

```

#zoom random 5e-4 500 epochs then 1100

#Have commented out a lot of print statements to try and save time
#03/07/25 Code updated to remove z score norm and check channel dims
#LR scheduler added
#Stratified split not random
#27/06/25 PVA calcs added and to print at end of all training done

import os
import re
import torch
import numpy as np
import torch.nn as nn
import torch.optim as optim
from collections import Counter
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import torchvision.models
from sklearn.model_selection import train_test_split
import timm
import matplotlib.pyplot as plt
from google.colab import files
uploaded = files.upload()
import sys
from math import sqrt
#From [name of imported file] import [name of class within that file]
from MBConvBlock import MBConvBlock
#From [name of imported file] import [name of class within that file]
from ScaledDotProductAttention import ScaledDotProductAttention
sys.path.append('.')
from torch.utils.data import random_split
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.metrics import classification_report
from torch.utils.data import Subset
from torch.optim import lr_scheduler
from torch.nn.functional import pad
from torch.optim import lr_scheduler
import random

# Set device to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
def set_seed(seed=42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

set_seed(42)
#The pre-processing pipeline already performed z-score normalisation and channel changes, therefore
#the tensors being loaded are already shape [1, 224, 224]

def extract_number(filename):
    """Extracts numbers for sorting files like 'file_23.pt'."""
    match = re.search(r'(\d+)', filename)
    return int(match.group(1)) if match else 0

def generate_labels_from_filenames(mel_spectrogram_files, files_per_class=500):
    """
    Generates integer class labels based on file order.
    Example: 0 for first 25 files, 1 for next 25, etc.
    """
    mel_spectrogram_files.sort(key=extract_number)
    labels = [idx // files_per_class for idx in range(len(mel_spectrogram_files))]

    for idx, file in enumerate(mel_spectrogram_files):
        print(f"File: {file}, Label: {labels[idx]}")
    return labels

def check_labels(mel_spectrogram_files, labels):
    print("Checking file-label mapping:")
    for file, label in zip(mel_spectrogram_files, labels):
        print(f"File: {file} -> Label: {label}")

def collate_pad(batch):
    tensors, labels = zip(*batch)

    # Find max time dimension
    max_len = max(tensor.shape[-1] for tensor in tensors)

```

```

# Pad all tensors to max_len
padded_tensors = []
for tensor in tensors:
    pad_len = max_len - tensor.shape[-1]
    padded_tensor = pad(tensor, (0, pad_len)) # pad last dimension
    padded_tensors.append(padded_tensor)

return torch.stack(padded_tensors), torch.tensor(labels)
class MelSpectrogramDataset(Dataset):
    def __init__(self, mel_spectrogram_dir, mel_spectrogram_files, labels, transform=None):
        self.mel_spectrogram_files = mel_spectrogram_files
        self.labels = labels
        self.transform = transform
        self.mel_spectrogram_dir = mel_spectrogram_dir

        if len(self.mel_spectrogram_files) != len(self.labels):
            raise ValueError("Mismatch between number of files and labels.")

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

    def __getitem__(self, idx):
        file_name = self.mel_spectrogram_files[idx]
        path = os.path.join(self.mel_spectrogram_dir, file_name)

        mel = torch.load(path)
        label = int(self.labels[idx]) # Ensure label is integer
        #shouldn't need this next line anymore as my tensors should be
        #shape 1, 224, 224
        #if len(mel.shape) == 2:
        #    mel = mel.unsqueeze(0) # [1, H, W] - #✅ # Add channel dimensions i.e. change from [224, 224] to [1, 224, 224]

        # if self.transform:
        #    mel = nn.functional.interpolate(mel.unsqueeze(0), size=(224, 224), mode='bilinear', align_corners=False).squeeze(0)
        #    mel = self.transform(mel) # Resize + Normalize

        if mel.shape[0] == 1:
            mel = mel.repeat(3, 1, 1) #Duplicate channels to match CoAtNet input ([3, H, W]) i.e. [3, 224, 224]
        #print(f"Tensor shape in Dataset __getitem__: {mel.shape}") #Check that tensor is [3, 224, 224]
        return mel, label

class CoAtNet(nn.Module):
    def __init__(self, in_ch, image_size, num_classes=36, out_chs=[64,96,192,384,768]):
        super(CoAtNet, self).__init__()
        self.out_chs = out_chs
        self.maxpool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        self.maxpool1d = nn.MaxPool1d(kernel_size=2, stride=2)

        self.s0 = nn.Sequential(
            nn.Conv2d(in_ch, in_ch, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(in_ch, in_ch, kernel_size=3, padding=1)
        )
        self.mlp0 = nn.Sequential(
            nn.Conv2d(in_ch, out_chs[0], kernel_size=1),
            nn.ReLU(),
            nn.Conv2d(out_chs[0], out_chs[0], kernel_size=1)
        )
        self.s1 = MBConvBlock(ksize=3, input_filters=out_chs[0], output_filters=out_chs[0], image_size=image_size//2)
        self.mlp1 = nn.Sequential(
            nn.Conv2d(out_chs[0], out_chs[1], kernel_size=1),
            nn.ReLU(),
            nn.Conv2d(out_chs[1], out_chs[1], kernel_size=1)
        )
        self.s2 = MBConvBlock(ksize=3, input_filters=out_chs[1], output_filters=out_chs[1], image_size=image_size//4)
        self.mlp2 = nn.Sequential(
            nn.Conv2d(out_chs[1], out_chs[2], kernel_size=1),
            nn.ReLU(),
            nn.Conv2d(out_chs[2], out_chs[2], kernel_size=1)
        )
        self.s3 = ScaledDotProductAttention(out_chs[2], out_chs[2]//8, out_chs[2]//8, 8)
        self.mlp3 = nn.Sequential(
            nn.Linear(out_chs[2], out_chs[3]),
            nn.ReLU(),
            nn.Linear(out_chs[3], out_chs[3])
        )
        self.s4 = ScaledDotProductAttention(out_chs[3], out_chs[3]//8, out_chs[3]//8, 8)
        self.mlp4 = nn.Sequential(
            nn.Linear(out_chs[3], out_chs[4]),
            nn.ReLU(),
            nn.Linear(out_chs[4], out_chs[4])

```

```

    )

    self.avgpool = nn.AdaptiveAvgPool1d(1) # Avg pool over the sequence length (N)
    self.fc = nn.Linear(out_chs[4], num_classes)

    # Define softmax for output probabilities
    self.softmax = nn.Softmax(dim=1)

def forward(self, x):
    B, C, H, W = x.shape
    #print(f"Input shape: {x.shape}") # Expect [B, 3, 224, 224] #Debugging to check input shape as expected

    # Stage 0: Conv + MLP + MaxPool
    y = self.mlp0(self.s0(x))
    #print(f"After s0 and mlp0: {y.shape}") # Should keep spatial dims same as s0 output
    # show_feature_map(y, "Stage 0")
    y = self.maxpool2d(y)
    #print(f"After maxpool2d 0: {y.shape}") # spatial dims should halve here

    # Stage 1: MBConv + MLP + MaxPool
    y = self.mlp1(self.s1(y))
    #print(f"After s1 and mlp1: {y.shape}")
    #show_feature_map(y, "Stage 1")
    y = self.maxpool2d(y)
    #print(f"After maxpool2d 1: {y.shape}")

    # Stage 2: MBConv + MLP + MaxPool
    y = self.mlp2(self.s2(y))
    #print(f"After s2 and mlp2: {y.shape}")
    #show_feature_map(y, "Stage 2")
    y = self.maxpool2d(y)
    #print(f"After maxpool2d 2: {y.shape}")

    B, C, H, W = y.shape
    # Stage 3: Self Attention + MLP + MaxPool1d
    y = y.reshape(B, self.out_chs[2], -1).permute(0, 2, 1) # (B, N, C)
    #print(f"After reshape and permute for attention (stage 3): {y.shape}")
    y = self.mlp3(self.s3(y, y))
    #print(f"After s3 and mlp3: {y.shape}")
    y = self.maxpool1d(y.permute(0, 2, 1)).permute(0, 2, 1) # MaxPool over N
    #print(f"After maxpool1d 3: {y.shape}")

    # Stage 4: Self Attention + MLP + Global Average Pool + FC + Softmax
    y = self.mlp4(self.s4(y, y)) # y: (B, N, C)
    #print(f"After s4 and mlp4: {y.shape}")

    #print("Shape before permute:", y.shape) # (B, N, C)
    y = y.permute(0, 2, 1) # (B, C, N)
    #print("Shape after permute:", y.shape)

    y = self.avgpool(y) # (B, C, 1)
    #print("Shape after avgpool:", y.shape)

    y = y.squeeze(-1) # (B, C)
    #print("Shape after squeeze:", y.shape)

    y = self.fc(y) # (B, C)
    #print("Shape after fc:", y.shape)

    # Plot class probabilities for the first example in batch
    class_names = [f"Class {i}" for i in range(y.shape[1])]
    probs = y[0] # since batch size = 1

    #plt.figure(figsize=(10, 4))
    #plt.bar(class_names, probs.detach().cpu().numpy())
    #plt.title("Class Probabilities")
    #plt.xlabel("Classes")
    #plt.ylabel("Probability")
    #plt.xticks(rotation=45)
    #plt.show()

    return y

def main():
    tensor_folder = "/content/drive/MyDrive/ColabNotebooks/ZoomRecordings/ZoomTensorsOnly" #These are the Direct Phone recordings - all
    mel_files = [f for f in os.listdir(tensor_folder) if f.endswith(".pt")]

    # Create labels for 5 mel specs per keystroke i.e. 125 tensors per class
    labels = generate_labels_from_filenames(mel_files, files_per_class=500)
    num_classes = len(set(labels))
    print(f"Number of classes: {num_classes}")

```

```

#Check labels
mel_files = [f for f in os.listdir(tensor_folder) if f.endswith(".pt")]
labels = generate_labels_from_filenames(mel_files, files_per_class=500)
check_labels(mel_files, labels)

# Create Dataset & DataLoader
#dataset = MelSpectrogramDataset(tensor_folder, mel_files, labels, transform=transform)
# Full dataset
full_dataset = MelSpectrogramDataset(tensor_folder, mel_files, labels, transform=None)

# Convert labels to numpy for sklearn
labels_np = np.array(labels)
indices = np.arange(len(labels))
#Stratified split of data
# First split: Train (80%) vs Temp (20%)
# train_indices, temp_indices, y_train, y_temp = train_test_split(
#     indices,
#     labels_np,
#     test_size=0.2,
#     stratify=labels_np,
#     random_state=42
# )

# # Second split: Temp → Validation (10%) and Test (10%)
# val_indices, test_indices, y_val, y_test = train_test_split(
#     temp_indices,
#     y_temp,
#     test_size=0.5,
#     stratify=y_temp,
#     random_state=42
# )

train_indices, temp_indices = train_test_split(
    indices,
    test_size=0.2,
    random_state=42,
    shuffle=True
)

# Second split: Temp → Validation (10%) and Test (10%)
val_indices, test_indices = train_test_split(
    temp_indices,
    test_size=0.5,
    random_state=42,
    shuffle=True
)

# Create Subsets
train_dataset = Subset(full_dataset, train_indices)
validation_dataset = Subset(full_dataset, val_indices)
test_dataset = Subset(full_dataset, test_indices)

#Print the length of the dataset
print("Total number of samples in the dataset:", len(full_dataset))
train_ratio=0.6
validation_ratio=0.2
test_ratio=0.2
dataset_size = len(full_dataset)
train_size = int(train_ratio * dataset_size)
test_size=int(test_ratio * dataset_size)
validation_size = int(validation_ratio * dataset_size)

print("Train labels distribution:", np.bincount([label for _, label in train_dataset]))
print("Validation labels distribution:", np.bincount([label for _, label in validation_dataset]))
print("Test labels distribution:", np.bincount([label for _, label in test_dataset]))

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, collate_fn=collate_pad)
validation_loader=DataLoader(validation_dataset, batch_size=32, shuffle=False)
test_loader=DataLoader(test_dataset, batch_size=32, shuffle=False)
print(f'Total dataset size: {dataset_size}')
print(f'Training dataset size: {len(train_dataset)}')
print(f'Validation dataset size: {len(validation_dataset)}')
print(f'Test dataset size: {len(test_dataset)}')

def count_labels(subset, name):
    subset_labels = [full_dataset[i][1] for i in subset.indices]
    label_count = Counter(subset_labels)
    #print(f"{name} labels distribution:")

```

```

    #print(sorted(label_count.items()))
    #print()

    count_labels(train_dataset, "Train")
    count_labels(validation_dataset, "Validation")
    count_labels(test_dataset, "Test")

#The CoAtNet model is defined in it's own CoAtNet custom class above
#3 input channels, image dimensions 224x224, no. output classes for classification
    input_height= 224#no. mel freq bins
    model = CoAtNet(in_ch=3, image_size=input_height, num_classes=num_classes) #Calls to my Custom CoAtNet model/matches format of it

#Moves model to GPU or CPU for training
    model.to(device)

#Loss function is set to Cross entropy loss critereon
    criterion = nn.CrossEntropyLoss()
#Sets optimiser to Adam and learning rate is specified
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    train_losses = []
    val_losses = []
    train_accuracies = []
    val_accuracies = []
#TRAINING LOOP#
    scheduler = lr_scheduler.StepLR(optimizer, step_size=20, gamma=0.5) #LR scheduler
    num_epochs = 1300
    best_val_accuracy=0.0 #Track peak validation accuracy (PVA)

#Saving data to checkpoint as model keeps timing out
    checkpoint_path = "/content/drive/MyDrive/CoAtNet45498744546546546556475426_checkpoint.pth"
    start_epoch = 0

# Load checkpoint if it exists
    if os.path.exists(checkpoint_path):
        print("Loading checkpoint...")
        checkpoint = torch.load(checkpoint_path, map_location=device)
        file_to_label=checkpoint.get('file_to_label', None)
        model.load_state_dict(checkpoint['model_state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
        scheduler.load_state_dict(checkpoint['scheduler_state_dict'])
        start_epoch = checkpoint['epoch'] + 1 # resume from next epoch
        best_val_accuracy = checkpoint.get('best_val_accuracy', 0.0)

        train_losses = checkpoint.get('train_losses', [])
        val_losses = checkpoint.get('val_losses', [])
        train_accuracies = checkpoint.get('train_accuracies', [])
        val_accuracies = checkpoint.get('val_accuracies', [])

        # FIX: Truncate longer list to match shortest one
        min_len = min(len(train_losses), len(val_losses))
        train_losses = train_losses[:min_len]
        val_losses = val_losses[:min_len]
        train_accuracies = train_accuracies[:min_len]
        val_accuracies = val_accuracies[:min_len]

        print(f"Resumed from epoch {start_epoch}, best validation accuracy so far: {best_val_accuracy:.2f}%")
    total_epochs = 1300
    for epoch in range(start_epoch, num_epochs):
        model.train() #Sets the model to training mode enabling related features
        running_loss = 0.0 #Cumulative loss for the epoch
        correct = 0 #Correct prediction count
        total = 0 #Total sample count
#Iterates over batches of training data from train_loader
#Each batch contains images (input data) and labels (ground truth)

        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
            #print("Images shape:", images.shape) # e.g., torch.Size([64, 3, 224, 224])
            #print("Labels:", labels[:10]) # should be integers in [0, 35]
#Passes the input images through the model to get predictions
            outputs = model(images)
            #Computes the loss (how far the model predictions (outputs) are from the actual labels using a loss function called crite
            loss = criterion(outputs, labels)
#Clears any gradients from the previous step to avoid the accumulation of gradients
            optimizer.zero_grad()
            #Performs back propagation
            loss.backward()
            #Updates weights
            optimizer.step()
#Track the loss and accuracy
            running_loss += loss.item()#Adds current loss to total running loss
            _, predicted = outputs.max(1)#Checks how many predictions are correct
            total += labels.size(0)#No. samples processed

```

```

        correct += predicted.eq(labels).sum().item()#Number of correct predictions
#Prints the summary of each epoch
accuracy = 100 * correct / total
train_losses.append(running_loss / len(train_loader))
train_accuracies.append(accuracy)
scheduler.step() #Steps the learning rate scheduler after each epoch (not after each batch)

torch.save({
    'epoch': epoch,
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    'scheduler_state_dict': scheduler.state_dict(),
    'best_val_accuracy': best_val_accuracy,
    'train_losses': train_losses,
    'train_accuracies': train_accuracies,
    'val_losses': val_losses,
    'val_accuracies': val_accuracies,
    'file_to_label': {f: l for f, l in zip(mel_files, labels)}
}, checkpoint_path)
print(f"Checkpoint saved at epoch {epoch + 1}")

#EVALUATION LOOP#This is called immediately after the training loop within the same epoch "for" loop
model.eval() #set the model to evaluation mode (same as Validation)
val_loss = 0.0
val_correct = 0
val_total = 0
with torch.no_grad():
    for images, labels in validation_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)
        val_loss += loss.item()

        _, predicted = outputs.max(1)
        val_total += labels.size(0)
        val_correct += predicted.eq(labels).sum().item()

avg_val_loss = val_loss / len(validation_loader)
val_accuracy = 100 * val_correct / val_total
val_losses.append(avg_val_loss)
val_accuracies.append(val_accuracy)

# Update best validation accuracy if current is better
if val_accuracy > best_val_accuracy:
    best_val_accuracy = val_accuracy

print(f"Epoch [{epoch+1}/{num_epochs}] - Loss: {running_loss:.4f}, Accuracy: {accuracy:.2f}%")
print(f"Validation - Loss: {avg_val_loss:.4f}, Accuracy: {val_accuracy:.2f}%")

#Print PVA
print(f"\nPeak Validation Accuracy: {best_val_accuracy:.2f}%")

#Plot line plots of training & validation loss & accuracy per epoch
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Validation Loss')
plt.legend()
plt.title('Loss over Epochs')
plt.show()

plt.plot(train_accuracies, label='Train Acc')
plt.plot(val_accuracies, label='Val Acc')
plt.legend()
plt.title('Accuracy over Epochs')
plt.show()

# Evaluation on test set
model.eval()
all_preds = []
all_labels = []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = outputs.max(1)
        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())


# Confusion matrix
cm = confusion_matrix(all_labels, all_preds)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")

```

```
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()

# Classification report
print(classification_report(all_labels, all_preds))
#Produce a confusion matrix to analyse the results after the test loop
#Produce a classification report to analyse the results after the test loop

if __name__ == "__main__":
    main()
```

 No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to

enable.

Streaming output truncated to the last 5000 lines.

File: r_mel_spec_686_v14.pt -> Label: 27
File: r_mel_spec_686_v15.pt -> Label: 27
File: r_mel_spec_686_v16.pt -> Label: 27
File: r_mel_spec_686_v17.pt -> Label: 27
File: r_mel_spec_686_v18.pt -> Label: 27
File: r_mel_spec_686_v19.pt -> Label: 27
File: r_mel_spec_687_v0.pt -> Label: 27
File: r_mel_spec_687_v1.pt -> Label: 27
File: r_mel_spec_687_v2.pt -> Label: 27
File: r_mel_spec_687_v3.pt -> Label: 27
File: r_mel_spec_687_v4.pt -> Label: 27
File: r_mel_spec_687_v5.pt -> Label: 27
File: r_mel_spec_687_v6.pt -> Label: 27
File: r_mel_spec_687_v7.pt -> Label: 27
File: r_mel_spec_687_v8.pt -> Label: 27
File: r_mel_spec_687_v9.pt -> Label: 27
File: r_mel_spec_687_v10.pt -> Label: 27
File: r_mel_spec_687_v11.pt -> Label: 27
File: r_mel_spec_687_v12.pt -> Label: 27
File: r_mel_spec_687_v13.pt -> Label: 27
File: r_mel_spec_687_v14.pt -> Label: 27
File: r_mel_spec_687_v15.pt -> Label: 27
File: r_mel_spec_687_v16.pt -> Label: 27
File: r_mel_spec_687_v17.pt -> Label: 27
File: r_mel_spec_687_v18.pt -> Label: 27
File: r_mel_spec_687_v19.pt -> Label: 27
File: r_mel_spec_688_v0.pt -> Label: 27
File: r_mel_spec_688_v1.pt -> Label: 27
File: r_mel_spec_688_v2.pt -> Label: 27
File: r_mel_spec_688_v3.pt -> Label: 27
File: r_mel_spec_688_v4.pt -> Label: 27
File: r_mel_spec_688_v5.pt -> Label: 27
File: r_mel_spec_688_v6.pt -> Label: 27
File: r_mel_spec_688_v7.pt -> Label: 27
File: r_mel_spec_688_v8.pt -> Label: 27
File: r_mel_spec_688_v9.pt -> Label: 27
File: r_mel_spec_688_v10.pt -> Label: 27
File: r_mel_spec_688_v11.pt -> Label: 27
File: r_mel_spec_688_v12.pt -> Label: 27
File: r_mel_spec_688_v13.pt -> Label: 27
File: r_mel_spec_688_v14.pt -> Label: 27
File: r_mel_spec_688_v15.pt -> Label: 27
File: r_mel_spec_688_v16.pt -> Label: 27
File: r_mel_spec_688_v17.pt -> Label: 27
File: r_mel_spec_688_v18.pt -> Label: 27
File: r_mel_spec_688_v19.pt -> Label: 27
File: r_mel_spec_689_v0.pt -> Label: 27
File: r_mel_spec_689_v1.pt -> Label: 27
File: r_mel_spec_689_v2.pt -> Label: 27
File: r_mel_spec_689_v3.pt -> Label: 27
File: r_mel_spec_689_v4.pt -> Label: 27
File: r_mel_spec_689_v5.pt -> Label: 27
File: r_mel_spec_689_v6.pt -> Label: 27
File: r_mel_spec_689_v7.pt -> Label: 27
File: r_mel_spec_689_v8.pt -> Label: 27
File: r_mel_spec_689_v9.pt -> Label: 27
File: r_mel_spec_689_v10.pt -> Label: 27
File: r_mel_spec_689_v11.pt -> Label: 27
File: r_mel_spec_689_v12.pt -> Label: 27
File: r_mel_spec_689_v13.pt -> Label: 27
File: r_mel_spec_689_v14.pt -> Label: 27
File: r_mel_spec_689_v15.pt -> Label: 27
File: r_mel_spec_689_v16.pt -> Label: 27
File: r_mel_spec_689_v17.pt -> Label: 27
File: r_mel_spec_689_v18.pt -> Label: 27
File: r_mel_spec_689_v19.pt -> Label: 27
File: r_mel_spec_690_v0.pt -> Label: 27
File: r_mel_spec_690_v1.pt -> Label: 27
File: r_mel_spec_690_v2.pt -> Label: 27
File: r_mel_spec_690_v3.pt -> Label: 27
File: r_mel_spec_690_v4.pt -> Label: 27
File: r_mel_spec_690_v5.pt -> Label: 27
File: r_mel_spec_690_v6.pt -> Label: 27
File: r_mel_spec_690_v7.pt -> Label: 27
File: r_mel_spec_690_v8.pt -> Label: 27
File: r_mel_spec_690_v9.pt -> Label: 27
File: r_mel_spec_690_v10.pt -> Label: 27
File: r_mel_spec_690_v11.pt -> Label: 27
File: r_mel_spec_690_v12.pt -> Label: 27
File: r_mel_spec_690_v13.pt -> Label: 27
File: r_mel_spec_690_v14.pt -> Label: 27
File: r_mel_spec_690_v15.pt -> Label: 27
File: r_mel_spec_690_v16.pt -> Label: 27
File: r_mel_spec_690_v17.pt -> Label: 27
File: r_mel_spec_690_v18.pt -> Label: 27
File: r_mel_spec_690_v19.pt -> Label: 27
File: r_mel_spec_691_v0.pt -> Label: 27

lab_research.google.com/drive/1nBiAl_HU6cF

[illegible]

lab.research.google.com/drive/1pRiALHU6oF

lab.research.google.com/drive/1pRjALHU6oF

lab research google.com/drive/1pRiAl_HU6oF

lab.research.google.com/drive/1pRjALHU6oF

lab research google.com/drive/1pRiAl_HU6oF

[illegible]

— 100 —

[illegible]

— 17 —

<https://colab.research.google.com/drive/1pRjALHU6oFLhBqJnvJ7T546r5kyq3Zmd>

— 116 —

[illegible]

— 117 —

<https://colab.research.google.com/drive/1pRjALHU6oFLhBgJnvJ7T546r5kyq3Zmd>

<https://colab.research.google.com/drive/1pRjALHU6oFLhBqJnvJ7T546r5kyq3Zmd>

[illegible]

— — — — —

[illegible]

<https://colab.research.google.com/drive/1pRjALHU6oFLhBgJnvJ7T546r5kyq3Zmd>

[illegible]

2000 1000 500 0

[illegible]

lab.research.google.com/drive/1pRjALHU6o

<https://colab.research.google.com/drive/1pRjALHU6oFLhBqJnvJ7T546r5kyq3Zmd>

4.1.4. DIALING

[illegible]

lab.research.google.com/drive/1pRjALHU6oI

[illegible]

lab.research.google.com/drive/1pRjALHU6oI

[illegible]

<https://colab.research.google.com/drive/1pRiALHU6oFLhBqJnvJ7T546r5kyq3Zmd>

<https://colab.research.google.com/drive/1pRjALHU6oFLhBqJnvJ7T546r5kyq3Zmd>

— — — — —

[illegible]

<https://colab.research.google.com/drive/1pRjALHU6oFLhBqJnvJ7T546r5kyq3Zmd>

<https://colab.research.google.com/drive/1pRjALHU6oFLhBqJnvJ7T546r5kyq3Zmd>

1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35. 36. 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50. 51. 52. 53. 54. 55. 56. 57. 58. 59. 60. 61. 62. 63. 64. 65. 66. 67. 68. 69. 70. 71. 72. 73. 74. 75. 76. 77. 78. 79. 80. 81. 82. 83. 84. 85. 86. 87. 88. 89. 90. 91. 92. 93. 94. 95. 96. 97. 98. 99. 100. 101. 102. 103. 104. 105. 106. 107. 108. 109. 110. 111. 112. 113. 114. 115. 116. 117. 118. 119. 120. 121. 122. 123. 124. 125. 126. 127. 128. 129. 130. 131. 132. 133. 134. 135. 136. 137. 138. 139. 140. 141. 142. 143. 144. 145. 146. 147. 148. 149. 150. 151. 152. 153. 154. 155. 156. 157. 158. 159. 160. 161. 162. 163. 164. 165. 166. 167. 168. 169. 170. 171. 172. 173. 174. 175. 176. 177. 178. 179. 180. 181. 182. 183. 184. 185. 186. 187. 188. 189. 190. 191. 192. 193. 194. 195. 196. 197. 198. 199. 200. 201. 202. 203. 204. 205. 206. 207. 208. 209. 210. 211. 212. 213. 214. 215. 216. 217. 218. 219. 220. 221. 222. 223. 224. 225. 226. 227. 228. 229. 230. 231. 232. 233. 234. 235. 236. 237. 238. 239. 240. 241. 242. 243. 244. 245. 246. 247. 248. 249. 250. 251. 252. 253. 254. 255. 256. 257. 258. 259. 260. 261. 262. 263. 264. 265. 266. 267. 268. 269. 270. 271. 272. 273. 274. 275. 276. 277. 278. 279. 280. 281. 282. 283. 284. 285. 286. 287. 288. 289. 290. 291. 292. 293. 294. 295. 296. 297. 298. 299. 300. 301. 302. 303. 304. 305. 306. 307. 308. 309. 310. 311. 312. 313. 314. 315. 316. 317. 318. 319. 320. 321. 322. 323. 324. 325. 326. 327. 328. 329. 330. 331. 332. 333. 334. 335. 336. 337. 338. 339. 340. 341. 342. 343. 344. 345. 346. 347. 348. 349. 350. 351. 352. 353. 354. 355. 356. 357. 358. 359. 360. 361. 362. 363. 364. 365. 366. 367. 368. 369. 370. 371. 372. 373. 374. 375. 376. 377. 378. 379. 380. 381. 382. 383. 384. 385. 386. 387. 388. 389. 390. 391. 392. 393. 394. 395. 396. 397. 398. 399. 400. 401. 402. 403. 404. 405. 406. 407. 408. 409. 410. 411. 412. 413. 414. 415. 416. 417. 418. 419. 420. 421. 422. 423. 424. 425. 426. 427. 428. 429. 430. 431. 432. 433. 434. 435. 436. 437. 438. 439. 440. 441. 442. 443. 444. 445. 446. 447. 448. 449. 450. 451. 452. 453. 454. 455. 456. 457. 458. 459. 460. 461. 462. 463. 464. 465. 466. 467. 468. 469. 470. 471. 472. 473. 474. 475. 476. 477. 478. 479. 480. 481. 482. 483. 484. 485. 486. 487. 488. 489. 490. 491. 492. 493. 494. 495. 496. 497. 498. 499. 500. 501. 502. 503. 504. 505. 506. 507. 508. 509. 510. 511. 512. 513. 514. 515. 516. 517. 518. 519. 520. 521. 522. 523. 524. 525. 526. 527. 528. 529. 530. 531. 532. 533. 534. 535. 536. 537. 538. 539. 540. 541. 542. 543. 544. 545. 546. 547. 548. 549. 550. 551. 552. 553. 554. 555. 556. 557. 558. 559. 560. 561. 562. 563. 564. 565. 566. 567. 568. 569. 570. 571. 572. 573. 574. 575. 576. 577. 578. 579. 580. 581. 582. 583. 584. 585. 586. 587. 588. 589. 590. 591. 592. 593. 594. 595. 596. 597. 598. 599. 600. 601. 602. 603. 604. 605. 606. 607. 608. 609. 610. 611. 612. 613. 614. 615. 616. 617. 618. 619. 620. 621. 622. 623. 624. 625. 626. 627. 628. 629. 630. 631. 632. 633. 634. 635. 636. 637. 638. 639. 640. 641. 642. 643. 644. 645. 646. 647. 648. 649. 650. 651. 652. 653. 654. 655. 656. 657. 658. 659. 660. 661. 662. 663. 664. 665. 666. 667. 668. 669. 670. 671. 672. 673. 674. 675. 676. 677. 678. 679. 680. 681. 682. 683. 684. 685. 686. 687. 688. 689. 690. 691. 692. 693. 694. 695. 696. 697. 698. 699. 700. 701. 702. 703. 704. 705. 706. 707. 708. 709. 710. 711. 712. 713. 714. 715. 716. 717. 718. 719. 720. 721. 722. 723. 724. 725. 726. 727. 728. 729. 730. 731. 732. 733. 734. 735. 736. 737. 738. 739. 740. 741. 742. 743. 744. 745. 746. 747. 748. 749. 750. 751. 752. 753. 754. 755. 756. 757. 758. 759. 760. 761. 762. 763. 764. 765. 766. 767. 768. 769. 770. 771. 772. 773. 774. 775. 776. 777. 778. 779. 780. 781. 782. 783. 784. 785. 786. 787. 788. 789. 790. 791. 792. 793. 794. 795. 796. 797. 798. 799. 800. 801. 802. 803. 804. 805. 806. 807. 808. 809. 810. 811. 812. 813. 814. 815. 816. 817. 818. 819. 820. 821. 822. 823. 824. 825. 826. 827. 828. 829. 830. 831. 832. 833. 834. 835. 836. 837. 838. 839. 840. 84

[illegible]

11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 10

[illegible]

Figure 1

[illegible]

— 1 —

<https://colab.research.google.com/drive/1pRjALHU6oFLhBqJnvJ7T546r5kyq3Zmd>

```
File: z_mel_spec_899_v4.pt -> Label: 35
File: z_mel_spec_899_v5.pt -> Label: 35
File: z_mel_spec_899_v6.pt -> Label: 35
File: z_mel_spec_899_v7.pt -> Label: 35
File: z_mel_spec_899_v8.pt -> Label: 35
File: z_mel_spec_899_v9.pt -> Label: 35
File: z_mel_spec_899_v10.pt -> Label: 35
File: z_mel_spec_899_v11.pt -> Label: 35
File: z_mel_spec_899_v12.pt -> Label: 35
File: z_mel_spec_899_v13.pt -> Label: 35
File: z_mel_spec_899_v14.pt -> Label: 35
File: z_mel_spec_899_v15.pt -> Label: 35
File: z_mel_spec_899_v16.pt -> Label: 35
File: z_mel_spec_899_v17.pt -> Label: 35
File: z_mel_spec_899_v18.pt -> Label: 35
File: z_mel_spec_899_v19.pt -> Label: 35
File: z_mel_spec_900_v0.pt -> Label: 35
File: z_mel_spec_900_v1.pt -> Label: 35
File: z_mel_spec_900_v2.pt -> Label: 35
File: z_mel_spec_900_v3.pt -> Label: 35
File: z_mel_spec_900_v4.pt -> Label: 35
File: z_mel_spec_900_v5.pt -> Label: 35
File: z_mel_spec_900_v6.pt -> Label: 35
File: z_mel_spec_900_v7.pt -> Label: 35
File: z_mel_spec_900_v8.pt -> Label: 35
File: z_mel_spec_900_v9.pt -> Label: 35
File: z_mel_spec_900_v10.pt -> Label: 35
File: z_mel_spec_900_v11.pt -> Label: 35
File: z_mel_spec_900_v12.pt -> Label: 35
File: z_mel_spec_900_v13.pt -> Label: 35
File: z_mel_spec_900_v14.pt -> Label: 35
File: z_mel_spec_900_v15.pt -> Label: 35
File: z_mel_spec_900_v16.pt -> Label: 35
File: z_mel_spec_900_v17.pt -> Label: 35
File: z_mel_spec_900_v18.pt -> Label: 35
File: z_mel_spec_900_v19.pt -> Label: 35
Total number of samples in the dataset: 18000
Train labels distribution: [394 419 390 412 393 399 397 398 395 395 403 401 391 403 400 387 402 402
 399 390 402 408 385 411 397 409 398 392 407 401 401 403 403 409 407 397]
Validation labels distribution: [49 43 48 48 54 47 53 51 48 51 52 59 52 47 57 51 57 46 46 48 56 47 49 55
 57 36 50 52 52 55 44 46 52 42 44 56]
Test labels distribution: [57 38 62 40 53 54 50 51 57 54 45 40 57 50 43 62 41 52 55 62 42 45 66 34
 46 55 52 56 41 44 55 51 45 49 49 47]
Total dataset size: 18000
Training dataset size: 14400
Validation dataset size: 1800
Test dataset size: 1800
Loading checkpoint...
Resumed from epoch 1067, best validation accuracy so far: 3.17%
Checkpoint saved at epoch 1068
Epoch [1068/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1069
Epoch [1069/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1070
Epoch [1070/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1071
Epoch [1071/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1072
Epoch [1072/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1073
Epoch [1073/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1074
Epoch [1074/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1075
Epoch [1075/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1076
Epoch [1076/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1077
Epoch [1077/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1078
Epoch [1078/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1079
Epoch [1079/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1080
Epoch [1080/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1081
Epoch [1081/1300] - Loss: 1612.5090, Accuracy: 2.91%
```

<https://colab.research.google.com/drive/1pRjALHU6oFLhBqJnvJ7T546r5kyq3Zmd>

lab.research.google.com/drive/1pRiALHU6oFLhBqJnvJ7T546r5kyg3Zmd

<https://colab.research.google.com/drive/1pRjALHU6oFLhBqJnvJ7T546r5kyq3Zmd>

— 1 —

<https://colab.research.google.com/drive/1pRjALHU6oFLhBgJnvJ7T546r5kyq3Zmd>

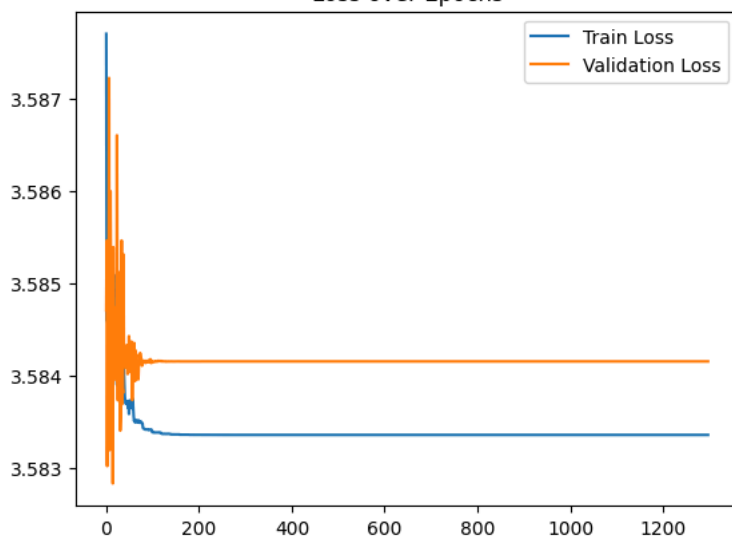
<https://colab.research.google.com/drive/1pRjALHU6oFLhBgJnvJ7T546r5kyq3Zmd>

<https://colab.research.google.com/drive/1pRjALHU6oFLhBqJnvJ7T546r5kyq3Zmd>

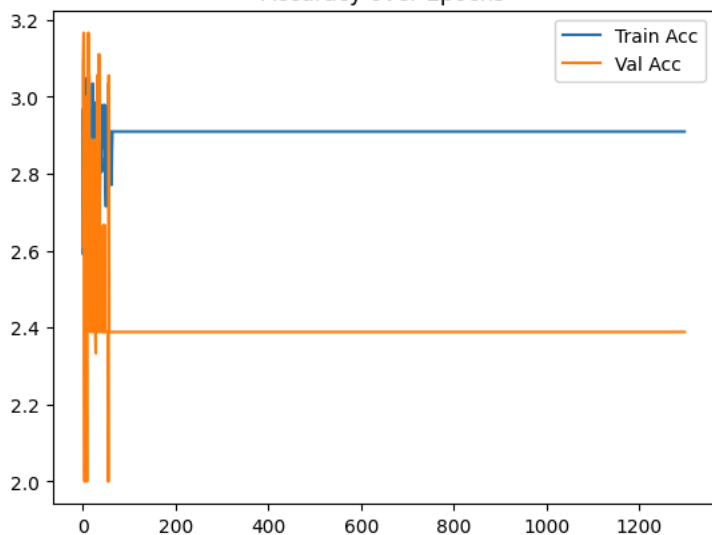
Checkpoint saved at epoch 1293
Epoch [1293/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1294
Epoch [1294/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1295
Epoch [1295/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1296
Epoch [1296/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1297
Epoch [1297/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1298
Epoch [1298/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1299
Epoch [1299/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%
Checkpoint saved at epoch 1300
Epoch [1300/1300] - Loss: 1612.5090, Accuracy: 2.91%
Validation - Loss: 3.5842, Accuracy: 2.39%

Peak Validation Accuracy: 3.17%

Loss over Epochs



Accuracy over Epochs



Confusion Matrix

