CLIP Chronicles: Bridging Pixels and Prose in Deep Learning

Paper Chosen – Learning Transferable Visual Models From Natural Language supervision

(Group 6)

Moise Brutus **Chinmay Shah** **Praya Cheekapara** **Joshua J. Cook**

1. Related Work

The inception of one of the first artificial neural networks is credited to Dr. Kunihiko Fukushima in the 1980s. In his landmark paper, Fukushima builds on earlier work by Hubel and Wiesel who conceptualized a network of simple and complex “cells” that are capable of recognizing stimulus patterns based on geometrical similarity that simulated the way the human visual cortex processes visual information [1]. This network was nicknamed the “Neocognitron,” and was built off of Fukushima’s original concept of the “Cognitron" in 1975, which was also a self-organizing multilayered neural network model [1]. The main improvement introduced in the Neocognitron was the response of the network not being affected by the position of individual stimulus patterns (i.e., shifted patterns) [1]. Additionally, the Neocognitron introduced architecture capable of supervised learning in addition to the unsupervised methods that were present in the Cognitron [1]. Both the Cognitron and the Neocognitron laid the foundation for the development of more advanced convolutional neural networks (CNNs) throughout the 1990s and to modern models, as shown in Figure 1.

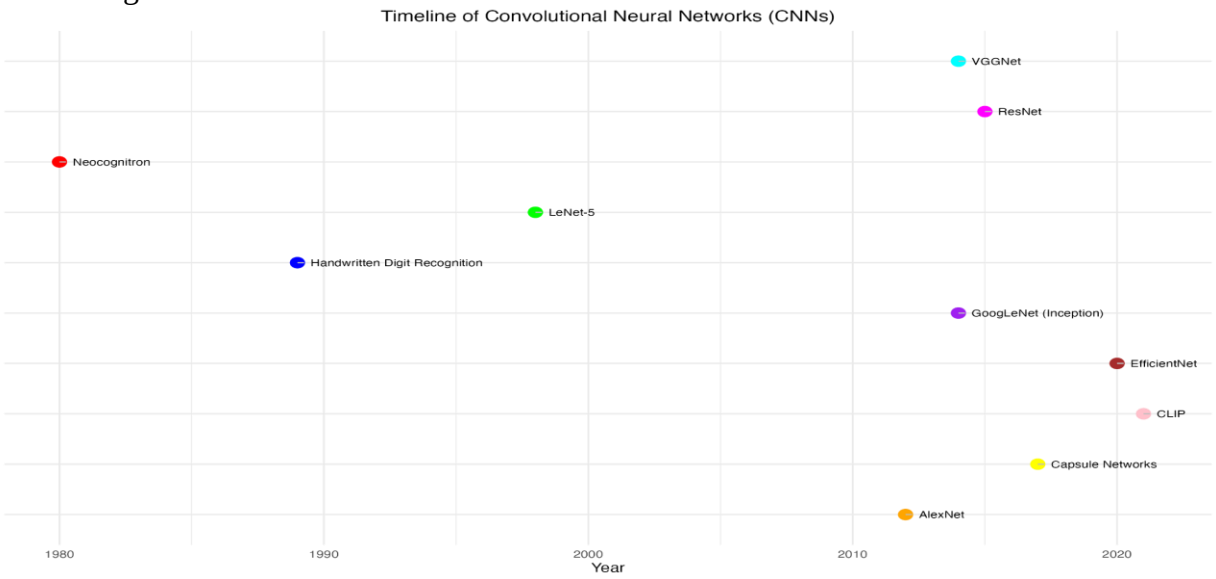

Figure 1. Timeline of CNN development. CNN development was initiated by the work of Fukushima in the 1970s and 1980s via the introduction of the Cognitron and Neocognitron. This spurred the development of more complex CNNs during the 1990s such as LeNet-5, and modern models such as GoogLeNet and CLIP [1-6].

In the late 1990s, LeCun Et. Al. applied the concept of backpropagation to neural networks with the goal of recognizing digits of different handwriting styles [2]. Backpropagation is a gradient estimation algorithm used to train neural networks, including CNNs. In their paper, LeCun Et. Al. utilized the MNIST database that contained 60,000 training and 10,000 testing images of handwriting from the American Census Bureau [2]. LeCun Et. Al. directly built on the work of Fukushima, creating the LeNet CNN by giving the model an example image, asking it to predict the image (i.e., training), and then updating the model setting by comparing the ground truth to the predicted label (i.e., testing) [2]. The LeNet model (now known as LeNet-5) was a groundbreaking application of artificial neural networks from which further development in the 2000s and 2010s was built on [2].

One of the primary limitations of developing models like LeCun-5 stemmed from the availability of large-scale image data. Due to this, in 2006 Fei-Fei Li began working on a database of image data that could be widely utilized by artificial intelligence researchers. This database became ImageNet, which currently houses over 14 million images that are organized according to WordNet hierarchy. ImageNet is available to researchers but is also the focus of the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which evaluates algorithms for object detection and image classification. The introduction of ImageNet did two things: first, a large image-based dataset became widely available for researchers, and two, an annual competition was created that encouraged innovations in CNN development.

In 2012, the third year of the ILSVRC, Alex Krizhevsky Et. Al. won the competition with a top-5 error of 15.3%, which was 10.8% lower than the runner-up and utilizes a CNN [3]. The network, now known as AlexNet, has 8 layers (5 convolutional and 3 fully connected) that were trained on the ImageNet dataset (which included 1.2 million high-resolution images at the time) [3]. AlexNet represented a huge leap forward in the complexity of CNNs, with 60 million parameters and 650,000 neurons [3]. Building on this, the 2014 and 2015 ILSVRC were won by Karen Simonyan Et. Al. for their development of VGGNet and Microsoft for their development of ResNet respectively [4-5]. Both of these implementations of CNN were focused at increasing the depth of the networks [4-5].

More recently, in 2017, Geoffrey Hinton Et. Al. introduced the idea of Capsule Networks, which are groups of neurons in a network (capsules) that can detect objects or parts of objects across different viewpoints [6]. This innovation provided several improvements on traditional CNNs, including the preservation of spatial hierarchies, generalization to new views, and reduced data requirements [6]. A few years later, in 2020, Google introduced EfficientNet, which can scale CNNs in a very structured measure and provides high accuracy with fewer parameters using the ImageNet database. Most recently in 2021, Radford Et. Al. at OpenAI introduced the contrastive language-image pre-training (CLIP) [7]. CLIP is a CNN that learns visual concepts from natural language supervision and has been trained on a combination of images and their captions [7]. CLIP bridges the gap between traditional CNN and natural language processing, providing several innovations and advancements such as generalization across tasks, increased flexibility, and scalability, and having an interface to natural language [7].

2. Methodology

The CLIP (Contrastive Language Image Pretraining) model is founded on the concept of acquiring perceptual understanding through natural language.

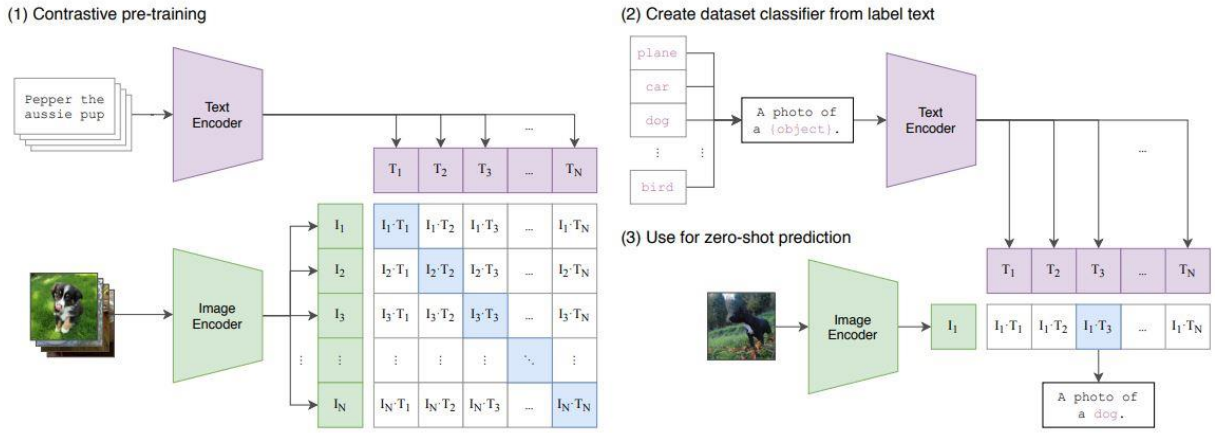

Figure 2. Summary of the CLIP architecture. CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes. [7]

The fundamental concept behind CLIP involves utilizing captioned images picked up from the internet to build a model that is capable of predicting text or a bunch of words associated with an image. The CLIP model achieves this by pretraining on a set of images and captions and generating a comparison between their encoding's output using an image and text encoder respectively. In this process, images that match well with corresponding text are assigned high values, whereas those that do not match receive low values. Here the model learns to place matching pairs nearby and place non-matching pairs further away. This approach, known as "contrastive learning," entails training the model to discern whether elements belong together or not.

2.1 Setting up a large dataset.

Establishing a comprehensive dataset presented a significant challenge in training this model. Previous efforts primarily relied on three datasets—MS-COCO, Visual Genome, and YFCC100M. MS-COCO and Visual Genome, each comprising approximately 100,000 crowd-labeled images, proved insufficient. Although YFCC100M boasted over 100 million images, the authors narrowed down the selection by filtering for quality and English-language descriptions, resulting in 15 million images comparable to those in ImageNet. The abundance of publicly available data served as a primary motivation for incorporating natural language supervision. The authors curated a dataset consisting of 400 million image-text pairs, averaging around 20,000 pairs per query.

2.2 Components of the CLIP model.

Clip as a model is a combination of a variety of subcomponents that are used to achieve this association between images and text. At its core, CLIP consists of a dual encoder system, which includes separate encoders for processing text and images. The image encoder is typically based on a convolutional neural network (CNN), capable of extracting hierarchical visual features from input images. Conversely, the text encoder leverages transformer-based models to encode textual descriptions or prompts. These encoders learn to represent images and text in a shared embedding space, where semantically similar concepts are positioned closer to each other. Additionally, CLIP incorporates a contrastive learning objective, where the model is trained to maximize the similarity between matching pairs of text and images while minimizing the similarity between non-matching pairs.

2.2.1 What is a Text Encoder?

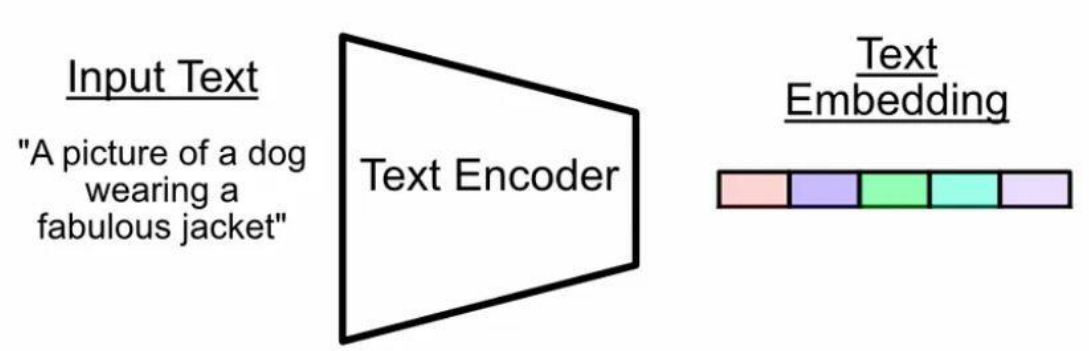

Figure 3. The above figure describes the role of a text encoder.

The text encoder within CLIP is a standard transformer encoder. A transformer can be thought of as a system which takes an entire input sequence of words, then re-presents, and compares those words to create an abstract and contextualized representation of the entire input. The self-attention mechanism within a transformer is the main mechanism that creates that contextualized representation.

2.2.2 Transformers

Transformers are a powerful class of deep learning models that have revolutionized natural language processing (NLP) tasks. Unlike traditional recurrent neural networks (RNNs) and convolutional neural networks (CNNs), which process data sequentially or locally, transformers leverage attention mechanisms to capture long-range dependencies and relationships within input sequences. At the heart of a transformer architecture are self-attention mechanisms, which enable the model to weigh the importance of each token in the input sequence with respect to every other token. By attending to relevant tokens and aggregating information across the entire sequence, transformers can effectively capture global context and semantic relationships, making them highly effective for tasks such as language translation, text generation, and sentiment analysis.

Additionally, transformers consist of multiple layers of self-attention and feedforward neural networks, allowing them to learn hierarchical representations of input sequences.

2.2.3 What is an Image Encoder ?

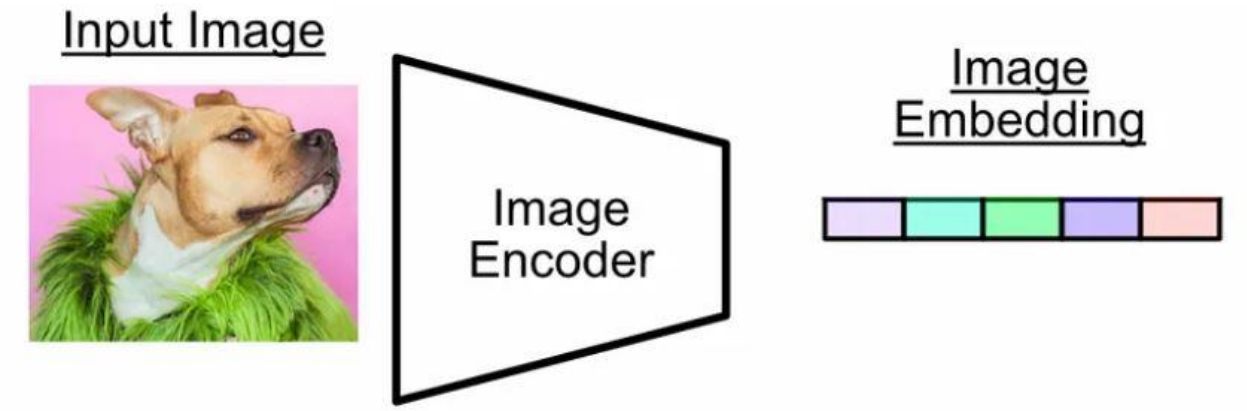

Figure 4. The above figure describes the role of an image encoder.

The image encoder converts an image into a vector (a list of numbers) that represents the image's meaning. In the CLIP paper we employ a RESNET-50 architecture as an image encoder.

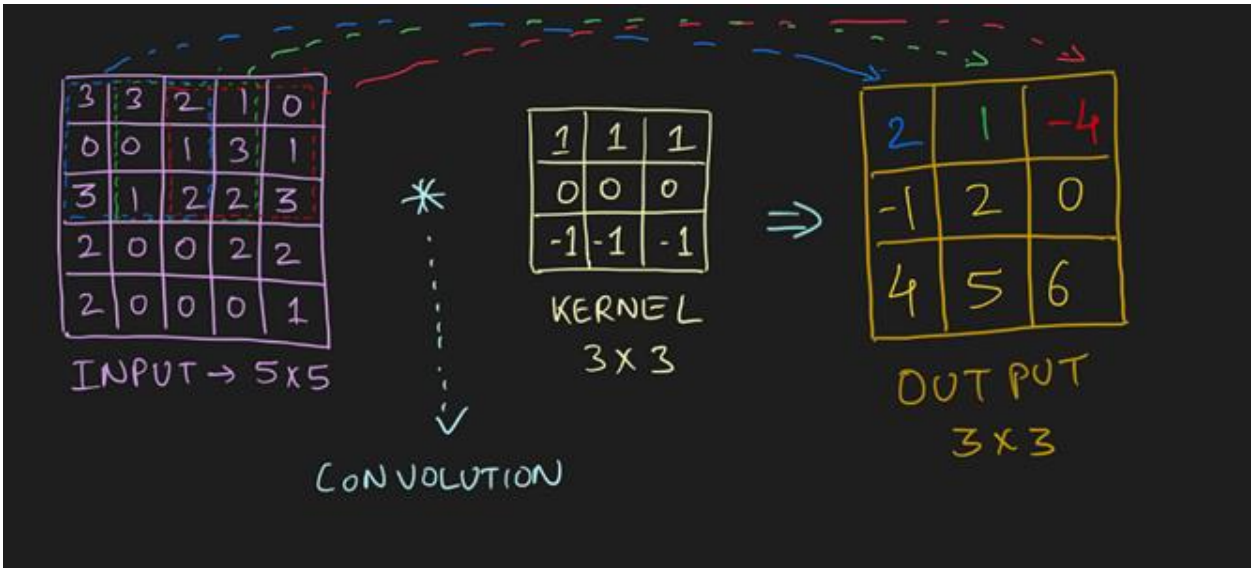

Figure 5. The above figure describes a simple convolution operation where we use a horizontal line detection kernel used to detect horizontal line features in the input image.

2.2.4. What is Convolution ?

Convolutional layers, the building blocks of CNNs, operate by applying filters (also known as kernels) to input images. These filters slide across the input image, computing a dot product

between the filter weights and the pixel values within each receptive field. Fig 4 below describes a simple convolution where we use a horizontal line detecting kernel to detect horizontal line features in the input image. By stacking these convolutional kernels together, we build a CNN. The following equation (eqn 1.) defines the convolution operation.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau \quad -(1)$$

2.2.5. Convolutional Neural Network

Convolutional neural networks (CNNs) are a class of deep learning models specifically designed for processing structured grid-like data, such as images. CNNs are characterized by their ability to automatically learn hierarchical representations of visual features directly from raw pixel data. CNNs can effectively learn increasingly abstract and hierarchical representations of visual features, leading to robust performance in tasks such as image classification, object detection, and image segmentation. The whole idea behind a convolutional network is, by doing a combination of convolutions and down sampling of an image, you can extract more and more subtle feature representations which can be turned into some final output. Fig 6. describes a classic CNN architecture from a very famous object detection model YOLO[8]. In the figure each of these boxes describe an input image's horizontal and vertical size, as well as their "depth", in terms of number of features. The input image is an RGB image, so it has a depth of 3. By extracting more and more features, and down sampling the image via max pooling, the network distills the image into an abstract representation which is trained to hold some meaning about the image.

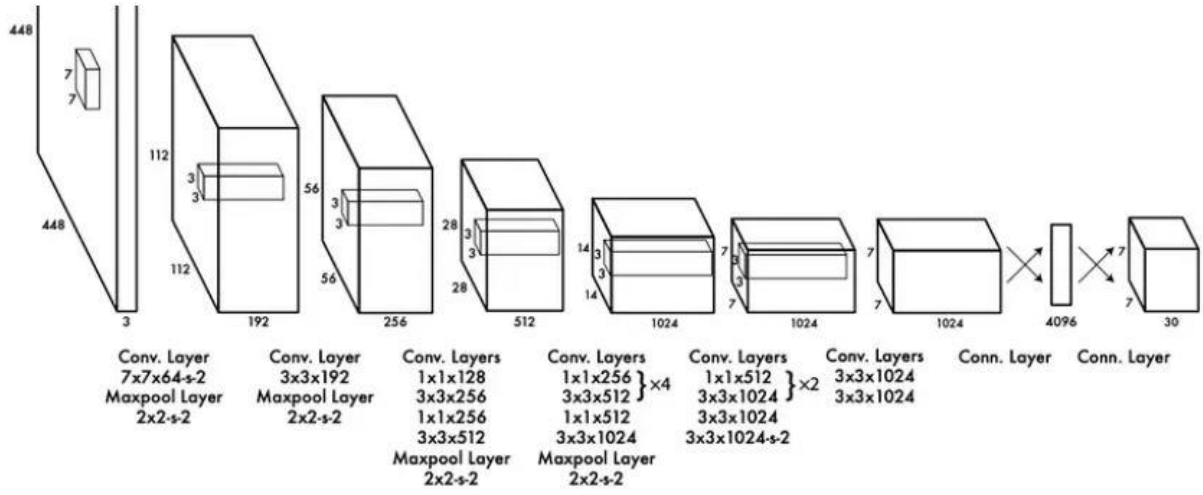

Figure 6. A classic convolutional architecture from the YOLO paper, an landmark object detection model.

2.2.3 Comparing the Cosine Similarity.

Cosine similarity is a measure used to determine the similarity between two vectors in a high-dimensional space by computing the cosine of the angle between them. In machine learning and

natural language processing, cosine similarity is valuable for comparing the similarity of documents, sentences, or words represented as vectors in a vector space model. Widely used in machine learning and natural language processing, it compares vectors' directions rather than magnitudes, making it robust for tasks like document retrieval, text classification, and recommendation systems. The following equation 2 describes how the cosine similarity is calculated.

$$\text{cosine similarity} = \frac{A \cdot B}{\| A \| \, \| B \|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \quad - (2)$$

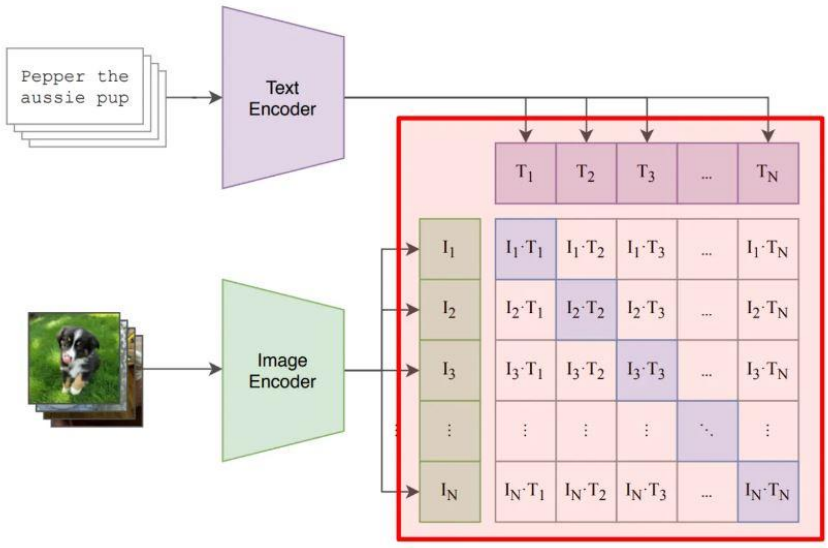

Figure 7. The figure focuses on putting the first and second part together and calculating the contrastive loss.

In the CLIP model as seen in fig 7. cosine similarity is employed to calculate the contrastive loss by comparing the cosine similarity between text and image embeddings. This involves computing the cosine similarity between the normalized text embedding and the normalized image embedding, which helps quantify the semantic similarity between the given text and image pair. By maximizing the cosine similarity for positive pairs (matching text and image) and minimizing it for negative pairs (mismatched text and image), the model learns to effectively align semantic representations across different modalities.

2.3 Zero Shot Classification

Zero-shot classification in the CLIP model enables it to recognize objects or concepts that it hasn't been explicitly trained on. By leveraging a large pre-trained language and vision model, CLIP learns to associate text prompts with corresponding images during training, allowing it to

generalize to unseen classes at inference time. This capability is achieved through the model's ability to understand natural language descriptions and relate them to visual content, enabling versatile and adaptive classification without the need for fine-tuning on specific classes.

3. Impact and Critical Assessment

Impact - Since CLIP allows people to design their own classifiers, how classes are designed influences model performance and model biases. There are also discrepancies across gender and race for the people categorized into “crime” and “non-human” categories even when care is taken into thoughtful class design. Since CLIP does not need task-specific training data, it can unlock specific tasks with greater ease. This also means that it may raise privacy or surveillance related risks [7].

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

Table 1. Comparing CLIP to prior zero-shot transfer image classification work. CLIP improves performance on all three datasets by a large amount. This improvement reflects many differences since the development of Visual N-Grams (Li et al., 2017).

Accuracy - Table 1,compares visual n-Grams to CLIP. The best CLIP model improves accuracy on ImageNet from a proof of concept 11.5% to 76.2% and matches the performance of the original ResNet50 despite using none of the crowd-labeled training examples[7].

Results - Figure 8 showed CLIP’s zero-shot classifiers performance when compared to a simple off-the-shelf baseline which is fitting a logistic regression classifier on the features of the canonical ResNet50. This comparison was across 27 datasets. Zero-shot CLIP outperforms this baseline slightly on 16 of the 27 datasets. CLIP underperforms when it comes to complex or abstract tasks such as counting objects in synthetic scenes (CLEVR Counts).This can show the the poorer capabilities of zero-shot CLIP on more complex tasks. Figure 9 summarizes the findings of the representation learning capabilities of a model. The performance on the from dataset evaluation suite was from Kornblith et al [9]. Models trained with CLIP scale well and the largest model outperforms the best existing model (a Noisy Student EfficientNet-L2) on the compute efficiency. They also found that CLIP vision transformers are about three times more compute efficient than CLIP ResNets. This allows higher overall performance within the compute budget. These results replicate previous findings where it was reported that vision transformers are more compute efficient than convnets when trained on large datasets[7].

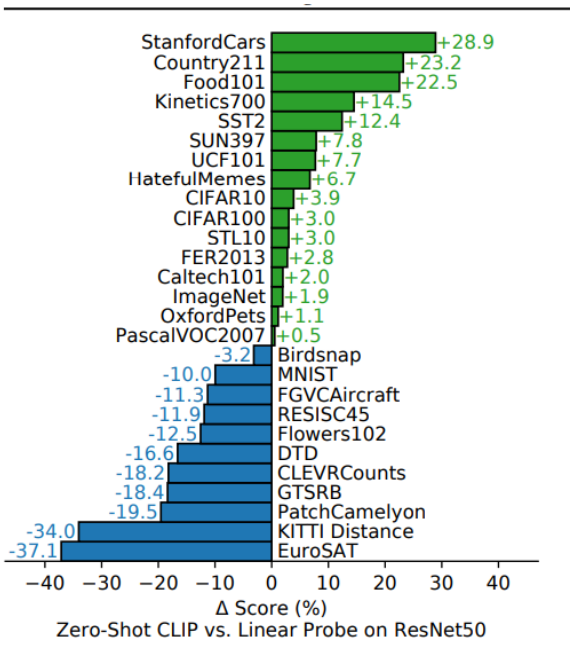

Figure 8 – The Zero Shot CLIP is fully comparable with all fully supervised baselines across all 27 datasets. [7]

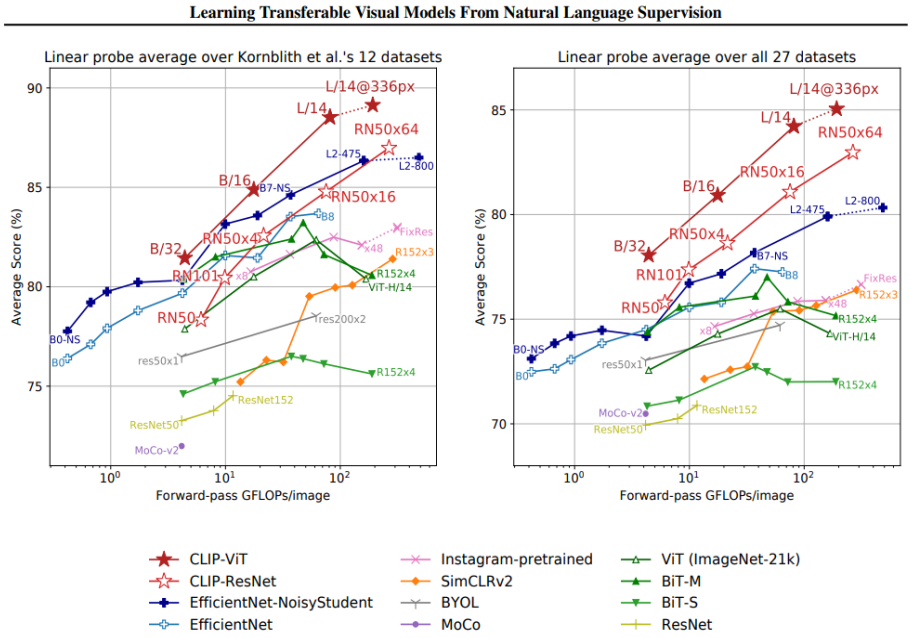

Figure 9 – Linear probe performance of CLIP models in comparison with state-of-the-art computer vision models [7]

Scalability - There is concern that pre-training a large internet dataset would have unintentional overlap with downstream evals. De-duplication analysis was done to detect this. Although there was found to be a small amount of overlap, the overall accuracy was rarely shifted by more than 0.1% with only seven datasets above this threshold. This was similar to previous findings of similar duplicate analysis in previous work on large scale pre-training where there were minimal changes in overall performance. Future research would be on the characterization of the capabilities, faults, and biases in such models[7].

4. Replication of Results

4.1 - Dataset used

The CIFAR-10 dataset is a collection of 60,000 32X32 pixel images divided into two subsets. The main subset consists of 50,000 images referred to as the training set. The secondary subset consists of 10,000 images referred to as the testing set. The images are broken into the 10 following individual classes: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Each class comprising 6,000 images further broken down into 5,000 in the training set and 1,000 in the testing set. CIFAR-10 is one of the most widely used data sets used to train computer vision tasks. We gained access to the dataset using the built-in Torchvision Dataset library.

To test the model out and try to train it on a broader dataset we also explored the caltech 256 dataset. The Caltech 256 dataset is a widely used benchmark dataset in computer vision, consisting of 30,607 images across 256 object categories. The dataset covers a diverse set of object classes, including animals, vehicles, household items, and natural scenes. This rich diversity makes the dataset challenging and suitable for evaluating the performance of computer vision algorithms, particularly in tasks such as object recognition, classification, and scene understanding. The dataset was broken down into smaller sections, where we picked up like 5000 images for training and 1000 images for validation. This dataset was available online and is free to download.

4.2 Model training

The pretrained CLIP model available on GitHub offers a selection of five different models, each varying in architecture size and complexity. For this project, we opted to utilize the "ViT-B32" model, notable for its extensive parameter count of over 155 million, making it capable of capturing rich visual and textual features during training. With our chosen model in hand, we embarked on the training process using the dataset described in the previous section.

The training loop unfolds in several key steps to iteratively refine the model's performance:

- Within each iteration, a batch of images along with their corresponding captions is loaded into memory.
- The data is then fed through the CLIP model, which generates predictions based on the learned associations between textual and visual representations.
- These predictions are compared against the ground truth annotations to compute the loss, which quantifies the disparity between the predicted and actual labels.

- Through backpropagation, the computed loss is propagated back through the network, enabling the model to adjust its parameters in the direction that minimizes the loss, thereby improving its performance.
- This fine-tuning process continues for a predetermined number of epochs, gradually refining the model's understanding of the intricate relationship between our specific set of images and their corresponding textual descriptions.

Fig 10 represents the result that we got during training where we have plotted validation accuracy and loss vs no. of epochs.

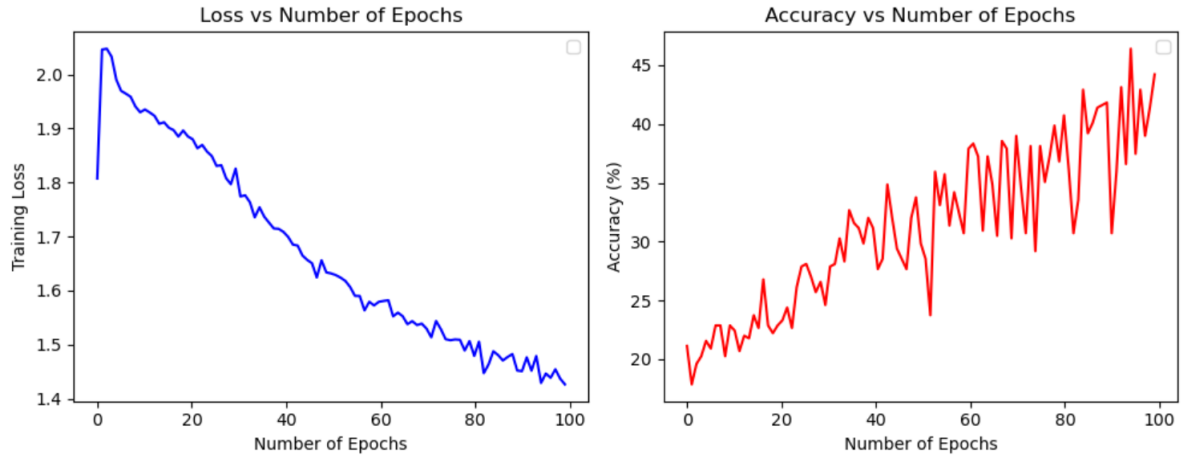

Fig 10. The above plot represents the loss and validation accuracy vs no. of epochs.

As you can see from the plot there is a lot more data and a lot more many epochs required in addition with some tunning to get the loss to be zero and increase the accuracy. Due to equipment limitation we had to run a smaller subset of the data for a lesser number of epochs due to the shear size of the model. The model contains over 155 million parameters and the huge dataset posed a huge challenge for our personal computers.

4.3 Code Listing

```
# Setting Up Everything
! conda install --yes -c pytorch pytorch=1.7.1 torchvision cudatoolkit=11.0
! pip install ftfy regex tqdm
! pip install git+https://github.com/openai/CLIP.git

import os
import torch
import torchvision
from torchvision import datasets, transforms
from torchvision.transforms import ToTensor, Compose, Resize, Normalize
import matplotlib.pyplot as plt
import clip
from transformers import CLIPProcessor, CLIPModel
from PIL import Image
from torch import nn
from tqdm import tqdm
import random
import pandas as pd
import numpy as np
from torch.utils.data import Dataset, DataLoader

# Displaying all available models in Clip
clip.available_models()

# Selecting the Device
device = "cuda:0" if torch.cuda.is_available() else "cpu"

# Load the Model
model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

model, preprocess = clip.load("ViT-B/32", device=device, jit=False)

# Printing the model size and parameters
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)

allImagesPathsandTexts = []



# Path to data directory
dataDir = "/home/cshah/workspaces/Deep Learning/DL Project/data/256_ObjectCategories/"

# Loading the Data and saving it in the right format
for file in os.listdir(dataDir):

    dotInFileName = os.path.basename(file).rfind(".")
    if dotInFileName == 0:
        continue
    name = os.path.basename(file)[dotInFileName+1:]
    filePath = os.path.join(dataDir, os.path.basename(file))

    for fileImages in [fileImages for fileImages in os.listdir(filePath) if fileImages.endswith(".jpg")]:

        allImagesPathsandTexts.append((os.path.join(filePath, fileImages),name))

# Define a custom dataset nad data preprocessing
class imageDataset():
    def __init__(self, imagePathsAndlstTxts):
        # Initialize image paths and corresponding texts

        imagePathList = [pair[0] for pair in imagePathsAndlstTxts]
        txtList = [pair[1] for pair in imagePathsAndlstTxts]
        self.imagePaths = imagePathList
        # Tokenize text using CLIP's tokenizer
        self.title  = clip.tokenize(txtList)

    def __len__(self):
        return len(self.title)

    def __getiten__(self, idx):
```

```
        # Preprocess image using CLIP's preprocessing function
        imageReturn = preprocess(Image.open(self.imagePaths[idx]))
        title = self.title[idx]
        return imageReturn, title


# Parameters
batch_size = 8

# Splitting into training(80%) and Validation Sets(20%)
trainingDataPer = 30;
trainDataLen = int(len(allImagesPathsandTexts) * trainingDataPer / 100)
validationDataPer = 3;
validationDataLen = int(len(allImagesPathsandTexts) * validationDataPer / 100)

# Shuffle the dataset
random.shuffle(allImagesPathsandTexts)

# Setting up the dataset
# Training Dataset
trainingDataset = imageDataset(allImagesPathsandTexts[0:trainDataLen][:])
trainDatasetLoader = DataLoader(trainingDataset, batch_size=batch_size, shuffle=True, num_workers=8)

# Validation Dataset
validationDataset = imageDataset(allImagesPathsandTexts[0:validationDataLen][:])
validationDatasetLoader = DataLoader(validationDataset, batch_size=batch_size, shuffle=False, num_workers=8)

def convert_models_to_fp32(model):
    for p in model.parameters():
        p.data = p.data.float()
        p.grad.data = p.grad.data.float()

# Check if the device is set to CPU
if device == "cpu":
    model.float()  # Convert the model's parameters to float if using CPU

# Prepare the optimizer
optimizer = torch.optim.Adam(
    model.parameters(), lr=5e-5, betas=(0.9, 0.98), eps=1e-6 ,weight_decay=0.2)

# Specify the loss function for images
lossImg = nn.CrossEntropyLoss()

# Specify the loss function for text
lossTxt = nn.CrossEntropyLoss()



# Model Evaluation
def evaluateModel(model, dataLoader, device):
    model.eval()
    correctPredictions = 0
    total = 0
    with torch.no_grad():
        for batch in dataLoader:
            features, labels = batch

            features = features.to(device)
            labels = labels.to(device)

            # Calculate logit and prob
            logitsPerImage, logitsPerText = model(features, labels)

            # Ground Truth
            groundTruth = torch.arange(len(features), dtype=torch.long, device=device)

            # Take argmax of logits
            argMaxImages = torch.argmax(logitsPerImage, dim=1);
            argMaxText = torch.argmax(logitsPerText, dim=0);

            # Calculate accuracy
            compareImages = (argMaxImages == groundTruth);
            compareTexts = (argMaxImages == groundTruth);
            correctPredictions += (torch.sum(compareImages).item() + torch.sum(compareTexts).item())/2

            total += labels.size(0)

    accuracy = 100 * correctPredictions / total
    return accuracy
```

```
    # Model Training
trainingLoss = []
whileTrainAcc = []

num_epochs = 100
for epoch in range(num_epochs):
    totalLossValue = 0
    pbar = tqdm(trainDatasetLoader, total=len(trainDatasetLoader))
    for batch in pbar:
        optimizer.zero_grad()
        images,texts = batch
        images= images.to(device)
        texts = texts.to(device)


        logitsPerImage, logitsPerText = model(images, texts)

        groundTruth = torch.arange(len(images), dtype=torch.long, device=device)
        totalLoss = (lossImg(logitsPerImage, groundTruth) + lossTxt(logitsPerText, groundTruth)) / 2

        # Backward pass and update the model's parameters
        totalLoss.backward()

        # If the device is CPU, directly update the model
        if device == "cpu":
            optimizer.step()
        else:
            # Convert model's parameters to FP32 format, update, and convert back
            convert_models_to_fp32(model)
            optimizer.step()
            clip.model.convert_weights(model)

        # Update the progress bar with the current epoch and loss
        pbar.set_description(f"Epoch {epoch}/{num_epochs}, Loss: {totalLoss.item():.4f}")
        totalLossValue += totalLoss.item()

    # Storing the loss
    trainingLoss.append(totalLossValue/len(pbar))
    # Storing the accuracy
    midAcc = evaluateModel(model, validationDatasetLoader, device)
    whileTrainAcc.append(midAcc)
```

4.4 Classification examples

4.4.1 Principle of Similarity Matrix

In the context of the CLIP model, a similarity matrix serves as a vital tool for quantifying the semantic relationships between textual descriptions and associated images. Represented as a square matrix, it captures pairwise similarities between elements in the dataset, aiding tasks such as image retrieval and clustering. By computing similarity scores between image-caption pairs, the matrix facilitates the exploration of underlying patterns and structures within multimodal data. This matrix plays a crucial role in enhancing our understanding of the intricate connections between textual and visual information, driving advancements in multimodal learning within the CLIP framework.

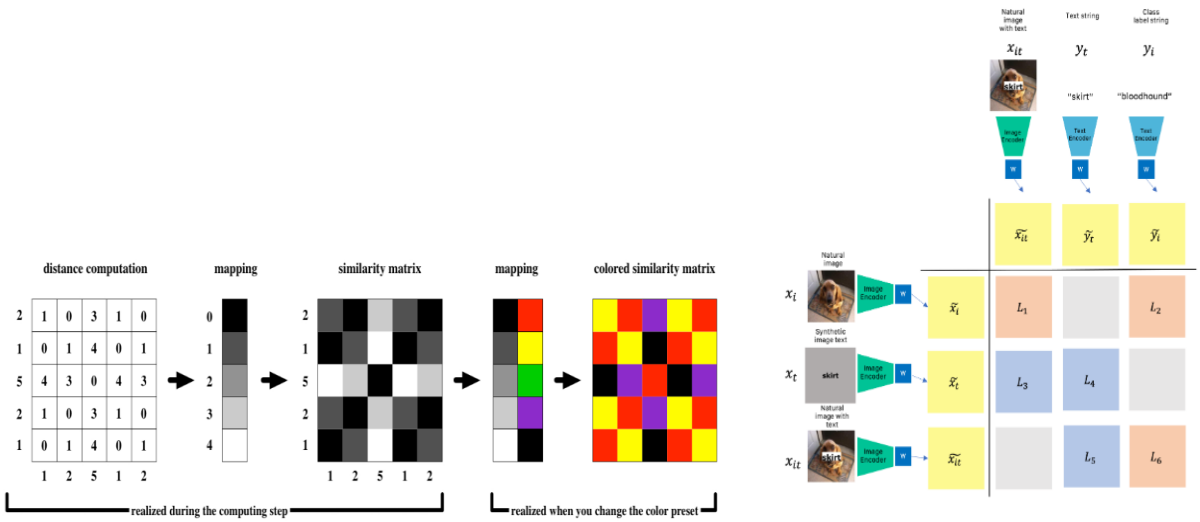

Fig 11. The above figure describes a similarity matrix. And how the clip model is structured to find the similarity between text and images.

4.4.2. Cosine similarity between text and image features

In the illustration below, we present a straightforward classification example for the CLIP model. The image depicts a cosine similarity matrix comparing five images with five distinct captions. The diagonal of the matrix represents the similarity index, showcasing how each image aligns with its corresponding caption. This visualization offers a clear depiction of the model's ability to discern semantic relationships between images and textual descriptions through cosine similarity computations.

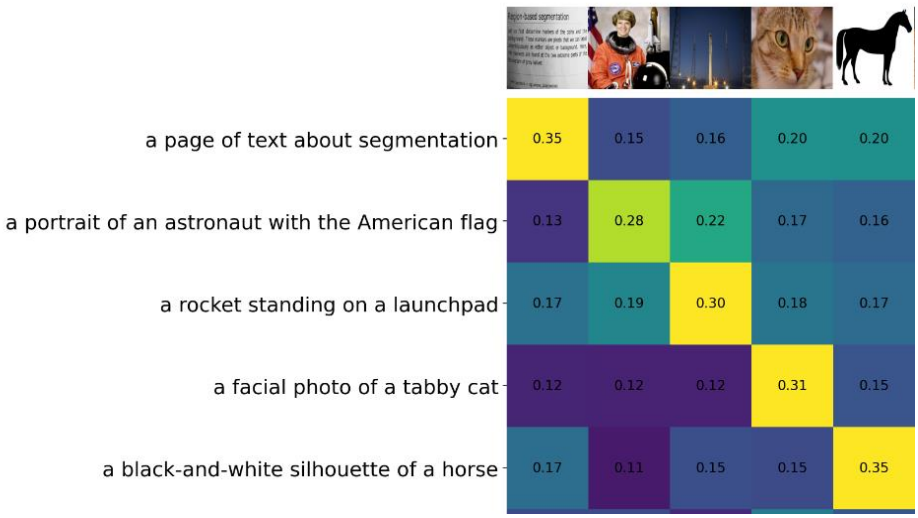

Fig 12. The above image illustrates the working of the CLIP model where we see the highest similarity between images and respective captions.

4.5 Possible real-world applications

4.5.1 Robotics Application

In today's rapidly advancing world, there is a growing demand for robots designed to perform intricate tasks aimed at enhancing our daily lives. Among the recent trends, there's a notable surge in the development of autonomous robots. In this context, we advocate for the integration of vision and language-based models into the field of robotics. These models enable robots to leverage camera input, identify various objects within their view, and generate tailored prompts for user interaction.

For instance, imagine a humanoid robot equipped with vision capabilities observing objects on a table. The robot can then provide users with a range of options, such as 'Pick up the bottle' or 'Dispose of the trashcan.' This approach empowers users to receive prompts that outline the robot's potential actions, allowing us to select the most appropriate response.

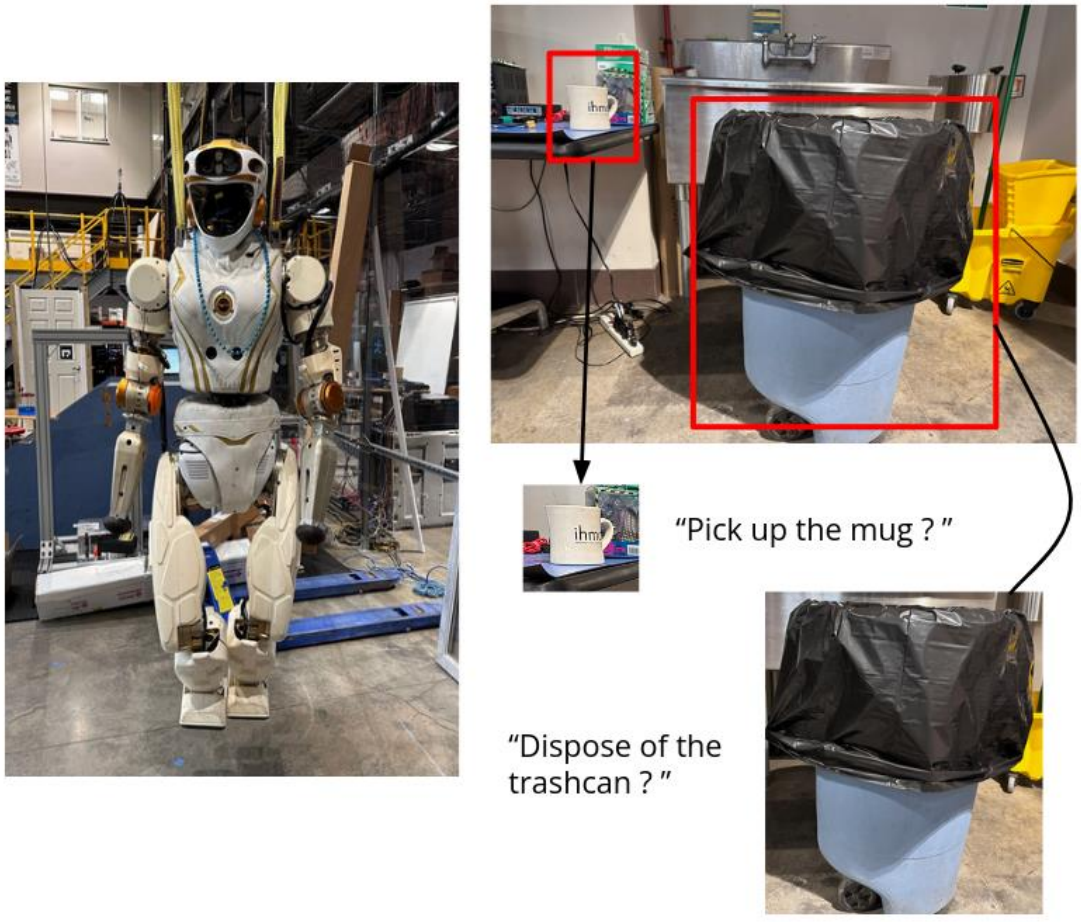

Figure 13. Humanoid Robotics Application. The image on the right shows the prompts that the user will see as possible actions.

4.4.2 Healthcare Application

There are numerous examples throughout the healthcare field where deep learning models such as CLIP may open the door to multimodal data integration which could enhance our understanding of individual health (i.e., personalized medicine) as well as population-level epidemiology (iex: pandemic surveillance). A recent article by Acosta Et. Al. has highlighted 7 opportunities for general multimodal biomedical AI as shown in Figure X [10].

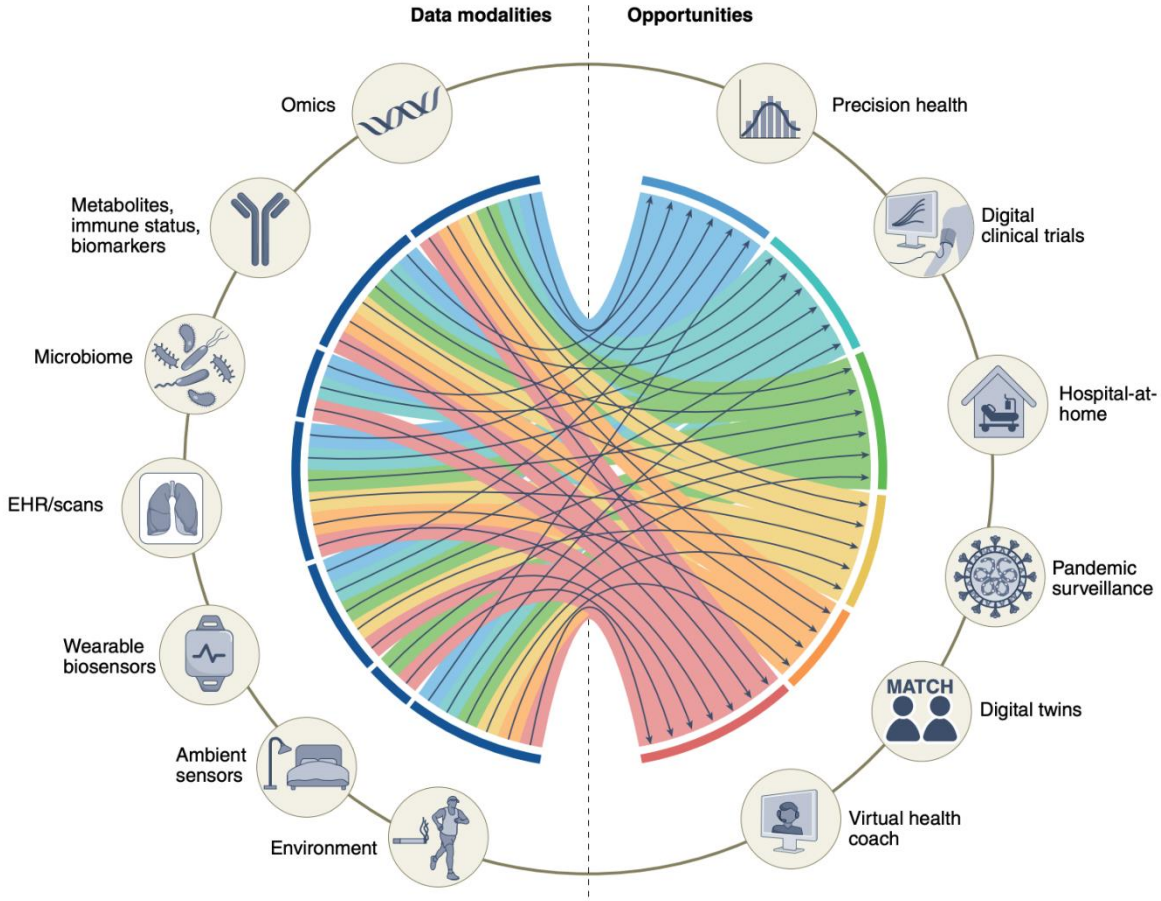

Figure 14. Data modalities and their applications in healthcare [10]. CLIP specifically has applications with EHR data that is uploaded as images, such as radiological scans and environmental monitoring.

Healthcare would specifically benefit from the CLIP model where images and scans are utilized, which is common in specialities such as oncology, neurology, radiology, and orthopedics. Some CNN models are already being implemented to separate “noise’ from the true image that is produced by scans in these specialities. As explained by Acosta Et. Al., models like CLIP could play a crucial role in not just separating noise, but also in extracting and interpreting information from scans and EHR data that are uploaded as images, which is common in Electronic Health Record (EHR) systems [10]. Taking this a step further, once analyzed by CLIP, this data could then be integrated into the existing collection of personalized omics data that is maintained in EHRs, providing a much more robust assessment of individual health.

Environmental monitoring is an area of medicine that is frequently overlooked but could potentially benefit from models like CLIP. Acosta Et. Al. briefly mention that these models could be used in conjunction with ambient sensors to improve remote care systems both at home and in assisted living facilities (ALFs) [10]. The CLIP model could be trained to recognize high-risk situations, such as stroke symptoms (which include physical changes) and accidents such as falls. CLIP may even be able to recognize activities of mental stimulation compared to boredom, which would benefit the goals of ALFs to provide physical as well as mental support to the aging population.

4.4.3 Assistive Device Application

Another real-world application of the CLIP model is to employ the model as an assistive technology to aid people with visual impairments. Historically, the majority of blind people have had to rely on passive devices such as a walking cane to navigate their surroundings using the cane to detect objects in their path. A small number utilize guide dogs to navigate around obstacles. But these methods are not ideal. As an example, a guide dog is unable to interpret street signs. We imagine a CLIP model being combined with a large language model deployed to a device that can recognize & identify objects for the blind and for people with visual impairments. The advantage of the CLIP model is that the output object identification won't be limited to a one-word response. Leveraging the fact that model is trained on captions and textual descriptions. The output response has the potential to be able to provide a vivid sentence describing the object as it relates to its surroundings. This would allow the visually impaired person to essentially have awareness of their environment.

5. Limitations and Conclusion

5.1 Limitations

CLIP is geared towards a contrastive objective rather than a prediction objective. While this approach requires relatively significant less computational resources it does it at a cost to accuracy. A contrastive objective tends to be better at predicting the relationship between two objects, in this case being the text and the image instead of predicting it. As such the model tends to have difficulty distinguishing between objects of a specific class. As an example, being able to correctly label different breeds of dogs, rather having more success in just being able to classify the imagine as a dog. CLIP is a generalized model meaning that it was created to be able to recognize a wide variety of tasks by taking images and relating them to text to improve zero-shot performance. However, that generality underperforms on several specialized tasks such as medical imaging, self-driving tasks, and satellite image classification to name a few [7].

CLIP is a new model having been introduced in 2021 and naturally the CLIP model has yet to reach the zero-shot performance accuracy of the SOTA models available today. The accuracy margin increases even more when compared to specialty models trained for a specific task as an example celebrity identification. Achieving such accuracy would require an additional 1000 times increase in the computation resources which is impossible given current hardware limitations [7]. GPU, memory requirements, and energy consumption being at the forefront of those limitations given the use of today's hardware. The hardware limitations could be one of the reasons why the researchers at OpenAI used a contrastive approach mentioned in the previous paragraph.

5.2 Conclusion

Regardless of these limitations the CLIP model advances image classification by combining images and text to learn. It accomplishes this by its pre-training task which harnesses image-text pairs which are jointly trained on an image encoder in addition to a text encoder. CLIP shows great promise in its efficiency in relation to its performance. By contrast the models presented in the paper were trained on a few datasets, with the largest Vision Transformer taking 12 days to train on only 592 V100 GPUs and the largest Vision Transformer taking 18 days to train on 256 V100 GPUs [7]. Compare that with some of the larger image classification models taking months and some even more than a year to train.

Furthermore, the CLIP model is scalable given the number of captions enabled images that can be found on the internet on websites such as social media sites making it ideal for self-supervised learning. Self-supervised in the sense that the training data can be scrapped directly from the internet instead of relying on people or machines to annotate and label the data. This step alone significantly reduces the pre-training time. Just as it took about a decade for the leap from LeNet to Alexnet to materialize in part due to the hardware limitations, mostly the storage capacity of computers and the advancement of the GPU. We predict that with continuing research and the advancement of the next generation of hardware, the CLIP model will play a pivotal role in the future of classification problems.

6. Statement of Contribution

- **Moise Brutus:** Worked on the limitation and conclusion sections. Was responsible for finding and writing a script for loading the dataset section. Created the Discord server to facilitate group communication/group meetings. Added to the assistive devices possible applications section.
- **Chinmay Sanjay Shah:** Worked on writing the code and replication of the results section to train the model on a subset of data. Additionally worked on writing the Methodology section and the replication of the results part of the report and formatting of the report. Added to the possible robotics real-world applications section.
- **Praya Cheekapara:** Worked on writing the Impact and Critical Assessment(analysis), Replication and Extension of Results. Worked on written portion of this section in the paper and classification examples (4.3) Added to the possible applications section.
- **Joshua J. Cook:** Worked on project outlining, and literature survey and writing the related Work section. Worked on the code for the replication of results. Added to the possible health field real-world applications section.

7. References

1. Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol. Cybernetics* 36, 193–202 (1980). https://doi.org/10.1007/BF00344251
2. Lecun, Yann & Bottou, Leon & Bengio, Y. & Haffner, Patrick. (1998). Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE. 86. 2278 - 2324. 10.1109/5.726791.

3. Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25. 10.1145/3065386.
4. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
5. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).
6. Sabour, S., Frosst, N., & Hinton, G.E. (2017). Dynamic Routing Between Capsules. *ArXiv, abs/1710.09829*.
7. Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... & Sutskever, I. (2021, July). Learning transferable visual models from natural language supervision. In International conference on machine learning (pp. 8748-8763). PMLR.
8. Redmon J., Divvala S., Girshick R., Farhadi A.(2016, May). You Only Look Once: Unified Realtime Object Detection. https://arxiv.org/pdf/1506.02640.pdf
9. Chen, T., Kornblith, S., Swersky, K., Norouzi, M., and Hinton, G. Big self-supervised models are strong semisupervised learners. arXiv preprint arXiv:2006.10029, 2020a.
10. Acosta, J.N., Falcone, G.J., Rajpurkar, P. et al. Multimodal biomedical AI. *Nat Med* 28, 1773–1784 (2022). https://doi.org/10.1038/s41591-022-01981-2