Week 8: Unstructured data; preprocessing

Objective: In this lab session, you will learn:

1. Image preprocessing
2. Text preprocessing

We will use Google Colab - Python IDE. You will need to log in to use the Google Colab. You can use your university email or your personal.

Image preprocessing

Images, like other forms of data, can also pass through the data pipelines, meaning images can be ingested, transformed and loaded into storage. This section is mainly focused on ingestion and transformation.

1. Download the fruit image from the BB.
2. Upload the data to your Colab.
3. Import all the necessary libraries.

```
#import necessary libraries
import cv2
import os
import json
import numpy as np
from skimage import feature
from matplotlib import pyplot as plt
```

4. Ingest image data

```
# Load the image (adjust the file path to your image)
image_path = '/content/fruits.jpg'
image = cv2.imread(image_path)
```

You may want to copy the image data file path from collab and replace '/content/fruits.jpg'.

5. Display the image

```
# Convert the image from BGR to RGB format (OpenCV loads images in BGR format)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Display the image using matplotlib
plt.imshow(image_rgb)
plt.axis()
plt.show()
```

OpenCV loads images in BGR format, as such, there is a need to change BGR to RGB.

Now, let's begin to transform(preprocess) the image.

6. Resize the image to a certain width * height

```
# Resize the image to 500x500 pixels
resized_image = cv2.resize(image_rgb, (500, 500))

# Save the preprocessed image
cv2.imwrite('resized_image.jpg', resized_image)

# Display the resized image using matplotlib
plt.imshow(resized_image)
plt.axis()
plt.show()
```

Here, the image was resized to 500 x 500, saved and then displayed.

7. Rotate the image

```
# Rotate the image (90 degrees clockwise)
rotated_image_90 = cv2.rotate(image_rgb, cv2.ROTATE_90_CLOCKWISE)

# Save the rotated image
cv2.imwrite('rotated_image_90.jpg', rotated_image_90)

# Display the rotated image using matplotlib
plt.imshow(rotated_image_90)
plt.axis()
plt.show()
```

Here, the image was rotated 90 degrees clockwise, saved and then displayed. We can rotate it to other degrees to enhance the quality of the image.

8. Denoise the image

```
# Denoise the image using Gaussian blur
denoised_image = cv2.GaussianBlur(image_rgb, (5, 5), 0)

# Save the denoised image
cv2.imwrite('denoised_image.jpg', denoised_image)

# Display the denoised image using matplotlib
plt.imshow(denoised_image)
plt.axis()
plt.show()
```

Here, the image was denoised using GaussianBlur, saved and then displayed.

You may have noticed that we've been using the original image_rgb file separately for each individual processing technique. Instead, let's try applying all the techniques in sequence—first resizing, then rotating, and finally denoising—to obtain a fully processed image.

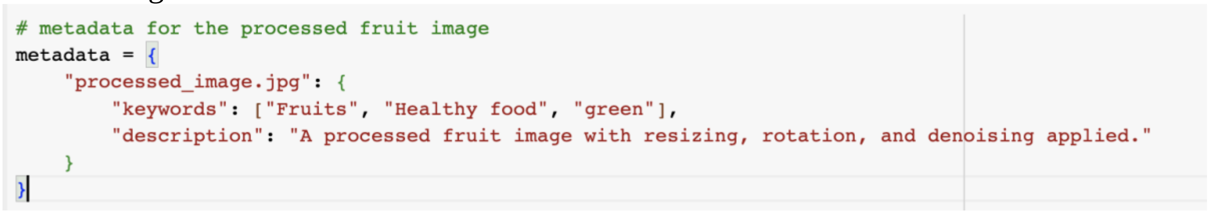
9. Apply resize, rotation and denoise on an image.

```
#apply resizing, rotation and denoising to the image and save it as processed image
image = cv2.imread('/content/fruits.jpg')

# Convert the image from BGR to RGB format (OpenCV loads images in BGR format)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

#resize thesame image to 500x500 pixel
image_rgb = cv2.resize(image_rgb, (500, 500))

# Rotate the image (90 degrees clockwise)
image_rgb = cv2.rotate(image_rgb, cv2.ROTATE_90_CLOCKWISE)

# Denoise the image using Gaussian blur
image_rgb = cv2.GaussianBlur(image_rgb, (5, 5), 0)

# Save the processed image
cv2.imwrite('processed_image.jpg', image_rgb)

# Display the denoised image using matplotlib
plt.imshow(image_rgb)
plt.axis()
plt.show()
```

Here, we applied the techniques in a specific order: resizing the image, rotating, and finally denoising. However, you can follow a different sequence. Instead, choose the order and particular techniques that best enhance the image quality based on your needs.

The image was processed, saved and then displayed.

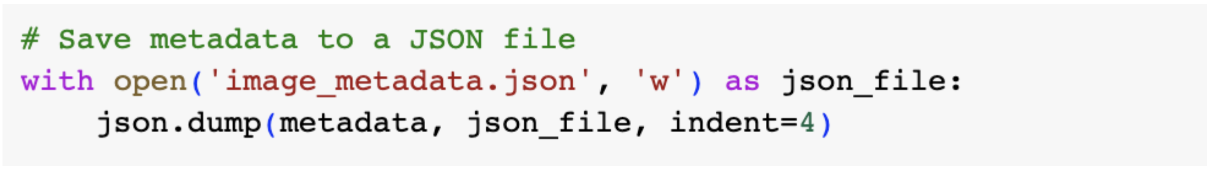
10. Image Annotation



```
# metadata for the processed fruit image
metadata = {
    "processed_image.jpg": {
        "keywords": ["Fruits", "Healthy food", "green"],
        "description": "A processed fruit image with resizing, rotation, and denoising applied."
    }
}
```

We manually created metadata(annotation) by making a dictionary with the image name as the key, containing keywords and descriptions.

11. Save the metadata as a JSON file.



```
# Save metadata to a JSON file
with open('image_metadata.json', 'w') as json_file:
    json.dump(metadata, json_file, indent=4)
```

Here, the metadata is saved as a JSON file for later use.

Image Feature Extraction
Extract features such as mean intensity and norm intensity.

12. Calculate the mean and norm intensity of the processed image.

```python
# Calculate mean and norm of pixel intensities
mean_intensity = np.mean(processed_image)
norm_intensity = np.linalg.norm(processed_image)
```

Here, calculate the processed data's mean and norm intensity. We saved the mean and norm as variables, each for later use.

13. Extract shape features

```python
# Convert to grayscale
gray_image = cv2.cvtColor(processed_image, cv2.COLOR_BGR2GRAY)

# Apply Canny edge detection
edges = cv2.Canny(gray_image, 100, 200)

# Find contours from the edge-detected image
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Iterate through each contour to calculate the shape features
for contour in contours:
    # Area
    area = cv2.contourArea(contour)

    # Perimeter (arc length)
    perimeter = cv2.arcLength(contour, True)  # True means the contour is closed

    # Centroid (center of mass)
    moments = cv2.moments(contour)
    if moments["m00"] != 0:
        cX = int(moments["m10"] / moments["m00"])
        cY = int(moments["m01"] / moments["m00"])
    else:
        cX, cY = 0, 0

    # Bounding box (smallest rectangle that contains the object)
    x, y, w, h = cv2.boundingRect(contour)


    # Print the shape features
    print(f"Area: {area}, Perimeter: {perimeter}, Centroid: ({cX}, {cY}), Bounding Box: ({x}, {y}), Width: {w}, Height: {h}")
```

Here, we started by converting the image to grayscale, and then we computed the area, perimeter, centroid, and bounding box as examples of shape features.

14. Metadata for all the image features extracted from tasks 12-13.

```python
#collect all the shape features as a dictionary
shape_feature = {
        "area": area,
        "perimeter": perimeter,
        "centroid": (cX, cY),
        "bounding_box": (x, y, w, h)
    }
```

```python
# Collect all the features extracted from the processed_image and store as a meta data
features_extracted = {
    "mean_intensity": mean_intensity,  # Mean intensity calculated earlier
    "norm_intensity": norm_intensity,  # Norm intensity calculated earlier
    "shape_features": shape_feature  # List to hold shape features for each object
}
```

```python
# Save the features to a JSON file
with open('image_features.json', 'w') as json_file:
    json.dump(features_extracted, json_file, indent=4)
```

Here, the shape features (area, perimeter, centroid, bounding box) were first put together and then combined with mean and norm intensity. All the image features were extracted and put together as a dictionary and metadata file for the processed image. The features extracted were saved as a JSON file for later use.

Text processing

This section focuses on text data's ingestion and transformation(preprocessing).

1. Download the document named 101551 from the BB.
2. Upload the data to your Colab.
3. Import all the necessary libraries.

```
#import necessary library
import os
import nltk
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
```

4. Download stopwords and punkt tokeniser

```
# Download stopwords and punkt tokenizer
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
```

Here, we download the list of stop words for the English language. Note that each language typically has its own set of rules, so different languages may require their specific dictionary to identify stop words.

5. Load the document file

```
#Load the content of the specified file
with open('/content/101551') as file:
  document_101551 = file.read()
```

6. Filter out some parts of the document and create data out of that.

```
document_sample=document_101551[621:-15]
```

We created new text data by filtering out the first 621 and last 15 characters.

Preprocessing Techniques:

- Text Normalization (Lowercasing)
- Tokenization & Removal of Punctuation and Stop Words
- Stemming reduces a word to its root or base form by removing prefixes and suffixes.
- Lemmatization reduces word to their dictionary base form (lemma)

7. Convert the sample document to lowercase all through.

```
# Text Normalization: Convert document_101551 to lowercase and rename as document_101551_lower
document_sample_lower = document_sample.lower()
```

8. Tokenise the sample document.

```
# Tokenization: Split document_101551_lower into words
tokens = word_tokenize(document_sample_lower)
```

9. Remove all punctuation marks from the tokens.

```
#Removal of punctuation
tokens = [word for word in tokens if word not in string.punctuation]
```

10. Remove stop words from the tokens.

```
#Removal of stop words
stop_words = set(stopwords.words('english'))
tokens = [word for word in tokens if word not in stop_words]
```

11. Print the processed tokens.

```
# Print the preprocessed tokens
print("Preprocessed Tokens:", tokens)
```

12. Apply stemming on the tokens.

```
# Initialize stemmer and lemmatizer
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()
```

```
# Apply stemming to the tokens
stemmed_words = [stemmer.stem(word) for word in tokens]

# Print the stems from the tokens
print("Stemmed Words:", stemmed_words)
```

Here, we initialise a stemmer and a lemmatiser because they have to do with rules; as such, a sort of word dictionary that helps define suffixes, prefixes, and root words is required.

13. Apply lemmatisation to the tokens

```
# Apply lemmatization to the tokens
lemmatized_words = [lemmatizer.lemmatize(word) for word in tokens]

# Print the lemma from the tokens
print("Lemmatized Words:", lemmatized_words)
```

Extra Exercise.
Convert the sample document from task 7 into a numerical representation (i.e., vectorisation).

1. Using Bag of Words (BoW)
2. Using Term Frequency-Inverse Document Frequency (TF-IDF)

Metadata and labelling for text data

14. Create metadata for the processed text data.

```
# Metadata about the vectorization process
metadata = {
    "vectorization_methods": {
        "bag_of_words": "Each word in the corpus is treated as a feature, and the text is represented by a count of each word.",
        "tf_idf": "Each word in the corpus is weighted based on its frequency in the document and its inverse frequency across all documents."
    },
    "document_summary": "This is a summary of the text data preprocessing and vectorization for document 101551.",
    "sentiment_label": "neutral"  # You can update this label based on your analysis
}
```

Here, we created a metadata document explaining the vectorisation process and labelled the text data.

15. Save the metadata file for the text data as a JSON file.

```
# Save metadata to a JSON file
with open('text_metadata.json', 'w') as json_file:
    json.dump(metadata, json_file, indent=4)
```