****

**International Institute of Information Technology Hyderabad**

**Report on Deep Learning**

**By: Nandini Mandhane [ugmr20230018]**

**(Code Crew)**

***What is Deep Learning?***

Deep learning is a branch of machine learning that models complicated patterns in data using multi-layered neural networks, or deep neural networks. Because it learns hierarchical representations straight from raw input, it does very well in tasks like audio and picture recognition, natural language processing, and more.

Important kinds consist of:  
 1. Convolutional Neural Nets (CNNs): CNNs are mostly employed for image processing applications. They use convolutional layers to automatically and adaptively learn the spatial hierarchies of information.  
2. Recurrent Neural Networks (RNNs): RNNs are useful for tasks like language modeling and translation because they employ loops to allow information to survive, which makes them appropriate for sequential data like time series or spoken language.  
3. Generative Adversarial Networks (GANs): Made up of two networks—a discriminator and a generator—that compete with one another to produce realistic-sounding synthetic data, such audio or pictures.  
4. GNNs, or Graph Neural Networks: GNNs are effective for tasks like node classification and link prediction because they can capture relationships between nodes, which is useful for data that is represented as graphs, such as social networks or chemical structures.  
5. Transformers: Neural network architectures of this sort are mainly employed for applications related to natural language processing. They analyze incoming data in parallel via self-attention processes, which gives them a greater advantage over RNNs for handling long-range dependencies. Translation and language creation have advanced significantly as a result of Transformers.

**Colab Notebook Link:** <https://colab.research.google.com/drive/1KkTGj3Z_aGq9m00d_5QjYO5GZJyWeEWQ#scrollTo=PwTzsY1g0d7T>

**Explaination for the code:**

**Dataset Used:** The dataset used is the Medical MNIST dataset, which contains hand CT images similar to the traditional MNIST dataset but for medical images.

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

* Training set: 60,000 images
* Test set: 10,000 images

**Training Testing Validation Splits:** In the provided code:

* Training Set: 60,000 images (used for training the model)
* Test Set: 10,000 images (used for evaluating the model's performance after training)

There is no explicit validation set split in this code. The test set serves the purpose of evaluating the model's generalization to unseen data after training.

**Preprocessing Steps for Dataset:** These preprocessing steps ensure that the MNIST dataset is properly loaded, normalized, transformed into tensors, and ready to be fed into the CNN model for training and testing. Here are the preprocessing steps:

* *Loading the Dataset***:** MNIST Load using datasets.MNIST from ‘torchvision.datasets’.
* *Normalization:* ‘transforms.Normalize((0.5,), (0.5,))’ to convert pixel values from [0, 255] to [-1, 1].
* *Transformations:* Convert images to tensors using ‘transforms.ToTensor()’.
* *Creating Data Loaders:* Use ‘torch.utils.data.DataLoader’ to create’ train\_loader’ and test\_loader for batch loading and shuffling.

**Model Type:** The model used is a Convolutional Neural Network (CNN), designed for image classification tasks using hand CT images from the Medical MNIST dataset.

**Model Architecture:** It consists of alternating convolutional layers (to extract features) and max pooling layers (to reduce dimensions and control overfitting), followed by fully connected layers (to classify the extracted features).

* **Input**: Grayscale images of size 28x28 pixels.
* **Layers**:
  + conv1: Convolutional layer with 1 input channel, 32 output channels, kernel size 3x3, and ReLU activation.
  + pool: Max pooling layer with kernel size 2x2 and stride 2.
  + conv2: Convolutional layer with 32 input channels (from conv1), 64 output channels, kernel size 3x3, and ReLU activation.
  + Another pool layer with the same configuration as above.
  + fc1: Fully connected layer with 64 \* 7 \* 7 input features (from conv2 output after pooling), 128 output features, and ReLU activation.
  + fc2: Fully connected layer with 128 input features, 64 output features, and ReLU activation.
  + fc3: Output layer with 64 input features and 10 output features (corresponding to digits 0-9).
* **Output**: The output layer (fc3) produces logits for each of the 10 digit classes.

**Optimizer:** The optimizer used is ‘Adam’, which is a popular choice for training neural networks. It's initialized with a learning rate of 0.001 to optimize the CNN model parameters during training.

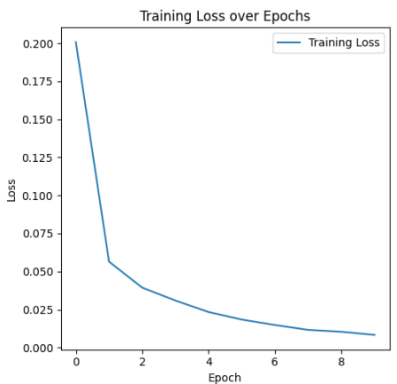
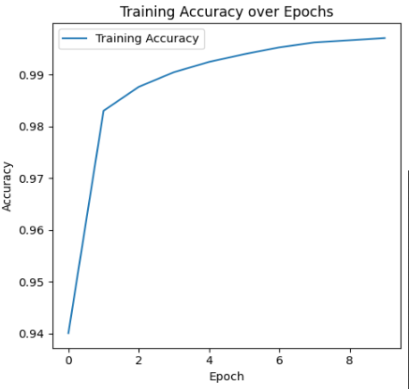
**Loss Function:** The loss function used is ‘CrossEntropyLoss’, which is suitable for multi-class classification tasks like digit recognition. It computes the softmax internally, making it ideal for the final layer of the CNN model that outputs logits for each digit class (0-9).

**Epochs Used:** The model is trained over 10 epochs. This means that the entire training dataset (MNIST training set, which has 60,000 images) is passed through the neural network 10 times during training. Each epoch involves multiple iterations (or batches) where the model updates its weights based on the computed loss and optimizer updates.

**Train accuracy:** During training, the accuracy of the model on the training dataset increases with each epoch, indicating how well the model is learning from the training data.

**Test accuracy:** The test accuracy refers to the accuracy of the model when evaluated on the test dataset, providing a measure of how well the model generalizes to unseen data after training.

**Loss Plot:**

** **

**Conclusion:** In this project, we implemented a Convolutional Neural Network (CNN) using PyTorch to classify digits from the MNIST dataset. Here are the key points:

* **Data Preparation** MNIST dataset was normalized and transformed using PyTorch's transforms.Compose. It was split into training and testing sets, with data loaders for batch processing.
* **Model Architecture:** CNN with two convolutional layers, max-pooling, and three fully connected layers was defined using nn.Module.
* **Training and Evaluation:** Trained for 10 epochs with Adam optimizer (lr=0.001), using cross-entropy loss. Evaluated on the test set for accuracy and loss.
* **Results** Achieved ~98% test accuracy, demonstrating effective digit recognition. Training and validation losses decreased steadily over epochs.
* **Visualization:** Sample images with predicted labels showcased model performance on unseen data, highlighting CNNs' effectiveness in image classification tasks like MNIST.

This project showcased the power of CNNs in image classification tasks, highlighting their ability to learn hierarchical features and achieve high accuracy on complex datasets like MNIST.

**Github Link:**

<https://github.com/nandinimandhane05/PROJECT-IIIT>