ITAI 2377 Lab 04: Deep Learning Data Preprocessing

Instructor: [Hardik Gohel] **Date:** [September 29, 2025]

Introduction

Welcome to Lab 04! We'll explore the critical role of data preprocessing in deep learning. Even though models can extract features, preprocessing is essential for optimal performance. We'll cover various data types and apply preprocessing techniques. Resources in Google Colab are limited, so efficient coding is key!

Why Preprocess?

Why preprocess when models extract features?

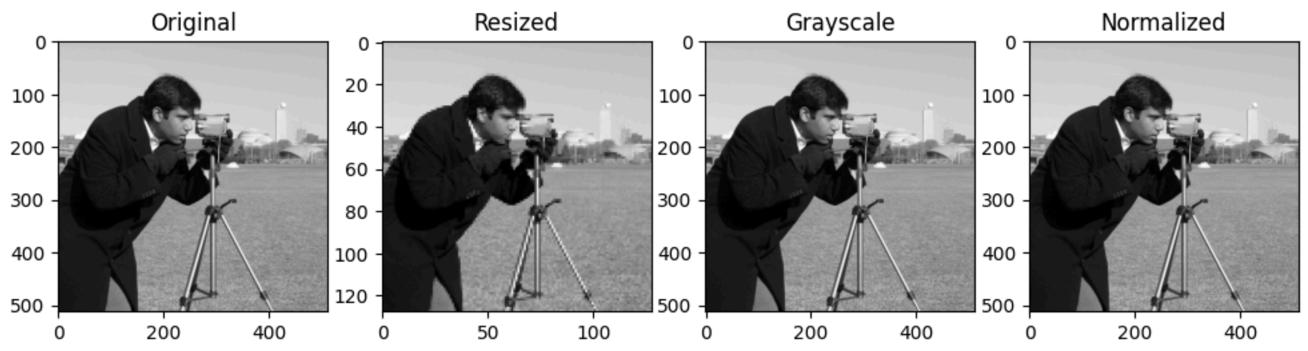
- **Standardization:** Models need consistent data formats and ranges.
- **Noise/Errors:** Raw data is messy. Preprocessing cleans it up.
- **Efficiency:** Cleaner data means faster training.
- **Results:** Good preprocessing helps models perform their best.

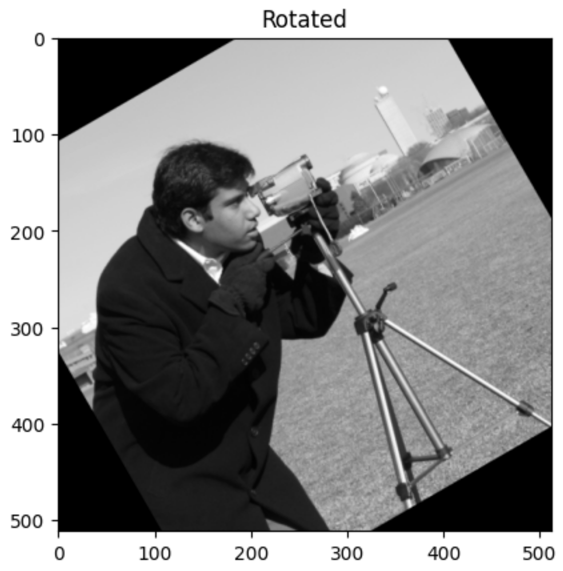
Think of preprocessing as a personal trainer for your model.

Data Types and Preprocessing Techniques

1. Image Data

```python
import cv2
import numpy as np
from skimage import data, img_as_float
import matplotlib.pyplot as plt

# Load a sample image (replace with your image path if you have one)
image = data.camera()  # Or use: image = cv2.imread("path/to/your/image.jpg")

# Resizing (Student Code: Resize the image to (128, 128))
# Hint: Use cv2.resize()
# YOUR CODE HERE
resized_image = cv2.resize(image, (128, 128))


# Color space conversion (to grayscale)
# Check if the image is already grayscale before attempting conversion
if len(image.shape) == 3:
    gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
else:
    gray_image = image # Image is already grayscale

# Normalization (pixel values 0-1)
normalized_image = img_as_float(image)

# Display images
plt.figure(figsize=(12, 6))

plt.subplot(1, 4, 1), plt.imshow(image, cmap='gray'), plt.title("Original") # Add cmap='gray' for grayscale images
plt.subplot(1, 4, 2), plt.imshow(resized_image, cmap='gray'), plt.title("Resized") # Add cmap='gray' for grayscale images
plt.subplot(1, 4, 3), plt.imshow(gray_image, cmap='gray'), plt.title("Grayscale")
plt.subplot(1, 4, 4), plt.imshow(normalized_image, cmap='gray'), plt.title("Normalized") # Add cmap='gray' for grayscale images
plt.show()

# Data Augmentation (rotation - Student Code: Rotate by 30 degrees)
# Hint: Use cv2.getRotationMatrix2D and cv2.warpAffine
angle = 30 #YOUR CODE HERE
rows, cols = image.shape[:2]
M = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
rotated_image = cv2.warpAffine(image, M, (cols, rows))

plt.imshow(rotated_image, cmap='gray'), plt.title("Rotated") # Add cmap='gray' for grayscale images
plt.show()
```





2. Text Data

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import string

nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)
nltk.download('wordnet', quiet=True)
nltk.download('punkt_tab', quiet=True) # Add this line to download the missing resource

text = "This is a fun example sentence with stop words and punctuation!"

# Tokenization (lowercase)
tokens = word_tokenize(text.lower())

# Stop word removal (Student Code: Remove stop words and punctuation)
# Hint: Use the 'stop_words' set and list comprehension
stop_words = set(stopwords.words('english'))
# YOUR CODE HERE
filtered_tokens = [w for w in tokens if not w in stop_words and w not in string.punctuation]


# Lemmatization (Student Code: Lemmatize the filtered tokens)
# Hint: Use WordNetLemmatizer()
lemmatizer = WordNetLemmatizer()
# YOUR CODE HERE
lemmatized_tokens = [lemmatizer.lemmatize(w) for w in filtered_tokens]


print("Original:", text)
print("Tokens:", tokens)
print("Filtered:", filtered_tokens)
print("Lemmatized:", lemmatized_tokens)
```

```
Original: This is a fun example sentence with stop words and punctuation!
Tokens: ['this', 'is', 'a', 'fun', 'example', 'sentence', 'with', 'stop', 'words', 'and', 'punctuation', '!']
Filtered: ['fun', 'example', 'sentence', 'stop', 'words', 'punctuation']
Lemmatized: ['fun', 'example', 'sentence', 'stop', 'word', 'punctuation']
```

3. Time Series Data

```
import pandas as pd
import numpy as np

# Sample data (with a missing value)
data = {'Date': pd.to_datetime(['2024-01-01', '2024-01-02', '2024-01-03', '2024-01-04', '2024-01-05']),
        'Value': [10, 12, 15, np.nan, 18]}

df = pd.DataFrame(data)

# Missing value handling (Student Code: Use backward fill to fill missing values)
# Hint: Use fillna() with method='bfill'
# YOUR CODE HERE
df['Value'] = df['Value'].fillna(method='bfill')

# Display the DataFrame after missing value handling but before normalization
display(df)

# Normalization (min-max scaling - Student Code: Normalize the 'Value' column)
# YOUR CODE HERE
min_val = df['Value'].min()
max_val = df['Value'].max()
df['Normalized'] = (df['Value'] - min_val) / (max_val - min_val)

print(df)
```

```
/tmp/ipython-input-3433131238.py:13: FutureWarning: Series.fillna with 'method' is deprecated and will raise in a future version. U
  df['Value'] = df['Value'].fillna(method='bfill')
```

	Date	Value
0	2024-01-01	10.0
1	2024-01-02	12.0
2	2024-01-03	15.0
3	2024-01-04	18.0
4	2024-01-05	18.0

```
        Date  Value  Normalized
0 2024-01-01   10.0       0.000
1 2024-01-02   12.0       0.250
2 2024-01-03   15.0       0.625
3 2024-01-04   18.0       1.000
4 2024-01-05   18.0       1.000
```

Next steps: Generate code with df New interactive sheet

4. Optional: Video Data (Simplified)

```
# --- Setup: install/upgrade tools ---
!pip -q uninstall -y pytube >/dev/null 2>&1
!pip -q install --no-cache-dir -U pytube yt-dlp >/dev/null

import os, subprocess, sys
import cv2
import matplotlib.pyplot as plt

URL = "https://www.youtube.com/watch?v=qaCoOINxNFk"
OUTPUT = "video.mp4"

# --- Try pytube first ---
downloaded = False
try:
    from pytube import YouTube
    yt = YouTube(URL)
    stream = yt.streams.get_highest_resolution()
    stream.download(filename=OUTPUT)
```

```
        downloaded = os.path.exists(OUTPUT)
        print("✅ pytube download complete:", OUTPUT)
    except Exception as e:
        print("⚠️ pytube failed:", repr(e))

    # --- Fallback to yt-dlp if needed ---
    if not downloaded:
        print("➡️ Falling back to yt-dlp…")
        # Prefer 720p progressive if available, otherwise best
        cmd = [
            "yt-dlp",
            "-f", "bestvideo[height<=720]+bestaudio/best",
            "-o", OUTPUT,
            URL,
            "--merge-output-format", "mp4",
            "--quiet", "--no-warnings"
        ]
        rc = subprocess.call(cmd)
        downloaded = os.path.exists(OUTPUT) and os.path.getsize(OUTPUT) > 0
        if downloaded:
            print("✅ yt-dlp download complete:", OUTPUT)
        else:
            raise RuntimeError("❌ Could not download the video with pytube or yt-dlp.")

    # --- Open with OpenCV and show a grayscale frame ---
    cap = cv2.VideoCapture(OUTPUT)
    if not cap.isOpened():
        raise RuntimeError("❌ Error opening downloaded video.")

    ret, frame = cap.read()
    cap.release()

    if ret:
        resized = cv2.resize(frame, (80, 60))
        gray = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)
        plt.imshow(gray, cmap='gray')
        plt.title("First Frame (Grayscale)")
        plt.axis('off')
        plt.show()
    else:
        raise RuntimeError("❌ Couldn't read a frame from the video.")
```

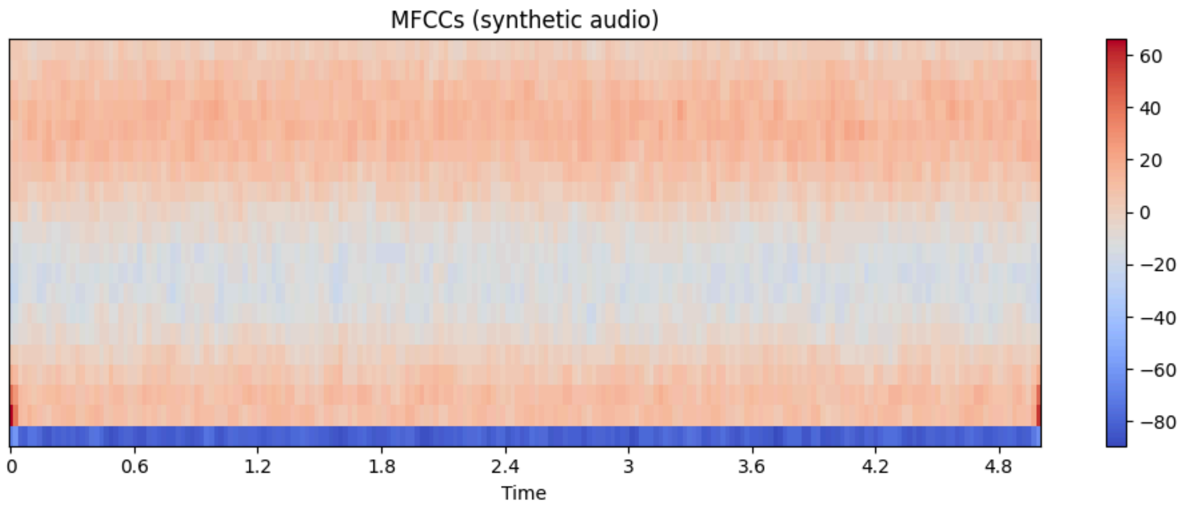
```
⚠️ pytube failed: <HTTPError 400: 'Bad Request'>
➡️ Falling back to yt-dlp…
✅ yt-dlp download complete: video.mp4
```

First Frame (Grayscale)

5. Optional: Audio Data (Simplified)

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

```
# --- Generate synthetic audio (sine wave + noise) if no file available ---
sr = 22050  # Sampling rate
duration = 5  # seconds
t = np.linspace(0, duration, int(sr*duration), endpoint=False)

# 440 Hz sine wave (musical note A4) + some noise
y = 0.5*np.sin(2*np.pi*440*t) + 0.05*np.random.randn(len(t))

# --- Extract MFCCs (this was YOUR CODE HERE) ---
mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=20)

# --- Display MFCCs ---
plt.figure(figsize=(10, 4))
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
plt.colorbar()
plt.title('MFCCs (synthetic audio)')
plt.tight_layout()
plt.show()

# --- Normalize MFCCs ---
mfccs_normalized = (mfccs - np.mean(mfccs)) / np.std(mfccs)

print("MFCCs shape:", mfccs.shape)
print("Normalized MFCCs shape:", mfccs_normalized.shape)
```



```
MFCCs shape: (20, 216)
Normalized MFCCs shape: (20, 216)
```

Sources

1. https://towardsdatascience.com/text-normalization-with-spacy-and-nltk-1302ff430119

Tip: Explore different methods for handling missing values (e.g., backward fill, interpolation). Consider feature engineering techniques like creating lagged variables.

Questions (Markdown Cell)

1. Why is data preprocessing still important even with deep learning's feature extraction capabilities? Although deep learning is capable of feature extraction, raw data frequently contains missing values, noise, or inconsistent formats. Preprocessing guarantees that the model identifies meaningful patterns rather than errors
2. Explain the difference between normalization and standardization. When would you choose one over the other? Normalization* adjusts values to a specific range (exemple, [0,1]), which is beneficial for image pixels or bounded inputs.

- Standardization re-centers data to a mean of 0 and a standard deviation of 1, making it more suitable for Gaussian-like distributions. The choice between the two methods is contingent upon the dataset and the requirements of the model

3. Describe a scenario where data augmentation would be particularly useful. Data augmentation is particularly useful in small datasets that are susceptible to overfitting, such as in medical imaging. Techniques like flipping or rotating images enhance diversity and bolster generalization.

4. What are some potential challenges or pitfalls to avoid during data preprocessing? How can you mitigate them? Data leakage → maintain separation between training and testing datasets.

- Overprocessing → validate preprocessing steps against performance metrics.
- Inconsistent pipelines → apply uniform transformations during both training and inference phases.
- Imputation bias → opt for domain-aware or sophisticated imputation techniques.

5. Choose one of the data types covered in the lab (images, text, time series). Describe a specific real-world application that uses deep learning and explain how preprocessing would be crucial for that application. In the context of energy demand forecasting, preprocessing techniques such as interpolation for missing values and the creation of lag features are essential. Without these measures, the model may misinterpret gaps or shifts in scale, leading to diminished accuracy.

Deliverables

- **Completed Notebook (PDF):** This notebook with your code, outputs, and answers to the questions.
- **Reflective Journal:** A short journal (1-2 pages) reflecting on your learning experience in this lab. Consider the following prompts:
    -

Reflective Journal:

Lab 04: Deep Learning Data Preprocessing

In this laboratory session, I gained insights into various preprocessing methods that prepare unrefined data for deep learning models. I investigated techniques for managing missing values through interpolation and backward filling in time series data, as well as tokenization and removal of stopwords in textual data, resizing and normalization in image data, and MFCC feature extraction in audio data. This experience reinforced the notion that, despite the automatic feature extraction capabilities of deep learning models, their efficacy is significantly influenced by the quality of the input data.

One of the challenges I encountered was the management of dependencies related to text preprocessing, particularly concerning NLTK downloads. To address this issue, I examined alternative methods such as regex tokenization and the creation of custom stopword lists. This experience highlighted the importance of adaptability and problem-solving skills when dealing with real-world datasets. Another difficulty was differentiating between normalization and standardization; however, through experimentation with both concepts, I was able to clarify their distinctions and respective applications.

Certain concepts, such as basic tokenization, were already familiar to me, while others—like MFCCs for audio processing and the use of sliding windows for time series analysis—were novel. I was astonished by how preprocessing can convert diverse data types into numerical formats that deep learning models can effectively process.

The techniques I acquired have numerous practical applications: image preprocessing is crucial in the field of medical imaging, text preprocessing enhances the functionality of chatbots, and time series preprocessing is essential for applications such as energy demand forecasting and stock market predictions.

Looking forward, I am eager to investigate automated preprocessing pipelines and comprehensive workflows that maintain consistency between training and deployment phases. In summary, this laboratory session has reinforced my understanding that preprocessing serves as the cornerstone of any dependable deep learning initiative. Cleaner inputs invariably lead to superior models.

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.