

In []:

PREPROCESSING DATASET

In [2]:

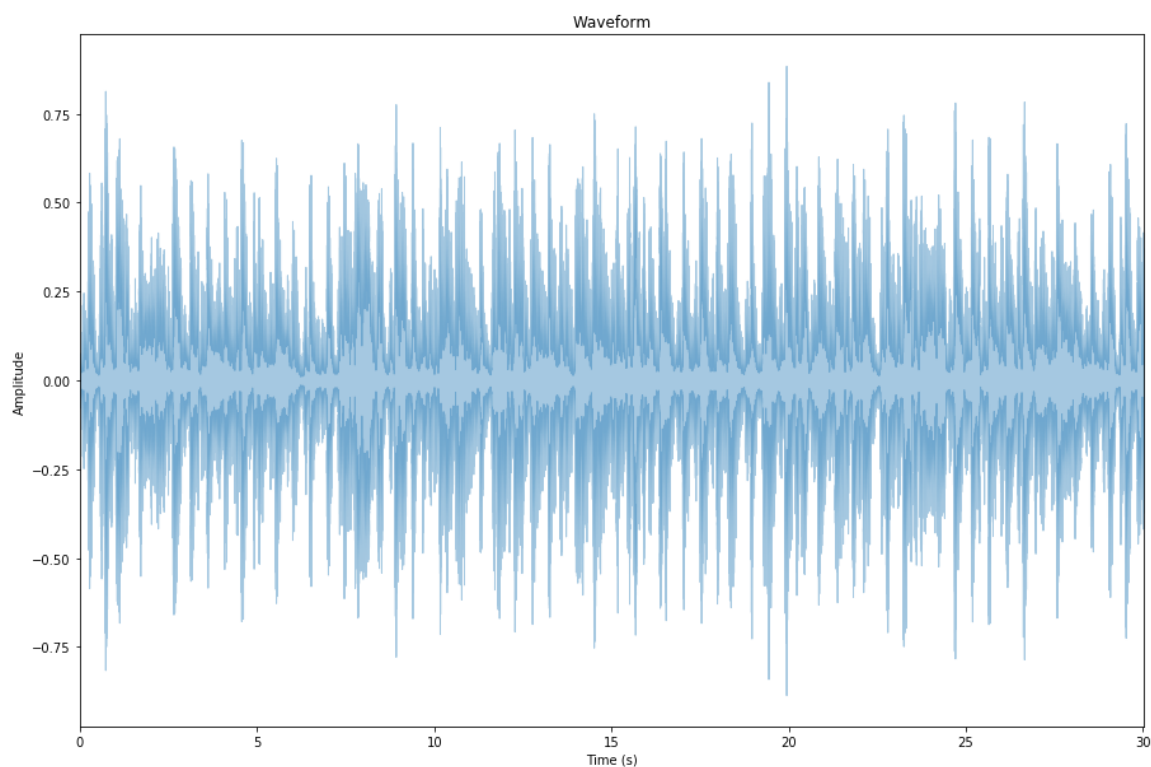
```
import numpy as np
import librosa, librosa.display
import matplotlib.pyplot as plt

FIG_SIZE = (15,10)

file = "Desktop/genres/blues/blues.00000.wav"

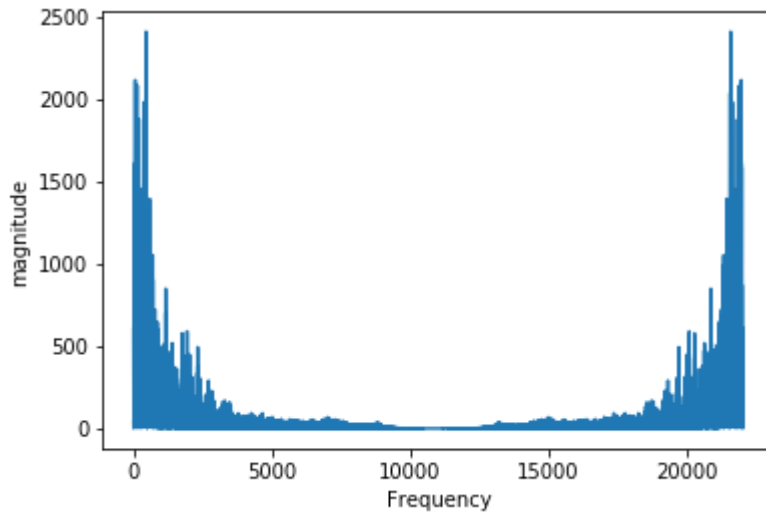
signal, sample_rate = librosa.load(file, sr=22050)

plt.figure(figsize=FIG_SIZE)
librosa.display.waveplot(signal, sample_rate, alpha=0.4)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.title("Waveform")
plt.show()
```



In [3]:

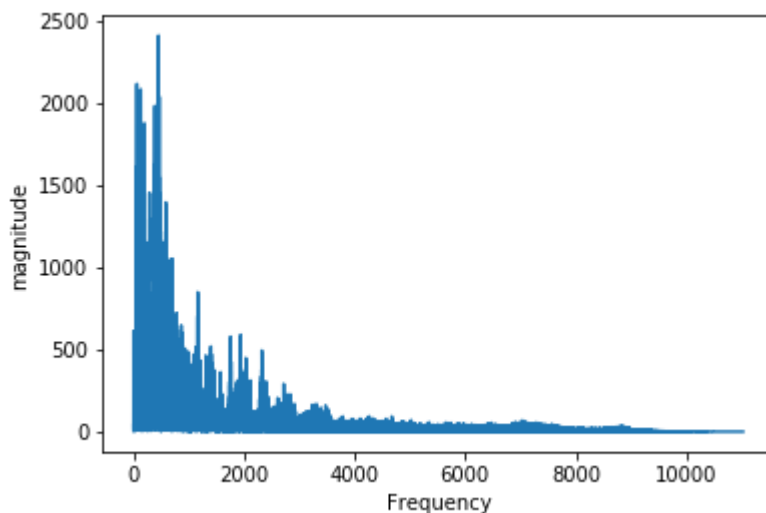
```
fft = np.fft.fft(signal)
magnitude=np.abs(fft)
frequency=np.linspace(0,sample_rate,len(magnitude))
plt.plot(frequency , magnitude)
plt.xlabel("Frequency")
plt.ylabel("magnitude")
plt.show()
```



In [4]:

```
left_frequency=frequency[:int(len(frequency)/2)]
left_magnitude=magnitude[:int(len(frequency)/2)]

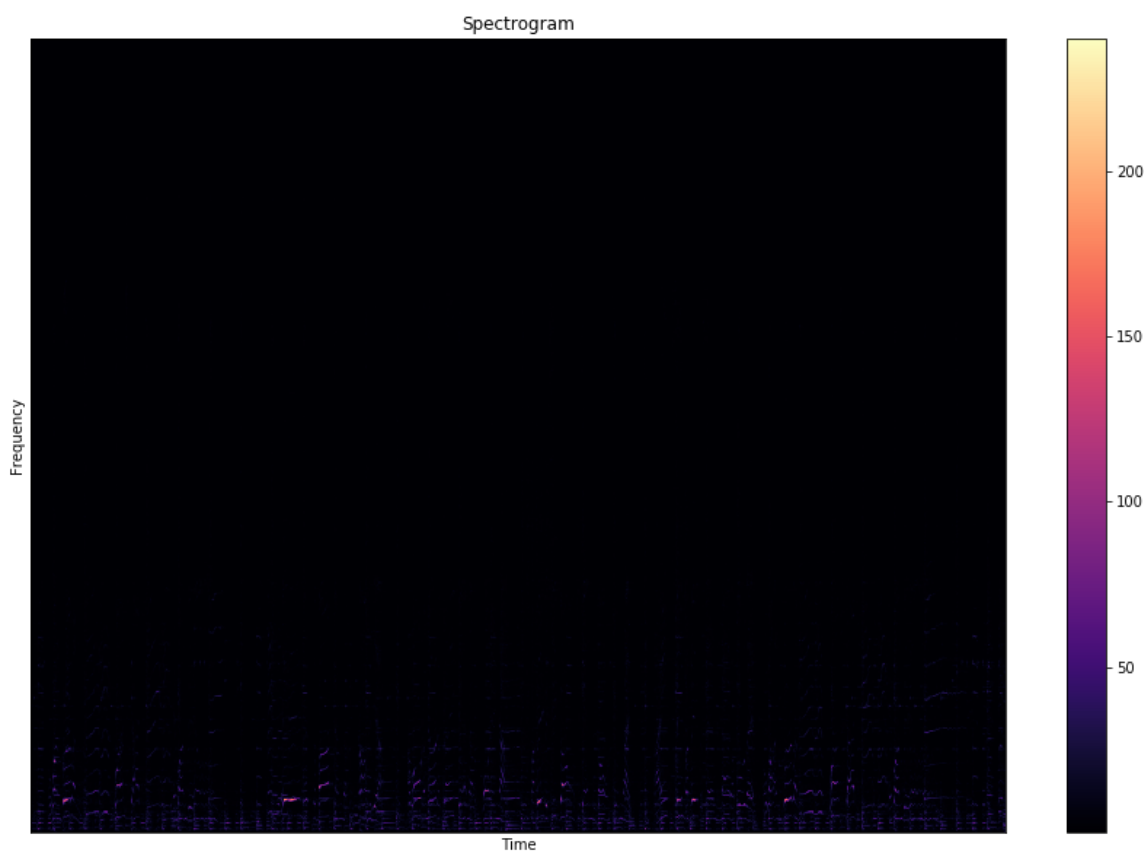
plt.plot(left_frequency , left_magnitude)
plt.xlabel("Frequency")
plt.ylabel("magnitude")
plt.show()
```



In [5]:

```
n_fft=2048
hop_length = 512
stft=librosa.core.stft(signal , n_fft=n_fft, hop_length=hop_length)
spectrogram = np.abs(stft)

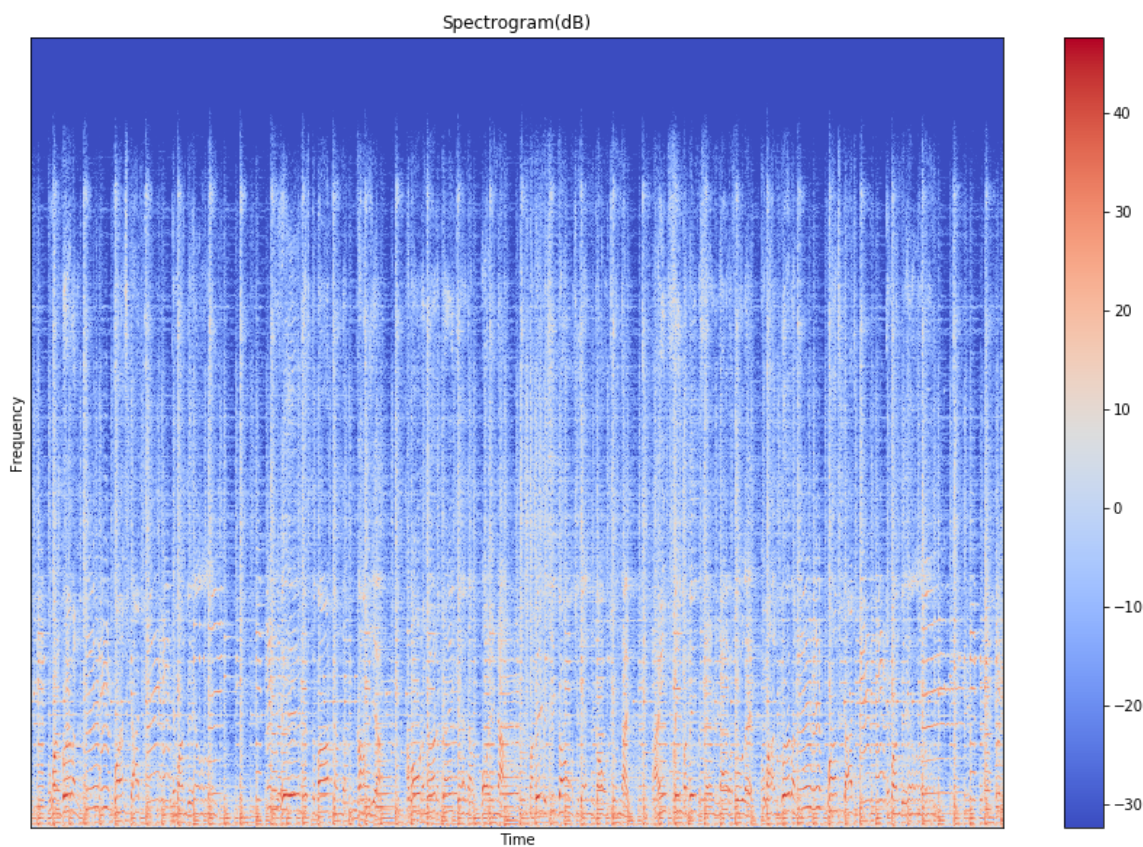
plt.figure(figsize=FIG_SIZE)
librosa.display.specshow(spectrogram, sr=sample_rate, hop_length=hop_length)
plt.xlabel("Time")
plt.ylabel("Frequency")
plt.colorbar()
plt.title("Spectrogram")
plt.show()
```



In [6]:

```
log_spectrogram = librosa.amplitude_to_db(spectrogram)

plt.figure(figsize=FIG_SIZE)
librosa.display.specshow(log_spectrogram, sr=sample_rate, hop_length=hop_length)
plt.xlabel("Time")
plt.ylabel("Frequency")
plt.colorbar()
plt.title("Spectrogram(dB)")
plt.show()
```

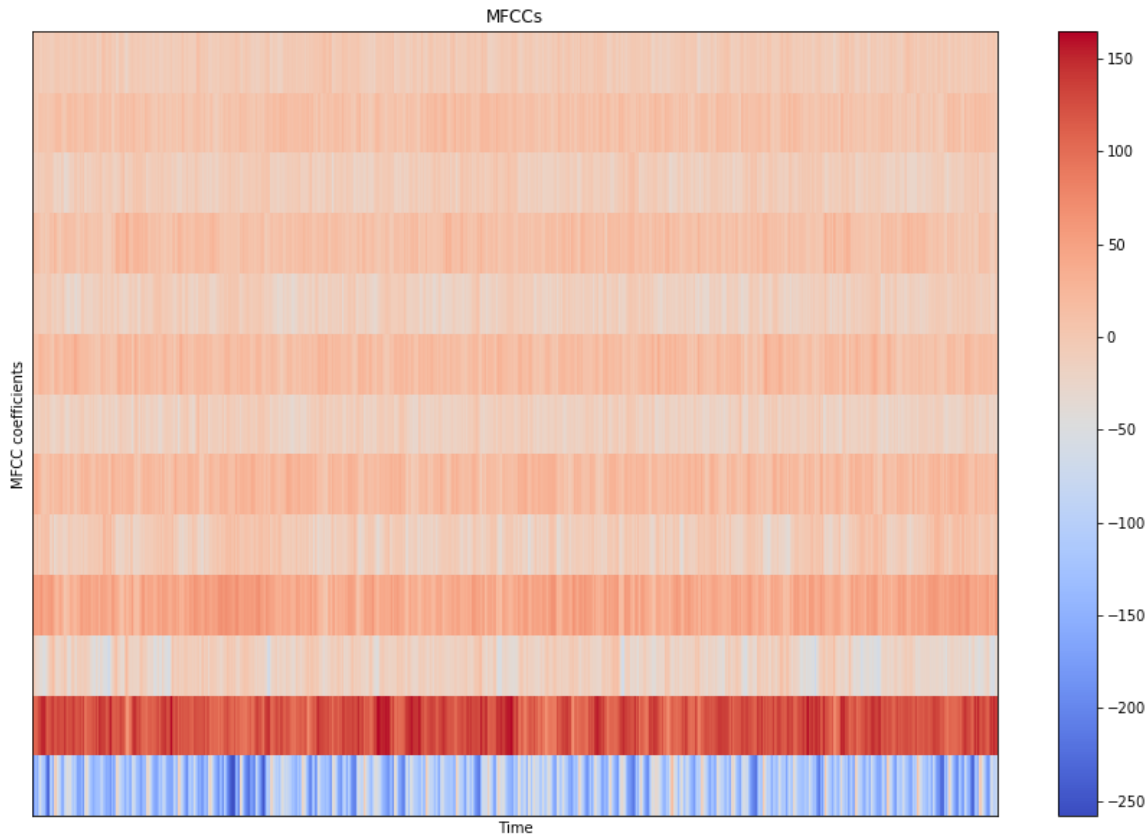


In [7]:

```
MFCCs = librosa.feature.mfcc(signal, sample_rate, n_fft=n_fft, hop_length=hop_length, n_mfcc=13)

plt.figure(figsize=FIG_SIZE)
librosa.display.specshow(MFCCs, sr=sample_rate, hop_length=hop_length)
plt.xlabel("Time")
plt.ylabel("MFCC coefficients")
plt.colorbar()
plt.title("MFCCs")

plt.show()
```



CREATING DATASET

In [8]:

```
import json
import os
import math
import librosa
```

In [9]:

```
DATASET_PATH = "Desktop/genres"
JSON_PATH = "data_10.json"
SAMPLE_RATE = 22050
TRACK_DURATION = 30
SAMPLES_PER_TRACK = SAMPLE_RATE * TRACK_DURATION

def save_mfcc(dataset_path, json_path, num_mfcc=13, n_fft=2048, hop_length=512, num_segments=5):

    data = {
        "mapping": [],
        "labels": [],
        "mfcc": []
    }

    samples_per_segment = int(SAMPLES_PER_TRACK / num_segments)
    num_mfcc_vectors_per_segment = math.ceil(samples_per_segment / hop_length)

    for i, (dirpath, dirnames, filenames) in enumerate(os.walk(dataset_path)):
        if dirpath is not dataset_path:
            semantic_label = dirpath.split("/")[-1]
            data["mapping"].append(semantic_label)
            print("\nProcessing: {}".format(semantic_label))
            for f in filenames:
                file_path = os.path.join(dirpath, f)
                signal, sample_rate = librosa.load(file_path, sr=SAMPLE_RATE)
                for d in range(num_segments):
                    start = samples_per_segment * d
                    finish = start + samples_per_segment
                    mfcc = librosa.feature.mfcc(signal[start:finish], sample_rate, n_mfcc=num_mfcc, n_fft=n_fft, hop_length=hop_length)
                    mfcc = mfcc.T

                    if len(mfcc) == num_mfcc_vectors_per_segment:
                        data["mfcc"].append(mfcc.tolist())
                        data["labels"].append(i-1)
                        print("{} , segment:{}".format(file_path, d+1))

    with open(json_path, "w") as fp:
        json.dump(data, fp, indent=4)

if __name__ == "__main__":
    save_mfcc(DATASET_PATH, JSON_PATH, num_segments=10)
```

Processing: genres\blues

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Desktop/genres\hiphop\hiphop.00099.wav, segment:10

```
Processing: genres\jazz
```

Desktop\genres\jazz\jazz.00005.wav, segment:6

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Desktop/genres/rock/rock.00098.wav, segment:10
Desktop/genres/rock/rock.00099.wav, segment:1
Desktop/genres/rock/rock.00099.wav, segment:2
Desktop/genres/rock/rock.00099.wav, segment:3
Desktop/genres/rock/rock.00099.wav, segment:4
Desktop/genres/rock/rock.00099.wav, segment:5
Desktop/genres/rock/rock.00099.wav, segment:6
Desktop/genres/rock/rock.00099.wav, segment:7
Desktop/genres/rock/rock.00099.wav, segment:8
Desktop/genres/rock/rock.00099.wav, segment:9
Desktop/genres/rock/rock.00099.wav, segment:10

Implementing MUSIC GENRE CLASSIFIER USING CNN

In [10]:

```
import json
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras
import matplotlib.pyplot as plt

DATA_PATH = "data_10.json"

def load_data(data_path):

    with open(data_path, "r") as fp:
        data = json.load(fp)

    X = np.array(data["mfcc"])
    y = np.array(data["labels"])
    return X, y

def plot_history(history):

    fig, axs = plt.subplots(2)

    axs[0].plot(history.history["accuracy"], label="train accuracy")
    axs[0].plot(history.history["val_accuracy"], label="test accuracy")
    axs[0].set_ylabel("Accuracy")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Accuracy eval")

    axs[1].plot(history.history["loss"], label="train error")
    axs[1].plot(history.history["val_loss"], label="test error")
    axs[1].set_ylabel("Error")
    axs[1].set_xlabel("Epoch")
    axs[1].legend(loc="upper right")
    axs[1].set_title("Error eval")

    plt.show()

def prepare_datasets(test_size, validation_size):
    X, y = load_data(DATA_PATH)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)
    X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train, test_size=validation_size)
    X_train = X_train[..., np.newaxis]
    X_validation = X_validation[..., np.newaxis]
    X_test = X_test[..., np.newaxis]
    return X_train, X_validation, X_test, y_train, y_validation, y_test

def build_model(input_shape):
    model = keras.Sequential()
    model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
```

```

model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Conv2D(32, (2, 2), activation='relu'))
model.add(keras.layers.MaxPooling2D((2, 2), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dropout(0.3))
model.add(keras.layers.Dense(10, activation='softmax'))
return model

```

```

def predict(model, X, y):
    X = X[np.newaxis, ...]
    prediction = model.predict(X)
    predicted_index = np.argmax(prediction, axis=1)
    print("Target: {}, Predicted label: {}".format(y, predicted_index))

```

```

if __name__ == "__main__":
    X_train, X_validation, X_test, y_train, y_validation, y_test = prepare_datasets(0.2
5, 0.2)
    input_shape = (X_train.shape[1], X_train.shape[2], 1)
    model = build_model(input_shape)
    optimiser = keras.optimizers.Adam(learning_rate=0.0001)
    model.compile(optimizer=optimiser,
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    model.summary()
    history = model.fit(X_train, y_train, validation_data=(X_validation, y_validation),
batch_size=32, epochs=30)
    plot_history(history)
    test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
    print('\nTest accuracy:', test_acc)

    X_to_predict = X_test[100]
    y_to_predict = y_test[100]

    predict(model, X_to_predict, y_to_predict)

```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 128, 11, 32)	320
max_pooling2d (MaxPooling2D)	(None, 64, 6, 32)	0
batch_normalization (Batch Normalization)	(None, 64, 6, 32)	128
conv2d_1 (Conv2D)	(None, 62, 4, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 31, 2, 32)	0
batch_normalization_1 (Batch Normalization)	(None, 31, 2, 32)	128
conv2d_2 (Conv2D)	(None, 30, 1, 32)	4128
max_pooling2d_2 (MaxPooling2D)	(None, 15, 1, 32)	0
batch_normalization_2 (Batch Normalization)	(None, 15, 1, 32)	128
flatten (Flatten)	(None, 480)	0
dense (Dense)	(None, 64)	30784
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 10)	650

=====

Total params: 45,514

Trainable params: 45,322

Non-trainable params: 192

=====

Train on 5997 samples, validate on 1500 samples

Epoch 1/30

5997/5997 [=====] - 47s 8ms/sample - loss: 2.4690
- accuracy: 0.2131 - val_loss: 1.9132 - val_accuracy: 0.3107

Epoch 2/30

5997/5997 [=====] - 8s 1ms/sample - loss: 1.8688
- accuracy: 0.3470 - val_loss: 1.6170 - val_accuracy: 0.4080

Epoch 3/30

5997/5997 [=====] - 8s 1ms/sample - loss: 1.6554
- accuracy: 0.4124 - val_loss: 1.4692 - val_accuracy: 0.4493

Epoch 4/30

5997/5997 [=====] - 9s 1ms/sample - loss: 1.5196
- accuracy: 0.4524 - val_loss: 1.3394 - val_accuracy: 0.5067

Epoch 5/30

5997/5997 [=====] - 10s 2ms/sample - loss: 1.4279
- accuracy: 0.4907 - val_loss: 1.2738 - val_accuracy: 0.5287

Epoch 6/30

5997/5997 [=====] - 9s 1ms/sample - loss: 1.3614
- accuracy: 0.5156 - val_loss: 1.2386 - val_accuracy: 0.5367

Epoch 7/30

5997/5997 [=====] - 8s 1ms/sample - loss: 1.3086
- accuracy: 0.5314 - val_loss: 1.1718 - val_accuracy: 0.5760

Epoch 8/30

5997/5997 [=====] - 8s 1ms/sample - loss: 1.2487
- accuracy: 0.5538 - val_loss: 1.1445 - val_accuracy: 0.5760

Epoch 9/30

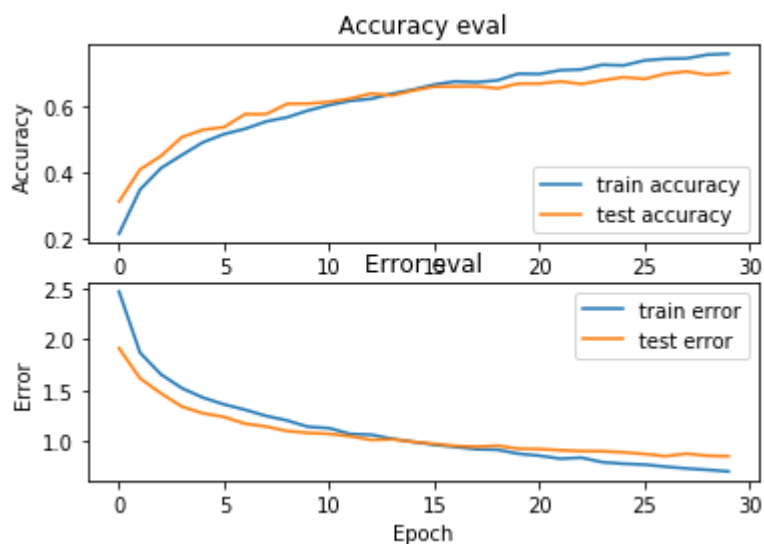
5997/5997 [=====] - 9s 1ms/sample - loss: 1.2038

- accuracy: 0.5664 - val_loss: 1.1007 - val_accuracy: 0.6073
Epoch 10/30
5997/5997 [=====] - 9s 1ms/sample - loss: 1.1417
- accuracy: 0.5871 - val_loss: 1.0822 - val_accuracy: 0.6080
Epoch 11/30
5997/5997 [=====] - 9s 1ms/sample - loss: 1.1285
- accuracy: 0.6036 - val_loss: 1.0728 - val_accuracy: 0.6133
Epoch 12/30
5997/5997 [=====] - 9s 1ms/sample - loss: 1.0713
- accuracy: 0.6163 - val_loss: 1.0494 - val_accuracy: 0.6233
Epoch 13/30
5997/5997 [=====] - 9s 1ms/sample - loss: 1.0641
- accuracy: 0.6225 - val_loss: 1.0131 - val_accuracy: 0.6387
Epoch 14/30
5997/5997 [=====] - 9s 1ms/sample - loss: 1.0246
- accuracy: 0.6383 - val_loss: 1.0192 - val_accuracy: 0.6327
Epoch 15/30
5997/5997 [=====] - 9s 1ms/sample - loss: 0.9942
- accuracy: 0.6505 - val_loss: 0.9942 - val_accuracy: 0.6480
Epoch 16/30
5997/5997 [=====] - 9s 1ms/sample - loss: 0.9650
- accuracy: 0.6657 - val_loss: 0.9745 - val_accuracy: 0.6593
Epoch 17/30
5997/5997 [=====] - 9s 1ms/sample - loss: 0.9442
- accuracy: 0.6752 - val_loss: 0.9544 - val_accuracy: 0.6600
Epoch 18/30
5997/5997 [=====] - 9s 2ms/sample - loss: 0.9221
- accuracy: 0.6730 - val_loss: 0.9465 - val_accuracy: 0.6607
Epoch 19/30
5997/5997 [=====] - 9s 2ms/sample - loss: 0.9168
- accuracy: 0.6783 - val_loss: 0.9565 - val_accuracy: 0.6540
Epoch 20/30
5997/5997 [=====] - 9s 1ms/sample - loss: 0.8779
- accuracy: 0.6987 - val_loss: 0.9265 - val_accuracy: 0.6687
Epoch 21/30
5997/5997 [=====] - 9s 1ms/sample - loss: 0.8589
- accuracy: 0.6977 - val_loss: 0.9235 - val_accuracy: 0.6687
Epoch 22/30
5997/5997 [=====] - 9s 1ms/sample - loss: 0.8290
- accuracy: 0.7092 - val_loss: 0.9120 - val_accuracy: 0.6753
Epoch 23/30
5997/5997 [=====] - 9s 1ms/sample - loss: 0.8392
- accuracy: 0.7115 - val_loss: 0.9037 - val_accuracy: 0.6673
Epoch 24/30
5997/5997 [=====] - 9s 1ms/sample - loss: 0.7951
- accuracy: 0.7259 - val_loss: 0.9024 - val_accuracy: 0.6787
Epoch 25/30
5997/5997 [=====] - 9s 1ms/sample - loss: 0.7799
- accuracy: 0.7237 - val_loss: 0.8916 - val_accuracy: 0.6880
Epoch 26/30
5997/5997 [=====] - 9s 1ms/sample - loss: 0.7714
- accuracy: 0.7389 - val_loss: 0.8756 - val_accuracy: 0.6827
Epoch 27/30
5997/5997 [=====] - 9s 2ms/sample - loss: 0.7509
- accuracy: 0.7439 - val_loss: 0.8538 - val_accuracy: 0.6987
Epoch 28/30
5997/5997 [=====] - 9s 2ms/sample - loss: 0.7334
- accuracy: 0.7450 - val_loss: 0.8786 - val_accuracy: 0.7053
Epoch 29/30
5997/5997 [=====] - 9s 1ms/sample - loss: 0.7198
- accuracy: 0.7564 - val_loss: 0.8576 - val_accuracy: 0.6953

Epoch 30/30

5997/5997 [=====] - 9s 2ms/sample - loss: 0.7041

- accuracy: 0.7590 - val_loss: 0.8536 - val_accuracy: 0.7013



2499/1 - 1s - loss: 1.2033 - accuracy: 0.6775

Test accuracy: 0.677471

Target: 8, Predicted label: [8]

In []: