FinScraper: Financial Statement Scraping and Table Detection Project

Table of Contents

1. Introduction

FinScraper is a comprehensive project designed to automate the extraction and analysis of financial statements from the U.S. Securities and Exchange Commission's (SEC) EDGAR database. The project consists of two main components: a web scraping module to download 10-K filings, and a deep learning-based table detection system to identify and extract tables from these filings.

2. Project Overview

The FinScraper project aims to streamline the process of financial data extraction and analysis through the following steps:

1. Scrape 10-K filings from the SEC's EDGAR database for specified companies.
2. Convert the downloaded HTML files to images.
3. Preprocess the images for consistency and quality.
4. Train a Faster R-CNN model to detect tables within these images.
5. Use the trained model to extract table information from financial statements.

This end-to-end pipeline allows for efficient extraction of structured financial data from unstructured SEC filings.

3. Dependencies

The project relies on several Python libraries:

- `requests`: For making HTTP requests to the SEC website.
- `BeautifulSoup` from `bs4`: For parsing HTML content.
- `pandas`: For data manipulation (imported but not used in the current version).
- `os` and `glob`: For file and directory operations.
- `PIL` (Python Imaging Library): For image processing.
- `imgkit`: For converting HTML to images.
- `torch` and `torchvision`: For creating and training the neural network model.
- `tqdm`: For progress bars.
- `numpy` and `cv2`: For image processing and analysis.
- Custom `preprocessor` module: Contains functions for HTML to image conversion and image preprocessing.

4. Web Scraping Component

4.1 Functions

get_10k_urls

```
def get_10k_urls(cik, num_docs=10):
    # Function implementation
```

Purpose: Retrieves URLs for 10-K filings of a specified company.

Parameters:

- `cik`: The Central Index Key (CIK) of the company.
- `num_docs`: The number of documents to retrieve (default is 10).

Process:

1. Constructs the URL for the SEC's EDGAR search page using the provided CIK.
2. Sends a GET request to the constructed URL.
3. Parses the HTML response using BeautifulSoup.
4. Finds all links with the ID "documentsbutton".
5. Returns a list of full URLs for the document pages.

download_10k

```
def download_10k(url, save_dir):
    # Function implementation
```

Purpose: Downloads the 10-K filing from a given URL and saves it as an HTML file.

Parameters:

- `url`: The URL of the document page.
- `save_dir`: The directory where the downloaded file will be saved.

Process:

1. Sends a GET request to the provided URL.
2. Parses the HTML response using BeautifulSoup.
3. Finds the table containing document links.
4. Searches for the row containing the 10-K document link.
5. Constructs the full URL for the 10-K document.
6. Downloads the 10-K document.
7. Saves the document as an HTML file in the specified directory.

main (Scraping)

```
def main():
    # Function implementation
```

Purpose: Orchestrates the scraping process.

Process:

1. Sets the CIK for the desired company (Apple Inc. in this case).
2. Defines the directory for saving downloaded files.
3. Creates the save directory if it doesn't exist.
4. Retrieves the URLs for 10-K filings using `get_10k_urls()`.
5. Iterates through the URLs and downloads each 10-K filing using `download_10k()`.

4.2 Execution Flow

1. The script starts by executing the `main()` function.
2. It sets up the necessary parameters (CIK and save directory).
3. Retrieves a list of 10-K filing URLs for the specified company.
4. For each URL, it downloads the corresponding 10-K document and saves it as an HTML file.

4.3 Challenges and Considerations (Scraping)

1. **Rate Limiting**: The SEC website may have rate limiting in place. The current script doesn't implement any delay between requests, which could lead to IP blocking if too many requests are made in a short time.

2. **User Agent**: The script uses a custom User-Agent header, which is good practice. However, it's hardcoded and should ideally be configurable.

3. **Error Handling**: The current implementation lacks robust error handling. Network issues or changes in the SEC website structure could cause the script to fail.
4. **File Naming**: The file naming convention uses part of the URL, which may not always result in unique or meaningful names.
5. **Scalability**: The script is designed to work with a single CIK at a time. Scaling this to multiple companies would require additional logic.

5. Preprocessing

5.1 HTML to Image Conversion

```
def html_to_image(html_file, output_dir, max_images=10):
    # Function implementation
```

Purpose: Converts HTML tables to PNG images.

Process:

1. Reads the HTML file
2. Uses BeautifulSoup to find all table elements
3. Converts each table to an image using `imgkit`
4. Checks if the resulting image is empty (using `is_empty_image` function)
5. Saves non-empty images, up to a maximum of `max_images`

5.2 Image Cleaning

```
def clean_images(image_dir, output_dir, target_size=(800, 600)):
    # Function implementation
```

Purpose: Processes and standardizes the images.

Process:

1. Iterates through PNG files in the input directory
2. Checks if each image is empty
3. For non-empty images:
    - Converts to RGB if necessary
    - Resizes to a target size (default 800x600)
    - Enhances contrast
    - Saves the processed image

5.3 Dataset Splitting

```
def split_dataset(image_dir, train_dir, val_dir, test_dir, split_ratio=
(0.7, 0.15, 0.15)):
    # Function implementation
```

Purpose: Splits the preprocessed images into training, validation, and test sets.

Process:

1. Collects all non-empty images
2. Splits the images into train, validation, and test sets according to the specified ratio
3. Moves the images into their respective directories

6. Dataset Creation

```
class FinancialTableDataset(Dataset):
    # Class implementation
```

Purpose: Creates a custom PyTorch dataset for the financial statement images.

Key Methods:

- `__init__`: Initializes the dataset with image paths and transforms
- `__len__`: Returns the number of images in the dataset
- `__getitem__`: Loads an image and its corresponding target (bounding box and label)

Note: This implementation assumes each image contains one table and uses the entire image as the bounding box. In a real scenario, you would need actual bounding box annotations.

7. Model Architecture

```
def get_model(num_classes):
    # Function implementation
```

Purpose: Creates and configures the Faster R-CNN model.

Process:

1. Initializes a pre-trained Faster R-CNN model with a ResNet-50 backbone
2. Modifies the model's region proposal network for the specific number of classes (in this case, 2: background and table)

8. Training Process

```
def train_model(model, train_loader, val_loader, optimizer,
num_epochs=10):
```

```
    # Function implementation
```

Purpose: Trains the Faster R-CNN model on the financial statement dataset.

Process:

1. Iterates through the specified number of epochs
2. For each epoch:
    - Trains the model on the training dataset
    - Validates the model on the validation dataset
    - Computes and displays the training and validation losses
3. Saves the trained model

9. Key Concepts

- **SEC**: Securities and Exchange Commission, the U.S. government agency responsible for enforcing federal securities laws and regulating the securities industry.
- **EDGAR**: Electronic Data Gathering, Analysis, and Retrieval system, the SEC's database of electronic filings.
- **10-K**: An annual report filed by public companies, providing a comprehensive summary of the company's financial performance.
- **CIK**: Central Index Key, a unique identifier assigned by the SEC to each entity that files reports with them.
- **BeautifulSoup**: A Python library for pulling data out of HTML and XML files.
- **Faster R-CNN**: A two-stage object detection model that uses a Region Proposal Network (RPN) to generate region proposals, followed by a Fast R-CNN detector to classify these proposals.
- **Transfer Learning**: The use of a pre-trained model (in this case, Faster R-CNN with ResNet-50 backbone) as a starting point, which is then fine-tuned on a specific task.
- **Data Augmentation**: The process of creating modified versions of images in the training dataset to improve model generalization (not implemented in the current code but could be added).
- **Bounding Box**: A rectangular area in an image that encloses an object of interest (in this case, a table).

10. Overall Challenges and Considerations

1. **Data Quality**: The quality of the scraped HTML and converted images can significantly impact the performance of the table detection model.

2. **Scalability**: Both the scraping and model training components need to be scalable to handle large numbers of companies and documents.

3. **Bounding Box Annotations**: The current implementation assumes one table per image and uses the entire image as the bounding box. Real-world applications would require accurate bounding box annotations.

4. **Class Imbalance**: With only two classes (background and table), there might be a significant class imbalance, which could affect model performance.

5. **Model Complexity**: Faster R-CNN is a complex model that might be overkill for simple table detection. Simpler models could be explored for this task.

6. **Computational Resources**: Training object detection models is computationally intensive and may require significant GPU resources.

7. **Dataset Size**: The effectiveness of the model will depend heavily on the size and quality of the dataset. Ensuring a large and diverse dataset is crucial.

8. **Regulatory Compliance**: Ensure that the scraping process complies with SEC's terms of service and doesn't overwhelm their servers.

11. Future Improvements

1. Implement rate limiting in the scraping process to avoid overwhelming the SEC servers.
2. Add more robust error handling and logging throughout the pipeline.
3. Improve the file naming convention to ensure uniqueness and readability.
4. Add support for other types of filings beyond 10-K.
5. Implement a command-line interface for easier use with different companies.
6. Add functionality to parse the downloaded HTML and extract specific financial data.
7. Implement multi-threading to speed up the download process for multiple documents.
8. Implement proper bounding box annotations for more accurate table detection.
9. Explore data augmentation techniques to increase dataset diversity.
10. Experiment with different model architectures (e.g., YOLO, SSD) to find the best balance of speed and accuracy.
11. Implement early stopping and learning rate scheduling for more efficient training.
12. Add post-processing steps to refine the model's predictions.
13. Explore techniques for handling multiple tables in a single image.
14. Implement a more robust evaluation metric (e.g., mean Average Precision) for model performance.
15. Develop a user interface for easier interaction with the FinScraper system.
16. Implement a database to store extracted financial data for easier querying and analysis.

This comprehensive documentation provides a detailed overview of the FinScraper project, covering both the web scraping component and the table detection model training process. It explains the key steps, concepts, and considerations involved in each part of the pipeline, from data collection to model deployment, and outlines potential areas for future improvement and expansion of the project's capabilities.