```
import pandas as pd
import re
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from torch.nn.utils.rnn import pad_sequence
from collections import Counter

# Load and clean data
df = pd.read_csv("/UpdatedResumeDataSet.csv")
def clean_text(text):
    text = text.lower()
    text = re.sub(r"\n", " ", text)
    text = re.sub(r"[^\w\s]", "", text)
    text = re.sub(r"\s+", " ", text).strip()
    return text

df["Cleaned_Resume"] = df["Resume"].apply(clean_text)
label_encoder = LabelEncoder()
df["Label"] = label_encoder.fit_transform(df["Category"])

# Train-test split
train_texts, val_texts, train_labels, val_labels = train_test_split(
    df["Cleaned_Resume"].tolist(),
    df["Label"].tolist(),
    test_size=0.2,
    random_state=42,
    stratify=df["Label"]
)

# Tokenization
def tokenize(texts, vocab=None):
    tokenized = [text.split() for text in texts]
    if vocab is None:
        word_counts = Counter(word for sentence in tokenized for word in sentence)
        vocab = {word: idx + 2 for idx, (word, _) in enumerate(word_counts.items())}
        vocab["<PAD>"] = 0
        vocab["<UNK>"] = 1
    indexed = [[vocab.get(word, 1) for word in sentence] for sentence in tokenized]
    return indexed, vocab

train_tokens, vocab = tokenize(train_texts)
val_tokens, _ = tokenize(val_texts, vocab)

# Dataset & DataLoader
class ResumeDataset(Dataset):
    def __init__(self, inputs, labels):
        self.inputs = [torch.tensor(seq) for seq in inputs]
        self.labels = torch.tensor(labels)

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

    def __getitem__(self, idx):
        return self.inputs[idx], self.labels[idx]

def collate_fn(batch):
    inputs, labels = zip(*batch)
    padded_inputs = pad_sequence(inputs, batch_first=True, padding_value=0)
    return padded_inputs, torch.tensor(labels)

train_dataset = ResumeDataset(train_tokens, train_labels)
val_dataset = ResumeDataset(val_tokens, val_labels)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, collate_fn=collate_fn)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, collate_fn=collate_fn)
```

```
class LSTMClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, output_dim):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0)
        self.lstm = nn.LSTM(embed_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = self.embedding(x)
        _, (hidden, _) = self.lstm(x)
        return self.fc(hidden[-1])
```

```
class BiLSTMClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, output_dim):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0)
        self.bilstm = nn.LSTM(embed_dim, hidden_dim, batch_first=True, bidirectional=True)
        self.fc = nn.Linear(hidden_dim * 2, output_dim)

    def forward(self, x):
        x = self.embedding(x)
        _, (hidden, _) = self.bilstm(x)
        hidden = torch.cat((hidden[-2], hidden[-1]), dim=1)
        return self.fc(hidden)
```

```
class GRUClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, output_dim):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0)
        self.gru = nn.GRU(embed_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = self.embedding(x)
        _, hidden = self.gru(x)
        return self.fc(hidden[-1])
```

```
# Store final accuracies
final_accuracies = {}

def train_model(model, name, train_loader, val_loader, num_epochs=3):
    print(f"\nTraining {name}...")
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=1e-3)

    for epoch in range(num_epochs):
        model.train()
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        model.eval()
        all_preds, all_labels = [], []
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                preds = torch.argmax(outputs, dim=1)
                all_preds.extend(preds.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())
        acc = accuracy_score(all_labels, all_preds)
        print(f"{name} - Epoch {epoch + 1}: Validation Accuracy = {acc:.4f}")

    final_accuracies[name] = acc  # store final acc for model

# Run all models
vocab_size = len(vocab)
embed_dim = 128
hidden_dim = 128
output_dim = len(label_encoder.classes_)

lstm_model = LSTMClassifier(vocab_size, embed_dim, hidden_dim, output_dim)
bilstm_model = BiLSTMClassifier(vocab_size, embed_dim, hidden_dim, output_dim)
gru_model = GRUClassifier(vocab_size, embed_dim, hidden_dim, output_dim)

train_model(lstm_model, "LSTM", train_loader, val_loader)
train_model(bilstm_model, "BiLSTM", train_loader, val_loader)
train_model(gru_model, "GRU", train_loader, val_loader)


print("\n Final Validation Accuracies:")
for model_name, accuracy in final_accuracies.items():
    print(f"{model_name}: {accuracy:.4f}")
```

```
Training LSTM...
LSTM - Epoch 1: Validation Accuracy = 0.1295
LSTM - Epoch 2: Validation Accuracy = 0.1295
LSTM - Epoch 3: Validation Accuracy = 0.1140

Training BiLSTM...
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-5-638df0880059> in <cell line: 0>()
     44 
     45 train_model(lstm_model, "LSTM", train_loader, val_loader)
---> 46 train_model(bilstm_model, "BiLSTM", train_loader, val_loader)
     47 train_model(gru_model, "GRU", train_loader, val_loader)
     48 

3 frames

/usr/local/lib/python3.11/dist-packages/torch/autograd/graph.py in _engine_run_backward(t_outputs, *args, **kwargs)
    821         unregister_hooks = _register_logging_hooks_on_whole_graph(t_outputs)
    822     try:
--> 823         return Variable._execution_engine.run_backward(  # Calls into the C++ engine to run the backward pass
    824             t_outputs, *args, **kwargs
    825         )  # Calls into the C++ engine to run the backward pass

KeyboardInterrupt: 
```

```
import pandas as pd
import numpy as np
import re
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, GRU, Bidirectional, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping

nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')

data = pd.read_csv("UpdatedResumeDataSet.csv")

lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'\d+', '', text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    return text.strip()

def preprocess_text(text):
    text = clean_text(text)
    tokens = word_tokenize(text)
    tokens = [t for t in tokens if t not in stop_words]
    tokens = [lemmatizer.lemmatize(t) for t in tokens]
    return ' '.join(tokens)

data = data.dropna(subset=['Resume', 'Category'])
data['Processed_Resume'] = data['Resume'].apply(preprocess_text)
data = data[data['Processed_Resume'].str.split().str.len() > 3]

le = LabelEncoder()
data['Category_Code'] = le.fit_transform(data['Category'])
num_classes = data['Category_Code'].nunique()

max_words = 10000
max_len = 200

tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
tokenizer.fit_on_texts(data['Processed_Resume'])
sequences = tokenizer.texts_to_sequences(data['Processed_Resume'])
X = pad_sequences(sequences, maxlen=max_len)
y = to_categorical(data['Category_Code'])
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

def build_model(model_type='bilstm'):
    model = Sequential()
    model.add(Embedding(input_dim=max_words, output_dim=128))
    if model_type == 'bilstm':
        model.add(Bidirectional(LSTM(64)))
    elif model_type == 'gru':
        model.add(GRU(64))
    elif model_type == 'lstm':
        model.add(LSTM(64))
    model.add(Dropout(0.5))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

for model_type in ['bilstm', 'gru', 'lstm']:
    model = build_model(model_type)
    model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1, callbacks=[early_stopping], verbose=0)
    _, accuracy = model.evaluate(X_test, y_test, verbose=0)
    print(f"{model_type.upper()} Test Accuracy: {accuracy:.4f}")
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
BILSTM Test Accuracy: 0.9741
GRU Test Accuracy: 0.9689
LSTM Test Accuracy: 0.9534
```