CS203 LAB 11
Team Number:37
Team Members:

1. **Maharshi Patel(24210059)**
2. **Saravanan S (24210090)**

1. Dataset Preparation

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
import torch
from torch.utils.data import TensorDataset, DataLoader

# Load data
train_df = pd.read_csv("https://raw.githubusercontent.com/clairett/pytorch-sentiment-classification/master/data/SST2/train.tsv", sep="\t", header=0)
train_df.columns = ['sentence', 'label']
test_df = pd.read_csv("https://raw.githubusercontent.com/clairett/pytorch-sentiment-classification/master/data/SST2/test.tsv", sep="\t", header=0)
test_df.columns = ['sentence', 'label']


# Vectorize using TF-IDF
vectorizer = TfidfVectorizer(max_features=10000)
X_train_full = vectorizer.fit_transform(train_df['sentence']).toarray()
y_train_full = train_df['label'].values

X_test = vectorizer.transform(test_df['sentence']).toarray()

# Train/Validation Split
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.2, random_state=42)

# Convert to tensors
train_dataset = TensorDataset(torch.tensor(X_train, dtype=torch.float32), torch.tensor(y_train))
val_dataset   = TensorDataset(torch.tensor(X_val, dtype=torch.float32), torch.tensor(y_val))
test_dataset  = TensorDataset(torch.tensor(X_test, dtype=torch.float32), torch.zeros(len(X_test), dtype=torch.long))
```

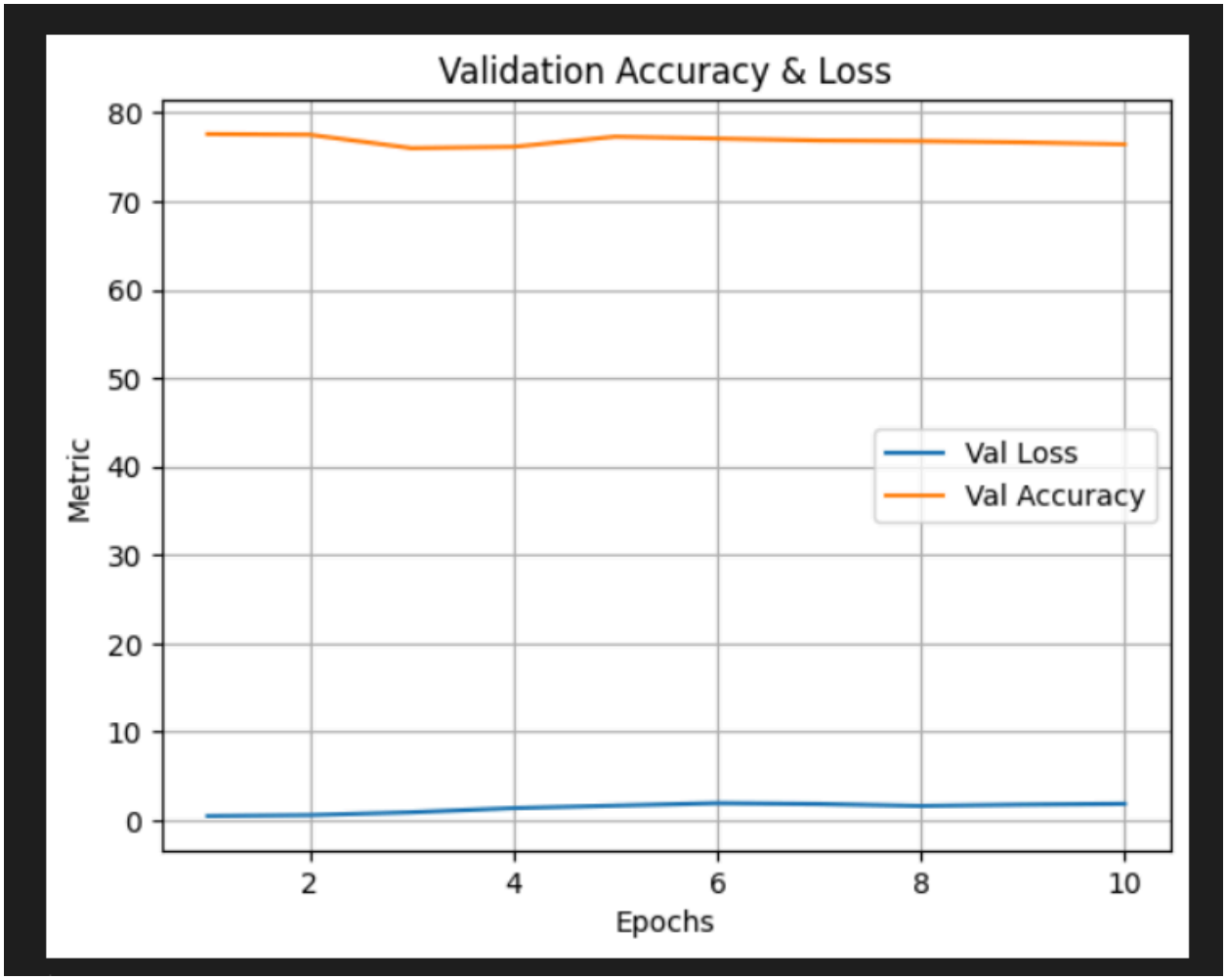
2. Construct a **Multi-Layer Perceptron (MLP) model**

```
[11]
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

print(f"Trainable Parameters: {count_parameters(mlp)}")
```

```
Trainable Parameters: 5293122
```

```python
class MLP(nn.Module):
    def __init__(self, input_size, hidden_sizes=[512, 256, 128, 64], output_size=2):
        super(MLP, self).__init__()
        layers = []
        sizes = [input_size] + hidden_sizes
        for i in range(len(sizes) - 1):
            layers.append(nn.Linear(sizes[i], sizes[i+1]))
            layers.append(nn.ReLU())
        layers.append(nn.Linear(sizes[-1], output_size))
        self.model = nn.Sequential(*layers)

    def forward(self, x):
        return self.model(x)
```

3. Train the model with 10 epochs and create the best-performing model and the plot the loss curves



```
        if val_acc > best_val_acc:
            torch.save(mlp.state_dict(), "checkpoint.pt")
            best_val_acc = val_acc

        print(f"Epoch {epoch+1}: Train Loss={train_losses[-1]:.4f}, Val Loss={val_losses[-1]:.4f}, Val Acc={val_acc:.2f}%")

    # Plot
    plt.plot(range(1, 11), val_losses, label='Val Loss')
    plt.plot(range(1, 11), val_accuracies, label='Val Accuracy')
    plt.xlabel("Epochs")
    plt.ylabel("Metric")
    plt.legend()
    plt.title("Validation Accuracy & Loss")
    plt.grid(True)
    plt.show()
```

[12]

```
Epoch 1: Train Loss=0.5570, Val Loss=0.4628, Val Acc=77.60%
Epoch 2: Train Loss=0.1968, Val Loss=0.5611, Val Acc=77.53%
Epoch 3: Train Loss=0.0428, Val Loss=0.8784, Val Acc=76.01%
Epoch 4: Train Loss=0.0108, Val Loss=1.3520, Val Acc=76.16%
Epoch 5: Train Loss=0.0036, Val Loss=1.6346, Val Acc=77.31%
Epoch 6: Train Loss=0.0014, Val Loss=1.9122, Val Acc=77.10%
Epoch 7: Train Loss=0.0022, Val Loss=1.8173, Val Acc=76.88%
Epoch 8: Train Loss=0.0020, Val Loss=1.5998, Val Acc=76.81%
Epoch 9: Train Loss=0.0001, Val Loss=1.7473, Val Acc=76.66%
Epoch 10: Train Loss=0.0000, Val Loss=1.8395, Val Acc=76.45%
```

1. **Dynamic Quantization with INT4 or INT8**

```
import torch.quantization

quantized_model = torch.quantization.quantize_dynamic(
    mlp.cpu(), {nn.Linear}, dtype=torch.qint8
)
torch.save(quantized_model.state_dict(), "quantized_dynamic.pt")
```

2. **Half precision**

```
fp16_model = MLP(input_size=input_size)  # Reload model fresh
fp16_model.load_state_dict(torch.load("checkpoint.pt"))
fp16_model = fp16_model.half()
torch.save(fp16_model.state_dict(), "half_precision.pt")
```

5. Fill the table for different quantization techniques

Model	Accuracy (%)	Size (MB)	Infer Time (ms)
Original	77.60	21.18	1.59
Dynamic	76.45	5.30	0.99
Half	77.60	10.59	1.51