CS FINAL PROJECT

D E M O

Multi-task learning of objects and parts

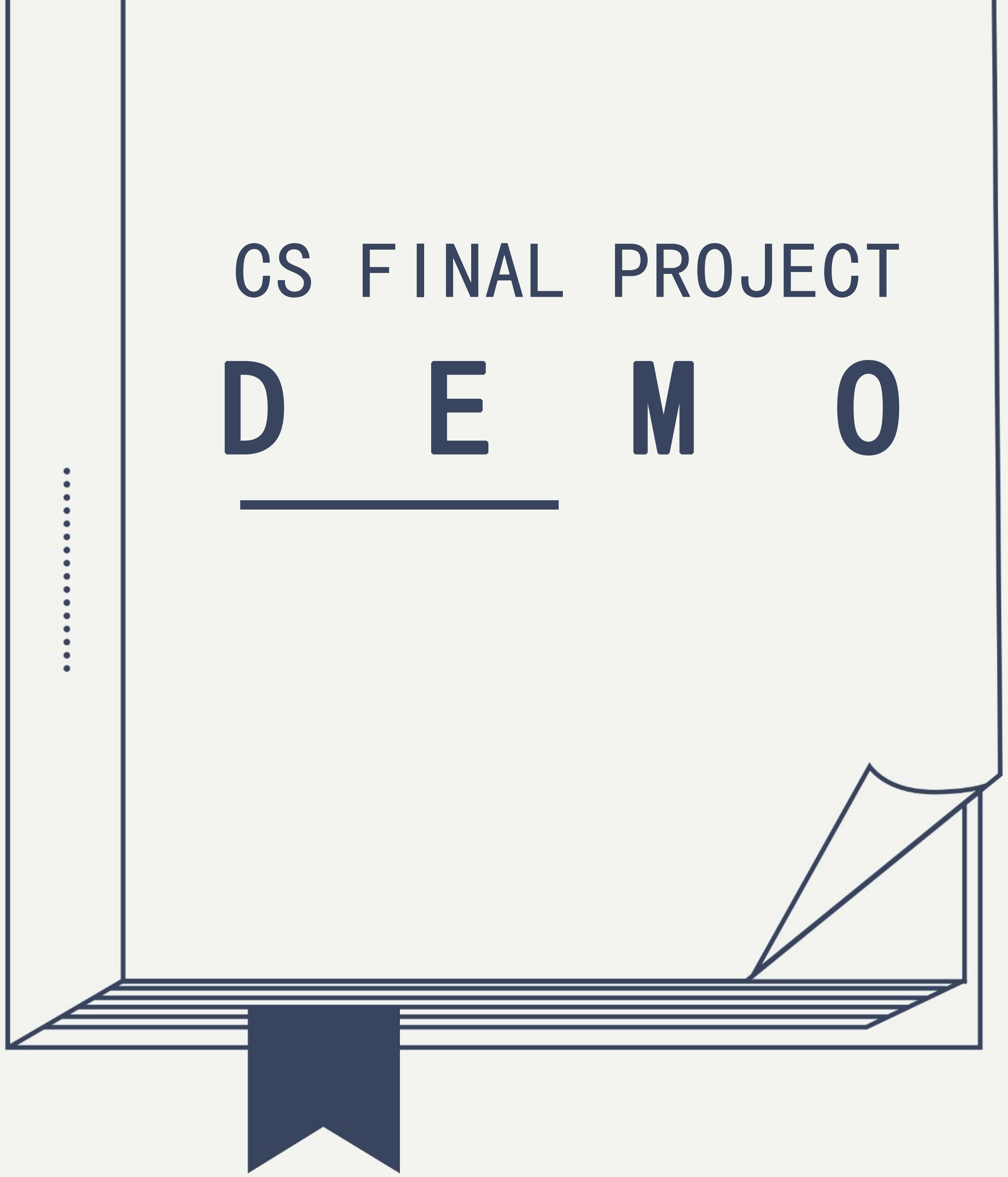
Student: Hailin Weng

Student ID: 40381868

Supervisor: Dr.Barry Devereux

12/01/2024

CONTENTS



CS FINAL PROJECT

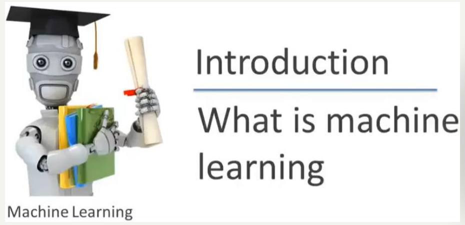
D E M O

01 Foundation

Machine Learning
Deep Learning
Computer Vision
Model Implementation

Tutorials

A


Introduction

What is machine learning

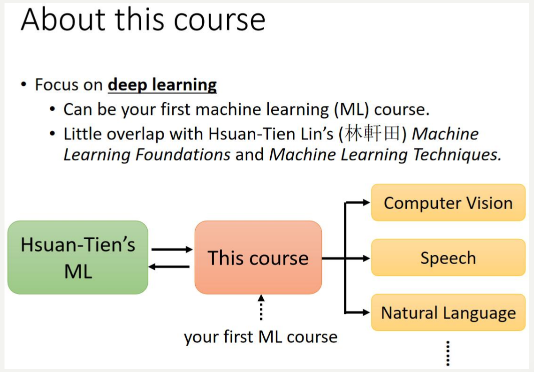
Machine Learning

Machine Learning

Andrew Ng's Machine Learning Course – Standford University

https://www.coursera.org/specializations/machine-learning-introduction

B


About this course

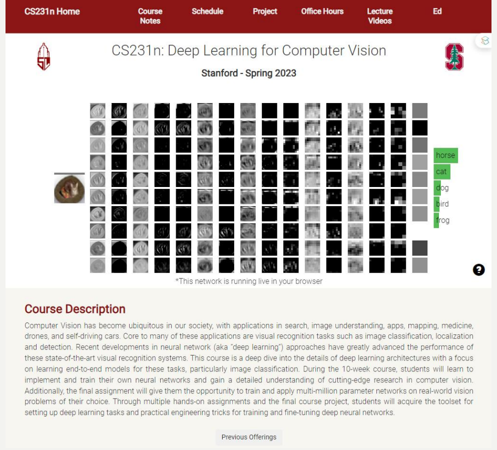
- Focus on **deep learning**
  - Can be your first machine learning (ML) course.
  - Little overlap with Hsuan-Tien Lin's (林軒田) *Machine Learning Foundations* and *Machine Learning Techniques.*

Deep Learning

Hung-yi Lee Machine Learning (Deep Learning) – National Taiwan University

https://speech.ee.ntu.edu.tw/~hylee/ml/2022-spring.php

C



CS231n: Deep Learning for Computer Vision

Stanford - Spring 2023

*This network is running live in your browser

Course Description

Computer Vision has become ubiquitous in our society, with applications in search, image understanding, apps, mapping, medicine, drones, and self-driving cars. Core to many of these applications are visual recognition tasks such as image classification, localization and detection. Recent developments in neural network (aka "deep learning") approaches have greatly advanced the performance of these state-of-the-art visual recognition systems. This course is a deep dive into the details of deep learning architectures with a focus on learning end-to-end models for these tasks, particularly image classification. During the 10-week course, students will learn to implement and train their own neural networks and gain a detailed understanding of cutting-edge research in computer vision. Additionally, the final assignment will give them the opportunity to train and apply multi-million parameter networks on real-world vision problems of their choice. Through multiple hands-on assignments and the final course project, students will acquire the toolset for setting up deep learning tasks and practical engineering tricks for training and fine-tuning deep neural networks.

Computer Vision

Fei-Fei Li & Justin Johnson & Serena Yeung CS231n: Deep Learning for Computer Vision – Standford University

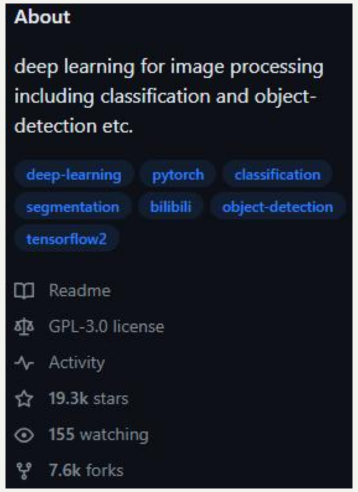
http://cs231n.stanford.edu/

D

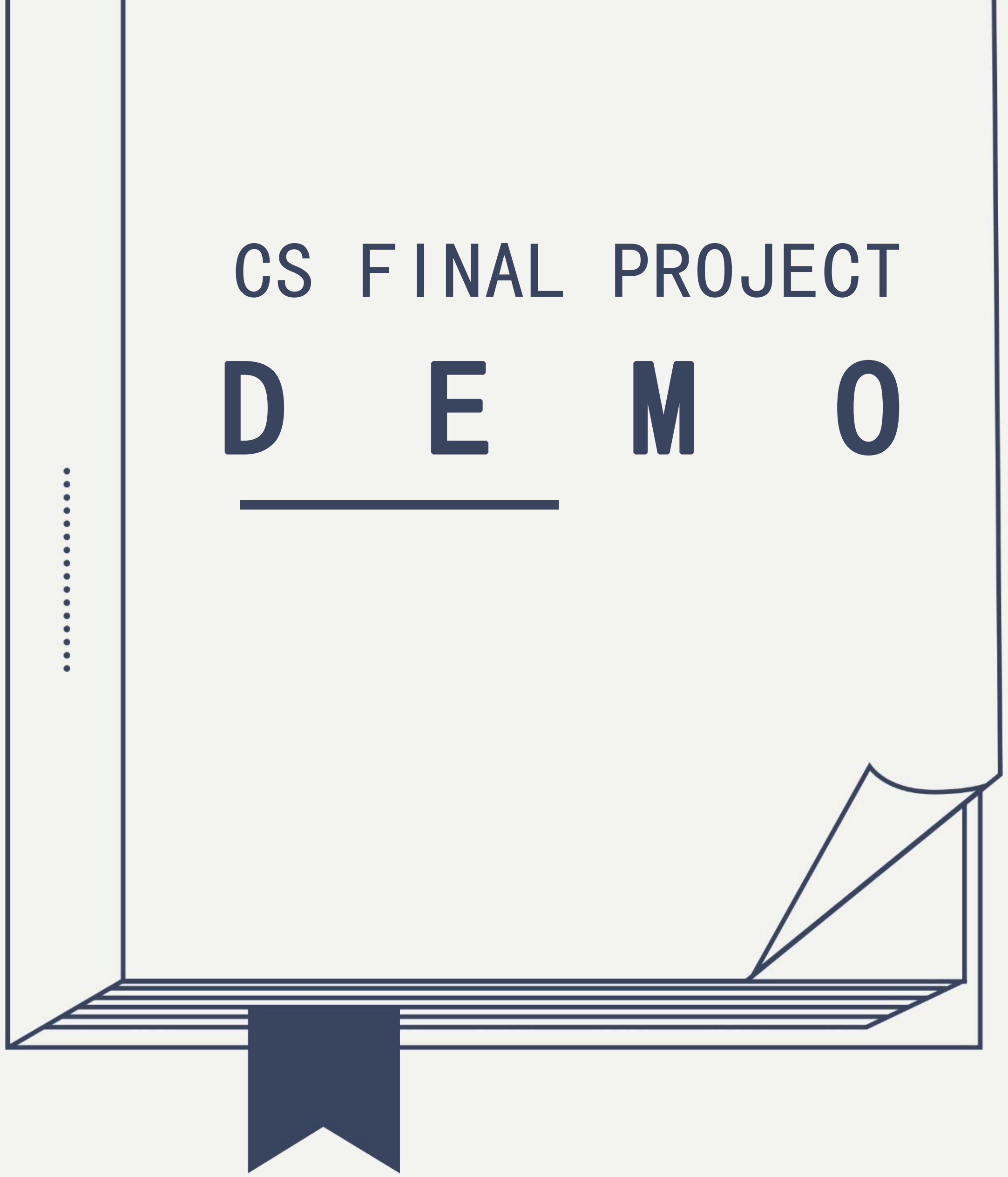
Implementations

Some online tutorials and codes:

https://github.com/WZMIAOMIAO/deep-learning-for-image-processing


About

deep learning for image processing including classification and object-detection etc.



CS FINAL PROJECT

DEMO

02

CSLB Property Norms filling

Cosine Similarity
GPT4 API

CSLB Property Norms Filling

Limitation

Limitation

Property norm datasets often lack exhaustive feature information, as participants tend to provide only what they consider informative. For instance, only six animals in the property norms dataset may be labeled with the feature "does breathe," although this property applies to all animal concepts.

Excessive data

Excessive data

There are 638 concepts and 2725 features in the property norms. This gives ~ 2 million possible combinations of concept and feature pair. It is not feasible to test all 2 million combinations.

Cosine Similarity

Cosine Similarity

We can target features that are true of one concept and not true of a similar concept. For example, giraffe has the property "has a neck" in the property norms, whilst elephant does not. Giraffe and elephant will be similar in terms of their existing CSLB properties (has legs, is a mammal, etc). We can measure the similarity of concepts based on the cosine similarity of their feature vectors.

GPT4

GPT4 API

To address this limitation, we leverage ChatGPT (GPT4)'s API to programmatically acquire "yes" or "no" responses for all features in the property norms dataset. This approach would allow us to create a more extensive set of true features.

Code Logic

1. Loading and Preprocessing Data:

- The code begins by loading a feature matrix from a CSV file called 'feature_matrix.csv.' This matrix contains information about semantic features associated with different object concepts.
- It then binarizes the features in the matrix, converting values greater than 2 to 1, while values less than or equal to 2 become 0. This binarization simplifies feature representation.

2. Calculating Cosine Similarity:

- The code calculates pairwise cosine similarities between object concepts based on their binarized features. This step creates a cosine similarity matrix that quantifies the similarity between concepts.

3. Filtering and Sorting:

- The cosine similarity matrix is flattened to create a list of concept pairs along with their cosine similarities. Self-comparisons and duplicate pairs are filtered out.
- The list of concept pairs is then sorted in descending order of cosine similarity to prioritize the most similar pairs.

4. Querying GPT-4 for Feature Verification:

- The code iterates through the sorted list of concept pairs and compares their features.
- For each pair, it retrieves the features associated with each concept.
- It then queries GPT-4 using a customized prompt to verify if a specific feature is associated with a concept. This step is crucial for identifying informative features.
- If GPT-4 confirms that a feature is associated with a concept, that feature is added to the concept's feature set in the feature matrix.
- The process continues until a predefined number of features (100 in this case) have been added to the feature matrix.

5. Logging and Caching:

- The code logs each query made to GPT-4, noting whether a feature was successfully added to a concept.
- To improve efficiency and prevent duplicate queries, the code maintains a cache of responses and processed concept pairs.

6. Final Saving of Updated Feature Matrix:

- After processing a batch of concept pairs or reaching the maximum number of added features, the code saves the updated feature matrix to a CSV file called 'updated_feature_matrix.csv.'

Code Explanation:

- The primary objective of this code is to enhance the feature information associated with object concepts by leveraging GPT-4's natural language understanding capabilities. It achieves this by iteratively querying GPT-4 for feature verification and updating the feature matrix accordingly.

- The code is designed for flexibility and scalability, allowing the addition of a large number of features to object concepts in an automated manner. This approach ensures that the feature matrix becomes richer and more informative, ultimately benefiting various downstream tasks in computer vision and object recognition.

- The code's modular structure facilitates easy maintenance, logging of queries and results, and efficient caching of responses. It represents a critical component of Property Norm Infilling, a process that contributes to a more comprehensive understanding of object properties in the context of computer vision.

- By effectively combining data preprocessing, cosine similarity calculation, AI model querying, and data logging, this code represents a robust solution for infilling property norms and enhancing feature information about object concepts.

CS FINAL PROJECT

D E M O

03

Mapping Between CSLB Property Norms and THINGS Dataset

WordNet & ConceptNet
GPT4 API

Mapping Between CSLB Property Norms and THINGS Dataset

Mapping

Mapping

Many of the object concepts present in the THINGS dataset exhibit significant overlap with those found in the CSLB property norms. To effectively train our computer vision model, it is imperative that we possess both images and feature vectors for these shared object concepts. While some instances align seamlessly, with concept names matching between the two datasets, challenges arise when the same concept appears under different names. For instance, "courgette" in the CSLB norms may correspond to "zucchini" in the THINGS database.

To seamlessly integrate these datasets, we meticulously crafted a mapping process that establishes a robust correspondence between CSLB names and THINGS names. This mapping is pivotal for ensuring that we can accurately associate the correct images and features with each concept during the model training process. The mapping results in a structured dictionary known as "namemap," characterized by entries in the form of namemap["cslbname"] = "THINGS name."

Refine

Refine

It is noteworthy that the initial mapping process, while comprehensive, contained inaccuracies and mismatches. To address this, we undertook a refinement process using WordNet and ConceptNet. However, we encountered challenges and inconsistencies in these efforts. Subsequently, we leveraged the power of GPT-4's API to filter and optimize the mapping, resulting in an improved mapping dictionary stored as "updated_namemap.json."

Through this meticulous mapping effort, we have successfully identified and corrected erroneous concept pairs, ensuring accurate mappings between CSLB Property Norms and the THINGS dataset. These meticulously curated concept pairs, along with their associated images and feature vectors, will serve as the foundational building blocks for training our feature prediction model. This robust dataset empowers our model to make precise predictions about object attributes, thereby enhancing the success and accuracy of our research endeavors.

Code Logic: Mapping

1. Loading CSLB and THINGS Datasets:

- The code begins by importing necessary libraries and modules, including Pandas for data manipulation, requests for making HTTP requests, and NLTK (Natural Language Toolkit) for text processing. It also downloads the WordNet corpus from NLTK, which is a lexical database for the English language.

2. Extracting Terms from Datasets:

- The code extracts the lists of terms from both the CSLB and THINGS datasets. These terms represent object concepts.

3. Finding Synonyms from WordNet and ConceptNet:

- The code defines two functions, get_synonyms_from_wordnet and get_synonyms_from_conceptnet, to find synonyms of a given term. It leverages WordNet and ConceptNet, which are lexical resources containing information about words, their meanings, and relationships.

4. Finding Mappings between CSLB and THINGS:

- The find_mappings function iterates through the CSLB terms and attempts to find synonyms from both WordNet and ConceptNet. It then creates mappings between CSLB terms and the first synonym found in the THINGS dataset. This process is performed for all CSLB terms.

5. Combining Automatic and Manual Mappings:

- The code combines automatic mappings generated by the find_mappings function with any manual mappings provided in the manual_mappings dictionary. The manual mappings allow for specific term associations that may not be found through automated methods.

6. Saving the Mapping Results:

- Finally, the code saves the namemap as a JSON file named "namemap.json".

Code Explanation

- This code logic systematically processes the CSLB and THINGS datasets, identifies synonyms using WordNet and ConceptNet, creates mappings, and saves the results as a JSON file, providing a structured and organized mapping between object concepts in the two datasets.

Code Logic: Refine

1. OpenAI GPT-4 Configuration:

- The code begins by configuring the OpenAI GPT-4 API with the provided API key.

2. Loading Existing Namemap and Log:

- The code checks if an existing "namemap.json" file exists. If it does, it loads the mapping between CSLB terms and THINGS terms. It also checks for the existence of processed status and query log files, loading them if available.

3. Querying GPT-4 for Term Verification:

- The code defines a function called query_gpt that queries the GPT-4 model to verify if a CSLB term is a suitable equivalent or closely related term to a THINGS term. It constructs a prompt for GPT-4 to provide a 'yes' or 'no' response.

4. Iterating Through Namemap Entries:

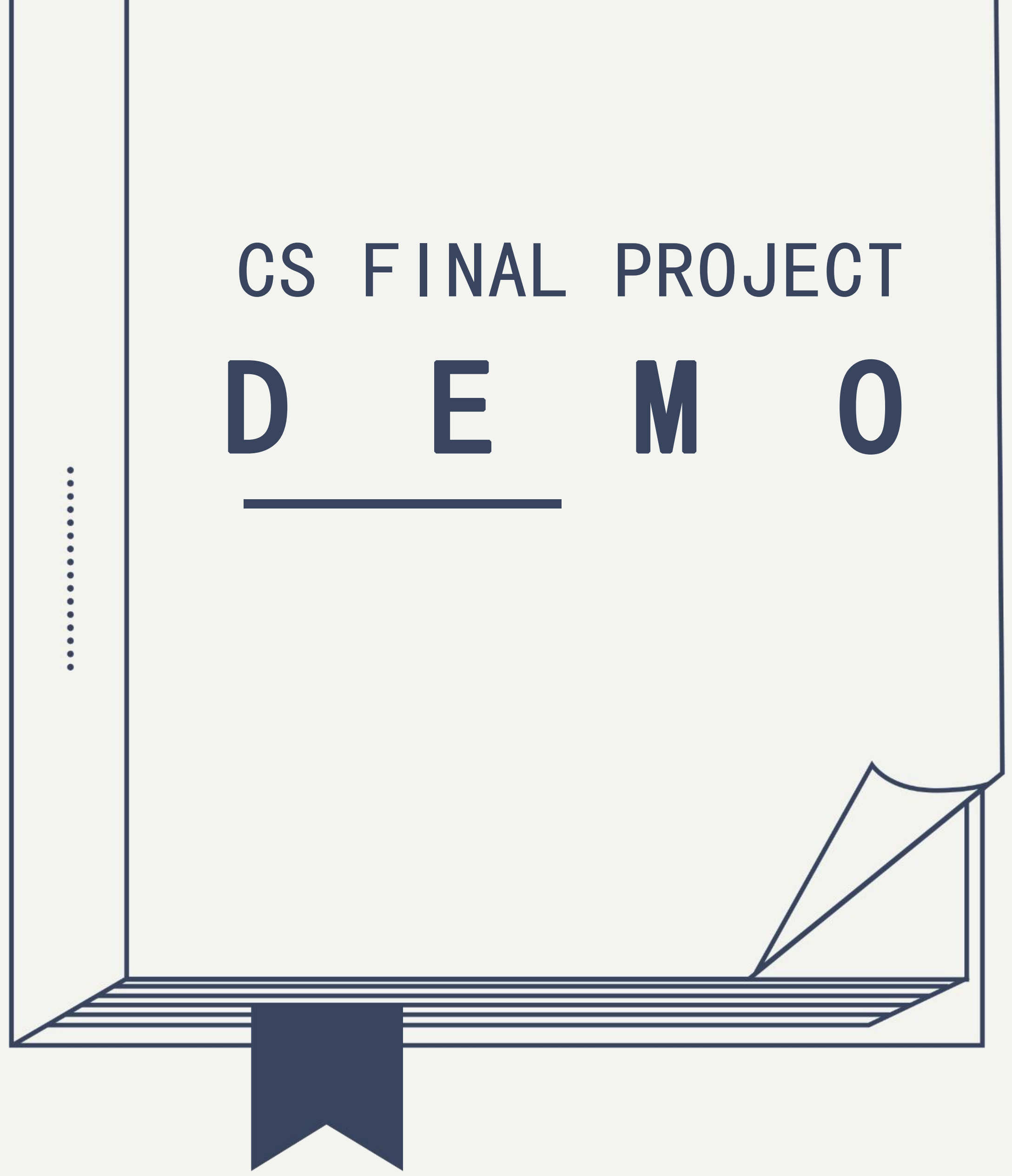
- The code iterates through each entry in the existing namemap (mapping between CSLB and THINGS terms). For each entry, it checks if the CSLB term has been processed before.

5. Updating Processed Status and Saving Data:

- After querying GPT-4, the code records the verification result and updates the processed status for the CSLB term. If the verification result indicates a positive match, the term is added to the updated_namemap. The code also saves the updated namemap, log, and processed status.

Code Explanation

- In summary, this code logic systematically queries GPT-4 to verify the equivalence of CSLB and THINGS terms, updates the namemap, logs the verification results, and maintains processed status records for CSLB terms. The result is an enriched and verified mapping between object concepts in the two datasets, stored in "updated_namemap.json."



CS FINAL PROJECT

D E M O

04

Model Training & Testing

Based on AlextNet

Unified Model

Separate Models

Model Training & Testing

Preprocessing

Preprocessing

The preprocessing steps involve two scripts, filter.py and split_data.py, with distinct purposes:

filter.py:

Purpose: Reverse a namemap, filter images based on the namemap, and organize them into class-specific directories.

Role: Reverses a namemap mapping THINGS dataset terms to CSLB terms, then filters THINGS dataset images based on this mapping, copying them to class-specific directories.

split_data.py:

Purpose: Split the filtered image dataset into training and validation sets.

Role: Allocates a specified percentage (e.g., 10%) of the filtered dataset to a validation set, creating separate folders for training and validation data. Images are randomly assigned to these sets based on the split rate.

These preprocessing steps prepare the dataset for machine learning tasks such as object recognition and classification.

Based on AlexNet

Based on AlexNet

I need to make modifications to the existing PyTorch implementation of the AlexNet model. These adjustments will be made to the model, training, and prediction code based on the foundation provided by AlexNet.

Model Training & Testing

Modify

Modify

Development and Testing of Training Code: The development and rigorous testing of the training code are pivotal aspects of the project. Special emphasis is placed on handling and processing the dataset effectively. This includes optimizing data preprocessing, ensuring data integrity, and fine-tuning training parameters to achieve optimal model performance.
Exploration of AlexNet Architecture: A comprehensive exploration of the AlexNet architecture is underway, with a specific focus on its outputs and loss functions. These architectural components are being scrutinized to determine how they can be adapted and customized to align with the project's objectives. This investigation aims to harness the full potential of AlexNet for concept and feature prediction tasks.

Models

Unified Model VS Separate Models

Unified Model Approach: One approach involves the utilization of a single model for both concept and feature prediction tasks. This unified model is designed to handle both tasks within the same architecture. It aims to capture and integrate diverse information required for concept and feature prediction in a cohesive manner.

Separate Models Strategy: The project considers the deployment of separate models—one dedicated to concept prediction and another to feature prediction. This separation allows for the distinct modeling of different types of information in the final layers of each model. By decoupling these tasks, the project seeks to enhance model specialization and overall prediction accuracy.
These two approaches are under evaluation to determine their suitability and effectiveness in achieving the project's objectives. The decision regarding which approach to adopt will be based on empirical evidence and alignment with the specific requirements of the task at hand.

Code Logic:
Dataset.py

Purpose

The purpose of this code is to create a custom image dataset for training a deep learning model within the context of the Unified Model.

1. Dataset Creation:

- The main purpose of this code is to create a custom image dataset for training a deep learning model. This dataset includes both images and their corresponding feature vectors, allowing the model to be trained for both object category (concept) classification and object feature prediction simultaneously.

2. Data Preprocessing:

- The code loads a CSV file containing feature vectors and a JSON file containing class mapping relationships, which are essential for subsequent data loading and processing.

3. Image Transformation:

- Using the PyTorch transforms module, a series of image transformation operations are defined. These operations include resizing images and converting them into tensors. These transformations are applied to ensure that the image data is in the correct format for input into the deep learning model.

4. Data Loader Creation:

- Ultimately, the code creates data loaders using PyTorch, which wrap the dataset into batches of data that can be fed into the model during training. These data loaders can be iterated over in the training loop, providing batches of images and their associated feature vectors for model training.

Divergences from the Original AlexNet

1. Task-Specific Modification:

- Unlike the conventional AlexNet, which is primarily designed for image classification tasks, this modified AlexNet serves a dual-purpose. It has been tailored to address the specific needs of the project, which involve not only classifying objects but also predicting their associated features.

2. Additional Output Layers:

- The modified AlexNet architecture extends beyond the conventional classification task. It incorporates two additional output layers: one for concept prediction and another for feature prediction.

3. Loss Functions:

- In this modified version, the loss functions employed during training are adapted to accommodate the dual tasks of concept prediction and feature prediction. The loss functions used include the cross-entropy loss for concept prediction and the mean squared error (MSE) loss for feature prediction.

4. Training Procedure:

- The training loop has been customized to handle the joint optimization of concept and feature prediction. This involves the calculation of losses for both tasks and the backpropagation of gradients accordingly.

Key Modifications

1. Concept and Feature Prediction:

- The most notable modification lies in the incorporation of two separate tasks within the same architecture. The model is equipped to simultaneously predict object concepts and associated features, providing a holistic understanding of the objects in question.

2. Dual Output Layers:

- The introduction of two output layers in the final stages of the network allows for the simultaneous extraction of concept-related and feature-related information. This enables the model to generate predictions that encompass both high-level class labels (concepts) and quantitative attribute values (features).

3. Task-Specific Losses:

- Task-specific loss functions are employed during training. Cross-entropy loss is utilized for concept prediction, emphasizing the correct classification of objects into predefined categories. Meanwhile, mean squared error (MSE) loss is applied for feature prediction, ensuring accurate numerical estimations of object attributes.

4. Adaptive Optimization:

- The training process incorporates optimization techniques that account for the dual nature of the tasks. The model learns to balance concept classification and feature estimation simultaneously, resulting in a more comprehensive representation of objects.

In summary, this modified AlexNet diverges from the conventional architecture by accommodating two distinct prediction tasks: object concept classification and feature estimation. Through the incorporation of additional output layers and task-specific loss functions, the model is tailored to meet the project's objectives of jointly predicting concepts and features. These modifications enhance the model's capacity to provide a richer understanding of objects beyond traditional image classification.

Divergences from the Original AlexNet

1. Specialization for Dual Tasks:

- Unlike the conventional AlexNet, which is primarily designed for image classification tasks, this modified AlexNet serves a dual-purpose. It has been tailored to address the specific needs of the project, which involve not only classifying objects but also predicting their associated features.

2. Multiple Models:

- Instead of a single monolithic architecture, this approach employs two separate instances of the modified AlexNet architecture, each dedicated to one of the tasks.

3. Task-Optimized Loss Functions:

- The separate models use task-specific loss functions during training. Cross-entropy loss is employed for concept prediction, ensuring accurate classification of objects into predefined categories. For feature prediction, mean squared error (MSE) loss is utilized to guarantee precise numerical estimations of object attributes.

4. Data Handling:

- Data Handling: The data pipeline is tailored to the dual-task nature of the models. It involves organizing and preprocessing the data to facilitate both concept and feature predictions.

Key Modifications

1. Concept and Feature Specialization:

- The most notable modification lies in the incorporation of two separate tasks within the same architecture. The model is equipped to simultaneously predict object concepts and associated features, providing a holistic understanding of the objects in question.

2. Parallel Processing:

- By employing separate models for concepts and features, the architecture allows for parallel processing of images. This means that while one model predicts concepts, the other simultaneously predicts features, reducing computation time..

3. Customized Loss Functions:

- Task-specific loss functions are employed to optimize the models for their individual tasks. Cross-entropy loss prioritizes correct concept classification, while mean squared error (MSE) loss emphasizes precise feature estimation.

4. Enhanced Predictive Capacity:

- The specialization and separation of models enhance their predictive capacity. Each model becomes an expert in its designated task, leading to more accurate and informative results for both concepts and features.

In summary, the separate models for concepts and features represent a departure from the original AlexNet by specializing in dual tasks, employing task-specific loss functions, and facilitating parallel processing. These modifications enhance the models' capacity to provide both categorical and quantitative insights into the content of images, resulting in a more comprehensive understanding of objects beyond traditional image classification.

Insufficiencies (to be improved)

Abstract

There are some significant shortcomings in both the Unified Model, which jointly predicts concepts and features, and the Separate Models, which separate these tasks. Notably, misclassifications, such as labeling 'Yogurt' as 'bathtub,' underscore the need for improvement.

Unified Model vs. Separate Models:

Comparative tests have shown that the Unified Model often outperforms the Separate Models. For example, the Unified Model correctly identified 'boot_01b.jpg' as a 'boot' with 'is_long,' while the Separate Models struggled with misclassifications.

The Interplay of Concepts and Features:

These findings raise the question of whether feature recognition inherently aids conceptual identification. The poorer performance of Separate Models hints at the intricate relationship between features and concepts in image recognition.

Future Directions:

Addressing these insufficiencies requires further research. Potential strategies include architectural refinements, leveraging complementary information from features and concepts, and advanced training techniques. Our goal is to enhance conceptual and feature recognition, ultimately advancing image analysis systems.

CS FINAL PROJECT

D E M O

Multi-task learning of objects and parts

Thank you for watching!

Hailin Weng 12/01/2024