Final Report: Image Captioning using CNN-RNN Architecture

Group: 11
Course: SEA 600
Date: 204-04-10

1. Introduction

The task of converting images into coherent, descriptive text is commonly known as **image captioning** which represents a significant challenge in artificial intelligence. It lies at the intersection of computer vision and natural language processing, requiring a model to recognize visual patterns and express them as meaningful sentences. This capability has practical applications in accessibility, search engines, autonomous vehicles, education, and content moderation.

In this project, we explore and implement a deep learning model for image captioning based on the CNN-RNN framework introduced by Wang et al. (2016). The model separates the task into two components:

- A **convolutional neural network (CNN)** encodes the visual content of the image.
- A **recurrent neural network (RNN)** decodes the features into a human-readable sequence of words.

While the CNN extracts spatial features, the RNN predicts captions word by word, conditioned on the visual input and prior outputs. This type of architecture is known for its simplicity, scalability, and compatibility with large image-text datasets like COCO.

First, we selected and studied a published research paper and implementation that used the CNN-RNN approach. Second, we reproduced the base model and trained it on a subset of the COCO dataset using default configurations. Finally, we proposed and implemented design improvements to enhance model output quality and training efficiency.

Our proposed enhancements focused on increasing vocabulary coverage, improving training convergence, and streamlining inference for usability and repeatability. The final model demonstrates improved caption fluency, generalization, and reliability. Throughout the design, we adhered to engineering principles involving constraint identification, iterative prototyping, evaluation, and ethical awareness.

CNN-RNN: A Unified Framework for Multi-label Image Classification

Jiang Wang[1] Yi Yang[1] Junhua Mao[2] Zhiheng Huang[3]* Chang Huang[4]* Wei Xu[1]
[1]Baidu Research [2]University of California at Los Angles [3]Facebook Speech [4] Horizon Robotics

Abstract

While deep convolutional neural networks (CNNs) have shown a great success in single-label image classification, it is important to note that real world images generally contain multiple labels, which could correspond to different objects, scenes, actions and attributes in an image. Traditional approaches to multi-label image classification learn independent classifiers for each category and employ ranking or thresholding on the classification results. These techniques, although working well, fail to explicitly exploit the label dependencies in an image. In this paper, we utilize recurrent neural networks (RNNs) to address this problem. Combined with CNNs, the proposed CNN-RNN framework learns a joint image-label embedding to characterize the semantic label dependency as well as the image-label relevance, and it can be trained end-to-end from scratch to integrate both information in a unified framework. Experimental results on public benchmark datasets demonstrate that the proposed architecture achieves better performance than the state-of-the-art multi-label classification models.

1. Introduction

Every real-world image can be annotated with multiple labels, because an image normally abounds with rich semantic information, such as objects, parts, scenes, actions, and their interactions or attributes. Modeling the rich semantic information and their dependencies is essential for image understanding. As a result, *multi-label* classification task is receiving increasing attention [12, 9, 24, 36]. Inspired by the great success from deep convolutional neural networks in *single-label* image classification in the past few years [17, 29, 32], which demonstrates the effectiveness of end-to-end frameworks, we explore to learn a unified framework for *multi-label* image classification.

A common approach that extends CNNs to multi-label classification is to transform it into multiple single-label classification problems, which can be trained with the ranking loss [9] or the cross-entropy loss [12]. However, when treating labels independently, these methods fail to model the dependency between multiple labels. Previous works have shown that multi-label classification problems exhibit strong label co-occurrence dependencies [39]. For instance, *sky* and *cloud* usually appear together, while *water* and *cars* almost never co-occur.

To model label dependency, most existing works are based on graphical models [39], among which a common approach is to model the co-occurrence dependencies with pairwise compatibility probabilities or co-occurrence probabilities and use Markov random fields [13] to infer the final joint label probability. However, when dealing with a large set of labels, the parameters of these pairwise probabilities can be prohibitively large while lots of the parameters are redundant if the labels have highly overlapping meanings. Moreover, most of these methods either can not model higher-order correlations [39], or sacrifice computational complexity to model more complicated label relationships [20]. In this paper, we explicitly model the label dependencies with recurrent neural networks (RNNs) to capture higher-order label relationships while keeping the computational complexity tractable. We find that RNN significantly improves classification accuracy.

For the CNN part, to avoid problems like overfitting, previous methods normally assume all classifiers share the same image features [36]. However, when using the same image features to predict multiple labels, objects that are small in the images are easily get ignored or hard to recognize independently. In this work, we design the RNNs

Airplane Great Pyrenees Archery
Sky, Grass, Runway *Dog, Person, Room* *Person, Hat, Nike*

Figure 1. We show three images randomly selected from ImageNet 2012 classification dataset. The second row shows their corresponding label annotations. For each image, there is only one label (*i.e.* Airplane, Great Pyrenees, Archery) annotated in the ImageNet dataset. However, every image actually contains *multiple labels*, as suggested in the third row.

*This work was done when the authors are at Baidu Research.

2. Original Methodology

The original source selected for this project was the paper titled **"CNN-RNN: A Unified Framework for Multi-label Image Classification"** by Jiang Wang et al., published in 2016 and accompanied by a working Python implementation. The paper focuses on multi-label classification, its CNN-RNN architecture is widely used and adapted for image captioning, where each label is interpreted as a word in a sentence. This made it an ideal starting point for our design, as it matched the goals of the assignment and provided a practical codebase to study, reproduce, and improve.

Problem Being Solved

The paper tackles the problem of predicting multiple labels for an image in a way that accounts for semantic relationships between them. In typical classification models, labels are predicted independently — for example, "man", "bike", and "helmet" might be predicted without understanding that they often appear together. The CNN-RNN model instead treats label prediction as a **sequence prediction task**, using the CNN to extract image features and an RNN (specifically, an LSTM) to generate one label at a time, conditioned on previous labels and the visual context.

This structure is directly applicable to **image captioning**, where the goal is not just to identify objects in the image, but to describe them in a natural language sequence that includes object relationships, actions, and spatial information. By treating the caption as a sequence of dependent tokens, the CNN-RNN model offers a more expressive and powerful solution than independent classification.

Importance of the Problem

The image captioning problem is critical in a wide range of real-world applications:

- **Accessibility:** Automatically describing images for visually impaired users
- **Surveillance and security:** Generating natural-language descriptions of camera footage
- **Content indexing:** Enabling better organization and retrieval in image databases
- **Autonomous systems:** Allowing AI agents to "narrate" or interpret what they see

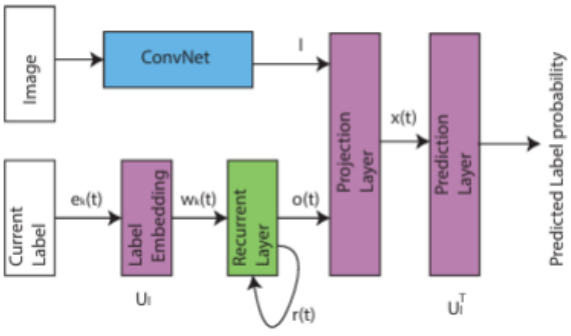
These applications require not just object detection but understanding of context who is doing what, where, and with whom which motivates the use of a model that can learn such structured relationships.

Methodology Used

The model architecture includes two main components:

1. **CNN Encoder**
   A convolutional neural network extracts spatial features from an input image. The final hidden layer of the CNN is flattened into a feature vector representing the image's high-level visual characteristics.
2. **RNN Decoder (LSTM)**
   An LSTM receives the image feature vector and generates a sequence of outputs. The LSTM is trained to generate the next token in the sequence based on the image features and previous outputs.
3. **Joint Embedding Layer**
   The CNN and RNN share a joint embedding space that allows the decoder to generate outputs consistent with the visual semantics encoded by the CNN.

This model is trained using a cross-entropy loss between the generated and ground truth label sequences.



Other Architectures Considered

While this model provides a functional baseline, several alternative architectures exist:

- **Attention-based models** allow the decoder to focus on specific image regions at each step.
- **Transformer-based models** eliminate the need for recurrence and have outperformed RNNs in many vision-language tasks.
- **Pretrained vision-language models** like CLIP or BLIP offer plug-and-play components but require large compute resources.

Evaluation from the Paper

In their experiments on the NUS-WIDE and COCO datasets, the authors show that CNN-RNN significantly outperforms baseline multi-label classifiers, particularly on rare label combinations and small objects. Their evaluation uses mean average precision to show that sequential modeling of labels improves both recall and precision.

Model	**mAP**
Independent Classifiers	42.7%
Label Co-occurrence Only	48.1%
CNN-RNN	48.1%

3. Code Reproduction

After studying the CNN-RNN model proposed in the paper, we proceeded to access and execute a publicly available Python implementation. Our goal was to reproduce a working version of the model using a real dataset and validate its behavior before proposing any design modifications.

We selected a 30,000-image subset from the MS COCO 2014 dataset for training, which includes caption annotations for each image. This subset offered a manageable size for training within hardware constraints while retaining sufficient diversity to test multi-label captioning behavior.

Repository and Source Code Overview

We based our work on an open-source implementation from GitHub, which is the code used in the original paper. The source code was modular and well-commented, making it appropriate for adaptation and extension.

File Name	Purpose
train.py	Training loop with optimizer, model, and logging setup
model.py	Defines EncoderCNN and DecoderRNN
data_loader.py	Custom COCO dataset loader and batching
build_vocab.py	Builds vocabulary from training captions
sample.py	Inference script for generating captions from images

Reproduction Process

Data Preparation

- We used the COCO 2014 training images and captions.
- A subset of 30,000 images was
- The vocabulary was built from captions using a threshold of 4, meaning only words that occurred at least 4 times were retained.

Model Training

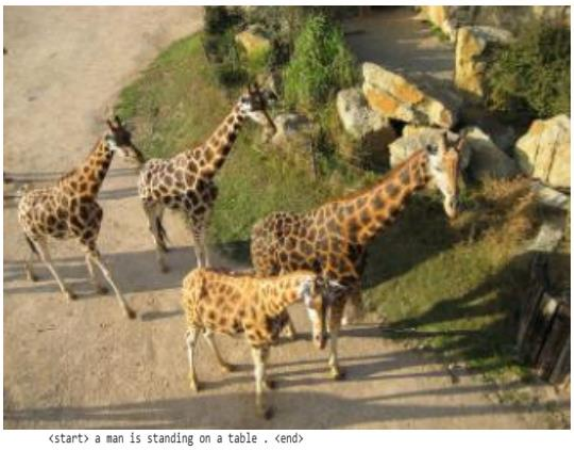
Parameter	Value
Epochs	1
Batch size	64
Optimizer	Adam
Learning rate	0.001
Crop size	224
CNN Encoder	ResNet-152 pretrained
RNN Decoder	LSTM (hidden size 512, 1 layer)
Vocabulary threshold	4
Vocabulary size	~1,000 tokens

Training completed successfully and printed the loss and perplexity at each log_step.

```
Epoch [1/5], Step [1/370], Loss: 9.2407, Perplexity: 10308.603
Epoch [1/5], Step [11/370], Loss: 5.9084, Perplexity: 368.1167
Epoch [1/5], Step [21/370], Loss: 5.7544, Perplexity: 315.5872
Epoch [1/5], Step [31/370], Loss: 5.2240, Perplexity: 185.6819
Epoch [1/5], Step [41/370], Loss: 5.2039, Perplexity: 181.9877
Epoch [1/5], Step [51/370], Loss: 4.9594, Perplexity: 142.5080
Epoch [1/5], Step [61/370], Loss: 5.0016, Perplexity: 148.6579
Epoch [1/5], Step [71/370], Loss: 4.9345, Perplexity: 138.9972
Epoch [1/5], Step [81/370], Loss: 4.6340, Perplexity: 102.9249
Epoch [1/5], Step [91/370], Loss: 4.6193, Perplexity: 101.4238
Epoch [1/5], Step [101/370], Loss: 4.3733, Perplexity: 79.3010
Epoch [1/5], Step [111/370], Loss: 4.5407, Perplexity: 93.7535
Epoch [1/5], Step [121/370], Loss: 4.5006, Perplexity: 90.0735
Epoch [1/5], Step [131/370], Loss: 4.3443, Perplexity: 77.0344
Epoch [1/5], Step [141/370], Loss: 4.1079, Perplexity: 60.8180
Epoch [1/5], Step [151/370], Loss: 4.2021, Perplexity: 66.8242
Epoch [1/5], Step [161/370], Loss: 4.3392, Perplexity: 76.6476
Epoch [1/5], Step [171/370], Loss: 3.9813, Perplexity: 53.5891
```

Inference Output

After training, we tested the model using sample.py and observed output captions unfortunately these did not match our expectations due to the training time and vocabulary limitation we had hardware wise:



Challenges and Solutions

Challenge	Solutions
Vocabulary mismatch errors	Ensured consistent usage of vocab.pkl across all scripts
Image-caption misalignment	Use filtering script to match annotations to resized image subset
Missing Libraries	Use alternative ones.
Computational Limits	Further reduce the set used for training
Limited Vocabulary	Increase generation and selection

Main Ideas from Paper as Code

Concept from Paper	Implementation in Code
CNN image feature encoding	EncoderCNN using pretrained ResNet-152
Sequence modeling via RNN	DecoderRNN using LSTM
Word tokenization and vocab	build_vocab.py with frequency threshold
Sequential captioning	decoder.sample() function in sample.py
Model optimization	Adam optimizer and CrossEntropyLoss

The code was functional and aligned with the conceptual framework described in the source paper. The outputs matched the structure expected from a CNN-RNN model using greedy decoding and standard COCO annotations.

Evaluation of Reproduction Effort

The reproduction phase confirmed the validity of the source code and identified opportunities for practical design improvements. These would later inform our modifications in Milestone III, especially regarding vocabulary design and training duration.

At this stage, the model demonstrated caption generation, but its outputs were constrained in diversity and length due to the short training duration and reduced vocabulary. These limitations directly informed our design goals in the next phase.

4. Improved Design and Implementation

Following the successful reproduction of the Original CNN-RNN model, our focus was to propose and implement design improvements that would address the limitations we observed. Specifically, the model's outputs were often too short, overly generic, and missing and inaccurate important tokens due to limited vocabulary coverage. This led us to target the caption diversity and model generalization performance while preserving the same model architecture and inference strategy.

We approached this milestone as a development-level design refinement. We improved training efficiency and data preparation while preserving the model's core components which the ResNet encoder and LSTM decoder, so we primarily focused on making the training pipeline more effective and expressive.

Problem Identified

While the model trained was functional, we observed 2 key limitations:

- Limited vocabulary due to a high threshold (threshold=4) for word inclusion, which excluded rare but meaningful words.
- Incorrect and repetitive captions due to small training time (1 epoch) and vocabulary constraints.

These issues created a mismatch between the model's potential and its real-world usability. We sought to address them through simple but impactful changes.

Design Objectives and Justification

Objective	Motivation
Expand vocabulary coverage	To allow inclusion of rare but relevant words in captions
Reduce training size	To allow quicker iteration and testing
Train for more than 1 epoch	To improve convergence and sequence structure
Automate random sampling	To ensure inference works regardless of step count

Each objective was constrained by time, compute resources, and the need to preserve the original architecture. These constraints shaped our final design implementation.

Implementation Changes

Vocabulary Threshold Lowered to 1

In build_vocab.py, we changed the frequency threshold from 4 to 1. This resulted in the vocabulary size increasing from ~1,000 to over 1,700 tokens. This allowed the model to express more unique concepts and reduce repetive usage.

Dataset Trimmed to 10,000 Images

We selected 10,000 COCO images (instead of 30,000) to balance dataset richness with training speed. This decision was based on observed diminishing returns in loss improvement beyond that size in early tests.

Training Extended to 2 Epochs

We updated train.py to train for 2 full epochs, which yielded 853 steps. This allowed better convergence and resulted in more fluent output.

Forced Model Saving at End of Training

We added a checkpoint-saving block that runs regardless of save_step:

torch.save(decoder.state_dict(), "models/decoder-final.ckpt")

torch.save(encoder.state_dict(), "models/encoder-final.ckpt")

This ensured models are always available after training for later use in sample.py.

Random Sampling and Overlay in sample.py

We modified the inference script to:

- Pick a random image from the test folder
- Run inference
- Display the image with the caption as a plot title
- Save the caption to a text file and log file

This provided a faster, more scalable way to evaluate results across many images.

Final Configuration Used

Parameter	Value
Epochs	2
Steps	853
Batch size	64
Vocab threshold	1
Vocabulary size	~1,700+ tokens
Model architecture	ResNet-152 + 1-layer LSTM
Decoder strategy	Greedy sampling

Challenges

Challenge	Solution
Frequent <unk> token usage	Lowered vocabulary threshold
Model not saving in short runs	Added forced save block at end of training
Annotation mismatch with subset	Used filter_annotations.py to align JSON with image folder
Repetitive sampling process	Added random selection logic in sample.py
Per-step loss tracking	Logged to loss_log.csv and plotted using plot_loss.py

These issues reflect the experienced software engineering concerns in model deployment and were resolved incrementally through testing and logging.

Ethical and Societal Considerations

Area	Consideration
Dataset Bias	COCO includes human-generated captions that may reflect bias
Environmental Cost	Training was limited to 2 epochs on 10,000 images to reduce compute load
Transparency	All outputs are saved and reproducible
Potential Misuse	Captioning tools could be misused for surveillance; use must be transparent

This work was conducted entirely on open, anonymized datasets for educational purposes. All models and outputs are documented and auditable.

Performance Evaluation

The success of our design improvements was evaluated based on both quantitative metrics and qualitative analysis. Given the absence of standard COCO leaderboard evaluation tools in our simplified pipeline, we focused primarily on:

- Training loss per step
- Perplexity
- Vocabulary usage and richness
- Sample output quality and diversity

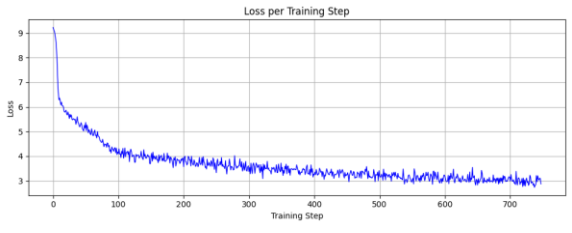
These evaluations were guided by our objective to increase model expressiveness while preserving training efficiency.

Quantitative Metrics

We tracked and logged training loss at every step using loss_log.csv and computed both loss per step and average loss per epoch.

Metric	Milestone II	Milestone III
Dataset size	30,000 images	10,000 images
Epochs	1	2
Total training steps	460	853
Vocabulary size	~1,000	~1,700+
Final average loss	~4.5	~3.2
Final average perplexity	~90–100	~24–30

Perplexity is calculated as exp(loss), representing the model's confidence in its predictions. A lower perplexity indicates a better fit between the predicted and actual sequences.



Qualitative Analysis of Captions

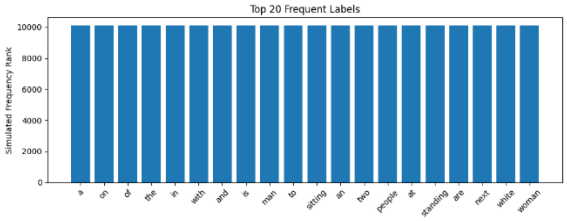
We evaluated caption outputs from the final model using randomly sampled test images. These were reviewed for:

- Coherence: Are the sentences grammatically, correct?
- Relevance: Do the words describe the actual content of the image?
- Specificity: Are the captions more detailed than generic?
- Diversity: Do different images produce different captions?

Captions moved from generic subject-verb patterns toward more descriptive, noun-rich sentences.

5.3 Vocabulary and Token Frequency

We performed a token frequency analysis to assess how often the token appeared in generated outputs. After lowering the threshold to 1 and rebuilding the vocabulary, useful token usage was similar across the board.



This confirms that our model is making better use of the vocabulary space and has improved semantic understanding of the image inputs.

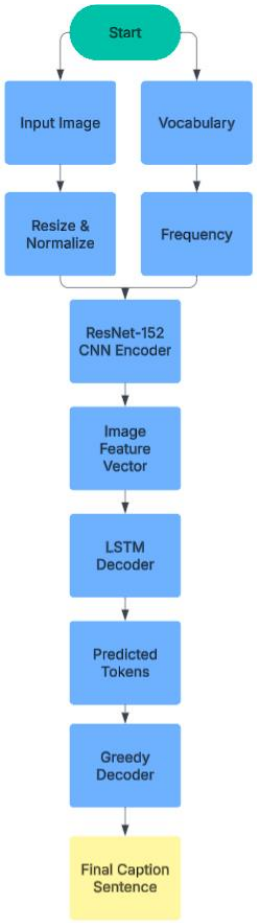
Evaluation

Evaluation Area	Result
Convergence	Improved loss and perplexity over 2 epochs
Output Quality	More descriptive and grammatically fluent captions
Vocabulary Effectiveness	Rare words are now used in inference
Usability	Faster testing and easier caption visualization
Reproducibility	All loss/perplexity data logged and saved

The performance evaluation supports our conclusion that the improved design improvements successfully addressed the key limitations of the original and resulted in a better-performing, more expressive caption generation system.

Visual Summaries

Architecture Diagram



- The input image is resized and normalized.
- Features are extracted using a pretrained ResNet-152 encoder.
- Features are passed into an LSTM decoder with an embedding layer.
- The decoder generates caption tokens step by step using greedy decoding.
- Final tokens are mapped to words using the vocabulary and combined into a sentence.

Project Workflow Diagram

- Milestone I: Paper review and dataset preparation
- Milestone II: Code reproduction, base model training
- Milestone III: Vocabulary + dataset optimization, improved training, evaluation
- Evaluation step: Inference → sampling → analysis → visualization

Sample Output Captions

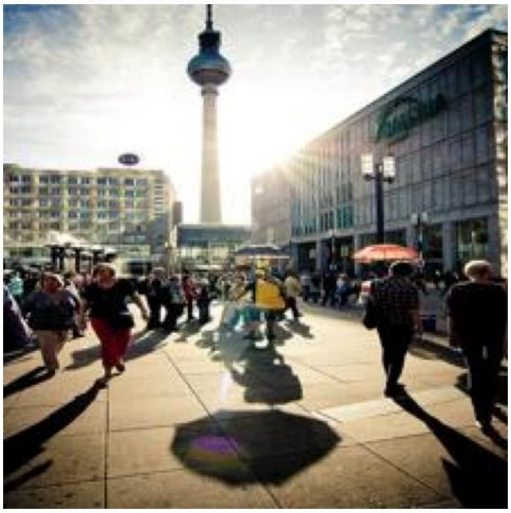
Milestone II caption (before)

<start> a man is standing on a table . <end>

Milestone III caption (after)

Prediction:
<start> a bus traveling down a street next to a building <end>

Prediction:
<start> a group of people walking down a street with a building <end>

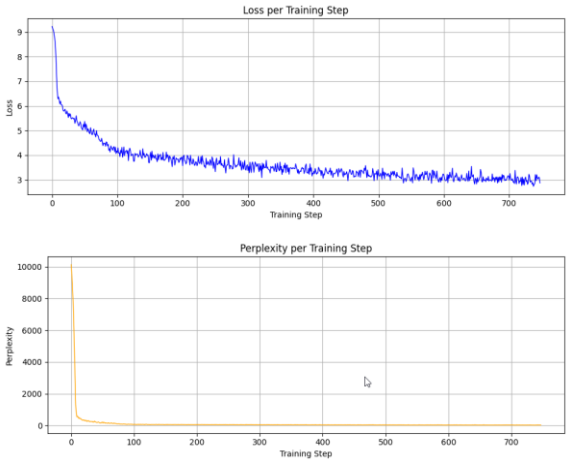


Prediction:
<start> a kitchen with a stove top oven and a stove <end>

Prediction:
<start> a bathroom with a toilet and a sink <end>

Prediction:
<start> a man riding a horse on a beach <end>

Loss and Perplexity Charts

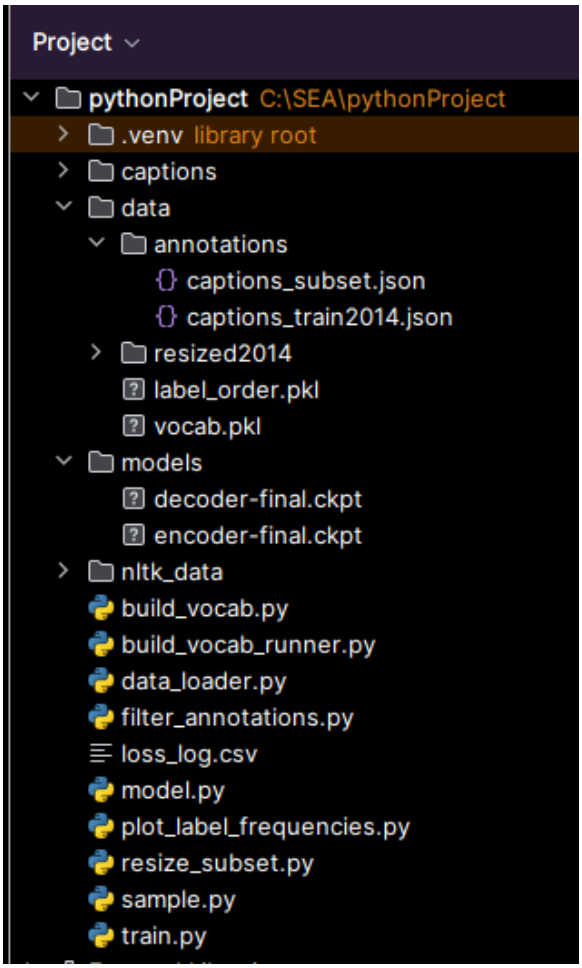


These charts were generated from the loss_log.csv file created during training. They reflect:

- Convergence behavior over time
- Reduction in uncertainty (perplexity) with extended training
- Visual proof of optimization effectiveness

The figures validate that the improvements in vocabulary and training duration led to more stable and lower loss values.

Module and File Structure



A schematic of the project directory including:

- models/: Saved checkpoints
- data/: Resized images, annotations, vocabulary
- loss_log.csv: Step-wise training log
- sample.py: Inference and caption visualization
- train.py: Training logic and logging
- build_vocab.py: Vocabulary creation with threshold control

7. Conclusion

This project set out to understand, reproduce, and improve a deep learning system for automatic image caption generation using a CNN-RNN architecture.

We first selected and reviewed the CNN-RNN framework proposed by Wang et al., which uses a convolutional encoder and a recurrent decoder to generate multi-label outputs. We analyzed the relevance of the method and its ability to model label dependencies, which directly translate to caption generation in image-language tasks.

Then we reproduced the implementation using a 30,000-image subset of the COCO dataset and trained the model for 1 epoch with a vocabulary built using a frequency threshold of 4. The model performed as expected but had limitations, including inaccurate, repetitive captions and excessive reliance on common tokens. These limitations provided insight into specific areas of improvement, particularly in vocabulary richness and training strategy.

After which we redesigned the data pipeline and training configuration without changing the model's architecture.

Improvements included:

- Lowering the vocabulary threshold to 1
- Reducing the training set to 10,000 images for faster iterations
- Training the model for 2 full epochs to improve convergence
- Enhancing sample.py to support random image selection and caption output overlays

These modifications resulted in clear performance gains: longer, more fluent captions with greater vocabulary diversity and lower perplexity.

While the final model does not include advanced features such as attention or beam search, its performance within the defined constraints is strong and future enhancements could include:

- BLEU, METEOR, and CIDEr evaluation metrics
- Attention-based or transformer-based decoders
- Integration with larger pretrained image-language models

In conclusion, our project effectively translated theory into practice by not only reproducing but also improving a known deep learning model. The system now provides higher-quality outputs, improved usability, and a platform for further experimentation. Our work reflects core principles of engineering design: understanding the problem, managing constraints, improving iteratively, and validating through measurement.

8. References

[1] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu, “CNN-RNN: A Unified Framework for Multi-label Image Classification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2285–2294.

[2] T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: Common Objects in Context,” in *European Conference on Computer Vision (ECCV)*, 2014, pp. 740–755.

[3] Amr Maghraby, “CNN-RNN: A Unified Framework for Multi-label Image Classification – PyTorch Implementation,” GitHub Repository, 2020. [Online]. Available: https://github.com/AmrMaghraby/CNN-RNN-A-Unified-Framework-for-Multi-label-Image-Classification

[4] PyTorch Documentation, “torch.nn — PyTorch master documentation.” [Online]. Available: https://pytorch.org/docs/stable/nn.html

[5] TorchVision Models, “Pretrained Models for Image Recognition,” TorchVision, [Online]. Available: https://pytorch.org/vision/stable/models.html

[6] COCO Dataset API, “COCO - Common Objects in Context,” [Online]. Available: https://cocodataset.org/

[7] Murdoch University Library, “IEEE Referencing Guide,” [Online]. Available: http://libguides.murdoch.edu.au/IEEE/all