Deep Learning with Pytorch

By

Kachinga Silwimba (kachingasilwimba@u.boisestate.edu)

Boise State University (BSU), Idaho, USA

November 2022

Deep Learning Class-Project ECE-662

1. Handwritten Pattern Recognition

This project presents the implementation process using a Convolutional Neural Network with Pytorch to classify the EMNIST-balanced handwritten dataset.

1.1 Problem Statement

Designing a Convolutional Neural Network (CNN) to classify the EMNIST handwritten balanced dataset images using PyTorch. EMNIST balanced dataset is considered one of the most challenging benchmark datasets in pattern recognition and classification (Zhu, 2018).

1.2 Background

According to Cohen et al. (2017), OPIUM-based classifiers with varying hidden layer size was trained on the entire EMNIST dataset. The maximum mean accuracy on the classification for the twenty trials of the network was 78.02%. However, a linear classifier trained achieved a classification accuracy of only 50.93%. A comparison was made by training identical network structures with the same weights and hidden layer size on the original EMNIST datasets using the OPIUM-based classifier. The classification accuracy achieved on the EMNIST dataset scored an accuracy of approximately 78%. These results show how the EMNIST dataset provides a network classification challenge. Additionally, EMNIST database has been utilized in certain application studies. As an illustration, the dataset has also been used by Shu et al. (2018) for pairwise classification and Neftci et al. (2017) for training neuromorphic deep neural networks (CNN). Moreover, when evaluating deep learning networks, EMNIST has achieved remarkable success. A recent research by Botalb et al. (2018) compared a multi-layer perceptron with CNNs in EMNIST and provides evidence of this. Although the test set utilized by the authors was not standard, they reported a maximum accuracy of 89.47% when using a multi-layer perceptron and an accuracy of 99.2% when using a CNN.

1.3 Project Ideas and Approaches

This section lists some of the techniques we used to design the network.

(i) Design and implement a Convolution Neural Network (CNN) architecture using EMNIST balanced dataset and apply various hyper-parameter tuning techniques to the CNN.

(ii) Construct and train networks containing one and two convolutional layers and max-pooling layers using the Balanced EMNIST data and investigate it using different feature maps in the convolutional layers.

(v) Use the dropout in the CNN layers and try different activation functions, optimizers, and also preprocessing of the EMNIST balanced dataset (e.g., normalization).

(vi) Compare the performance of the convolutional neural network with other image classification algorithms on the EMNIST dataset and also create CNN models with varying depth (number of convolutional layers) to observe performance.

(v) Using small convolutional kernels, fully connected layers, and a minimum number of filters per convolutional layer without impairing classification performance.

1.4 Data Description

The EMNIST balanced dataset consists of the Digits 0 to 9, and the letter of the English alphabet it has a total of **47 (balanced) Classes** as shown in Figure 1.1. The dataset contains a total **131,600 images** with **112,800 images for training** and **18,800 images for testing** (Cohen et al., 2017). The creators of EMNIST purposefully designed the database to be structurally compatible with MNIST. As a result, they used a comparable processing method, which includes the following steps: Storing original images in NIST SD 19 as 128×128-pixels black and white images, applying a Gaussian blur filter to soften edges, removing empty padding to reduce the image to the region of interest, centering the digit in a square box preserving the aspect ration, adding blank padding of 2-pixels per side (Baldominos et al., 2019).

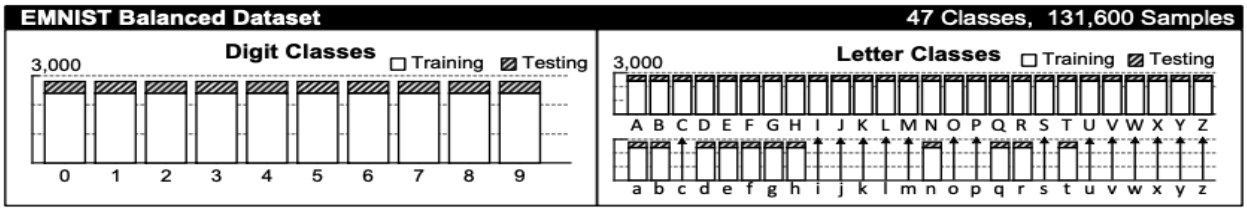

Figure 1.1: Visual breakdown of the EMNIST balanced dataset. The training and testing split for each class is shown using either solid or hatched portions of the bar graphs (Cohen et al., 2017).

1.5 Training and Validation of the Neural Network

In this project, we train, validate and test our model using the balanced EMNIST dataset as given in Figure 1.1 above (Cohen et al., 2017). The dataset of **112,800 images** was subdivided into **100,000 training images**, **12,800 validation images** there were also **18,800 test images**. Table 1.1 and Table 1.2 below shows the summary of independent trials of the two different network architecture for implementing the CNN using different parameters, activation functions, dropout, and pooling layers to select the best CNN architecture with the best classification validation accuracy on the balanced EMNIST dataset. The CNN algorithm is implemented using the Pytorch library. In this project, we tried various network architectures with a different number of convolutional and fully connected layers. We observed that increasing the convolutional layers above two didn't improve the performance of our model design.

1.5.1 Convolutional Neural Network Architecture 1. The architecture followed here is two convolution layers with a pooling layer, two fully connected layers, and a dropout layer.

```
Model(
  (conv_1): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv_2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (max_pool2d): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=1568, out_features=32, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=32, out_features=47, bias=True)
```

```
(relu): ReLU())
```

Table 1.1 shows the Hyper-Parameters and selected **7 trials** during the fitting process for The first CNN network architecture that we designed. The structure of the network was two convolutional layers and two fully connected layers, as shown above.

Table 1.1: Training Hyper-parameters for the CNN architecture selected

Parameters	**Trial-1**	**Trial-2**	**Trial-3**	**Trial-4**	**Trial-5**	**Trial-6**	**Trial-7**
v Learning rate	0.001	0.001	0.001	0.01	0.0001	0.001	0.001
Batch size	10	30	100	256	190	100	230
Epochs	5	5	5	5	5	5	5
conv_1 (In_channels=)	1	1	1	1	1	1	1
conv_1 (kernel_size=)	3	3	3	3	3	3	3
conv_1 (Out_channels=)	32	40	48	56	64	72	80
conv_1 (stride=)	1	1	1	1	1	1	1
conv_1 (padding=)	1	1	1	1	1	1	1
conv_2 (In_channels=)	32	40	48	56	64	72	80
conv_2 (kernel_size=)	3	3	3	3	3	3	3
conv_2 (Out_channels=)	64	80	88	96	104	112	120
conv_2 (stride=)	1	1	1	1	1	1	1
conv_2 (padding=)	1	1	1	1	1	1	1
max_pool2d (kernel_size=)	2	2	2	2	2	2	2
max_pool2d (stride=)	2	2	2	2	2	2	2
fc1(in_features=)	3136	3920	4312	4704	5096	5488	5880
fc1(out_features=)	128	192	256	320	384	448	512
fc2(in_features=)	128	192	256	320	384	448	512
fc2(out_features=)	47	47	47	47	47	47	47
Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
Dropout	0.3	0.5	0.4	0.2	0.1	0.5	0.5
Shuffle	True	True	True	True	True	True	True
Cross entropy	Adam	Adam	Adam	Adam	Adam	Adam	Adam
Validation accuracy	53%	66%	71%	62%	42%	74%	78%

1.5.2 Convolutional Neural Network Architecture 2. We designed this network with two convolutional layers and two fully connected layers. Moreover, we added the pooling layer, batch normalization, and the ReLU activation function between each convolutional layer added the dropout layer between the two fully connected layers and tried different hyper-parameters. We recorded the hyper-parameter values and the validation accuracy for the **7 trials** selected during the training process. This network was designed based on the results from https://www.kaggle.com/code/cdeotte/how-to-choose-cnn-architecture-mnist/notebook. The designed network comprised two convolutional layers and two fully connected layers. Additionally, batch normalization after each convolutional layer was added to speed up the learning process, and also batch normalization lessens the network's internal covariate shift, which impacts the learning. A change in data distribution is the covariate shift (Ioffe and Szegedy, 2015). For this framework, each convolution layer is followed by a max-pool layer and the ReLU activation function. Non-linearity is introduced through ReLU because images are naturally non-linear, and noise is reduced with the aid of max-pooling.

```
EmnistModel(
  (conv1): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm1):BatchNorm2d(8,eps=1e-05, momentum=0.1,affine=True,track_running_stats=True)
  (relu): ReLU()
  (maxpool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(8, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (batchnorm2):BatchNorm2d(32,eps=1e-05,momentum=0.1, affine=True,track_running_stats=True)
  (maxpool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=1568, out_features=600, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=600, out_features=47, bias=True))
```

Table 1.2 shows the number of trials for the second designed CNN network architecture with hyper-parameters during the training process. The structure of the network was two convolutional layers and two fully connected layers, as shown above.

Table 1.2: Training Hyper-parameters for the CNN above

Parameters	**Trial-1**	**Trial-2**	**Trial-3**	**Trial-4**	**Trial-5**	**Trial-6**	**Trial-7**
Learning rate	0.001	0.001	0.001	0.01	0.001	0.001	0.001
Batch size	290	300	100	256	190	100	230
Epochs	20	20	20	20	20	20	20
conv_1 (In_channels=)	1	1	1	1	1	1	1
conv_1 (kernel_size=)	3	3	3	3	3	3	3
conv_1 (Out_channels=)	8	16	24	32	40	48	56
conv_1 (stride=)	1	1	1	1	1	1	1
conv_1 (padding=)	1	1	1	1	1	1	1
max_pool2d (kernel_size=)	2	2	2	2	2	2	2
max_pool2d (stride=)	2	2	2	2	2	2	2
conv_2 (In_channels=)	8	16	24	32	40	48	56
conv_2 (kernel_size=)	5	5	5	5	5	5	5
conv_2 (Out_channels=)	32	40	48	56	64	72	80
conv_2 (stride=)	1	1	1	1	1	1	1
conv_2 (padding=)	2	2	2	2	2	2	2
max_pool2d (kernel_size=)	2	2	2	2	2	2	2
max_pool2d (stride=)	2	2	2	2	2	2	2
fc1(in_features=)	1568	1960	2352	2744	3136	3528	3920
fc1(out_features=)	600	500	256	320	384	448	512
Dropout	0.5	0.5	0.4	0.2	0.1	0.5	0.5
fc2(in_features=)	600	500	256	320	384	448	512
fc2(out_features=)	47	47	47	47	47	47	47
Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
Shuffle	True	True	True	True	True	True	True
Cross entropy	Adam	Adam	Adam	Adam	Adam	Adam	Adam
Validation accuracy	88.7%	86%	84%	87%	78%	82%	78%

Finally, from the training Table 1.5, we selected **Trial 1** because it gave better validation accuracy of **88.7** compared to other trials and minimized the loss function by giving the smallest loss score. Most importantly, our selected network architecture is not overfitting. Hence, the framework will generalize well to unseen EMNIST-balanced (test) datasets.

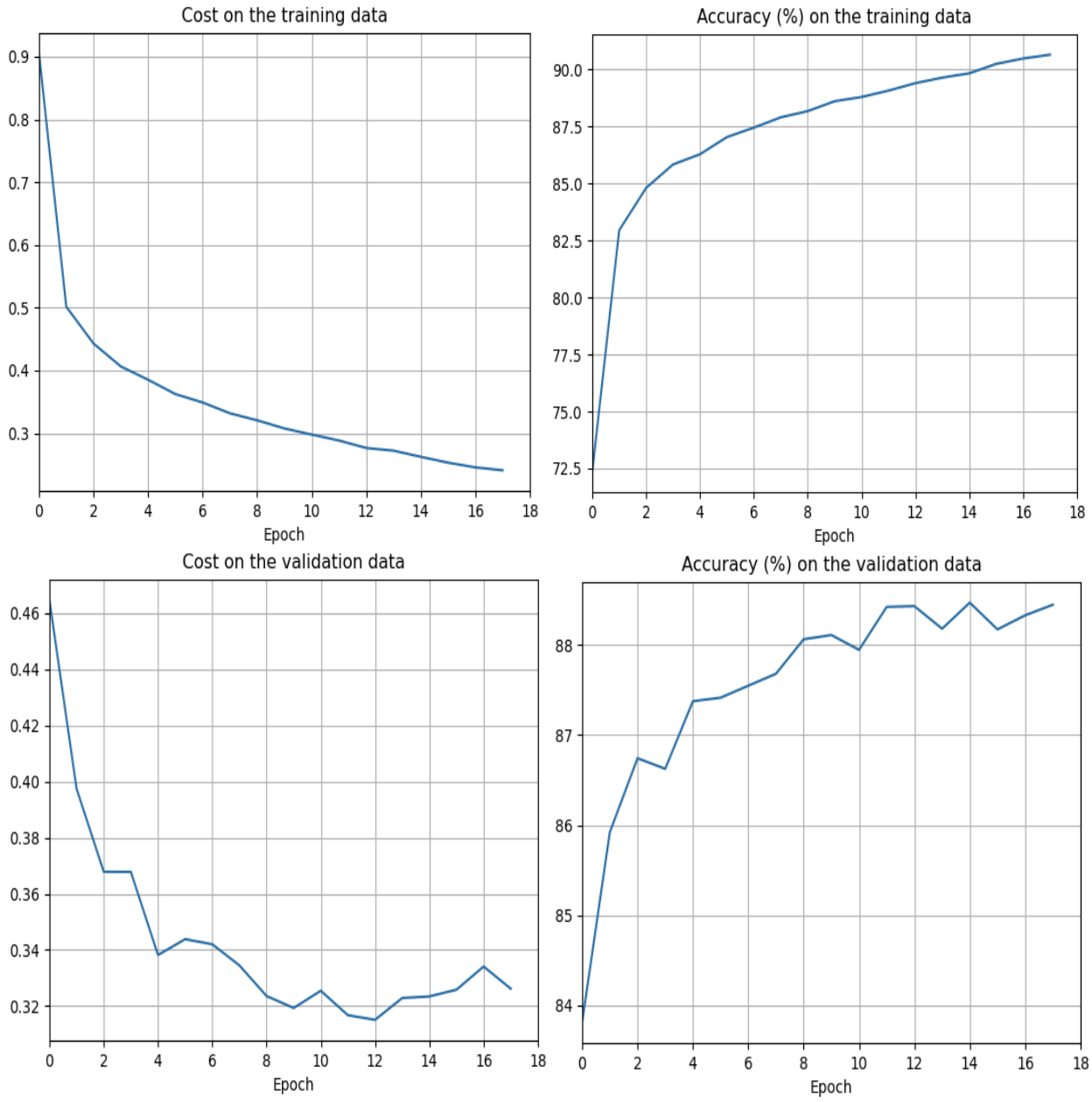

Figure 1.2: Figures show the validation and training cost and also validation and training accuracy

1.6 Final Network Architecture

The final CNN architecture consists of three convolutional layers and three fully connected layers. Table 1.4 shows the full model summary of our final proposed CNN architecture. The network used **Adam** (Kingma and Ba, 2014) optimizer to update the weights and used the **ReLU** activation function, and Table 1.3 shows the parameters used. Moreover, a **cross-entropy** as our loss function was used to train the network. The final model architecture is dependent on some training parameters shown in Table 1.3. Additionally, a **dropout rate** of 50% was used for regularization to prevent the CNN from overfitting on the fully connected layer (`fc1`). Additionally, batch normalization was added to avoid the covariate shift, which impacts the learning process of the neurons.

```
EmnistModel(
  (conv1): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm1): BatchNorm2d(8,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True)
  (relu): ReLU()
  (maxpool1): MaxPool2d(kernel_size=2,stride=2,padding=0, dilation=1,ceil_mode=False)
  (conv2): Conv2d(8, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (batchnorm2): BatchNorm2d(32,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True)
  (maxpool2): MaxPool2d(kernel_size=2,stride=2,padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=1568, out_features=600, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=600, out_features=47, bias=True))
```

The first convolutional layer (`Conv1`) takes in a channel of 1 dimension, and the kernel size of 3×3 with a stride of 1 and output of this convolutional layer is set to be 16 channels implying that it will extract **8 feature maps** using **8 kernels**. To ensure that the input and output dimensions are identical, we pad the image using a padding size of 1. The **Output dimension** at this convolutional layer is $\mathbf{8 \times 28 \times 28}$. Then, a max-pooling layer with a kernel size of 2 and a stride of 2 is applied to it. The feature translates to dimensions of $\mathbf{8 \times 14 \times 14}$ as a result of the downsampling.

The second convolution layer (`Conv2`) has **8 input channels**. It extracts **32 feature maps** because we

chose an output channel size of 32 with stride 1. The kernel size for this layer is 5. To keep the input and output dimensions constant, we once more utilize a padding size of 1. This layer's output size is going to be $\mathbf{32} \times \mathbf{14} \times \mathbf{14}$. Then, a max-pooling layer with a kernel size of 2 and a stride of 2 is applied to it. The feature translates to dimensions of $\mathbf{32} \times \mathbf{7} \times \mathbf{7}$ as a result of the downsampling.

Finally, `fc1` and `fc2` two fully connected layers are used. The first fully connected layer (`fc1`) receives a flattened copy of the feature maps. Therefore, it must have a dimension of $\mathbf{32} \times \mathbf{7} \times \mathbf{7}$, or 1568 nodes. This layer joins a 600 node layer that is fully connected (`fc2`). The output dimension matches the total number of classes, which is 47, as this is the last layer. Thus, we have two fully connected layers that are $\mathbf{1568} \times \mathbf{600}$ and $\mathbf{600} \times \mathbf{47}$, respectively. A dropout layer between the two fully connected layers was added to dropout connections to reduce over-fitting with a connection dropout probability of 0.5. Table 1.4 shows a summary of the final network architecture selected, while Table 1.3 shows the final hyper-parameter used for fitting the network on the EMNIST balanced dataset.

Table 1.3: Hyper-parameters of final training CNN model

Parameter Name	**Value**
`Learning rate (lr)`	0.001
`Batch size (batch_size)`	20
`Epochs (num_epochs)`	290
`Dropout (`$\mathcal{P}$`)`	0.5
`Shuffle`	True

Table 1.4: Pytorch model summary of our final CNN architecture

Operational Layer		Input Shape	Kernel Size	Stride Size	Padding	Output Shape
Convolution Layer (`Conv2D_1`)	Covolutional	(1,28,28)	(3,3)	(1,1)	(1,1)	(8,28,28)
Batch Normalization (`BatchNorm2d`)		(8,28,28)	-	-	-	(8,28,28)
Relu (`ReLU`)		(8,28,28)	-	-	-	(8,28,28)
Pooling Layer (`MaxPool2D_1`)	Max pooling	(8,28,28)	2	2	0	(8,14,14)
Convolution Layer (`Conv2D_2`)	Covolutional	(8,14,14)	(5,5)	(1,1)	(2,2)	(32,14,14)
Batch Normalization (`BatchNorm2d`)		(32,14,14)	-	-	-	(32,14,14)
Relu (`ReLU`)		(32,14,14)	-	-	-	(32,14,14)
Pooling Layer (`MaxPool2D_2`)	Max pooling	(32,14,14)	2	2	0	(32,7,7)
Dense Layer (`fcon_1`)	Fully connected	(1568)	-	-	-	(600)
Dropout Layer (`drpout`)	Dropout (p=0.5)	(600)	-	-	-	(600)
Dense Layer (`fcon_2`)	Fully connected	(600)	-	-	-	(47)

1.7 Network Results On Test Dataset

This project aimed to minimize the classification error on the balanced EMNIST dataset using the deep network. The final network above in Table 1.4, with the Adam Optimizer and a learning rate of 0.001, was trained with cross-entropy as our loss function. We obtained approximately $\mathbf{88.43\%}$ or $\mathbf{0.8843}$ accuracy on the test dataset after training the model with 20 epochs. Comparing our test score results with the literature, Anuradha et al. (2019) compared four CNN models, AlexNet, CGGNet, GoogleNet,

and CapsNet, using the balanced EMNIST dataset. The CNNs were shown to achieve the following test accuracy scores: **AlexNet** $\mathbf{96.9}\%$, **VGGNet** $\mathbf{98.3}\%$, **GoogleNet** $\mathbf{98.9}\%$, **CapsNet** $\mathbf{99.7}\%$. The CapsNet gave a better performance compared to the other three CNNs. Moreover, Vaila et al. (2020) used the CNN on the balanced EMNIST dataset and reported the accuracy of $\mathbf{98.54}\%$ and $\mathbf{97.77}\%$. Furthermore, Agarap and Azcarraga (2020) used the K-Means Clustering and Autoencoder fully connected networks Model on the EMNIST balanced dataset. They achieved clustering accuracy of $\mathbf{78.5}\%$ for the **K-Means Clustering** and $\mathbf{36.1}\%$ for the **Autoencoder** on the test dataset. Additionally, Kabir et al. (2022) reported these accuracy results on the EMNIST balanced for the following models default **Pytorch CNN** $\mathbf{79.61}\%$, **VGG-5** $\mathbf{91.04}\%$. Dufourq and Bassett (2017) used the neural architecture for automatically optimizing the hyperparameters of CNN classifiers using a neuroevolution on **EDEN**. They reported an accuracy of $\mathbf{88.3}\%$ in EMNIST Balanced and 99.3% in EMNIST Digits.

Additionally, the EMNIST database has also been used in some application papers. For example, Shawon et al. (2018) developed a system using a CNN with six convolutional layers and two fully connected layers for the recognition of Bangla handwritten digits. They also tested the system in EMNIST, reporting accuracy levels of $\mathbf{90.59}\%$ in EMNIST Balanced and $\mathbf{99.79}\%$ in EMNIST Digits. Jayasundara et al. (2019) used the capsule layers (a concept introduced by Sabour et al. (2017)) a CNN with three convolutional layers, and two capsule layers called **TextCaps** achieved accuracy in the EMNIST letters and digits of $\mathbf{95.36}\%$ and $\mathbf{99.79}\%$, respectively. Moreover, by automatically examining a "flat" (regular) classifier's confusion matrix, Cavalin and Oliveira (2018) developed a technique for creating a hierarchical classifier. However, the hierarchical classification did not outperform the flat setting with this method. They evaluated the logistic regression, multi-layer perceptrons, and CNN's, reporting the best results for the CNN. For EMNIST letters, they claim a $\mathbf{93.63}\%$ accuracy rate, and for EMNIST digits, $\mathbf{99.46}\%$. Therefore, our best test classification accuracy of $\mathbf{88.43}\%$ on the test dataset compares well with the literature reviewed.

References

Agarap, A. F. and A. P. Azcarraga (2020). Improving k-means clustering performance with disentangled internal representations. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE.

Anuradha, R., N. Saranya, M. Priyadharsini, and G. D. Kumar (2019). Assessment of extended mnist (emnist) dataset using capsule networks. In *2019 International Conference on Intelligent Sustainable Systems (ICISS)*, pp. 263–266. IEEE.

Baldominos, A., Y. Saez, and P. Isasi (2019). A survey of handwritten character recognition with mnist and emnist. *Applied Sciences 9*(15), 3169.

Bluche, T., H. Ney, and C. Kermorvant (2013). Feature extraction with convolutional neural networks for handwritten word recognition. In *2013 12th international conference on document analysis and recognition*, pp. 285–289. IEEE.

Botalb, A., M. Moinuddin, U. Al-Saggaf, and S. S. Ali (2018). Contrasting convolutional neural network (cnn) with multi-layer perceptron (mlp) for big data analysis. In *2018 International conference on intelligent and advanced system (ICIAS)*, pp. 1–5. IEEE.

Cavalin, P. and L. Oliveira (2018). Confusion matrix-based building of hierarchical classification. In *Iberoamerican Congress on Pattern Recognition*, pp. 271–278. Springer.

Chandel, S. (2021). pytorch-summary. *https://github.com/sksq96/pytorch-summary*.

Cohen, G., S. Afshar, J. Tapson, and A. Van Schaik (2017). Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pp. 2921–2926. IEEE.

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine 29*(6), 141–142.

Dufourq, E. and B. A. Bassett (2017). Eden: Evolutionary deep networks for efficient machine learning. In *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*, pp. 110–115. IEEE.

Ioffe, S. and C. Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR.

Jayasundara, V., S. Jayasekara, H. Jayasekara, J. Rajasegaran, S. Seneviratne, and R. Rodrigo (2019). Textcaps: Handwritten character recognition with very small datasets. In *2019 IEEE winter conference on applications of computer vision (WACV)*, pp. 254–262. IEEE.

Kabir, H. D., M. Abdar, A. Khosravi, S. M. J. Jalali, A. F. Atiya, S. Nahavandi, and D. Srinivasan (2022). Spinalnet: Deep neural network with gradual input. *IEEE Transactions on Artificial Intelligence*.

Kingma, D. P. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*(11), 2278–2324.

MachineCurve (2020). extra keras dataset. *https://github.com/machinecurve/extra*$_k eras_d atasets$.

Neftci, E. O., C. Augustine, S. Paul, and G. Detorakis (2017). Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Frontiers in neuroscience 11*, 324.

Niu, X.-X. and C. Y. Suen (2012). A novel hybrid cnn–svm classifier for recognizing handwritten digits. *Pattern Recognition 45*(4), 1318–1325.

Rajalakshmi, M., P. Saranya, and P. Shanmugavadivu (2019). Pattern recognition-recognition of handwritten document using convolutional neural networks. In *2019 IEEE International Conference on Intelligent Techniques in Control, Optimization and Signal Processing (INCOS)*, pp. 1–7. IEEE.

Sabour, S., N. Frosst, and G. E. Hinton (2017). Dynamic routing between capsules. *Advances in neural information processing systems 30*.

Shawon, A., M. J.-U. Rahman, F. Mahmud, and M. A. Zaman (2018). Bangla handwritten digit recognition using deep cnn for large and unbiased dataset. In *2018 International Conference on Bangla Speech and Language Processing (ICBSLP)*, pp. 1–6. IEEE.

Shu, L., H. Xu, and B. Liu (2018). Unseen class discovery in open-world classification. *arXiv preprint arXiv:1801.05609*.

Vaila, R., J. Chiasson, and V. Saxena (2020). A deep unsupervised feature learning spiking neural network with binarized classification layers for the emnist classification. *IEEE transactions on emerging topics in computational intelligence*.

Zhu, W. (2018). Classification of mnist handwritten digit database using neural network. *Proceedings of the research school of computer science. Australian National University, Acton, ACT 2601*.