Vision and Language

Sanghyuk Chu
Hyewon Yoo
Department of Electrical and Computer Engineering

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

CLIP Learning

CLIP: Learning transferable visual models from natural language supervision

- Can pre-training method be applied to computer vision?
  - Pre-training methods are widely used and demonstrated to be efficient in NLP. e.g. BERT, GPT
- CLIP: Contrastive Language-Image Pretraining
  - natural language supervision for image representation learning
  - image captioning: the simple pre-training task of predicting which caption goes with which image
  - dataset: 400 million image-text pairs collected from the internet

https://arxiv.org/pdf/2103.00020.pdf

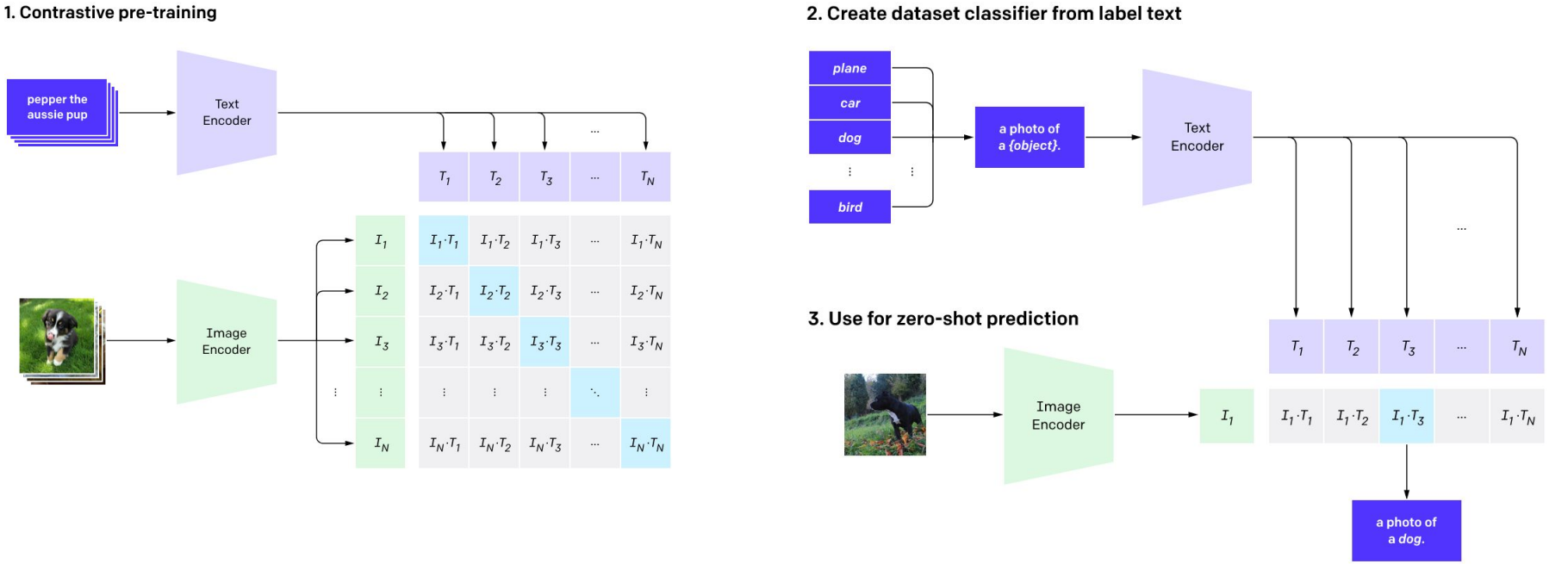
Method

- Natural language supervision: does not just learn a representation but connects the representation to language
- Creating a sufficiently large dataset
  - MS-COCO, Visual Genome, YFCC100M → quality and dataset size issues
  - new dataset WebImageText (WIT) → collected from the internet
- Selecting an efficient pre-training method
- Choosing and scaling a model
  - image encoder: ResNet-50 (baseline), ResNet-D, ViT. increase the width, depth, and resolution of the model.
  - text encoder: Transformer. increase the width
- Training

Method - selecting an efficient pre-training method

- Existing methods
  - jointly train an image CNN and text transformer from scratch → slow
  - try to predict the exact words of the text accompanying each image
- proxy task of predicting only which text as a whole is paired with which image and not the exact words of the text
- Objective is changed from predictive to contrastive
- Given a batch of N (image, text) pairs, precit which of the NxN possible pairs across a batch actually occurred
- Maximize the cosine similarity of the image and text embeddings of the N real pairs in the batch while minimizing the other (incorrect) pairs

Method


1. Contrastive pre-training
pepper the aussie pup
Text Encoder
Image Encoder
2. Create dataset classifier from label text
plane
car
dog
bird
a photo of a {object}.
Text Encoder
3. Use for zero-shot prediction
Image Encoder
a photo of a dog.

Method - pseudocode of CLIP

```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

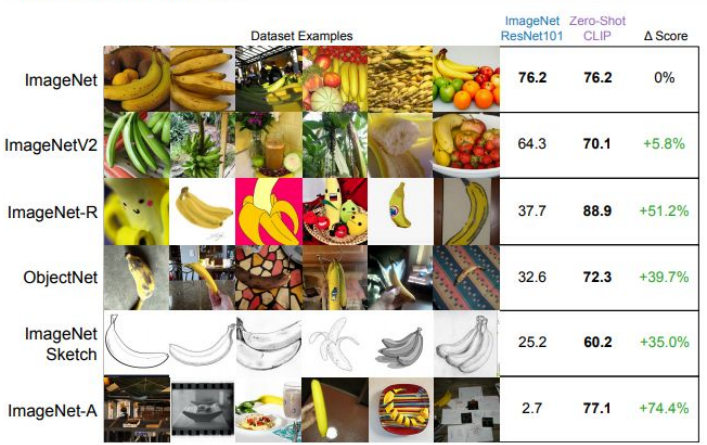
Experiment

- zero-shot transfer
  - generalization to unseen datasets
  - an evaluation of CLIP's robustness to distribution shift and domain generalization
- Prompt engineering and ensembling
  - pre-training dataset consists of full sentence rather than a single word
  - zero-shot performance can be improved by customizing the prompt text to eash task
  - ensembling over multiple zero-shot classifiers improves performance

	aYahoo	ImageNet	SUN
Visual N-Grams	72.4	11.5	23.0
CLIP	**98.4**	**76.2**	**58.5**

Experiment

- zero-shot transfer
  - tested on 27 datasets
  - wins the baseline (ResNet-50) on 16 of 27 datasets
  - weak on specialized, complex, or abstract tasks



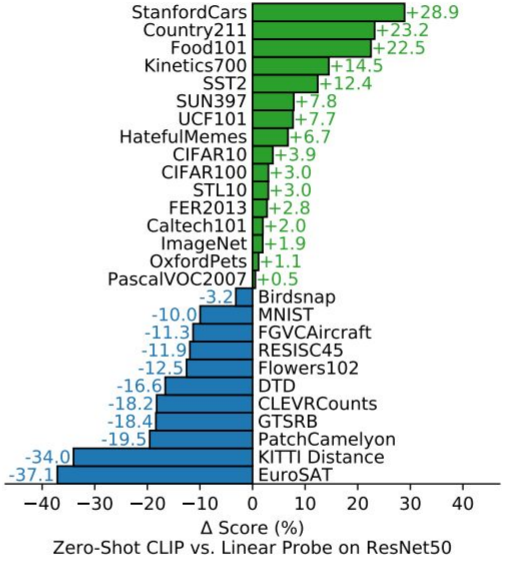


Figure 5. **Zero-shot CLIP is competitive with a fully supervised baseline.** Across a 27 dataset eval suite, a zero-shot CLIP classifier outperforms a fully supervised linear classifier fitted on ResNet-50 features on 16 datasets, including ImageNet.

FOOD101

guacamole (90.1%) Ranked 1 out of 101 labels

✓ a photo of **guacamole**, a type of food.

× a photo of **ceviche**, a type of food.

× a photo of **edamame**, a type of food.

× a photo of **tuna tartare**, a type of food.

× a photo of **hummus**, a type of food.

SUN397

television studio (90.2%) Ranked 1 out of 397

✓ a photo of a **television studio**.

× a photo of a **podium indoor**.

× a photo of a **conference room**.

× a photo of a **lecture room**.

× a photo of a **control room**.

YOUTUBE-BB

airplane, person (89.0%) Ranked 1 out of 23

✓ a photo of a **airplane**.

× a photo of a **bird**.

× a photo of a **bear**.

× a photo of a **giraffe**.

× a photo of a **car**.

EUROSAT

annual crop land (12.9%) Ranked 4 out of 10

× a centered satellite photo of **permanent crop land**.

× a centered satellite photo of **pasture land**.

× a centered satellite photo of **highway or road**.

✓ a centered satellite photo of **annual crop land**.

× a centered satellite photo of **brushland or shrubland**.

Advantages

- uses text-image pairs that are already publicly available on the internet
- flexible and general: CLIP can be evaluated on benchmarks without having to train on their data
- allows people to design their own classifiers and removes the need for task-specific training data
- robust to image distribution shift, have impressive zero-shot capabilities
- achieve state-of-the-art results on a wide variety of vision and language tasks just using fine-tuning

Limitations

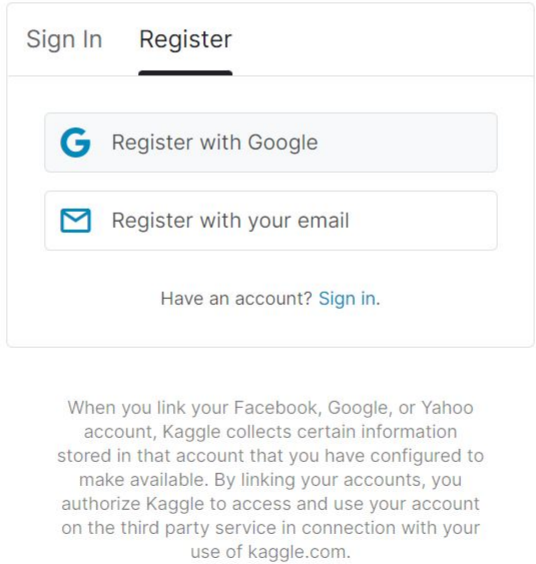
- struggles on abstract or systematic tasks such as counting the number of objects in an image and on more complex tasks
- struggles compared to task specific models on very fine-grained classification

Practice

Create Kaggle Account

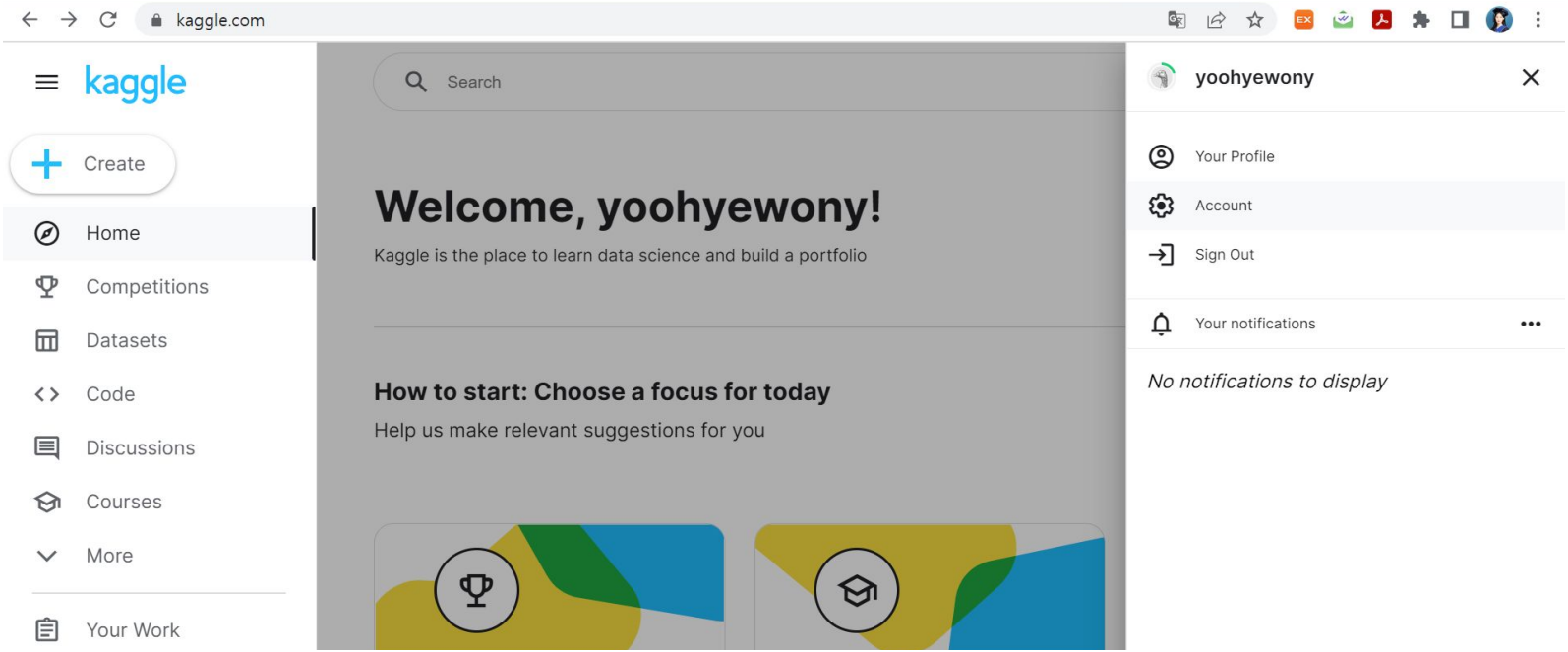
kaggle

1. Go to Kaggle website https://www.kaggle.com/
2. Click register and sign up with email



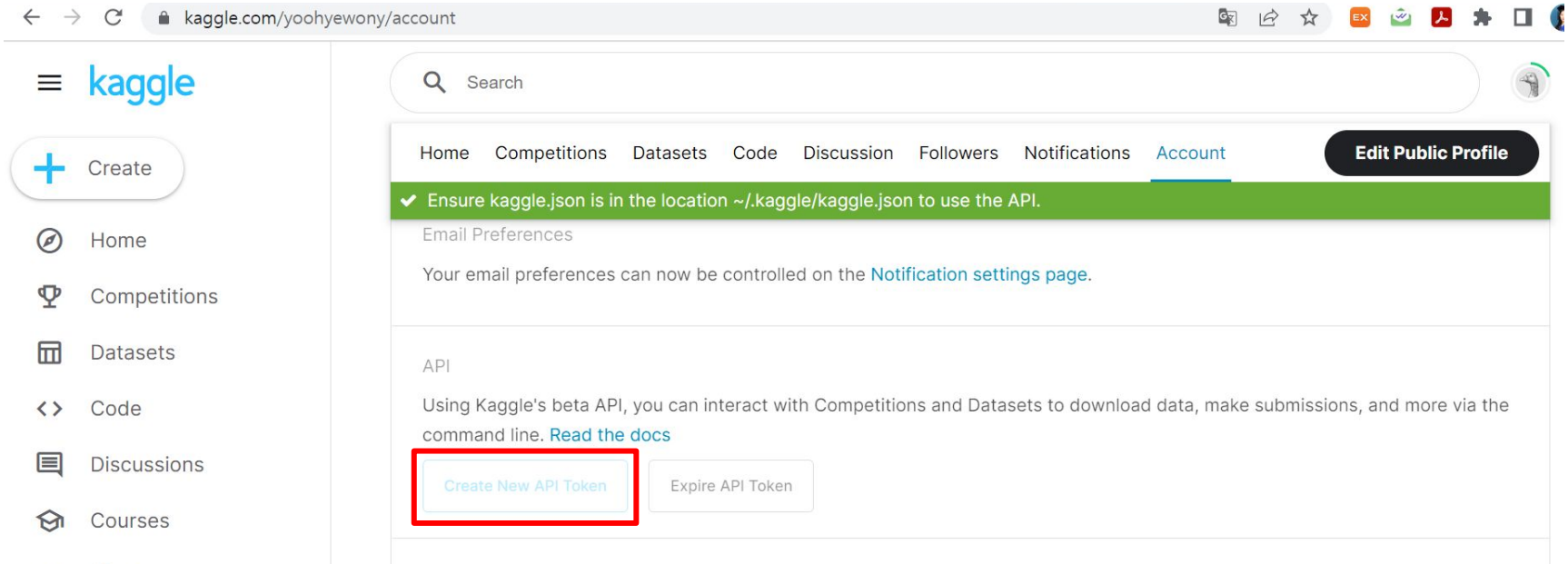
Create Kaggle Account

3. After finish signing up, go to account



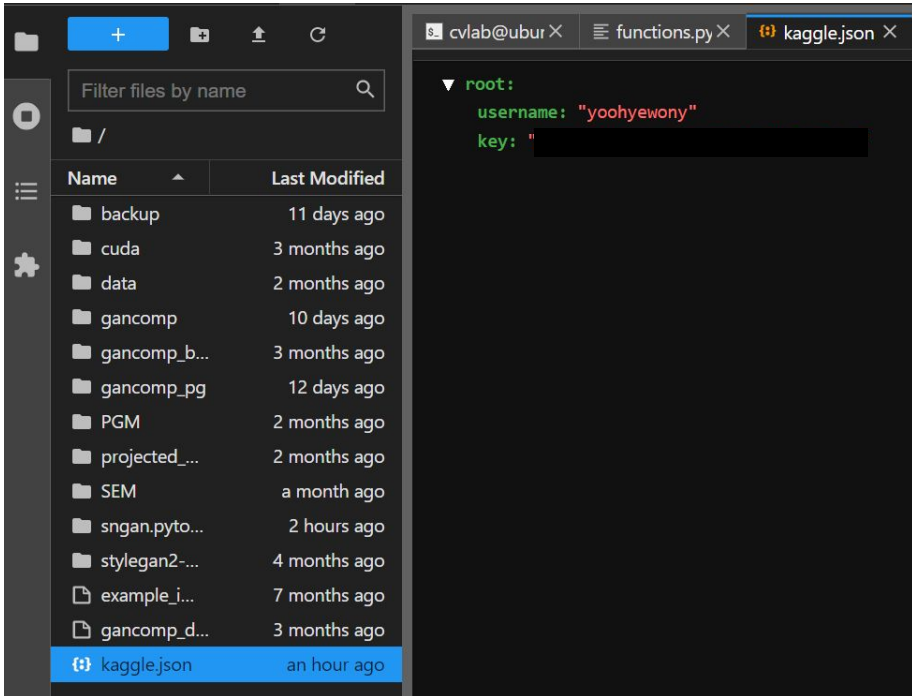
Create Kaggle Account

4. Click create new API token



Create Kaggle Account

5. Open json file to check account key



Type kaggle username and key

https://colab.research.google.com/drive/1CNdjDTTVXK9YzuaxoRMcbk2JNR9GBDM_?usp=sharing

```
[ ]  !pip install kaggle --upgrade
     os.environ['KAGGLE_USERNAME'] = " "
     os.environ['KAGGLE_KEY'] = " "

     ### For Flickr 8k
     !kaggle datasets download -d adityajn105/flickr8k
     !unzip flickr8k.zip
     dataset = "8k"


     ### For Flickr 30k
     # !kaggle datasets download -d hsankesara/flickr-image-dataset
     # !unzip flickr-image-dataset.zip
     # dataset = "30k"
```

```
스트리밍 출력 내용이 길어서 마지막 5000줄이 삭제되었습니다.
  inflating: Images/2844846111_8c1cbfc75d.jpg
  inflating: Images/2844963839_ff09cdb81f.jpg
  inflating: Images/2845246160_d0d1bbd6f0.jpg
  inflating: Images/2845691057_d4ab89d889.jpg
  inflating: Images/2845845721_d0bc113ff7.jpg
  inflating: Images/2846037553_1a1de50709.jpg
```

Make CLIP dataset

- read and load image and transform
- find corresponding text
- return (image, text) pair

```python
class CLIPDataset(torch.utils.data.Dataset):
    def __init__(self, image_filenames, captions, tokenizer, transforms):
        """
        image_filenames and cpations must have the same length; so, if there are
        multiple captions for each image, the image_filenames must have repetitive
        file names
        """

        self.image_filenames = image_filenames
        self.captions = list(captions)
        self.encoded_captions = tokenizer(
            list(captions), padding=True, truncation=True, max_length=CFG.max_length
        )
        self.transforms = transforms

    def __getitem__(self, idx):
        item = {
            key: torch.tensor(values[idx])
            for key, values in self.encoded_captions.items()
        }

        image = cv2.imread(f"{CFG.image_path}/{self.image_filenames[idx]}")
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = self.transforms(image=image)['image']
        item['image'] = torch.tensor(image).permute(2, 0, 1).float()
        item['caption'] = self.captions[idx]

        return item
```

Image & Text Encoder

- Image encoder
  - ResNet50 pre-trained on ImageNet
- Text encoder
  - tokenize text with DistilBERT from HugginFace
  - DistilBERT: compressed version of BERT which is faster and memory efficient

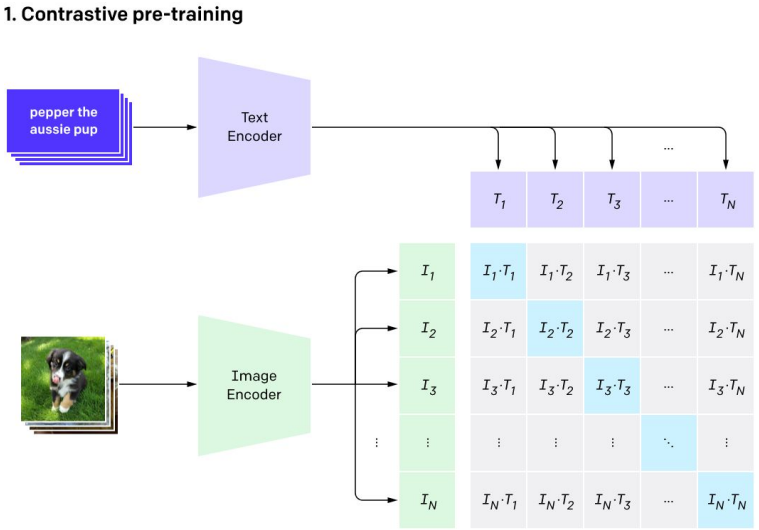
Projection Head

- to match the dimensions of image and text
  - img dim: 2048
  - text dim: 768
  - projection dim: 256
- embedding_dim → projection_dim
- forwarding: projection →gelu → fc → dropout → skip-connection → layer normalization

```
class ProjectionHead(nn.Module):
    def __init__(
        self,
        embedding_dim,
        projection_dim=CFG.projection_dim,
        dropout=CFG.dropout
    ):
        super().__init__()
        self.projection = ### Fill this line
        self.gelu = nn.GELU()
        self.fc = nn.Linear(projection_dim, projection_dim)
        self.dropout = nn.Dropout(dropout)
        self.layer_norm = nn.LayerNorm(projection_dim)

    def forward(self, x):
        '''
        Write your code


        '''
        return x
```

CLIP Model

- extract image and text features by encoders
- match image and text dimensions by projection
- calculate loss
  - extract joint multimodal embeddings
  - calculate image, text similarity using dot product
  - compute cross entropy loss for image and text, respectively
  - average the losses



Train

- construct CLIP model
- load data using dataloader
- train the model
- save the best model

```
best_loss = float('inf')
for epoch in range(CFG.epochs):
    print(f"Epoch: {epoch + 1}")
    model.train()
    train_loss = train_epoch(model, train_loader, optimizer, lr_scheduler, step)
    model.eval()
    with torch.no_grad():
        valid_loss = valid_epoch(model, valid_loader)

    ############ Write your code ####################
    # If current model is better than the best model,
    # then replace the best model as current model
    # file name should be "best.pt"


        #print("Saved Best Model!")
    ################################################

    lr_scheduler.step(valid_loss.avg)
```

Find matches

- normalize embeddings
  - hint: use F.normalize
- compute dot similarity

```python
def find_matches(model, image_embeddings, query, image_filenames, n=9):
    tokenizer = DistilBertTokenizer.from_pretrained(CFG.text_tokenizer)
    encoded_query = tokenizer([query])
    batch = {
        key: torch.tensor(values).to(CFG.device)
        for key, values in encoded_query.items()
    }
    with torch.no_grad():
        text_features = model.text_encoder(
            input_ids=batch["input_ids"], attention_mask=batch["attention_mask"]
        )
        text_embeddings = model.text_projection(text_features)

    ################ Write your code ######################
    image_embeddings_n =
    text_embeddings_n =
    dot_similarity =
    ########################################################

    values, indices = torch.topk(dot_similarity.squeeze(0), n * 5)
    matches = [image_filenames[idx] for idx in indices[::5]]
```

Result

```
find_matches(model,
             image_embeddings,
             query="dogs on the grass",
             image_filenames=valid_df['image'].values,
             n=9)
```

DALL-E: Zero-shot text to image generation

- 2-stage model
  - learning the visual codebook via discrete VAE (dVAE)
  - concatenate image and text tokens, and train an autoregressive transformer to model the joint distribution over the text and image tokens
- VQ-VAE + GPT3
- https://openai.com/blog/dall-e/

https://arxiv.org/pdf/2102.12092.pdf

DALL-E-2: Hierarchical text-conditional image generation with CLIP latents

- 2-stage model
  - a prior that generates a CLIP image embedding given a text caption
  - a decoder that generates an image conditioned on the image embedding
- CLIP + diffusion
- train a diffusion decoder to invert the CLIP image encoder
- combine the CLIP image embedding decoder with a prior model, which generates possible CLIP image embeddings from a given text caption
- https://openai.com/dall-e-2/

Method

- Decoder
  - invert CLIP image embeddings z_i to produce images x
  - use diffusion models to produce images conditioned on CLIP image embeddings
  - guidance improves the quality of images. but also enable guidance-free diffusion model by randomly setting the CLIP embeddings to zero
  - to improve the robustness of up-samplers, corrupt the conditioning images during training (Gaussian blur,

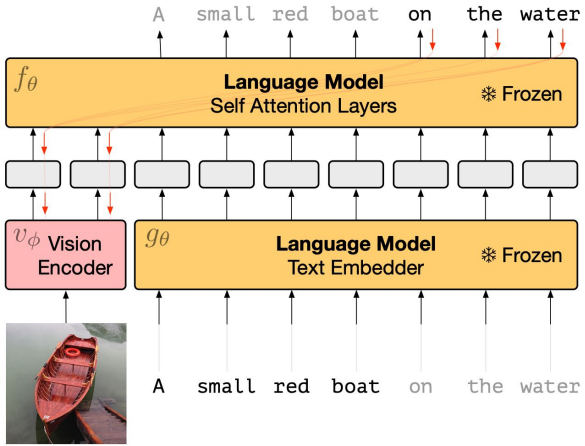
Method

- Prior
  - a prior model that produces z_i from captions y to enable image generation from text captions
  - Autoregressive (AR) prior: the CLIP image embedding is converted into a sequence of discrete codes and predicted autoregressively conditioned on the caption y
  - Diffusion prior: the continuous vector z_i is directly modelled using a Gaussian diffusion model conditioned on the caption y
  - condition the prior on the CLIP text embedding since it is a deterministic function of the caption
  - condition the AR prior on the text caption and the CLIP text embedding by encoding them as a prefix to the sequence
  - prepend the dot product of the text embedding and image embedding
  -

Transformer tutorial

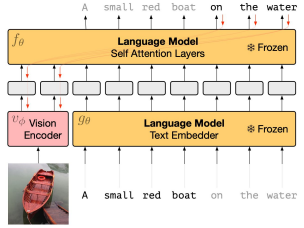
Frozen

- Way of utilizing a large-scale pre-trained LM in the multimodal setting.
  - large-scale, pretrained LM is kept frozen.
  - Only a vision encoder is fine-tuned on the image-caption dataset.
  - Utilize zero/few-shot performance of LM, and do similar things in the multi-modal setting.



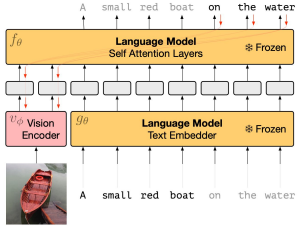
Multimodal Few-Shot Learning with Frozen Language Models, In NeurIPS, 2021.

Architecture (Pseudo code)



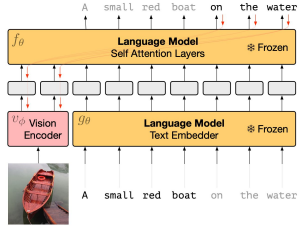

```python
from torch import nn
from transformers import GPT2LMHeadModel, ResNetModel

class Frozen(nn.Module):
    def __init__(self, config, tokenizer):
        super().__init__()
        self.config = config
        self.tokenizer = tokenizer
        self.num_visual_prefix = 2
        self.text_encoder = GPT2LMHeadModel.from_pretrained('gpt2')
        self.proj_dim = self.text_encoder.config.n_embd
        self.text_encoder.resize_token_embeddings(len(self.tokenizer))

        self.vision_encoder = ResNetModel.from_pretrained('microsoft/resnet-50')
        self.vision_proj_m = nn.Linear(self.vision_encoder.config.hidden_sizes[-1], self.num_visual_prefix * self.proj_dim)

        self.freeze_LM()

    def freeze_LM(self):
        # Freeze all params in Pretrained LM. LMs usually adopts LayerNorm, which does not changes during training/validation stages.
        for param in self.text_encoder.parameters():
            param.requires_grad = False
```

Architecture (Pseudo code)




```python
def get_visual_prefix(self, image):
    outputs = self.vision_encoder(image).pooler_output

    b = outputs.shape[0]
    outputs = outputs.reshape(b, -1)
    visual_prefix = self.vision_proj_m(outputs)
    visual_prefix = visual_prefix.reshape(b, self.num_visual_prefix, self.proj_dim)
    return visual_prefix

def forward(self, gen_flag=False, *input, **kwargs):
    if 'pixel_values' in kwargs:
        pixel_values = kwargs.pop('pixel_values')
        kwargs['image_features'] = self.get_visual_prefix(pixel_values)

        kwargs['image_features_mask'] = kwargs['input_ids'] == self.tokenizer.image_token_id
    if gen_flag:
        return self.text_encoder.generate(*input, **kwargs)
    else:
        return self.text_encoder(*input, **kwargs)

def generate(self, *input, **kwargs):
    return self.forward(gen_flag=True, *input, **kwargs)
```

Training




```python
for i, (image, text) in enumerate(metric_logger.log_every(data_loaders['train'], config['print_freq'], header)):
    optimizer.zero_grad()

    image = image['pixel_values'][0].to(device, non_blocking=True)

    # text processing: captioning
    text_ids = tokenizer(text, padding='longest', truncation=True, max_length=config['max_length'], return_tensors="pt")['input_ids']
    image_ids = torch.tensor([tokenizer.image_token_id for _ in range(config['num_image_tokens'])]).unsqueeze(0).repeat(image.shape[0], 1)
    text_input = torch.cat((image_ids, text_ids), dim=1)
    text_input = text_input.to(device)

    model_kwargs = {'input_ids': text_input, 'pixel_values': image}

    output = model(**model_kwargs)

    loss = cal_celoss(output.logits, text_input, num_logits_shift=config['vision_encoder']['num_visual_prefix'])

    loss.backward()
    optimizer.step()
```

Perplexity

- What is perplexity?
- Does early stopping with perplexity value is a good way to judge the overfitting in Frozen?

$$P(W) = P(w_1, w_2, ..., W_n) = P(w_1)P(w_2|w_1)P(w_3|w_2, w_1)...P(w_N|w_{N-1}, ..., w_1)$$

$$PP(W) = \sqrt[N]{\frac{1}{P(w_1, w_2, \ldots, w_N)}}$$

Frozen demo

GT caption: actor attends the opening ceremony of exhibition
Generated caption: actor attends the premiere at the theater

GT caption: political map with the several counties wherea city is highlighted
Generated caption: political map with the several states where the state is highlighted

Prompting

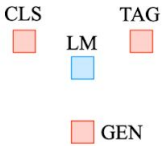
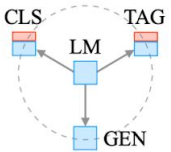
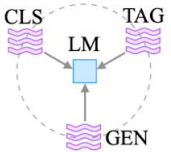
- What is prompting?
  - A new way of using language models for downstream tasks in NLP.
  - Technically, prompting is a (packaging) process that modifies an input with “prompt”.
    - A piece of text is inserted in the input examples, so that the original task can be formulated as a (masked) language modeling problem.

- Example - sentiment analysis

	Name	Notation	Example	Description
Sample in dataset →	*Input*	$\boldsymbol{x}$	I love this movie.	One or multiple texts
	Output	$\boldsymbol{y}$	++ (very positive)	Output label or text
Template →	*Prompting Function*	$f_{\text{prompt}}(\boldsymbol{x})$	`[X]` Overall, it was a `[Z]` movie.	A function that converts the input into a specific form by inserting the input $\boldsymbol{x}$ and adding a slot `[Z]` where answer $\boldsymbol{z}$ may be filled later.
Input to the model →	*Prompt*	$\boldsymbol{x}'$	I love this movie. Overall, it was a `[Z]` movie.	A text where `[X]` is instantiated by input $\boldsymbol{x}$ but answer slot `[Z]` is not.

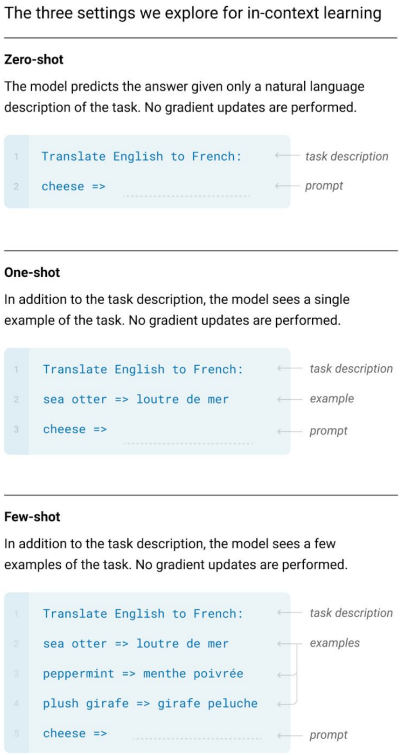
Prompting

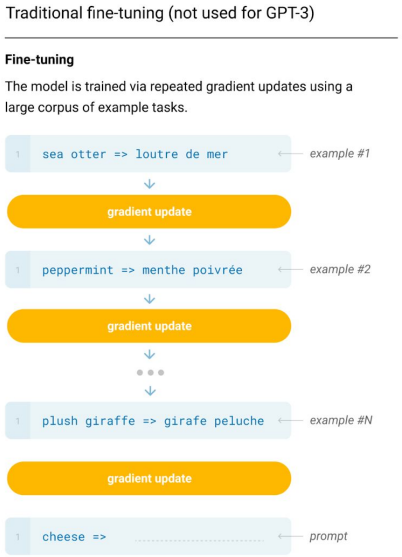
- A new paradigm
  - Fine-tuning: “LM→Task”, adapting LMs (objectives) to downstream tasks.
  - Prompting: “Task→LM”, adapting downstream tasks (formulations) to LMs.

b. Fully Supervised Learning (Neural Network)	Architecture (e.g. convolutional, recurrent, self-attentional)	
c. Pre-train, Fine-tune	Objective (e.g. masked language modeling, next sentence prediction)	
d. Pre-train, Prompt, Predict	Prompt (e.g. cloze, prefix)	

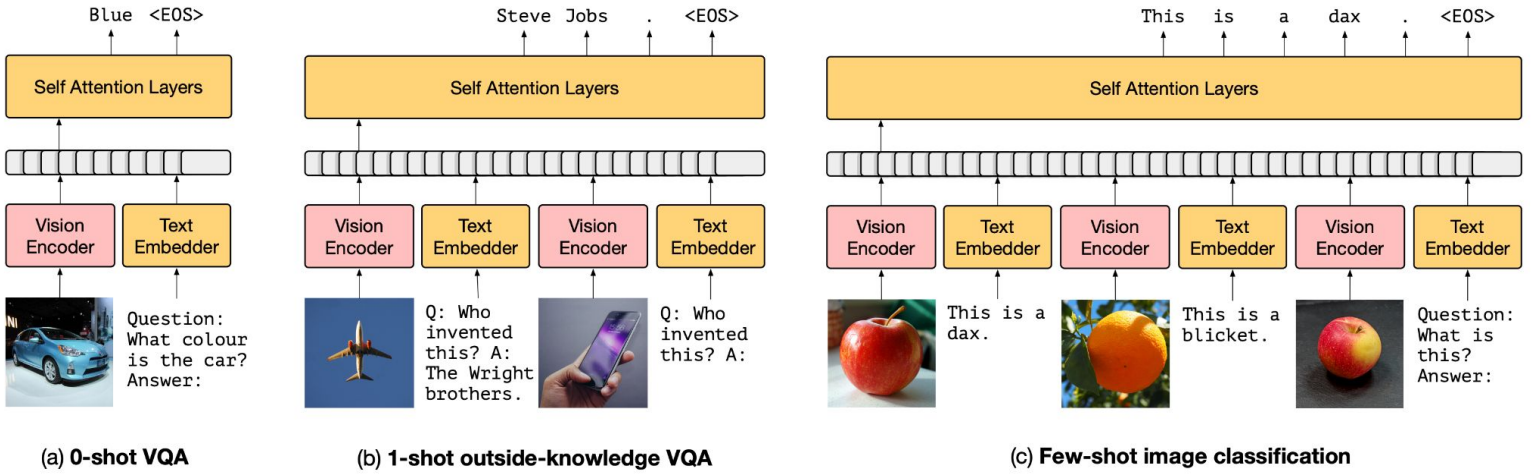
Prompting

- Zero/few-shot capabilities
  - First introduced in GPT-2.
  - Boosted with hyperscale GPT-3 175B model.
  - Do **not** require expensive fine-tuning stage.
    - Building dataset
    - Labeling
    - Updating parameters
    - Storing checkpoints
  - Even able to match fine-tuning performances.
  - examples(1), examples(2)





Few/zero-shot with prompting



(a) **0-shot VQA** (b) **1-shot outside-knowledge VQA** (c) **Few-shot image classification**