**Algorithm for dl:**

**1d cnn:**

**Step 1: Import Libraries**

Import Python libraries for data handling, model building, and evaluation.

**Step 2: Load EEG Data**

* Load three Excel files:
  + Epileptic seizure data
  + Epileptic non-seizure data
  + Normal data
* Assign labels:
  + 2 = Seizure,
  + 1 = Non-Seizure,
  + 0 = Normal

**Step 3: Data Augmentation**

For each EEG signal, create variations by:

* Adding noise
* Scaling amplitude
* Flipping signal

**Step 4: Preprocess Data**

* Pad all signals to the same length.
* Normalize using standard scaling.
* Convert labels to one-hot format.

**Step 5: Split the Data**

Split into:

* Training set (70%)
* Validation set (15%)
* Test set (15%)

**Step 6: Compute Class Weights**

Calculate weights to handle class imbalance during training.

**Step 7: Build 1D CNN Model**

Design a neural network with:

* Conv1D layers
* Pooling and normalization
* Dropout for regularization
* Dense layers with softmax output

**Step 8: Compile the Model**

* Use Adam optimizer and categorical crossentropy loss.

**Step 9: Set Callbacks**

Use:

* Early stopping
* Learning rate reduction on plateau

**Step 10: Train the Model**

Train the model using training and validation data.

**Step 11: Evaluate the Model**

* Test the model on unseen test data.
* Print accuracy and loss.

**Step 12: Save and Load Model**

Save the trained model and reload it for future use.

**Step 13: Make Predictions**

Predict on test data and get predicted class labels.

**Step 14: Show Results**

* Display classification report (precision, recall, F1-score).
* Plot confusion matrix.
* Plot ROC curves for each class.
* List misclassified samples.

**Gru**:

**Step 1: Import Libraries**

Use PyTorch and scikit-learn for deep learning and evaluation.

**Step 2: Define the Model**

Create a GRU-based model with:

* GRU layers to learn time patterns in EEG,
* Dropout to avoid overfitting,
* Fully connected (Linear) layer for classification.

**Step 3: Prepare the Data**

* Convert EEG data to PyTorch tensors.
* Organize the data into training and validation sets.

**Step 4: Handle Class Imbalance**

Calculate class weights to help the model treat all classes fairly.

**Step 5: Set Parameters**

Choose values like:

* Input size (number of features),
* Hidden units in GRU,
* Number of layers,
* Learning rate,
* Batch size,
* Number of epochs.

**Step 6: Create Data Loaders**

Batch and shuffle the training and validation data using DataLoader.

**Step 7: Train the Model**

For each epoch:

* Train the model on training data.
* Calculate loss and accuracy.
* Evaluate on validation data.
* Use scheduler to adjust learning rate.
* Save the model if validation improves.
* Stop early if performance doesn’t improve after a few tries.

**Step 8: Evaluate the Model**

* Load the best saved model.
* Predict on validation data.
* Print confusion matrix and classification report.

**1dcnn+gru:**

**Step 1: Import Libraries**

Import PyTorch, pandas, NumPy, matplotlib, and scikit-learn for data loading, deep learning, and performance visualization.

**Step 2: Load EEG Data**

* Load EEG signals from 3 Excel files:
  + Normal data → label 0
  + Epilepsy without seizure → label 1
  + Epilepsy with seizure → label 2
* Parse and convert signals into float arrays.

**Step 3: Create EEG Dataset Class**

* Store signals as 1D tensors with shape [1, signal\_length].
* Assign each sample its respective class label.

**Step 4: Stratified Data Split**

* Use StratifiedKFold to split data into training and validation sets while maintaining class balance.

**Step 5: Save Validation Data**

* Save the validation EEG signals and labels into .npy files for future use or testing.

**Step 6: Define Mixup Augmentation**

* Combine two samples randomly using a weighted average to improve generalization and prevent overfitting.

**Step 7: Define Label Smoothing Loss**

* Implement custom label smoothing to soften the targets and prevent the model from becoming overconfident.

**Step 8: Build EEG Classification Model**

* 1D CNN layer extracts features from raw EEG signals.
* Max pooling reduces dimensionality.
* GRU layers capture temporal patterns.
* Fully connected layer outputs class predictions.

**Step 9: Set Up Training**

* Use Adam optimizer and learning rate scheduler.
* Apply mixup and label smoothing during training.
* Monitor validation accuracy and save the best model.
* Use early stopping if no improvement after 5 epochs.

**Step 10: Evaluate Model**

* Load the best saved model.
* Generate:
  + Classification report (precision, recall, F1-score),
  + Confusion matrix (true vs. predicted),

**ResNet 18:**

1 **Load EEG Data from Excel**

* For each file:
  + Read signals
  + Convert string data to float arrays
  + Assign class label (0, 1, or 2)

2 **Convert EEG Signals to Mel Spectrograms**

* For each signal:
  + Use MelSpectrogram transformation
  + Apply log scaling
  + Skip invalid or empty signals

3 **Split Dataset**

* Use Stratified K-Fold to split data into training and validation sets

4 **Define Data Augmentations**

* **Mixup**: Blend two samples for regularization
* **Label Smoothing**: Prevent overconfidence in predictions

5 **Build ResNet18 Model**

* Modify first layer to accept 1-channel EEG spectrograms
* Modify last layer to output 3 classes

6 **Train the Model**

* For each epoch:
  + Apply mixup and train on batches
  + Evaluate on validation set
  + Save best model (early stopping)

7 **Evaluate Performance**

* Predict on validation set
* Show:
  + Classification Report
  + Confusion Matrix
  + ROC Curves

**LSTM:**

#### **Step 1: Import Libraries**

* Import required libraries:
  + torch, torch.nn, torch.utils.data for deep learning.
  + pandas, numpy for data handling.
  + matplotlib.pyplot, seaborn for visualization.
  + sklearn for data preprocessing and evaluation.

#### **Step 2: Load EEG Data**

* Load EEG signals from 3 Excel files:
  + normal\_data.xlsx → label 0
  + epilepsy\_without\_seizure\_data.xlsx → label 1
  + epilepsy\_with\_seizure\_data.xlsx → label 2
* Convert signals to 1D float arrays for each sample.

#### **Step 3: Create EEG Dataset Class**

* Define a custom PyTorch Dataset class.
* Store each EEG signal as a tensor of shape [signal\_length] (or [sequence\_length, 1] if needed for LSTM).
* Associate each sample with its class label.

#### **Step 4: Stratified Data Split**

* Use StratifiedKFold or train\_test\_split with stratify to split into training and validation sets, preserving class balance.

**Step 5: Save Validation Data**

* Save validation EEG signals and labels as .npy files for reuse or comparison with other models.

#### **Step 6: Define Label Smoothing Loss**

* Implement custom label smoothing loss to prevent overfitting and make the model less confident in its predictions.

#### **Step 7: Build LSTM Model**

* Model Architecture:
  + Input: [batch\_size, sequence\_length, input\_size]
  + LSTM layers to capture temporal dependencies in EEG signal sequences.
  + Optional: Dropout layer for regularization.
  + Fully connected layer to output class probabilities (3 classes).
  + Softmax activation for classification.

#### **Step 8: Set Up Training**

* Use:
  + Adam optimizer and learning rate scheduler.
  + CrossEntropyLoss or smoothed loss.
  + Early stopping if validation performance stagnates.
  + Save the best model based on validation accuracy

#### **Step 9: Evaluate Model**

* Load the best saved model.
* Use the validation set to evaluate:
  + Confusion matrix (true vs. predicted labels)
  + Classification report (precision, recall, F1-score)
  + Accuracy

**1DCNN+LSTM:**

#### **Step 1: Import Libraries**

* Import required Python libraries:
  + torch, torch.nn, torch.utils.data for model development.
  + pandas, numpy for data manipulation.
  + matplotlib.pyplot, seaborn for plotting.
  + sklearn for metrics and data splitting.

#### **Step 2: Load EEG Data**

* Load EEG signals from 3 Excel files:
  + normal\_data.xlsx → label 0
  + epilepsy\_without\_seizure\_data.xlsx → label 1
  + epilepsy\_with\_seizure\_data.xlsx → label 2
* Parse each signal and convert it into a 1D float array.

#### **Step 3: Create EEG Dataset Class**

* Define a custom PyTorch Dataset class.
* Each EEG signal is:
  + Converted to a 1D tensor of shape [1, signal\_length].
  + Associated with its respective label (0, 1, or 2).

#### **Step 4: Stratified Data Split**

* Use StratifiedKFold to split data into training and validation sets while keeping class distributions balanced.

#### **Step 5: Save Validation Data**

* Save the validation signals and labels as .npy files for future testing or model comparison.

#### **Step 6: Define Mixup Augmentation**

* Implement Mixup:
  + Blend two EEG samples with a random weight λ for data diversity.
  + Mix both signals and corresponding labels.

#### **Step 7: Define Label Smoothing Loss**

* Create a custom label smoothing loss function to prevent the model from becoming overconfident and improve generalization.

#### **Step 8: Build 1D CNN + LSTM Model**

* **1D CNN Layers**: Extract spatial features from raw EEG signals.
* **Max Pooling**: Reduce feature size and capture essential patterns.
* **LSTM Layer**: Capture temporal dependencies and signal progression.
* **Fully Connected (FC) Layer**: Output class probabilities (3 classes).

#### **Step 9: Set Up Training**

* Use:
  + Adam optimizer with a learning rate scheduler.
  + Mixup and label smoothing during training.
  + Early stopping if validation accuracy doesn’t improve for 5 epochs.
  + Save the best-performing model.

#### **Step 10: Evaluate Model**

* Load the best model checkpoint.
* Use the saved validation set for testing.
* Generate:
  + Confusion matrix
  + Classification report (precision, recall, F1-score)

**CUSTOM 2D CNN**

**Step 1: Load and Label Data**

* Read EEG signals from 3 Excel files.
* Label them as:
  + 0 → Seizure
  + 1 → Non-Seizure
  + 2 → Normal

**Step 2: Augment Signals**

* For each signal, create:
  + Jittered (with noise)
  + Scaled
  + Flipped
* Keep the original signal too.

**Step 3: Preprocess Data**

* Pad signals to the same length.
* Normalize using StandardScaler.

**Step 4: Reshape to 2D**

* Convert 1D signals to 2D matrices (e.g., 64×64).
* Reshape for CNN input: (64, 64, 1)

**Step 5: Split Dataset**

* Split data into:
  + Training set
  + Validation set
  + Test set

**Step 6: Define CNN Model**

* Start with a Conv2D layer and MaxPooling.
* Add 3 Residual Blocks:
  + Each block has Conv layers, BatchNorm, and skip connections.
* Apply Global Average Pooling.
* Add Dense layer with dropout.
* Final Dense layer with softmax for 3-class output.

**Step 7: Compile Model**

* Use Adam optimizer.
* Use categorical cross-entropy loss.
* Use accuracy as a metric.

**Step 8: Train Model**

* Train using training data.
* Use EarlyStopping and ReduceLROnPlateau callbacks.
* Apply class weights to balance classes.

**Step 9: Evaluate Model**

* Test on the unseen test data.
* Show:
  + Accuracy
  + Confusion matrix
  + Classification report
  + ROC curves