Update (1/2)

- 11/2:
  - We recommend you first download all the pretrained models in the `hw3_download.sh` (e.g python -c "import clip; clip.load('ViT-B/32')") to avoid the time limited issue.
- 11/4:
  - Update the hw3_2.sh input args in p.34.
  - In the implementation of CLIPScore, you should use "ViT-B/32" as the vision encoder.
  - Update p.8 (jpg -> png).
- 11/5:
  - Update p.27 and p.33(jpg -> png).
- 11/6:
  - Update new package “pandas=1.5.1” in p.40.
- 11/13:
  - Since some classmates don’t understand the detailed implementation of `word embedding` and `position embedding` in decoder, we provide another tutorial, the annotated transformer, in p.51. This tutorial provide examples for them.

Homework #3

Deep Learning for Computer Vision

NTU, Fall 2022

111/11/1

111/11/21 (Mon.) 11:59 PM (GMT+8) due

Update (2/2)

- 11/14: We provide some **hints** here
  - In training, you can use `nn.Transformer.generate_square_subsequent_mask` as an input of decoder to create casual attention mask to achieve the purpose of teacher-forcing.
  - In testing, you can set a `max_len` hyperparameter, and autoregressively generate the caption until the output caption length exceeds `max_len` or the `[EOS]` is generated.
  - When you use the CrossEntropy as caption loss, you should ignore the `[PAD]` token by setting `nn.CrossEntropyLoss(ignore_index=tokenizer.token_to_id('[PAD]'))`
  - In testing, when you caculate CIDEr, you should first collect **all** the generated captions and **all** the GT captions. Then, use them to calculate the CIDEr. Because CIDEr computes tf-idf score over the whole dataset. Example:

```
predicts = []  # list
answers = []   # list of list
for batch_inputs, batch_gts in data_loader:
    captions = model.generate(batch_inputs)
    predicts.append(captions)
    answers.append(batch_gts)
CIDEr = evaluator.run_evaluation(predicts, answers)
```

Update (3/3)

- 11/15:
  - We decide to extend the inference time limit in problem2 to 40 mins in p.34.
  - We add the time limit (20 min) of download.sh in p.36.
- 11/16:
  - We provide our p2_evaluate.py for reference.
  - We lower the simple and strong CLIPScore baseline in the problem2 to 0.67 and 0.70, respectively in p.18.
  - We change the metric from CIDEr to CLIPScore for the second question in the problem3 report (p.24).
- 11/17:
  - There are 1789 images in val dataset in problem2.
  - We provide a more detailed pseudo code of CLIPScore in p.17.
- 11/20:
  - Add *skimage* package.

Outline

- Problems & Grading
- Dataset
- Submission & Rules
- Supplementary

Problems – Overview

- **Problem 1:** Zero-shot image classification with CLIP (30%)

  [hw3_data/p1_data]

- **Problem 2:** Image Captioning with Vision and Language Model (50%)

  [hw3_data/p2_data]

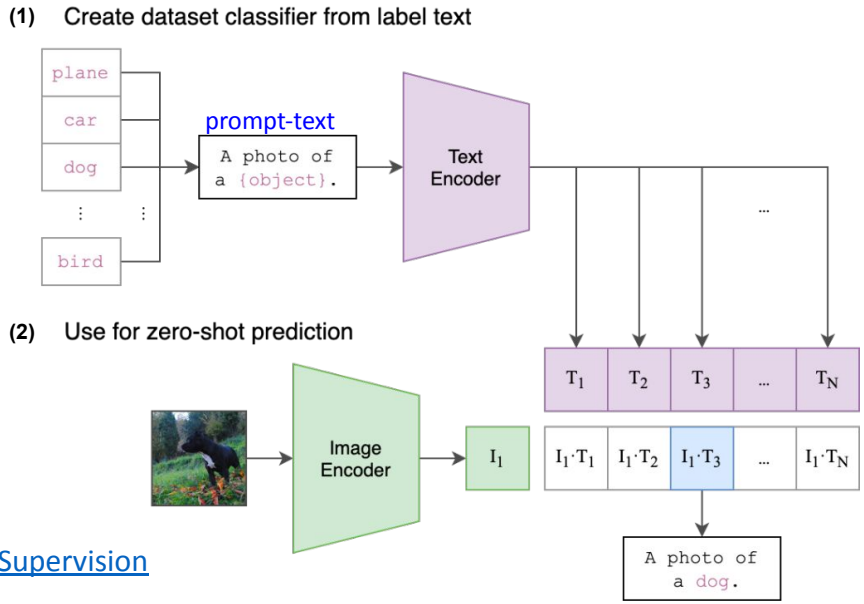
- **Problem 3:** Visualization of Attention in Image Captioning (20%)

  [hw3_data/p3_data]

Please refer to “Dataset” section for more details about p1_data/p2_data/P3_data datasets.

Problem 1: Zero-shot image Classification with CLIP

- Zero-shot: You **cannot** do any finetuning for the pretrained model.
- In this problem, you only need to evaluate the **pretrained** CLIP on image classification task.
  - Input:
    - image
    - text = [prompt-text; label set]
  - Output: class label
    (determined by similarity scores)



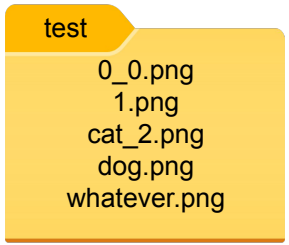
Reference: Learning Transferable Visual Models From Natural Language Supervision
GitHub: OpenAI-CLIP

Problem 1: Evaluation

- Evaluation metric: Accuracy
  - Accuracy is calculated over all test images.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

- Sample Output
  - Output format: CSV file with the first row being ‘filename,label’
  - Output filename must be identical to input filename



	A	B
1	filename	label
2	0_0.png	0
3	1.png	0
4	cat_2.png	0
5	dog.png	0
6	whatever.png	0

Problem 1: Grading - Baselines (15%)

- Public Baseline (10%) - Accuracy of 60% (validation data is in p1_data/val/)
- Private Baseline (5%) - TBD (testing data is not available for students)

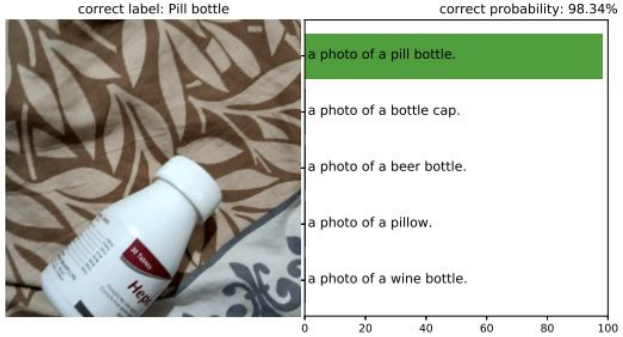
Problem 1: Grading - Report (15%)

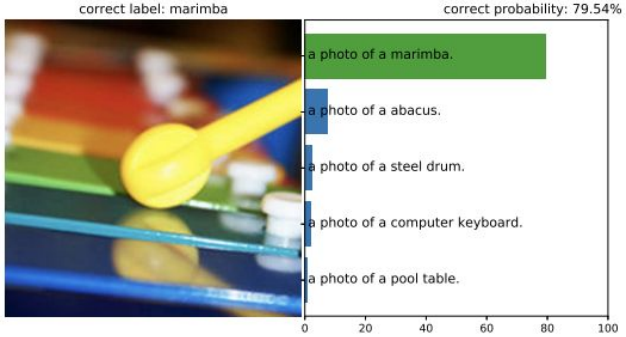
1. Methods analysis (3%)
    - Previous methods (e.g. VGG and ResNet) are good at one task and one task only, and requires significant efforts to adapt to a new task. Please explain why CLIP could achieve competitive zero-shot performance on a great variety of image classification datasets.
2. Prompt-text analysis (6%)
    - Please compare and discuss the performances of your model with the following **three** prompt templates:
        i. *“This is a photo of {object}”*
        ii. *“This is a {object} image.”*
        iii. *“No {object}, no score.”*

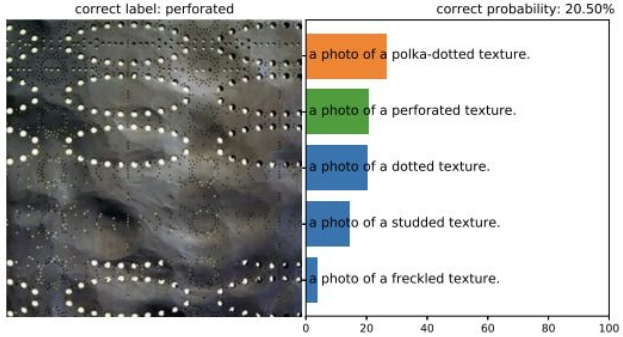
Problem 1: Grading - Report (cont'd) (15%)

3. Quantitative analysis (6%)

- Please sample **three** images from the validation dataset and then visualize the probability of the top-5 similarity scores as following example:



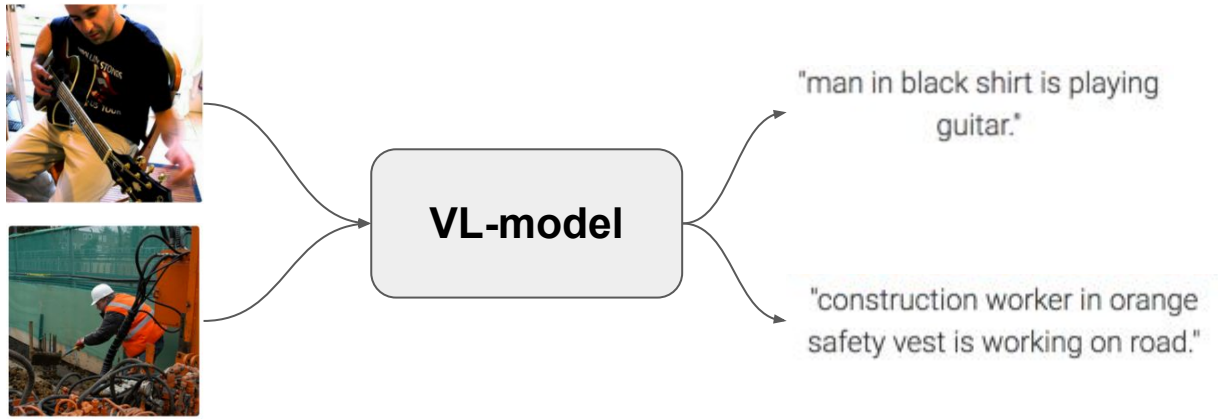




Problem 2: Image Captioning with VL-model

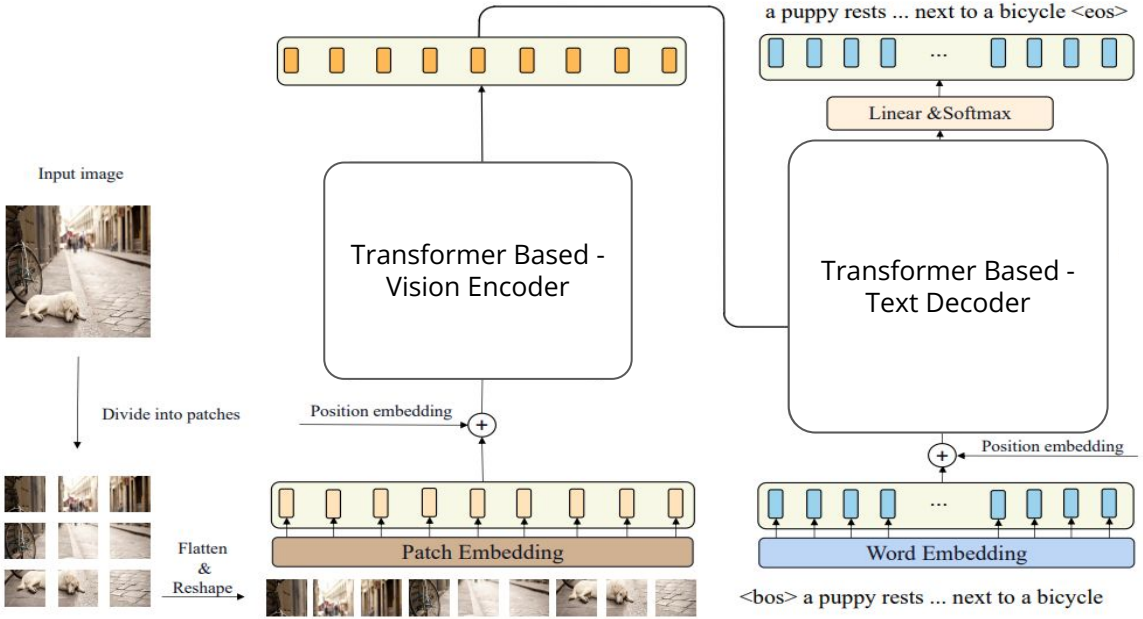
In this problem, you need to:

1. Implement a **Vision and Language (VL) model** for image captioning.
    - Input: RGB image
    - Output: image caption (text)
2. Implement a quantitative metric, namely CLIPScore, for evaluation.



Problem 2: Model overview

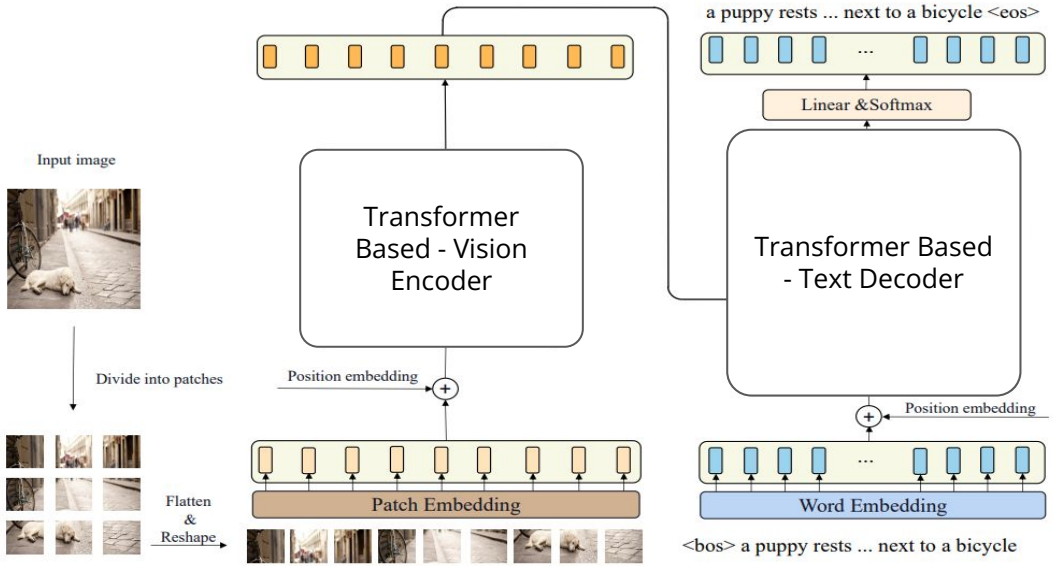
- You are limited to use **transformer encoder-decoder** architecture in this assignment.



Reference: CPTR: FULL TRANSFORMER NETWORK FOR IMAGE CAPTIONING

Problem 2: Model implementation

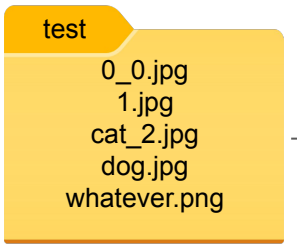
- You are limited to use vision transformer (ViT) as vision encoder. And, any **pretrained** ViTs (e.g. ViT-Base, CLIP-ViT-Base) are allowed.
- As for text decoder, you need to implement and train a transformer decoder **from scratch**.



Reference code: timm, pytorch-transformer

Problem 2: Evaluation

- Evaluation metric: CIDEr and CLIPScore (explained in next two page)
- Sample Output
  - Output format: json file with key and value being ‘filename’ and ‘predicted_caption’, respectively
  - Example:




```
{
    "0_0": "man in black shirt is playing gitar.",
    "1": "construction worker in orange safety vest is working.",
    "cat_2": "two young girls are playing with lego toy.",
    "dog": "a little girl in a pink hat is blowing bubbles.",
    "whatever": "two dogs play in the grass."
}
```

⚠ **Please remove the filename extension**

Problem 2: Evaluation - CIDEr

- CIDEr is an quantitative metric to evaluate whether the candidate sentences (your prediction) have high consensus with reference sentences (ground truth).
  - Candidate sentence will have high consensus if it captures the more high tf-idf words (distinct and frequent) in reference sentences.

Reference Sentences

R1: A bald eagle sits on a perch.

R2: An american bald eagle sitting on a branch in the zoo.

R3: Bald eagle perched on piece of lumber.

...

R50: A large bird standing on a tree branch.

Candidate Sentences

C1: An eagle is perched among trees.

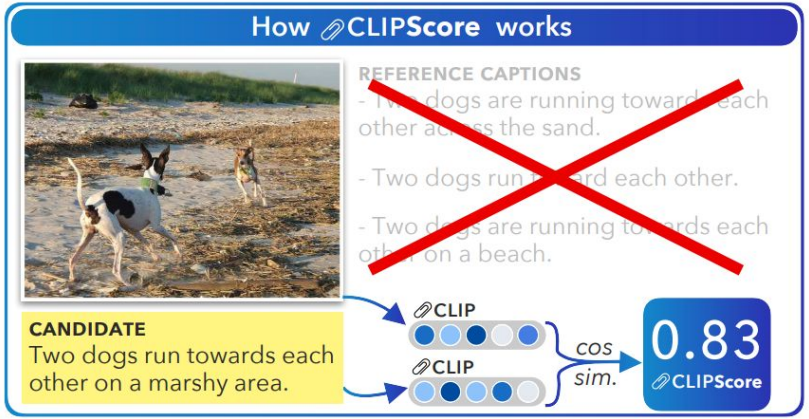
C2: A picture of a bald eagle on a rope stem.

CIDEr scores: C1 < C2

Reference: CIDEr: Consensus-based Image Description Evaluation
Tookit: coco evaluation

Problem 2: Evaluation - CLIPScore

- CLIPScore uses CLIP to assess image-caption compatibility without any ground truth, just like humans.
- Then, you need to implement the CLIPScore by yourself according to the following given pseudocode.




```
Class CLIPScore:
    def __init__(self):
        ...
    def __call__(self):
        ...
    def getCLIPScore(self, image, caption):
        """
        This function computes CLIPScore based on the pseudocode in the slides.
        Input:
            image: PIL.Image (e.g. RGB dogs image)
            caption: str (e.g. Two dogs run towards each other on a marshy area.)

        Return:
            cilp_score: float

        Pseudo code:
            w = 2.5

            # Hint: Use self.preprocess in constructor as img_preprocess
            image_input = img_preprocess(image)

            # Hint: Use clip.tokenize as tokenize
            text_input = tokenize(caption)

            with torch.no_grad():
                img_embed = CLIP.image_encode(image_input)
                text_embed = CLIP.text_encode(text_input)

            return w * max(cos-similarity(img_embed, text_embed), 0)
        """
        # TODO
        return NotImplementError
```

Reference: [CLIPScore: A Reference-free Evaluation Metric for Image Captioning]

Problem 2: Grading - Baselines (40%)

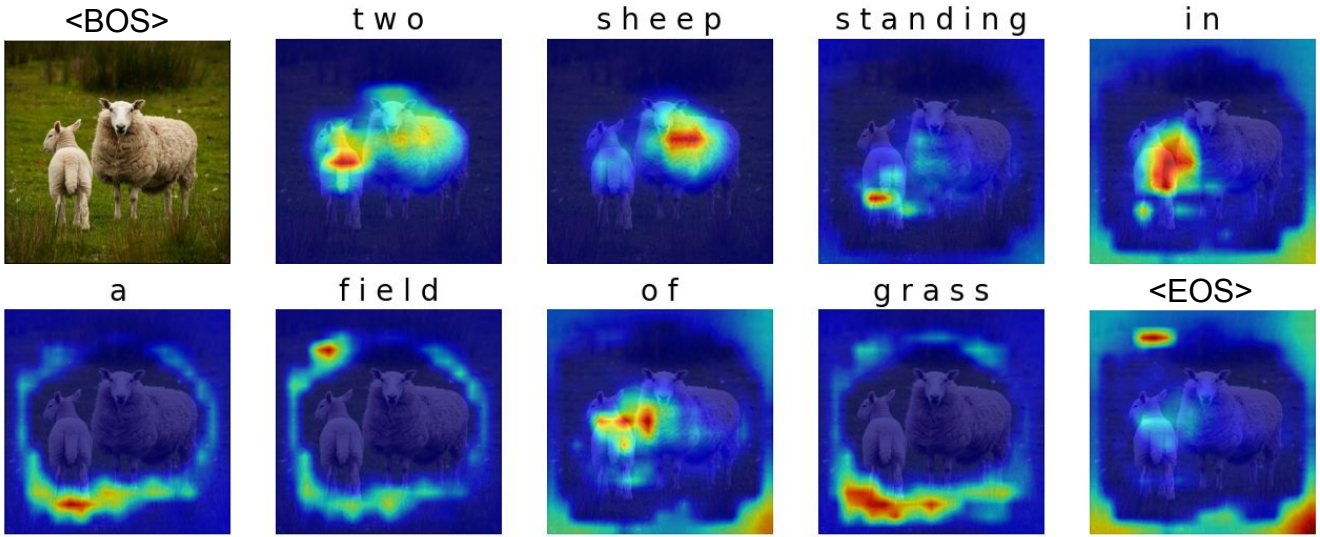
- Public Baseline (20%) - 1789 validation data ([p2_data/images/val])
  - Simple baseline (13%) - CIDEr of **0.72**, CLIPScore of **0.67**
  - Strong baseline (7%) - CIDEr of **0.87**, CLIPScore of **0.70**
- Private Baseline (20%) - 1404 test data (not available for students)
  - Simple baseline (13%) - TBD
  - Strong baseline (7%) - TBD

Problem 2: Grading - Report (10%)

1. Report your best setting and its corresponding CIDEr & CLIPScore on the validation data. (TA will reproduce this result) (2.5%)
2. Report other 3 different attempts (e.g. pretrain or not, model architecture, freezing layers, decoding strategy, etc.) and their corresponding CIDEr & CLIPScore. (7.5%, each setting for 2.5%)

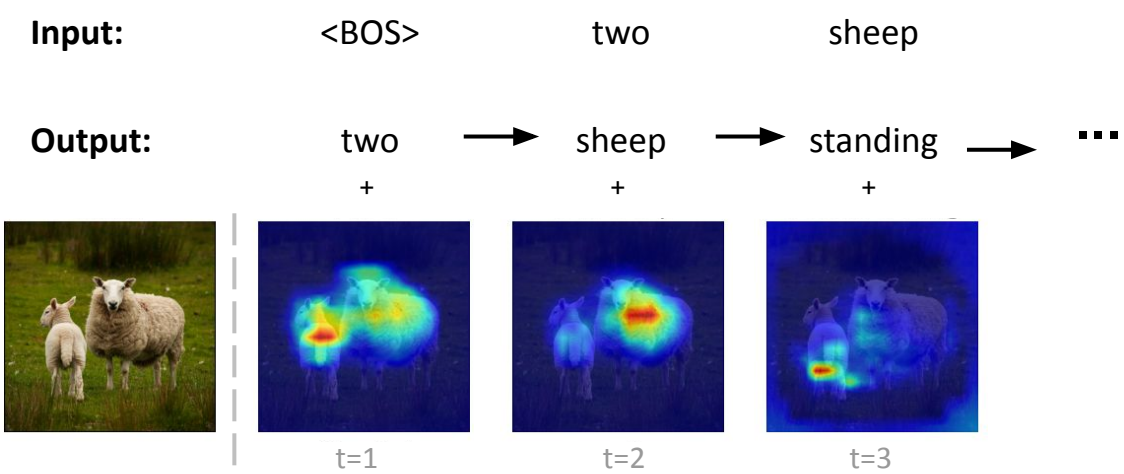
Problem 3: Visualization of Attention in Image Captioning

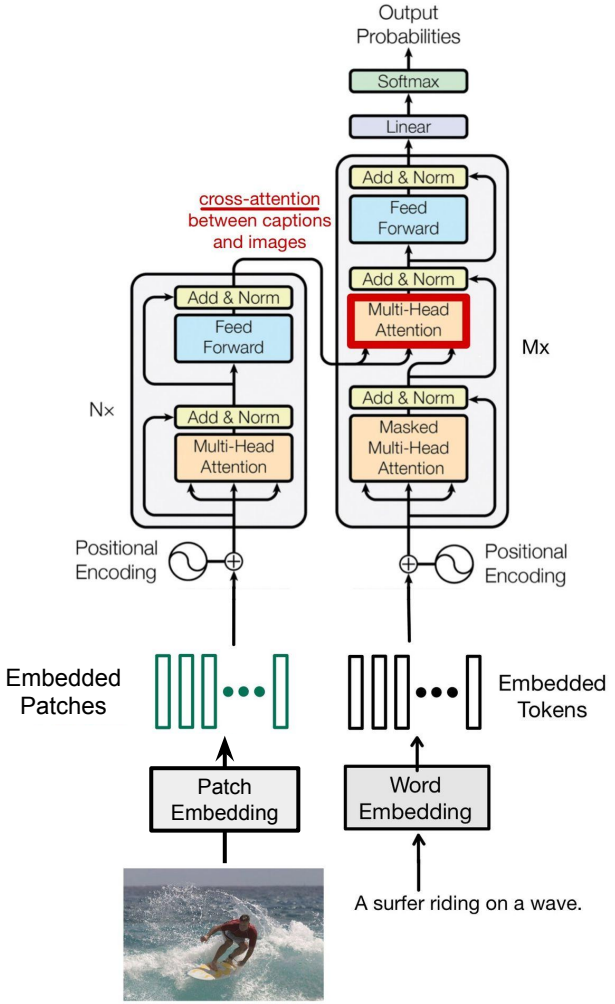
- In this problem, you have to analyze your image captioning model **in problem 2** by visualizing the **cross-attention** between images and generated captions.



Problem 3: Models and Settings

- Given an input image, your model would be able to generate a corresponding caption sequentially, and you have to visualize the cross-attention between the image patches and each predicted word in your own caption.

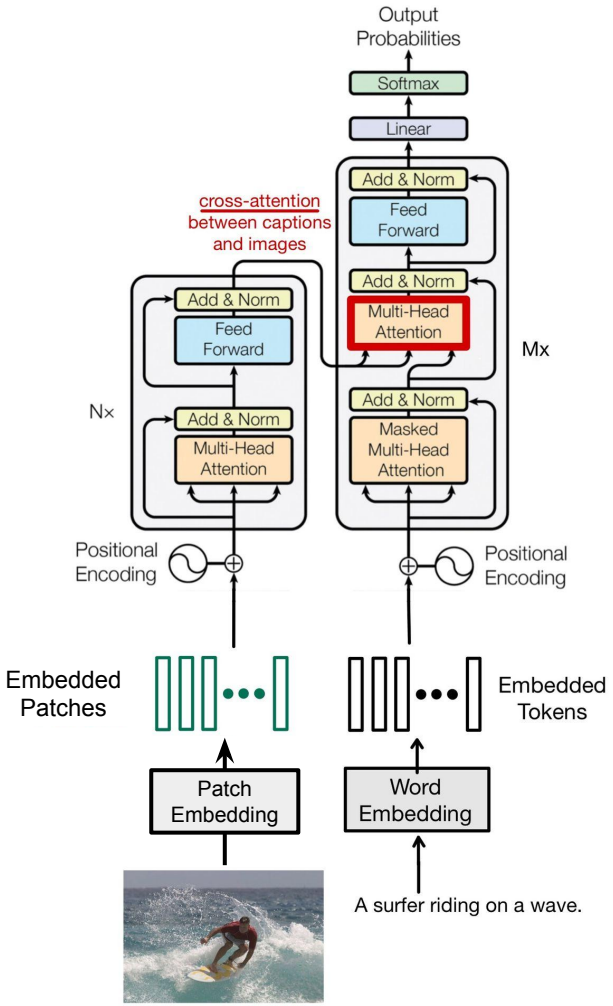




Problem 3: Models and Settings

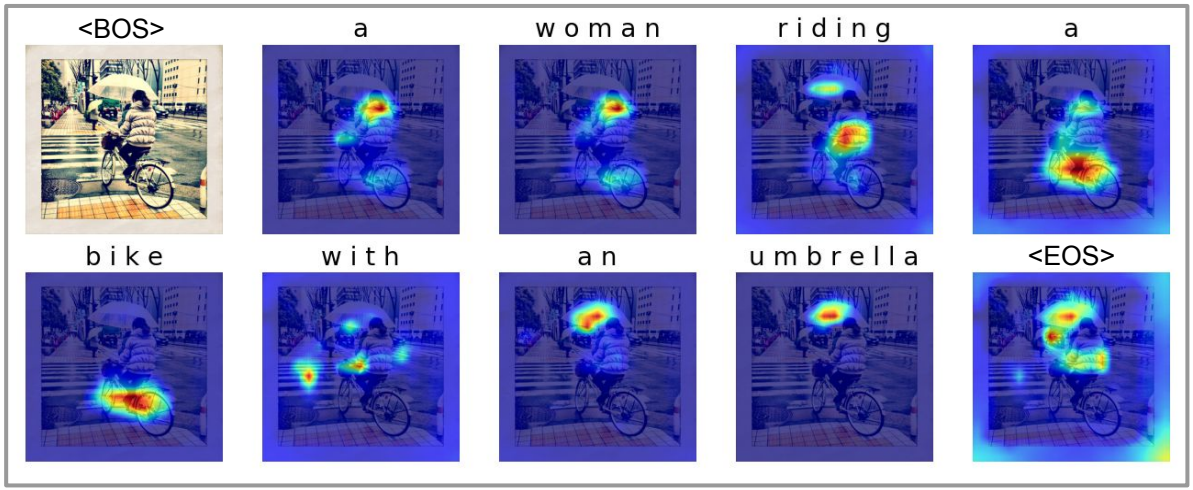
- The cross-attention weights you have to visualize are in the **decoder** part of the Transformer architecture.
  - The cross-attention weights in the last decoder layer (or the second to last) have better visulization results.

- **Practically, you should modify the code in problem 2 to get the attention weights for visualization.**



Problem 3: Grading - Report (20%)

1. TA will give you five test images ([p3_data/images/]), and please visualize the **predicted caption** and the corresponding series of **attention maps** in your report with the following template: (10%, each image for 2%)



In your report, a visualization example for bike.jpg should be like above.

Problem 3: Grading - Report (Cont'd) (20%)

2. According to **CLIPScore**, you need to visualize:
    i. top-1 and last-1 image-caption pairs
    ii. its corresponding CLIPScore

   in the validation dataset of problem 2. (5%)

3. Analyze the predicted captions and the attention maps for each word according to the previous question. Is the caption reasonable? Does the attended region reflect the corresponding word in the caption? (5%)

Outline

- Problems & Grading
- Dataset
- Submission & Rules
- Supplementary

Tools for Dataset

- Download the dataset
  - (Option 1) Manually download the dataset

    https://drive.google.com/file/d/1hJCiOeuLoICaKaFMsXMByl89b6jNmgz9/view?usp=share_link
  - (Option 2) Run the bash script provided in the hw3 repository

    $ bash get_dataset.sh

Problem 1: Dataset

- The dataset is the same as the dataset in problem 1 of homework1, which consists of images in **50** classes
- Since problem 1 is a zero-shot task, we don't provide training data.
- We only provide validation data for evaluation:
  - p1_data/val/
    - **2500** images
    - Images are named '{class_label}_{image_id}.png'
  - p1_data/id2label.json
    - This file contains mapping from *class_label* to *class_name*.

Problem 2: Dataset

- The dataset consists of images with each of them contains multiple captions.
- Format

p2_data/

```
├── images/
│   ├── train/    # 10604 images for training
│   └── val/      # 1789 images for validation
├── train.json    # captions for each training image
└── val.json      # captions for each val image
```


```
{
    "info": "dlcv2022-hw3-train",
    "licenses": "",
    "annotations": [
        {
            "caption": "a young girl inhales with the intent of blowing out a candle. ",
            "id": 152106,
            "image_id": 60623
        },...
    ],
    "images": [
        {
            "file_name": "000000060623.jpg",
            "id": 60623
        },...
    ]
}
```


A brief example for train.json

Problem 3: Dataset - MS COCO 2017

- COCO (Common Objects in Context) is a large-scale object detection, segmentation, and captioning dataset. Each image has five captions.
- We provide **five** images (from MS COCO 2017 Explore) for captioning with these five images.
- These five test images and as below:
  - **p3_data/images/**
    - **bike.jpg**
    - **girl.jpg**
    - **sheep.jpg**
    - **ski.jpg**
    - **umbrella.jpg**





Outline

- Problems & Grading
- Dataset
- Submission & Rules
- Supplementary

Submission

- Click the following link and sign in to your GitHub account to get your submission repository

  https://classroom.github.com/a/T5GYWdaX

  - You should connect your Github account to the classroom with your **student ID**
  - If you cannot find your student ID in the list, please contact us (ntudlcv@gmail.com)
- By default, we will only grade your last submission before the deadline (**NOT** your last submission). Please e-mail the TAs if you'd like to submit another version of your repository, and let us know which commit to grade.
- We will clone the **main** branch of your repository.

Submission

- Your GitHub repository should include the following files
  - hw3_<studentID>.pdf (report)
  - hw3_1.sh (for Problem 1)
  - hw3_2.sh (for Problem 2)
  - Python files (e.g., training code & inference code & visualization code)
  - Model files (can be loaded by your python file)
- **Don’t push the dataset to your repo.**
- If any of the file format is wrong, you will get zero point.

Bash Script - Problem 1

- TA will run your code as shown below
  - bash hw3_1.sh $1 $2 $3
    - $1: path to the **folder** containing test images (images are named xxxx.png, where xxxx could be any string)
    - $2: path to the **id2label.json**
    - $3: path of the output **csv file** (e.g. output_p1/pred.csv)
- Please follow the naming rules in **p.8**
- Note that you should **NOT** hard code any path in your file or script.
- Your testing code have to be finished in **10 mins.**

Bash Script - Problem 2

- TA will run your code as shown below
  - bash hw3_2.sh $1 $2
    - $1: path to the **folder** containing test images (e.g. hw3/p2_data/images/test/)
    - $2: path to the output **json file** (e.g. hw3/output_p2/pred.json)
- Please follow the naming rules in **p.15**
- Note that you should **NOT** hard code any path in your file or script.
- Your testing code have to be finished in **40 mins.**

Bash Script (cont'd)

- You must **not** use commands such as **rm**, **sudo**, **CUDA_VISIBLE_DEVICES**, **cp**, **mv**, **mkdir**, **cd**, **pip** or other commands to change the Linux environment.
- In your submitted script, please use the command **python3** to execute your testing python files.
  - For example: python3 test.py $1 $2
- We will execute you code on **Linux** system, so try to make sure you code can be executed on Linux system before submitting your homework.

Rules – Submission

- If your model checkpoints are larger than GitHub's maximum capacity (50 MB), you could download and preprocess (e.g. unzip, tar zxf, etc.) them in hw3_download.sh.
  - TAs will run `bash hw3_download.sh` prior to any inference if the download script exists, i.e. it is **NOT** necessary to create a blank `hw3_download.sh` file.
- Do **NOT** delete your model checkpoints before the TAs release your score and before you have ensured that your score is correct.
- Your download script have to be finished in **20 mins.**

Rules – Submission

- Please use **wget** to download the model checkpoints from cloud drive (e.g. Dropbox) or your working station.
    - You should use **-O argument** to specify the filename of the downloaded checkpoint.
- Please refer to this Dropbox Guide for a detailed tutorial.
- Google Drive is a widely used cloud drive, so it is allowed to use **gdown** to download your checkpoints from your drive.
    - It is also recommended to use -O argument to specify the filename.
    - Remember to set the permission visible to public, otherwise TAs are unable to grade your submission, resulting in zero point.
    - If you have set the permission correspondingly but failed to download with **gdown** because of Google's policy, TAs will manually download them, no worries!!

Rules – Environment

- Ubuntu 20.04.1 LTS
- NVIDIA GeForce RTX 2080 Ti (11 GB)
- GNU bash, version 5.0.17(1)-release
- Python 3.8

Rules – Environment

- Ensure your code can be executed successfully on **Linux** system before your submission.
- Use only **Python3** and **Bash** script conforming to our environment, do not use other languages (e.g. CUDA) and other shell (e.g. zsh, fish) during inference.
    - Use the command **“python3”** to execute your testing python files.
- You must **NOT** use commands such as **sudo, CUDA_VISIBLE_DEVICES** or other commands to interfere with the environment; **any malicious attempt against the environment will lead to zero point in this assignment**.
- You shall **NOT** hardcode any path in your python files or scripts, while the dataset given would be the absolute path to the directory.

Packages

- imageio==2.21.3
- matplotlib==3.6.1
- numpy==1.23.4
- Pillow==9.2.0
- scipy==1.9.1
- opencv-python==4.6.0.66
- tokenizers==0.13.1
- pycocotools==2.0.5
- timm==0.6.11
- torch==1.12.1
- torchvision==0.13.1
- pandas==1.5.1
- tqdm, gdown, glob, yaml, skimage
- language-evaluation、clip (please follow these github repos to install the packages)
- Any dependencies of above packages, and other standard python packages

- **E-mail or ask TA first if you want to import other packages.**

Packages and Reminders

- Python==3.8
- Do not use **imshow()** or **show()** in your code or your code will crash.
- Use **os.path.join** to deal with path as often as possible.
- If you train on GPU ids other than 0, remember to deal with the "**map location**" issue when you load model. (More details about this issue, please refer to https://github.com/pytorch/pytorch/issues/15541)

Deadline and Academic Honesty

- Deadline: **111/11/21 (Mon.) 11:59 PM (GMT+8)**
- Late policy : Up to 3 free late days in a semester. After that, late homework will be deducted 30% each day.
- Taking any unfair advantages over other class members (or letting anyone do so) is strictly prohibited. Violating university policy would result in F for this course.
- Students are encouraged to discuss the homework assignments, but you must complete the assignment by yourself. TA will compare the similarity of everyone's homework. Any form of cheating or plagiarism will not be tolerated, which will also result in F for students with such misconduct.

Penalty

- If we cannot execute your code, TAs will give you a chance to make minor modifications to your code. After you modify your code,
  - If we can execute your code, you will still receive a 30% penalty in your model performance score.
  - If we still cannot execute your code, no point will be given.

Reminder

- Please start working on this homework as early as possible.
- The training may take hours on a GPU or days on CPUs.
- Please read and follow the HW rules carefully.
- If not sure, please ask your TAs!

How to Find Help

- Google!
- Use TA hours (please check course website for time/location).
- Post your question under HW3 discussion section on NTU COOL.
- Contact TAs by e-mail: ntudlcv@gmail.com.

DOs and DONTs for the TAs (& Instructor)

- Do NOT send private messages to TAs via Facebook.
- TAs are happy to help, but they are not your tutors 24/7.
- TAs will NOT debug for you, including addressing coding, environmental, library dependency problems.
- TAs do NOT answer questions not related to the course.
- If you cannot make the TA hours, please email the TAs to schedule an appointment instead of stopping by the lab directly.

Outline

- Problems & Grading
- Dataset
- Submission & Rules
- Supplementary

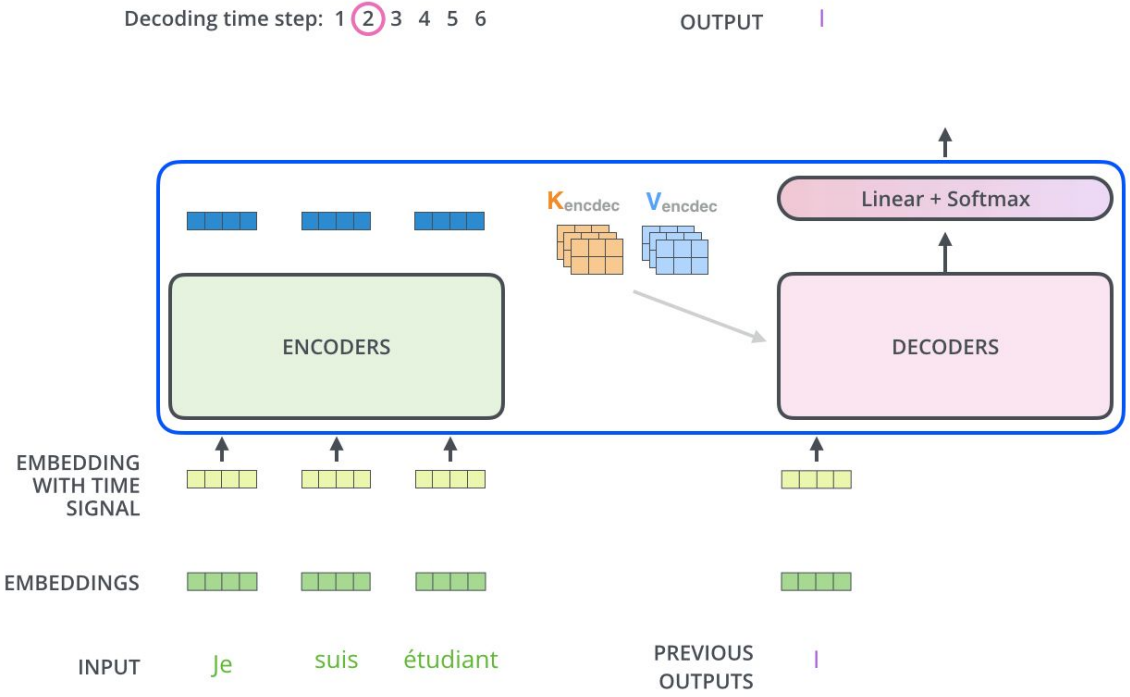
Supplementary - tokenizer

- Since the text-decoder generate the caption autoregresively, you should first use the tokenizer to tokenize the caption in data pre-processing.
- We provide a pretrained tokenizer based on Tokenizers.
  - *hw3_data/caption_tokenizer.json*
  - You can simply use it with:

```
from tokenizers import Tokenizer

tokenizer = Tokenizer.from_file("caption_tokenizer.json")

caption = "a baby is playing with a teddy bear."
tokenized_caption = tokenizer.encode(caption)

print(tokenized_caption.ids)
print(tokenized_caption.tokens)
```

```
[6]  ✓ 0.1s
...  [2, 32, 697, 131, 301, 121, 32, 736, 448, 13, 3]
     ['[BOS]', 'a', 'baby', 'is', 'playing', 'with', 'a', 'teddy', 'bear', '.', '[EOS]']
```

Supplementary - what is autoregressive?



Reference: [The Illustrated Transformer]

Supplementary - Decoding strategy

- You could choose the advanced decoding strategy to improve your model when you autoregressively generate captions. For example:
  - Greedy search
  - Beam search
  - Nucleus sampling
  - …etc.

  Please read the reference tutorial for details.

Supplementary - Toolkit summary

We provide some toolkits for you to more efficiently finish this homework. Therefore, we summarize them in the end as follows:

- Pretrained OpenAI-CLIP
- pytorch-transformer
- CIDEr (language evaluation)
- Pytorch Image Models (timm)
- The annotated transformer