Deep Learning – Assignment 1: Single-Word Audio Classification (FNN)

Library

In [1]:
```
!nvidia-smi
```

```
Fri May  2 22:28:38 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 572.42                 Driver Version: 572.42         CUDA Version: 12.8     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                  Driver-Model | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 3060 ...  WDDM  |   00000000:01:00.0 Off |                  N/A |
| N/A   51C    P8             13W /   75W |     281MiB /   6144MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                        GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|    0   N/A  N/A           28676      C   ...s\Python\Python310\python.exe      N/A      |
+-----------------------------------------------------------------------------------------+
```

In [2]:
```
import os

import time
import torch
import torch.nn as nn
import torch.optim as optim
import torchaudio
import librosa
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score
from torch.utils.data import Dataset, DataLoader
from tqdm import tqdm
```

In [3]:
```
# Check if GPU is available
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda
```

Extract Audio

In [4]:
```
class AudioDataset(Dataset):
    def __init__(self, root_dir):
        self.root_dir = root_dir
        self.samples = []
        self.labels = []
        self.label_map = {}

        for idx, label in enumerate(sorted(os.listdir(root_dir))):
            label_path = os.path.join(root_dir, label)
            if os.path.isdir(label_path):
                self.label_map[idx] = label
                for file in os.listdir(label_path):
                    if file.endswith(".wav"):
```

```
                        self.samples.append(os.path.join(label_path, file))
                        self.labels.append(idx)

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

    def __getitem__(self, idx):
        file_path = self.samples[idx]
        label = self.labels[idx]
        y, sr = librosa.load(file_path, sr=None)
        mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
        feature = np.mean(mfcc.T, axis=0)
        return torch.tensor(feature, dtype=torch.float32), label
```

In [5]:
```
train_dataset = AudioDataset("Data_People/Training") # type: ignore
test_dataset = AudioDataset("Data_People/Testing") # type: ignore

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
```

Training

In [ ]:
```
class AudioFNN(nn.Module):
    def __init__(self, input_size, num_classes):
        super(AudioFNN, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_size, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, num_classes)
        )

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

In [7]:
```
input_size = 13
num_classes = len(train_dataset.label_map)
model = AudioFNN(input_size, num_classes) # type: ignore

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.003)
```

In [8]:
```
# from torchsummary import summary

# summary(model, input_size=(1, 28, 28))
```

In [9]:
```
# Train in one epoch function
def train_one_epoch(model, train_loader, loss_fn, optimizer, device):
    model.train()
    train_loss, train_correct = 0, 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = loss_fn(outputs, labels)
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * inputs.size(0)
        _, predictions = torch.max(outputs, 1)
        train_correct += torch.sum(predictions == labels.data)

    return train_loss / len(train_loader.dataset), train_correct.double() / len(train_loader.dataset) # type

# Validation function
```

```
def validate(model, val_loader, loss_fn, device):
    model.eval()
    val_loss, val_correct = 0, 0

    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = loss_fn(outputs, labels)

            val_loss += loss.item() * inputs.size(0)
            _, predictions = torch.max(outputs, 1)
            val_correct += torch.sum(predictions == labels.data)

    return val_loss / len(val_loader.dataset), val_correct.double() / len(val_loader.dataset) # type: ignore

# Training and validation loop with timing
def train_and_validate(model, train_loader, val_loader, loss_fn, optimizer, epochs, device='cuda'):
    model.to(device)
    history = {
        'train_loss': [],
        'train_accuracy': [],
        'val_loss': [],
        'val_accuracy': []
    }

    for epoch in tqdm(range(epochs), desc="Training Progress", leave=True):
        epoch_start_time = time.time()

        train_loss, train_accuracy = train_one_epoch(model, train_loader, loss_fn, optimizer, device)
        val_loss, val_accuracy = validate(model, val_loader, loss_fn, device)

        history['train_loss'].append(train_loss)
        history['train_accuracy'].append(train_accuracy.item())
        history['val_loss'].append(val_loss)
        history['val_accuracy'].append(val_accuracy.item())

        epoch_end_time = time.time()

        tqdm.write(f'Epoch {epoch+1}/{epochs}: Train loss: {train_loss:.4f}, Train accuracy: {train_accuracy
                   f'Val loss: {val_loss:.4f}, Val accuracy: {val_accuracy:.4f}, '
                   f'Time: {(epoch_end_time - epoch_start_time):.2f}s')

    return model, history
```

In [10]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
model, history = train_and_validate(model, train_loader, test_loader, criterion, optimizer, epochs=30, devic
```

```
Training Progress:   3%|▍         | 1/30 [00:04<02:04,  4.30s/it]
Epoch 1/30: Train loss: 3.8544, Train accuracy: 0.4028, Val loss: 1.8342, Val accuracy: 0.3889, Time: 4.30s
Training Progress:   7%|▋         | 2/30 [00:05<01:01,  2.21s/it]
Epoch 2/30: Train loss: 2.3409, Train accuracy: 0.3750, Val loss: 1.2645, Val accuracy: 0.5556, Time: 0.74s
Training Progress:  10%|█         | 3/30 [00:05<00:42,  1.56s/it]
Epoch 3/30: Train loss: 1.2278, Train accuracy: 0.5000, Val loss: 1.2531, Val accuracy: 0.3889, Time: 0.78s
Training Progress:  13%|█▎        | 4/30 [00:06<00:31,  1.23s/it]
Epoch 4/30: Train loss: 1.1522, Train accuracy: 0.5833, Val loss: 1.1342, Val accuracy: 0.3889, Time: 0.72s
Training Progress:  17%|█▋        | 5/30 [00:07<00:26,  1.07s/it]
Epoch 5/30: Train loss: 1.1086, Train accuracy: 0.4583, Val loss: 1.0132, Val accuracy: 0.5000, Time: 0.77s
Training Progress:  20%|██        | 6/30 [00:08<00:22,  1.05it/s]
Epoch 6/30: Train loss: 0.7311, Train accuracy: 0.5278, Val loss: 0.8167, Val accuracy: 0.5556, Time: 0.72s
Training Progress:  23%|██▎       | 7/30 [00:08<00:20,  1.13it/s]
Epoch 7/30: Train loss: 0.6713, Train accuracy: 0.6250, Val loss: 0.6978, Val accuracy: 0.6667, Time: 0.75s
Training Progress:  27%|██▋       | 8/30 [00:09<00:18,  1.20it/s]
Epoch 8/30: Train loss: 0.5721, Train accuracy: 0.7500, Val loss: 0.6641, Val accuracy: 0.5556, Time: 0.73s
Training Progress:  30%|███       | 9/30 [00:10<00:16,  1.24it/s]
Epoch 9/30: Train loss: 0.5004, Train accuracy: 0.7222, Val loss: 0.6668, Val accuracy: 0.6111, Time: 0.74s
Training Progress:  33%|███▎      | 10/30 [00:11<00:15,  1.28it/s]
```

```
Epoch 10/30: Train loss: 0.4930, Train accuracy: 0.7639, Val loss: 0.5841, Val accuracy: 0.6667, Time: 0.73s
Training Progress:  37%|███▋      | 11/30 [00:11<00:14,  1.28it/s]
Epoch 11/30: Train loss: 0.4633, Train accuracy: 0.7778, Val loss: 0.5869, Val accuracy: 0.6111, Time: 0.76s
Training Progress:  40%|████      | 12/30 [00:12<00:13,  1.32it/s]
Epoch 12/30: Train loss: 0.4379, Train accuracy: 0.7639, Val loss: 0.5327, Val accuracy: 0.7222, Time: 0.71s
Training Progress:  43%|████▎     | 13/30 [00:13<00:12,  1.34it/s]
Epoch 13/30: Train loss: 0.4283, Train accuracy: 0.7639, Val loss: 0.5070, Val accuracy: 0.7222, Time: 0.72s
Training Progress:  47%|████▋     | 14/30 [00:13<00:11,  1.34it/s]
Epoch 14/30: Train loss: 0.3903, Train accuracy: 0.8194, Val loss: 0.5669, Val accuracy: 0.7222, Time: 0.74s
Training Progress:  50%|█████     | 15/30 [00:14<00:11,  1.35it/s]
Epoch 15/30: Train loss: 0.5028, Train accuracy: 0.7639, Val loss: 0.7538, Val accuracy: 0.6111, Time: 0.72s
Training Progress:  53%|█████▎    | 16/30 [00:15<00:10,  1.35it/s]
Epoch 16/30: Train loss: 0.4829, Train accuracy: 0.7639, Val loss: 1.0022, Val accuracy: 0.6111, Time: 0.74s
Training Progress:  57%|█████▋    | 17/30 [00:16<00:09,  1.36it/s]
Epoch 17/30: Train loss: 0.6829, Train accuracy: 0.6944, Val loss: 0.5724, Val accuracy: 0.6667, Time: 0.71s
Training Progress:  60%|██████    | 18/30 [00:16<00:08,  1.38it/s]
Epoch 18/30: Train loss: 0.4007, Train accuracy: 0.8056, Val loss: 0.5256, Val accuracy: 0.7778, Time: 0.71s
Training Progress:  63%|██████▎   | 19/30 [00:17<00:08,  1.36it/s]
Epoch 19/30: Train loss: 0.4365, Train accuracy: 0.8056, Val loss: 0.4090, Val accuracy: 0.8333, Time: 0.75s
Training Progress:  67%|██████▋   | 20/30 [00:18<00:07,  1.37it/s]
Epoch 20/30: Train loss: 0.3665, Train accuracy: 0.8333, Val loss: 0.4103, Val accuracy: 0.7778, Time: 0.72s
Training Progress:  70%|███████   | 21/30 [00:19<00:06,  1.36it/s]
Epoch 21/30: Train loss: 0.3431, Train accuracy: 0.8750, Val loss: 0.3899, Val accuracy: 0.8333, Time: 0.74s
Training Progress:  73%|███████▎  | 22/30 [00:19<00:05,  1.37it/s]
Epoch 22/30: Train loss: 0.3525, Train accuracy: 0.8194, Val loss: 0.4862, Val accuracy: 0.7778, Time: 0.72s
Training Progress:  77%|███████▋  | 23/30 [00:20<00:05,  1.37it/s]
Epoch 23/30: Train loss: 0.3260, Train accuracy: 0.8611, Val loss: 0.4300, Val accuracy: 0.8333, Time: 0.71s
Training Progress:  80%|████████  | 24/30 [00:21<00:04,  1.35it/s]
Epoch 24/30: Train loss: 0.3230, Train accuracy: 0.8889, Val loss: 0.6136, Val accuracy: 0.7222, Time: 0.76s
Training Progress:  83%|████████▎ | 25/30 [00:22<00:03,  1.37it/s]
Epoch 25/30: Train loss: 0.4578, Train accuracy: 0.7639, Val loss: 0.4184, Val accuracy: 0.8333, Time: 0.71s
Training Progress:  87%|████████▋ | 26/30 [00:22<00:02,  1.34it/s]
Epoch 26/30: Train loss: 0.3600, Train accuracy: 0.8194, Val loss: 0.3814, Val accuracy: 0.7778, Time: 0.77s
Training Progress:  90%|█████████ | 27/30 [00:23<00:02,  1.35it/s]
Epoch 27/30: Train loss: 0.3637, Train accuracy: 0.8056, Val loss: 0.4532, Val accuracy: 0.8333, Time: 0.73s
Training Progress:  93%|█████████▎| 28/30 [00:24<00:01,  1.33it/s]
Epoch 28/30: Train loss: 0.2898, Train accuracy: 0.8611, Val loss: 0.4413, Val accuracy: 0.8333, Time: 0.77s
Training Progress:  97%|█████████▋| 29/30 [00:25<00:00,  1.35it/s]
Epoch 29/30: Train loss: 0.3231, Train accuracy: 0.8056, Val loss: 0.5257, Val accuracy: 0.7778, Time: 0.71s
Training Progress: 100%|██████████| 30/30 [00:25<00:00,  1.16it/s]
Epoch 30/30: Train loss: 0.3082, Train accuracy: 0.8472, Val loss: 0.3796, Val accuracy: 0.8333, Time: 0.77s
```

Result

In [14]:

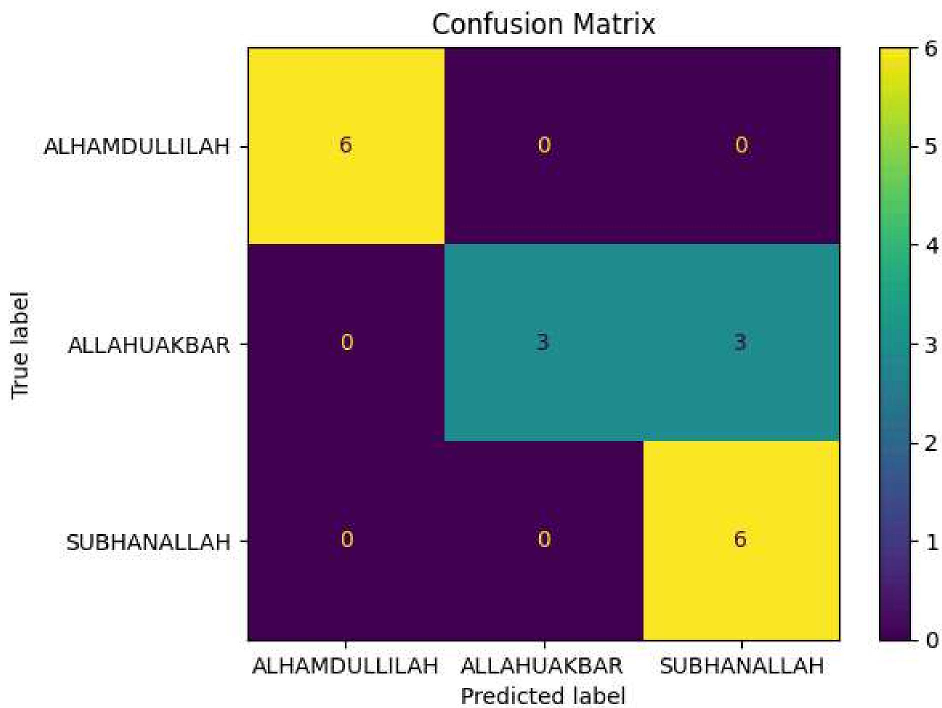
```
model.eval()

y_true, y_pred = [], []

with torch.no_grad():
    for features, labels in test_loader:
        features, labels = features.to(device), labels.to(device)
        outputs = model(features)
        _, predicted = torch.max(outputs.data, 1)
        y_true.extend(labels.cpu().tolist())
        y_pred.extend(predicted.cpu().tolist())

print('Accuracy of the model is:',accuracy_score(y_true, y_pred))
```

```
Accuracy of the model is: 0.8333333333333334
```

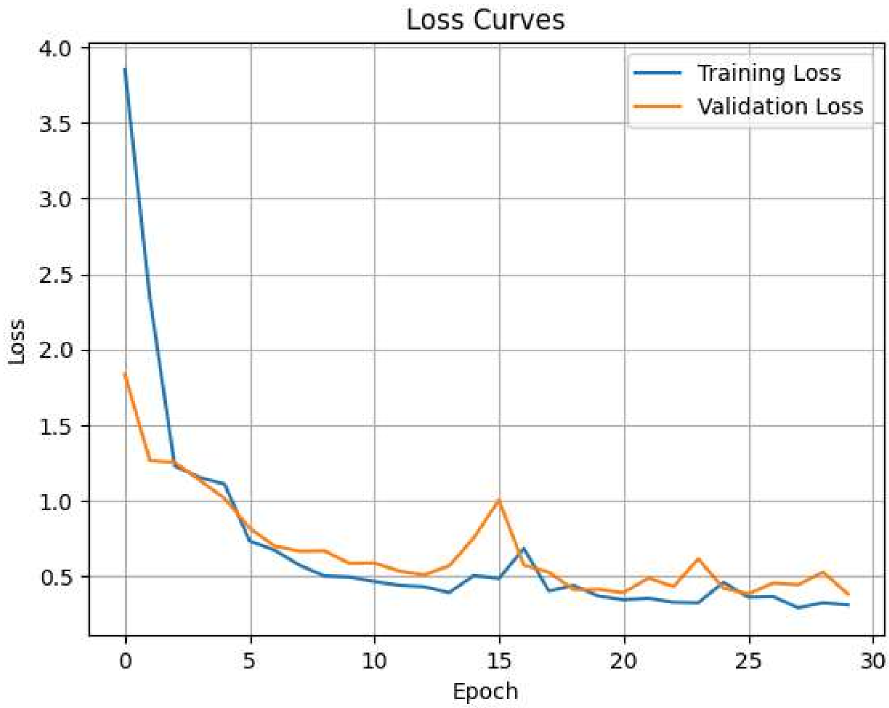
In [15]:

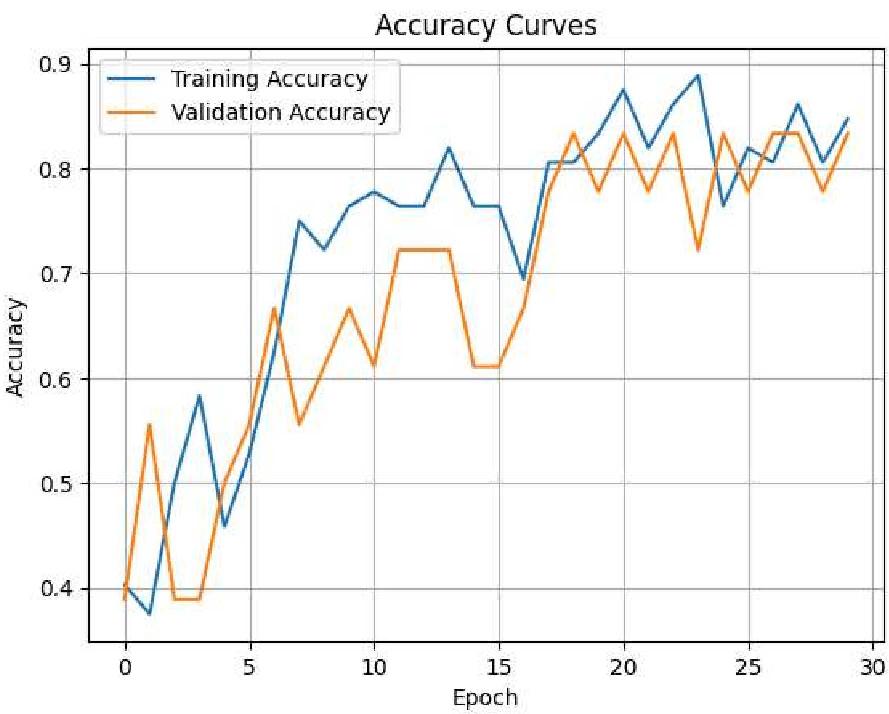
```
# Confusion Matrix
cm = confusion_matrix(y_true, y_pred)
ConfusionMatrixDisplay(cm, display_labels=[train_dataset.label_map[i] for i in range(num_classes)]).plot()
```

```
plt.title("Confusion Matrix")
plt.show()
```



In [13]:
```
plt.figure()
plt.plot(history['train_loss'], label="Training Loss")
plt.plot(history['val_loss'], label="Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Loss Curves")
plt.legend()
plt.grid(True)
plt.show()

plt.figure()
plt.plot(history['train_accuracy'], label="Training Accuracy")
plt.plot(history['val_accuracy'], label="Validation Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Accuracy Curves")
plt.legend()
plt.grid(True)
plt.show()
```





Discussion

This is my first model, a simple one using FNN. I also make a second model which is CNN which could be check after this and all of the other discussion in the CNN.