Complex PyTorch for Music Genre Classification

In [17]:
```
# 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 [18]:
```
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 % 10 == 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 [19]:

```
DATASET_PATH = "Data/binary_data/train"
SAMPLE_RATE = 22050
TRACK_DURATION = 30 # measured in seconds
SAMPLES_PER_TRACK = SAMPLE_RATE * TRACK_DURATION
BATCH_SIZE = 32
NUM_EPOCHS = 10
```

In [20]:

```
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)
```

```
{'classical': 0, 'rock': 1}
```

MFCCS

In [21]:
```
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
        return frames

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

    def met_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 +
        freqs = self.met_to_freq(mels)
        return np.floor((self.FFT_size + 1) / sample_rate * freqs).astype

    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.linsp
            filters[n, filter_points[n + 1]: filter_points[n + 2]] = np.l
        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
        for i in range(1, self.dct_filter_num):
            basis[i, :] = np.cos(i * samples) * np.sqrt(2.0 / self.mel_fi
        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.sha
        for n in range(audio_fft.shape[1]):
            audio_fft[:, n] = fft.fft(audio_winT[:, n], axis=0)[:audio_ff
```

```
        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_
        filters = self.get_filters(filter_points)
        audio_filtered = np.dot(filters, np.transpose(audio_fft))
        audio_filtered = np.maximum(audio_filtered, self.epsilon)  # Repl
        audio_log = 10.0 * np.log10(audio_filtered)
        dct_filters = self.dct()
        cepstral_coefficents = np.dot(dct_filters, audio_log)
        return np.array([cepstral_coefficents])

class MusicFeatureExtractorComplex:
    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
        return frames

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

    def met_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 +
        freqs = self.met_to_freq(mels)
        return np.floor((self.FFT_size + 1) / sample_rate * freqs).astype

    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.linsp
            filters[n, filter_points[n + 1]: filter_points[n + 2]] = np.l
        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
        for i in range(1, self.dct_filter_num):
            basis[i, :] = np.cos(i * samples) * np.sqrt(2.0 / self.mel_fi
        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.sha
        for n in range(audio_fft.shape[1]):
            audio_fft[:, n] = fft.fft(audio_winT[:, n], axis=0)[:audio_ff
        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_
        filters = self.get_filters(filter_points)
        audio_filtered = np.dot(filters, np.transpose(audio_fft))
        audio_filtered[audio_filtered == 0] = self.epsilon # Replace zerc
        audio_log = 10.0 * np.log10(audio_filtered)
        dct_filters = self.dct()
        cepstral_coefficents = np.dot(dct_filters, audio_log)
        return np.array([cepstral_coefficents])
```

In [22]:
```
class GenreDatasetMFCC(Dataset):

    def __init__(self, train_path, n_fft=2048, hop_length=512, num_segmen
        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_fil
        self.dct_filter_num = dct_filter_num
        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.uni
        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_leng
            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("/")[3]]
        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_signa
        cur_mfcc = self.mfcc_extractor.get_mfcc_features(cur_signal, samp
        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_segmen
        super().__init__(train_path, n_fft, hop_length, num_segments, mel
        self.mfcc_extractor = MusicFeatureExtractorComplex(
            FFT_size=n_fft, HOP_SIZE=hop_length, mel_filter_num = mel_fil

    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("/")[3]]
        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_signa
        cur_mfcc = self.mfcc_extractor.get_mfcc_features(cur_signal, samp
        cur_mfcc = self.adjust_shape(cur_mfcc)
        return torch.tensor(cur_mfcc, dtype=torch.complex64), target
```

1. No phase data

In [30]:
```
train_dataset = GenreDatasetMFCC("Data/binary_data/train/", n_fft=2048, h
test_dataset = GenreDatasetMFCC("Data/binary_data/test/", n_fft=2048, hop
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, shuffle
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, shuffle=F
```

In [31]:
```
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, 2)

    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/  1680 (  0%)] Loss: 0.715544  Accuracy: 34.38%
Train Epoch:   0 [   320/  1680 ( 19%)] Loss: 0.425783  Accuracy: 57.95%
Train Epoch:   0 [   640/  1680 ( 38%)] Loss: 0.214916  Accuracy: 72.17%
Train Epoch:   0 [   960/  1680 ( 58%)] Loss: 0.330717  Accuracy: 77.92%
Train Epoch:   0 [  1280/  1680 ( 77%)] Loss: 0.253875  Accuracy: 80.72%
Train Epoch:   0 [  1600/  1680 ( 96%)] Loss: 0.151264  Accuracy: 82.60%
Epoch 0 - Time: 37.52s - Train Loss: 0.361252 - Train Accuracy: 81.90%
Test Loss: 0.184620 - Test Accuracy: 95.00%

Train Epoch:   1 [     0/  1680 (  0%)] Loss: 0.116859  Accuracy: 93.75%
Train Epoch:   1 [   320/  1680 ( 19%)] Loss: 0.181167  Accuracy: 91.48%
Train Epoch:   1 [   640/  1680 ( 38%)] Loss: 0.205668  Accuracy: 91.52%
Train Epoch:   1 [   960/  1680 ( 58%)] Loss: 0.297967  Accuracy: 91.94%
Train Epoch:   1 [  1280/  1680 ( 77%)] Loss: 0.142673  Accuracy: 91.92%
Train Epoch:   1 [  1600/  1680 ( 96%)] Loss: 0.586391  Accuracy: 91.42%
Epoch 1 - Time: 51.89s - Train Loss: 0.233470 - Train Accuracy: 90.65%
Test Loss: 0.176717 - Test Accuracy: 96.25%

Train Epoch:   2 [     0/  1680 (  0%)] Loss: 0.191501  Accuracy: 93.75%
Train Epoch:   2 [   320/  1680 ( 19%)] Loss: 0.473941  Accuracy: 92.61%
Train Epoch:   2 [   640/  1680 ( 38%)] Loss: 0.150896  Accuracy: 91.52%
Train Epoch:   2 [   960/  1680 ( 58%)] Loss: 0.240908  Accuracy: 91.43%
Train Epoch:   2 [  1280/  1680 ( 77%)] Loss: 0.205249  Accuracy: 91.16%
Train Epoch:   2 [  1600/  1680 ( 96%)] Loss: 0.145471  Accuracy: 91.24%
Epoch 2 - Time: 39.21s - Train Loss: 0.231075 - Train Accuracy: 90.42%
Test Loss: 0.192560 - Test Accuracy: 96.25%

Train Epoch:   3 [     0/  1680 (  0%)] Loss: 0.165502  Accuracy: 96.88%
Train Epoch:   3 [   320/  1680 ( 19%)] Loss: 0.410278  Accuracy: 91.48%
Train Epoch:   3 [   640/  1680 ( 38%)] Loss: 0.271823  Accuracy: 92.41%
Train Epoch:   3 [   960/  1680 ( 58%)] Loss: 0.213464  Accuracy: 93.45%
Train Epoch:   3 [  1280/  1680 ( 77%)] Loss: 0.102448  Accuracy: 92.76%
Train Epoch:   3 [  1600/  1680 ( 96%)] Loss: 0.182957  Accuracy: 91.79%
Epoch 3 - Time: 41.20s - Train Loss: 0.215985 - Train Accuracy: 90.65%
Test Loss: 0.199063 - Test Accuracy: 94.38%

Train Epoch:   4 [     0/  1680 (  0%)] Loss: 0.283567  Accuracy: 90.62%
Train Epoch:   4 [   320/  1680 ( 19%)] Loss: 0.214640  Accuracy: 92.05%
Train Epoch:   4 [   640/  1680 ( 38%)] Loss: 0.326049  Accuracy: 93.01%
Train Epoch:   4 [   960/  1680 ( 58%)] Loss: 0.086493  Accuracy: 93.55%
Train Epoch:   4 [  1280/  1680 ( 77%)] Loss: 0.070953  Accuracy: 93.52%
Train Epoch:   4 [  1600/  1680 ( 96%)] Loss: 0.176417  Accuracy: 93.44%
Epoch 4 - Time: 38.53s - Train Loss: 0.190518 - Train Accuracy: 92.68%
Test Loss: 0.184343 - Test Accuracy: 95.31%

Train Epoch:   5 [     0/  1680 (  0%)] Loss: 0.078587  Accuracy: 96.88%
Train Epoch:   5 [   320/  1680 ( 19%)] Loss: 0.150306  Accuracy: 92.33%
Train Epoch:   5 [   640/  1680 ( 38%)] Loss: 0.165681  Accuracy: 91.82%
Train Epoch:   5 [   960/  1680 ( 58%)] Loss: 0.112581  Accuracy: 92.84%
Train Epoch:   5 [  1280/  1680 ( 77%)] Loss: 0.181829  Accuracy: 92.68%
Train Epoch:   5 [  1600/  1680 ( 96%)] Loss: 0.231670  Accuracy: 93.01%
Epoch 5 - Time: 40.21s - Train Loss: 0.179768 - Train Accuracy: 92.08%
Test Loss: 0.237013 - Test Accuracy: 95.00%

Train Epoch:   6 [     0/  1680 (  0%)] Loss: 0.332018  Accuracy: 81.25%
Train Epoch:   6 [   320/  1680 ( 19%)] Loss: 0.118397  Accuracy: 91.19%
Train Epoch:   6 [   640/  1680 ( 38%)] Loss: 0.219945  Accuracy: 92.56%
Train Epoch:   6 [   960/  1680 ( 58%)] Loss: 0.310991  Accuracy: 92.84%
Train Epoch:   6 [  1280/  1680 ( 77%)] Loss: 0.037779  Accuracy: 93.67%
Train Epoch:   6 [  1600/  1680 ( 96%)] Loss: 0.269485  Accuracy: 93.63%
```

```
Epoch 6 - Time: 52.36s - Train Loss: 0.172697 - Train Accuracy: 92.74%
Test Loss: 0.295163 - Test Accuracy: 91.88%

Train Epoch:   7 [     0/  1680 (  0%)] Loss: 0.177706  Accuracy: 90.62%
Train Epoch:   7 [   320/  1680 ( 19%)] Loss: 0.142058  Accuracy: 91.76%
Train Epoch:   7 [   640/  1680 ( 38%)] Loss: 0.085112  Accuracy: 93.60%
Train Epoch:   7 [   960/  1680 ( 58%)] Loss: 0.366243  Accuracy: 94.25%
Train Epoch:   7 [  1280/  1680 ( 77%)] Loss: 0.244307  Accuracy: 93.90%
Train Epoch:   7 [  1600/  1680 ( 96%)] Loss: 0.198373  Accuracy: 93.38%
Epoch 7 - Time: 36.99s - Train Loss: 0.185385 - Train Accuracy: 92.44%
Test Loss: 0.174413 - Test Accuracy: 96.88%

Train Epoch:   8 [     0/  1680 (  0%)] Loss: 0.159421  Accuracy: 96.88%
Train Epoch:   8 [   320/  1680 ( 19%)] Loss: 0.228357  Accuracy: 91.76%
Train Epoch:   8 [   640/  1680 ( 38%)] Loss: 0.124323  Accuracy: 93.30%
Train Epoch:   8 [   960/  1680 ( 58%)] Loss: 0.052267  Accuracy: 93.25%
Train Epoch:   8 [  1280/  1680 ( 77%)] Loss: 0.523930  Accuracy: 92.99%
Train Epoch:   8 [  1600/  1680 ( 96%)] Loss: 0.190568  Accuracy: 93.44%
Epoch 8 - Time: 35.39s - Train Loss: 0.179532 - Train Accuracy: 92.50%
Test Loss: 0.201300 - Test Accuracy: 92.50%

Train Epoch:   9 [     0/  1680 (  0%)] Loss: 0.179819  Accuracy: 93.75%
Train Epoch:   9 [   320/  1680 ( 19%)] Loss: 0.231372  Accuracy: 94.03%
Train Epoch:   9 [   640/  1680 ( 38%)] Loss: 0.117419  Accuracy: 95.09%
Train Epoch:   9 [   960/  1680 ( 58%)] Loss: 0.182112  Accuracy: 95.26%
Train Epoch:   9 [  1280/  1680 ( 77%)] Loss: 0.084204  Accuracy: 94.97%
Train Epoch:   9 [  1600/  1680 ( 96%)] Loss: 0.281102  Accuracy: 94.49%
Epoch 9 - Time: 36.27s - Train Loss: 0.138495 - Train Accuracy: 93.69%
Test Loss: 0.196841 - Test Accuracy: 96.56%

--------------------------------------------------------------------------
--------------------------
--------------------------------------------------------------------------
--------------------------
FINAL RESULTS:
--------------------------------------------------------------------------
--------------------------
epoch_times: [37.51734495162964, 51.892842054367065, 39.20804286003113, 4
1.200674057006836, 38.53182005882263, 40.20562291145325, 52.3609809875488
3, 36.98588514328003, 35.39342212677002, 36.267234086990356]
train_losses: [0.3612523521654881, 0.23346950615254733, 0.2310745389415667
6, 0.2159847513271066, 0.19051814960459104, 0.17976838558052594, 0.1726974
0708602163, 0.18538537392249474, 0.17953154215445885, 0.1384946137953263]
train_accuracies: [81.9047619047619, 90.6547619047619, 90.41666666666667,
90.6547619047619, 92.67857142857143, 92.08333333333333, 92.73809523809524,
92.44047619047619, 92.5, 93.69047619047619]
test_losses: [0.18461954668164254, 0.1767165631055832, 0.1925600398331880
6, 0.19906260073184967, 0.18434346728026868, 0.23701292220503092, 0.295162
65785787255, 0.17441322961822153, 0.20130045646801592, 0.1968405670020729
2]
test_accuracies: [95.0, 96.25, 96.25, 94.375, 95.3125, 95.0, 91.875, 96.87
5, 92.5, 96.5625]
```

In [32]:

```python
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, 2)

    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/  1680 (  0%)] Loss: 0.679613  Accuracy: 62.50%
Train Epoch:   0 [   320/  1680 ( 19%)] Loss: 0.446472  Accuracy: 73.58%
Train Epoch:   0 [   640/  1680 ( 38%)] Loss: 0.108056  Accuracy: 80.80%
Train Epoch:   0 [   960/  1680 ( 58%)] Loss: 0.184048  Accuracy: 84.48%
Train Epoch:   0 [  1280/  1680 ( 77%)] Loss: 0.135176  Accuracy: 87.04%
Train Epoch:   0 [  1600/  1680 ( 96%)] Loss: 0.307315  Accuracy: 87.19%
Epoch 0 - Time: 42.76s - Train Loss: 0.312190 - Train Accuracy: 86.43%
Test Loss: 0.181000 - Test Accuracy: 94.69%

Train Epoch:   1 [     0/  1680 (  0%)] Loss: 0.220294  Accuracy: 93.75%
Train Epoch:   1 [   320/  1680 ( 19%)] Loss: 0.210621  Accuracy: 93.75%
Train Epoch:   1 [   640/  1680 ( 38%)] Loss: 0.052198  Accuracy: 93.90%
Train Epoch:   1 [   960/  1680 ( 58%)] Loss: 0.055498  Accuracy: 93.95%
Train Epoch:   1 [  1280/  1680 ( 77%)] Loss: 0.133853  Accuracy: 93.67%
Train Epoch:   1 [  1600/  1680 ( 96%)] Loss: 0.059262  Accuracy: 93.93%
Epoch 1 - Time: 43.32s - Train Loss: 0.173853 - Train Accuracy: 93.04%
Test Loss: 0.183872 - Test Accuracy: 95.31%

Train Epoch:   2 [     0/  1680 (  0%)] Loss: 0.021537  Accuracy: 100.00%
Train Epoch:   2 [   320/  1680 ( 19%)] Loss: 0.261391  Accuracy: 94.32%
Train Epoch:   2 [   640/  1680 ( 38%)] Loss: 0.105262  Accuracy: 93.75%
Train Epoch:   2 [   960/  1680 ( 58%)] Loss: 0.164997  Accuracy: 93.95%
Train Epoch:   2 [  1280/  1680 ( 77%)] Loss: 0.156987  Accuracy: 93.67%
Train Epoch:   2 [  1600/  1680 ( 96%)] Loss: 0.117551  Accuracy: 93.26%
Epoch 2 - Time: 43.32s - Train Loss: 0.181060 - Train Accuracy: 92.38%
Test Loss: 0.294938 - Test Accuracy: 92.19%

Train Epoch:   3 [     0/  1680 (  0%)] Loss: 0.141607  Accuracy: 96.88%
Train Epoch:   3 [   320/  1680 ( 19%)] Loss: 0.124201  Accuracy: 93.47%
Train Epoch:   3 [   640/  1680 ( 38%)] Loss: 0.124381  Accuracy: 92.71%
Train Epoch:   3 [   960/  1680 ( 58%)] Loss: 0.291106  Accuracy: 92.44%
Train Epoch:   3 [  1280/  1680 ( 77%)] Loss: 0.289438  Accuracy: 92.68%
Train Epoch:   3 [  1600/  1680 ( 96%)] Loss: 0.119186  Accuracy: 92.71%
Epoch 3 - Time: 42.87s - Train Loss: 0.175521 - Train Accuracy: 91.90%
Test Loss: 0.160474 - Test Accuracy: 95.94%

Train Epoch:   4 [     0/  1680 (  0%)] Loss: 0.117496  Accuracy: 100.00%
Train Epoch:   4 [   320/  1680 ( 19%)] Loss: 0.089305  Accuracy: 94.60%
Train Epoch:   4 [   640/  1680 ( 38%)] Loss: 0.145823  Accuracy: 94.94%
Train Epoch:   4 [   960/  1680 ( 58%)] Loss: 0.103748  Accuracy: 95.26%
Train Epoch:   4 [  1280/  1680 ( 77%)] Loss: 0.242247  Accuracy: 94.97%
Train Epoch:   4 [  1600/  1680 ( 96%)] Loss: 0.122512  Accuracy: 94.85%
Epoch 4 - Time: 44.11s - Train Loss: 0.134570 - Train Accuracy: 93.87%
Test Loss: 0.228681 - Test Accuracy: 90.94%

Train Epoch:   5 [     0/  1680 (  0%)] Loss: 0.101651  Accuracy: 96.88%
Train Epoch:   5 [   320/  1680 ( 19%)] Loss: 0.208671  Accuracy: 94.89%
Train Epoch:   5 [   640/  1680 ( 38%)] Loss: 0.072090  Accuracy: 94.94%
Train Epoch:   5 [   960/  1680 ( 58%)] Loss: 0.259198  Accuracy: 94.35%
Train Epoch:   5 [  1280/  1680 ( 77%)] Loss: 0.106292  Accuracy: 94.13%
Train Epoch:   5 [  1600/  1680 ( 96%)] Loss: 0.110741  Accuracy: 94.36%
Epoch 5 - Time: 43.58s - Train Loss: 0.141572 - Train Accuracy: 93.45%
Test Loss: 0.118991 - Test Accuracy: 97.19%

Train Epoch:   6 [     0/  1680 (  0%)] Loss: 0.238775  Accuracy: 84.38%
Train Epoch:   6 [   320/  1680 ( 19%)] Loss: 0.122788  Accuracy: 94.03%
Train Epoch:   6 [   640/  1680 ( 38%)] Loss: 0.012993  Accuracy: 95.98%
Train Epoch:   6 [   960/  1680 ( 58%)] Loss: 0.094011  Accuracy: 95.87%
Train Epoch:   6 [  1280/  1680 ( 77%)] Loss: 0.145716  Accuracy: 95.58%
Train Epoch:   6 [  1600/  1680 ( 96%)] Loss: 0.544419  Accuracy: 95.47%
```

```
Epoch 6 - Time: 58.67s - Train Loss: 0.129391 - Train Accuracy: 94.46%
Test Loss: 0.152756 - Test Accuracy: 96.56%

Train Epoch:   7 [     0/  1680 (  0%)] Loss: 0.110688  Accuracy: 96.88%
Train Epoch:   7 [   320/  1680 ( 19%)] Loss: 0.115284  Accuracy: 95.45%
Train Epoch:   7 [   640/  1680 ( 38%)] Loss: 0.113117  Accuracy: 95.39%
Train Epoch:   7 [   960/  1680 ( 58%)] Loss: 0.078003  Accuracy: 95.97%
Train Epoch:   7 [  1280/  1680 ( 77%)] Loss: 0.191643  Accuracy: 96.04%
Train Epoch:   7 [  1600/  1680 ( 96%)] Loss: 0.159519  Accuracy: 95.96%
Epoch 7 - Time: 45.36s - Train Loss: 0.102312 - Train Accuracy: 95.00%
Test Loss: 0.215539 - Test Accuracy: 95.00%

Train Epoch:   8 [     0/  1680 (  0%)] Loss: 0.151010  Accuracy: 96.88%
Train Epoch:   8 [   320/  1680 ( 19%)] Loss: 0.082022  Accuracy: 94.03%
Train Epoch:   8 [   640/  1680 ( 38%)] Loss: 0.019679  Accuracy: 95.54%
Train Epoch:   8 [   960/  1680 ( 58%)] Loss: 0.090209  Accuracy: 94.66%
Train Epoch:   8 [  1280/  1680 ( 77%)] Loss: 0.244851  Accuracy: 94.59%
Train Epoch:   8 [  1600/  1680 ( 96%)] Loss: 0.130030  Accuracy: 94.24%
Epoch 8 - Time: 49.18s - Train Loss: 0.145277 - Train Accuracy: 93.39%
Test Loss: 0.354122 - Test Accuracy: 86.56%

Train Epoch:   9 [     0/  1680 (  0%)] Loss: 0.343555  Accuracy: 84.38%
Train Epoch:   9 [   320/  1680 ( 19%)] Loss: 0.080193  Accuracy: 94.89%
Train Epoch:   9 [   640/  1680 ( 38%)] Loss: 0.105633  Accuracy: 95.09%
Train Epoch:   9 [   960/  1680 ( 58%)] Loss: 0.054809  Accuracy: 94.76%
Train Epoch:   9 [  1280/  1680 ( 77%)] Loss: 0.095516  Accuracy: 95.05%
Train Epoch:   9 [  1600/  1680 ( 96%)] Loss: 0.084872  Accuracy: 95.22%
Epoch 9 - Time: 44.47s - Train Loss: 0.117733 - Train Accuracy: 94.29%
Test Loss: 0.195516 - Test Accuracy: 95.00%

---------------------------------------------------------------------------
--------------------------
---------------------------------------------------------------------------
--------------------------
FINAL RESULTS:
---------------------------------------------------------------------------
--------------------------
epoch_times: [42.76304793357849, 43.31869101524353, 43.31773591041565, 42.
86594581604004, 44.11113405227661, 43.584967851638794, 58.66502928733826,
45.360710859298706, 49.17667317390442, 44.471909046173096]
train_losses: [0.31219020901391137, 0.17385263096254605, 0.181060227780387
94, 0.17552123677272063, 0.13456989785369772, 0.1415715190367O567, 0.12939
078459301248, 0.10231199516699864, 0.14527743521074837, 0.1177330625315125
1]
train_accuracies: [86.42857142857143, 93.03571428571429, 92.3809523809523
8, 91.9047619047619, 93.86904761904762, 93.45238095238095, 94.464285714285
71, 95.0, 93.39285714285714, 94.28571428571429]
test_losses: [0.18100014608353376, 0.18387161334976554, 0.2949376486241817
6, 0.16047407081350684, 0.22868142360821367, 0.11899070171639323, 0.152755
83432521672, 0.2155387081365916, 0.3541224246728234, 0.19551602460269352]
test_accuracies: [94.6875, 95.3125, 92.1875, 95.9375, 90.9375, 97.1875, 9
6.5625, 95.0, 86.5625, 95.0]
```

In [33]:
```
train_dataset = GenreDatasetPhaseMFCC("Data/binary_data/train/", n_fft=20
test_dataset = GenreDatasetPhaseMFCC("Data/binary_data/test/", n_fft=2048
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, 2)

    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_e3)

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/  1680 (  0%)] Loss: 0.735650  Accuracy: 53.12%
Train Epoch:   0 [   320/  1680 ( 19%)] Loss: 0.868494  Accuracy: 51.14%
Train Epoch:   0 [   640/  1680 ( 38%)] Loss: 0.549072  Accuracy: 55.95%
Train Epoch:   0 [   960/  1680 ( 58%)] Loss: 0.466943  Accuracy: 62.40%
Train Epoch:   0 [  1280/  1680 ( 77%)] Loss: 0.244168  Accuracy: 68.29%
Train Epoch:   0 [  1600/  1680 ( 96%)] Loss: 0.429500  Accuracy: 71.69%
Epoch 0 - Time: 73.58s - Train Loss: 0.603171 - Train Accuracy: 71.31%
Test Loss: 0.336265 - Test Accuracy: 87.50%

Train Epoch:   1 [     0/  1680 (  0%)] Loss: 0.425153  Accuracy: 78.12%
Train Epoch:   1 [   320/  1680 ( 19%)] Loss: 0.595793  Accuracy: 77.27%
Train Epoch:   1 [   640/  1680 ( 38%)] Loss: 0.317432  Accuracy: 82.44%
Train Epoch:   1 [   960/  1680 ( 58%)] Loss: 0.321401  Accuracy: 82.66%
Train Epoch:   1 [  1280/  1680 ( 77%)] Loss: 0.155262  Accuracy: 83.08%
Train Epoch:   1 [  1600/  1680 ( 96%)] Loss: 0.352507  Accuracy: 82.60%
Epoch 1 - Time: 51.37s - Train Loss: 0.424381 - Train Accuracy: 81.61%
Test Loss: 0.602541 - Test Accuracy: 66.56%

Train Epoch:   2 [     0/  1680 (  0%)] Loss: 0.850426  Accuracy: 50.00%
Train Epoch:   2 [   320/  1680 ( 19%)] Loss: 0.239433  Accuracy: 78.98%
Train Epoch:   2 [   640/  1680 ( 38%)] Loss: 0.500215  Accuracy: 82.29%
Train Epoch:   2 [   960/  1680 ( 58%)] Loss: 0.218858  Accuracy: 83.97%
Train Epoch:   2 [  1280/  1680 ( 77%)] Loss: 0.332864  Accuracy: 85.59%
Train Epoch:   2 [  1600/  1680 ( 96%)] Loss: 0.262290  Accuracy: 86.95%
Epoch 2 - Time: 57.94s - Train Loss: 0.305707 - Train Accuracy: 86.13%
Test Loss: 0.280999 - Test Accuracy: 90.31%

Train Epoch:   3 [     0/  1680 (  0%)] Loss: 0.145187  Accuracy: 93.75%
Train Epoch:   3 [   320/  1680 ( 19%)] Loss: 0.264619  Accuracy: 88.35%
Train Epoch:   3 [   640/  1680 ( 38%)] Loss: 0.230185  Accuracy: 87.65%
Train Epoch:   3 [   960/  1680 ( 58%)] Loss: 0.407401  Accuracy: 87.80%
Train Epoch:   3 [  1280/  1680 ( 77%)] Loss: 0.211505  Accuracy: 88.49%
Train Epoch:   3 [  1600/  1680 ( 96%)] Loss: 0.181106  Accuracy: 89.09%
Epoch 3 - Time: 47.14s - Train Loss: 0.285976 - Train Accuracy: 88.33%
Test Loss: 0.276645 - Test Accuracy: 91.56%

Train Epoch:   4 [     0/  1680 (  0%)] Loss: 0.497322  Accuracy: 87.50%
Train Epoch:   4 [   320/  1680 ( 19%)] Loss: 0.536946  Accuracy: 87.50%
Train Epoch:   4 [   640/  1680 ( 38%)] Loss: 0.299088  Accuracy: 88.54%
Train Epoch:   4 [   960/  1680 ( 58%)] Loss: 0.295729  Accuracy: 87.90%
Train Epoch:   4 [  1280/  1680 ( 77%)] Loss: 0.254625  Accuracy: 88.34%
Train Epoch:   4 [  1600/  1680 ( 96%)] Loss: 0.490942  Accuracy: 88.30%
Epoch 4 - Time: 51.99s - Train Loss: 0.285149 - Train Accuracy: 87.50%
Test Loss: 0.246197 - Test Accuracy: 90.94%

Train Epoch:   5 [     0/  1680 (  0%)] Loss: 0.327977  Accuracy: 84.38%
Train Epoch:   5 [   320/  1680 ( 19%)] Loss: 0.147034  Accuracy: 91.76%
Train Epoch:   5 [   640/  1680 ( 38%)] Loss: 0.187032  Accuracy: 90.48%
Train Epoch:   5 [   960/  1680 ( 58%)] Loss: 0.338331  Accuracy: 90.22%
Train Epoch:   5 [  1280/  1680 ( 77%)] Loss: 0.390736  Accuracy: 88.26%
Train Epoch:   5 [  1600/  1680 ( 96%)] Loss: 0.275194  Accuracy: 88.05%
Epoch 5 - Time: 51.88s - Train Loss: 0.301933 - Train Accuracy: 87.32%
Test Loss: 0.271775 - Test Accuracy: 89.06%

Train Epoch:   6 [     0/  1680 (  0%)] Loss: 0.331382  Accuracy: 84.38%
Train Epoch:   6 [   320/  1680 ( 19%)] Loss: 0.304538  Accuracy: 90.62%
Train Epoch:   6 [   640/  1680 ( 38%)] Loss: 0.093887  Accuracy: 91.37%
Train Epoch:   6 [   960/  1680 ( 58%)] Loss: 0.245574  Accuracy: 91.03%
Train Epoch:   6 [  1280/  1680 ( 77%)] Loss: 0.315051  Accuracy: 89.10%
Train Epoch:   6 [  1600/  1680 ( 96%)] Loss: 0.260195  Accuracy: 89.28%
```

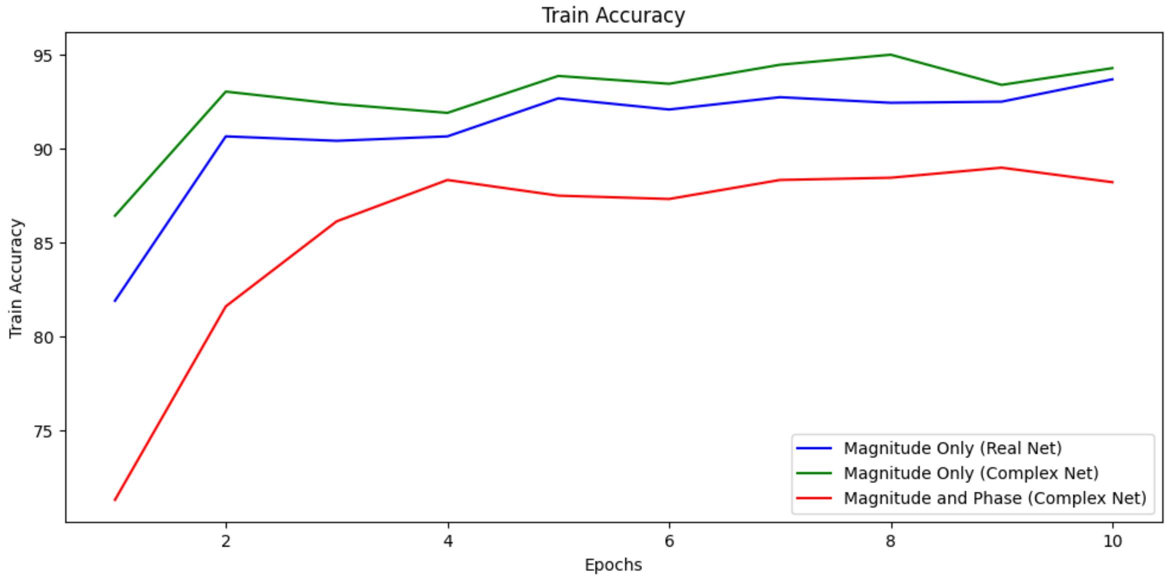
```
Epoch 6 - Time: 48.36s - Train Loss: 0.282665 - Train Accuracy: 88.33%
Test Loss: 0.210059 - Test Accuracy: 90.94%

Train Epoch:   7 [     0/  1680 (  0%)] Loss: 0.095637  Accuracy: 96.88%
Train Epoch:   7 [   320/  1680 ( 19%)] Loss: 0.291555  Accuracy: 90.34%
Train Epoch:   7 [   640/  1680 ( 38%)] Loss: 0.313008  Accuracy: 89.73%
Train Epoch:   7 [   960/  1680 ( 58%)] Loss: 0.348511  Accuracy: 89.82%
Train Epoch:   7 [  1280/  1680 ( 77%)] Loss: 0.470430  Accuracy: 89.33%
Train Epoch:   7 [  1600/  1680 ( 96%)] Loss: 0.302118  Accuracy: 89.15%
Epoch 7 - Time: 48.26s - Train Loss: 0.269804 - Train Accuracy: 88.45%
Test Loss: 0.201279 - Test Accuracy: 92.81%

Train Epoch:   8 [     0/  1680 (  0%)] Loss: 0.291109  Accuracy: 87.50%
Train Epoch:   8 [   320/  1680 ( 19%)] Loss: 0.220785  Accuracy: 89.49%
Train Epoch:   8 [   640/  1680 ( 38%)] Loss: 0.301213  Accuracy: 88.54%
Train Epoch:   8 [   960/  1680 ( 58%)] Loss: 0.490905  Accuracy: 88.81%
Train Epoch:   8 [  1280/  1680 ( 77%)] Loss: 0.228495  Accuracy: 89.25%
Train Epoch:   8 [  1600/  1680 ( 96%)] Loss: 0.160624  Accuracy: 89.71%
Epoch 8 - Time: 48.60s - Train Loss: 0.256629 - Train Accuracy: 88.99%
Test Loss: 0.293889 - Test Accuracy: 90.00%

Train Epoch:   9 [     0/  1680 (  0%)] Loss: 0.375799  Accuracy: 81.25%
Train Epoch:   9 [   320/  1680 ( 19%)] Loss: 0.247870  Accuracy: 89.49%
Train Epoch:   9 [   640/  1680 ( 38%)] Loss: 0.087902  Accuracy: 90.62%
Train Epoch:   9 [   960/  1680 ( 58%)] Loss: 0.392372  Accuracy: 90.02%
Train Epoch:   9 [  1280/  1680 ( 77%)] Loss: 0.262468  Accuracy: 88.95%
Train Epoch:   9 [  1600/  1680 ( 96%)] Loss: 0.162650  Accuracy: 88.91%
Epoch 9 - Time: 48.17s - Train Loss: 0.255111 - Train Accuracy: 88.21%
Test Loss: 0.277363 - Test Accuracy: 91.56%

---------------------------------------------------------------------------
--------------------------
---------------------------------------------------------------------------
--------------------------
FINAL RESULTS:
---------------------------------------------------------------------------
--------------------------
epoch_times: [73.57843279838562, 51.3704469203949, 57.93836307525635, 47.1
35130882263184, 51.98855710029602, 51.87985110282898, 48.36233329772949, 4
8.26110076904297, 48.60330104827881, 48.16680717468262]
train_losses: [0.603170841645736, 0.4243808463215828, 0.30570677667856216,
0.285976423523747, 0.2851490078923794, 0.30193280887145263, 0.282665444108
1561, 0.2698039590882567, 0.25662878531819355, 0.25511147984518456]
train_accuracies: [71.30952380952381, 81.60714285714286, 86.1309523809523
8, 88.33333333333333, 87.5, 87.32142857142857, 88.33333333333333, 88.45238
095238095, 88.98809523809524, 88.21428571428571]
test_losses: [0.3362645523622632, 0.6025413427501917, 0.2809987593907863,
0.2766452480107546, 0.24619730543345214, 0.27177466712892057, 0.2100593810
9010458, 0.20127928256988525, 0.29388898848555983, 0.2773632241412997]
test_accuracies: [87.5, 66.5625, 90.3125, 91.5625, 90.9375, 89.0625, 90.93
75, 92.8125, 90.0, 91.5625]
```

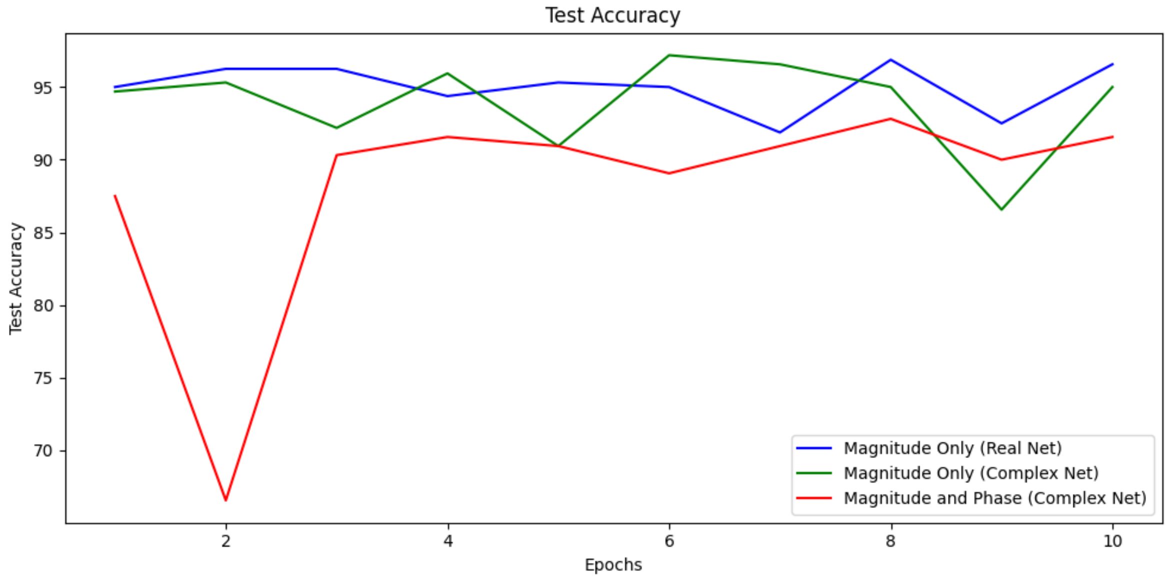
Plots

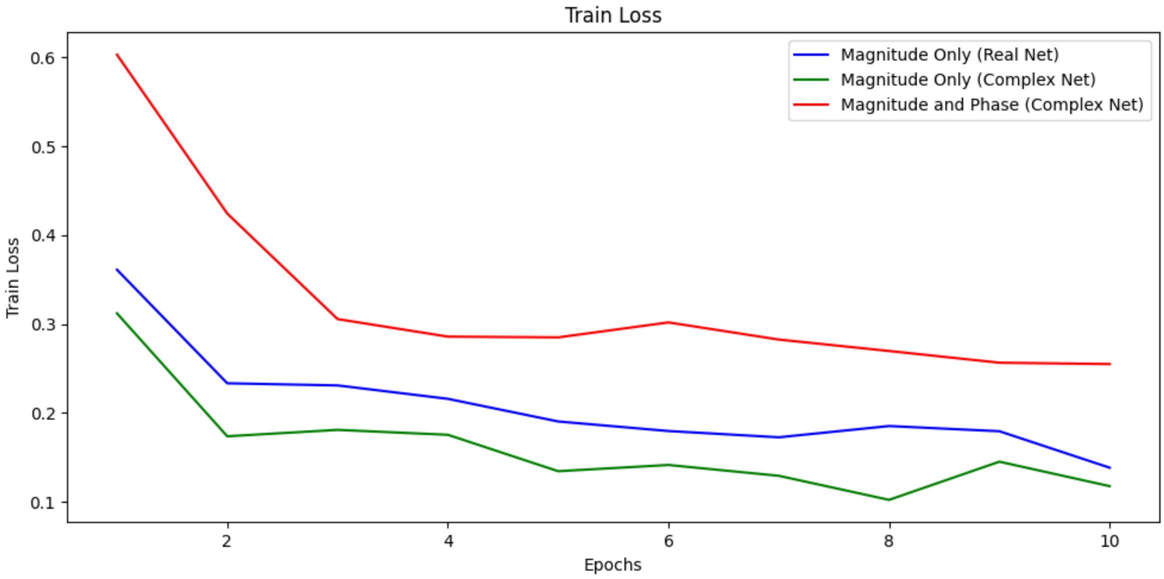
In [35]:

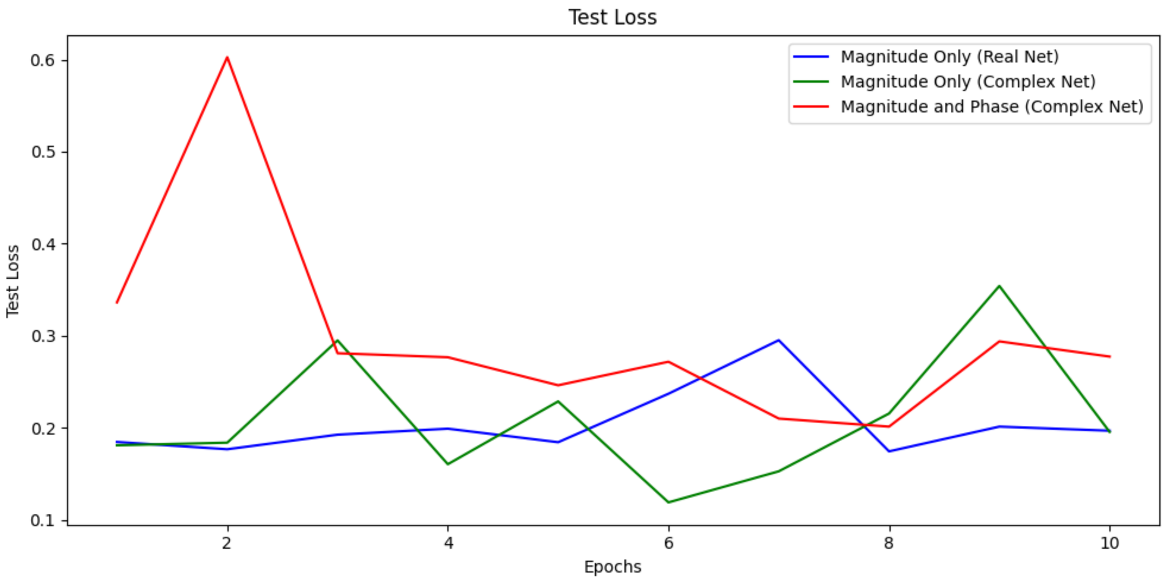
```
# Data for the four scenarios
data = {
    "Magnitude Only (Real Net)": metrics_dict_e1,
    "Magnitude Only (Complex Net)": metrics_dict_e2,
```

```
    "Magnitude and Phase (Complex Net)": metrics_dict_e3
}

# Data for plotting
epochs = range(1, 11)
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=scenar

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=scenari

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,

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, c

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, colo
axes.set_title("Time")
axes.set_xlabel("Epochs")
axes.set_ylabel("Time (secs)")
axes.legend()
```

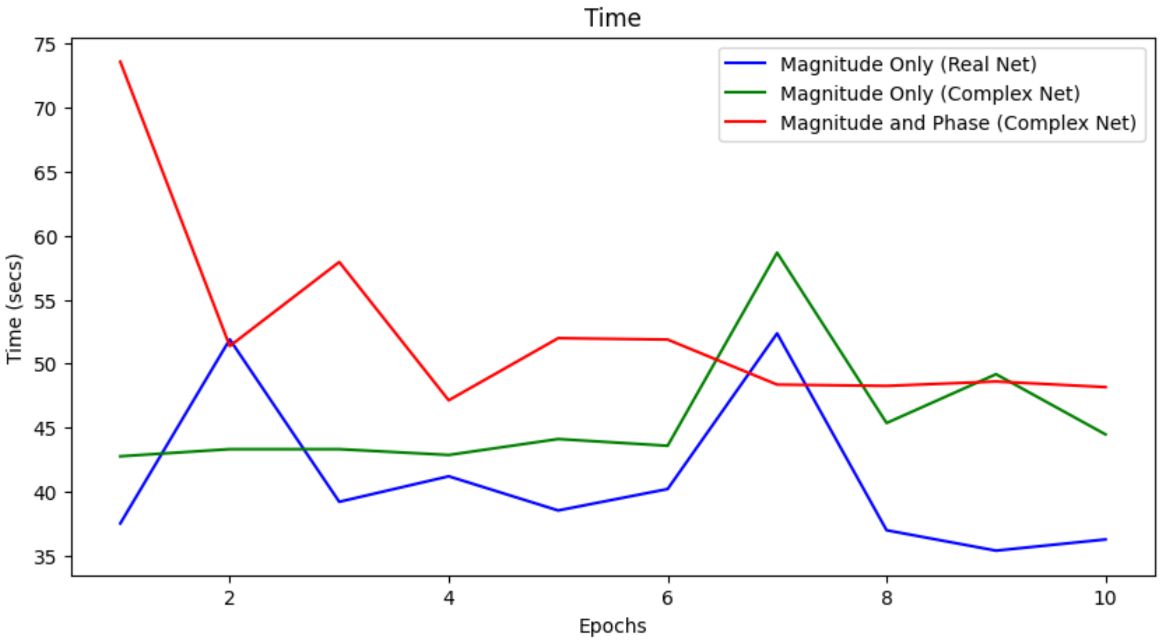

Train Accuracy
Train Accuracy
Epochs
95
90
85
80
75
2
4
6
8
10
Magnitude Only (Real Net)
Magnitude Only (Complex Net)
Magnitude and Phase (Complex Net)


Test Accuracy
Test Accuracy
Epochs
95
90
85
80
75
70
2
4
6
8
10
Magnitude Only (Real Net)
Magnitude Only (Complex Net)
Magnitude and Phase (Complex Net)





Out[35]: <matplotlib.legend.Legend at 0x2a9d28a90>



New way to Extract complex valued mfccs

In [52]:
```
class MusicFeatureExtractorComplex2:
    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
        return frames

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

    def met_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 +
        freqs = self.met_to_freq(mels)
        return np.floor((self.FFT_size + 1) / sample_rate * freqs).astype

    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.linsp
            filters[n, filter_points[n + 1]: filter_points[n + 2]] = np.l
        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
        for i in range(1, self.dct_filter_num):
            basis[i, :] = np.cos(i * samples) * np.sqrt(2.0 / self.mel_fi
        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.sha
        for n in range(audio_fft.shape[1]):
            audio_fft[:, n] = fft.fft(audio_winT[:, n], axis=0)[:audio_ff
        audio_fft = np.transpose(audio_fft)
        mag_fft = np.square(np.abs(audio_fft))
        phase_fft = np.angle(audio_fft)
```

```
        freq_min = 0
        freq_high = sample_rate / 2
        filter_points, mel_freqs = self.get_filter_points(freq_min, freq_
        filters = self.get_filters(filter_points)
        audio_filtered = np.dot(filters, np.transpose(mag_fft))
        phase_filtered = np.dot(filters, np.transpose(phase_fft))
        audio_filtered = np.maximum(audio_filtered, self.epsilon)  # Repl
        audio_log = 10.0 * np.log10(audio_filtered)
        dct_filters = self.dct()
        cepstral_coefficents = np.dot(dct_filters, audio_log)
        phase_coefficents = np.dot(dct_filters, phase_filtered)
        return np.array([cepstral_coefficents*np.exp(1j*phase_coefficents

class GenreDatasetPhaseMFCC2(GenreDatasetMFCC):

    def __init__(self, train_path, n_fft=2048, hop_length=512, num_segmen
        super().__init__(train_path, n_fft, hop_length, num_segments, mel
        self.mfcc_extractor = MusicFeatureExtractorComplex2(
            FFT_size=n_fft, HOP_SIZE=hop_length, mel_filter_num = mel_fil

    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("/")[3]]
        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_signa
        cur_mfcc = self.mfcc_extractor.get_mfcc_features(cur_signal, samp
        cur_mfcc = self.adjust_shape(cur_mfcc)
        return torch.tensor(cur_mfcc, dtype=torch.complex64), target
```

In [53]:

```
train_dataset = GenreDatasetPhaseMFCC2("Data/binary_data/train/", n_fft=2
test_dataset = GenreDatasetPhaseMFCC2("Data/binary_data/test/", n_fft=204
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, 2)

    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_e4 = {
    '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_e4)

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

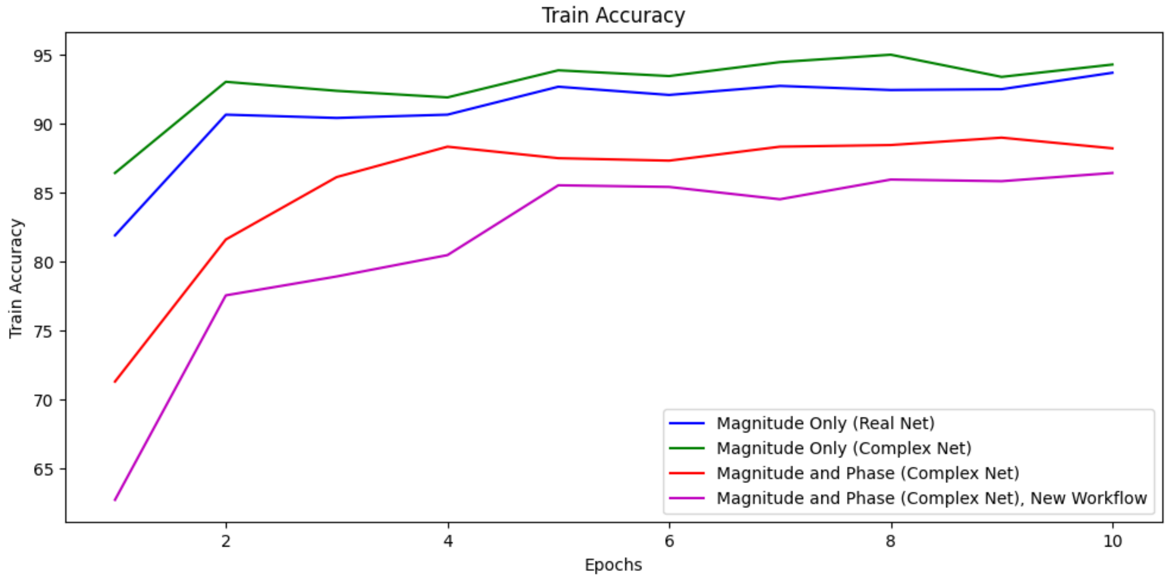
```
Train Epoch:   0 [     0/  1680 (  0%)] Loss: 0.756448  Accuracy: 43.75%
Train Epoch:   0 [   320/  1680 ( 19%)] Loss: 0.943351  Accuracy: 51.70%
Train Epoch:   0 [   640/  1680 ( 38%)] Loss: 0.641432  Accuracy: 51.64%
Train Epoch:   0 [   960/  1680 ( 58%)] Loss: 0.497405  Accuracy: 56.96%
Train Epoch:   0 [  1280/  1680 ( 77%)] Loss: 0.510098  Accuracy: 61.36%
Train Epoch:   0 [  1600/  1680 ( 96%)] Loss: 0.745694  Accuracy: 62.99%
Epoch 0 - Time: 47.31s - Train Loss: 0.669079 - Train Accuracy: 62.74%
Test Loss: 0.466456 - Test Accuracy: 78.75%

Train Epoch:   1 [     0/  1680 (  0%)] Loss: 0.427284  Accuracy: 81.25%
Train Epoch:   1 [   320/  1680 ( 19%)] Loss: 0.414837  Accuracy: 79.26%
Train Epoch:   1 [   640/  1680 ( 38%)] Loss: 0.425875  Accuracy: 77.83%
Train Epoch:   1 [   960/  1680 ( 58%)] Loss: 0.465073  Accuracy: 79.23%
Train Epoch:   1 [  1280/  1680 ( 77%)] Loss: 0.417217  Accuracy: 79.04%
Train Epoch:   1 [  1600/  1680 ( 96%)] Loss: 0.525346  Accuracy: 78.37%
Epoch 1 - Time: 49.40s - Train Loss: 0.464559 - Train Accuracy: 77.56%
Test Loss: 0.380537 - Test Accuracy: 81.56%

Train Epoch:   2 [     0/  1680 (  0%)] Loss: 0.328772  Accuracy: 90.62%
Train Epoch:   2 [   320/  1680 ( 19%)] Loss: 0.275012  Accuracy: 79.55%
Train Epoch:   2 [   640/  1680 ( 38%)] Loss: 0.526225  Accuracy: 74.55%
Train Epoch:   2 [   960/  1680 ( 58%)] Loss: 0.534228  Accuracy: 78.12%
Train Epoch:   2 [  1280/  1680 ( 77%)] Loss: 0.513447  Accuracy: 79.04%
Train Epoch:   2 [  1600/  1680 ( 96%)] Loss: 0.389921  Accuracy: 79.66%
Epoch 2 - Time: 49.73s - Train Loss: 0.462108 - Train Accuracy: 78.93%
Test Loss: 0.376359 - Test Accuracy: 83.12%

Train Epoch:   3 [     0/  1680 (  0%)] Loss: 0.300185  Accuracy: 87.50%
Train Epoch:   3 [   320/  1680 ( 19%)] Loss: 0.346790  Accuracy: 80.97%
Train Epoch:   3 [   640/  1680 ( 38%)] Loss: 0.440910  Accuracy: 81.10%
Train Epoch:   3 [   960/  1680 ( 58%)] Loss: 0.374527  Accuracy: 80.65%
Train Epoch:   3 [  1280/  1680 ( 77%)] Loss: 0.237164  Accuracy: 81.33%
Train Epoch:   3 [  1600/  1680 ( 96%)] Loss: 0.439932  Accuracy: 81.31%
Epoch 3 - Time: 48.36s - Train Loss: 0.390184 - Train Accuracy: 80.48%
Test Loss: 0.349970 - Test Accuracy: 86.25%

Train Epoch:   4 [     0/  1680 (  0%)] Loss: 0.272466  Accuracy: 78.12%
Train Epoch:   4 [   320/  1680 ( 19%)] Loss: 0.492965  Accuracy: 85.80%
Train Epoch:   4 [   640/  1680 ( 38%)] Loss: 0.516339  Accuracy: 84.67%
Train Epoch:   4 [   960/  1680 ( 58%)] Loss: 0.494687  Accuracy: 84.98%
Train Epoch:   4 [  1280/  1680 ( 77%)] Loss: 0.296013  Accuracy: 86.05%
Train Epoch:   4 [  1600/  1680 ( 96%)] Loss: 0.349828  Accuracy: 86.27%
Epoch 4 - Time: 47.35s - Train Loss: 0.330548 - Train Accuracy: 85.54%
Test Loss: 0.290894 - Test Accuracy: 85.94%

Train Epoch:   5 [     0/  1680 (  0%)] Loss: 0.229566  Accuracy: 87.50%
Train Epoch:   5 [   320/  1680 ( 19%)] Loss: 0.342616  Accuracy: 86.93%
Train Epoch:   5 [   640/  1680 ( 38%)] Loss: 0.348982  Accuracy: 87.95%
Train Epoch:   5 [   960/  1680 ( 58%)] Loss: 0.236092  Accuracy: 88.31%
Train Epoch:   5 [  1280/  1680 ( 77%)] Loss: 0.178407  Accuracy: 86.81%
Train Epoch:   5 [  1600/  1680 ( 96%)] Loss: 0.365560  Accuracy: 86.15%
Epoch 5 - Time: 51.33s - Train Loss: 0.331476 - Train Accuracy: 85.42%
Test Loss: 0.256633 - Test Accuracy: 90.62%

Train Epoch:   6 [     0/  1680 (  0%)] Loss: 0.229851  Accuracy: 93.75%
Train Epoch:   6 [   320/  1680 ( 19%)] Loss: 0.202679  Accuracy: 88.92%
Train Epoch:   6 [   640/  1680 ( 38%)] Loss: 0.370229  Accuracy: 86.61%
Train Epoch:   6 [   960/  1680 ( 58%)] Loss: 0.450266  Accuracy: 86.49%
Train Epoch:   6 [  1280/  1680 ( 77%)] Loss: 0.280140  Accuracy: 85.52%
Train Epoch:   6 [  1600/  1680 ( 96%)] Loss: 0.274103  Accuracy: 85.17%
```

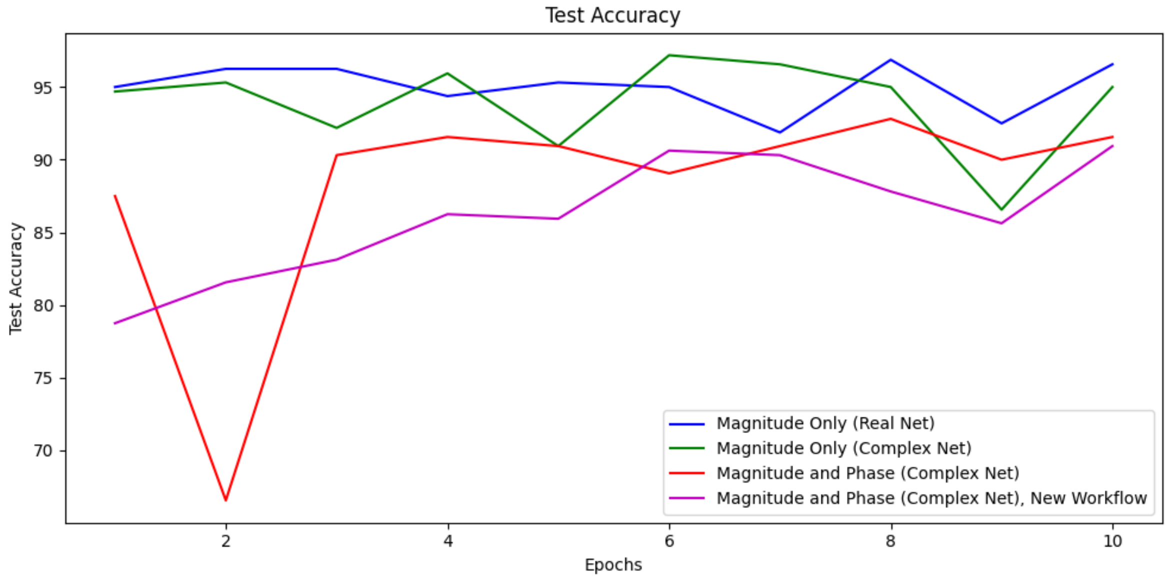
```
Epoch 6 - Time: 49.00s - Train Loss: 0.342386 - Train Accuracy: 84.52%
Test Loss: 0.264726 - Test Accuracy: 90.31%

Train Epoch:   7 [     0/  1680 (  0%)] Loss: 0.195107  Accuracy: 96.88%
Train Epoch:   7 [   320/  1680 ( 19%)] Loss: 0.341948  Accuracy: 85.80%
Train Epoch:   7 [   640/  1680 ( 38%)] Loss: 0.352814  Accuracy: 86.01%
Train Epoch:   7 [   960/  1680 ( 58%)] Loss: 0.173795  Accuracy: 86.90%
Train Epoch:   7 [  1280/  1680 ( 77%)] Loss: 0.332959  Accuracy: 86.51%
Train Epoch:   7 [  1600/  1680 ( 96%)] Loss: 0.185387  Accuracy: 86.83%
Epoch 7 - Time: 49.10s - Train Loss: 0.324903 - Train Accuracy: 85.95%
Test Loss: 0.259553 - Test Accuracy: 87.81%

Train Epoch:   8 [     0/  1680 (  0%)] Loss: 0.358597  Accuracy: 75.00%
Train Epoch:   8 [   320/  1680 ( 19%)] Loss: 0.416254  Accuracy: 87.50%
Train Epoch:   8 [   640/  1680 ( 38%)] Loss: 0.467524  Accuracy: 87.35%
Train Epoch:   8 [   960/  1680 ( 58%)] Loss: 0.174863  Accuracy: 86.59%
Train Epoch:   8 [  1280/  1680 ( 77%)] Loss: 0.237571  Accuracy: 87.27%
Train Epoch:   8 [  1600/  1680 ( 96%)] Loss: 0.638832  Accuracy: 86.83%
Epoch 8 - Time: 48.76s - Train Loss: 0.315972 - Train Accuracy: 85.83%
Test Loss: 0.280964 - Test Accuracy: 85.62%

Train Epoch:   9 [     0/  1680 (  0%)] Loss: 0.251841  Accuracy: 90.62%
Train Epoch:   9 [   320/  1680 ( 19%)] Loss: 0.296167  Accuracy: 87.50%
Train Epoch:   9 [   640/  1680 ( 38%)] Loss: 0.168065  Accuracy: 87.65%
Train Epoch:   9 [   960/  1680 ( 58%)] Loss: 0.304702  Accuracy: 87.10%
Train Epoch:   9 [  1280/  1680 ( 77%)] Loss: 0.210842  Accuracy: 87.20%
Train Epoch:   9 [  1600/  1680 ( 96%)] Loss: 0.331602  Accuracy: 87.19%
Epoch 9 - Time: 49.46s - Train Loss: 0.307399 - Train Accuracy: 86.43%
Test Loss: 0.240680 - Test Accuracy: 90.94%

---------------------------------------------------------------------------
--------------------------
---------------------------------------------------------------------------
--------------------------
FINAL RESULTS:
---------------------------------------------------------------------------
--------------------------
epoch_times: [73.57843279838562, 51.3704469203949, 57.93836307525635, 47.1
35130882263184, 51.98855710029602, 51.87985110282898, 48.36233329772949, 4
8.26110076904297, 48.60330104827881, 48.16680717468262]
train_losses: [0.603170841645736, 0.4243808463215828, 0.30570677667856216,
0.285976423523747, 0.2851490078923794, 0.30193280887145263, 0.282665444108
1561, 0.2698039590882567, 0.25662878531819355, 0.25511147984518456]
train_accuracies: [71.30952380952381, 81.60714285714286, 86.1309523809523
8, 88.33333333333333, 87.5, 87.32142857142857, 88.33333333333333, 88.45238
095238095, 88.98809523809524, 88.21428571428571]
test_losses: [0.3362645523622632, 0.6025413427501917, 0.2809987593907863,
0.2766452480107546, 0.24619730543345214, 0.27177466712892057, 0.2100593810
9010458, 0.20127928256988525, 0.29388898848555983, 0.2773632241412997]
test_accuracies: [87.5, 66.5625, 90.3125, 91.5625, 90.9375, 89.0625, 90.93
75, 92.8125, 90.0, 91.5625]
```

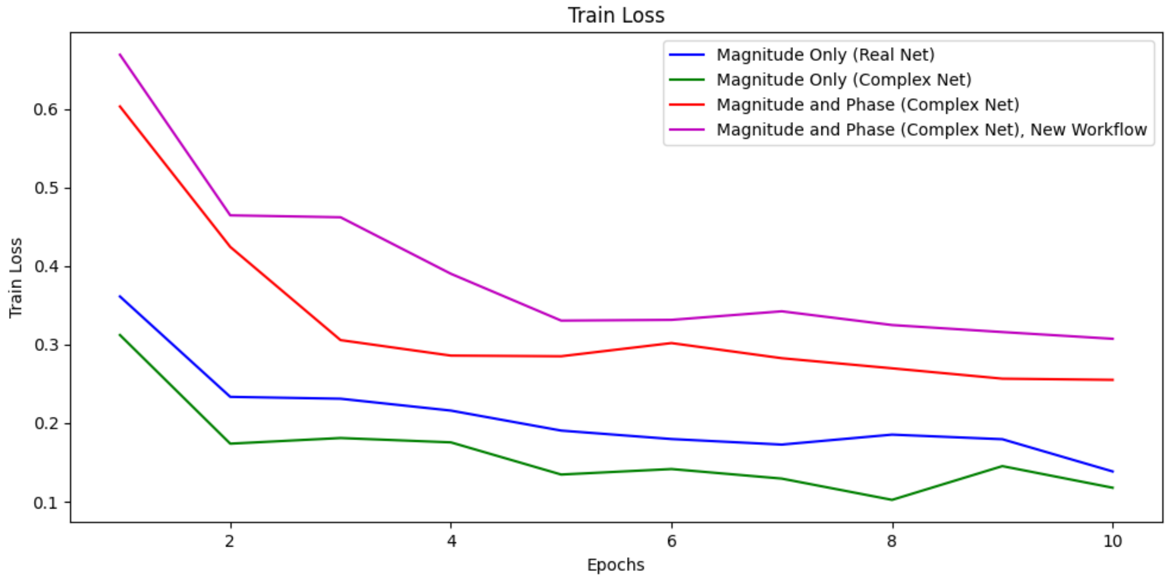
In [55]:

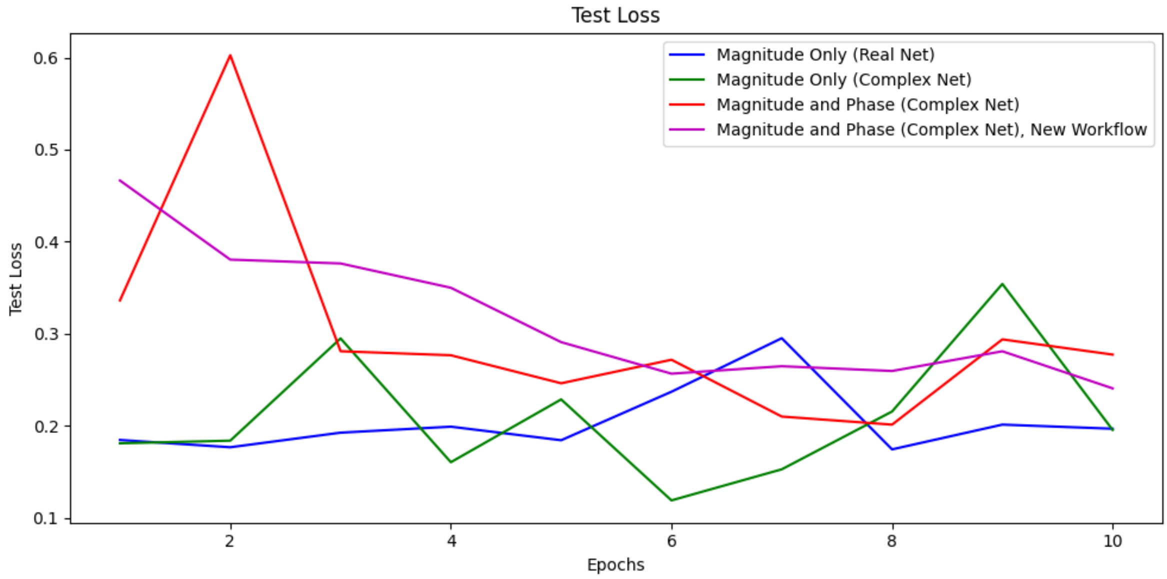
```python
# Data for the four scenarios
data = {
    "Magnitude Only (Real Net)": metrics_dict_e1,
    "Magnitude Only (Complex Net)": metrics_dict_e2,
    "Magnitude and Phase (Complex Net)": metrics_dict_e3,
    "Magnitude and Phase (Complex Net), New Workflow": metrics_dict_e4
}
```

```python
# Data for plotting
epochs = range(1, 11)
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=scenar

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=scenari

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,

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, c

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, colo
axes.set_title("Time")
axes.set_xlabel("Epochs")
axes.set_ylabel("Time (secs)")
axes.legend()
```


Train Accuracy
Train Accuracy
Epochs
Magnitude Only (Real Net)
Magnitude Only (Complex Net)
Magnitude and Phase (Complex Net)
Magnitude and Phase (Complex Net), New Workflow


Test Accuracy
Test Accuracy
Epochs
Magnitude Only (Real Net)
Magnitude Only (Complex Net)
Magnitude and Phase (Complex Net)
Magnitude and Phase (Complex Net), New Workflow






```
Out[55]: <matplotlib.legend.Legend at 0x2b777f7d0>
```