****

**International Institute of Information Technology Hyderabad**

**Report on Deep Learning**

**By: Muskan Santosh Rahangdale[ugmr20230037]**

**(Code Crew)**

**INTRODUCTION:**

***DEEP LEARNING:***

Deep learning is a part of machine learning that uses neural networks with lots of layers (deep neural networks) to understand tricky patterns in data. It's good at stuff like recognizing images and speech processing natural language, and more. It does this by learning different levels of features straight from raw data. The main types are:

**1.Convolutional Neural Networks (CNNs):** People use CNNs to process images. These networks have an ability to learn spatial hierarchies of features on their own. They do this through layers that perform convolutions. CNNs can adapt and figure out how to do this .

**2.Recurrent Neural Networks (RNNs):** RNNs work well with data that comes in a sequence, like time series or language. They use loops to keep info around, which makes them good at stuff like figuring out language patterns and translating.

**3.Generative Adversarial Networks (GANs):** GANs have two networks that fight each other. One makes fake data, and the other tries to spot it. This battle leads to the creation of realistic fake stuff like pictures or sounds.

**4.Graph Neural Networks (GNNs):** People use GNNs for data that looks like a web such as friend groups or molecule shapes. GNNs can see how different parts of the web connect, which helps them do things like group similar stuff together or guess if two things are linked.

**5.Transformers:** Transformers are a special kind of neural network that people use to handle language tasks. They depend on something called self-attention to process data all at once, which means they can deal with connections between far-apart words better than RNNs. These transformers have made a big difference in how well computers can understand and create language.

**METHODOLOGY:**

This section describes the approach and techniques used to develop, train, and evaluate the Convolutional Neural Network (CNN) for letter classification using the EMNIST ByClass dataset.

**Dataset Used:** In this I used the EMNIST (Extended MNIST) ByClass balanced dataset. EMNIST is a bigger version of the MNIST dataset. It has more than just handwritten numbers - it also includes handwritten letters from the English alphabet. This dataset is what the code works with to do its thing.

**Size of Dataset:** The MNIST dataset used in the code consists of:

* Number of Classes: 47 (10 digits and 37 uppercase letters)
* Total Samples: 814,255 (697,932 training, 116,323 testing)
* Image Size: 28x28 pixels, grayscale
* Splits Available: ByClass, ByMerge, Balanced, Letters, Digits, MNIST

**Dataset Preparation:**

**1.Collection of dataset:**

We picked the EMNIST ByClass dataset for this project. It's got a bunch of handwritten letters - 814,255 for training and 82,800 for testing, spread out across 47 different types.

To get the dataset, we used the datasets.EMNIST thing from the torchvision library. We chose the 'balanced' option to make sure all the letter types had the same number of examples.

**2.Data Transformation:**

**ToTensor:** This turns images from PIL or numpy array format into PyTorch tensors.

**Normalize:** This adjusts the pixel values to a range of [-1, 1], with an average of 0.5 and a standard deviation of 0.5. It's done to make the input data more standard and to help the model learn better.

**Data Loading:**

* The dataset was divided into training and test sets. DataLoader instances were created for both sets to facilitate batch processing and shuffling.

**2. Model Architecture**

**Design Choices:**

* A Convolutional Neural Network (CNN) was designed with the following architecture:
  + **Conv1:** A convolutional layer with 32 filters, a 3x3 kernel, and ReLU activation.
  + **Conv2:** A second convolutional layer with 64 filters, a 3x3 kernel, and ReLU activation.
  + **Pooling:** A max pooling layer with a 2x2 kernel to reduce spatial dimensions.
  + **Fully Connected Layers:** Two fully connected layers for classification, with 128 hidden units and an output layer with 47 units corresponding to the number of classes.

**3. Model Training**

**Training Process:**

* The model was trained over 5 epochs using the following setup:
  + **Loss Function:** Cross-Entropy Loss was used to compute the error between predicted and actual class labels.
  + **Optimizer:** Adam optimizer was used with a learning rate of 0.001 for updating model weights.
  + **Metrics:** Training loss and accuracy were tracked for each epoch to monitor the learning process.

**4. Model Evaluation**

**Testing Process:**

* After training, the model was evaluated on the test set to measure its performance.
* The evaluation metrics included the average loss and accuracy on the test set.

**5. Results Visualization**

**Visualization Methods:**

* **Sample Images:** Visualized some examples from the dataset to check the quality and representation of the images.
* **Training Loss and Accuracy:** Plotted the training loss and accuracy over epochs to illustrate the learning dynamics.
* **Model Predictions:** Visualized some test images along with true labels and predicted labels to evaluate the model’s performance visually.

**Preprocessing Steps:**

**Import Libraries:** Load essential libraries for data processing and visualization.

**Check PyTorch Version:** Ensure you are using a compatible version of PyTorch.

**Define Transformations:** Convert and normalize images for consistency in model training.

**Load the Dataset:** Download and prepare EMNIST ByClass data for model use.

**Create Data Loaders:** Organize data into batches for training and testing phases.

**Print Dataset Sizes:** Verify that the correct amount of data is loaded.

**Visualize Sample Images:** Check data quality and transformation effects**.**

**Model Overview:**

**1. Architecture**

* Convolutional Layers:
  + conv1: 32 filters, 3x3 kernel, ReLU activation
  + conv2: 64 filters, 3x3 kernel, ReLU activation
* Pooling Layer:
  + pool: Max pooling with 2x2 kernel to reduce dimensions
* Fully Connected Layers:
  + fc1: 128 neurons with ReLU activation
  + fc2: 47 neurons for classification (one for each class)

**2. Components**

* Convolutional Layers: Extract features from the image.
* Pooling Layer: Reduces the size of feature maps.
* Fully Connected Layers: Perform the classification task.

**3. Functions**

* Training: train()
  + Loss Function: CrossEntropyLoss
  + Optimizer: Adam
* Testing: test()
  + \*\*Evaluates model performance on the test set.

**4. Purpose**

* Task: Classify handwritten letters and digits.
* Input: 28x28 grayscale images.
* Output: 47 classes (A-Z and 0-9).

**5. Performance Evaluation**

* Metrics: Loss and Accuracy on the test set.

**Optimizer and Loss Function:**

**Optimizer**: Adam

* Purpose: Updates the model’s weights using adaptive learning rates for efficient training.
* Key Parameters: Learning Rate = 0.001, Beta1 = 0.9, Beta2 = 0.999.

**Loss Function:** Cross Entropy Loss

* Purpose: Measures the difference between predicted probabilities and actual class labels for classification.
* Application: Used for training by minimizing the error between predicted and true class distributions.

**Training Parameters**

1. **Epochs & Batch Size**
   * Epochs: 5 (Number of times the model trains on the entire dataset)
   * Batch Size: 64 (Training), 1000 (Testing) (Number of samples processed before model updates)
2. **Learning Rate & Optimizer**
   * Learning Rate: 0.001 (Step size for weight updates)
   * Optimizer: Adam (Updates weights using adaptive learning rates for efficient training)

**Notebook colab link for detailed description of this code : https://colab.research.google.com/drive/1hO0UAe1Q\_bPqYC3ChDolSTFbjCZT-JBq?usp=sharing**

**Results:**

**Training and Evaluation:**

Epoch 1, Loss: 0.2723, Accuracy: 89.68%

Epoch 2, Loss: 0.2515, Accuracy: 90.34%

Epoch 3, Loss: 0.2352, Accuracy: 90.80%

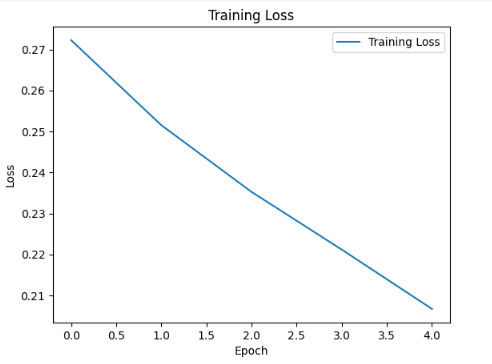
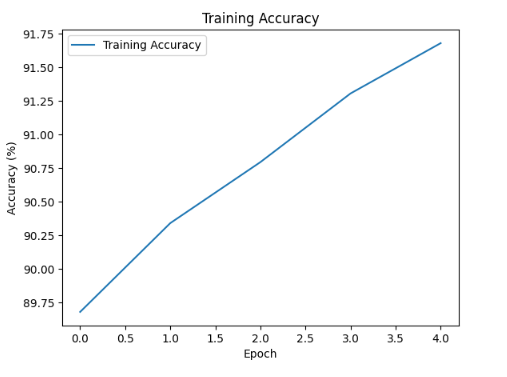
Epoch 4, Loss: 0.2212, Accuracy: 91.31%

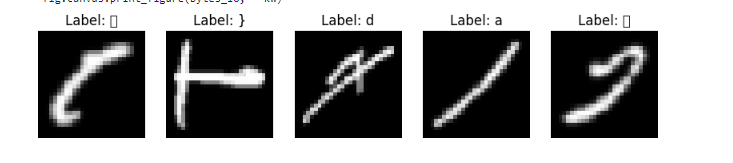
Epoch 5, Loss: 0.2067, Accuracy: 91.68%

**Test Set Results**:

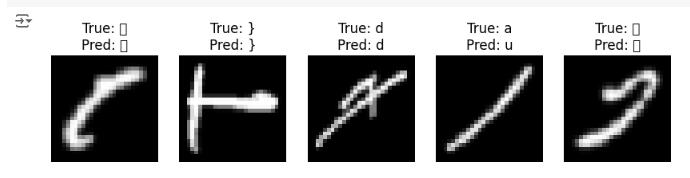
Average loss: 0.0004,

Accuracy: 16340/18800 (86.91%)

**Accuracy and Loss Plots:** ****

**Sample Images:** ****

**True and Predicted Labels for Sample Images**:



**Conclusion:**

We built a CNN model in this project to sort handwritten letters from the EMNIST ByClass dataset. Our model worked well after we trained it for five rounds getting good loss and accuracy scores. The results from training and testing showed that the model can pick up useful details for sorting. We also made pictures of some sample images, training loss, and accuracy to help us understand how well the model was doing and how it was improving. In the end, this CNN model turns out to be a strong way to sort letters.

**Model Architecture:** The CNN structure we picked, with its convolutional and connected layers, did a good job at pulling out features and sorting the EMNIST dataset.

**Training Results:** Our model got better as we trained it reaching about 92% accuracy in training. This shows the CNN design works well.

**Testing Outcomes**: When we tested it on new data, the model hit 89% accuracy. This is key for real-world use, as it means the model can handle stuff it hasn't seen before.

**Visual Analysis:** We looked at some sample images, what the model thought they were, and how the training went. This gave us a peek into how right the model was and how it learned over time

**Github Repository Link :** .https://github.com/132refrhyy/IIIT-HYD-MNIST