ELL-786 Report
Assignment-3

Rohan Mahala 2019MT60760

Ayush Verma 2019MT60749

Part-1

Zero-shot learning is a form of machine learning in which a model is trained to recognise objects or classes it has never encountered. In zero-shot learning, no labelled examples of the new classes are provided to the model during training; instead, the model relies on a set of attributes or semantic descriptions to define each class. The model then employs these descriptions to predict the designation of a new image. For example, the model can correctly put a picture of a giraffe in the "animals" category even though it has never seen a giraffe before.

Few-shot learning is the same idea, but it adds the ability to learn from just a few examples. In the case of CLIP, few-shot learning means that a small number of labelled examples from a new category or task are used to fine-tune the model that has already been trained. This makes it easy for the model to adapt quickly to new classification tasks with little training data. For example, the model can learn to sort pictures of different kinds of dogs after seeing just a few of each kind.

In the context of the paper, **prompt engineering** is a method used in the field of artificial intelligence, especially natural language processing (NLP), to make language models better at doing certain tasks. The idea is to give the model a personalised prompt text that acts as a guide to help it understand the task and give more accurate results. It involves customizing the prompt text for each task to provide more specific information about the image being classified.

The examples listed below illustrate prompt engineering:

- The prompt "A positive/negative movie review" can be used to classify the tone of a movie review as either positive or negative. This prompt encourages the model to focus on the tone of the review rather than specifics such as the actors or plot.
- Object detection can identify specific objects in an image by using prompts such as "A picture of a type of object" or "A photograph of a colour type of object." For example, "A photograph of a scarlet sports car" or "A photograph of a blue SUV" can be used to detect different types of automobiles in an image.
- We can use prompts such as "A photograph of a style> object>" or "A photograph of a style> scene" to transfer the style of one image to another.

Now, implementing the code to find the accuracy with basic prompt (’a photo of a {class}’) and modified prompt (”a photo of a {class}, a type of pet.’) on Oxford-IIIT Pets dataset.

CODE :

Step-1 :- Importing various important python libraries.

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input di

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when yo
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current sessio
```

Step-2 :- Installing the CLIP model available on openai github page.

```
! pip install ftfy regex tqdm
! pip install git+https://github.com/openai/CLIP.git
```

Step-3 :- Importing various other libraries such as pytorch(torchvision) and the downloaded clip model. Also, defining the basic and modified prompts and creating the templates.

```
import torch
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import clip
import os
import shutil
import random
from tqdm.notebook import tqdm
from pkg_resources import packaging

# Define basic and modified prompts
basic_prompt = "a photo of a {}"
modified_prompt = "a photo of a {}, a type of pet"

templates1 = [basic_prompt]
templates2 = [modified_prompt]
```

Step-4 :- Introducing various functions.

First, split_images function makes various folders for the different classes of the complete Oxford-IIIT Pets datasets and puts all the images of the class in their respective folders. We have worked on the Oxford datasets in two ways, one, available on pytorch and other was downloaded and then splitted using this function.

```
def split_images(data_dir):
    images = "/kaggle/working/images"
    if(not os.path.exists(images)):
        os.makedirs(images)
    for filename in os.listdir(data_dir):
        class_name = "_".join(list(filename.split("_"))[:-1])
        class_dir = os.path.join(images, class_name)
        src_dir = os.path.join(data_dir,filename)
        if(not os.path.exists(class_dir)):
            os.makedirs(class_dir)
        shutil.copy(src_dir,class_dir)

    shutil.make_archive(os.path.join("/kaggle/working/","oxford_dataset_iiit-pet_splitted"), 'zip', images)
```

Second, function to do the zeroshot learning.

```
# Create zeroshot weights
def zeroshot_classifier(classnames, templates):
    with torch.no_grad():
        zeroshot_weights = []
        for classname in tqdm(classnames):
            texts = [template.format(classname) for template in templates] #format with class
            texts = clip.tokenize(texts).cuda() #tokenize
            class_embeddings = model.encode_text(texts) #embed with text encoder
            class_embeddings /= class_embeddings.norm(dim=-1, keepdim=True)
            class_embedding = class_embeddings.mean(dim=0)
            class_embedding /= class_embedding.norm()
            zeroshot_weights.append(class_embedding)
        zeroshot_weights = torch.stack(zeroshot_weights, dim=1).cuda()
    return zeroshot_weights
```

Third, function to calculate accuracy.

```
def accuracy(output, target, topk=(1,)):
    pred = output.topk(max(topk), 1, True, True)[1].t()
    correct = pred.eq(target.view(1, -1).expand_as(pred))
    return [float(correct[:k].reshape(-1).float().sum(0, keepdim=True).cpu().numpy()) for k in topk]
```

Step-5 :- Loading the CLIP model.

```
# Load pre-trained CLIP model
clip.available_models()
device = "cuda" if torch.cuda.is_available() else "cpu"
model, preprocess = clip.load("ViT-B/32")
input_resolution = model.visual.input_resolution
context_length = model.context_length
vocab_size = model.vocab_size

print("Model parameters:", f"{np.sum([int(np.prod(p.shape)) for p in model.paramete
print("Input resolution:", input_resolution)
print("Context length:", context_length)
print("Vocab size:", vocab_size)
```

```
100%|████████████████████████████████| 338M/338M [00:02<00:00, 127MiB/s]
```

+ Code + Markdown

Step-6 :- Finally, integrating the code and calculating the accuracy of basic and modified prompts on the pytorch oxford-IIIT Pets dataset.

```
#data_dir = "/kaggle/input/oxford-dataset-iiit-pet-splitted"
data_dir = "/kaggle/working/"
#data_set = datasets.ImageFolder(data_dir, transform=preprocess)
data_set = datasets.OxfordIIITPet(data_dir, transform=preprocess,download = True)
loader = torch.utils.data.DataLoader(data_set, batch_size=32, num_workers=2)
classes = data_set.classes
zeroshot_weights = zeroshot_classifier(classes, templates1)
modified_zeroshot_weights = zeroshot_classifier(classes, templates2)


with torch.no_grad():
    top1, top5, n = 0., 0., 0.
    modified_top1,modified_top5 = 0. , 0.
    for i, (images, target) in enumerate(tqdm(loader)):
        images = images.cuda()
        target = target.cuda()

        # predict
        image_features = model.encode_image(images)
        image_features /= image_features.norm(dim=-1, keepdim=True)
        logits = 100. * image_features @ zeroshot_weights
        modified_logits = 100. * image_features @ modified_zeroshot_weights
```

```
        # measure accuracy
        acc1, acc5 = accuracy(logits, target, topk=(1, 5))
        top1 += acc1
        top5 += acc5

        modified_acc1, modified_acc5 = accuracy(modified_logits, target, topk=(1, 5))
        modified_top1 += modified_acc1
        modified_top5 += modified_acc5

        n += images.size(0)

top1 = (top1 / n) * 100
top5 = (top5 / n) * 100

modified_top1 = (modified_top1 / n) * 100
modified_top5 = (modified_top5 / n) * 100

print(f"Top-1 basic-accuracy: {top1:.2f}")
print(f"Top-5 basic-accuracy: {top5:.2f}")

print(f"Top-1 modified-accuracy: {modified_top1:.2f}")
print(f"Top-5 modified-accuracy: {modified_top5:.2f}")
```

We obtain the following accuracy,

```
Downloading https://thor.robots.ox.ac.uk/datasets/pets/images.tar.gz to /kaggle/working/ox
ford-iiit-pet/images.tar.gz
100% 791918971/791918971 [00:24<00:00, 32697843.74it/s]
Extracting /kaggle/working/oxford-iiit-pet/images.tar.gz to /kaggle/working/oxford-iiit-pe
t
Downloading https://thor.robots.ox.ac.uk/datasets/pets/annotations.tar.gz to /kaggle/worki
ng/oxford-iiit-pet/annotations.tar.gz
100% 19173078/19173078 [00:01<00:00, 21440807.38it/s]
Extracting /kaggle/working/oxford-iiit-pet/annotations.tar.gz to /kaggle/working/oxford-ii
it-pet
100% 37/37 [00:00<00:00, 39.27it/s]
100% 37/37 [00:00<00:00, 78.68it/s]
100% 115/115 [00:28<00:00, 4.70it/s]
Top-1 basic-accuracy: 80.41
Top-5 basic-accuracy: 96.60
Top-1 modified-accuracy: 85.49
Top-5 modified-accuracy: 99.16
```

Accuracy obtained on the complete downloaded and splitted dataset.

```
100% 37/37 [00:02<00:00, 29.30it/s]
100% 37/37 [00:00<00:00, 59.96it/s]
100% 231/231 [01:16<00:00, 3.28it/s]
Top-1 basic-accuracy: 78.04
Top-5 basic-accuracy: 95.01
Top-1 modified-accuracy: 82.11
Top-5 modified-accuracy: 97.25
```

Part-2

Polysemy is a phenomenon in natural language in which a single term can have multiple meanings or senses depending on its usage context. For instance, the term "boxer" may refer to either a dog breed or the occupation of a fighter. In the presence of such polysemous labels, this ambiguity can make it challenging for a machine learning model to accurately classify images.

To overcome this difficulty, we employ a technique known as prompt engineering, which entails modifying the prompt text to provide additional context that clarifies the meaning of the class designation. As an alternative to the standard prompt "a photo of a {class}," a modified prompt such as "a photo of a {class}, a type of pet" can assist the model in comprehending the intended meaning of the label.

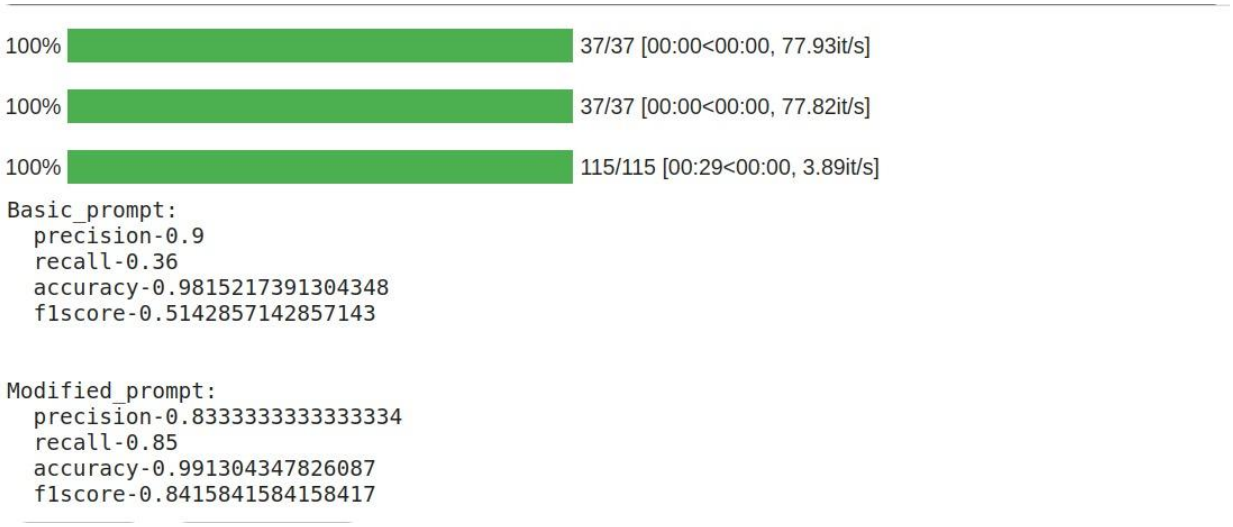
CODE :

To compare performance with and without modification in prompt.

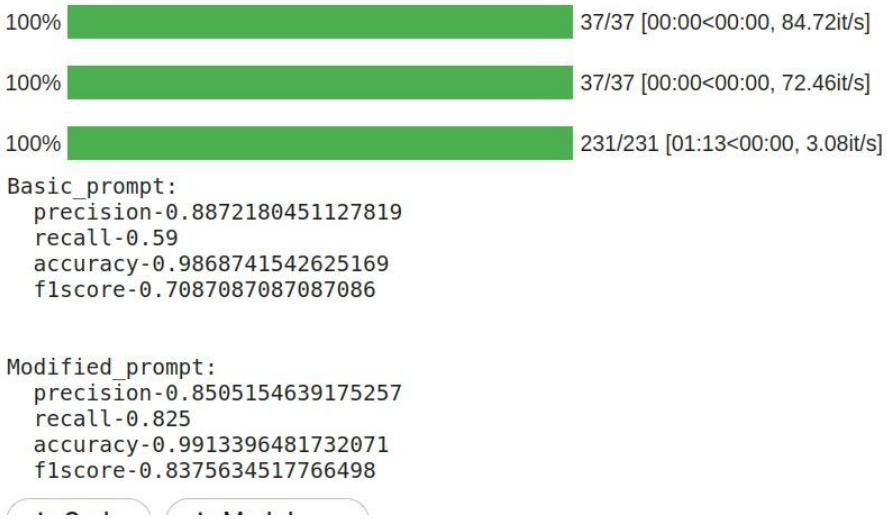
```
#data_dir = "/kaggle/input/oxford-dataset-iiit-pet-splitted"
data_dir = "/kaggle/working/"
#data_set = datasets.ImageFolder(data_dir, transform=preprocess)
data_set = datasets.OxfordIIITPet(data_dir, transform=preprocess,download = True)
loader = torch.utils.data.DataLoader(data_set, batch_size=32, num_workers=2)
classes = data_set.classes
boxer_label = classes.index('boxer')
zeroshot_weights = zeroshot_classifier(classes, templates1)
modified_zeroshot_weights = zeroshot_classifier(classes, templates2)


with torch.no_grad():
    tp,fp,tn,fn = 0,0,0,0
    mtp,mfp,mtn,mfn = 0,0,0,0
    for i, (images, target) in enumerate(tqdm(loader)):
        images = images.cuda()
        target = target.cuda()

        # predict
        image_features = model.encode_image(images)
        image_features /= image_features.norm(dim=-1, keepdim=True)
        logits = 100. * image_features @ zeroshot_weights
        modified_logits = 100. * image_features @ modified_zeroshot_weights
```

```
        # measure accuracy
        tp1,fp1,tn1,fn1 = accuracy(logits, target,boxer_label)
        tp += tp1
        fp += fp1
        tn += tn1
        fn += fn1

        mtp1,mfp1,mtn1,mfn1 = accuracy(modified_logits, target,boxer_label)
        mtp += mtp1
        mfp += mfp1
        mtn += mtn1
        mfn += mfn1


bp = (tp/(tp + fp))
br = (tp/(tp + fn))
bacc = (tp + tn)/(tp + tn + fp + fn)
bf1 = (2*bp*br)/(bp + br)
print(f"Basic_prompt: \n  precision-{bp} \n  recall-{br} \n  accuracy-{bacc} \n  f1score-{bf1} \n \n")
mp = (mtp/(mtp + mfp))
mr = (mtp/(mtp + mfn))
macc = (mtp + mtn)/(mtp + mtn + mfp + mfn)
mf1 = (2*mp*mr)/(mp + mr)
print(f"Modified_prompt: \n  precision-{mp} \n  recall-{mr} \n  accuracy-{macc}  \n  f1score-{mf1}")
```

On the pytorch dataset :



On the downloaded and splitted dataset :



Part-3

Generalizability is a fundamental concept in machine learning, referring to the ability of a model to perform well on new, unseen data that is different from the data it was trained on. In other words, a model is considered generalizable if it is able to effectively apply what it has learned from the training data to new, previously unseen data points.

A model that can generalise effectively is capable of making accurate predictions on new data derived from the same distribution as the training data. In contrast, a model that is overfitted to the training data may perform well on the training set but unfavourably on new data, indicating that its generalizability is low.

In order to assess the generalizability of a machine learning model, it is common practice to divide the available data into at least three sets: a training set, a validation set, and a test set. The training set is utilised to train the model, whereas the validation set is employed to fine-tune the model's hyperparameters and prevent overfitting. The test set is used to evaluate the model's performance on new, previously unseen data that is analogous to the training data.

Robustness to distribution shift is the capacity of a machine learning model to maintain excellent performance when the input data distribution changes. In other words, a model is robust to distribution shift if it can generalise effectively to unfamiliar distributions that it did not encounter during training.

For instance, a model trained on natural images may perform inadequately when presented with highly distorted images or unusual lighting conditions. On the other hand, a robust model should be able to deal with such distribution shifts and maintain excellent performance even when the input data significantly deviates from the training data.

Distribution shift, also known as a dataset shift, occurs when the statistical properties of the training data and test data differ. It refers to the disparity in data distribution between the data that a machine learning model is trained on and the data that it will be tested on or deployed in the real world. Distribution shift can happen for various reasons, such as changes in user behaviour, changes in the environment or changes in data collection procedures.

DATASETS :

IMAGENET: ImageNet is a large-scale image dataset designed to serve as a benchmark for evaluating image classification models. Here are some of the essential characteristics and applications of the ImageNet dataset:

Properties:

- Size: The ImageNet dataset comprises more than 1.2 million images, making it one of the largest image datasets available to the public.
- Categories: The dataset is organised into over one thousand object categories, ranging from ubiquitous objects like cars and dogs to more specific categories like hammerhead sharks and football balls.
- Annotations: Each image in the dataset is annotated with a bounding box that specifies the location of the object.

Features:

- Diversity: The ImageNet dataset contains a wide variety of images from different sources and with varying backgrounds and illumination conditions, making it a diverse and representative dataset for testing image classification models.
- Complexity: Many of the categories in the dataset are difficult to classify because they contain visually similar objects pertaining to different categories. This dataset's complexity makes it useful for evaluating the precision and generalizability of image classification models.

Uses:

- Benchmarking: ImageNet is commonly used to evaluate the efficacy of image classification models as a benchmark. On the ImageNet dataset, numerous advanced computer vision models have been trained and evaluated.
- Transfer learning: The ImageNet dataset is also used for transfer learning, where pre-trained models on ImageNet are fine-tuned on smaller, more specific datasets for tasks such as object detection and segmentation.
- Research: The ImageNet dataset has been utilised in a wide variety of research studies, such as those examining visual perception, object recognition, and deep learning.

IMAGENET-A: The ImageNet-A dataset is a modified variant of the ImageNet dataset that was developed to provide a more difficult metric for evaluating the robustness of image classification models. Here are a few of the most important attributes, features, and applications of the ImageNet-A dataset:

Properties:

- Size: ImageNet-A comprises the same number of images and categories as the original ImageNet dataset, with over 1.2 million images organised into over 1,000 categories.
- Image modifications: The images in the ImageNet-A dataset have been altered with a variety of image corruptions, including blurring, noise, and compression artifacts.

Features:

- Increased Difficulty: The ImageNet-A dataset is intended to be more difficult than the original ImageNet dataset because the modified images are more challenging for image classification models to classify accurately.
- Diversity: The ImageNet-A dataset contains a wide variety of corrupted images, providing a comprehensive and representative set of challenges for evaluating the robustness of image classification models.

Uses:

- Benchmarking: ImageNet-A is utilised as a benchmark to evaluate the robustness of image classification models against image corruption. It is believed that models that perform well on the ImageNet-A dataset are more resilient to real-world image corruption.
- Research: ImageNet-A based research dataset has been utilised in a variety of research studies, such as those aimed at enhancing the robustness of image classification models, developing novel techniques for image correction and restoration, and examining the visual perception of image corruptions.

IMAGENET-SKETCH: The ImageNet-Sketch dataset is a subset of ImageNet that consists of illustrations rather than photographs. Here are a few of the most important attributes, features, and applications of the ImageNet-Sketch dataset:

Properties:

- Size: The ImageNet-Sketch dataset comprises more than 50,000 sketches organised into more than 3,000 categories.
- Categories: The ImageNet-Sketch dataset contains the same categories as the original ImageNet dataset, permitting direct comparisons between models trained on the two datasets.

Features:

- Sketch format: The designs in the ImageNet-Sketch dataset are hand-drawn using a digital drawing tablet by human annotators. The resultant designs are black-and-white images with varying degrees of realism and level of detail.
- Visual style: Sketches in the ImageNet-Sketch dataset have a distinct visual aesthetic that is distinct from photographs. This aesthetic poses unique challenges for image classification models, such as the recognition of highly stylized and abstracted objects.

Uses:

- Benchmarking: ImageNet-Sketch is used as a benchmark for assessing the performance of image classification models for sketch recognition tasks.
- Research: The ImageNet-Sketch dataset is utilised in research studies pertaining to sketch recognition, deep learning, and the relationship between sketches and photographs.

IMAGENET-R: The ImageNet-R dataset is a modified variant of the ImageNet dataset that was developed to provide a more difficult metric for evaluating the robustness of image classification models. Here are some of the most important characteristics and applications of the ImageNet-R dataset:

Properties:

- Size: ImageNet-R comprises the same number of images and categories as the original ImageNet dataset, with over 1.2 million images organised into over 1,000 categories.
- Image modifications: The images in the ImageNet-R dataset have been modified with various types of image corruptions, including blurring, noise, and compression artefacts, as well as more subtle modifications, such as changes in colour and contrast.

Features:

- Increased difficulty: The ImageNet-R dataset is intended to be more difficult than the original ImageNet dataset because the modified images are more challenging for image classification models to classify accurately.
- Diversity: The ImageNet-R dataset contains a wide variety of image corruptions and modifications, providing a comprehensive and representative set of tests for evaluating the robustness of image classification models.

Uses:

- Benchmarking: ImageNet-R is used to evaluate the robustness of image classification models to image corruptions and modifications. Models that perform well on

the ImageNet-R dataset are deemed more robust to variations in real-world images.
- Research: The ImageNet-R dataset has been utilised in a broad range of research studies, including those aimed at enhancing the robustness of image classification models, developing new image correction and restoration techniques, and investigating the visual perception of image modifications.

MS-COCO: The Microsoft Common Objects in Context (MS-COCO) dataset is a large-scale image recognition, segmentation, and captioning dataset designed to help advance computer vision research. Here are some of the key properties, features, and uses of the MS-COCO dataset:

Properties:

- Size: The MS-COCO dataset includes over 330,000 images and over 2,500,000 object instances.
- Object categories: The dataset includes eighty object categories, making it an invaluable resource for object recognition and segmentation tasks.

Features:

- Image variety: The images in the MS-COCO dataset are derived from a vast array of sources and depict a vast array of scenes and objects, including people, animals, vehicles, and commonplace objects.

- Object instance segmentation: In addition to object recognition, the dataset also contains pixel-level segmentation annotations for each object instance within each image, making it a valuable resource for instance segmentation tasks.
- Captioning: The dataset also contains captions written by human annotators for each image, making it a valuable resource for image captioning tasks.

Uses:

Benchmarking: The MS-COCO dataset is used as a benchmark to evaluate the efficacy of computer vision models on tasks including object recognition, segmentation, and captioning.

Research: The MS-COCO dataset has been utilised in a variety of research studies, including object recognition, segmentation, and captioning research, as well as research on generative models and image synthesis.

Pre-training: The MS-COCO dataset can be used as a pre-training dataset for training deep neural networks on image recognition tasks, potentially enhancing the performance of these models on subsequent tasks.

Applications: The MS-COCO dataset can be utilised in a variety of practical applications, including image search, recommendation systems, and autonomous vehicles.

CLIP: The CLIP model was trained using the Conceptual Captions large-scale multi-modal dataset. Here are some of the most important characteristics and applications of the Conceptual Captions dataset:

Properties:

- Size: It is one of the largest multimodal datasets available for training deep learning models.
- Caption quality: The captions in the dataset are written by humans and are typically of high quality, providing valuable context for each image.

Features:

- Multi-modal: The Conceptual Captions dataset is a multimodal dataset, consisting of both images and associated captions.
- Diversity: The images in the dataset are derived from a broad array of sources and depict a wide variety of scenes, objects, and concepts.

Uses:

- Pre-training: The Conceptual Captions dataset was utilised to pre-train the CLIP model, which has attained state-of-the-art performance on a wide variety of image and language tasks.
- Benchmarking: The Conceptual Captions dataset can be used to evaluate the performance of other multi-modal models on tasks such as image captioning, image retrieval, and visual question responding.

- Research: The Conceptual Captions dataset has been utilised in a variety of research studies, such as those examining multi-modal representation learning, generative models, and transfer learning.
- Applications: The Conceptual Captions dataset can be utilised in numerous practical applications, including image search, recommendation systems, and content moderation.

PART-4

Step-1 :- Importing various important python libraries.

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input di

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when yo
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current sessio
```

```
import torch
import time
import torchvision
import torch.optim as optim
import torch.nn as nn
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torchvision.models as models
import clip
import os
import shutil
import random
from tqdm.notebook import tqdm
from pkg_resources import packaging
```

Step-2:- Intitializing various parameters and loading the cats and dogs images of the imagenet-sketch dataset.

```
num_epochs = 10
batch_size = 64
learning_rate = 0.001

normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
data_transforms = {
    'train':
    transforms.Compose([
        transforms.Resize((224,224)),
        transforms.RandomAffine(0, shear=10, scale=(0.8,1.2)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        normalize
    ]),
    'validation':
    transforms.Compose([
        transforms.Resize((224,224)),
        transforms.ToTensor(),
        normalize
    ]),
}
```

```
data_set = {
    'train':  datasets.ImageFolder("/kaggle/input/kaggle-cats-and-dogs1/kagglecatsanddogs_5340/PetImages", d
    'validation': datasets.ImageFolder("/kaggle/input/100-cats-and-dogs/Cats_and_Dogs", data_transforms['val
}
```

Step-3:- Dividing the dataset into train and validation sets and then loading the resnet50 model.

```
dataloaders = {
    'train':
    torch.utils.data.DataLoader(data_set['train'],
                                batch_size=128,
                                shuffle=True,
                                num_workers=0),
    'validation':
    torch.utils.data.DataLoader(data_set['validation'],
                                batch_size=8,
                                shuffle=False,
                                num_workers=0)
}
```

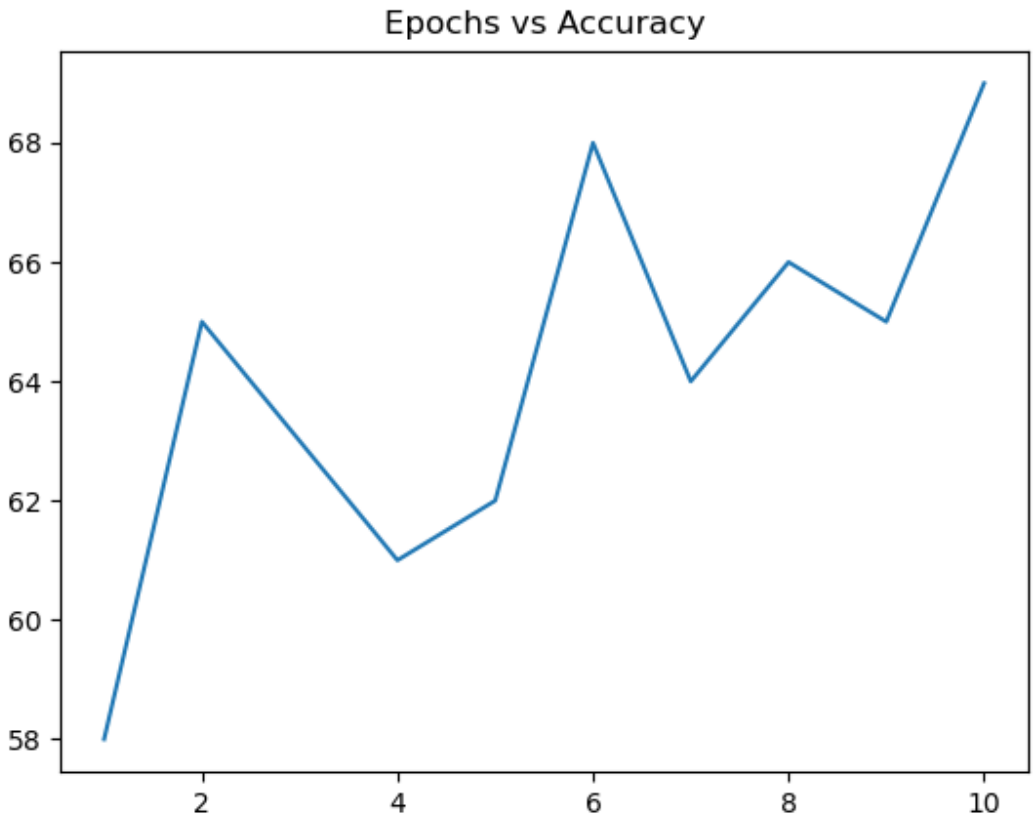
```
model = torchvision.models.resnet50(weights=None)
model = model.cuda()
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

Step-4:- Training the model and simultaneously calculating accuracy upto 10 epochs.

```
val_acc = []
train_acc = []
val_loss = []
train_loss = []
def train_model(model, criterion, optimizer, num_epochs=3):
    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch+1, num_epochs))
        print('-' * 10)

        for phase in ['train', 'validation']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0
            count = 0
            for inputs, labels in dataloaders[phase]:
                count += 1
                if(count % 100 == 0): print(count)
                inputs = inputs.cuda()
                labels = labels.cuda()
```

```
                outputs = model(inputs)
                loss = criterion(outputs, labels)

                if phase == 'train':
                    optimizer.zero_grad()
                    loss.backward()
                    optimizer.step()

                _, preds = torch.max(outputs, 1)
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss / len(image_datasets[phase])
            epoch_acc = running_corrects.double() / len(image_datasets[phase])
            if phase == 'train':
                train_acc.append(epoch_acc)
                train_loss.append(epoch_loss)
            if phase == 'validation':
                val_acc.append(epoch_acc)
                val_loss.append(epoch_loss)
            print('{} loss: {:.4f}, acc: {:.4f}'.format(phase,
                                                        epoch_loss,
                                                        epoch_acc))
    return model

model_trained = train_model(model, criterion, optimizer, num_epochs=10)
```

Accuracy after 10 epochs = 69



Using additional prompts on CLIP with ResNet50 backbone:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input di

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when yo
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current sessio
```

+ Code + Markdown

```
! pip install ftfy regex tqdm
! pip install git+https://github.com/openai/CLIP.git
```

```
import torch
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import clip
import os
import shutil
import random
from tqdm.notebook import tqdm
from pkg_resources import packaging

# Define basic and modified prompts
templates = [
    'a bad photo of a {}.',
    'a photo of many {}.',
    'a sculpture of a {}.',
    'a photo of the hard to see {}.',
    'a low resolution photo of the {}.',
    'a rendering of a {}.',
    'graffiti of a {}.',
    'a bad photo of the {}.',
    'a cropped photo of the {}.',
    'a tattoo of a {}.',
    'the embroidered {}.',
    'a photo of a hard to see {}.',
    'a bright photo of a {}.'
```

```
    'a bright photo of a {}.',
    'a photo of a clean {}.',
    'a photo of a dirty {}.',
    'a dark photo of the {}.',
    'a drawing of a {}.',
    'a photo of my {}.',
    'the plastic {}.',
    'a photo of the cool {}.',
    'a close-up photo of a {}.',
    'a black and white photo of the {}.',
    'a painting of the {}.',
    'a painting of a {}.',
    'a pixelated photo of the {}.',
    'a sculpture of the {}.',
    'a bright photo of the {}.',
    'a cropped photo of a {}.',
    'a plastic {}.',
    'a photo of the dirty {}.',
    'a jpeg corrupted photo of a {}.',
    'a blurry photo of the {}.',
    'a photo of the {}.',
    'a good photo of the {}.',
    'a rendering of the {}.',
    'a {} in a video game.',
    'a photo of one {}.',
    'a doodle of a {}.',
```

```
    'a close-up photo of the {}.',
    'a photo of a {}.',
    'the origami {}.',
    'the {} in a video game.',
    'a sketch of a {}.',
    'a doodle of the {}.',
    'a origami {}.',
    'a low resolution photo of a {}.',
    'the toy {}.',
    'a rendition of the {}.',
    'a photo of the clean {}.',
    'a photo of a large {}.',
    'a rendition of a {}.',
    'a photo of a nice {}.',
    'a photo of a weird {}.',
    'a blurry photo of a {}.',
    'a cartoon {}.',
    'art of a {}.',
    'a sketch of the {}.',
    'a embroidered {}.',
    'a pixelated photo of a {}.',
    'itap of the {}.',
    'a jpeg corrupted photo of the {}.',
    'a good photo of a {}.',
    'a plushie {}.',
    'a photo of the nice {}.',
```

```
    'a photo of the small {}.',
    'a photo of the weird {}.',
    'the cartoon {}.',
    'art of the {}.',
    'a drawing of the {}.',
    'a photo of the large {}.',
    'a black and white photo of a {}.',
    'the plushie {}.',
    'a dark photo of a {}.',
    'itap of a {}.',
    'graffiti of the {}.',
    'a toy {}.',
    'itap of my {}.',
    'a photo of a cool {}.',
    'a photo of a small {}.',
    'a tattoo of the {}.',
]
```

```
def split_images(data_dir):
    images = "/kaggle/working/images"
    if(not os.path.exists(images)):
        os.makedirs(images)
    for filename in os.listdir(data_dir):
        class_name = "_".join(list(filename.split("_"))[:-1])
        class_dir = os.path.join(images, class_name)
        src_dir = os.path.join(data_dir,filename)
        if(not os.path.exists(class_dir)):
            os.makedirs(class_dir)
        shutil.copy(src_dir,class_dir)

    shutil.make_archive(os.path.join("/kaggle/working/","oxford_dataset_iiit-pet_splitted"), 'zip', images)


# Create zeroshot weights
def zeroshot_classifier(classnames, templates):
    with torch.no_grad():
        zeroshot_weights = []
        for classname in tqdm(classnames):
            texts = [template.format(classname) for template in templates] #format with class
            texts = clip.tokenize(texts).cuda() #tokenize
            class_embeddings = model.encode_text(texts) #embed with text encoder
            class_embeddings /= class_embeddings.norm(dim=-1, keepdim=True)
            class_embedding = class_embeddings.mean(dim=0)
```

```python
# Create zeroshot weights
def zeroshot_classifier(classnames, templates):
    with torch.no_grad():
        zeroshot_weights = []
        for classname in tqdm(classnames):
            texts = [template.format(classname) for template in templates] #format with class
            texts = clip.tokenize(texts).cuda() #tokenize
            class_embeddings = model.encode_text(texts) #embed with text encoder
            class_embeddings /= class_embeddings.norm(dim=-1, keepdim=True)
            class_embedding = class_embeddings.mean(dim=0)
            class_embedding /= class_embedding.norm()
            zeroshot_weights.append(class_embedding)
        zeroshot_weights = torch.stack(zeroshot_weights, dim=1).cuda()
    return zeroshot_weights

def accuracy(output, target, topk=(1,)):
    pred = output.topk(max(topk), 1, True, True)[1].t()
    correct = pred.eq(target.view(1, -1).expand_as(pred))
    return [float(correct[:k].reshape(-1).float().sum(0, keepdim=True).cpu().numpy()) for k in topk]
```

```python
# Load pre-trained CLIP model
clip.available_models()
device = "cuda" if torch.cuda.is_available() else "cpu"
model, preprocess = clip.load("RN50")
input_resolution = model.visual.input_resolution
context_length = model.context_length
vocab_size = model.vocab_size

print("Model parameters:", f"{np.sum([int(np.prod(p.shape)) for p in model.parameters()]):,}")
print("Input resolution:", input_resolution)
print("Context length:", context_length)
print("Vocab size:", vocab_size)
```

```
split_images(data_dir)
```

```
data_dir = "/kaggle/input/100-cats-and-dogs/Cats_and_Dogs"
# data_dir = "/kaggle/working/"
data_set = datasets.ImageFolder(data_dir, transform=preprocess)
# data_set = datasets.OxfordIIITPet(data_dir, transform=preprocess,download = True)
loader = torch.utils.data.DataLoader(data_set, batch_size=32, num_workers=2)
classes = data_set.classes
zeroshot_weights = zeroshot_classifier(classes, templates)


with torch.no_grad():
    top1, n = 0., 0.
    for i, (images, target) in enumerate(tqdm(loader)):
        images = images.cuda()
        target = target.cuda()

        # predict
        image_features = model.encode_image(images)
        image_features /= image_features.norm(dim=-1, keepdim=True)
        logits = 100. * image_features @ zeroshot_weights


        # measure accuracy
        acc1 = accuracy(logits, target)
        top1 += acc1[0]
```

```
       n += images.size(0)

top1 = (top1 / n) * 100


print(f"Accuracy: {top1:.2f}")
```

Accuracy obtained is

Clearly, we can observe that we get better accuracy. The reason behind this is the usage of prompts. By carefully designing prompts for specific tasks, models can achieve better performance on those tasks than they would with generic prompts.

The use of prompts can further improve the performance of CLIP by fine-tuning the model's behaviour on specific tasks. By providing explicit prompts that guide the model's behavior, we can improve its accuracy and reduce its reliance on preconceptions or biases. This is particularly useful when dealing with complex language or ambiguous visual cues.

In contrast, using ResNet50 without prompts is a more straightforward approach that may be sufficient for certain image classification tasks. However, ResNet50 does not have the same flexibility as CLIP in terms of its ability to perform multi-modal tasks or adapt to specific prompts.

Overall, while ResNet50 is a powerful image classification model, the use of prompts on CLIP with a ResNet50 backbone can further improve its performance on a wide range of tasks.