



دانشگاه تهران

دانشکده علوم و فنون نوین

تمرین شماره چهار پردازش گفتار

فاطمه چیت ساز	نام و نام خانوادگی
830402092	شماره دانشجویی
1403 10 خرداد	تاریخ ارسال گزارش

سوال یک

1. حوزه بازشناسی گفتار

بازشناسی گفتار به فرآیند تبدیل گفتار به متن گفته می‌شود. در این حوزه دو پروژه برجسته شامل Wav2Vec و Whisper مورد بررسی قرار می‌گیرند.

Wav2Vec

- مشخصات: Facebook AI Research توسط Wav2Vec توسعه داده شده است و هدف آن بهبود کیفیت بازشناسی گفتار با استفاده از یادگیری خودنظری است.
- الگوریتم‌ها: این مدل از معماری ترانسفورمر بهره می‌برد و با استفاده از Self-Supervised Learning ابتدا ویژگی‌های گفتار را استخراج کرده و سپس با استفاده از مقادیر بزرگ داده‌های بدون برچسب مدل را آموزش می‌دهد.
- مقایسه: یکی از مزیت‌های اصلی Wav2Vec کاهش نیاز به داده‌های برچسب‌گذاری شده است که می‌تواند هزینه و زمان لازم برای برچسب‌گذاری داده‌ها را به‌طور قابل توجهی کاهش دهد.

Whisper

- مشخصات: OpenAI توسط Whisper توسعه داده شده و به عنوان یک مدل ترانسفورمر بزرگ برای بازشناسی گفتار استفاده می‌شود.
- الگوریتم‌ها: Whisper از یک مدل ترانسفورمر بزرگ با تعداد لایه‌های عمیق استفاده می‌کند که قادر به پردازش طولانی‌مدت ویژگی‌های صوتی است.
- مقایسه: در مقایسه با Wav2Vec دارای مدل‌های بزرگتری است و برای دقت بالاتر نیاز به داده‌های بیشتر و توان محاسباتی بیشتری دارد.

2. حوزه تبدیل متن به گفتار

تبدیل متن به گفتار به فرآیند تبدیل متن نوشته شده به گفتار مصنوعی گفته می‌شود. در این حوزه دو پروژه برجسته شامل Tacotron و Vall-E مورد بررسی قرار می‌گیرند.

Tacotron

- مشخصات: Tacotron توسط Google توسعه داده شده است و یک سیستم تولید گفتار مبتنی بر شبکه عصبی است که متن را به شکل موج صوتی تبدیل می‌کند.
- الگوریتم‌ها: از یک معماری شبکه عصبی بازگشتی و ترانسفورمر برای تبدیل متن به ویژگی‌های گفتاری و سپس ویژگی‌ها به شکل موج صوتی استفاده می‌کند.
- مقایسه: Tacotron به دلیل معماری ساده‌تر و سرعت پردازش مناسب‌تر می‌تواند گفتار با کیفیت بالا در زمان واقعی تولید کند.

Vall-E

- مشخصات: Vall-E توسط تیم تحقیقاتی Microsoft توسعه داده شده است و از معماری ترانسفورمر برای تولید گفتار استفاده می‌کند.
- الگوریتم‌ها: Vall-E از مدل ترانسفورمر برای تبدیل متن به ویژگی‌های گفتاری استفاده می‌کند و سپس این ویژگی‌ها را به شکل موج صوتی تبدیل می‌کند.
- مقایسه: Vall-E در مقایسه با Tacotron به دلیل استفاده از معماری ترانسفورمر قابلیت تولید گفتار با کیفیت بالاتر و طبیعی‌تر را دارد اما نیاز به توان محاسباتی بیشتری دارد.

3. حوزه بهسازی گفتار

بهسازی گفتار به فرآیند بهبود کیفیت و وضوح گفتار به خصوص در محیط‌های نویزی گفته می‌شود. در این حوزه دو پروژه برجسته شامل DeepSpeech و SEGAN مورد بررسی قرار می‌گیرند.

a. SEGAN (Speech Enhancement Generative Adversarial Network)

- مشخصات: SEGAN توسط Facebook AI Research توسعه داده شده است و از شبکه‌های GAN برای بهبود کیفیت گفتار استفاده می‌کند.
- الگوریتم‌ها: SEGAN از یک Generator و یک شبکه Discriminator برای تولید گفتار بهبود یافته از ورودی‌های نویزی استفاده می‌کند.
- مقایسه: یکی از مزیت‌های اصلی SEGAN استفاده از GAN‌ها است که می‌تواند گفتار بهبود یافته با کیفیت بالا و طبیعی‌تری تولید کند.

DeepSpeech

- مشخصات: Mozilla DeepSpeech توسط هدف آن بهبود بازشناسی گفتار با استفاده از معماری‌های عمیق یادگیری است.
- الگوریتم‌ها: DeepSpeech از معماری‌های شبکه‌های عصبی عمیق برای تبدیل گفتار نویزی به گفتار بهبود یافته و متن استفاده می‌کند.
- مقایسه: در مقایسه با SEGAN بیشتر بر بازشناسی گفتار تمرکز دارد و کمتر به بهسازی گفتار به عنوان یک هدف مستقل می‌پردازد اما توانایی‌های مشابهی در بهبود کیفیت گفتار دارد.

سوال دو

ابتدا، برای دسترسی به داده‌های Kaggle API AudioMNIST، باید از داده‌های استفاده کنیم. برای این کار، باید تنظیمات اولیه را انجام دهیم و دادگان را دانلود کنیم.

```

!pip install -q kaggle

!mkdir -p ~/.kaggle
!cp /content/kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

cp: cannot stat '/content/kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory

! kaggle datasets download sripaadsrinivasan/audio-mnist

Dataset URL: https://www.kaggle.com/datasets/sripaadsrinivasan/audio-mnist
License(s): CC0-1.0
Downloading audio-mnist.zip to /content
99% 942M/948M [00:11<00:00, 62.6MB/s]
100% 948M/948M [00:11<00:00, 89.7MB/s]

! unzip /content/audio-mnist.zip

```

پس از دانلود دادگان و استخراج فایل‌ها، دادگان را از پوشه مربوطه بارگذاری می‌کنیم. دادگان شامل 60 پوشه است که هر پوشه مربوط به یک گوینده است و هر فایل صوتی در این پوشه‌ها صدای یک عدد است. ما همه فایل‌های صوتی را بارگذاری می‌کنیم و به دو قسمت آموزش و آزمون تقسیم می‌کنیم.

```

# Path to the dataset
dataset_path = '/content/data'

# Load dataset
def load_audio_files(dataset_path):
    X = []
    y = []
    for folder in tqdm(os.listdir(dataset_path), desc='Loading dataset'): # Add tqdm here
        folder_path = os.path.join(dataset_path, folder)
        if not os.path.isdir(folder_path): # Check if it's a directory
            continue
        for file in os.listdir(folder_path):
            if file.endswith('.wav'):
                file_path = os.path.join(folder_path, file)
                audio, sr = librosa.load(file_path, sr=16000)
                X.append(audio)
                label = int(file.split('_')[0])
                y.append(label)
    return X, y

# Load the dataset
X, y = load_audio_files(dataset_path)

# Split dataset into train/test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Output the sizes of the splits
print(f"Training set size: {len(X_train)}")
print(f"Test set size: {len(X_test)}")

```

برای پیش‌پردازش داده‌ها از ویژگی‌های MFCC استفاده کردیم. MFCC ویژگی‌های مفیدی از سیگنال صوتی استخراج می‌کند که به مدل کمک می‌کند صدای را بهتر تشخیص دهد. ما برای هر فایل صوتی MFCC را با طول فریم 25 میلی‌ثانیه و 24 فیلتر مل استخراج کردیم و سپس مشتق‌های مرتبه یک و دو آن‌ها را نیز محاسبه کردیم.

```
def extract_mfcc_features(audio, sr, n_mfcc=12, frame_length=0.025, hop_length=0.01):
    mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=n_mfcc, n_fft=int(frame_length*samplerate), n_mels=24)
    mfcc_delta = librosa.feature.delta(mfcc)
    mfcc_delta2 = librosa.feature.delta(mfcc, order=2)
    combined = np.vstack([mfcc, mfcc_delta, mfcc_delta2])
    return combined.flatten()

# Extract MFCC features for the dataset
X_train_mfcc = [extract_mfcc_features(x, 16000) for x in X_train]
X_test_mfcc = [extract_mfcc_features(x, 16000) for x in X_test]

# Padding the MFCC features to have the same length
def pad_sequences(sequences, maxlen):
    return np.array([np.pad(seq, (0, maxlen - len(seq)), mode='constant') if len(seq) < maxlen else seq[:maxlen] for seq in sequences])

maxlen = max(max(len(x) for x in X_train_mfcc), max(len(x) for x in X_test_mfcc))
X_train_mfcc = pad_sequences(X_train_mfcc, maxlen)
X_test_mfcc = pad_sequences(X_test_mfcc, maxlen)

# Convert to numpy arrays
X_train_mfcc = np.array(X_train_mfcc)
X_test_mfcc = np.array(X_test_mfcc)
y_train = np.array(y_train)
y_test = np.array(y_test)

print("MFCC feature extraction complete.")

[6]
...
MFCC feature extraction complete.
```

شبکه عصبی پرسپترون چندلایه (MLP) را به صورت دستی پیاده‌سازی کردیم. تعداد نرون‌های ورودی بر اساس ویژگی‌های استخراج شده از فایل‌های صوتی و تعداد نرون‌های خروجی برابر با 10 (تعداد کلاس‌ها) است. لایه میانی (مخفي) تعداد نرون‌هاییش به اندازه میانگین تعداد نرون‌های ورودی و خروجی است.

ساختار شبکه ما به این صورت است:

لایه ورودی: تعداد نرون‌های این لایه برابر با تعداد ویژگی‌های استخراج شده از فایل‌های صوتی (MFCC) است.

لایه مخفی: تعداد نرون‌های این لایه برابر با میانگین تعداد نرون‌های ورودی و خروجی است.

لایه خروجی: تعداد نرون‌های این لایه برابر با تعداد کلاس‌ها (10 کلاس برای اعداد ۰ تا ۹) است.

تابع فعال‌سازی

برای لایه‌های میانی و خروجی از تابع فعال‌سازی سیگموید دو قطبی (Bipolar Sigmoid) استفاده کردیم:

```
def bipolar_sigmoid(x):
    return (2 / (1 + np.exp(-x))) - 1
```

مشتق این تابع به صورت زیر است:

```
def bipolar_sigmoid_derivative(x):
    return 0.5 * (1 + bipolar_sigmoid(x)) * (1 - bipolar_sigmoid(x))
```

تابع خطأ

برای ارزیابی عملکرد مدل از خطای میانگین مربعات (MSE) استفاده کردیم:

```
# Mean Squared Error
def mse(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)
```

توضیح متدهای کلاس MLP

__init__: این متدهایی برای مقداردهی اولیه شبکه عصبی استفاده می‌شود. در این متدها وزن‌ها و بایاس‌ها به صورت تصادفی مقداردهی می‌شوند.

forward: این متدهایی برای اجرای مرحله پیشرو (forward pass) شبکه عصبی استفاده می‌شود. ابتدا ورودی‌ها به لایه مخفی منتقل می‌شوند و خروجی لایه مخفی محاسبه می‌شود. سپس خروجی لایه مخفی به لایه خروجی منتقل شده و خروجی نهایی محاسبه می‌شود.

:backward این متده برای اجرای مرحله پس انتشار خطا (backward pass) استفاده می‌شود. خطای خروجی محاسبه شده و سپس دلتاهای لایه‌های خروجی و مخفی محاسبه می‌شوند. وزن‌ها و بایاس‌ها بر اساس این دلتاهای بهروزرسانی می‌شوند.

:train این متده برای آموزش شبکه عصبی استفاده می‌شود. برای هر نمونه در داده‌های آموزشی، مرحله پیشرو و پس انتشار خطا اجرا می‌شود. خطای میانگین مربعات (MSE) و دقت در هر دوره (epoch) محاسبه و نمایش داده می‌شوند.

```
def train(self, X, y, epochs=30):
    mse_errors = []
    accuracies = []

    for epoch in range(epochs):
        total_error = 0
        correct_predictions = 0
        class_correct = np.zeros(self.output_size)
        class_total = np.zeros(self.output_size)

        for i in range(len(X)):
            X_sample = X[i].reshape(1, -1)
            y_sample = y[i].reshape(1, -1)

            output = self.forward(X_sample)

            self.backward(X_sample, y_sample, output)

            # Calculate accuracy
            predicted_class = np.argmax(softmax(output))
            true_class = np.argmax(y_sample)
            if predicted_class == true_class:
                correct_predictions += 1
                class_correct[predicted_class] += 1
            class_total[true_class] += 1

            total_error += mse(y_sample, output)

        avg_error = total_error / len(X)
        accuracy = correct_predictions / len(X)
        mse_errors.append(avg_error)
        accuracies.append(accuracy)
        print(f"Epoch {epoch+1}/{epochs}, Avg Error: {avg_error:.6f}, Accuracy: {accuracy:.6f}")

    return mse_errors, accuracies
```

:predict این متده برای پیش‌بینی خروجی شبکه عصبی استفاده می‌شود. خروجی شبکه محاسبه شده و کلاس با بیشترین مقدار خروجی به عنوان پیش‌بینی شبکه برگردانده

می‌شود.

```
# Predict on test data
y_pred = mlp.predict(X_test_mfcc)

# Calculate test accuracy
test_accuracy = np.mean(y_pred == y_test)
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')

# Calculate accuracy for each class on test data
class_correct = np.zeros(output_size)
class_total = np.zeros(output_size)

for i in range(len(y_test)):
    true_class = y_test[i]
    predicted_class = y_pred[i]
    if true_class == predicted_class:
        class_correct[true_class] += 1
    class_total[true_class] += 1

for i in range(output_size):
    class_accuracy = class_correct[i] / class_total[i] if class_total[i] > 0 else 0
    print(f'Test Class {i} Accuracy: {class_accuracy * 100:.2f}%')
```

مدل MLP را با استفاده از دادگان آموزش تمرین می‌دهیم. از الگوریتم پس انتشار خطا برای بهروزرسانی وزن‌ها و بایاس‌ها استفاده می‌کنیم.

خروجی در حین آموزش :

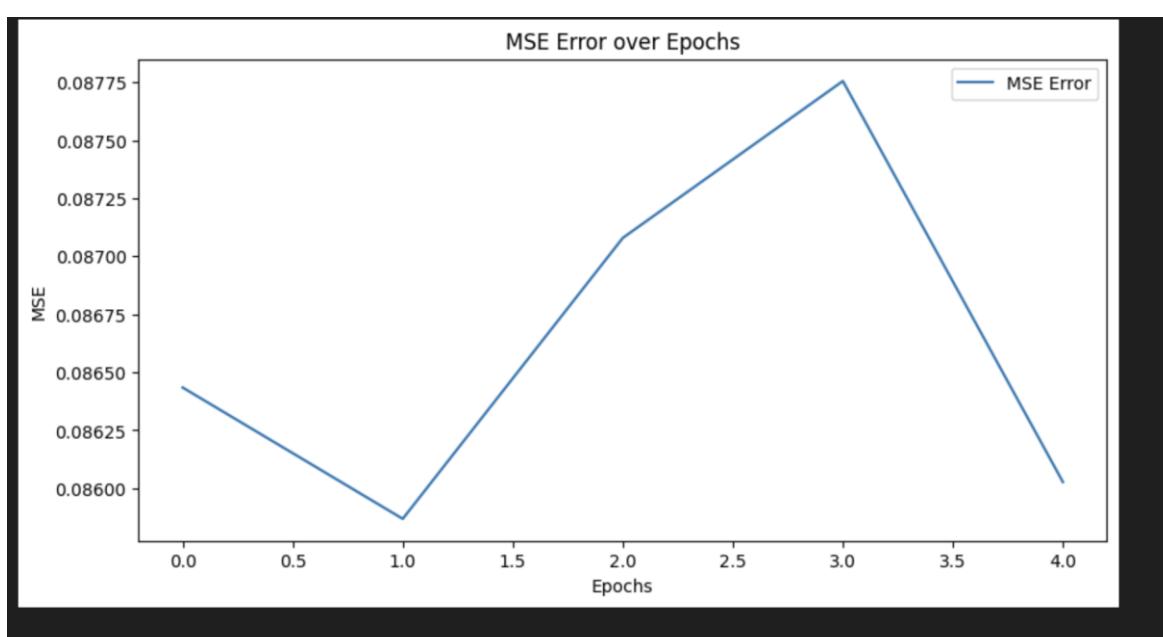
```
Epoch 1/5, Avg Error: 0.086434, Accuracy: 0.245667
Class 0 Accuracy: 0.173345
Class 1 Accuracy: 0.253399
Class 2 Accuracy: 0.251575
Class 3 Accuracy: 0.129357
Class 4 Accuracy: 0.147412
Class 5 Accuracy: 0.095594
Class 6 Accuracy: 0.469422
Class 7 Accuracy: 0.290108
Class 8 Accuracy: 0.444812
Class 9 Accuracy: 0.202229
Epoch 2/5, Avg Error: 0.085869, Accuracy: 0.258375
Class 0 Accuracy: 0.182623
Class 1 Accuracy: 0.248043
Class 2 Accuracy: 0.215876
Class 3 Accuracy: 0.119278
Class 4 Accuracy: 0.173913
Class 5 Accuracy: 0.106816
Class 6 Accuracy: 0.563475
Class 7 Accuracy: 0.277224
Class 8 Accuracy: 0.479537
Class 9 Accuracy: 0.217912
```

```
Epoch 3/5, Avg Error: 0.087079, Accuracy: 0.233542
Class 0 Accuracy: 0.157318
Class 1 Accuracy: 0.293366
Class 2 Accuracy: 0.271315
Class 3 Accuracy: 0.107098
Class 4 Accuracy: 0.146998
Class 5 Accuracy: 0.105985
Class 6 Accuracy: 0.433572
Class 7 Accuracy: 0.267664
Class 8 Accuracy: 0.360066
Class 9 Accuracy: 0.192324
Epoch 4/5, Avg Error: 0.087755, Accuracy: 0.219375
Class 0 Accuracy: 0.144243
Class 1 Accuracy: 0.274001
Class 2 Accuracy: 0.271315
Class 3 Accuracy: 0.101638
Class 4 Accuracy: 0.141615
Class 5 Accuracy: 0.108479
Class 6 Accuracy: 0.396035
Class 7 Accuracy: 0.286783
Class 8 Accuracy: 0.337743
Class 9 Accuracy: 0.132480
```

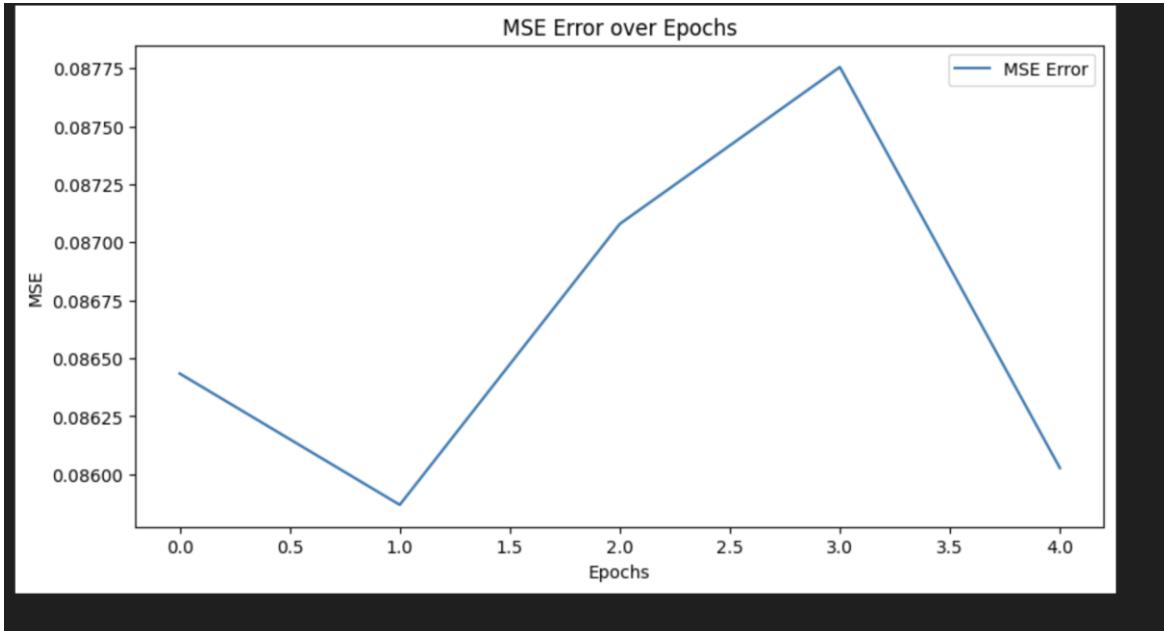
```
Epoch 5/5, Avg Error: 0.086027, Accuracy: 0.258875
Class 0 Accuracy: 0.184732
Class 1 Accuracy: 0.323033
Class 2 Accuracy: 0.312054
Class 3 Accuracy: 0.128937
Class 4 Accuracy: 0.185507
Class 5 Accuracy: 0.105569
Class 6 Accuracy: 0.473218
Class 7 Accuracy: 0.260183
Class 8 Accuracy: 0.464655
Class 9 Accuracy: 0.151465
```

: نمودار ها

خطا در حین اموزش:



دقت در حین آموزش :



بعد از آموزش مدل، آن را روی دادگان آزمون تست می کنیم و دقت مدل را برای هر کلاس محاسبه می کنیم.

```
# Predict on test data
y_pred = mlp.predict(X_test_mfcc)

# Calculate test accuracy
test_accuracy = np.mean(y_pred == y_test)
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')

# Calculate accuracy for each class on test data
class_correct = np.zeros(output_size)
class_total = np.zeros(output_size)

for i in range(len(y_test)):
    true_class = y_test[i]
    predicted_class = y_pred[i]
    if true_class == predicted_class:
        class_correct[true_class] += 1
    class_total[true_class] += 1

for i in range(output_size):
    class_accuracy = class_correct[i] / class_total[i] if class_total[i] > 0 else 0
    print(f'Test Class {i} Accuracy: {class_accuracy * 100:.2f}%')
```

دقت تست :

```
Test Accuracy: 40.38%
Test Class 0 Accuracy: 66.30%
Test Class 1 Accuracy: 47.29%
Test Class 2 Accuracy: 74.64%
Test Class 3 Accuracy: 0.00%
Test Class 4 Accuracy: 0.00%
Test Class 5 Accuracy: 0.00%
Test Class 6 Accuracy: 79.81%
Test Class 7 Accuracy: 0.00%
Test Class 8 Accuracy: 82.96%
Test Class 9 Accuracy: 50.09%
```

حالا اگر به جای این خروار فیچر بیایم mfcc mean استفاده کنیم فقط متدهای مون عوض میشه :

```
def extract_mfcc_features(audio, sr, n_mfcc=12, frame_length=0.025, hop_length=0.01):
    mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=n_mfcc, n_fft=int(frame_length*sr), n_mels=24)
    mean = np.mean(mfcc, axis=1)
    return mean

# Extract MFCC features for the dataset
X_train_mfcc = [extract_mfcc_features(x, 16000) for x in X_train]
X_test_mfcc = [extract_mfcc_features(x, 16000) for x in X_test]

# Convert to numpy arrays
X_train_mfcc = np.array(X_train_mfcc)
X_test_mfcc = np.array(X_test_mfcc)
y_train = np.array(y_train)
y_test = np.array(y_test)

print("MFCC feature extraction complete.")
```

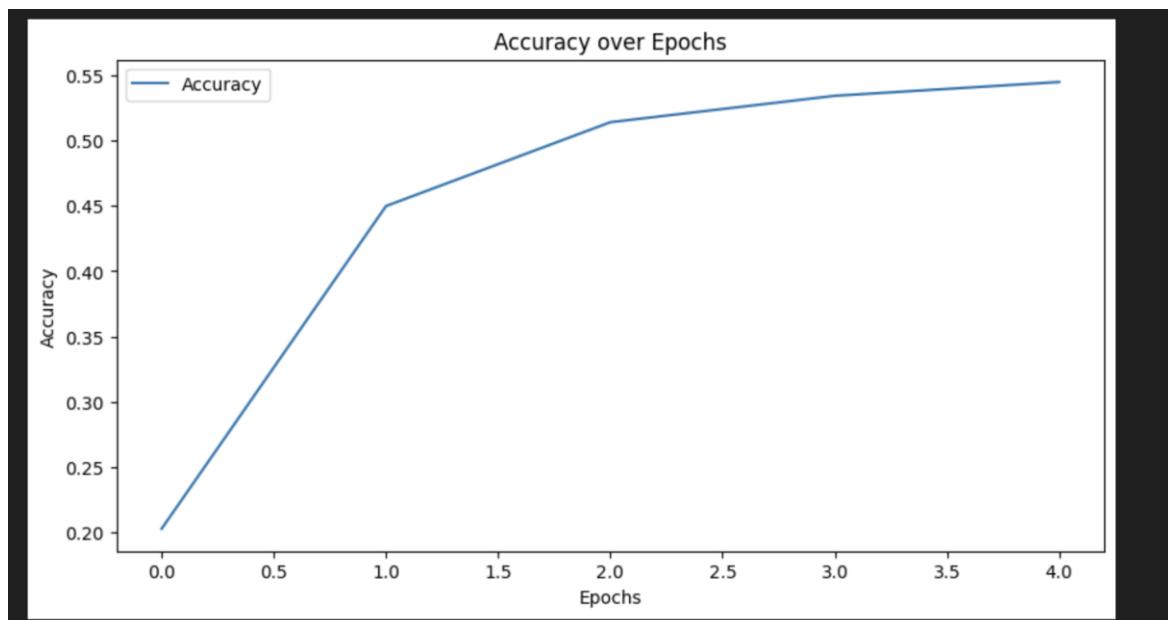
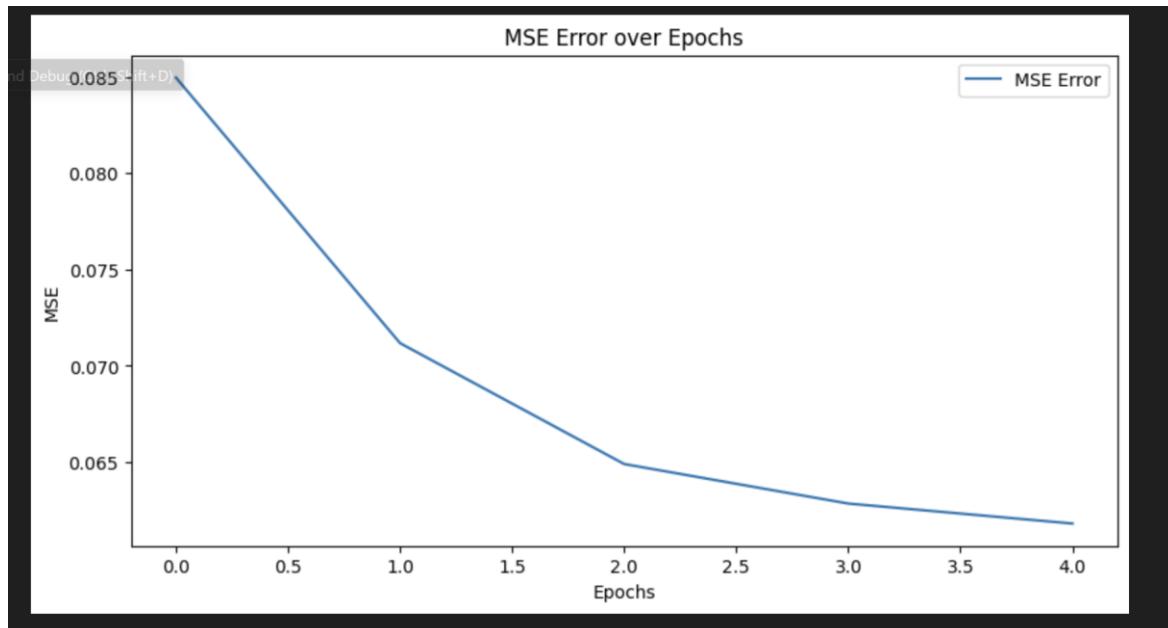
دیگ پدینگ هم نمیخوایم خب حالا شبکه قبلی رو اموزش میدهیم نتایج :

دقت و خطأ در حين اموزش :

```
Epoch 1/5, Avg Error: 0.084970, Accuracy: 0.202792
Class 0 Accuracy: 0.035850
Class 1 Accuracy: 0.254223
Class 2 Accuracy: 0.047879
Class 3 Accuracy: 0.083578
Class 4 Accuracy: 0.465010
Class 5 Accuracy: 0.154198
Class 6 Accuracy: 0.529734
Class 7 Accuracy: 0.022028
Class 8 Accuracy: 0.250930
Class 9 Accuracy: 0.182418
Epoch 2/5, Avg Error: 0.071180, Accuracy: 0.449833
Class 0 Accuracy: 0.055251
Class 1 Accuracy: 0.604038
Class 2 Accuracy: 0.082318
Class 3 Accuracy: 0.087358
Class 4 Accuracy: 0.749482
Class 5 Accuracy: 0.704904
Class 6 Accuracy: 0.953184
Class 7 Accuracy: 0.326268
Class 8 Accuracy: 0.850765
Class 9 Accuracy: 0.076764
Epoch 3/5, Avg Error: 0.064903, Accuracy: 0.514042
Class 0 Accuracy: 0.040911
Class 1 Accuracy: 0.587557
Class 2 Accuracy: 0.012600
Class 3 Accuracy: 0.041159
Class 4 Accuracy: 0.783023
Class 5 Accuracy: 0.744805
Class 6 Accuracy: 0.941797
```

```
Epoch 4/5, Avg Error: 0.062850, Accuracy: 0.534250
Class 0 Accuracy: 0.040489
Class 1 Accuracy: 0.544705
Class 2 Accuracy: 0.003360
Class 3 Accuracy: 0.033599
Class 4 Accuracy: 0.803313
Class 5 Accuracy: 0.766002
Class 6 Accuracy: 0.936314
Class 7 Accuracy: 0.731089
Class 8 Accuracy: 0.925589
Class 9 Accuracy: 0.542716
Epoch 5/5, Avg Error: 0.061813, Accuracy: 0.544792
Class 0 Accuracy: 0.043863
Class 1 Accuracy: 0.554182
Class 2 Accuracy: 0.003360
Class 3 Accuracy: 0.023940
Class 4 Accuracy: 0.817391
Class 5 Accuracy: 0.791355
Class 6 Accuracy: 0.942640
Class 7 Accuracy: 0.729842
Class 8 Accuracy: 0.935924
Class 9 Accuracy: 0.589352
```

: نمودار ها



دقت تست :

```
Test Accuracy: 55.22%
Test Class 0 Accuracy: 4.93%
Test Class 1 Accuracy: 53.23%
Test Class 2 Accuracy: 0.00%
Test Class 3 Accuracy: 0.48%
Test Class 4 Accuracy: 80.34%
Test Class 5 Accuracy: 78.28%
Test Class 6 Accuracy: 94.44%
Test Class 7 Accuracy: 60.94%
Test Class 8 Accuracy: 94.49%
Test Class 9 Accuracy: 92.55%
```

دیدیم وقتی که از mfcc mean استفاده میکنیم ترین بهتری خواهیم داشت

علاوه بر میانگین‌گیری از بردارهای ویژگی‌های MFCC برای هر فایل صوتی، می‌توان از روش‌های دیگری نیز برای ترکیب و نمایندگی بهتر ویژگی‌ها استفاده کرد مثلا:

1. استفاده از آماره‌های دیگر

به جای میانگین‌گیری، می‌توان از دیگر آماره‌ها مانند واریانس، میانه، مینیمم و ماکزیمم استفاده کرد. این کار می‌تواند اطلاعات بیشتری از توزیع ویژگی‌ها در طول فایل صوتی فراهم کند.

2. استفاده از PCA

استفاده از PCA می‌تواند به کاهش ابعاد ویژگی‌ها و حفظ مهم‌ترین مؤلفه‌ها کمک کند. این کار می‌تواند به کاهش نویز و اطلاعات غیرضروری کمک کند.

3. استفاده از شبکه‌های عصبی CNN

شبکه‌های عصبی کانولوشنی می‌توانند ویژگی‌های محلی و الگوهای زمانی را بهتر از روش‌های مبتنی بر میانگین‌گیری یا آماره‌ها کشف کنند.

4. استفاده از مدل‌های ترکیبی

می‌توانیم از مدل‌های ترکیبی که شامل CNN‌ها برای استخراج ویژگی‌ها و MLP‌ها برای دسته‌بندی هستند استفاده کنیم. این ترکیب می‌تواند به بهبود دقیق مدل کمک کند.

5. استفاده از توالی مدل‌ها مانند LSTM یا GRU

شبکه‌های عصبی بازگشتی مانند LSTM یا GRU برای مدل‌سازی داده‌های ترتیبی مناسب هستند و می‌توانند روابط زمانی پیچیده را بهتر از CNN‌ها و MLP‌ها کشف کنند.

کد‌های این قسمت در پوشش one_layer موجود است

قسمت بعدی :

در قسمت بعدی باید تعداد لایه‌های میانی را بیشتر کنیم که فقط کلاس mlp را عوض کنیم

ابتدا همه لایه‌ها را با مقادیر تصادفی مقدار دهی می‌کنیم

```
class MLP:
    def __init__(self, input_size, hidden_size, output_size, learning_rate=0.001):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.learning_rate = learning_rate

        # وزن‌ها
        self.W1 = np.random.randn(self.input_size, self.hidden_size)
        self.W2 = np.random.randn(self.hidden_size, self.hidden_size)
        self.W3 = np.random.randn(self.hidden_size, self.output_size)
        self.b1 = np.zeros((1, self.hidden_size))
        self.b2 = np.zeros((1, self.hidden_size))
        self.b3 = np.zeros((1, self.output_size))
```

حالا از ورودی یک بار میریم جلو تا به خروجی بررسیم فقط اینبار دو تا لایه مخفی داریم:

```
def forward(self, x):
    self.z1 = np.dot(x, self.W1) + self.b1
    self.a1 = self.bipolar_sigmoid(self.z1)
    self.z2 = np.dot(self.a1, self.W2) + self.b2
    self.a2 = self.bipolar_sigmoid(self.z2)
    self.z3 = np.dot(self.a2, self.W3) + self.b3
    self.a3 = self.bipolar_sigmoid(self.z3)
    return self.a3
```

حالا برای اپدیت وزن ها باید از خروجی بیایم عقب :

```
def backward(self, x, y, output):
    m = y.shape[0]
    delta3 = output - y
    dW3 = np.dot(self.a2.T, delta3) / m
    db3 = np.sum(delta3, axis=0, keepdims=True) / m

    delta2 = np.dot(delta3, self.W3.T) * self.bipolar_sigmoid_derivative(self.a2)
    dW2 = np.dot(self.a1.T, delta2) / m
    db2 = np.sum(delta2, axis=0, keepdims=True) / m

    delta1 = np.dot(delta2, self.W2.T) * self.bipolar_sigmoid_derivative(self.a1)
    dW1 = np.dot(x.T, delta1) / m
    db1 = np.sum(delta1, axis=0, keepdims=True) / m

    self.W1 -= self.learning_rate * dW1
    self.W2 -= self.learning_rate * dW2
    self.W3 -= self.learning_rate * dW3
    self.b1 -= self.learning_rate * db1
    self.b2 -= self.learning_rate * db2
    self.b3 -= self.learning_rate * db3
```

فرایند train دقیقا عین گذشتش :

```
def train(self, X, y, epochs=30):
    mse_errors = []
    accuracies = []

    for epoch in range(epochs):
        total_error = 0
        correct_predictions = 0
        class_correct = np.zeros(self.output_size)
        class_total = np.zeros(self.output_size)

        for i in range(len(X)):
            X_sample = X[i].reshape(1, -1)
            y_sample = y[i].reshape(1, -1)

            output = self.forward(X_sample)

            self.backward(X_sample, y_sample, output)

            # Calculate accuracy
            predicted_class = np.argmax(self.softmax(output))
            true_class = np.argmax(y_sample)
            if predicted_class == true_class:
                correct_predictions += 1
                class_correct[true_class] += 1
            class_total[true_class] += 1

            total_error += self.mse(y_sample, output)

        avg_error = total_error / len(X)
        accuracy = correct_predictions / len(X)
        mse_errors.append(avg_error)
        accuracies.append(accuracy)
        print(f"Epoch {epoch+1}/{epochs}, Avg Error: {avg_error:.6f}, Accuracy: {accuracy:.6f}")
```

خروجی در حین آموزش :

```
Epoch 1/5, Avg Error: 0.094799, Accuracy: 0.098333
Class 0 Accuracy: 0.098693
Class 1 Accuracy: 0.090235
Class 2 Accuracy: 0.052919
Class 3 Accuracy: 0.144477
Class 4 Accuracy: 0.068737
Class 5 Accuracy: 0.118869
Class 6 Accuracy: 0.130325
Class 7 Accuracy: 0.025353
Class 8 Accuracy: 0.110790
Class 9 Accuracy: 0.143211
Epoch 2/5, Avg Error: 0.090456, Accuracy: 0.097042
Class 0 Accuracy: 0.101645
Class 1 Accuracy: 0.100536
Class 2 Accuracy: 0.058379
Class 3 Accuracy: 0.145737
Class 4 Accuracy: 0.073292
Class 5 Accuracy: 0.071904
Class 6 Accuracy: 0.130747
Class 7 Accuracy: 0.026600
Class 8 Accuracy: 0.114924
Class 9 Accuracy: 0.146925
Epoch 3/5, Avg Error: 0.090447, Accuracy: 0.096500
Class 0 Accuracy: 0.100801
Class 1 Accuracy: 0.100124
...
Class 6 Accuracy: 0.129481
Class 7 Accuracy: 0.029094
Class 8 Accuracy: 0.116990
Class 9 Accuracy: 0.147338
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

با توجه به خروجی افتضاحمون من ی سرچی کردم گفته بودند مقدار دهی اولیه وزن ها را اگر تغیر دهیم و داده ها را اگر نرمال کنیم بهتر میشه پس :

```
# Xavier initialization for weights
self.W1 = np.random.randn(self.input_size, self.hidden_size) * np.sqrt(1. / self.input_size)
self.W2 = np.random.randn(self.hidden_size, self.hidden_size) * np.sqrt(1. / self.hidden_size)
self.W3 = np.random.randn(self.hidden_size, self.output_size) * np.sqrt(1. / self.hidden_size)
self.b1 = np.zeros((1, self.hidden_size))
self.b2 = np.zeros((1, self.hidden_size))
self.b3 = np.zeros((1, self.output_size))
```

۶

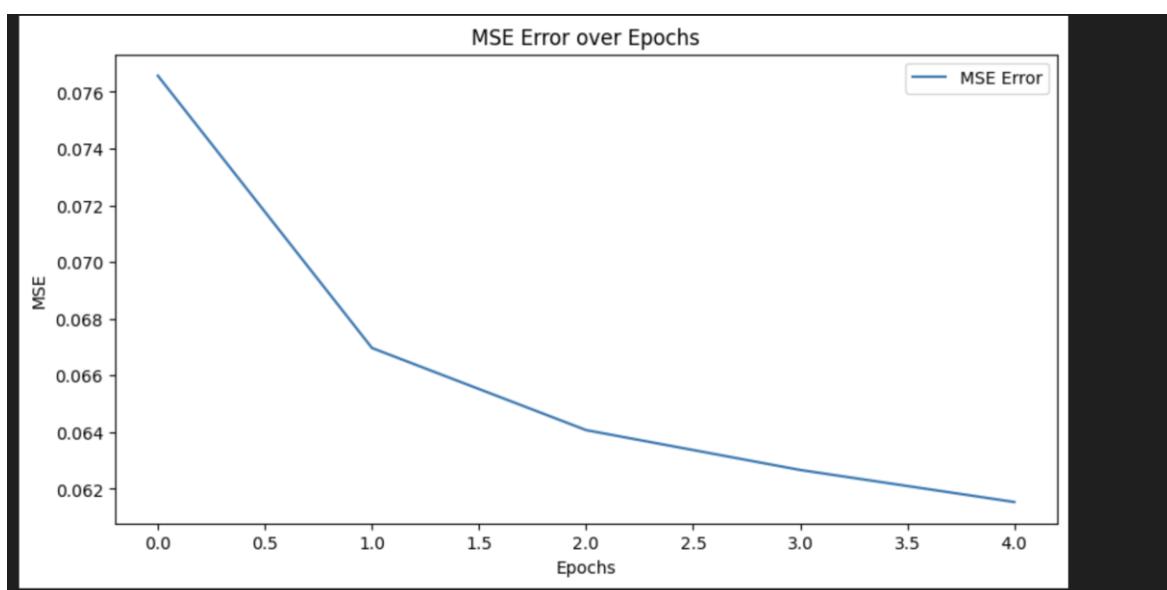
```
# Ensure the data is normalized
X_train_mfcc = (X_train_mfcc - np.mean(X_train_mfcc, axis=0)) / np.std(X_train_mfcc, axis=0)
X_test_mfcc = (X_test_mfcc - np.mean(X_test_mfcc, axis=0)) / np.std(X_test_mfcc, axis=0)
```

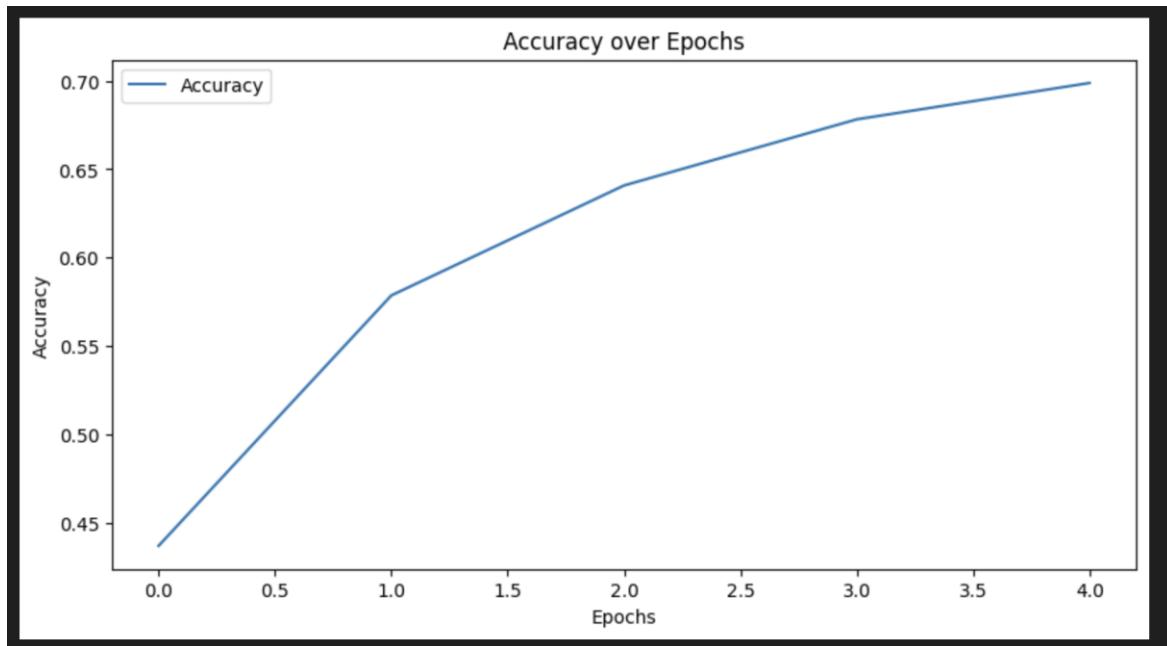
حال خروجی زیبا میشود :

```
Epoch 1/5, Avg Error: 0.076570, Accuracy: 0.437167
Class 0 Accuracy: 0.090679
Class 1 Accuracy: 0.214256
Class 2 Accuracy: 0.344393
Class 3 Accuracy: 0.434271
Class 4 Accuracy: 0.778468
Class 5 Accuracy: 0.539069
Class 6 Accuracy: 0.878954
Class 7 Accuracy: 0.293017
Class 8 Accuracy: 0.775940
Class 9 Accuracy: 0.024763
Epoch 2/5, Avg Error: 0.066970, Accuracy: 0.578708
Class 0 Accuracy: 0.109237
Class 1 Accuracy: 0.459415
Class 2 Accuracy: 0.528769
Class 3 Accuracy: 0.648047
Class 4 Accuracy: 0.921325
Class 5 Accuracy: 0.881546
Class 6 Accuracy: 0.980177
Class 7 Accuracy: 0.185786
Class 8 Accuracy: 0.875982
Class 9 Accuracy: 0.196863
Epoch 3/5, Avg Error: 0.064074, Accuracy: 0.640875
Class 0 Accuracy: 0.239561
Class 1 Accuracy: 0.479192
...
Class 6 Accuracy: 0.976381
Class 7 Accuracy: 0.529094
Class 8 Accuracy: 0.913187
Class 9 Accuracy: 0.595543
```

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

نمودار ها :





دقت تست :

```
Test Accuracy: 69.98%
Test Class 0 Accuracy: 52.78%
Test Class 1 Accuracy: 48.34%
Test Class 2 Accuracy: 42.00%
Test Class 3 Accuracy: 67.69%
Test Class 4 Accuracy: 93.50%
Test Class 5 Accuracy: 89.06%
Test Class 6 Accuracy: 97.93%
Test Class 7 Accuracy: 59.43%
Test Class 8 Accuracy: 91.39%
Test Class 9 Accuracy: 58.06%
```

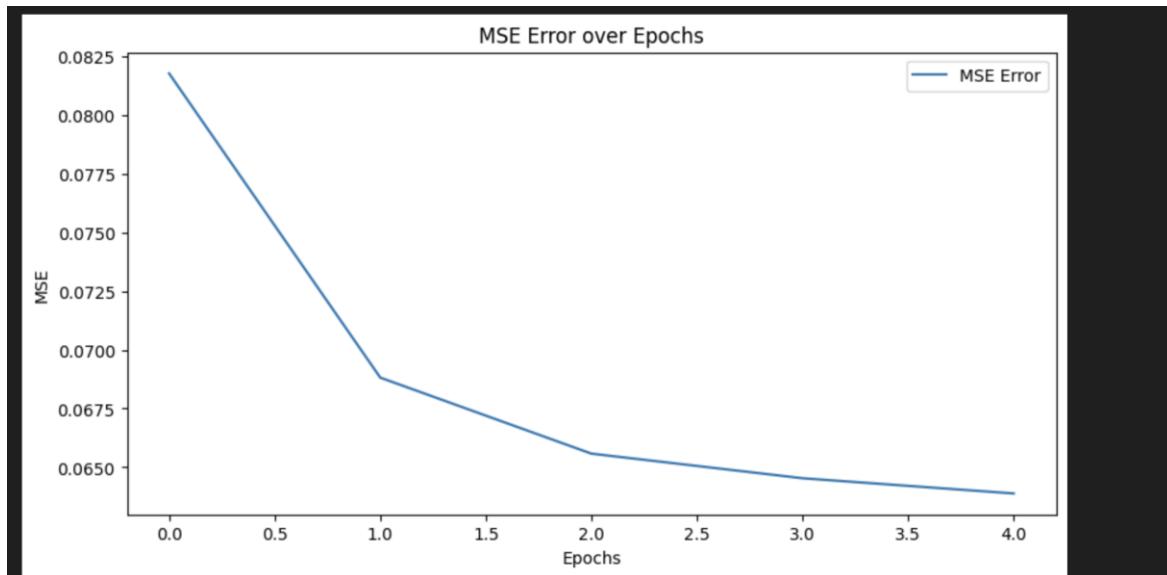
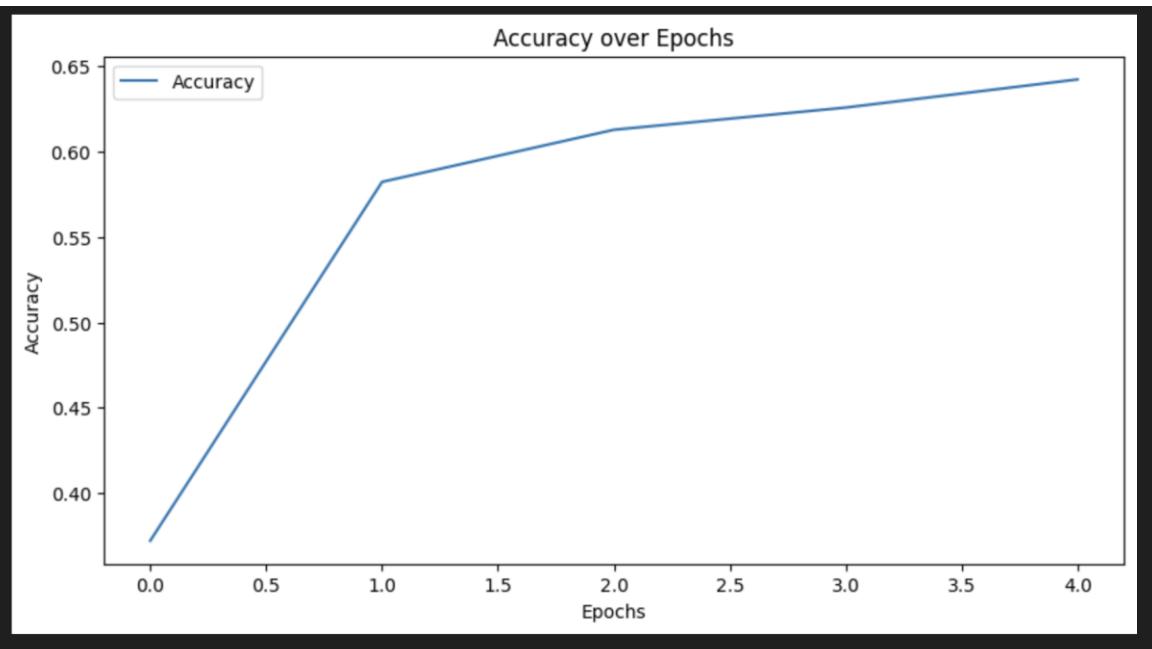
خب برای قسمت بعدی گفته تعداد لایه های هیدن ما باید :

```
input_size = x_train.shape[1]
hidden_size1 = int(2 * input_size / 3)
hidden_size2 = int(input_size / 2)
output_size = 10
```

پس تنها تغیری که داریم اینه که اینا رو ورودی متد میکنیم و حالا خروجی ها :

```
Epoch 1/5, Avg Error: 0.081776, Accuracy: 0.372208
Class 0 Accuracy: 0.025728
Class 1 Accuracy: 0.257520
Class 2 Accuracy: 0.363293
Class 3 Accuracy: 0.441831
Class 4 Accuracy: 0.378468
Class 5 Accuracy: 0.697007
Class 6 Accuracy: 0.625053
Class 7 Accuracy: 0.013716
Class 8 Accuracy: 0.664324
Class 9 Accuracy: 0.254643
Epoch 2/5, Avg Error: 0.068819, Accuracy: 0.582208
Class 0 Accuracy: 0.070856
Class 1 Accuracy: 0.618047
Class 2 Accuracy: 0.478790
Class 3 Accuracy: 0.657707
Class 4 Accuracy: 0.881159
Class 5 Accuracy: 0.869909
Class 6 Accuracy: 0.982708
Class 7 Accuracy: 0.000000
Class 8 Accuracy: 0.830922
Class 9 Accuracy: 0.428395
Epoch 3/5, Avg Error: 0.065581, Accuracy: 0.612708
Class 0 Accuracy: 0.133277
Class 1 Accuracy: 0.543469
...
Class 6 Accuracy: 0.979755
Class 7 Accuracy: 0.123441
Class 8 Accuracy: 0.915668
Class 9 Accuracy: 0.596368
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

: نمودار ها



دقت تست :

```
Test Accuracy: 64.93%
Test Class 0 Accuracy: 39.90%
Test Class 1 Accuracy: 53.93%
Test Class 2 Accuracy: 44.43%
Test Class 3 Accuracy: 59.45%
Test Class 4 Accuracy: 94.87%
Test Class 5 Accuracy: 88.05%
Test Class 6 Accuracy: 98.25%
Test Class 7 Accuracy: 21.72%
Test Class 8 Accuracy: 91.91%
Test Class 9 Accuracy: 57.89%
```

برای قسمت سوم قست ب تعداد لایه های هیدن :

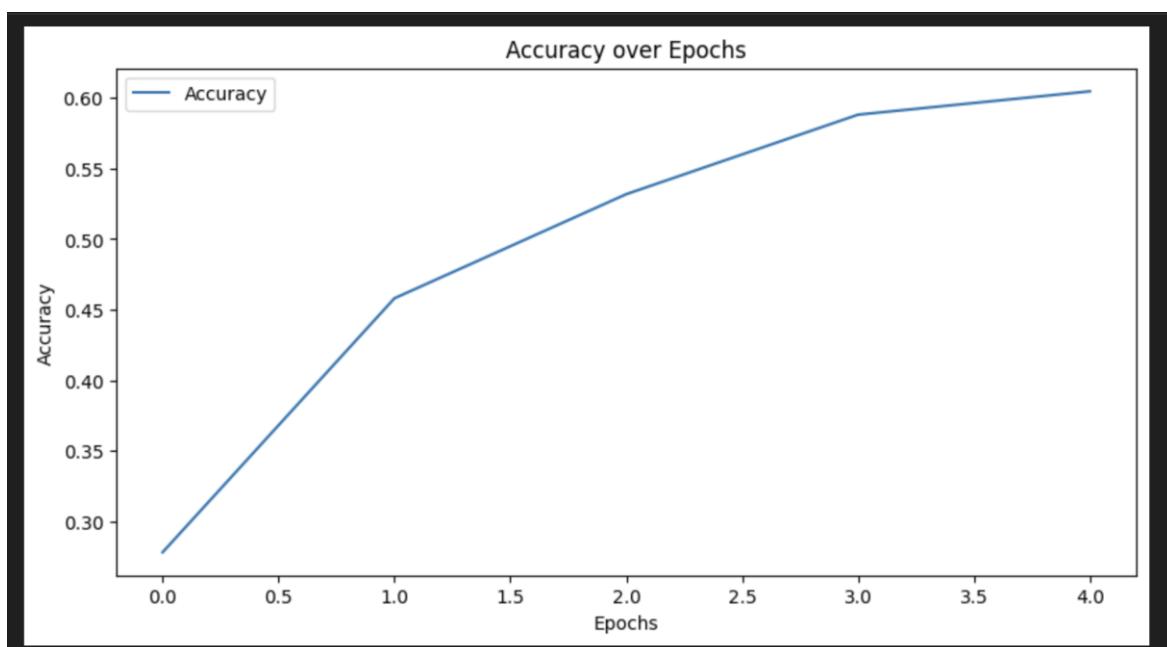
```
hidden_size1 = int(input_size / 2)
hidden_size2 = int( input_size / 3)
output_size = 10
```

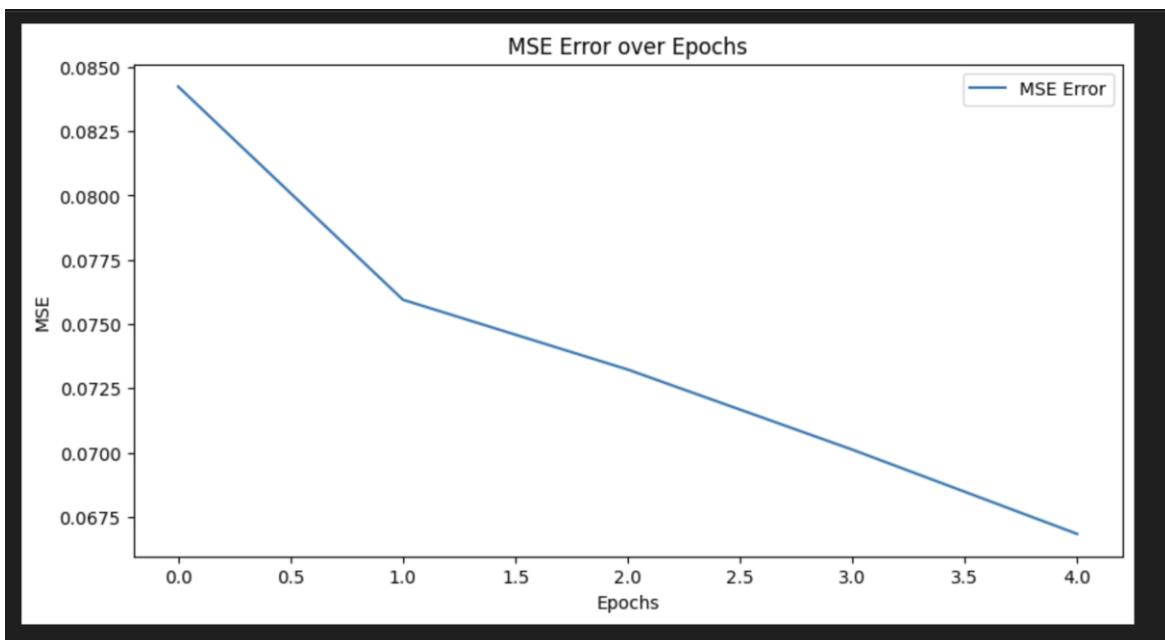
و نتایج میشه اینگونه :

```
Epoch 1/5, Avg Error: 0.084236, Accuracy: 0.278125
Class 0 Accuracy: 0.021088
Class 1 Accuracy: 0.084466
Class 2 Accuracy: 0.472911
Class 3 Accuracy: 0.463251
Class 4 Accuracy: 0.732919
Class 5 Accuracy: 0.484206
Class 6 Accuracy: 0.053986
Class 7 Accuracy: 0.274314
Class 8 Accuracy: 0.173625
Class 9 Accuracy: 0.019810
Epoch 2/5, Avg Error: 0.075945, Accuracy: 0.458042
Class 0 Accuracy: 0.071700
Class 1 Accuracy: 0.042439
Class 2 Accuracy: 0.467451
Class 3 Accuracy: 0.746325
Class 4 Accuracy: 0.710973
Class 5 Accuracy: 0.891937
Class 6 Accuracy: 0.531421
Class 7 Accuracy: 0.620532
Class 8 Accuracy: 0.460107
Class 9 Accuracy: 0.041684
Epoch 3/5, Avg Error: 0.073231, Accuracy: 0.531792
Class 0 Accuracy: 0.258541
Class 1 Accuracy: 0.003296
...
Class 6 Accuracy: 0.980177
Class 7 Accuracy: 0.397756
Class 8 Accuracy: 0.857379
Class 9 Accuracy: 0.641766
```

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

: نمودار





نتایج تست:

```

Test Accuracy: 58.92%
Test Class 0 Accuracy: 72.34%
Test Class 1 Accuracy: 0.17%
Test Class 2 Accuracy: 8.40%
Test Class 3 Accuracy: 32.15%
Test Class 4 Accuracy: 95.90%
Test Class 5 Accuracy: 90.07%
Test Class 6 Accuracy: 98.41%
Test Class 7 Accuracy: 35.35%
Test Class 8 Accuracy: 90.19%
Test Class 9 Accuracy: 65.68%

```

: قسمت پ :

باید relu مون رو عوض کنیم به relu و همه کارا رو دوباره انجام بدیم \heartsuit

relu تابع

```
def relu(x):  
    return np.maximum(0, x)
```

مشتق بزرگوار:

```
def relu_derivative(x):  
    return np.where(x > 0, 1, 0)
```

فقط باید همه جاهایی که سیگموید زدیم رو تبدیل کنیم به

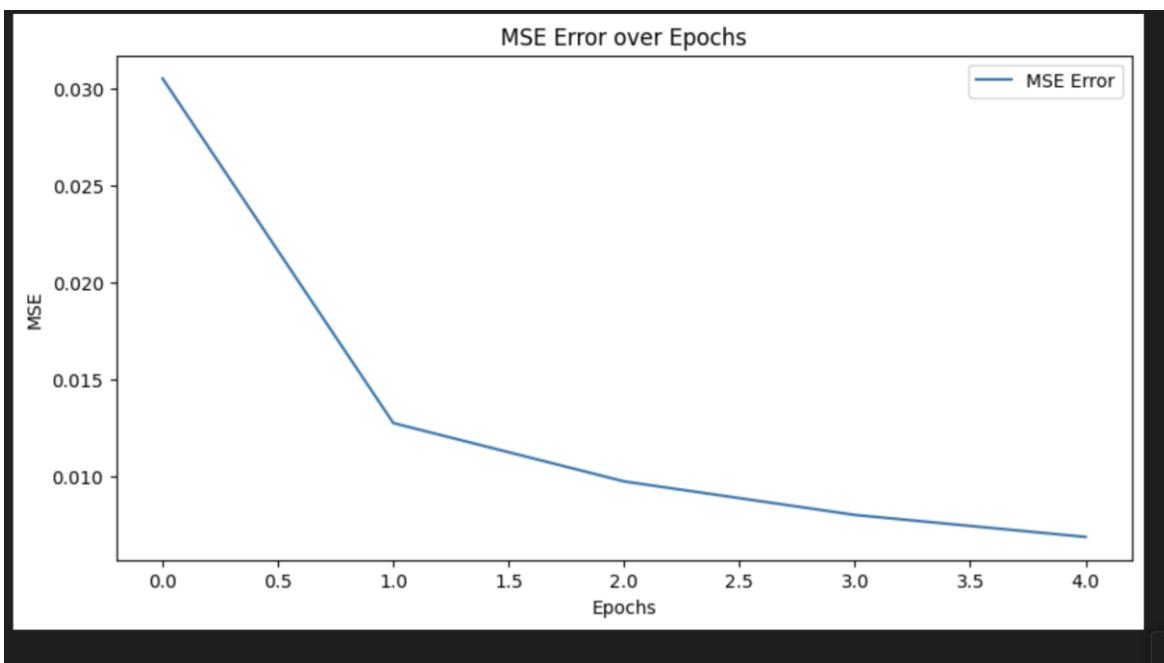
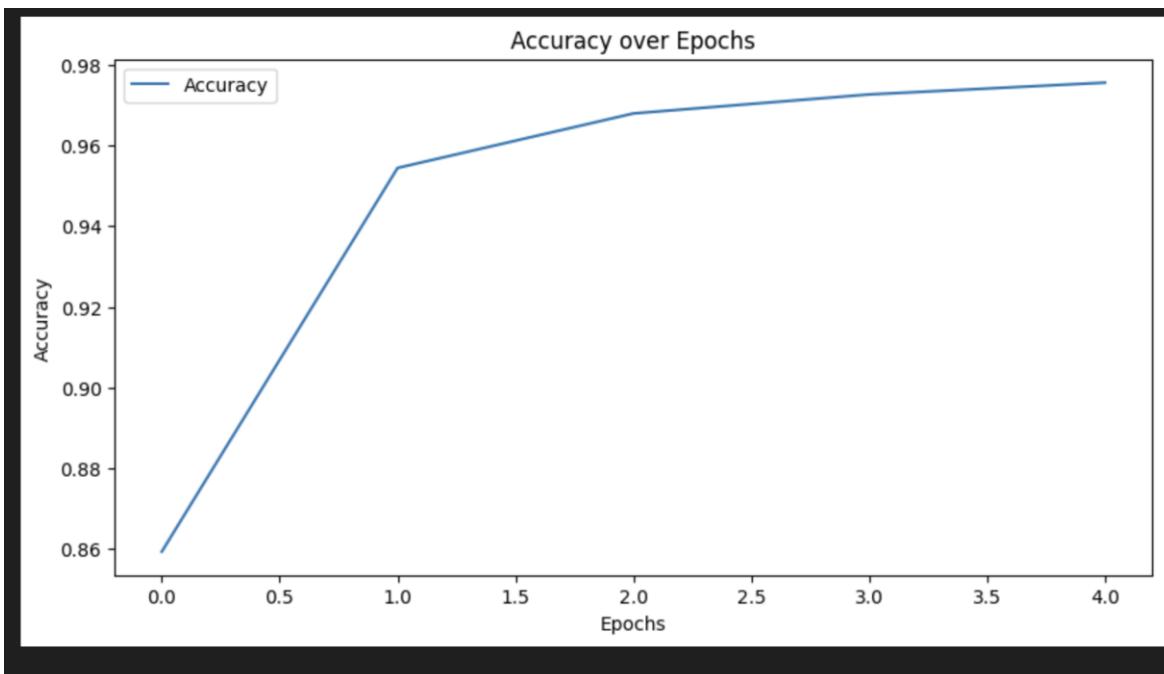
نتایج :

تک لایه با فیچر mfcc و مشتق یک و دوش:

حین اموزش :

```
Epoch 1/5, Avg Error: 0.030532, Accuracy: 0.859375  
Class 0 Accuracy: 0.857022  
Class 1 Accuracy: 0.861970  
Class 2 Accuracy: 0.830743  
Class 3 Accuracy: 0.832843  
Class 4 Accuracy: 0.900621  
Class 5 Accuracy: 0.844555  
Class 6 Accuracy: 0.916913  
Class 7 Accuracy: 0.837965  
Class 8 Accuracy: 0.912360  
Class 9 Accuracy: 0.799009  
Epoch 2/5, Avg Error: 0.012752, Accuracy: 0.954417  
Class 0 Accuracy: 0.961620  
Class 1 Accuracy: 0.955913  
Class 2 Accuracy: 0.950021  
Class 3 Accuracy: 0.947921  
Class 4 Accuracy: 0.969358  
Class 5 Accuracy: 0.957606  
Class 6 Accuracy: 0.969211  
Class 7 Accuracy: 0.934746  
Class 8 Accuracy: 0.966928  
Class 9 Accuracy: 0.931077  
Epoch 3/5, Avg Error: 0.009742, Accuracy: 0.967917  
Class 0 Accuracy: 0.974694  
Class 1 Accuracy: 0.976102  
...  
Class 6 Accuracy: 0.983129  
Class 7 Accuracy: 0.943475  
Class 8 Accuracy: 0.983051  
Class 9 Accuracy: 0.960792  
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

نمودار ها:



دقیقت تست:

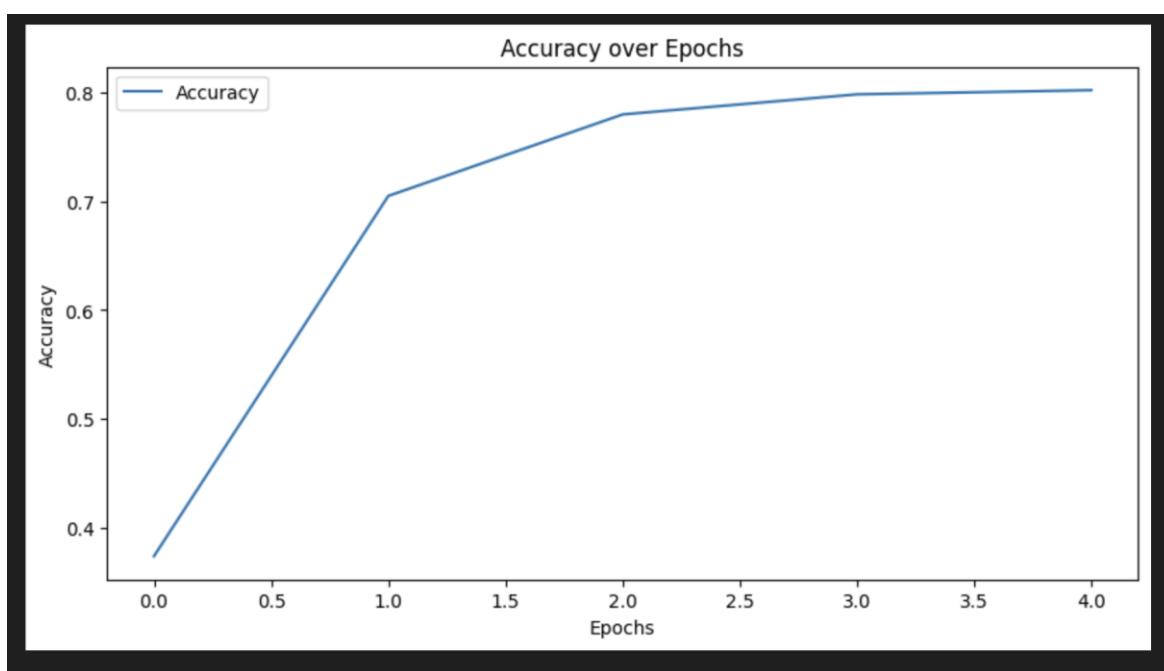
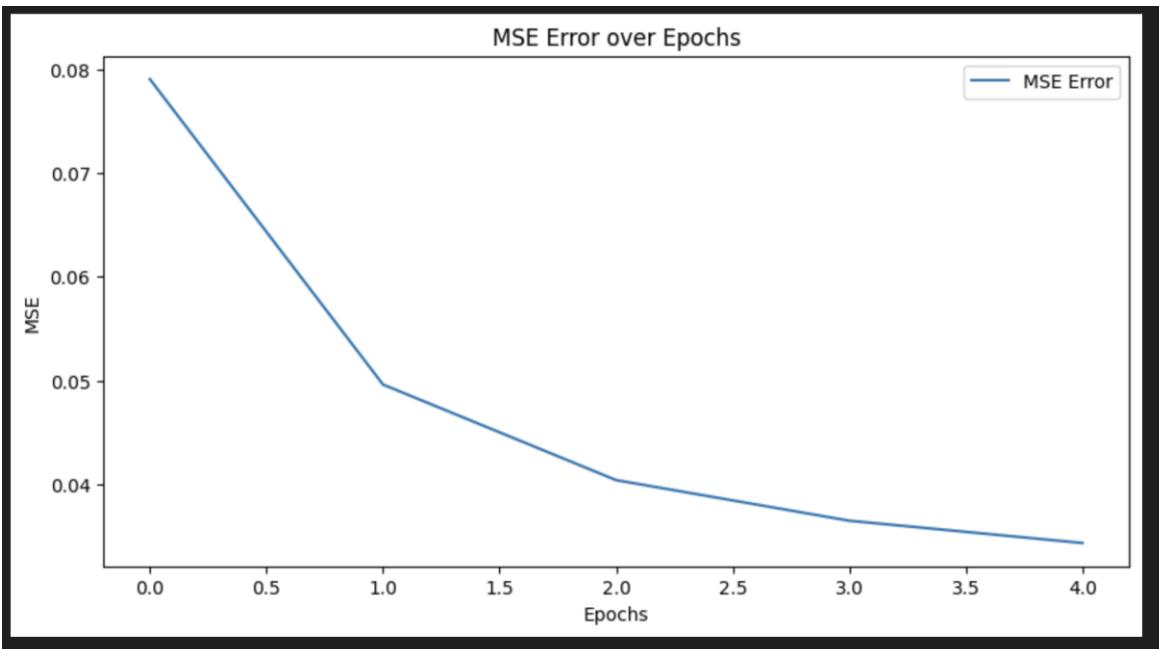
```
Test Accuracy: 96.12%
Test Class 0 Accuracy: 97.14%
Test Class 1 Accuracy: 97.38%
Test Class 2 Accuracy: 95.96%
Test Class 3 Accuracy: 96.45%
Test Class 4 Accuracy: 98.63%
Test Class 5 Accuracy: 97.14%
Test Class 6 Accuracy: 95.71%
Test Class 7 Accuracy: 92.09%
Test Class 8 Accuracy: 97.42%
Test Class 9 Accuracy: 93.24%
```

تک لایه مخفی با فیچر :mfcc mean

حین اموزش :

```
Epoch 1/5, Avg Error: 0.079026, Accuracy: 0.374042
Class 0 Accuracy: 0.026571
Class 1 Accuracy: 0.276473
Class 2 Accuracy: 0.412852
Class 3 Accuracy: 0.465351
Class 4 Accuracy: 0.743271
Class 5 Accuracy: 0.610141
Class 6 Accuracy: 0.676508
Class 7 Accuracy: 0.024938
Class 8 Accuracy: 0.291856
Class 9 Accuracy: 0.214197
Epoch 2/5, Avg Error: 0.049617, Accuracy: 0.704917
Class 0 Accuracy: 0.287642
Class 1 Accuracy: 0.633292
Class 2 Accuracy: 0.598488
Class 3 Accuracy: 0.660227
Class 4 Accuracy: 0.919255
Class 5 Accuracy: 0.924356
Class 6 Accuracy: 0.973429
Class 7 Accuracy: 0.388612
Class 8 Accuracy: 0.894998
Class 9 Accuracy: 0.763516
Epoch 3/5, Avg Error: 0.040415, Accuracy: 0.779750
Class 0 Accuracy: 0.546605
Class 1 Accuracy: 0.733004
...
Class 6 Accuracy: 0.964150
Class 7 Accuracy: 0.851621
Class 8 Accuracy: 0.860686
Class 9 Accuracy: 0.774247
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

نمودار ها:



دقت تست:

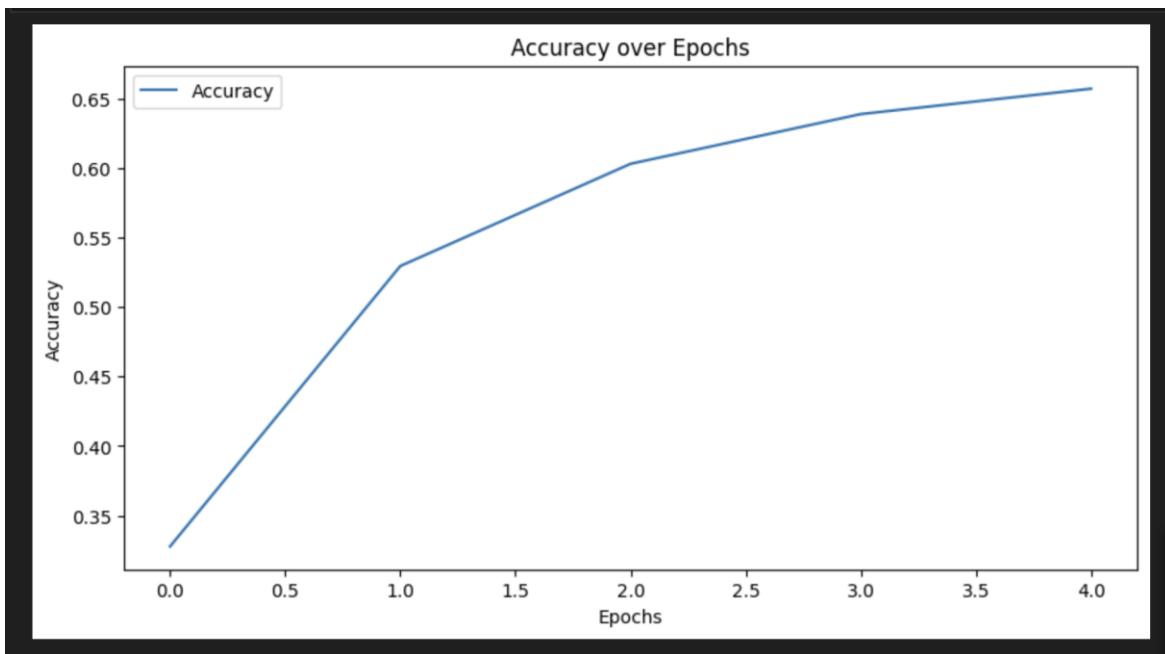
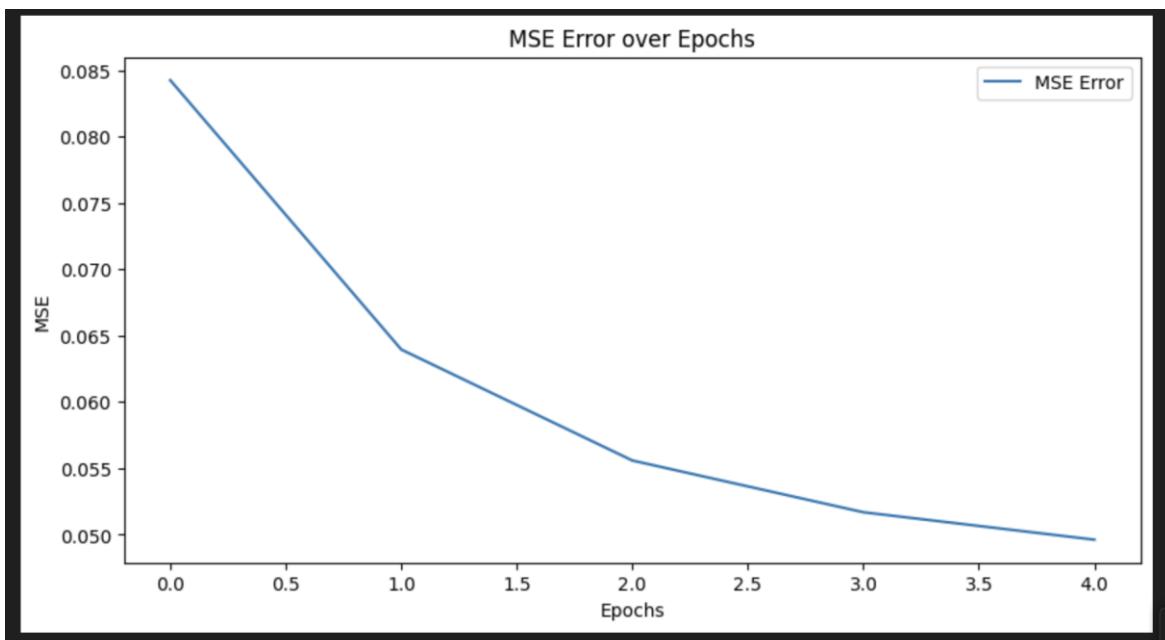
```
Test Accuracy: 80.17%
Test Class 0 Accuracy: 68.52%
Test Class 1 Accuracy: 75.92%
Test Class 2 Accuracy: 55.41%
Test Class 3 Accuracy: 72.86%
Test Class 4 Accuracy: 92.48%
Test Class 5 Accuracy: 89.90%
Test Class 6 Accuracy: 95.55%
Test Class 7 Accuracy: 86.36%
Test Class 8 Accuracy: 87.26%
Test Class 9 Accuracy: 78.68%
```

دو لایه مخفی با سایز قسمت الف

حین اموزش :

```
Epoch 1/5, Avg Error: 0.084240, Accuracy: 0.327750
Class 0 Accuracy: 0.018558
Class 1 Accuracy: 0.005768
Class 2 Accuracy: 0.000420
Class 3 Accuracy: 0.307014
Class 4 Accuracy: 0.534161
Class 5 Accuracy: 0.736908
Class 6 Accuracy: 0.618305
Class 7 Accuracy: 0.377805
Class 8 Accuracy: 0.573377
Class 9 Accuracy: 0.103591
Epoch 2/5, Avg Error: 0.063946, Accuracy: 0.529583
Class 0 Accuracy: 0.075496
Class 1 Accuracy: 0.000000
Class 2 Accuracy: 0.000000
Class 3 Accuracy: 0.511130
Class 4 Accuracy: 0.923395
Class 5 Accuracy: 0.878221
Class 6 Accuracy: 0.865036
Class 7 Accuracy: 0.627182
Class 8 Accuracy: 0.837950
Class 9 Accuracy: 0.571193
Epoch 3/5, Avg Error: 0.055587, Accuracy: 0.603167
Class 0 Accuracy: 0.206242
Class 1 Accuracy: 0.000000
...
Class 6 Accuracy: 0.959933
Class 7 Accuracy: 0.810474
Class 8 Accuracy: 0.873915
Class 9 Accuracy: 0.886092
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

نمودار ها:



دقت تست:

Test Accuracy: 65.47%
Test Class 0 Accuracy: 38.00%
Test Class 1 Accuracy: 0.00%
Test Class 2 Accuracy: 0.00%
Test Class 3 Accuracy: 74.31%
Test Class 4 Accuracy: 96.92%
Test Class 5 Accuracy: 91.25%
Test Class 6 Accuracy: 95.55%
Test Class 7 Accuracy: 83.84%
Test Class 8 Accuracy: 88.64%
Test Class 9 Accuracy: 87.69%

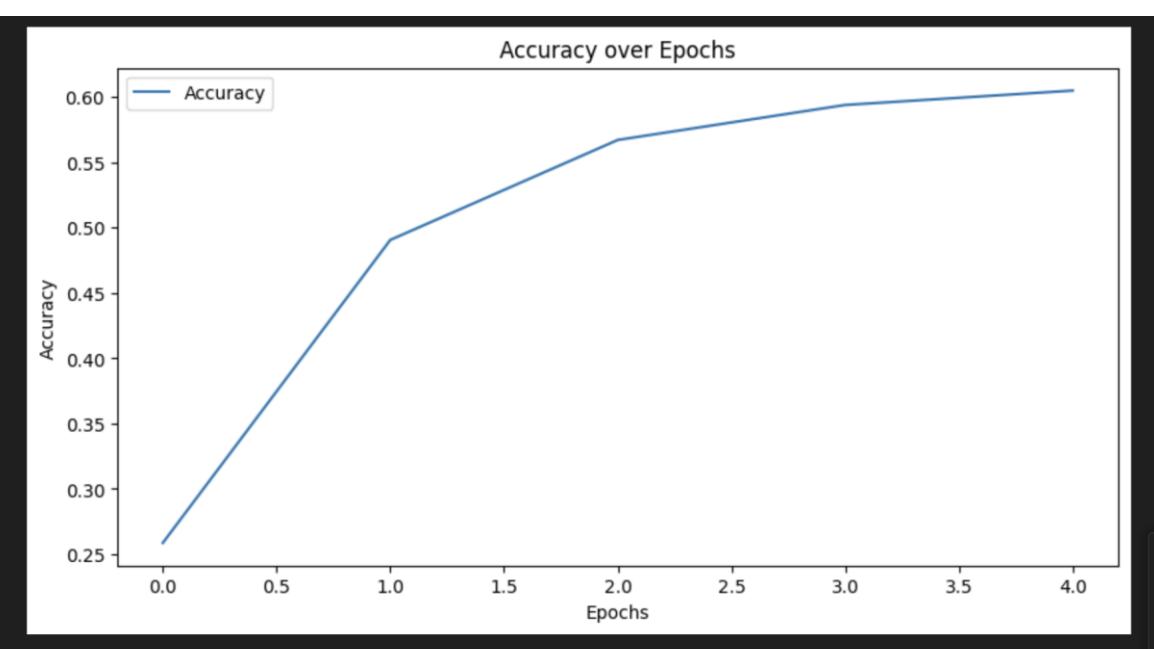
دو لایه مخفی با سایز دو سوم

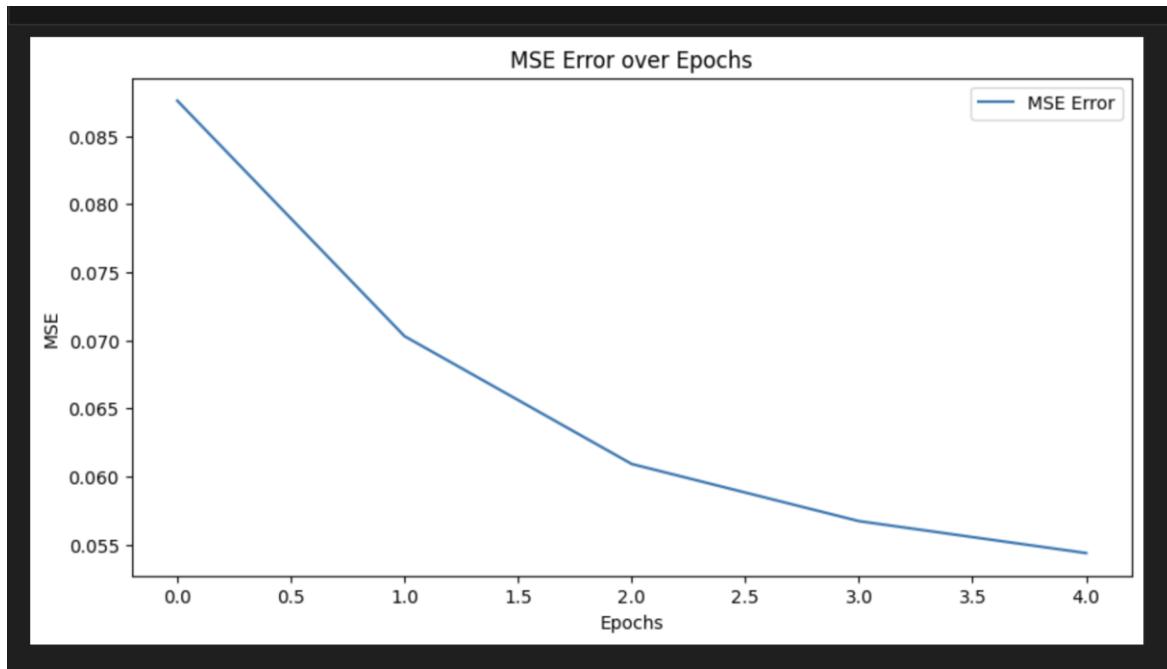
حین اموزش :

```
Epoch 1/5, Avg Error: 0.087589, Accuracy: 0.258708
Class 0 Accuracy: 0.036693
Class 1 Accuracy: 0.614751
Class 2 Accuracy: 0.402772
Class 3 Accuracy: 0.433851
Class 4 Accuracy: 0.000000
Class 5 Accuracy: 0.340815
Class 6 Accuracy: 0.000422
Class 7 Accuracy: 0.263092
Class 8 Accuracy: 0.256304
Class 9 Accuracy: 0.232769
Epoch 2/5, Avg Error: 0.070297, Accuracy: 0.490667
Class 0 Accuracy: 0.199072
Class 1 Accuracy: 0.756077
Class 2 Accuracy: 0.543889
Class 3 Accuracy: 0.646787
Class 4 Accuracy: 0.000000
Class 5 Accuracy: 0.788030
Class 6 Accuracy: 0.000000
Class 7 Accuracy: 0.640482
Class 8 Accuracy: 0.801571
Class 9 Accuracy: 0.519191
Epoch 3/5, Avg Error: 0.060912, Accuracy: 0.567208
Class 0 Accuracy: 0.525938
Class 1 Accuracy: 0.783684
...
Class 6 Accuracy: 0.000000
Class 7 Accuracy: 0.770158
Class 8 Accuracy: 0.785035
Class 9 Accuracy: 0.784565
```

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

نمودار ها:





دقت تست:

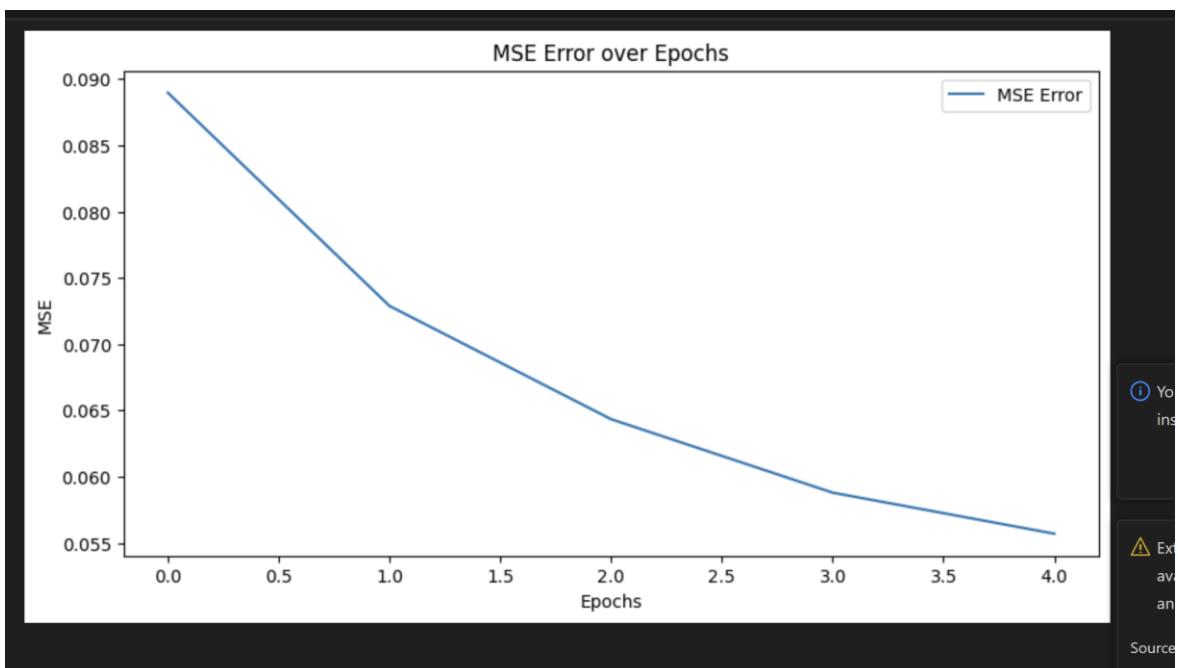
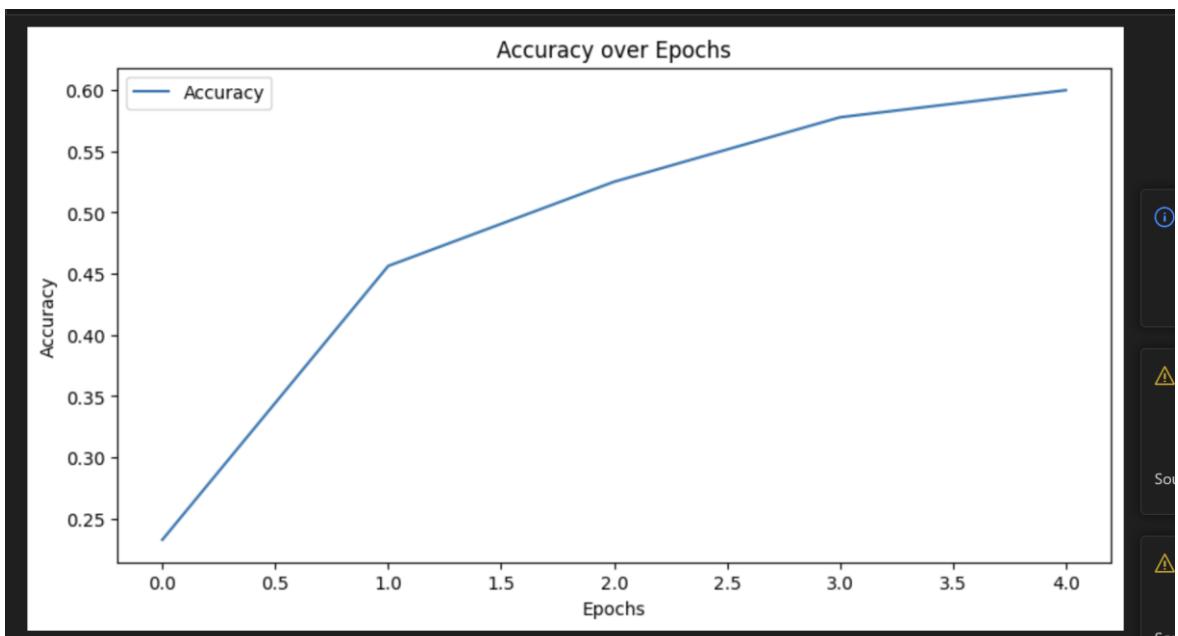
```
Test Accuracy: 60.47%
Test Class 0 Accuracy: 66.14%
Test Class 1 Accuracy: 82.90%
Test Class 2 Accuracy: 60.58%
Test Class 3 Accuracy: 69.14%
Test Class 4 Accuracy: 0.00%
Test Class 5 Accuracy: 89.90%
Test Class 6 Accuracy: 0.00%
Test Class 7 Accuracy: 79.80%
Test Class 8 Accuracy: 80.72%
Test Class 9 Accuracy: 79.20%
```

دو لایه مخفی با سایز یک سوم و یک دوم

حین اموزش :

```
Epoch 1/5, Avg Error: 0.088959, Accuracy: 0.232792
Class 0 Accuracy: 0.073809
Class 1 Accuracy: 0.000000
Class 2 Accuracy: 0.621588
Class 3 Accuracy: 0.367073
Class 4 Accuracy: 0.000000
Class 5 Accuracy: 0.862843
Class 6 Accuracy: 0.256854
Class 7 Accuracy: 0.035328
Class 8 Accuracy: 0.014055
Class 9 Accuracy: 0.104829
Epoch 2/5, Avg Error: 0.072876, Accuracy: 0.456333
Class 0 Accuracy: 0.016027
Class 1 Accuracy: 0.000000
Class 2 Accuracy: 0.672407
Class 3 Accuracy: 0.840403
Class 4 Accuracy: 0.000000
Class 5 Accuracy: 0.931837
Class 6 Accuracy: 0.779418
Class 7 Accuracy: 0.537406
Class 8 Accuracy: 0.020670
Class 9 Accuracy: 0.775485
Epoch 3/5, Avg Error: 0.064336, Accuracy: 0.525083
Class 0 Accuracy: 0.150991
Class 1 Accuracy: 0.000000
...
Class 6 Accuracy: 0.970898
Class 7 Accuracy: 0.836658
Class 8 Accuracy: 0.814800
Class 9 Accuracy: 0.859678
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

نمودار ها:



دقت تست:

```
Test Accuracy: 60.88%
Test Class 0 Accuracy: 46.26%
Test Class 1 Accuracy: 0.00%
Test Class 2 Accuracy: 52.18%
Test Class 3 Accuracy: 63.33%
Test Class 4 Accuracy: 0.00%
Test Class 5 Accuracy: 91.41%
Test Class 6 Accuracy: 97.30%
Test Class 7 Accuracy: 84.18%
Test Class 8 Accuracy: 87.78%
Test Class 9 Accuracy: 83.54%
```

دقت ها برای `relu` در تک لایه مخفی ها به مراتب بهتر شد و برای بعضی دو لایه مخفی ها بدتر شد و برای برخی بهتر ولی به طور کلی `relu` خیلی از مشکلات `sigmoid` را حل کرده

سوال سه :

در این تمرین هدف ما تشخیص ژانرهای موسیقیایی با استفاده از شبکه های عصبی پیچشی است. ما از دادگان GTZAN استفاده کردہ ایم که شامل 10 کلاس ژانر موسیقی (CNN) است. هر فایل صدا دارای طول 30 ثانیه است.

اینجا داریم ابزار لازم رو نصب و داده ها رو از سایت Kaggle دانلود می کنیم:

توضیح کد :

```
!pip install -q kaggle

!mkdir -p ~/.kaggle
!cp /content/kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

! kaggle datasets download andradaolteanu/gtzan-dataset-music-genre-classification

Dataset URL: https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification
License(s): other
Downloading gtzan-dataset-music-genre-classification.zip to /content
99% 1.20G/1.21G [00:08<00:00, 169MB/s]
100% 1.21G/1.21G [00:08<00:00, 152MB/s]

! unzip /content/gtzan-dataset-music-genre-classification.zip
```

اول باید کتابخونه Kaggle را نصب کنیم تا بتونیم داده‌ها را از سایت Kaggle بگیریم. با دستور pip install -q kaggle این کار را انجام میدیم. بعدش یه پوشه به اسم kaggle. می‌سازیم تا فایل API رو تو ش بذاریم. این پوشه رو با mkdir -p ~/.kaggle رو با کلید کلید API رو تو ش هست رو به این پوشه کپی کنیم. این کار حالا باید فایل kaggle.json که کلید API ما تو ش هست رو به این پوشه کپی کنیم. این کار رو با cp /content/kaggle.json ~/.kaggle می‌دانیم.

برای اینکه فقط خودمون بتونیم به این فایل دسترسی داشته باشیم دسترسی‌هاش رو تنظیم می‌کنیم با chmod 600 ~/.kaggle/kaggle.json. حالا با دستور kaggle datasets download .gtzan-dataset-music-genre-classification داده‌های مربوط به مجموعه داده GTZAN را از Kaggle دانلود می‌کنیم و بعدش فایل زیپ رو با unzip /content/gtzan-dataset- می‌دانیم. از حالت فشرده خارج music-genre-classification.zip می‌کنیم.

```

# Load data
data_dir = "/content/Data/genres_original"

genres = os.listdir(data_dir)
num_samples_per_genre = 100
audio_data = []
labels = []

# Define target size for spectrogram
target_height = 128
target_width = 128

for genre in genres:
    genre_dir = os.path.join(data_dir, genre)
    for filename in tqdm(os.listdir(genre_dir)[:num_samples_per_genre]):
        file_path = os.path.join(genre_dir, filename)
        try:
            signal, sr = librosa.load(file_path, sr=None)
            mel_spec = librosa.feature.melspectrogram(y=signal, sr=sr)
            mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
            mel_spec_db_resized = resize(mel_spec_db, (target_height, target_width), anti_aliasing=True)
            mel_spec_db_resized = mel_spec_db_resized[..., np.newaxis] # Add channel dimension
            audio_data.append(mel_spec_db_resized)
            labels.append(genre)
        except Exception as e:
            print(f"Error loading {file_path}: {str(e)}")

```

: خروجی :

```

100%|██████████| 100/100 [00:09<00:00, 10.70it/s]
100%|██████████| 100/100 [00:10<00:00,  9.10it/s]
 0%|          | 0/100 [00:00<?, ?it/s]<ipython-input-22-95e55a3d94d4>:19: UserWarning: PySoundFile failed to open jazz.00054.wav
  signal, sr = librosa.load(file_path, sr=None)
/usr/local/lib/python3.10/dist-packages/librosa/core/audio.py:184: FutureWarning: librosa.core.audio._audio.read is deprecated as of librosa version 0.10.0.
  It will be removed in librosa version 1.0.
  y, sr_native = __audioread_load(path, offset, duration, dtype)
  3%|█         | 3/100 [00:00<00:11,  8.72it/s]
Error loading /content/Data/genres_original/jazz/jazz.00054.wav:
100%|██████████| 100/100 [00:08<00:00, 12.25it/s]
100%|██████████| 100/100 [00:09<00:00, 10.84it/s]
100%|██████████| 100/100 [00:10<00:00,  9.71it/s]
100%|██████████| 100/100 [00:07<00:00, 13.20it/s]
100%|██████████| 100/100 [00:09<00:00, 10.12it/s]
100%|██████████| 100/100 [00:08<00:00, 11.55it/s]
100%|██████████| 100/100 [00:08<00:00, 11.72it/s]
100%|██████████| 100/100 [00:11<00:00,  8.48it/s]

```

بعد لیست ژانرهای موجود در پوشه داده‌ها رو با `os.listdir(data_dir)` می‌گیریم و تعیین می‌کنیم که از هر ژانر 100 نمونه می‌خواهیم. برای ذخیره داده‌های صوتی و برچسب‌ها دو تا لیست درست می‌کنیم: `labels` و `audio_data`:

حالا برای هر ژانر فایل‌های صوتی رو می‌خونیم. از tqdm استفاده می‌کنیم که بهمون یه نوار پیشرفت نشون بده و بفهمیم چقدر از کار مونده. هر فایل صوتی رو با librosa.load می‌خونیم و طیف‌نگارش رو با librosa.feature.melspectrogram استخراج می‌کنیم. بعدش طیف‌نگار رو به واحد دسیبل تبدیل می‌کنیم تا مقیاسش مناسب بشه.

چون اندازه طیف‌نگارها ممکنه متفاوت باشه با resize همشون رو به اندازه 128 تبدیل می‌کنیم. نهايتاً اين طیف‌نگارهای تغيير اندازه داده شده رو به لیست audio_data اضافه می‌کنیم و برچسب ژانر رو هم به لیست labels اضافه می‌کنیم.

```
audio_data = np.array(audio_data)
labels = np.array(labels)

labelencoder = LabelEncoder()
labels = labelencoder.fit_transform(labels)
len(labelencoder.classes_)

10

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(audio_data, labels, test_size=0.2, random_state=42)

len(np.unique(y_train))

10
```

حالا که داده‌ها رو جمع کردیم لیست‌ها رو به آرایه‌های numpy تبدیل می‌کنیم. برچسب‌ها رو با LabelEncoder به اعداد تبدیل می‌کنیم تا مدل بتونه باهاشون کار کنه.

برای اينکه مدل رو آموزش بدیم داده‌ها رو به دو دسته آموزش (80٪) و تست (20٪) تقسیم می‌کنیم

```

# Build CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(target_height, target_width, 1)),
    layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='valid'),
    layers.BatchNormalization(),
    layers.Flatten(),
    layers.Dense(10, activation='softmax')
])

# Compile model
adam = optimizers.Adam(learning_rate=1e-4)
model.compile(optimizer=adam,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
# Train model
history = model.fit(X_train, y_train, epochs=30, batch_size=32, validation_data=(X_test, y_test))

```

نمایشی از ساختار مدل:

```

Model: "sequential_6"

Layer (type)          Output Shape         Param #
===== 
conv2d_5 (Conv2D)     (None, 126, 126, 32)      320
max_pooling2d_5 (MaxPooling2D) (None, 62, 62, 32)      0
batch_normalization_5 (BatchNormalization) (None, 62, 62, 32)      128
flatten_5 (Flatten)   (None, 123008)           0
dense_10 (Dense)     (None, 10)                1230090
=====
Total params: 1230538 (4.69 MB)
Trainable params: 1230474 (4.69 MB)
Non-trainable params: 64 (256.00 Byte)

```

خروجی :

```

Epoch 1/30
25/25 [=====] - 14s 490ms/step - loss: 2.3407 - accuracy: 0.3579 - val_loss: 9.0888 - val_accuracy: 0.1850
Epoch 2/30
25/25 [=====] - 12s 481ms/step - loss: 0.6814 - accuracy: 0.7722 - val_loss: 10.3134 - val_accuracy: 0.2150
...
Epoch 29/30
25/25 [=====] - 13s 537ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 1.1132 - val_accuracy: 0.6850
Epoch 30/30
25/25 [=====] - 13s 533ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 1.1232 - val_accuracy: 0.6800
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```

مدل CNN را با استفاده از Keras می‌سازیم. اول یه مدل ترتیبی (Sequential) می‌سازیم و بهش لایه‌های پیچشی batch normalization و pooling و اضافه می‌کنیم. در نهایت یه لایه

Dense با 10 نورون (برای 10 ژانر موسیقی) و تابع فعالسازی softmax به مدل اضافه می‌کنیم.

مدل رو با بهینه‌ساز Adam و نرخ یادگیری 0.0001 کامپایل می‌کنیم و تابع خطا رو می‌ذاریم. بعد مدل رو با داده‌های آموزش و برای 30 دوره آموزش می‌دیم.

```
def plot_history(hist, accuracy=True):
    plt.figure()
    fig, axs = plt.subplots(1)
    if(accuracy):
        # accuracy subplot
        axs.plot(hist.history["accuracy"], label="train accuracy")
        axs.plot(hist.history["val_accuracy"], label="test accuracy")
        axs.set_ylabel("Accuracy")
        axs.legend(loc="lower right")
        axs.set_title("Accuracy eval")
    else:
        # Error subplot
        axs.plot(hist.history["loss"], label="train error")
        axs.plot(hist.history["val_loss"], label="test error")
        axs.set_ylabel("Error")
        axs.set_xlabel("Epoch")
        axs.legend(loc="upper right")
        axs.set_title("Error eval")

    plt.show()
```

```
# Retrieve training and validation accuracy
train_accuracy = history.history['accuracy'][-1] # Last epoch accuracy
test_accuracy = history.history['val_accuracy'][-1] # Last epoch validation accuracy

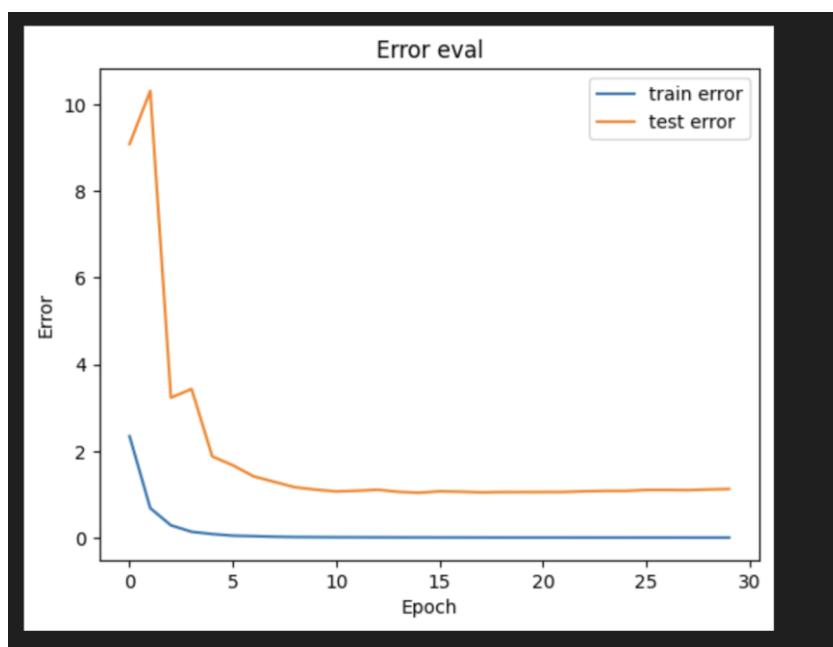
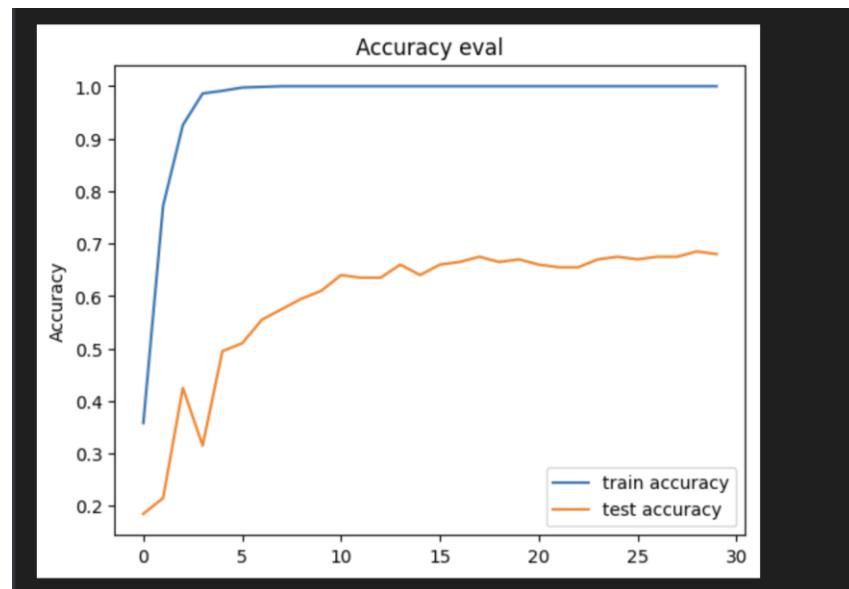
# Print accuracies
print('Training accuracy:', train_accuracy)
print('Test accuracy:', test_accuracy)
```

خروجی:

```
Training accuracy: 1.0
Test accuracy: 0.6800000071525574
```

در نهایت تاریخچه آموزش رو بررسی می‌کنیم تا ببینیم چقدر مدل خوب یاد گرفته. نمودار دقت و خطا رو رسم می‌کنیم تا به وضوح ببینیم که مدل چطور عمل کرده. همچنین دقت نهایی مدل روی داده‌های آموزش و تست رو چاپ می‌کنیم.

نمودار :



```

from sklearn.metrics import classification_report, accuracy_score
# Predict classes
y_pred = np.argmax(model.predict(X_test), axis=-1)

# Classification report
report = classification_report(y_test, y_pred, target_names=genres)
print(report)

```

مدل رو تست می‌کنیم تا ببینیم چقدر خوب ژانرهای موسیقی رو تشخیص میده. با استفاده از `classification_report` گزارش کاملی از عملکرد مدل روی داده‌های تست می‌گیریم و دقت نهایی رو چاپ می‌کنیم.

	precision	recall	f1-score	support
blues	0.73	0.52	0.61	21
country	0.78	0.90	0.84	20
jazz	0.42	0.83	0.56	12
disco	0.58	0.61	0.60	23
metal	0.94	0.73	0.82	22
rock	0.88	0.61	0.72	23
pop	0.71	1.00	0.83	15
reggae	0.62	0.83	0.71	18
hiphop	0.68	0.68	0.68	19
classical	0.59	0.37	0.45	27
accuracy			0.68	200
macro avg	0.69	0.71	0.68	200
weighted avg	0.71	0.68	0.68	200

کد این سوال در پوشه با نام 3 وجود دارد.

سوال چهار

قسمت الف:

ابتدا نیاز داریم کتابخانه‌ی Kaggle را نصب کنیم و کلید API را تنظیم کنیم تا داده‌ها را دانلود کنیم. این کار رو با دستورهای زیر انجام می‌دیم:

```

!pip install -q kaggle

!mkdir -p ~/.kaggle
!cp /content/kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

cp: cannot stat '/content/kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory

! kaggle datasets download kongaevans/speaker-recognition-dataset

Dataset URL: https://www.kaggle.com/datasets/kongaevans/speaker-recognition-dataset
License(s): unknown
Downloading speaker-recognition-dataset.zip to /content
95% 219M/231M [00:02<00:00, 123MB/s]
100% 231M/231M [00:02<00:00, 87.9MB/s]

! unzip /content/speaker-recognition-dataset.zip

```

با این دستورها، کتابخانه Kaggle نصب می‌شود، پوششی ساخته می‌شود و کلید API که در فایل kaggle.json هست به این پوشش کپی می‌شود و دسترسی‌های لازم بهش داده می‌شود. بعدها مجموعه داده‌ی مورد نظر را از Kaggle دانلود می‌کنیم و فایل فشرده را استخراج می‌کنیم.

سپس، یک تابع به نام load_data تعریف می‌کنیم که داده‌های صوتی را از پوشش‌ها بارگذاری می‌کند:

```

def load_data(data_dir):
    data = []
    labels = []
    speakers = ['Benjamin_Netanyahu', 'Jens_Stoltenberg', 'Julia_Gillard', 'Margaret_Tatcher', 'Nelson_Mandela']

    for speaker in speakers:
        speaker_dir = os.path.join(data_dir, speaker)
        for filename in os.listdir(speaker_dir):
            if filename.endswith('.wav'):
                file_path = os.path.join(speaker_dir, filename)
                data.append(file_path)
                labels.append(speaker)

    return data, labels

data_dir = '/content/16000_pcm_speeches/'
data, labels = load_data(data_dir)

```

سپس باید ویژگی‌های صوتی مورد نظر را از فایل‌های صوتی استخراج کنیم. برای این کار یک تابع به نام extract_features تعریف می‌کنیم:

```
def extract_features(file_path):
    y, sr = librosa.load(file_path, sr=16000)

    # Spectral Centroid
    spectral_centroid = np.mean(librosa.feature.spectral_centroid(y=y, sr=sr))

    # Zero Crossing Rate
    zero_crossing_rate = np.mean(librosa.feature.zero_crossing_rate(y))

    # Root Mean Square Energy
    rmse = np.mean(librosa.feature.rms(y=y))

    return [spectral_centroid, zero_crossing_rate, rmse]
```

این تابع فایل صوتی را بازگذاری می‌کنه و سه ویژگی Zero Crossing، Spectral Centroid و Root Mean Square Energy را محاسبه و بر می‌گردونه.

برای اینکه ویژگی‌ها مقیاس مناسبی داشته باشند، باید آن‌ها را نرمال‌سازی کنیم. این کار را با تابع normalize_features انجام می‌دهیم:

```
def normalize_features(features):
    scaler = StandardScaler()
    return scaler.fit_transform(features)
```

در اینجا از StandardScaler برای نرمال‌سازی داده‌ها استفاده می‌کنیم.

برای اندازه‌گیری زمان لازم برای استخراج ویژگی‌ها، از کد زیر استفاده می‌کنیم:

```
start_time = time.time()
features = [extract_features(file) for file in data]
end_time = time.time()
execution_time = end_time - start_time
print(f"Time taken: {execution_time} s")
```

این بخش زمان لازم برای استخراج ویژگی‌ها از تمام فایل‌های صوتی را اندازه‌گیری می‌کند و نشونم.^{۵۵}

```
time taken: 69.52728605270386 s
```

برای استفاده از پردازش موازی و سرعت بخشیدن به استخراج ویژگی‌ها، از استفاده می‌کنیم ThreadPoolExecutor:

```
# Timing feature extraction in parallel
start_time_parallel = time.time()

with ThreadPoolExecutor() as executor:
    features_parallel = executor.map(extract_features, data)

end_time_parallel = time.time()
execution_time_parallel = end_time_parallel - start_time_parallel
print(f"time taken for parallel: {execution_time_parallel} s")
```

این کد ویژگی‌ها را به صورت موازی استخراج می‌کند و زمان لازم را محاسبه و نشونم.^{۵۶}

```
time taken for parallel: 37.71994400024414 s
```

بعد از استخراج ویژگی‌ها، آن‌ها را نرمال‌سازی می‌کنیم:
سپس برچسب‌ها را به اعداد تبدیل می‌کنیم:

```
# Convert labels to numeric form
label_mapping = {label: idx for idx, label in enumerate(set(labels))}
numeric_labels = [label_mapping[label] for label in labels]
```

اینجا از یک دیکشنری برای نگاشت هر برچسب به یک عدد استفاده می‌کنیم.
برای تقسیم داده‌ها به مجموعه‌های آموزشی و آزمایشی، از train_test_split استفاده می‌کنیم:

```
# Split the data into training and testing sets randomly
X_train, X_test, y_train, y_test = train_test_split(features, numeric_labels, test_size=0.2, random_state=42)

# Ensure balanced data split
print(f"Number of samples in training set: {len(X_train)}")
print(f"Number of samples in testing set: {len(X_test)}")
```

اینجا داده‌ها رو به صورت 80٪ برای آموزش و 20٪ برای آزمایش تقسیم می‌کنیم.

```
Number of samples in training set: 6000
Number of samples in testing set: 1501
```

در نهایت، مدل LogisticRegression را با استفاده از SGDClassifier و Logistic Regression در نهایت، مدل LogisticRegression را با استفاده از SGDClassifier و Logistic Regression آموزش می‌دیم و ارزیابی می‌کنیم:

(من هر چی داخل logistic گشتم نتونستم learning rate رو پیدا کنم و وقتی یک ریسرچی کردم اکثرا ساجست داده بودن از sgd استفاده کن بنابراین اینجا استفاده از هر دو را گزارش میدهم)

اینجا مدل SGDClassifier را با استفاده از loss تابع log_loss و یادگیری constant با نرخ 0.01 آموزش می‌دیم. بعدش پیش‌بینی‌ها رو انجام می‌دیم و دقت، ماتریس در هم ریختگی و گزارش طبقه‌بندی رو چاپ می‌کنیم.

```
from sklearn.metrics import confusion_matrix, classification_report

# Define the SGDClassifier with logistic regression loss function
model = SGDClassifier(loss='log_loss', max_iter=1000, learning_rate='constant', eta0=0.01)

# Train the model
model.fit(X_train, y_train)

# Predict labels for the test set
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = model.score(X_test, y_test)
print(f"Accuracy: {accuracy}")

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate classification report (F1 score, recall, precision)
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

خروجی:

```
Accuracy: 0.664890073284477
Confusion Matrix:
[[202  73  15  17   3]
 [ 74 132  63  41   0]
 [  5  46 230   2   0]
 [ 14  10  16 170  73]
 [  0  0   0  51 264]]
Classification Report:
              precision    recall   f1-score   support
          0       0.68     0.65     0.67      310
          1       0.51     0.43     0.46      310
          2       0.71     0.81     0.76      283
          3       0.60     0.60     0.60      283
          4       0.78     0.84     0.81      315
          accuracy           0.66      1501
          macro avg       0.66     0.67     0.66      1501
          weighted avg    0.66     0.66     0.66      1501
```

سپس مدل LogisticRegression را با استفاده از solver sag و تعداد تکرار 1000 آموزش می‌دهیم و نتایج مشابه رو چاپ می‌کنیم.

```
# Define the Logistic Regression model with gradient descent approach
model = LogisticRegression(max_iter=1000, solver='sag', )

# Train the model
model.fit(X_train, y_train)

# Predict labels for the test set
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = model.score(X_test, y_test)
print(f"Accuracy: {accuracy}")

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate classification report (F1 score, recall, precision)
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

خروجی:

```
Accuracy: 0.715522984676882
Confusion Matrix:
[[190  80  15  25   0]
 [ 42 208  49  11   0]
 [  4  44 224  11   0]
 [ 14   5   5 206  53]
 [  0   0   0  69 246]]
Classification Report:
precision    recall    f1-score   support
          0       0.76      0.61      0.68      310
          1       0.62      0.67      0.64      310
          2       0.76      0.79      0.78      283
          3       0.64      0.73      0.68      283
          4       0.82      0.78      0.80      315
accuracy                           0.72      1501
macro avg       0.72      0.72      0.72      1501
weighted avg    0.72      0.72      0.72      1501
```

کد این قسمت در پوشه چهار در فایل a موجود است
برای قسمت ب دقیقا همین کد را باید ران کنیم فقط تابع feature extractor ما متفاوت
میشود

```
def extract_features(file_path):
    y, sr = librosa.load(file_path, sr=16000)
    n_mfcc = 13
    mfcc = librosa.feature.mfcc(y=y, n_mfcc=n_mfcc)
    mfcc_mean = np.mean(mfcc, axis=1)

    return mfcc_mean
```

اینجا از mfcc mean به عنوان فیچر هایمان استفاده میکنیم
نتایج با :sgd classifire

```

Accuracy: 0.9653564290473018
Confusion Matrix:
[[267  7  3  1  4]
 [ 10 301  0  0  3]
 [ 2  1 310  4  4]
 [ 0  0  5 290  0]
 [ 5  2  1  0 281]]
Classification Report:
precision    recall   f1-score   support
          0       0.94      0.95      0.94      282
          1       0.97      0.96      0.96      314
          2       0.97      0.97      0.97      321
          3       0.98      0.98      0.98      295
          4       0.96      0.97      0.97      289
accuracy           0.97      0.97      0.97      1501
macro avg       0.97      0.97      0.97      1501
weighted avg    0.97      0.97      0.97      1501

```

نتایج با logistic:

```

Accuracy: 0.976682211858/608
Confusion Matrix:
[[270  9  0  0  3]
 [ 9 300  1  0  4]
 [ 1  0 319  1  0]
 [ 0  0  0 295  0]
 [ 4  2  1  0 282]]
Classification Report:
precision    recall   f1-score   support
          0       0.95      0.96      0.95      282
          1       0.96      0.96      0.96      314
          2       0.99      0.99      0.99      321
          3       1.00      1.00      1.00      295
          4       0.98      0.98      0.98      289
accuracy           0.98      0.98      0.98      1501
macro avg       0.98      0.98      0.98      1501
weighted avg    0.98      0.98      0.98      1501

```

قسمت پ :

راه حل ها :

استفاده از مجموعه داده‌های بیشتر: افزایش تنوع و حجم داده‌ها می‌تواند به دقت مدل کمک کند. برای این کار می‌توان از منابع دیگری خارج از مجموعه داده‌های آموزشی استفاده کرد یا با جمع‌آوری داده‌های بیشتر و گسترده‌تر، مجموعه داده را گسترش داد.

۲. استفاده از معماری‌های پیچیده‌تر مدل: استفاده از معماری‌های یادگیری عمیق مثل CNN یا شبکه‌های RNN می‌تواند به دقت مدل کمک کند. این معماری‌ها قادر به استخراج ویژگی‌های پیچیده‌تری از داده‌های ورودی هستند.

۳. استفاده از تکنیک‌های آموزش نظارت شده: ممکن است از روش‌هایی مانند آموزش نظارت شده با استفاده از تکنیک‌های نظارت شده برای استفاده از داده‌های آموزشی بدون برچسب استفاده کنیم. این روش‌ها می‌توانند به مدل کمک کنند تا اطلاعات بیشتری از داده‌ها را درک کند و دقت بالاتری در پیش‌بینی داشته باشد.

۴. استفاده از روش‌های ارزیابی مناسب: استفاده از روش‌های ارزیابی مناسب می‌تواند به شناسایی نقاط ضعف مدل کمک کند و از طریق بهبود آن‌ها، دقت واقع گرایانه مدل را افزایش دهد. این شامل استفاده از معیارهای ارزیابی مناسب مثل دقت، حساسیت، و دقت متوازن است.

یا یک راه دیگر این است که یک کلاس داشته باشیم که داده‌های متفرقه را تشخیص دهد و با آموزش مدل هم با داده درست و هم داده متفرقه تشخیص ان کلاس را نیز انجام دهیم