

Final Project Assignment #2:

- Getting datasets ready for DL in Google Colab

In this round, you will **organize and systematize** your deep learning workflow in **Google Colab**.

Since you have already selected **three datasets (each with different modalities)**, your goal now is to prepare them for deep learning experimentation throughout the semester.

1. Group ID :_sp25_EtoKuySon

2. File naming convention:

- SMS Spam Detection (Text Dataset) : `sp25_EtoKuySon_SMS_prep.ipynb`
- 911 Recording (Audio Dataset) : `sp25_EtoKuySon_911_prep.ipynb`
- CatsVsDogs (Image Dataset) : `sp25_EtoKuySon_CatsVsDogs_prep.ipynb`

1. SMS Spam Detection (Text Dataset) :

Introduction:

The SMS Spam Detection dataset helps build a model to automatically classify text messages as either spam or ham. Spam messages are unwanted or malicious, while ham messages are legitimate. Detecting spam is important for improving user experience and security.

Dataset Overview:

The SMS Spam Detection dataset consists of 5,572 rows and 2 columns. The SMS Spam Detection dataset consists of text messages, each labeled as either "ham" (non-spam) or "spam." Each message is represented as a string of text, which the model will analyze to identify patterns and classify the message correctly.

Target Label (V1):

Spam: Unsolicited or unwanted messages, typically advertisements, phishing attempts, or scams.

Ham: Legitimate, non-spam messages that are genuine and not unsolicited.

```
# Reading the Csv file

sms_csv_path = sms_path / "spam_sms.csv"
sms_csv = pd.read_csv(sms_csv_path)
sms_csv.head()
```

	v1	v2
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Checking for Null Values

Checked for missing (null) values in the dataset by summing the null values for each column.

```
# Checking for Null Values
print(sms_csv.isnull().sum())
```

v1	0
v2	0

dtype: int64

Checking for Duplicates

Checked for duplicate rows in the dataset by counting how many duplicates exist.

```
# Checking for duplicates
print("Number of Duplicates: ", sms_csv.duplicated().sum())
```

Number of Duplicates: 403

Dropping Duplicates

Removed duplicate rows, keeping only the first occurrence of each.

```
[ ] # Dropping the duplicates
sms_csv.drop_duplicates(keep= 'first', inplace=True)

# Printing the new dimensions
sms_csv.shape

(5169, 2)
```

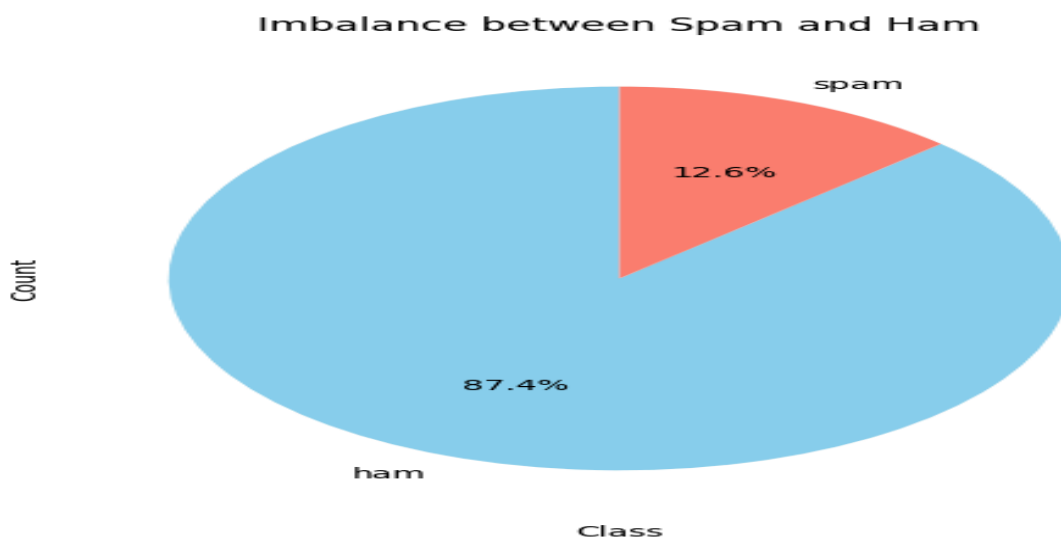
Checking for Class Imbalance

Checked the distribution of the target variable (v1) to see if there is an imbalance between the classes (spam and ham).

```
# Checking for imbalance
sms_csv['v1'].value_counts()

count
v1
ham    4516
spam    653
dtype: int64
```

Plotting the Class Imbalance



Visualized the class distribution in a pie chart to see the imbalance between spam and ham messages.

Used Undersampling to Balance the Data

Splitted the dataset into features (X) and target (y), then applied **Random Undersampling** to balance the data by reducing the number of spam or ham messages in the dataset.

```
# Using undersampling to balance the data

X = sms_csv.drop(columns = ['v1'])
y = sms_csv['v1']

rus_sms = RandomUnderSampler(random_state=42)
X_res, y_res = rus_sms.fit_resample(X, y)

print('Resampled dataset shape %s' % Counter(y_res))
```

Resampled dataset shape Counter({'ham': 653, 'spam': 653})

Resampled Dataset Shape

Printed the class distribution of the resampled dataset to confirm that the classes are now balanced.


Renamed Ham to '0' and Spam to '1'

Mapped the target labels ("ham" and "spam") to numerical values (0 and 1, respectively) for easier processing in machine learning models.

```
[ ] # Renaming ham to '0' and spam to '1'
y_res = y_res.map({'ham': 0, 'spam': 1})
```


Converting Text Data to Numerical Data using TF-IDF

Used **TF-IDF Vectorization** to convert the text data into numerical values that can be processed by machine learning models.

```
 # Since X_res_values is a text, we using TD-IDF to covert it into numerical data  
from sklearn.feature_extraction.text import TfidfVectorizer  
  
X_res_values = X_res.values  
  
vectorizer = TfidfVectorizer()  
X_res_numerical = vectorizer.fit_transform(X_res_values.ravel()).toarray()
```

Converting the Data to PyTorch Tensors

Converted the feature (X_res_numerical) and target (y_res_values) data into **PyTorch tensors** for use in PyTorch-based models.

```
[ ] # Converting to pytorch tensor (X_res)  
X_res_tensor = torch.tensor(X_res_numerical, dtype=torch.float32)  
  
[ ] #Converting to pytorch tensor (y_res)  
  
y_res_values = y_res.values  
  
y_res_tensor = torch.tensor(y_res_values, dtype=torch.float32)  
  
[ ] print("Shape of X_res_tensor:", X_res_tensor.shape)  
    print("Shape of y_res_tensor:", y_res_tensor.shape)  
  
 Shape of X_res_tensor: torch.Size([1306, 4383])  
    Shape of y_res_tensor: torch.Size([1306])
```

What I Learned: Tensors are the main way to store and process data in deep learning. They are like NumPy arrays, but PyTorch uses them to work efficiently with GPUs, which helps speed up computations.

Converting to a Tensor: I learned that text data (like SMS messages) needs to be converted into a tensor format so it can be used in a machine learning model.

Why It's Important: This conversion is essential because it allows the model to process the data correctly, making it easier for the neural network to learn from the data and make accurate predictions.

2. 911 Recording(Audio Dataset):

Introduction and dataset overview:

The 911 Recordings dataset contains real emergency call recordings collected by Gary Allen, former editor of Dispatch Monthly magazine, from public sources. These calls were retrieved from an Internet Archive capture of the now-defunct 911 Dispatch website and manually labeled based on descriptions, recordings, and news reports. The dataset includes audio recordings (MP3 format), brief text descriptions, and metadata such as date, location, citizen-initiated status, and emergency outcomes (e.g., deaths, false alarms). Some recordings may be censored, incomplete, or contain multiple calls. This dataset is valuable for building a machine learning model to classify calls as critical or unusual, helping emergency services prioritize urgent situations and improve response times.

Objective:

The goal of this project is to build a machine learning model that can classify 911 calls as either critical or unusual. This will help emergency responders quickly identify urgent calls, prioritize resources, and improve response times. By analyzing past call patterns, the model will support faster decision-making and better emergency management.

Data loading and data cleaning:

```
[ ] import librosa
import numpy as np
import os

[ ] # Loading dataset directly from kaggle Hub
Calls_911 = kagglehub.dataset_download("louisteitelbaum/911-recordings")

Warning: Looks like you're using an outdated `kagglehub` version (installed: 0.3.7), please consider upgrading to the latest version (0.3.8).

[ ] print(Calls_911)

/root/.cache/kagglehub/datasets/louisteitelbaum/911-recordings/versions/1

[ ] audio_data = os.path.join(Calls_911, "911-recordings")

[ ] if not os.path.exists(audio_data):
    print("Error : The directory does not exist",audio_data)
else:
    print("The directory exists",audio_data)

# Iterate through MP3 files
for audio_file in os.listdir(audio_data):
    audio_path = os.path.join(audio_data, audio_file)

# Checking if all files ends with .mp3 extension
if os.path.isfile(audio_path) and audio_path.endswith(".mp3"):
    try:
        audio, sr = librosa.load(audio_path, sr = None)
        print("Loaded Audio file:", audio_path)
        print("Audio Duration (in sec):", librosa.get_duration(y=audio, sr=sr))
        print("Sample Rate:", sr)
        print("-"*50)
    except Exception as e:
        print(f"Error loading audio file {audio_path}: {e}")

[ ] # Audio files are of different sampling rate, therefore resampling it
y_resampled = librosa.resample(audio, orig_sr=sr, target_sr=16000)

[ ] # Removing leading and trailing white nosie/ silence
y_trim, _ = librosa.effects.trim(y_resampled)

[ ] # Normalizing the data
y_normal = y_trim / np.max(np.abs(y_trim))
```

The script first verifies whether the dataset folder exists before proceeding with data processing. Each audio file, stored in MP3 format, is iterated through, ensuring only valid files are processed. The **librosa** library is used to load the audio while preserving the original sampling rate. Key details such as file name, duration, and sampling rate are extracted and displayed to confirm successful loading. Since the recordings have varying sampling rates, all files are resampled to 16,000 Hz for consistency. Additionally, leading and trailing silence is removed to eliminate unnecessary pauses, ensuring the dataset focuses only on relevant speech data. Finally, the audio is normalized so that its amplitude is within a standard range, preventing volume inconsistencies. This preprocessing step ensures that the dataset is clean, uniform, and ready for feature extraction and model training, improving the reliability of the classification model.

Conversion to Tensor Format

```
# Converting to tensor format
import torch
import torchaudio

mfccs_tensor = torch.from_numpy(mfccs)

print(mfccs_tensor)
print(mfccs_tensor.shape)
print(mfccs_tensor.dtype)
```

MFCCs: These are features extracted from the audio that represent the sound in a way that makes it easier for a model to understand.

Conversion to Tensor: `torch.from_numpy(mfccs)` takes the MFCCs (which are in a NumPy array) and converts them into a PyTorch tensor. This is necessary because deep learning models in PyTorch need the data in tensor format.

Tensor Data: This prints the actual values stored in the tensor.

Shape: `mfccs_tensor.shape` shows the size of the tensor. For example, it might tell you how many time frames (like seconds of audio) and MFCC coefficients (the features) are in the data.

Data Type: `mfccs_tensor.dtype` tells you the type of numbers used in the tensor (usually decimal numbers, like float32), which is important for deep learning calculations.

Tensors are the main data structure used in deep learning. They're like NumPy arrays, but PyTorch uses them to make the data ready for training on GPUs, which speeds things up.

Converting to a Tensor: I learned that audio data needs to be converted into a tensor format so that it can be used by a deep learning model.

Why It's Important: This conversion is necessary for training the model, as it helps the data get processed in the right way for the neural network to learn and make predictions.

3. CatsVsDogs (Image Dataset) :

Introduction

This project aims to build a model that can tell the difference between cats and dogs using images. It is a type of classification problem where the model looks at a picture and predicts whether it shows a cat or a dog. By training the model on many labeled images, it will learn to recognize features like fur patterns, ear shapes, and body structure to correctly classify the animals. This technology is useful for pet identification, security systems, and even mobile apps that recognize pets.

Dataset Overview

The dataset used for this project consists of labeled images of cats and dogs. It contains 25,000 labeled images (12,500 images of cats and 12,500 images of dogs).

Data loading, Data Cleaning, Data Pre processing, Conversion to Tensor Format

Data Preprocessing

```
path = cats_dogs
img_size = 224

# Data Preprocessing
transform = transforms.Compose([
    transforms.Resize((img_size, img_size)), # Resizing
    transforms.ToTensor(), # Transforming to sensor
    transforms.Normalize(mean = [0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

data = datasets.ImageFolder(root = cats_dogs / subfolders, transform = transform)
print(data)
```

A series of transformations were applied to each image to prepare it for model input:

Resize: All images were resized to a uniform size of 224x224 pixels using transforms.

ToTensor: The ToTensor() transformation converts the images from PIL Image or NumPy arrays into PyTorch tensors, making them compatible with neural networks.

Normalize: The transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) normalized the images, ensuring that the pixel values are in the same range as the pre-trained models (such as ResNet), which can help improve performance during model training.

Corrupted Image Handling

```
# Check for corrupted images
def is_image_corrupted(image_path):
    try:
        with Image.open(image_path) as img:
            img.verify()
        return False
    except (IOError, SyntaxError) as e:
        return True

def corrupted_image_check():
    corrupted_image = []
    for path, _ in data.samples:
        if is_image_corrupted(path):
            corrupted_image.append(path)

    if corrupted_image:
        print("Total number of corrupted images:", len(corrupted_image))
        for image_path in corrupted_image:
            os.remove(image_path)
            print("Corrupted images removed")
    else:
        print("No corrupted images found")
```

Before training the model, it is crucial to ensure that all images are in a valid format or not. The function `is_image_corrupted` checks whether an image can be opened and verified by attempting to read it. If an image is corrupted, it is removed from the dataset. The `corrupted_image_check` function scans all images in the dataset, identifies any corrupted ones, and deletes them. If there are no corrupted images, it will print that the dataset is clean.

Label Count and Dataset Distribution Random Image Selection and Visualization

```

# Checking if the data is Balanced by seeing the label count in each image folder

label_counts = Counter(data.targets)
print("Label Count : ")

for label, count in label_counts.items():
    print(f"Classes of {data.classes[label]}: {count} images")

Label Count :
Classes of Cat: 12500 images
Classes of Dog: 12500 images

# Selecting Random number of images from both classes and displaying it

random_number = 10

#Randomly selecting 10 images from both classes
rnd_images = random.sample(range(len(data)), random_number)

#Displaying the images
plt.figure(figsize=(10, 10))

for i, f in enumerate(rnd_images):
    img, classes = data[f]
    plt.subplot(5, 5, i+1)
    plt.imshow(img.permute(1, 2, 0))
    plt.title(data.classes[classes])
    plt.axis('off')
    plt.title(data.classes[classes])

plt.show()

```

To ensure that the dataset is balanced, I checked the distribution of images in each class using `Counter(data.targets)`. This shows how many images exist for each class (cats and dogs). I randomly selected 10 images from the dataset to visually inspect the content.

What I Learned : Tensors are the main way to store and process data in deep learning. They are like arrays but optimized for GPU use, which speeds up computations. In this project, images of cats and dogs need to be converted into tensors to be processed by the model.

Converting to a Tensor : I learned that images of cats and dogs need to be converted into tensor format using `transforms.ToTensor()`. This step is necessary because the model can only process data in tensor form, allowing it to learn patterns in the images.

Why It's Important : Converting images into tensors is important because it allows the model to process the image data and make accurate predictions. Without this conversion, the model wouldn't be able to handle images effectively, and training wouldn't be possible. Using tensors also ensures that the model can perform computations quickly on a GPU.

GitHub repo : <https://github.com/LokeshKuyyamudiBharath/DeepLearning5305>