

Complex PyTorch for Music Genre Classification

```
In [23]: # Complex pytorch
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from complexPyTorch.complexLayers import *
from complexPyTorch.complexFunctions import *

# Plot
import matplotlib.pyplot as plt
import seaborn as sns
import time

# Load Data
import numpy as np
import json
import os
import math
import librosa
import pathlib
from scipy.spatial.distance import cdist
from torch.utils.data import Dataset
from sklearn.model_selection import train_test_split
import random

# MFCCS
from scipy.io import wavfile
import scipy.fftpack as fft
from scipy.signal import get_window
```

```
In [2]: import os
import shutil
from sklearn.model_selection import train_test_split

# Path to the folder containing subfolders for each class
data_folder = 'Data/genres_original'

# List of subfolder names (class names)
class_names = os.listdir(data_folder)
class_names.remove('.DS_Store')

# Create train and test directories
train_dir = 'data/train'
test_dir = 'data/test'
os.makedirs(train_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)

# Split ratio (adjust as needed)
test_size = 0.16

# Loop through each class
for class_name in class_names:
    class_folder = os.path.join(data_folder, class_name)
```

```

class_files = os.listdir(class_folder)

# Split files into train and test sets
train_files, test_files = train_test_split(class_files, test_size=tes

# Create class subdirectories in train and test folders
train_class_dir = os.path.join(train_dir, class_name)
test_class_dir = os.path.join(test_dir, class_name)
os.makedirs(train_class_dir, exist_ok=True)
os.makedirs(test_class_dir, exist_ok=True)

# Move train files
for train_file in train_files:
    src_path = os.path.join(class_folder, train_file)
    dest_path = os.path.join(train_class_dir, train_file)
    shutil.copy(src_path, dest_path)

# Move test files
for test_file in test_files:
    src_path = os.path.join(class_folder, test_file)
    dest_path = os.path.join(test_class_dir, test_file)
    shutil.copy(src_path, dest_path)

print("Data split and saved successfully.")

```

Data split and saved successfully.

```

In [3]: def train(model, device, train_loader, test_loader, optimizer, epoch, met
model.train()
total_loss = 0
correct = 0
total_samples = len(train_loader.dataset)
start_time = time.time()

for batch_idx, (data, target) in enumerate(train_loader):
    data, target = data.to(device), target.to(device)
    if complexify: data = data.type(torch.complex64)
    if data_fn != None: data = data_fn(data)
    optimizer.zero_grad()
    output = model(data)
    loss = F.nll_loss(output, target)
    loss.backward()
    optimizer.step()
    total_loss += loss.item()
    pred = output.argmax(dim=1, keepdim=True)
    correct += pred.eq(target.view_as(pred)).sum().item()

    if batch_idx % 100 == 0:
        batch_accuracy = 100. * correct / ((batch_idx + 1) * len(data)
        print('Train Epoch: {:3} [{:6}/{:6} ({:3.0f}%)]\tLoss: {:.6f}
            epoch,
            batch_idx * len(data),
            total_samples,
            100. * batch_idx / len(train_loader),
            loss.item(),
            batch_accuracy)

end_time = time.time()
epoch_times = metrics_dict['epoch_times']

```

```

epoch_times.append(end_time - start_time)
epoch_loss = total_loss / len(train_loader)
epoch_accuracy = 100. * correct / total_samples
train_losses = metrics_dict['train_losses']
train_accuracies = metrics_dict['train_accuracies']
train_losses.append(epoch_loss)
train_accuracies.append(epoch_accuracy)
print('Epoch {} - Time: {:.2f}s - Train Loss: {:.6f} - Train Accuracy

# Evaluate on test data
model.eval()
test_loss = 0
correct = 0
with torch.no_grad():
    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        if complexify:
            data = data.type(torch.complex64)
        output = model(data)
        test_loss += F.nll_loss(output, target, reduction='sum').item()
        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()

test_loss /= len(test_loader.dataset)
test_accuracy = 100. * correct / len(test_loader.dataset)
test_losses = metrics_dict['test_losses']
test_accuracies = metrics_dict['test_accuracies']
test_losses.append(test_loss)
test_accuracies.append(test_accuracy)
print('Test Loss: {:.6f} - Test Accuracy: {:.2f}%\n'.format(test_loss

```

Data Preparation

```

In [4]: DATASET_PATH = "Data/genres_original/"
SAMPLE_RATE = 22050
TRACK_DURATION = 30 # measured in seconds
SAMPLES_PER_TRACK = SAMPLE_RATE * TRACK_DURATION
BATCH_SIZE = 32
NUM_EPOCHS = 20

```

```

In [5]: genre_list = os.listdir(DATASET_PATH)
if '.DS_Store' in genre_list: genre_list.remove('.DS_Store')
genre_mappings = dict(zip(genre_list, range(len(genre_list))))
print(genre_mappings)

```

```

{'pop': 0, 'metal': 1, 'disco': 2, 'blues': 3, 'reggae': 4, 'classical':
5, 'rock': 6, 'hiphop': 7, 'country': 8, 'jazz': 9}

```

MFCCS

```

In [6]: class MusicFeatureExtractor:
    def __init__(self, FFT_size=2048, HOP_SIZE=512, mel_filter_num=13, dc
        self.FFT_size = FFT_size
        self.HOP_SIZE = HOP_SIZE
        self.mel_filter_num = mel_filter_num
        self.dct_filter_num = dct_filter_num
        self.epsilon = 1e-10 # Added to log to avoid log10(0)

```

```

def normalize_audio(self, audio):
    audio = audio / np.max(np.abs(audio))
    return audio

def frame_audio(self, audio):
    frame_num = int((len(audio) - self.FFT_size) / self.HOP_SIZE) + 1
    frames = np.zeros((frame_num, self.FFT_size))
    for n in range(frame_num):
        frames[n] = audio[n * self.HOP_SIZE: n * self.HOP_SIZE + self.FFT_size]
    return frames

def freq_to_mel(self, freq):
    return 2595.0 * np.log10(1.0 + freq / 700.0)

def mel_to_freq(self, mels):
    return 700.0 * (10.0 ** (mels / 2595.0) - 1.0)

def get_filter_points(self, fmin, fmax, sample_rate):
    fmin_mel = self.freq_to_mel(fmin)
    fmax_mel = self.freq_to_mel(fmax)
    mels = np.linspace(fmin_mel, fmax_mel, num=self.mel_filter_num + 1)
    freqs = self.mel_to_freq(mels)
    return np.floor((self.FFT_size + 1) / sample_rate * freqs).astype(int)

def get_filters(self, filter_points):
    filters = np.zeros((len(filter_points) - 2, int(self.FFT_size / 2)))
    for n in range(len(filter_points) - 2):
        filters[n, filter_points[n]: filter_points[n + 1]] = np.linspace(0, 1, filter_points[n + 1] - filter_points[n])
        filters[n, filter_points[n + 1]: filter_points[n + 2]] = np.linspace(1, 0, filter_points[n + 2] - filter_points[n + 1])
    return filters

def dct(self):
    basis = np.empty((self.dct_filter_num, self.mel_filter_num))
    basis[0, :] = 1.0 / np.sqrt(self.mel_filter_num)
    samples = np.arange(1, 2 * self.mel_filter_num, 2) * np.pi / (2.0 * self.mel_filter_num)
    for i in range(1, self.dct_filter_num):
        basis[i, :] = np.cos(i * samples) * np.sqrt(2.0 / self.mel_filter_num)
    return basis

def get_mfcc_features(self, audio, sample_rate):
    audio = self.normalize_audio(audio)
    audio_framed = self.frame_audio(audio)
    window = get_window("hann", self.FFT_size, fftbins=True)
    audio_win = audio_framed * window
    audio_winT = np.transpose(audio_win)
    audio_fft = np.empty((int(1 + self.FFT_size // 2), audio_winT.shape[1]))
    for n in range(audio_fft.shape[1]):
        audio_fft[:, n] = fft.fft(audio_winT[:, n], axis=0)[:audio_fft.shape[0]]
    audio_fft = np.transpose(audio_fft)
    audio_fft = np.square(np.abs(audio_fft))
    freq_min = 0
    freq_high = sample_rate / 2
    filter_points, mel_freqs = self.get_filter_points(freq_min, freq_high)
    filters = self.get_filters(filter_points)
    audio_filtered = np.dot(filters, np.transpose(audio_fft))
    audio_filtered = np.maximum(audio_filtered, self.epsilon) # Replace 0 with self.epsilon
    audio_log = 10.0 * np.log10(audio_filtered)
    dct_filters = self.dct()
    cepstral_coefficients = np.dot(dct_filters, audio_log)
    return np.array([cepstral_coefficients])

```

```

class MusicFeatureExtractorComplex:
    def __init__(self, FFT_size=2048, HOP_SIZE=512, mel_filter_num=13, dct_filter_num=13, epsilon=1e-10):
        self.FFT_size = FFT_size
        self.HOP_SIZE = HOP_SIZE
        self.mel_filter_num = mel_filter_num
        self.dct_filter_num = dct_filter_num
        self.epsilon = 1e-10 # Added to log to avoid log10(0)

    def normalize_audio(self, audio):
        audio = audio / np.max(np.abs(audio))
        return audio

    def frame_audio(self, audio):
        frame_num = int((len(audio) - self.FFT_size) / self.HOP_SIZE) + 1
        frames = np.zeros((frame_num, self.FFT_size))
        for n in range(frame_num):
            frames[n] = audio[n * self.HOP_SIZE: n * self.HOP_SIZE + self.FFT_size]
        return frames

    def freq_to_mel(self, freq):
        return 2595.0 * np.log10(1.0 + freq / 700.0)

    def mel_to_freq(self, mels):
        return 700.0 * (10.0 ** (mels / 2595.0) - 1.0)

    def get_filter_points(self, fmin, fmax, sample_rate):
        fmin_mel = self.freq_to_mel(fmin)
        fmax_mel = self.freq_to_mel(fmax)
        mels = np.linspace(fmin_mel, fmax_mel, num=self.mel_filter_num + 1)
        freqs = self.mel_to_freq(mels)
        return np.floor((self.FFT_size + 1) / sample_rate * freqs).astype(int)

    def get_filters(self, filter_points):
        filters = np.zeros((len(filter_points) - 2, int(self.FFT_size / 2)))
        for n in range(len(filter_points) - 2):
            filters[n, filter_points[n]: filter_points[n + 1]] = np.linspace(0, 1, filter_points[n + 1] - filter_points[n])
            filters[n, filter_points[n + 1]: filter_points[n + 2]] = np.linspace(1, 0, filter_points[n + 2] - filter_points[n + 1])
        return filters

    def dct(self):
        basis = np.empty((self.dct_filter_num, self.mel_filter_num))
        basis[0, :] = 1.0 / np.sqrt(self.mel_filter_num)
        samples = np.arange(1, 2 * self.mel_filter_num, 2) * np.pi / (2.0 * self.FFT_size)
        for i in range(1, self.dct_filter_num):
            basis[i, :] = np.cos(i * samples) * np.sqrt(2.0 / self.mel_filter_num)
        return basis

    def get_mfcc_features(self, audio, sample_rate):
        audio = self.normalize_audio(audio)
        audio_framed = self.frame_audio(audio)
        window = get_window("hann", self.FFT_size, fftbins=True)
        audio_win = audio_framed * window
        audio_winT = np.transpose(audio_win)
        audio_fft = np.empty((int(1 + self.FFT_size // 2), audio_winT.shape[1]))
        for n in range(audio_fft.shape[1]):
            audio_fft[:, n] = fft.fft(audio_winT[:, n], axis=0)[:audio_fft.shape[0]]
        audio_fft = np.transpose(audio_fft)
        freq_min = 0
        freq_high = sample_rate / 2

```

```

filter_points, mel_freqs = self.get_filter_points(freq_min, freq_max)
filters = self.get_filters(filter_points)
audio_filtered = np.dot(filters, np.transpose(audio_fft))
audio_filtered[audio_filtered == 0] = self.epsilon # Replace zero
audio_log = 10.0 * np.log10(audio_filtered)
dct_filters = self.dct()
cepstral_coefficients = np.dot(dct_filters, audio_log)
return np.array([cepstral_coefficients])

```

In [7]: **class** GenreDatasetMFCC(Dataset):

```

def __init__(self, train_path, n_fft=2048, hop_length=512, num_segments=10):
    cur_path = pathlib.Path(train_path)
    self.files = []
    for i in list(cur_path.rglob("*.wav")):
        for j in range(num_segments):
            self.files.append([j, i])
    self.samples_per_segment = int(SAMPLES_PER_TRACK / num_segments)
    self.n_fft = n_fft
    self.hop_length = hop_length
    self.num_segments = num_segments
    self.mfcc_extractor = MusicFeatureExtractor(
        FFT_size=n_fft, HOP_SIZE=hop_length, mel_filter_num = mel_filters,
        dct_filter_num = dct_filters)
    self.training = training

def apply_augmentations(self, signal):
    # Apply augmentations to the audio signal
    if random.random() < 0.5:
        signal = librosa.effects.pitch_shift(signal, sr=SAMPLE_RATE,
        if random.random() < 0.5:
            signal = librosa.effects.time_stretch(signal, rate=random.uniform(0.8, 1.2))
    return signal

def adjust_shape(self, sequence, max_sequence_length = 126):
    current_length = sequence.shape[2]
    if current_length < max_sequence_length:
        padding = np.zeros((1, 13, max_sequence_length - current_length))
        padded_sequence = np.concatenate((sequence, padding), axis=2)
    else:
        padded_sequence = sequence[:, :, :max_sequence_length]
    return padded_sequence

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

def __getitem__(self, idx):
    cur_file = self.files[idx]
    d = cur_file[0]
    file_path = cur_file[1]
    target = genre_mappings[str(file_path).split("/")[-2]]
    signal, sample_rate = librosa.load(file_path, sr=SAMPLE_RATE)
    start = self.samples_per_segment * d
    finish = start + self.samples_per_segment
    cur_signal = signal[start:finish]
    if self.training: cur_signal = self.apply_augmentations(cur_signal)
    cur_mfcc = self.mfcc_extractor.get_mfcc_features(cur_signal, sample_rate)
    cur_mfcc = self.adjust_shape(cur_mfcc)
    return torch.tensor(cur_mfcc, dtype=torch.float32), target

```

```

class GenreDatasetPhaseMFCC(GenreDatasetMFCC):

    def __init__(self, train_path, n_fft=2048, hop_length=512, num_segments, mel_filters):
        super().__init__(train_path, n_fft, hop_length, num_segments, mel_filters)
        self.mfcc_extractor = MusicFeatureExtractorComplex(
            FFT_size=n_fft, HOP_SIZE=hop_length, mel_filter_num = mel_filters)

    def __getitem__(self, idx):
        cur_file = self.files[idx]
        d = cur_file[0]
        file_path = cur_file[1]
        target = genre_mappings[str(file_path).split("/")][2]
        signal, sample_rate = librosa.load(file_path, sr=SAMPLE_RATE)
        start = self.samples_per_segment * d
        finish = start + self.samples_per_segment
        cur_signal = signal[start:finish]
        if self.training: cur_signal = self.apply_augmentations(cur_signal)
        cur_mfcc = self.mfcc_extractor.get_mfcc_features(cur_signal, sample_rate)
        cur_mfcc = self.adjust_shape(cur_mfcc)
        return torch.tensor(cur_mfcc, dtype=torch.complex64), target

```

1. No phase data

```

In [8]: train_dataset = GenreDatasetMFCC("Data/train/", n_fft=2048, hop_length=512, num_segments=10, mel_filters=12)
test_dataset = GenreDatasetMFCC("Data/test/", n_fft=2048, hop_length=512, num_segments=10, mel_filters=12)
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, shuffle=False)

```

```

In [9]: class RealNet(nn.Module):

```

```

    def __init__(self):
        super(RealNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, 2, 1)
        self.bn = nn.BatchNorm2d(10)
        self.conv2 = nn.Conv2d(10, 20, 2, 1)
        self.fc1 = nn.Linear(30*2*20, 500)
        self.fc2 = nn.Linear(500, 10)

```

```

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2, 2)
        x = self.bn(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 30*2*20)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = x.abs()
        x = F.log_softmax(x, dim=1)
        return x

```

```

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = RealNet().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

```

```
metrics_dict_e1 = {
    'epoch_times': [],
    'train_losses': [],
    'train_accuracies': [],
    'test_losses': [],
    'test_accuracies': []
}

for epoch in range(NUM_EPOCHS):
    train(model,
          device,
          train_loader,
          test_loader,
          optimizer,
          epoch,
          metrics_dict_e1,
          complexify = False)

print("-"*100)
print("-"*100)
print("FINAL RESULTS:")
print("-"*100)
for key, value in metrics_dict_e1.items():
    print(f'{key}: {value}')
```


Train Epoch: 0 [0/ 8390 (0%)] Loss: 2.316185 Accuracy: 3.12%
Train Epoch: 0 [3200/ 8390 (38%)] Loss: 2.077888 Accuracy: 24.47%
Train Epoch: 0 [6400/ 8390 (76%)] Loss: 1.776363 Accuracy: 30.16%
Epoch 0 – Time: 176.49s – Train Loss: 1.842231 – Train Accuracy: 32.15%
Test Loss: 1.759411 – Test Accuracy: 38.69%

Train Epoch: 1 [0/ 8390 (0%)] Loss: 1.773670 Accuracy: 40.62%
Train Epoch: 1 [3200/ 8390 (38%)] Loss: 1.957644 Accuracy: 43.38%
Train Epoch: 1 [6400/ 8390 (76%)] Loss: 1.535713 Accuracy: 43.69%
Epoch 1 – Time: 175.82s – Train Loss: 1.561373 – Train Accuracy: 43.74%
Test Loss: 1.565164 – Test Accuracy: 41.88%

Train Epoch: 2 [0/ 8390 (0%)] Loss: 1.569602 Accuracy: 40.62%
Train Epoch: 2 [3200/ 8390 (38%)] Loss: 1.500369 Accuracy: 48.55%
Train Epoch: 2 [6400/ 8390 (76%)] Loss: 1.554265 Accuracy: 49.10%
Epoch 2 – Time: 182.97s – Train Loss: 1.428993 – Train Accuracy: 48.75%
Test Loss: 1.465071 – Test Accuracy: 46.12%

Train Epoch: 3 [0/ 8390 (0%)] Loss: 1.148391 Accuracy: 59.38%
Train Epoch: 3 [3200/ 8390 (38%)] Loss: 1.240188 Accuracy: 50.84%
Train Epoch: 3 [6400/ 8390 (76%)] Loss: 1.676979 Accuracy: 50.84%
Epoch 3 – Time: 175.02s – Train Loss: 1.357373 – Train Accuracy: 50.77%
Test Loss: 1.657636 – Test Accuracy: 43.38%

Train Epoch: 4 [0/ 8390 (0%)] Loss: 0.997623 Accuracy: 59.38%
Train Epoch: 4 [3200/ 8390 (38%)] Loss: 1.197162 Accuracy: 53.87%
Train Epoch: 4 [6400/ 8390 (76%)] Loss: 1.274162 Accuracy: 53.34%
Epoch 4 – Time: 187.68s – Train Loss: 1.274983 – Train Accuracy: 53.99%
Test Loss: 1.743243 – Test Accuracy: 44.62%

Train Epoch: 5 [0/ 8390 (0%)] Loss: 1.174444 Accuracy: 53.12%
Train Epoch: 5 [3200/ 8390 (38%)] Loss: 1.106565 Accuracy: 56.25%
Train Epoch: 5 [6400/ 8390 (76%)] Loss: 1.058037 Accuracy: 56.16%
Epoch 5 – Time: 179.67s – Train Loss: 1.214539 – Train Accuracy: 56.94%
Test Loss: 1.398109 – Test Accuracy: 51.44%

Train Epoch: 6 [0/ 8390 (0%)] Loss: 0.910724 Accuracy: 62.50%
Train Epoch: 6 [3200/ 8390 (38%)] Loss: 1.512703 Accuracy: 58.88%
Train Epoch: 6 [6400/ 8390 (76%)] Loss: 0.951155 Accuracy: 58.80%
Epoch 6 – Time: 177.37s – Train Loss: 1.167653 – Train Accuracy: 58.67%
Test Loss: 1.608052 – Test Accuracy: 43.88%

Train Epoch: 7 [0/ 8390 (0%)] Loss: 1.153452 Accuracy: 56.25%
Train Epoch: 7 [3200/ 8390 (38%)] Loss: 1.605653 Accuracy: 61.88%
Train Epoch: 7 [6400/ 8390 (76%)] Loss: 1.263472 Accuracy: 61.04%
Epoch 7 – Time: 2114.84s – Train Loss: 1.114955 – Train Accuracy: 60.64%
Test Loss: 1.444405 – Test Accuracy: 50.56%

Train Epoch: 8 [0/ 8390 (0%)] Loss: 0.997894 Accuracy: 59.38%
Train Epoch: 8 [3200/ 8390 (38%)] Loss: 0.927493 Accuracy: 61.36%
Train Epoch: 8 [6400/ 8390 (76%)] Loss: 1.140884 Accuracy: 62.33%
Epoch 8 – Time: 307.90s – Train Loss: 1.058625 – Train Accuracy: 62.94%
Test Loss: 1.366964 – Test Accuracy: 53.06%

Train Epoch: 9 [0/ 8390 (0%)] Loss: 1.066264 Accuracy: 65.62%
Train Epoch: 9 [3200/ 8390 (38%)] Loss: 1.137856 Accuracy: 65.28%
Train Epoch: 9 [6400/ 8390 (76%)] Loss: 1.076037 Accuracy: 64.46%
Epoch 9 – Time: 178.32s – Train Loss: 1.033641 – Train Accuracy: 63.69%
Test Loss: 1.412204 – Test Accuracy: 52.81%

Train Epoch: 10 [0/ 8390 (0%)] Loss: 0.944797 Accuracy: 68.75%
Train Epoch: 10 [3200/ 8390 (38%)] Loss: 0.705615 Accuracy: 67.61%
Train Epoch: 10 [6400/ 8390 (76%)] Loss: 1.042045 Accuracy: 66.34%
Epoch 10 – Time: 174.27s – Train Loss: 0.969864 – Train Accuracy: 66.11%
Test Loss: 1.472573 – Test Accuracy: 52.00%

Train Epoch: 11 [0/ 8390 (0%)] Loss: 1.073031 Accuracy: 65.62%
Train Epoch: 11 [3200/ 8390 (38%)] Loss: 1.021653 Accuracy: 67.14%
Train Epoch: 11 [6400/ 8390 (76%)] Loss: 0.704792 Accuracy: 67.27%
Epoch 11 – Time: 169.76s – Train Loss: 0.925578 – Train Accuracy: 67.58%
Test Loss: 1.561576 – Test Accuracy: 52.25%

Train Epoch: 12 [0/ 8390 (0%)] Loss: 1.120808 Accuracy: 59.38%
Train Epoch: 12 [3200/ 8390 (38%)] Loss: 1.080361 Accuracy: 69.15%
Train Epoch: 12 [6400/ 8390 (76%)] Loss: 1.099876 Accuracy: 68.89%
Epoch 12 – Time: 170.53s – Train Loss: 0.903229 – Train Accuracy: 68.64%
Test Loss: 1.646151 – Test Accuracy: 53.06%

Train Epoch: 13 [0/ 8390 (0%)] Loss: 0.975203 Accuracy: 56.25%
Train Epoch: 13 [3200/ 8390 (38%)] Loss: 0.643761 Accuracy: 71.10%
Train Epoch: 13 [6400/ 8390 (76%)] Loss: 0.590223 Accuracy: 70.88%
Epoch 13 – Time: 171.76s – Train Loss: 0.851645 – Train Accuracy: 70.70%
Test Loss: 1.415567 – Test Accuracy: 53.56%

Train Epoch: 14 [0/ 8390 (0%)] Loss: 0.651511 Accuracy: 81.25%
Train Epoch: 14 [3200/ 8390 (38%)] Loss: 0.735180 Accuracy: 71.88%
Train Epoch: 14 [6400/ 8390 (76%)] Loss: 1.041193 Accuracy: 72.22%
Epoch 14 – Time: 171.69s – Train Loss: 0.811780 – Train Accuracy: 72.17%
Test Loss: 1.562119 – Test Accuracy: 54.25%

Train Epoch: 15 [0/ 8390 (0%)] Loss: 0.853154 Accuracy: 65.62%
Train Epoch: 15 [3200/ 8390 (38%)] Loss: 1.031863 Accuracy: 73.51%
Train Epoch: 15 [6400/ 8390 (76%)] Loss: 0.822448 Accuracy: 73.03%
Epoch 15 – Time: 178.36s – Train Loss: 0.778017 – Train Accuracy: 72.94%
Test Loss: 1.672464 – Test Accuracy: 50.94%

Train Epoch: 16 [0/ 8390 (0%)] Loss: 0.686605 Accuracy: 78.12%
Train Epoch: 16 [3200/ 8390 (38%)] Loss: 0.703789 Accuracy: 76.33%
Train Epoch: 16 [6400/ 8390 (76%)] Loss: 0.488479 Accuracy: 74.63%
Epoch 16 – Time: 173.46s – Train Loss: 0.747625 – Train Accuracy: 73.99%
Test Loss: 1.446513 – Test Accuracy: 55.50%

Train Epoch: 17 [0/ 8390 (0%)] Loss: 0.429874 Accuracy: 90.62%
Train Epoch: 17 [3200/ 8390 (38%)] Loss: 0.538253 Accuracy: 76.52%
Train Epoch: 17 [6400/ 8390 (76%)] Loss: 0.879903 Accuracy: 75.25%
Epoch 17 – Time: 174.48s – Train Loss: 0.725593 – Train Accuracy: 75.01%
Test Loss: 1.639021 – Test Accuracy: 51.94%

Train Epoch: 18 [0/ 8390 (0%)] Loss: 1.111452 Accuracy: 65.62%
Train Epoch: 18 [3200/ 8390 (38%)] Loss: 0.493201 Accuracy: 78.06%
Train Epoch: 18 [6400/ 8390 (76%)] Loss: 1.000018 Accuracy: 76.99%
Epoch 18 – Time: 173.06s – Train Loss: 0.691851 – Train Accuracy: 76.60%
Test Loss: 1.669921 – Test Accuracy: 53.88%

Train Epoch: 19 [0/ 8390 (0%)] Loss: 0.497625 Accuracy: 84.38%
Train Epoch: 19 [3200/ 8390 (38%)] Loss: 0.629276 Accuracy: 78.65%
Train Epoch: 19 [6400/ 8390 (76%)] Loss: 1.373169 Accuracy: 77.88%
Epoch 19 – Time: 172.69s – Train Loss: 0.655765 – Train Accuracy: 77.43%
Test Loss: 1.952313 – Test Accuracy: 50.94%

FINAL RESULTS:

```
epoch_times: [176.49259638786316, 175.82231879234314, 182.96968507766724,
175.01975083351135, 187.68068408966064, 179.67215514183044, 177.3721632957
4585, 2114.843255996704, 307.8959949016571, 178.3186070919037, 174.2747299
671173, 169.76471304893494, 170.52626276016235, 171.7625710964203, 171.689
70608711243, 178.36060214042664, 173.4625279903412, 174.47915482521057, 17
3.05771207809448, 172.68643403053284]
train_losses: [1.8422306057151037, 1.561372934634449, 1.4289932237326644,
1.3573728212873444, 1.2749834552066017, 1.214539316546826, 1.1676533497471
846, 1.114954732301581, 1.05862481038989, 1.0336408935885393, 0.9698635776
534336, 0.9255781260155539, 0.9032293719644765, 0.8516452396643981, 0.8117
804636482064, 0.7780174973584314, 0.7476246215346205, 0.725593437561552,
0.6918507014749614, 0.6557649082809914]
train_accuracies: [32.145411203814064, 43.74255065554231, 48.7485101311084
6, 50.774731823599524, 53.99284862932062, 56.93682955899881, 58.6650774731
8236, 60.64362336114422, 62.94398092967819, 63.69487485101311, 66.11442193
087008, 67.58045292014303, 68.64123957091776, 70.70321811680571, 72.169249
10607867, 72.94398092967819, 73.99284862932062, 75.00595947556614, 76.6030
989272944, 77.42550655542313]
test_losses: [1.7594111448526382, 1.5651644814014434, 1.4650710982084274,
1.6576355692744256, 1.7432427006959914, 1.3981089863181113, 1.608052256107
3303, 1.4444050145149232, 1.3669641822576524, 1.4122039565443993, 1.472573
2989609241, 1.5615760169923305, 1.6461506137251853, 1.4155672320723534, 1.
5621185782551765, 1.6724637657403947, 1.4465129046142102, 1.63902138382196
43, 1.6699206562340259, 1.9523125983774663]
test_accuracies: [38.6875, 41.875, 46.125, 43.375, 44.625, 51.4375, 43.87
5, 50.5625, 53.0625, 52.8125, 52.0, 52.25, 53.0625, 53.5625, 54.25, 50.937
5, 55.5, 51.9375, 53.875, 50.9375]
```

```
In [10]: class ComplexNet(nn.Module):

    def __init__(self):
        super(ComplexNet, self).__init__()
        self.conv1 = ComplexConv2d(1, 10, 2, 1)
        self.bn = ComplexBatchNorm2d(10)
        self.conv2 = ComplexConv2d(10, 20, 2, 1)
        self.fc1 = ComplexLinear(30*2*20, 500)
        self.fc2 = ComplexLinear(500, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = complex_relu(x)
        x = complex_max_pool2d(x, 2, 2)
        x = self.bn(x)
        x = self.conv2(x)
        x = complex_relu(x)
        x = complex_max_pool2d(x, 2, 2)
        x = x.view(-1, 30*2*20)
        x = self.fc1(x)
        x = complex_relu(x)
        x = self.fc2(x)
        x = x.abs()
        x = F.log_softmax(x, dim=1)
        return x
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = ComplexNet().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

metrics_dict_e2 = {
    'epoch_times': [],
    'train_losses': [],
    'train_accuracies': [],
    'test_losses': [],
    'test_accuracies': []
}

for epoch in range(NUM_EPOCHS):
    train(model,
          device,
          train_loader,
          test_loader,
          optimizer,
          epoch,
          metrics_dict_e2)

print("-"*100)
print("-"*100)
print("FINAL RESULTS:")
print("-"*100)
for key, value in metrics_dict_e2.items():
    print(f'{key}: {value}')
```

Train Epoch: 0 [0/ 8390 (0%)] Loss: 2.298491 Accuracy: 12.50%
Train Epoch: 0 [3200/ 8390 (38%)] Loss: 1.557620 Accuracy: 33.94%
Train Epoch: 0 [6400/ 8390 (76%)] Loss: 1.463058 Accuracy: 37.13%
Epoch 0 – Time: 207.23s – Train Loss: 1.700529 – Train Accuracy: 38.96%
Test Loss: 1.837249 – Test Accuracy: 37.88%

Train Epoch: 1 [0/ 8390 (0%)] Loss: 1.486362 Accuracy: 50.00%
Train Epoch: 1 [3200/ 8390 (38%)] Loss: 1.224142 Accuracy: 49.57%
Train Epoch: 1 [6400/ 8390 (76%)] Loss: 1.203180 Accuracy: 50.39%
Epoch 1 – Time: 209.31s – Train Loss: 1.394651 – Train Accuracy: 50.64%
Test Loss: 1.502434 – Test Accuracy: 50.50%

Train Epoch: 2 [0/ 8390 (0%)] Loss: 1.243697 Accuracy: 59.38%
Train Epoch: 2 [3200/ 8390 (38%)] Loss: 1.305529 Accuracy: 55.72%
Train Epoch: 2 [6400/ 8390 (76%)] Loss: 1.287207 Accuracy: 55.64%
Epoch 2 – Time: 208.01s – Train Loss: 1.276541 – Train Accuracy: 55.55%
Test Loss: 1.503635 – Test Accuracy: 48.88%

Train Epoch: 3 [0/ 8390 (0%)] Loss: 1.319306 Accuracy: 59.38%
Train Epoch: 3 [3200/ 8390 (38%)] Loss: 0.906786 Accuracy: 60.33%
Train Epoch: 3 [6400/ 8390 (76%)] Loss: 1.429709 Accuracy: 59.17%
Epoch 3 – Time: 218.88s – Train Loss: 1.162604 – Train Accuracy: 59.06%
Test Loss: 1.531845 – Test Accuracy: 48.00%

Train Epoch: 4 [0/ 8390 (0%)] Loss: 0.983407 Accuracy: 71.88%
Train Epoch: 4 [3200/ 8390 (38%)] Loss: 1.189449 Accuracy: 63.40%
Train Epoch: 4 [6400/ 8390 (76%)] Loss: 1.243121 Accuracy: 62.38%
Epoch 4 – Time: 227.79s – Train Loss: 1.098549 – Train Accuracy: 62.16%
Test Loss: 1.480476 – Test Accuracy: 53.19%

Train Epoch: 5 [0/ 8390 (0%)] Loss: 0.945691 Accuracy: 68.75%
Train Epoch: 5 [3200/ 8390 (38%)] Loss: 1.595011 Accuracy: 65.38%
Train Epoch: 5 [6400/ 8390 (76%)] Loss: 1.228773 Accuracy: 65.13%
Epoch 5 – Time: 217.21s – Train Loss: 1.021704 – Train Accuracy: 65.40%
Test Loss: 1.419648 – Test Accuracy: 53.19%

Train Epoch: 6 [0/ 8390 (0%)] Loss: 1.023521 Accuracy: 65.62%
Train Epoch: 6 [3200/ 8390 (38%)] Loss: 0.868787 Accuracy: 68.22%
Train Epoch: 6 [6400/ 8390 (76%)] Loss: 1.304474 Accuracy: 67.35%
Epoch 6 – Time: 214.13s – Train Loss: 0.936772 – Train Accuracy: 67.21%
Test Loss: 1.766144 – Test Accuracy: 49.94%

Train Epoch: 7 [0/ 8390 (0%)] Loss: 1.172550 Accuracy: 68.75%
Train Epoch: 7 [3200/ 8390 (38%)] Loss: 0.928560 Accuracy: 69.12%
Train Epoch: 7 [6400/ 8390 (76%)] Loss: 1.150136 Accuracy: 68.58%
Epoch 7 – Time: 219.18s – Train Loss: 0.899686 – Train Accuracy: 68.67%
Test Loss: 1.508090 – Test Accuracy: 52.62%

Train Epoch: 8 [0/ 8390 (0%)] Loss: 0.720141 Accuracy: 84.38%
Train Epoch: 8 [3200/ 8390 (38%)] Loss: 0.978278 Accuracy: 72.87%
Train Epoch: 8 [6400/ 8390 (76%)] Loss: 1.115881 Accuracy: 72.17%
Epoch 8 – Time: 211.72s – Train Loss: 0.826719 – Train Accuracy: 71.47%
Test Loss: 1.482925 – Test Accuracy: 52.94%

Train Epoch: 9 [0/ 8390 (0%)] Loss: 0.771082 Accuracy: 68.75%
Train Epoch: 9 [3200/ 8390 (38%)] Loss: 0.770933 Accuracy: 74.91%
Train Epoch: 9 [6400/ 8390 (76%)] Loss: 0.638593 Accuracy: 74.52%
Epoch 9 – Time: 211.68s – Train Loss: 0.775972 – Train Accuracy: 73.81%
Test Loss: 2.327325 – Test Accuracy: 46.06%

Train Epoch: 10 [0/ 8390 (0%)] Loss: 0.718460 Accuracy: 81.25%
Train Epoch: 10 [3200/ 8390 (38%)] Loss: 0.490445 Accuracy: 76.45%
Train Epoch: 10 [6400/ 8390 (76%)] Loss: 0.763847 Accuracy: 74.47%
Epoch 10 – Time: 220.81s – Train Loss: 0.768677 – Train Accuracy: 73.90%
Test Loss: 1.624778 – Test Accuracy: 50.06%

Train Epoch: 11 [0/ 8390 (0%)] Loss: 0.837381 Accuracy: 71.88%
Train Epoch: 11 [3200/ 8390 (38%)] Loss: 0.955065 Accuracy: 76.76%
Train Epoch: 11 [6400/ 8390 (76%)] Loss: 0.871415 Accuracy: 76.32%
Epoch 11 – Time: 217.77s – Train Loss: 0.710534 – Train Accuracy: 75.29%
Test Loss: 1.927030 – Test Accuracy: 46.62%

Train Epoch: 12 [0/ 8390 (0%)] Loss: 0.437259 Accuracy: 84.38%
Train Epoch: 12 [3200/ 8390 (38%)] Loss: 0.542547 Accuracy: 75.96%
Train Epoch: 12 [6400/ 8390 (76%)] Loss: 0.763151 Accuracy: 76.43%
Epoch 12 – Time: 220.09s – Train Loss: 0.679895 – Train Accuracy: 76.85%
Test Loss: 1.866881 – Test Accuracy: 52.62%

Train Epoch: 13 [0/ 8390 (0%)] Loss: 0.471891 Accuracy: 78.12%
Train Epoch: 13 [3200/ 8390 (38%)] Loss: 0.483762 Accuracy: 79.39%
Train Epoch: 13 [6400/ 8390 (76%)] Loss: 0.462755 Accuracy: 79.00%
Epoch 13 – Time: 205.67s – Train Loss: 0.640420 – Train Accuracy: 78.55%
Test Loss: 1.900124 – Test Accuracy: 50.31%

Train Epoch: 14 [0/ 8390 (0%)] Loss: 0.535412 Accuracy: 78.12%
Train Epoch: 14 [3200/ 8390 (38%)] Loss: 1.018734 Accuracy: 80.38%
Train Epoch: 14 [6400/ 8390 (76%)] Loss: 0.481601 Accuracy: 79.80%
Epoch 14 – Time: 202.46s – Train Loss: 0.605839 – Train Accuracy: 79.49%
Test Loss: 1.946474 – Test Accuracy: 53.69%

Train Epoch: 15 [0/ 8390 (0%)] Loss: 0.608204 Accuracy: 75.00%
Train Epoch: 15 [3200/ 8390 (38%)] Loss: 0.225345 Accuracy: 79.27%
Train Epoch: 15 [6400/ 8390 (76%)] Loss: 0.510582 Accuracy: 79.23%
Epoch 15 – Time: 201.00s – Train Loss: 0.586624 – Train Accuracy: 79.30%
Test Loss: 2.127388 – Test Accuracy: 55.38%

Train Epoch: 16 [0/ 8390 (0%)] Loss: 0.638399 Accuracy: 65.62%
Train Epoch: 16 [3200/ 8390 (38%)] Loss: 0.393152 Accuracy: 82.05%
Train Epoch: 16 [6400/ 8390 (76%)] Loss: 0.281390 Accuracy: 81.00%
Epoch 16 – Time: 205.50s – Train Loss: 0.547327 – Train Accuracy: 81.12%
Test Loss: 2.033684 – Test Accuracy: 50.81%

Train Epoch: 17 [0/ 8390 (0%)] Loss: 0.874391 Accuracy: 68.75%
Train Epoch: 17 [3200/ 8390 (38%)] Loss: 0.404561 Accuracy: 82.77%
Train Epoch: 17 [6400/ 8390 (76%)] Loss: 0.610093 Accuracy: 82.11%
Epoch 17 – Time: 206.37s – Train Loss: 0.536022 – Train Accuracy: 81.75%
Test Loss: 1.759946 – Test Accuracy: 56.44%

Train Epoch: 18 [0/ 8390 (0%)] Loss: 0.479482 Accuracy: 87.50%
Train Epoch: 18 [3200/ 8390 (38%)] Loss: 0.659766 Accuracy: 84.65%
Train Epoch: 18 [6400/ 8390 (76%)] Loss: 0.344141 Accuracy: 84.05%
Epoch 18 – Time: 221.55s – Train Loss: 0.496238 – Train Accuracy: 83.54%
Test Loss: 2.168570 – Test Accuracy: 49.31%

Train Epoch: 19 [0/ 8390 (0%)] Loss: 0.513600 Accuracy: 75.00%
Train Epoch: 19 [3200/ 8390 (38%)] Loss: 0.243855 Accuracy: 83.66%
Train Epoch: 19 [6400/ 8390 (76%)] Loss: 0.496647 Accuracy: 83.19%
Epoch 19 – Time: 204.48s – Train Loss: 0.496304 – Train Accuracy: 82.96%
Test Loss: 1.894559 – Test Accuracy: 55.88%

FINAL RESULTS:

```
epoch_times: [207.22970986366272, 209.30913090705872, 208.01324605941772,
218.8806028366089, 227.79374074935913, 217.20957016944885, 214.13214111328
125, 219.17577195167542, 211.7231628894806, 211.67860794067383, 220.808238
9831543, 217.77472281455994, 220.09114789962769, 205.66677808761597, 202.4
6442699432373, 200.99517726898193, 205.49894618988037, 206.37325477600098,
221.5477020740509, 204.48290991783142]
train_losses: [1.700529125355582, 1.3946505498340112, 1.2765410285414631,
1.1626035025101582, 1.098549397619626, 1.0217041275428451, 0.9367715119178
058, 0.8996856087491713, 0.8267188028979847, 0.7759724028465402, 0.7686765
82297296, 0.7105342651369008, 0.6798951057077364, 0.6404201299634599, 0.60
58392998485165, 0.5866240029230373, 0.5473274279186744, 0.5360223032136,
0.4962378031083646, 0.4963039540550636]
train_accuracies: [38.963051251489865, 50.64362336114422, 55.5542312276519
7, 59.05840286054827, 62.157330154946365, 65.39928486293206, 67.2109654350
4171, 68.66507747318236, 71.46603098927294, 73.81406436233611, 73.89749702
026222, 75.29201430274136, 76.8533969010727, 78.54588796185935, 79.4874851
0131108, 79.29678188319429, 81.12038140643624, 81.75208581644816, 83.53992
84862932, 82.95589988081049]
test_losses: [1.8372494512796402, 1.5024336314201354, 1.5036351738870144,
1.5318452209234237, 1.4804760238528252, 1.4196479684114456, 1.766143504083
1566, 1.508089792728424, 1.482925257012248, 2.3273247413337232, 1.62477848
79803657, 1.9270301645994186, 1.866880871206522, 1.900124378465116, 1.9464
739935845137, 2.127388111501932, 2.033684199824929, 1.7599458007141948, 2.
168570462167263, 1.8945587299019098]
test_accuracies: [37.875, 50.5, 48.875, 48.0, 53.1875, 53.1875, 49.9375, 5
2.625, 52.9375, 46.0625, 50.0625, 46.625, 52.625, 50.3125, 53.6875, 55.37
5, 50.8125, 56.4375, 49.3125, 55.875]
```

```
In [11]: train_dataset = GenreDatasetPhaseMFCC("Data/train/", n_fft=2048, hop_leng
test_dataset = GenreDatasetPhaseMFCC("Data/test/", n_fft=2048, hop_length
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, shuffle
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, shuffle=F
```

```
class ComplexNet(nn.Module):
```

```
    def __init__(self):
        super(ComplexNet, self).__init__()
        self.conv1 = ComplexConv2d(1, 10, 2, 1)
        self.bn = ComplexBatchNorm2d(10)
        self.conv2 = ComplexConv2d(10, 20, 2, 1)
        self.fc1 = ComplexLinear(30*2*20, 500)
        self.fc2 = ComplexLinear(500, 10)
```

```
    def forward(self, x):
        x = self.conv1(x)
        x = complex_relu(x)
        x = complex_max_pool2d(x, 2, 2)
        x = self.bn(x)
        x = self.conv2(x)
        x = complex_relu(x)
        x = complex_max_pool2d(x, 2, 2)
        x = x.view(-1, 30*2*20)
        x = self.fc1(x)
```

```
x = complex_relu(x)
x = self.fc2(x)
x = x.abs()
x = F.log_softmax(x, dim=1)
return x

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = ComplexNet().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

metrics_dict_e3 = {
    'epoch_times': [],
    'train_losses': [],
    'train_accuracies': [],
    'test_losses': [],
    'test_accuracies': []
}

for epoch in range(NUM_EPOCHS):
    train(model,
          device,
          train_loader,
          test_loader,
          optimizer,
          epoch,
          metrics_dict_e2)

print("-"*100)
print("-"*100)
print("FINAL RESULTS:")
print("-"*100)
for key, value in metrics_dict_e3.items():
    print(f'{key}: {value}')
```


Train Epoch: 0 [0/ 8390 (0%)] Loss: 2.413850 Accuracy: 15.62%
Train Epoch: 0 [3200/ 8390 (38%)] Loss: 2.207769 Accuracy: 19.09%
Train Epoch: 0 [6400/ 8390 (76%)] Loss: 1.834011 Accuracy: 23.13%
Epoch 0 – Time: 223.27s – Train Loss: 2.042803 – Train Accuracy: 25.22%
Test Loss: 1.984808 – Test Accuracy: 29.25%

Train Epoch: 1 [0/ 8390 (0%)] Loss: 2.062810 Accuracy: 25.00%
Train Epoch: 1 [3200/ 8390 (38%)] Loss: 1.839269 Accuracy: 32.21%
Train Epoch: 1 [6400/ 8390 (76%)] Loss: 1.721043 Accuracy: 32.73%
Epoch 1 – Time: 337.22s – Train Loss: 1.861055 – Train Accuracy: 32.40%
Test Loss: 1.878305 – Test Accuracy: 31.12%

Train Epoch: 2 [0/ 8390 (0%)] Loss: 1.635835 Accuracy: 40.62%
Train Epoch: 2 [3200/ 8390 (38%)] Loss: 1.928107 Accuracy: 34.10%
Train Epoch: 2 [6400/ 8390 (76%)] Loss: 2.216387 Accuracy: 34.76%
Epoch 2 – Time: 286.50s – Train Loss: 1.809655 – Train Accuracy: 34.35%
Test Loss: 1.889286 – Test Accuracy: 30.88%

Train Epoch: 3 [0/ 8390 (0%)] Loss: 1.787891 Accuracy: 37.50%
Train Epoch: 3 [3200/ 8390 (38%)] Loss: 1.962163 Accuracy: 37.10%
Train Epoch: 3 [6400/ 8390 (76%)] Loss: 1.804685 Accuracy: 36.35%
Epoch 3 – Time: 319.34s – Train Loss: 1.759352 – Train Accuracy: 36.33%
Test Loss: 1.856480 – Test Accuracy: 35.00%

Train Epoch: 4 [0/ 8390 (0%)] Loss: 1.647825 Accuracy: 40.62%
Train Epoch: 4 [3200/ 8390 (38%)] Loss: 1.899797 Accuracy: 38.24%
Train Epoch: 4 [6400/ 8390 (76%)] Loss: 1.599051 Accuracy: 37.06%
Epoch 4 – Time: 298.26s – Train Loss: 1.723030 – Train Accuracy: 37.53%
Test Loss: 1.867167 – Test Accuracy: 31.50%

Train Epoch: 5 [0/ 8390 (0%)] Loss: 1.510139 Accuracy: 50.00%
Train Epoch: 5 [3200/ 8390 (38%)] Loss: 1.432850 Accuracy: 38.06%
Train Epoch: 5 [6400/ 8390 (76%)] Loss: 1.950091 Accuracy: 38.56%
Epoch 5 – Time: 378.20s – Train Loss: 1.717170 – Train Accuracy: 38.20%
Test Loss: 1.821705 – Test Accuracy: 34.25%

Train Epoch: 6 [0/ 8390 (0%)] Loss: 1.339843 Accuracy: 53.12%
Train Epoch: 6 [3200/ 8390 (38%)] Loss: 1.512588 Accuracy: 40.01%
Train Epoch: 6 [6400/ 8390 (76%)] Loss: 1.511724 Accuracy: 39.80%
Epoch 6 – Time: 327.84s – Train Loss: 1.674332 – Train Accuracy: 39.89%
Test Loss: 1.803232 – Test Accuracy: 34.06%

Train Epoch: 7 [0/ 8390 (0%)] Loss: 1.740120 Accuracy: 40.62%
Train Epoch: 7 [3200/ 8390 (38%)] Loss: 1.243231 Accuracy: 40.78%
Train Epoch: 7 [6400/ 8390 (76%)] Loss: 1.950737 Accuracy: 40.80%
Epoch 7 – Time: 353.24s – Train Loss: 1.654439 – Train Accuracy: 40.87%
Test Loss: 1.895607 – Test Accuracy: 32.88%

Train Epoch: 8 [0/ 8390 (0%)] Loss: 1.482600 Accuracy: 53.12%
Train Epoch: 8 [3200/ 8390 (38%)] Loss: 1.587255 Accuracy: 42.85%
Train Epoch: 8 [6400/ 8390 (76%)] Loss: 1.714218 Accuracy: 42.54%
Epoch 8 – Time: 297.25s – Train Loss: 1.620706 – Train Accuracy: 42.05%
Test Loss: 1.815371 – Test Accuracy: 36.19%

Train Epoch: 9 [0/ 8390 (0%)] Loss: 1.766228 Accuracy: 34.38%
Train Epoch: 9 [3200/ 8390 (38%)] Loss: 1.787615 Accuracy: 43.38%
Train Epoch: 9 [6400/ 8390 (76%)] Loss: 1.693290 Accuracy: 42.96%
Epoch 9 – Time: 287.32s – Train Loss: 1.599723 – Train Accuracy: 42.90%
Test Loss: 1.834403 – Test Accuracy: 34.81%

Train Epoch: 10 [0/ 8390 (0%)] Loss: 1.723920 Accuracy: 40.62%
Train Epoch: 10 [3200/ 8390 (38%)] Loss: 1.512322 Accuracy: 43.47%
Train Epoch: 10 [6400/ 8390 (76%)] Loss: 1.607760 Accuracy: 43.05%
Epoch 10 - Time: 1177.41s - Train Loss: 1.586791 - Train Accuracy: 43.02%
Test Loss: 1.872561 - Test Accuracy: 31.62%

Train Epoch: 11 [0/ 8390 (0%)] Loss: 1.793006 Accuracy: 34.38%
Train Epoch: 11 [3200/ 8390 (38%)] Loss: 1.809574 Accuracy: 45.85%
Train Epoch: 11 [6400/ 8390 (76%)] Loss: 1.698219 Accuracy: 44.29%
Epoch 11 - Time: 8451.60s - Train Loss: 1.559433 - Train Accuracy: 44.22%
Test Loss: 1.905067 - Test Accuracy: 34.38%

Train Epoch: 12 [0/ 8390 (0%)] Loss: 1.667610 Accuracy: 37.50%
Train Epoch: 12 [3200/ 8390 (38%)] Loss: 1.469992 Accuracy: 45.11%
Train Epoch: 12 [6400/ 8390 (76%)] Loss: 1.860880 Accuracy: 44.92%
Epoch 12 - Time: 210.94s - Train Loss: 1.559958 - Train Accuracy: 44.17%
Test Loss: 1.800275 - Test Accuracy: 34.94%

Train Epoch: 13 [0/ 8390 (0%)] Loss: 1.479374 Accuracy: 37.50%
Train Epoch: 13 [3200/ 8390 (38%)] Loss: 1.582445 Accuracy: 44.71%
Train Epoch: 13 [6400/ 8390 (76%)] Loss: 1.403867 Accuracy: 45.20%
Epoch 13 - Time: 217.98s - Train Loss: 1.539376 - Train Accuracy: 45.44%
Test Loss: 1.738834 - Test Accuracy: 38.12%

Train Epoch: 14 [0/ 8390 (0%)] Loss: 1.457347 Accuracy: 43.75%
Train Epoch: 14 [3200/ 8390 (38%)] Loss: 1.363334 Accuracy: 48.79%
Train Epoch: 14 [6400/ 8390 (76%)] Loss: 1.780651 Accuracy: 47.59%
Epoch 14 - Time: 245.03s - Train Loss: 1.514394 - Train Accuracy: 46.85%
Test Loss: 1.863470 - Test Accuracy: 35.62%

Train Epoch: 15 [0/ 8390 (0%)] Loss: 1.374776 Accuracy: 43.75%
Train Epoch: 15 [3200/ 8390 (38%)] Loss: 1.330341 Accuracy: 46.91%
Train Epoch: 15 [6400/ 8390 (76%)] Loss: 1.671826 Accuracy: 46.72%
Epoch 15 - Time: 301.61s - Train Loss: 1.497057 - Train Accuracy: 46.82%
Test Loss: 1.824527 - Test Accuracy: 36.12%

Train Epoch: 16 [0/ 8390 (0%)] Loss: 1.425750 Accuracy: 46.88%
Train Epoch: 16 [3200/ 8390 (38%)] Loss: 1.448989 Accuracy: 48.45%
Train Epoch: 16 [6400/ 8390 (76%)] Loss: 1.449083 Accuracy: 47.43%
Epoch 16 - Time: 299.60s - Train Loss: 1.486795 - Train Accuracy: 47.54%
Test Loss: 1.906221 - Test Accuracy: 34.56%

Train Epoch: 17 [0/ 8390 (0%)] Loss: 1.887340 Accuracy: 31.25%
Train Epoch: 17 [3200/ 8390 (38%)] Loss: 1.329155 Accuracy: 49.69%
Train Epoch: 17 [6400/ 8390 (76%)] Loss: 1.594913 Accuracy: 49.00%
Epoch 17 - Time: 300.35s - Train Loss: 1.454458 - Train Accuracy: 48.92%
Test Loss: 1.837543 - Test Accuracy: 37.62%

Train Epoch: 18 [0/ 8390 (0%)] Loss: 1.328355 Accuracy: 53.12%
Train Epoch: 18 [3200/ 8390 (38%)] Loss: 1.780887 Accuracy: 49.01%
Train Epoch: 18 [6400/ 8390 (76%)] Loss: 1.542195 Accuracy: 48.49%
Epoch 18 - Time: 295.70s - Train Loss: 1.448089 - Train Accuracy: 48.90%
Test Loss: 1.878924 - Test Accuracy: 34.25%

Train Epoch: 19 [0/ 8390 (0%)] Loss: 1.354482 Accuracy: 59.38%
Train Epoch: 19 [3200/ 8390 (38%)] Loss: 1.856767 Accuracy: 50.19%
Train Epoch: 19 [6400/ 8390 (76%)] Loss: 1.144183 Accuracy: 49.91%
Epoch 19 - Time: 248.67s - Train Loss: 1.423151 - Train Accuracy: 49.58%
Test Loss: 1.881986 - Test Accuracy: 35.38%

FINAL RESULTS:

```
epoch_times: [223.27179288864136, 337.2176299095154, 286.5028817653656, 31
9.344908952713, 298.26079392433167, 378.1967351436615, 327.8363778591156,
353.2444438934326, 297.2461521625519, 287.32417821884155, 1177.40505790710
45, 8451.597054958344, 210.93699979782104, 217.97601509094238, 245.0278248
7869263, 301.6087737083435, 299.5960121154785, 300.3493719100952, 295.6972
7897644043, 248.66599893569946]
train_losses: [2.042802960363053, 1.8610548781984635, 1.8096553337483006,
1.7593516279722898, 1.7230299820426767, 1.7171696833981813, 1.674332307040
2363, 1.6544389388033451, 1.6207061751198222, 1.5997228954584544, 1.586791
0953878446, 1.55943332191642, 1.5599576771714305, 1.539376305259821, 1.514
3942514448676, 1.497057060931475, 1.4867947952437948, 1.4544576249504817,
1.4480891391521191, 1.4231507835042385]
train_accuracies: [25.220500595947556, 32.39570917759237, 34.3504171632896
3, 36.32896305125149, 37.532777115613825, 38.20023837902264, 39.8927294398
09294, 40.87008343265793, 42.05005959475566, 42.896305125148984, 43.015494
63647199, 44.219308700834326, 44.17163289630513, 45.43504171632896, 46.853
396901072706, 46.8176400476758, 47.54469606674613, 48.91537544696067, 48.9
03456495828365, 49.582836710369484]
test_losses: [1.9848083889484405, 1.8783054047822951, 1.8892862010002136,
1.8564798241853715, 1.867166874408722, 1.821704980134964, 1.80323201656341
55, 1.895607076883316, 1.8153707492351532, 1.834402973651886, 1.8725606411
695481, 1.905067165493965, 1.8002750039100648, 1.7388343811035156, 1.86347
01204299926, 1.824527004957199, 1.9062212073802949, 1.837543227672577, 1.8
789236879348754, 1.8819858026504517]
test_accuracies: [29.25, 31.125, 30.875, 35.0, 31.5, 34.25, 34.0625, 32.87
5, 36.1875, 34.8125, 31.625, 34.375, 34.9375, 38.125, 35.625, 36.125, 34.5
625, 37.625, 34.25, 35.375]
```

Plots

```
In [22]: # Data for the four scenarios
data = {
    "Magnitude Only (Real Net)": metrics_dict_e1,
    "Magnitude Only (Complex Net)": metrics_dict_e2,
    "Magnitude and Phase (Complex Net)": metrics_dict_e3
}

# Data for plotting
epochs = range(1, 21)
colors = ['b', 'g', 'r', 'm', 'y']
scenarios = list(data.keys())

fig, axes = plt.subplots(2, 1, figsize=(10, 10))

for i, scenario in enumerate(scenarios):
    axes[0].plot(epochs, data[scenario]["train_accuracies"], label=scenario)

axes[0].set_title("Train Accuracy")
axes[0].set_xlabel("Epochs")
axes[0].set_ylabel("Train Accuracy")
axes[0].legend()
```

```
for i, scenario in enumerate(scenarios):
    axes[1].plot(epochs, data[scenario]["test_accuracies"], label=scenario)

axes[1].set_title("Test Accuracy")
axes[1].set_xlabel("Epochs")
axes[1].set_ylabel("Test Accuracy")
axes[1].legend()

plt.tight_layout()
plt.show()

fig, axes = plt.subplots(2, 1, figsize=(10, 10))

for i, scenario in enumerate(scenarios):
    axes[0].plot(epochs, data[scenario]["train_losses"], label=scenario, color=i)

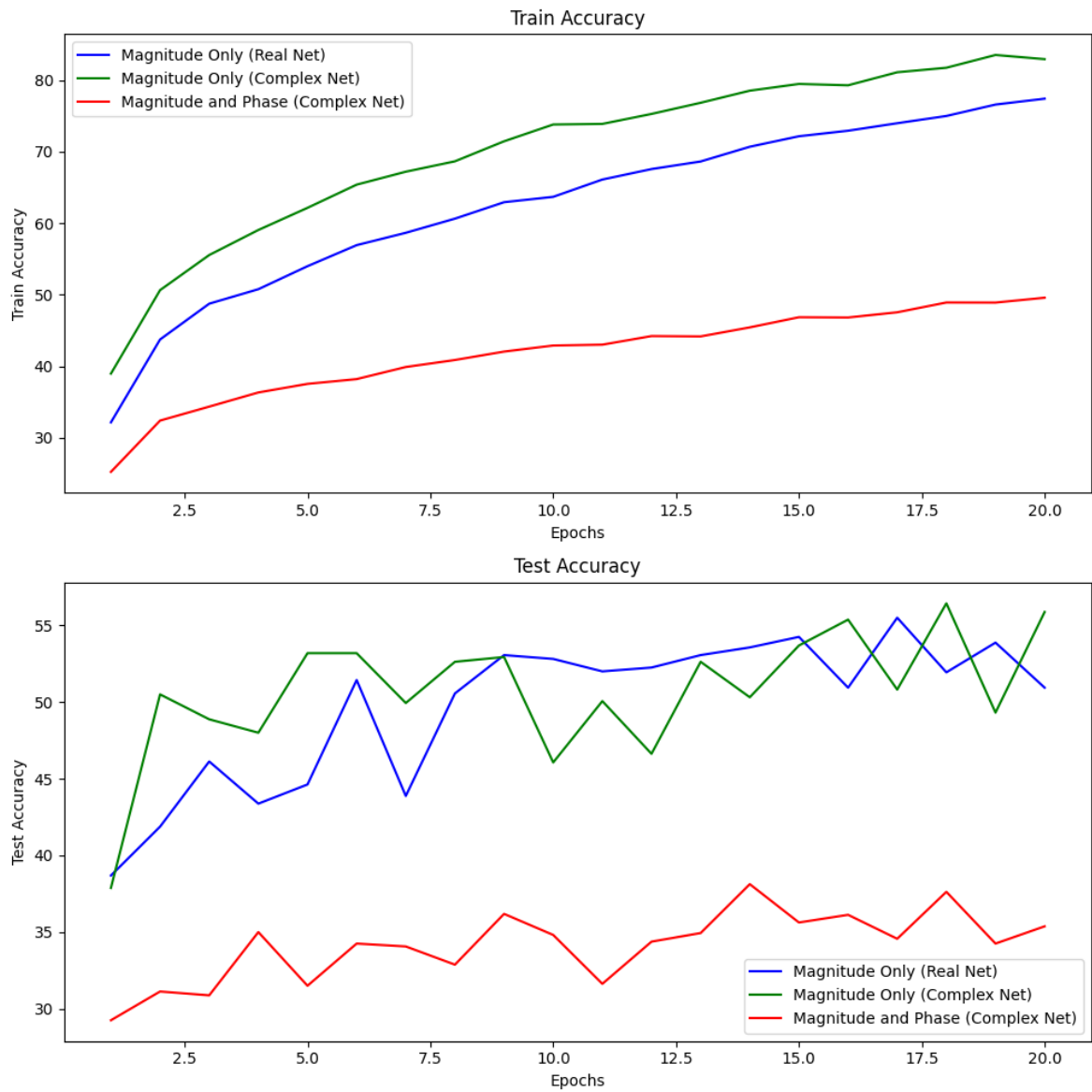
axes[0].set_title("Train Loss")
axes[0].set_xlabel("Epochs")
axes[0].set_ylabel("Train Loss")
axes[0].legend()

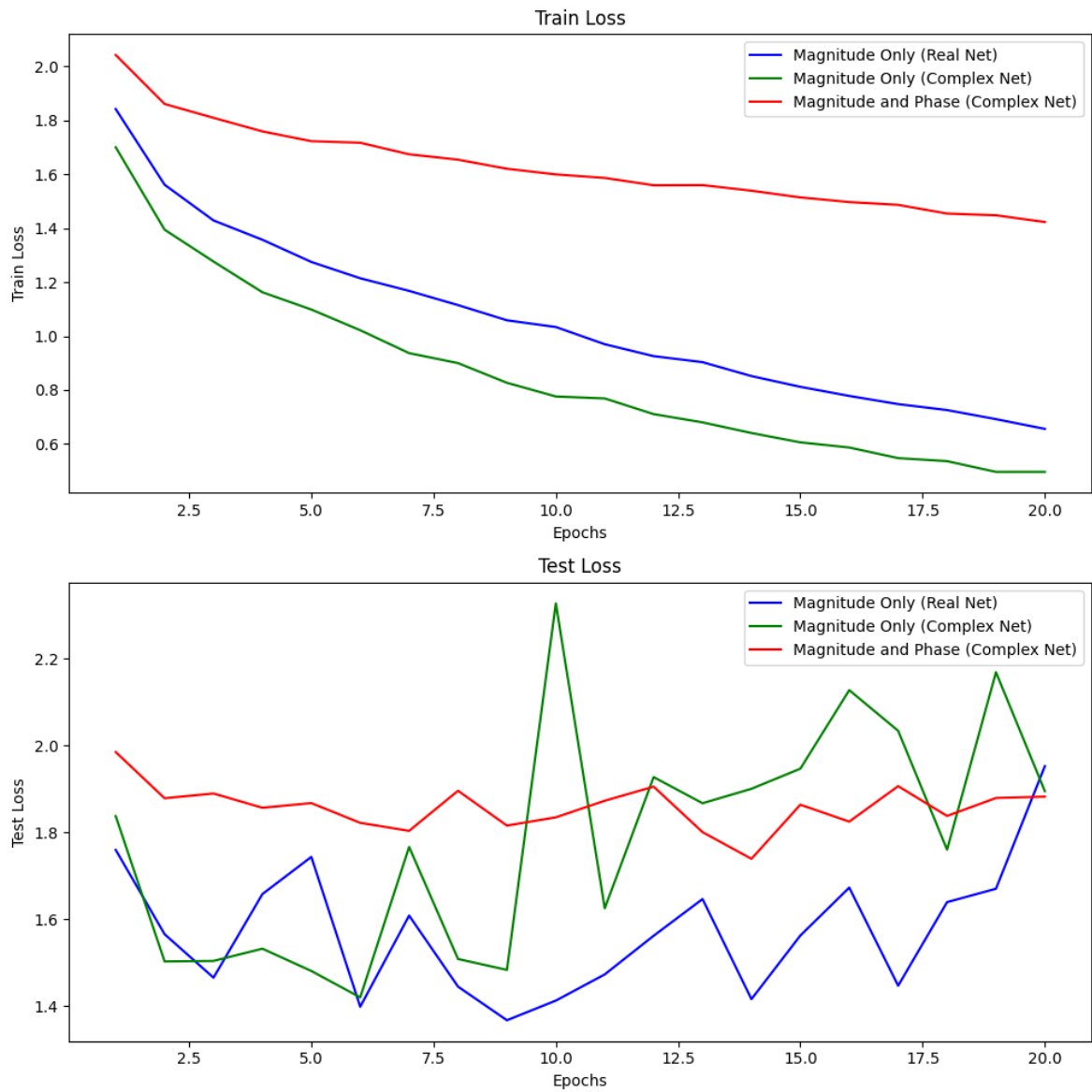
for i, scenario in enumerate(scenarios):
    axes[1].plot(epochs, data[scenario]["test_losses"], label=scenario, color=i)

axes[1].set_title("Test Loss")
axes[1].set_xlabel("Epochs")
axes[1].set_ylabel("Test Loss")
axes[1].legend()

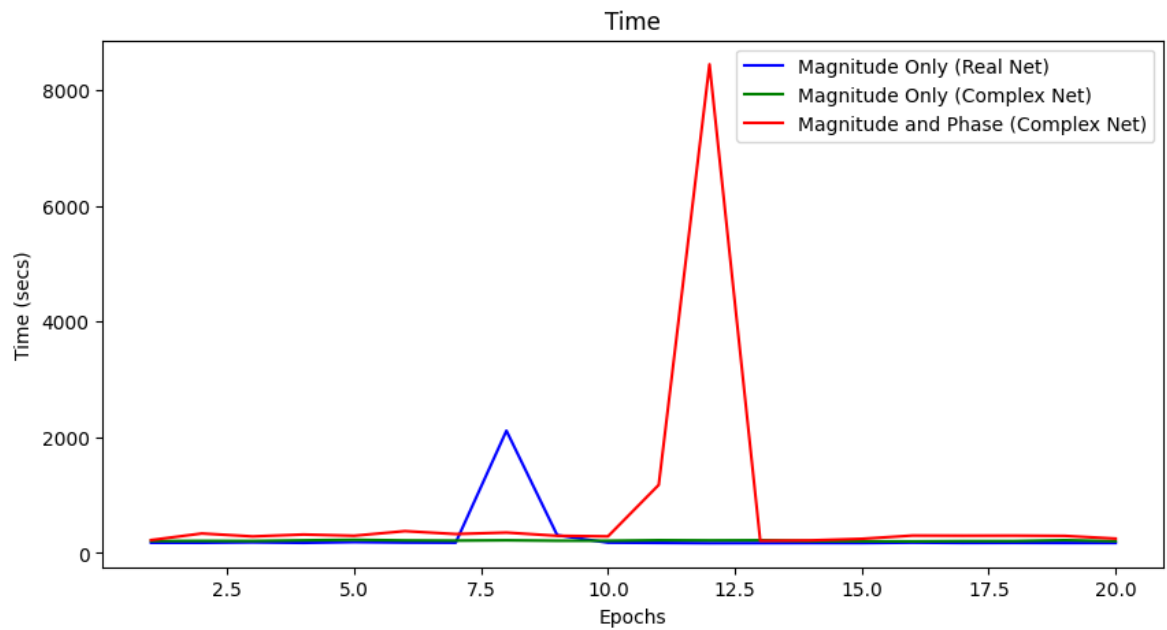
plt.tight_layout()
plt.show()

fig, axes = plt.subplots(1, 1, figsize=(10, 5))
for i, scenario in enumerate(scenarios):
    axes.plot(epochs, data[scenario]["epoch_times"], label=scenario, color=i)
axes.set_title("Time")
axes.set_xlabel("Epochs")
axes.set_ylabel("Time (secs)")
axes.legend()
```





Out[22]: <matplotlib.legend.Legend at 0x2b9e9f790>



New tests