataset consists of handwritten letters from various writers.



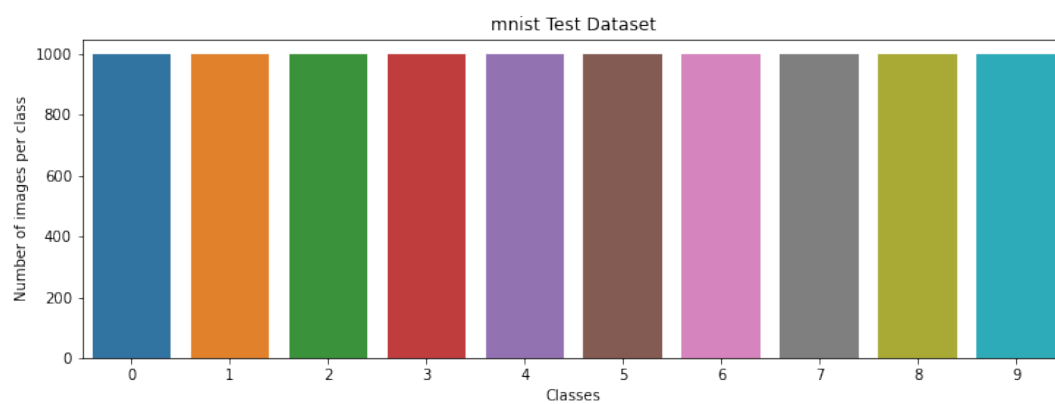
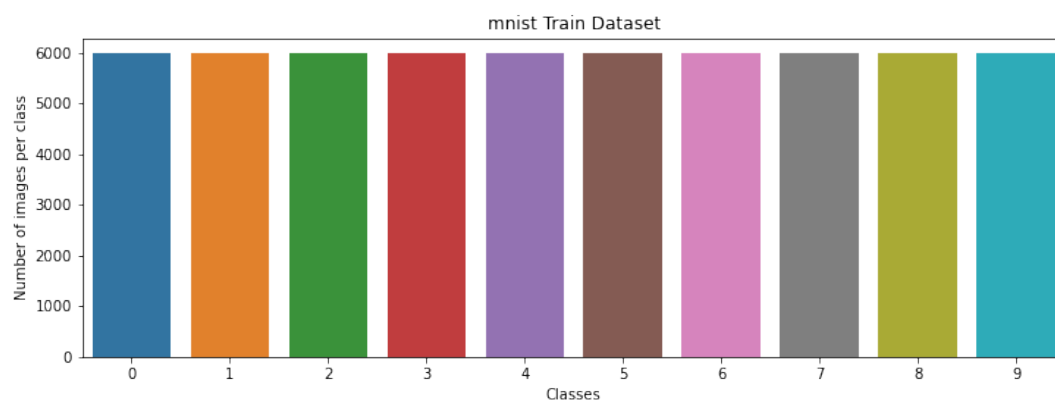
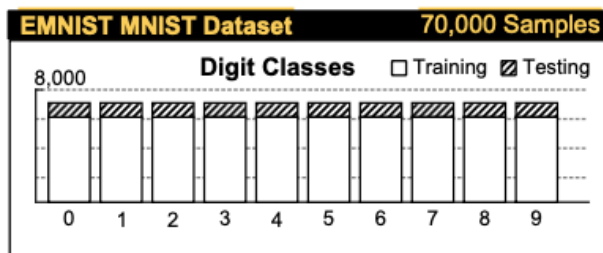
EMNIST MNIST

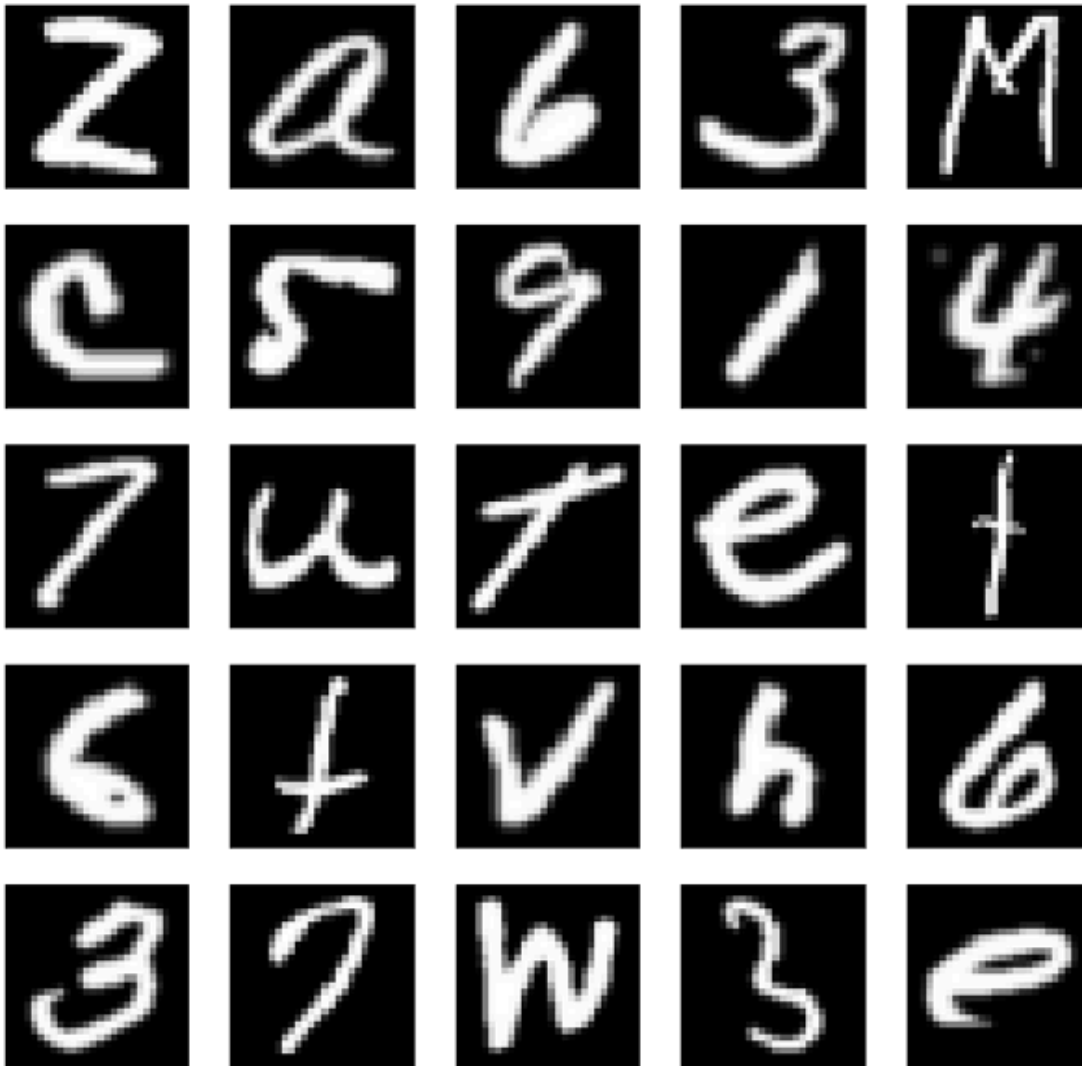
10 classes = 10 digits [0-9]

Classes are **balanced**.

Each image is 28x28 pixel, in png format.

Dataset consists of handwritten letters from various writers.



Sample Data Visualisation :

Models

Two model with 1 and 2 convolutional layers respectively.

Model 1 : Simple CNN - 1 Conv Layer

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 32)	0
flatten_1 (Flatten)	(None, 5408)	0
dense_3 (Dense)	(None, 512)	2769408
dense_4 (Dense)	(None, 128)	65664
dense_5 (Dense)	(None, 48)	6192
Total params: 2,841,584		
Trainable params: 2,841,584		
Non-trainable params: 0		

Model 2 : Complex CNN - 2 Conv Layers

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 64)	640
activation (Activation)	(None, 26, 26, 64)	0
max_pooling2d (MaxPooling2D)	(None, 13, 13, 64)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	36928
activation_1 (Activation)	(None, 11, 11, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 512)	819712
activation_2 (Activation)	(None, 512)	0
dense_1 (Dense)	(None, 128)	65664
activation_3 (Activation)	(None, 128)	0
dense_2 (Dense)	(None, 48)	6192
activation_4 (Activation)	(None, 48)	0
Total params: 929,136		
Trainable params: 929,136		
Non-trainable params: 0		

Design Choices :

Important design choices opted for training the both models.

Filter Kernel size = 3x3

Max Pooling size = 2x2

Optimiser = "Adam" [Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.]

Loss = "Categorical cross entropy" [Computes the cross-entropy loss between the labels and predictions]

Callbacks:

1. **Model Checkpoint** - Create model checkpoint with best validation accuracy
2. **Early Stopping** - stop training based on maximum validation accuracy if it do not improve for next 3 epochs
3. **Reduce LR on Plateau** : Reduce the value of learning rate if model start saturating with learning.

Batch size = 128 (Also tested with 64, both have similar result)

Epochs = 20 (generally stops at 10 epochs, depend on Early Stopping callback)

Dataset splits

When we loaded data, the default split for test and train was provided by the PyPI implementation. For validation, an 80:20 split was created on the train dataset. As a result, the overall dataset split ratios are listed below:

Training dataset : Validation dataset : Testing dataset = 69 : 17 : 14

Dataset split	#Total images	#Train Images	#Validation Images	#Test Images	Train : Val : Test
ByClass	814255	558345	139586	116323	0.69 : 0.17 : 0.14
ByMerge	814255	558345	139586	116323	0.69 : 0.17 : 0.14
Balanced	131600	90240	22560	18800	0.69 : 0.17 : 0.14
Letters	145600	99840	24960	20800	0.69 : 0.17 : 0.14
Digits	280000	192000	48000	40000	0.69 : 0.17 : 0.14
MNIST	70000	48000	12000	10000	0.69 : 0.17 : 0.14

Analysis :

For a better understanding of how our model struggles in some classes and excels in others, we chose four different analyses.

1. **Classification Report**
2. **Confusion Matrix**
3. **Visualise wrong predicted images**
4. **Activation Layer-wise Visualization**

Note : Model 2 [Complex - 2 Conv Layers] **was used to perform all of the analyses on the "Balanced" dataset.** Because a balanced dataset has all classes balanced, it is commonly used as the benchmark for testing. Model 2 was chosen because it has a higher overall accuracy.

Classification Report :

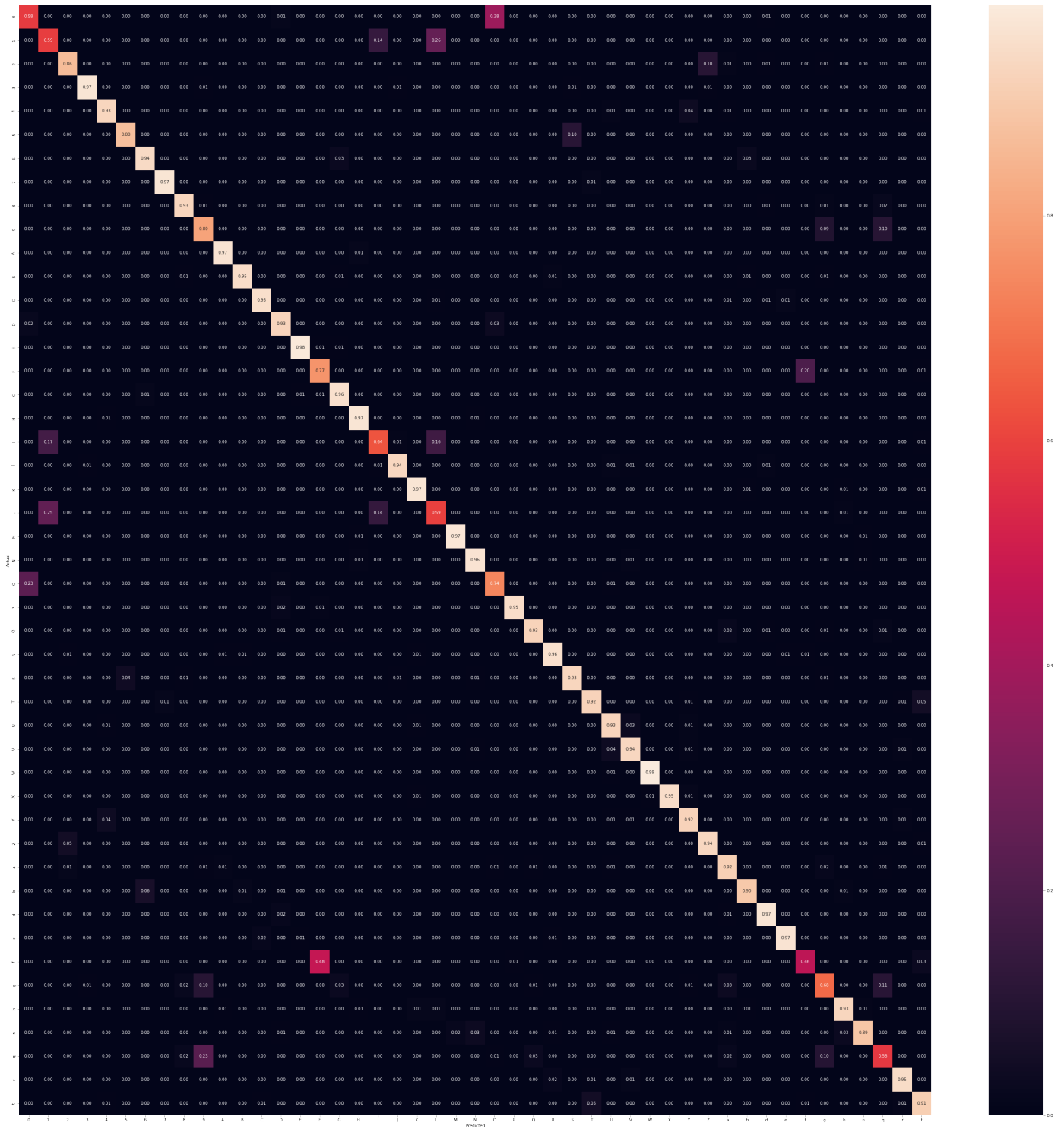
F1-score, precision, recall, and support for each class are discussed. The classification report also allows us to see how each class performed on the training dataset. When compared to other classes, the red marked classes performed significantly worse.

	precision	recall	f1-score	support
0	0.69	0.58	0.63	400
1	0.58	0.59	0.58	400
2	0.92	0.86	0.89	400
3	0.97	0.97	0.97	400
4	0.94	0.93	0.93	400
5	0.94	0.88	0.91	400
6	0.92	0.94	0.93	400
7	0.97	0.97	0.97	400
8	0.94	0.93	0.94	400
9	0.69	0.80	0.74	400
A	0.96	0.97	0.96	400
B	0.97	0.95	0.96	400
C	0.96	0.95	0.96	400
D	0.92	0.93	0.92	400
E	0.98	0.98	0.98	400
F	0.60	0.77	0.68	400
G	0.91	0.96	0.93	400
H	0.95	0.97	0.96	400
I	0.68	0.64	0.66	400
J	0.96	0.94	0.95	400
K	0.96	0.97	0.97	400
L	0.57	0.59	0.58	400
M	0.97	0.97	0.97	400
N	0.93	0.96	0.95	400
O	0.63	0.74	0.68	400
P	0.97	0.95	0.96	400
Q	0.94	0.93	0.93	400
R	0.96	0.96	0.96	400
S	0.87	0.93	0.90	400
T	0.92	0.92	0.92	400
U	0.91	0.93	0.92	400
V	0.92	0.94	0.93	400
W	0.99	0.99	0.99	400
X	0.98	0.95	0.97	400
Y	0.90	0.92	0.91	400
Z	0.88	0.94	0.91	400
a	0.88	0.92	0.89	400
b	0.94	0.90	0.92	400
d	0.95	0.97	0.96	400
e	0.96	0.97	0.97	400
f	0.67	0.46	0.55	400
g	0.73	0.68	0.70	400
h	0.93	0.93	0.93	400
n	0.96	0.89	0.92	400
q	0.68	0.58	0.62	400
r	0.94	0.95	0.95	400
t	0.88	0.91	0.90	400
accuracy			0.88	18800

Observation : Classes like **[0 , 1, 9, F, I, L, O, f , g, q]** have low precision, recall and F1 scores. It means our model is struggling to classify these class images.

Confusion Matrix :

Classification report insights provided a clearer picture of which classes classification by model are performing poorly. The **confusion matrix** provided a better understanding of which classes our model was confusing.



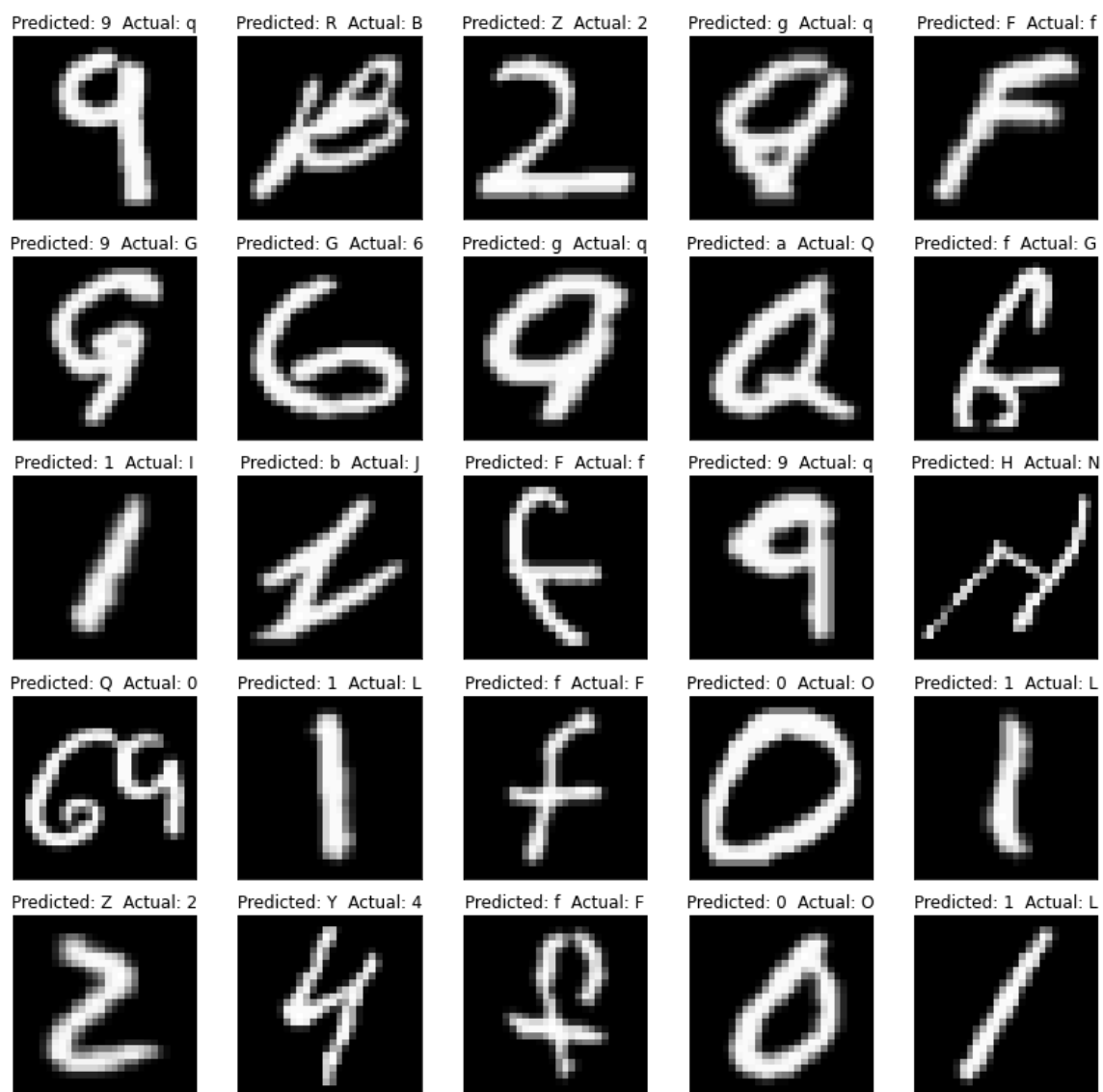
Observation from confusion matrix:

		Original Classes	Misclassified Classes(% percentage of total classification)	
Digits		0	O [uppercase] (38%)	
		1	I [uppercase] (14%)	L [uppercase] (26%)
		9	g [lowercase] (9%)	q [lowercase] (10%)
Alphabets	Uppercase	F	f [lowercase] (20%)	
		I	1 [digit] (17%)	L [uppercase] (16%)
		L	1 [digit] (25%)	I [uppercase] (14%)
		O	0 [digit] (23%)	
	Lowercase	f	F [uppercase] (48%)	
		g	9 [digit] (10%)	q [lowercase] (11%)
		q	9 [digit] (23%)	

The above table shows how some class labels are misclassified more frequently than others. The percentages represent the percent of total classifications that the model incorrectly classified.

Visualise wrong predicted images :

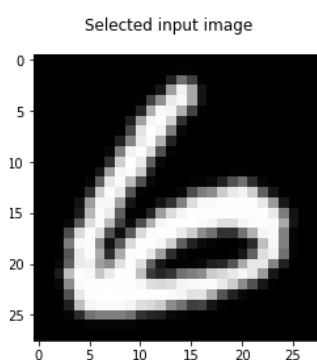
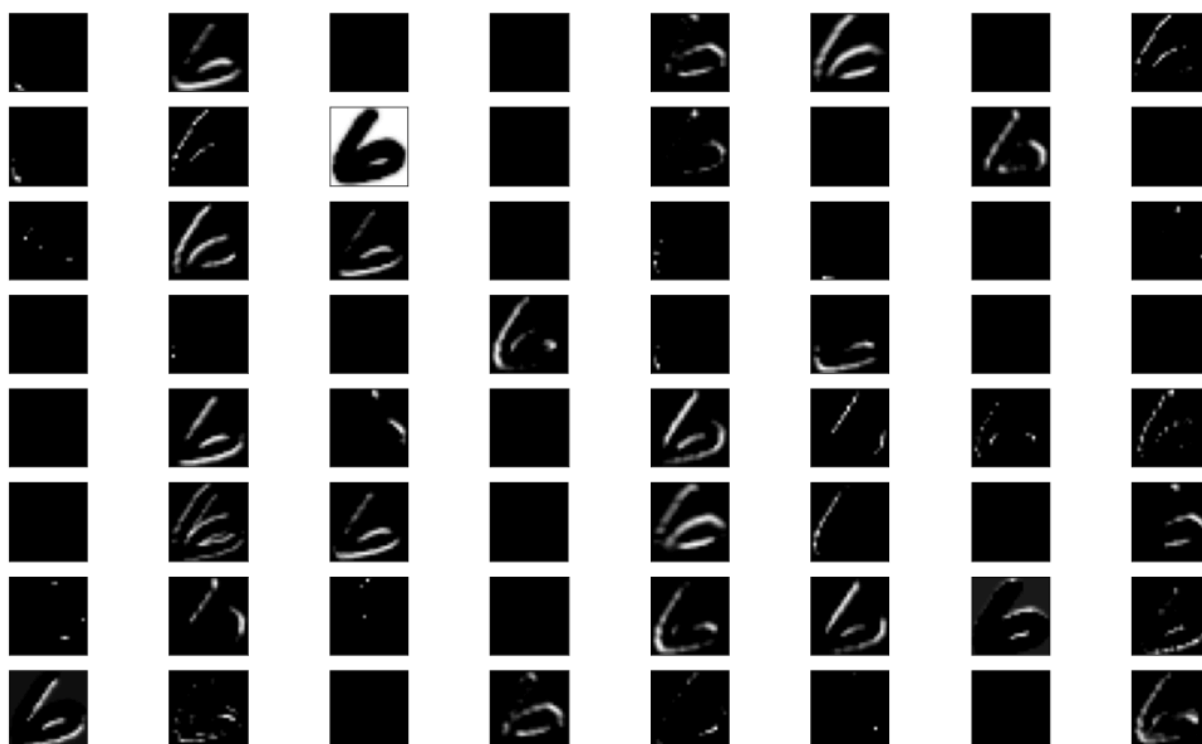
We can also visualise misclassified images to get a better understanding of why the model is failing in these situations. As we can see, writers write alphabets and numbers in a variety of ways, which creates ambiguity when analysing.



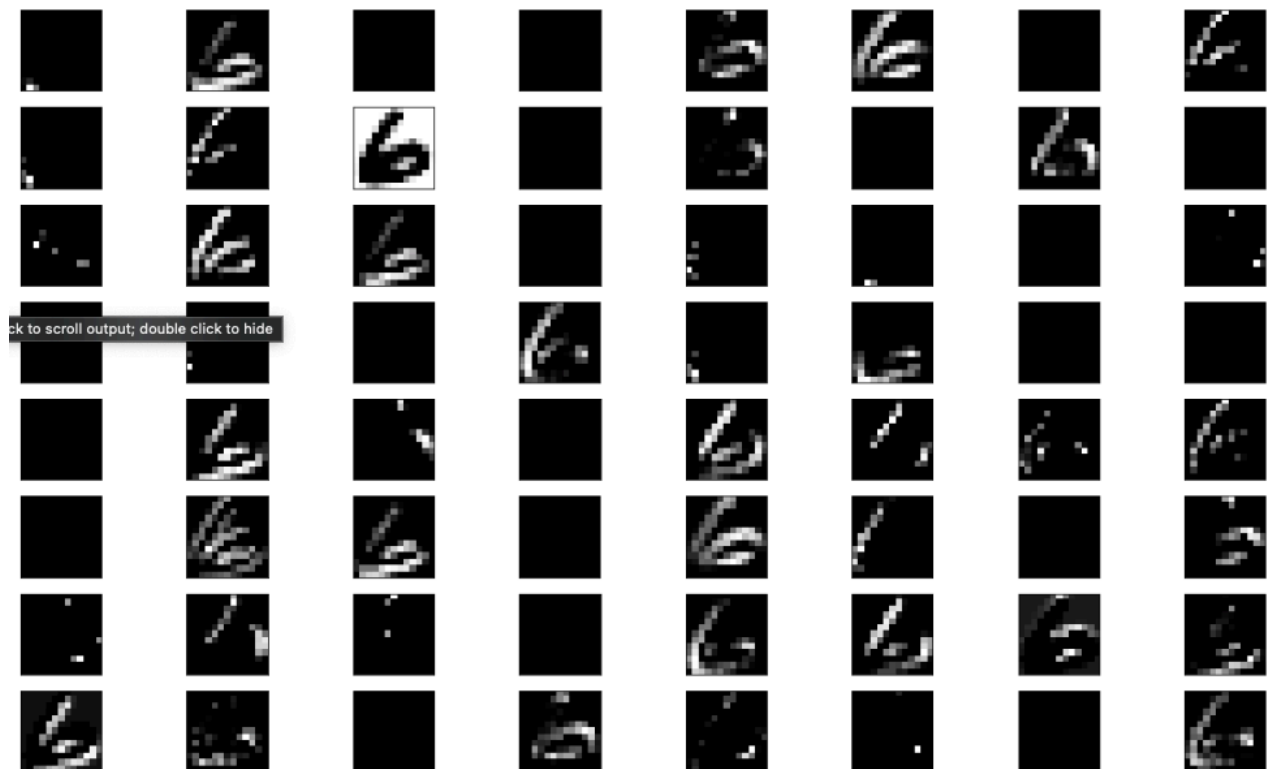
Activation Layer-wise Visualization :

We chose Model 2, which has a 64-filter architecture with a 3x3 kernel size. We chose a random input image for this analysis, which is shown below as "6" digits six. The images were then passed on to our trained model, which activated each layer. Each layer has an activation function that kicks in after the layer operation is completed. The model 2 is made up of five layers. We can visualise what the model has learned and the images after each layer.

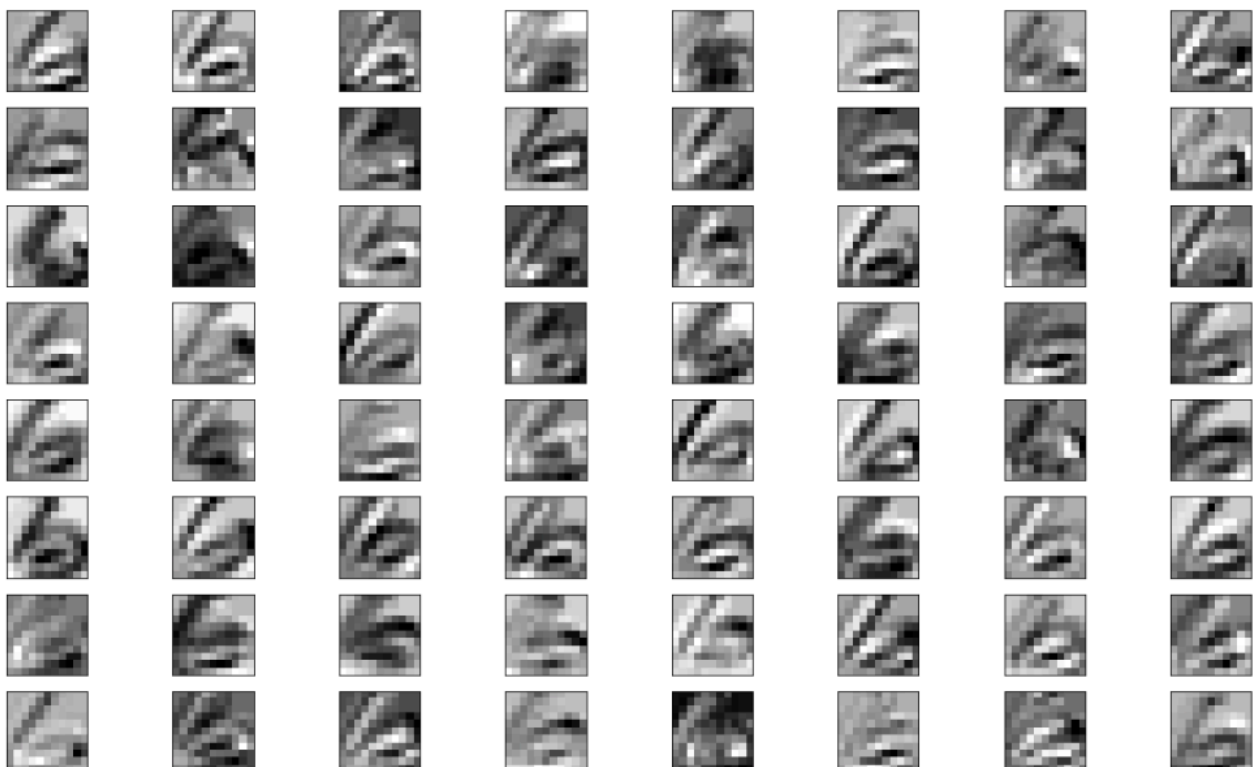
Note : Images with all black pixels represent filters doesn't learn any significant features in the images.

Input image :**Layer 1 :**

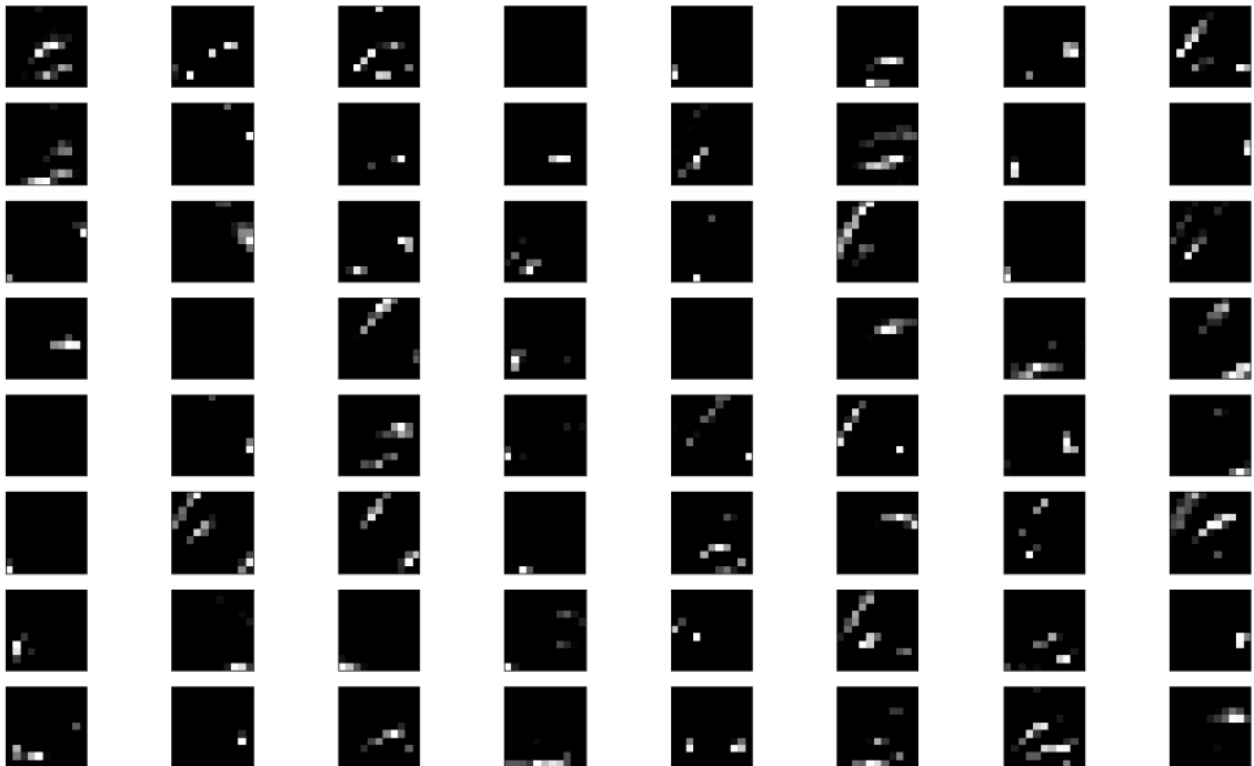
Layer 2 :



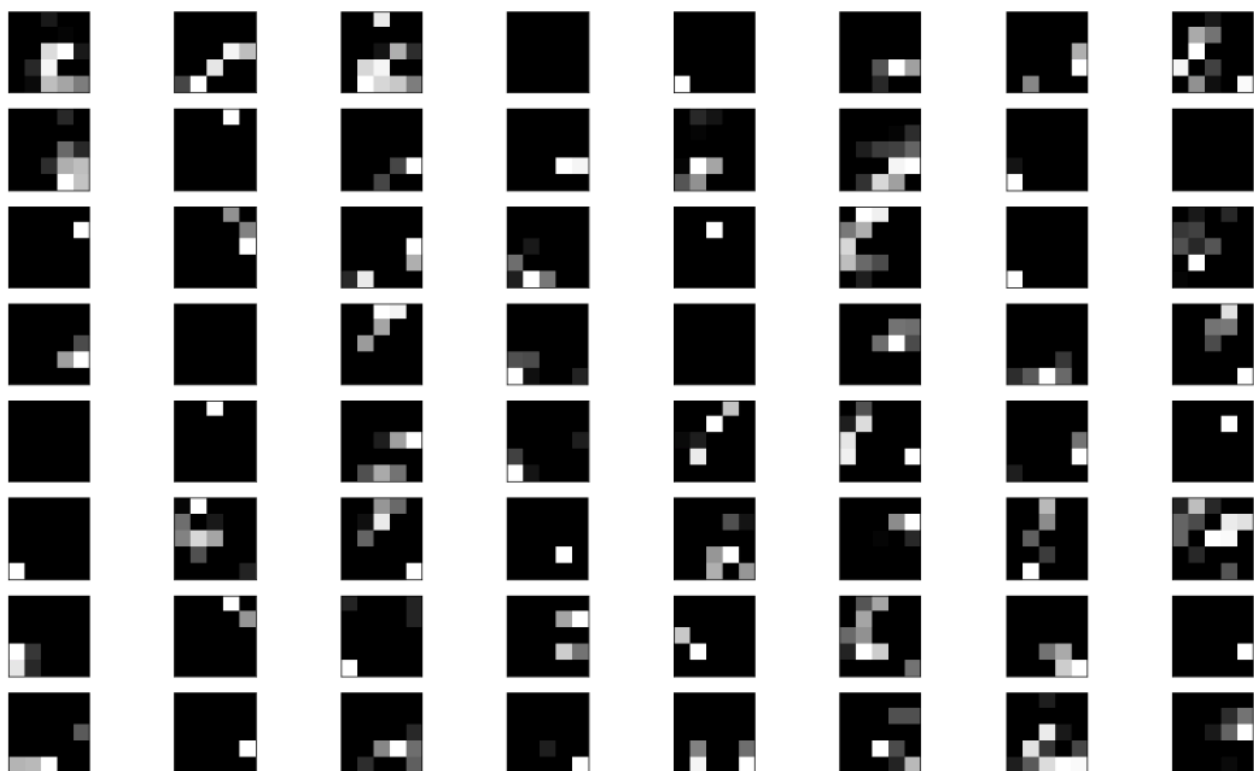
Layer 3 :



Layer 4 :



Layer 5 :



Baseline :

Below table is accuracy mentioned on original EMNIST dataset by Linear classifier and OPIUM Classifier.

Resource : Cohen, Gregory & Afshar, Saeed & Tapson, Jonathan & van Schaik, André. (2017). EMNIST: an extension of MNIST to handwritten letters.

TABLE III
SUMMARY OF THE RESULTS FOR THE LINEAR AND OPIUM-BASED CLASSIFIERS ON THE EMNIST DATASET.

	Linear Classifier	OPIUM Classifier
Balanced	50.93%	78.02% \pm 0.92%
By Merge	50.51%	72.57% \pm 1.18%
By Class	51.80%	69.71% \pm 1.47%
Letters	55.78%	85.15% \pm 0.12%
EMNIST MNIST	85.11%	96.22% \pm 0.14%
Digits	84.70%	95.90% \pm 0.40%

Technique	By_Class	By_Merge	Balanced	Letters	Digits
DWT-DCT + SVM [71]	–	–	–	89.51%	97.74%
Linear classifier [67]	51.80%	50.51%	50.93%	55.78%	84.70%
OPIUM [67]	69.71%	72.57%	78.02%	85.15%	95.90%
SVMs (one against all + sigmoid) [86]	–	–	–	–	98.75% *
Multi-layer perceptron [88]	–	–	–	–	98.39% *
Hidden Markov model [91]	–	–	–	90.00% *	98.00% *
Record-to-record travel [89]	–	–	–	93.78% *	96.53% *
PSO + fuzzy ARTMAP NNs [87]	–	–	–	–	96.49% *
Multi-layer perceptron [90]	–	–	–	87.79% *	–
CNN (6 conv + 2 dense) [85]	–	–	90.59%	–	99.79%
Markov random field CNN [73]	87.77%	90.94%	90.29%	95.44%	99.75%
TextCaps [76]	–	–	90.46%	95.36%	99.79%
CNN (2 conv + 1 dense) [75]	87.10%	–	–	–	–
Committees of neuroevolved CNNs [64]	–	–	–	95.35%	99.77%
Deep convolutional ELM [78]	–	–	–	–	99.775%
Parallelized CNN [74]	–	–	–	–	99.62%
CNN (flat; 2 conv + 1 dense) [79]	–	–	87.18%	93.63%	99.46%
EDEN [80]	–	–	–	88.30%	99.30%
Committee of 7 CNNs [19]	88.12% *	–	–	92.42% *	99.19% *

Above table is Side-by-side comparison of the results for the EMNIST dataset, including works using similar datasets from NIST Special Database 19. Best results are boldfaced. Results marked with a star (*) indicate that they refer to works using samples from NIST SD 19 which are similar but not equivalent to EMNIST.

Resource : Baldominos, A.; Saez, Y.; Isasi, P. A Survey of Handwritten Character Recognition with MNIST and EMNIST. *Appl. Sci.* **2019**, *9*, 3169. <https://doi.org/10.3390/app9153169>

Results :

Train Accuracy

	Byclass	Bymerge	Balanced	Letters	Digits	MNIST
Model 1	91.10%	88.08%	97.34%	98.6%	99.65%	99.71%
Model 2	89.30%	91.67%	96.70%	98.57%	100%	99.99%

Note: The accuracy of the train depends on the number of epochs that the model has run, and we used an **Early Stopping Callback**. As a result, different model and dataset combinations have produced different train accuracy at different epochs.

Validation Accuracy

	Byclass	Bymerge	Balanced	Letters	Digits	MNIST
Model 1	86.131%	85.489%	86.277%	92.813%	99.062%	98.51%
Model 2	86.783%	87.061%	88.019%	94.279%	99.579%	99.225%

Test Accuracy

	Byclass	Bymerge	Balanced	Letters	Digits	MNIST
Model 1	86.08%	85.515%	86.16%	92.889%	99.127%	98.53%
Model 2	86.781%	90.301%	88.292%	93.928%	99.568%	99.46%

Updates baseline table with our model values shown below:

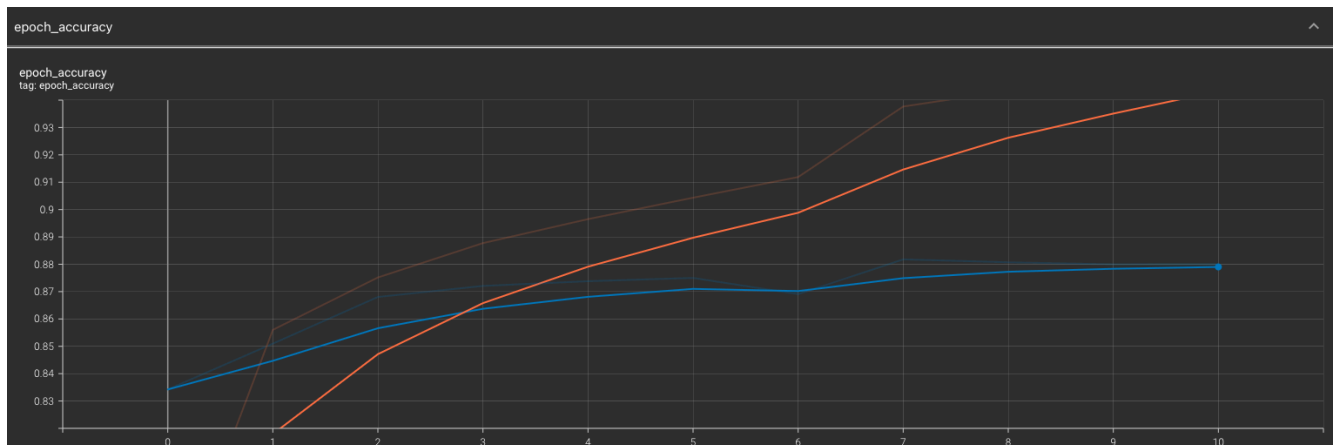
Technique	By_Class	By_Merge	Balanced	Letters	Digits
DWT-DCT + SVM	-	-	-	89.51%	97.74%
Linear classifier	51.80%	50.51%	50.93%	55.78%	84.70%
OPIUM	69.71%	72.57%	78.02%	85.15%	95.90%
SVMs (one against all + sigmoid)	-	-	-	-	98.75% *
Multi-layer perceptron	-	-	-	-	98.39%
Hidden Markov model	-	-	-	90.00% *	98.00%
Record-to-record travel	-	-	-	93.78% *	96.53%
PSO + fuzzy ARTMAP NNs	-	-	-	-	96.49% *
Multi-layer perceptron	-	-	-	87.79% *	-
CNN (6 conv + 2 dense)	-	-	90.59%	-	99.79%
Markov random field CNN	87.77%	90.94%	90.29%	95.44%	99.75%
TextCaps			90.46%	95.36%	99.79%
CNN (2 conv + 1 dense)	87.10%	-	-	-	-
Committees of neuroevolved CNNs	-	-	-	95.35%	99.77%
Deep convolutional ELM	-	-	-	-	99.775%
Parallelized CNN	-	-	-	-	99.62%
CNN (flat; 2 conv + 1 dense)	-	-	87.18%	93.63%	99.46%
EDEN	-	-	-	88.30%	99.30%
Committee of 7 CNNs	88.12% *	-	-	92.42% *	99.19% *
Model1 - Simple - 1 CNN Layer	86.08%	85.52%	86.16%	92.89%	99.13%
Model2 - Complex - 2 CNN Layers	86.78%	90.30%	88.29%	93.93%	99.57%

Plots:

Training Line —

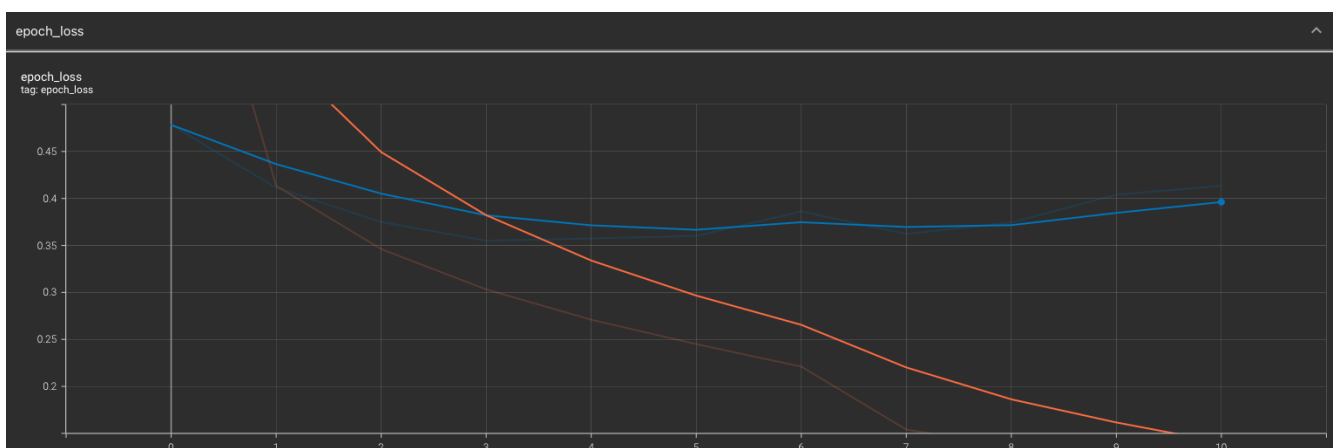
Validation line —

Epochs vs Accuracy



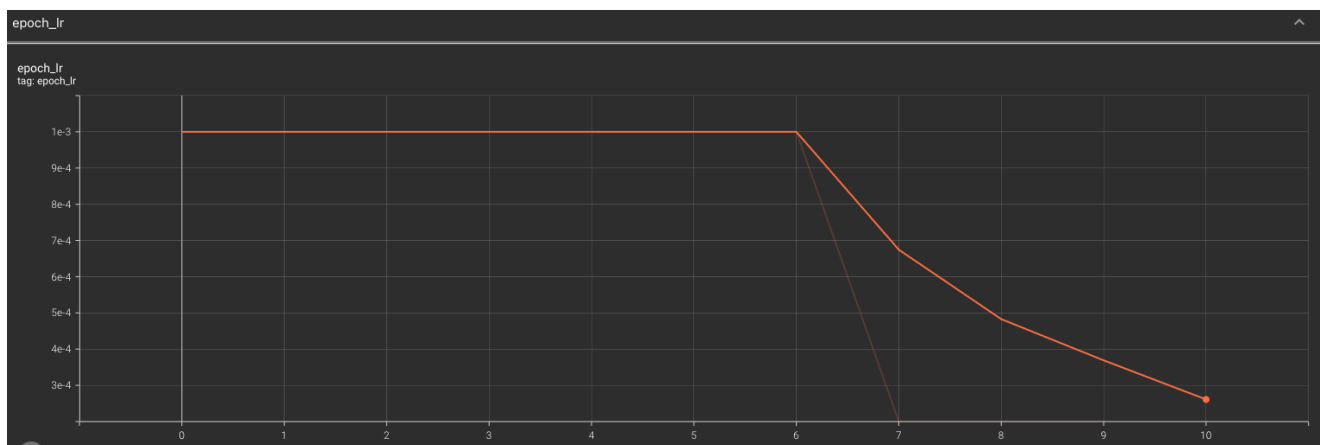
Observation : Validation accuracy increases gradually until 8 epochs, then flattens for 8+ epochs, whereas training accuracy increases gradually as epochs increase. Because validation accuracy did not increase with a significant threshold value, training was stopped after 10 epochs.

Epochs vs Loss



Observation : Training loss decreases as epoch increases almost to zero, whereas validation accuracy decreases gradually until 4 epochs, then flattens until 8 epochs, before increasing slightly again, indicating that this effect is most likely due to learning rate over shooting problem.

Epochs vs Learning Rate



Observation : The graph of epochs vs. learning rate displays the LR value per epoch. We chose the "Reduce on Plateau" callback function to help reduce the learning rate if validation loss does not decrease and learning reaches saturation. The learning rate of 0.001 was used for the first 6 epochs, then gradually decreased per epoch.

Conclusion :

Because the input images had very few features, both CNN models performed very well, as expected. Some classes, such as uppercase alphabet 'O' and digit zero '0,' uppercase 'F' and lower case 'f,' uppercase alphabet 'l' and digit '1,' and others mentioned in the preceding sections, had inherit ambiguity. Some of the key points discovered during the analysis are as follows:

1. The results were similar for models with two convolutional layers and one convolutional layer. They achieved 88.29 percent and 86.16 percent accuracy for the "Balanced" dataset, respectively. However, when we use six convolutional layers plus two dense layers, the accuracy is only 90.59 percent, which is only a 2% improvement. As a result, using fewer convolutional layers allows for better learning of image features. However, it is the inbuilt dataset inaccuracy caused by authors' inconsistent writing of the same letters and digits, as well as the ambiguity in alphabets and digits.
2. When alphabets and digits are trained separately, accuracy is around 95 percent and 99 percent, respectively. As a result, we can implement separate classifiers for both classes and use them to build a robust OCR system.
3. At current best model for EMNIST 'balanced' dataset is VGG-5(Spinal FC) with 91.05% accuracy. [<https://paperswithcode.com/paper/spinalnet-deep-neural-network-with-gradual-1>]
4. Slightly higher accuracy maybe possible with different combination of validation data and test data splits.

Note : All the trained models best weights in .h5 format, confusion matrix and tensorboard logs over various dataset are present in [GitHub](https://github.com/Prakhar-Bhartiya/EMNIST_evaluation_CSE598_Project) with Code. [https://github.com/Prakhar-Bhartiya/EMNIST_evaluation_CSE598_Project]

References:

- ◆ <https://keras.io/api/>
- ◆ <https://keras.io/api/optimizers/>
- ◆ https://keras.io/examples/vision/captcha_ocr/
- ◆ https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
- ◆ <https://www.kaggle.com/achintyatripathi/emnist-letter-dataset-97-9-acc-val-acc-91-78>
- ◆ <https://www.kaggle.com/crawford/emnist>
- ◆ <https://www.kaggle.com/amarjeet007/visualize-cnn-with-keras>
- ◆ https://www.tensorflow.org/tensorboard/scalars_and_keras
- ◆ <https://intellipaat.com/community/368/how-to-interpret-loss-and-accuracy-for-a-machine-learning-model>
- ◆ <https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/>
- ◆ https://en.wikipedia.org/wiki/Confusion_matrix
- ◆ https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
- ◆ <https://stackoverflow.com/questions/39770376/scikit-learn-get-accuracy-scores-for-each-class>

Libraries :

- Tensorflow v2.0
- PyPI Emnist dataset
- Matplotlib
- Jupiter Notebook
- Keras
- Pandas
- Numpy
- Pillow
- Scikit learn
- Seaborn
- Tqdm
- Scipy