Lab12.4: Text Classification using 1D CNN with Pretrained GloVe Embeddings

Dataset: SMS Spam Collection
Pretrained Embeddings: GloVe 6B (100d)

STEP 1 — Install & Import Libraries

```
!pip install torch --quiet
!pip install scikit-learn --quiet

import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
import matplotlib.pyplot as plt
```

STEP 2 — Download GloVe Embeddings

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip
```

```
--2026-02-19 03:55:31--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2026-02-19 03:55:31--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2026-02-19 03:55:31--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.02MB/s    in 2m 41s

2026-02-19 03:58:13 (5.09 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]
```

STEP 3 — Load SMS Spam Dataset

```
data = pd.read_csv('SMSSpamCollection', sep='\t', header=None, names=['label','text'])

print("Dataset Size:", len(data))
print(data.head())

data['label'] = data['label'].map({'ham':0, 'spam':1})
```

```
Dataset Size: 5572
  label                                               text
0   ham  Go until jurong point, crazy.. Available only ...
1   ham                      Ok lar... Joking wif u oni...
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...
3   ham  U dun say so early hor... U c already then say...
4   ham  Nah I don't think he goes to usf, he lives aro...
```

STEP 4 — Text Preprocessing

```
import re

def clean_text(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    return text

data['text'] = data['text'].apply(clean_text)
tokenized = data['text'].apply(lambda x: x.split())
```

STEP 5 — Vocabulary & Embedding Matrix

```
vocab = {}
for tokens in tokenized:
    for word in tokens:
        if word not in vocab:
            vocab[word] = len(vocab) + 1

vocab_size = len(vocab) + 1
embedding_dim = 100

embeddings_index = {}
with open('glove.6B.100d.txt', encoding='utf8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = vector

embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, idx in vocab.items():
    vector = embeddings_index.get(word)
    if vector is not None:
        embedding_matrix[idx] = vector

print("Embedding Matrix Shape:", embedding_matrix.shape)
```

```
Embedding Matrix Shape: (8630, 100)
```

STEP 6 — Padding & Train-Test Split

```
from torch.utils.data import Dataset, DataLoader

max_len = 50

def encode(tokens):
    seq = [vocab.get(word, 0) for word in tokens]
    if len(seq) < max_len:
        seq += [0] * (max_len - len(seq))
    else:
        seq = seq[:max_len]
    return seq

X = np.array([encode(tokens) for tokens in tokenized])
y = data['label'].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

class SpamDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.long)
        self.y = torch.tensor(y, dtype=torch.float32)

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

train_dataset = SpamDataset(X_train, y_train)
test_dataset = SpamDataset(X_test, y_test)
```

```
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64)
```

STEP 7 — Define 1D CNN Model

```
class TextCNN(nn.Module):
    def __init__(self, vocab_size, embedding_dim, embedding_matrix):
        super(TextCNN, self).__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.embedding.weight = nn.Parameter(
            torch.tensor(embedding_matrix, dtype=torch.float32)
        )
        self.embedding.weight.requires_grad = False  # freeze embeddings

        self.conv1 = nn.Conv1d(embedding_dim, 128, kernel_size=3)
        self.conv2 = nn.Conv1d(embedding_dim, 128, kernel_size=4)
        self.conv3 = nn.Conv1d(embedding_dim, 128, kernel_size=5)

        self.dropout = nn.Dropout(0.5)

        self.fc = nn.Linear(128 * 3, 1)

    def forward(self, x):
        x = self.embedding(x)
        x = x.permute(0, 2, 1)

        c1 = torch.relu(self.conv1(x))
        c2 = torch.relu(self.conv2(x))
        c3 = torch.relu(self.conv3(x))

        p1 = torch.max(c1, dim=2)[0]
        p2 = torch.max(c2, dim=2)[0]
        p3 = torch.max(c3, dim=2)[0]

        out = torch.cat([p1, p2, p3], dim=1)
        out = self.dropout(out)
        out = self.fc(out)

        return torch.sigmoid(out)

model = TextCNN(vocab_size, embedding_dim, embedding_matrix)
```

STEP 8 — Model Training

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

epochs = 10

for epoch in range(epochs):
    model.train()
    total_loss = 0

    for X_batch, y_batch in train_loader:
        X_batch = X_batch.to(device)
        y_batch = y_batch.to(device)

        optimizer.zero_grad()
        outputs = model(X_batch).squeeze()
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    print(f"Epoch {epoch+1}, Loss: {total_loss/len(train_loader):.4f}")
```

```
Epoch 1, Loss: 0.2615
Epoch 2, Loss: 0.1101
Epoch 3, Loss: 0.0700
Epoch 4, Loss: 0.0485
Epoch 5, Loss: 0.0375
Epoch 6, Loss: 0.0270
Epoch 7, Loss: 0.0200
Epoch 8, Loss: 0.0165
Epoch 9, Loss: 0.0124
Epoch 10, Loss: 0.0094
```

STEP 9 — Model Evaluation

```
model.eval()
all_preds = []
all_labels = []

with torch.no_grad():
    for X_batch, y_batch in test_loader:
        X_batch = X_batch.to(device)
        outputs = model(X_batch).squeeze()
        preds = (outputs > 0.5).int().cpu().numpy()

        all_preds.extend(preds)
        all_labels.extend(y_batch.numpy())

accuracy = accuracy_score(all_labels, all_preds)
precision = precision_score(all_labels, all_preds)
recall = recall_score(all_labels, all_preds)
f1 = f1_score(all_labels, all_preds)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

cm = confusion_matrix(all_labels, all_preds)
print("\nConfusion Matrix:\n", cm)
```

```
Accuracy: 0.979372197309417
Precision: 0.9632352941176471
Recall: 0.8791946308724832
F1-score: 0.9192982456140351

Confusion Matrix:
 [[961   5]
 [ 18 131]]
```

STEP 10 — Result Analysis

Pretrained GloVe embeddings provide semantic information. CNN captures local n-gram features. Embeddings improve convergence and classification quality. Spam detection benefits from distinctive word patterns. Limitations include simple CNN architecture and fixed embeddings.