GPT-3 & Clip for Image Classification

Gregory Presser, Jacob Khalili *

October 28, 2022

1 Introduction

Humans [and maybe (other) animals too] identify objects using features. For example, using a person may identify a dog, because it has 4 legs, fur, and a tail. In Visual Classification via Description from Large Language Models [MV22] the authors attempt to get descriptors of categories from GPT-3 to increase classification accuracy of a model with the CLIP style architecture. We attempt to reproduce this these results.

2 Methodology

2.1 Descriptor Generation via GPT-3

We utilized a Open AI's GPT-3 Davinci Model [BMR+20] for generating descriptors based on model categories, via API. We prompted the model in a similar manner to the paper, as shown in Figure 1.

GPT-3 was prompted with category-name replaced for each one of the categories in the Image-Net-1000 classification data-set. [RDS+15]

Q: What are useful visual features for distinguishing a lemur in a photo?
A: There are several useful visual features to tell there is a lemur in a photo:
- four-limbed primate
- black, grey, white, brown, or red-brown
- wet and hairless nose with curved nostrils
- long tail
- large eyes
- furry bodies
- clawed hands and feet

Q: What are useful visual features for distinguishing a {category-name} in a photo? A: There are several useful visual features to tell there is a {category-name} in a photo:

Figure 1: GPT-3 Prompt to obtain descriptors

2.2 Image Classification via Model with CLIP Architecture

We then passed the descriptors to a pre-trained model with a CLIP architecture. [RKH+21] We used the ViT-B/32 pre-trained model, which was available via Open-AI's Clip library. We passed in the text in form: "a photo of a [class-name]" for the Vanilla Clip Model, and for our model "a photo of a [class-name], which (has/is) [descriptor]"

For the Vanilla clip model, to obtain the make a prediction, we use the class with the greatest log-probabilities. However, our model sums the weights of each of the descriptors for a category. We were able to optimize this process via a clever matrix multiplication (similar technique to binary-coding in Comm-Theory).

*Students at the Cooper Union for the Advancement of Science & Art. This work was done as part of ECE-472, a graduate course in Deep Learning

3 Results

Figure 4 shows some sample results of both models test, and the ground truth data. Figure 2 shows the results for different architetures

Model	Ours	CLIP Baseline	Paper Results	Paper Baseline
ViT-B/32	60.52%	59.52%	62.97%	58.46%

Figure 2: Results when comparing a baseline CLIP model VS

4 Complaints/Future Work/Conclusion

4.1 Complaints

The paper did not give enough detail to accurately reproduce there work. There were several assumptions that were made, which may have differed from the authors. This includes: exactly where the data-set was obtained from, and pre-processing of the image (including cropping etc).

Image-Net classes are also confusing to me as a human (this paper was not written by GPT-3), and sometime wrong. A notable example that we discovered is shown in Figure 3, which is clearly not a cape.

Figure 3: This image was listed as a cape in the Image-Net dataset

4.2 Future Work

In order to properly replicate this paper, we would have needed more compute available to us. Running these model on the entire Image-Net data-set was not feasible given our compute constraints. This likely explains some of the variation from the the results shown in the paper.

4.3 Conclusion

We were able to reproduce the results of the paper to a lesser effect. The original paper yeilded a classification accuracy of 62.97% with a baseline clip model resulting in 58.46% which is approximetly a 4% improvment. In our experiments we were able to produce a classification accurarcy of 60.52% with

a baseline clip model resuling in 59.52% while classifying 5000 random images which is approximetly a 0.5% improvment.

References

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[MV22] Sachit Menon and Carl Vondrick. Visual classification via description from large language models, 2022.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and

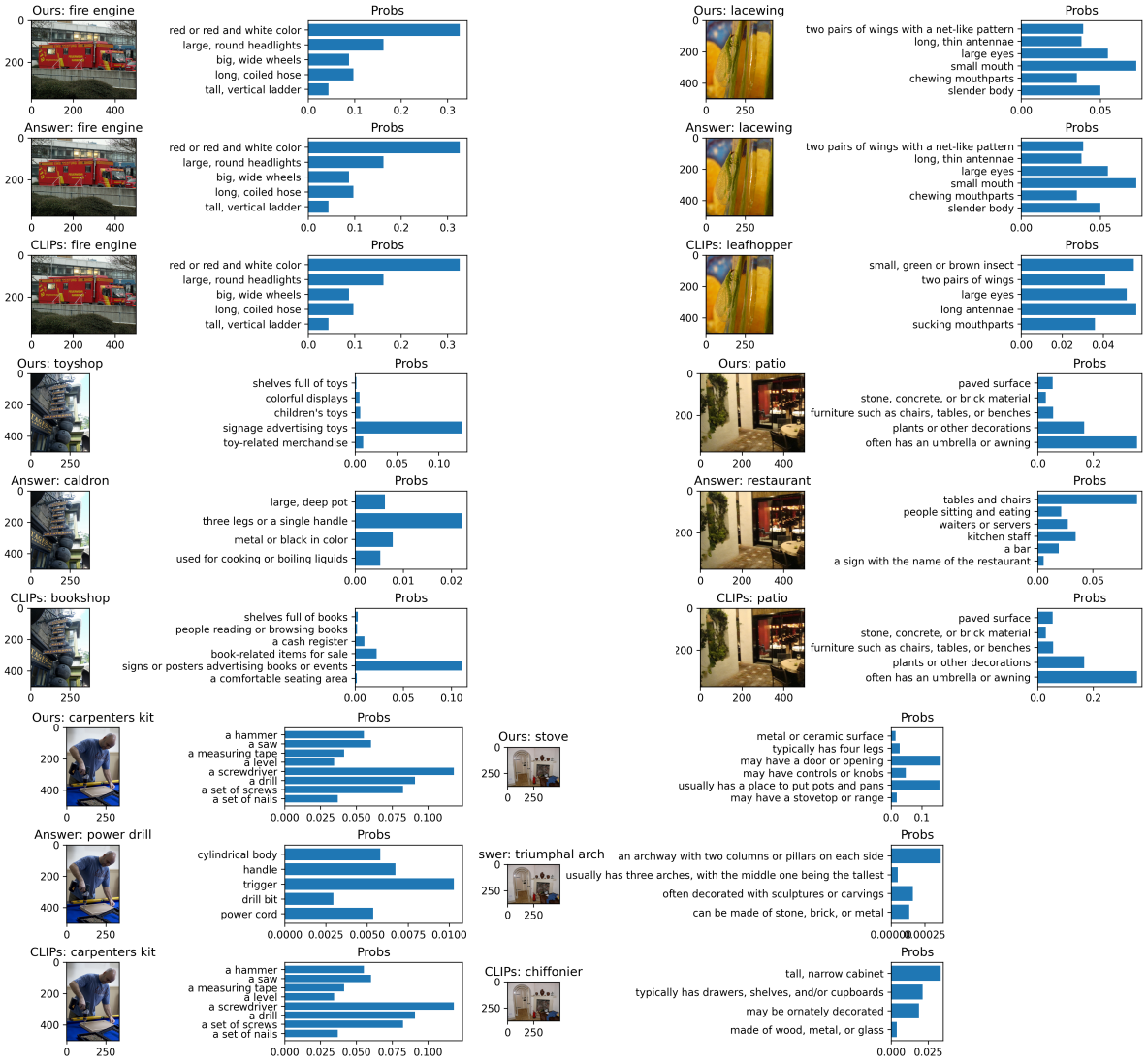

Figure 4: Sample Results from both models, and the correct answer, with probabilities of ground truth descriptors

Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[RKH+21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

```
import torch
import clip
from PIL import Image
import numpy as np


class ClipHandler:

    def __init__(self, labels=None):
        self.device = "cuda" if torch.cuda.is_available() else "cpu"
        self._model, self._preprocess = clip.load("ViT-B/32", device=self.device)
        self._labels = labels

    def _processImage(self, path:str):
        return self._preprocess(Image.open(path)).unsqueeze(0).to(self.device)

    def _processImageP(self, img):
        return self._preprocess(img).unsqueeze(0).to(self.device)

    @property
    def labels(self):
        return self._labels

    @labels.setter
    def labels(self, labels:list[str]):
        self._labels = self._tokenize(labels)

    def _tokenize(self, labels: list[str]):
        return clip.tokenize(labels).to(self.device)


    def predict(self, imagePaths: list[str]):
        if self._labels is None:
            raise ValueError("Please Set the label values first")

        images = torch.tensor(np.concatenate([self._processImage(i) for i in ima
gePaths]), device=self.device)

        with torch.no_grad():

            logits_per_image, logits_per_text = self._model(images, self._labels
)
            probs = logits_per_image.softmax(dim=-1).cpu().numpy()

        return probs

    def predictImages(self, X):
        if self._labels is None:
            raise ValueError("Please Set the label values first")

        images = torch.tensor(np.concatenate([self._processImageP(i) for i in X]
), device=self.device)

        with torch.no_grad():

            logits_per_image, logits_per_text = self._model(images, self._labels
)
            probs = logits_per_image.softmax(dim=-1).cpu().numpy()

        return probs
```



```

# Theses are the tests
if __name__ == "__main__":
    print("Starting")
    ch = ClipHandler()
    ch.labels = ["a diagram", "a dog", "a cat"]
    print(ch.predictBatch(["../data/clip.jpeg","../data/clip2.jpeg"]))
```

```
from CLIP_model import ClipHandler
from descriptor_generater import LabelsWithDescriptors
from datasets import load_dataset
import os
from sys import argv
import json
import numpy as np
from tqdm import tqdm

def get_label_descriptors(early_stop=None):
    if len(argv) != 3:
        print("Usage: python CLIP_runner.py –[g|r] [file_path]")
        exit(1)
    elif argv[1] != "-g" and argv[1] != "-r":
        print("Usage: python CLIP_runner.py –[g|r] [file_path]")
        exit(2)

    [_, flag, pathR] = argv
    path = os.getcwd() + "/" + pathR
    with open(path) as f:
        if flag == "-g":
            out_path = os.getcwd() + "/../data/new_cats.json"
            lst = LabelsWithDescriptors.create_descriptors_from_label_file(f, ea
rly_stop, out_path)
            return lst
        return LabelsWithDescriptors.read_list_from_file(f, early_stop)




class Experiment:

    def __init__(self, labelDescriptList):
        self.labelDescriptList = labelDescriptList
        self.folderIndexMap = {l.folder: l.index for l in labelDescriptList}
        self.originalLabels = np.array([Experiment._make_label_from_class(l.labe
ls[0]) for l in labelDescriptList])
        self.descriporLabels = np.array([ Experiment._combine_label_and_descript
or(l.labels[0], d) for l in labelDescriptList for d in l.descriptors])
        self.descriptor_matrix = np.zeros((len(self.descriporLabels), len(self.o
riginalLabels)))
        offset = 0
        for (ci,l) in enumerate(labelDescriptList):
            val = 1/len(l.descriptors)
            for _ in l.descriptors:
                self.descriptor_matrix[offset][ci] = val
                offset += 1
        self.modelHandler = ClipHandler()

    @staticmethod
    def _combine_label_and_descriptor(cls: str, descriptor: str) -> str:
        # TODO make the has/is more general maybe use a CKY parse or something a
long those lines?
        return f"a phot of a {cls}, which (has/is) {descriptor}"

    @staticmethod
    def _make_label_from_class(cls:str) -> str:
        return f"a photo of a {cls}"


    # TODO Test & Vectorize
    def run_original(self, X: list[str], Y:np.ndarray ) -> float:
```



```
        self.modelHandler.labels = self.originalLabels
        wrong = np.count_nonzero(self.modelHandler.predict(X).argmax(axis=1)
.T)
        return 1 - (float(wrong) / float(len(X)))

    def run_originalP(self, X,  Y: np.ndarray ) -> float:
        # self.modelHandler.labels = self.originalLabels
        wrong = np.count_nonzero(self.get_predictionsOP(X)[0] - Y.T)
        return 1 - (float(wrong) / float(len(X)))

    def get_predictionsOP(self, X) -> np.ndarray:
        self.modelHandler.labels = self.originalLabels
        PHI = self.modelHandler.predictImages(X)
        return PHI.argmax(axis=1), PHI

    # TODO Test & Vectorize
    def run_descriptor(self, X: list[str], Y: np.ndarray ) -> float:
        self.modelHandler.labels = self.descriporLabels
        PHI = self.modelHandler.predict(X)
        print("Prediction Complete Performing Matrix Multiply")
        wrong = np.count_nonzero((PHI @ self.descriptor_matrix).argmax(axis=
 Y.T)
        return 1 - (float(wrong) / float(len(Y)))


    def run_descriptorP(self, X, Y: np.ndarray ) -> float:
        # self.modelHandler.labels = self.descriporLabels
        # PHI = self.modelHandler.predictImages(X)
        # print("Prediction Complete Performing Matrix Multiply")
        wrong = np.count_nonzero(self.get_predictionsDP(X)[0] - Y.T)
        return 1 - (float(wrong) / float(len(Y)))

    def get_predictionsDP(self, X) -> np.ndarray:
        self.modelHandler.labels = self.descriporLabels
        PHI = self.modelHandler.predictImages(X)
        return (PHI @ self.descriptor_matrix).argmax(axis=1), PHI



def build_data(exp: Experiment, img_count=None):
    dataset = load_dataset("imagenet-1k", use_auth_token=True, split="validation"
uffle(seed=31415)
    X = []
    Y = []
    for (i,data) in tqdm(enumerate(dataset), total=img_count if img_count is
 None else dataset.num_rows):
        if data["label"] >= len(exp.labelDescriptList):
            continue
        if len(X) >= img_count:
            break
        X.append(data["image"])
        Y.append(data["label"])
    return X, np.array(Y)

def main():
    early_stop = None # so I can test it without running it on all of Imagel
    img_count = 5000

    exp = Experiment(get_label_descriptors(early_stop))
    # Sanity check
    assert(len(exp.labelDescriptList) ==1000 if early_stop is None else earl
op)
```

```
    X,Y = build_data(exp, img_count)
    assert(early_stop is None and len(X) == img_count)


    print("starting baseline CLIP model")
    original_acc = exp.run_originalP(X,Y)
    print("starting our test")
    our_acc = exp.run_descriptorP(X,Y)

    print(f"CLIP base acc: {original_acc}\nOur acc: {our_acc}")

# This is the main program
if __name__ == '__main__':
    main()
```

```
import openai
import json
import pathlib
import json
from time import sleep
from json import JSONEncoder
import os
from sys import argv
from tqdm import tqdm

openai.api_key = os.getenv("OPEN_AI_API_KEY")

class LabelsWithDescriptors:

    CLASS_NAME = "LabelsWithDescriptors"


    def __init__(self, index, labels,folder,descriptors=None):
        self.index = index
        self.labels = labels
        self.folder = folder
        if descriptors is not None:
            self.descriptors = descriptors
        else:
            self.descriptors = LabelsWithDescriptors._get_parsed_response(self.l
abels[0])

    def __str__(self):
        return f"#{self.index}, folder{self.folder}: {self.labels} -> {self.descriptors}"

    @staticmethod
    def _build_input(category_name: str) -> str:
        return f"""Q: What are useful visual features for distinguishing a lemur in a photo?
A: There are several useful visual features to tell there is a lemur in a
    photo:
– four–limbed primate
– black, grey, white, brown, or red–brown
– wet and hairless nose with curved nostrils
– long tail
– large eyes
– furry bodies
– clawed hands and feet

Q: What are useful visual features for distinguishing a {category_name} in a photo?
A: There are several useful visual features to tell there is a {category_name} in a
    photo:
–"""


    @staticmethod
    def _get_response(input: str):
        return openai.Completion.create(
          model="text–davinci–002",
          prompt=LabelsWithDescriptors._build_input(input),
          temperature=0.7,
          max_tokens=100,
          top_p=1,
          frequency_penalty=0,
          presence_penalty=0
        )
```



```
    @staticmethod
    def _get_parsed_response(input:str):
        raw = LabelsWithDescriptors._get_response(input).get("choices")[0]["te
        return [x[1:].strip() for x in raw.split("\n")]


    @staticmethod
    def read_list_from_file(file_path, early_stop = None):
        return json.load(file_path, object_hook=LabelsWithDescriptors.json_d
er)[:early_stop]

    @staticmethod
    def combine_decoded_output(file_paths: list[str], length = 1000):
        lst = [None] * length
        def inner(path:str):
            with open(path, "r") as f:
                return LabelsWithDescriptors.read_list_from_file(f)

        obj_lists = [inner(path) for path in file_paths]
        for obj_list in obj_lists:
            for obj in obj_list:
                lst[obj.index] = obj
        return lst


    @staticmethod
    def json_decoder(obj):
        if "__type__" in obj and obj["__type__"] == LabelsWithDescriptors.CLASS
E:
            return LabelsWithDescriptors(obj['index'], obj["labels"], obj["folder
obj["descriptors"])
        return obj

    @staticmethod
    def create_descriptors_from_label_file(f, early_stop=None, save_file=No
kip=0):
        cats = json.load(f)['cats']
        lst = []
        for (li, [folder,labels]) in tqdm(enumerate(cats[skip:early_stop]),
="Reading JSON: ", total = (1000 if early_stop is None else early_stop) - skip)
            i = li + skip
            if i % 50 == 0 and i != 0:
                sleep(240)
            lst.append(LabelsWithDescriptors(i,labels, folder))

            if save_file is not None:
                with open(save_file, "w+") as f:
                    json.dump(lst, f, cls=LabelsWithDescriptors.MyEncoder)
        return lst

    class MyEncoder(JSONEncoder):
        def default(self,o):
            d = o.__dict__
            d["__type__"] = LabelsWithDescriptors.CLASS_NAME
            return d


# These are the tests
if __name__ == "__main__":
    if len(argv) != 3:
        print("Usage: python descriptor_test.py –[g|r|p] [file_path]")
        exit(1)
```

```
    elif argv[1] != "-g" and argv[1] != "-r" and argv[1] != "-p" :
        print("Usage: python descriptor_test.py -[g|r|p] [file_path]")
        exit(2)

    [_, flag, pathR] = argv

    if flag == "-p":
        lst = LabelsWithDescriptors.combine_decoded_output(["../data/988_cats.json", "../
data/989_cats.json"])
        print(len(lst))
        print(lst[0])
        print(lst[999])
        with open(os.getcwd() + "/../data/new_cats.json", "w+") as f:
            json.dump(lst, f, cls=LabelsWithDescriptors.MyEncoder)

        exit(0)
    path = os.getcwd() + "/" + pathR
    with open(path) as f:
        if flag == "-g":
            newCats = LabelsWithDescriptors.create_descriptors_from_label_file(f
,1000,os.getcwd() + "/../data/new_cats.json", 989)
            exit(0)
        lables = LabelsWithDescriptors.read_list_from_file(f,1000)
        for label in lables:
            print(label)
        exit(0)
```

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from CLIP_runner import get_label_descriptors, build_data, Experiment
import numpy as np


def make_image(axis, title: str, img):
    axis.imshow(img)
    axis.set_title(title)
    return axis


def main():
    lbls = get_label_descriptors(None)
    exp = Experiment(lbls)

    X,Y = build_data(exp,50)

    print("Our predictions starting")
    (preds_o, phis_o) = exp.get_predictionsDP(X)

    print("CLIP predictions starting")
    (preds_c, phis_c) = exp.get_predictionsOP(X)
    print("Predictions finished")

    for (x,y,pred_o,phi_o,pred_c,phi_c) in zip(X,Y,preds_o,phis_o,preds_c,phis_c
):
        fig, axis = plt.subplots(3,2, layout="constrained")
        axis[0][0] = make_image(axis[0][0], f"Ours: {lbls[pred_o].labels[0]}", x)

        axis[1][0] = make_image(axis[1][0], f"Answer: {lbls[y].labels[0]}", x)

        axis[2][0] = make_image(axis[2][0], f"CLIPs: {lbls[pred_c].labels[0]}", x)

        descriptors_pred = lbls[pred_o].descriptors
        descriptor_pred_vals = phi_o[np.nonzero(exp.descriptor_matrix.T[pred_o])
]
        d_pos_pred = np.arange(len(descriptors_pred))
        axis[0][1].barh(d_pos_pred, descriptor_pred_vals)
        axis[0][1].set_yticks(d_pos_pred,descriptors_pred)
        axis[0][1].invert_yaxis()
        axis[0][1].set_title("Probs")

        descriptors_true = lbls[y].descriptors
        descriptor_true_vals = phi_o[np.nonzero(exp.descriptor_matrix.T[y])]
        d_pos_true = np.arange(len(descriptors_true))
        axis[1][1].barh(d_pos_true, descriptor_true_vals)
        axis[1][1].set_yticks(d_pos_true,descriptors_true)
        axis[1][1].invert_yaxis()
        axis[1][1].set_title("Probs")

        descriptors_CLIP = lbls[pred_c].descriptors
        descriptor_CLIP_vals = phi_o[np.nonzero(exp.descriptor_matrix.T[pred_c])
]
        d_pos_CLIP = np.arange(len(descriptors_CLIP))
        axis[2][1].barh(d_pos_CLIP, descriptor_CLIP_vals)
        axis[2][1].set_yticks(d_pos_CLIP,descriptors_CLIP)
        axis[2][1].invert_yaxis()
        axis[2][1].set_title("Probs")

        plt.savefig(f"{'_'.join(lbls[y].labels[0].split(' '))}.png", dpi=300)
```



```
if __name__ == '__main__':
    main()
```