Advanced Computer Vision
Assignment 1

Introduction

Fine-tuning large pre-trained models, such as CLIP (Contrastive Language-Image Pretraining), has gained significant attention for image classification tasks. As it achieves significant accuracy at zero-shot training, it seems suitable for training over small dataset.

Weights on:
https://drive.google.com/drive/folders/1CijX4V65w6rkezSEhkLfMYkBu5Tji4Lh?usp=sharing

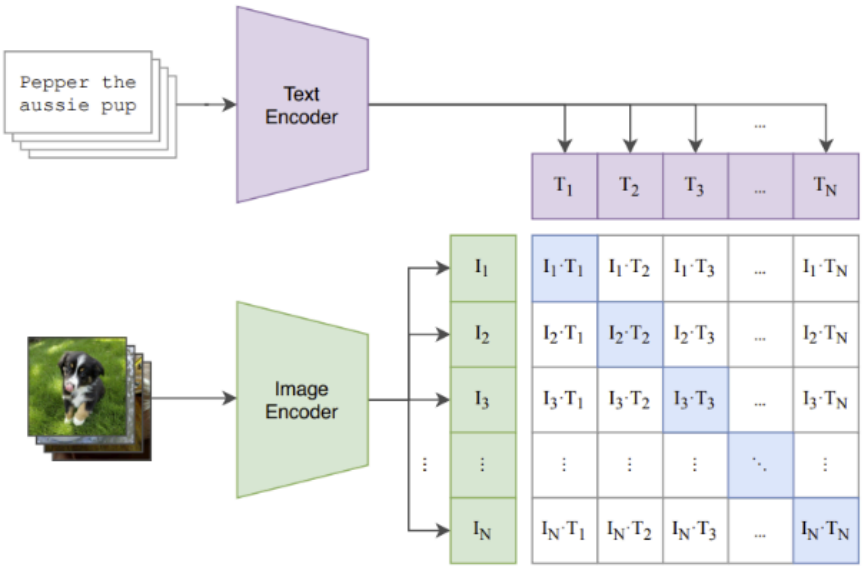


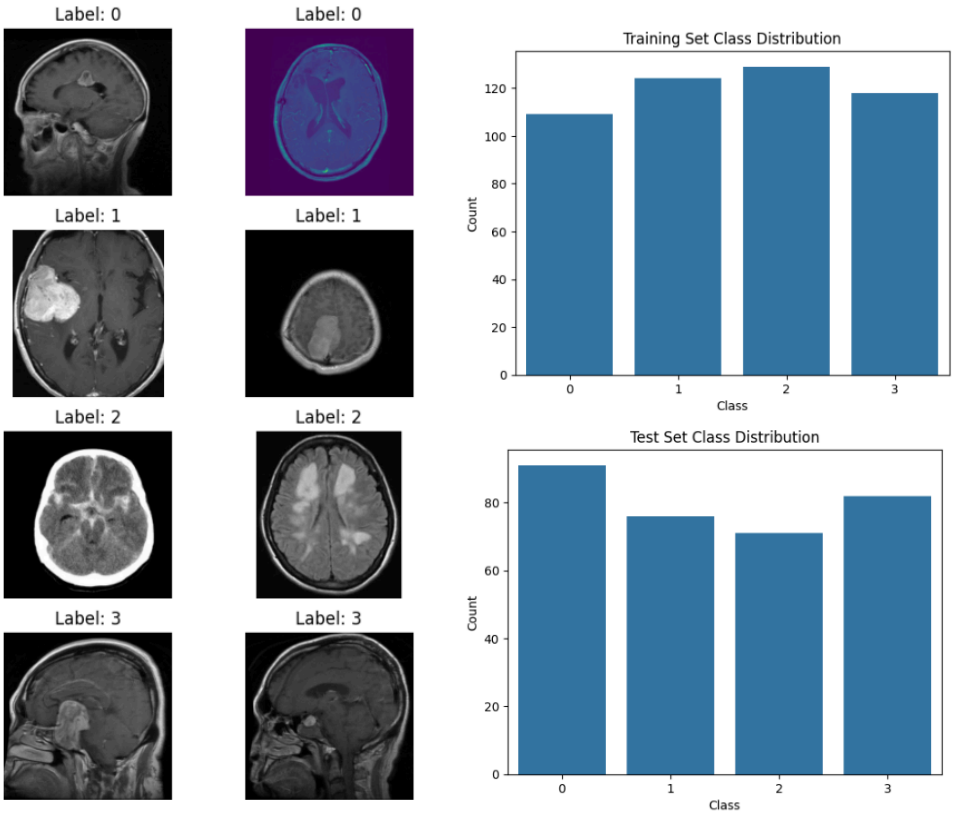
Fig: CLIP Architecture (image from the paper on CLIP)

Traditional methods involving full fine-tuning of these models require significant computational resources. **Parameter Efficient Fine-Tuning (PEFT)** offers an efficient alternative by modifying only a small portion of the model's parameters while keeping the rest frozen. One is **Visual Prompt Tuning (VPT)**, a PEFT approach where learnable prompts are added to the input space of Vision Transformers (ViTs) while the core architecture is frozen. VPT allows for efficient adaptation to new domains with minimal storage overhead. We will implement and evaluate VPT variants (shallow and deep) on the **CLIP ViT-base** model for brain tumor MRI image classification.

This report includes details on the implementation, experiments, and results.

Dataset details

The dataset used consists of 800 human brain MRI images classified into four categories: glioma, meningioma, pituitary tumor, and no tumor. The data is split into training (480 images) and test (320 images) sets. Further dataset analyses are shown in Figure 2. Classes are glioma, meningioma, no tumor and pituitary, respectively.

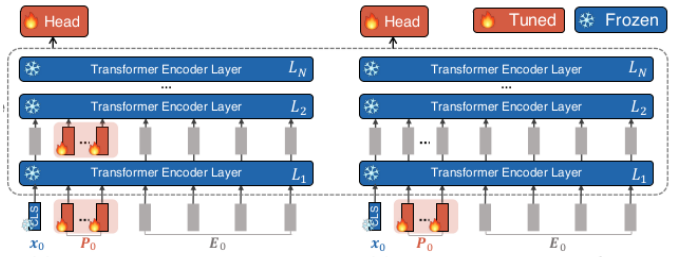


(a) (b)

Fig 2: Overview of dataset can be taken by (a) images present in dataset with labels (b) dataset distribution in 4 classes for training and testing dataset

VPT Training Overview

Reference: https://github.com/KMnP/vpt/blob/main/src/models/vit_prompt/vit.py



(a) Visual-Prompt Tuning: Deep (b) Visual-Prompt Tuning: Shallow

Models

1. Zero-Shot

We Perform zero-shot inference with the frozen CLIP model using an appropriate text prompt. But without any training, the model's performance was very bad. It classifies all test images to first class every time, it doesn't even depend on the text prompt.

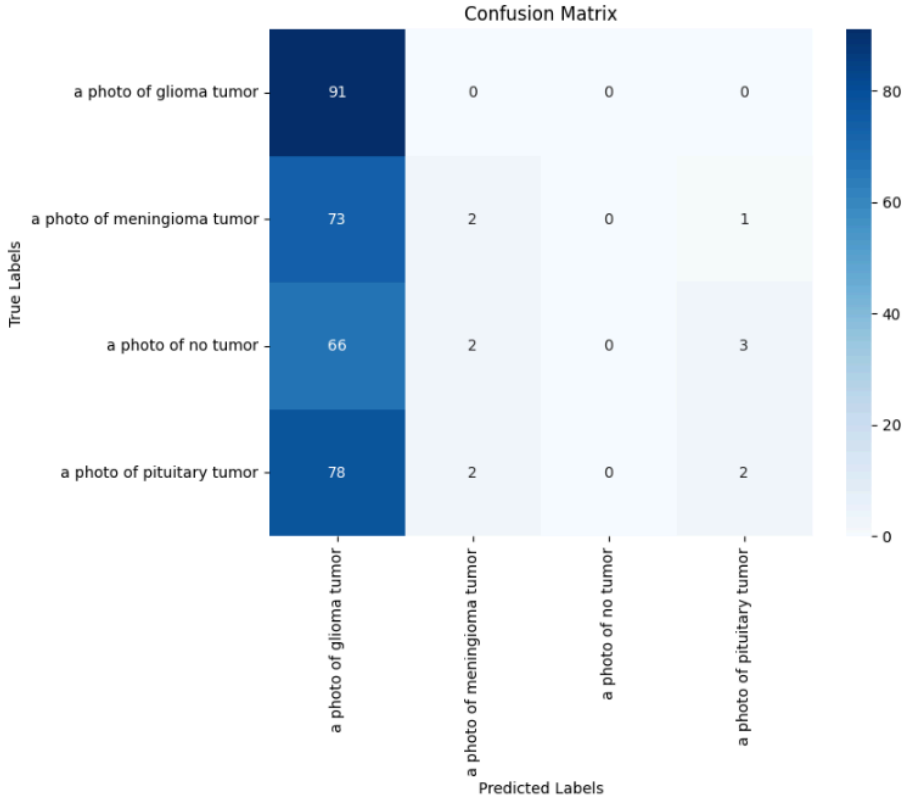


Figure 3: Confusion Matrix for zero-shot results. Accuracy achieved for it is `29.69%`

2. ViT w /Linear Training

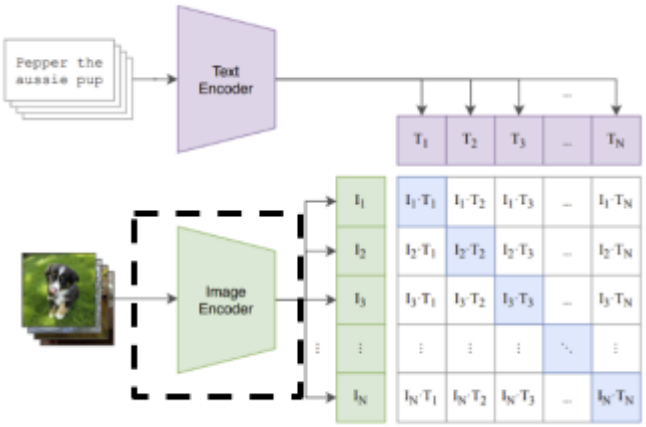


Figure 4: We utlized this Image encoder part and classified the image between 4 classes utilising linear head with it.

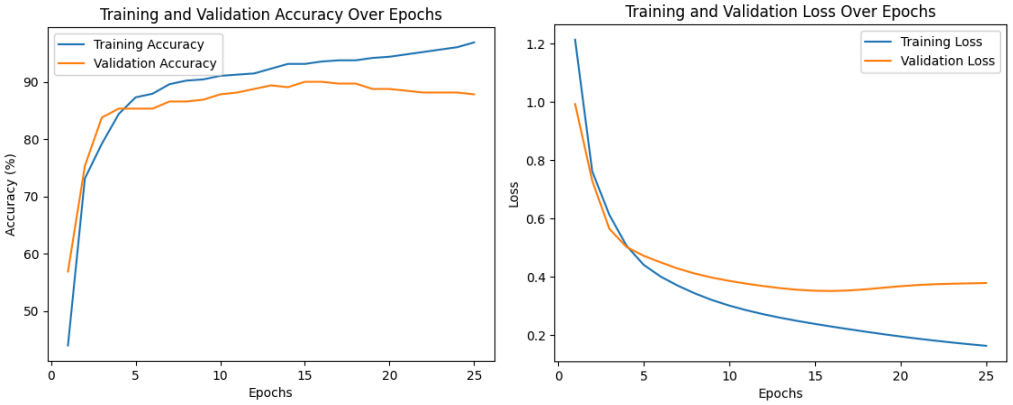


Figure 5: The accuracy and training curve for training ViT with linear head achieved the best test accuracy of 90% and training accuracy of 93.12%. After that, the model starts overfitting.

3. ViT w /Shallow VPT

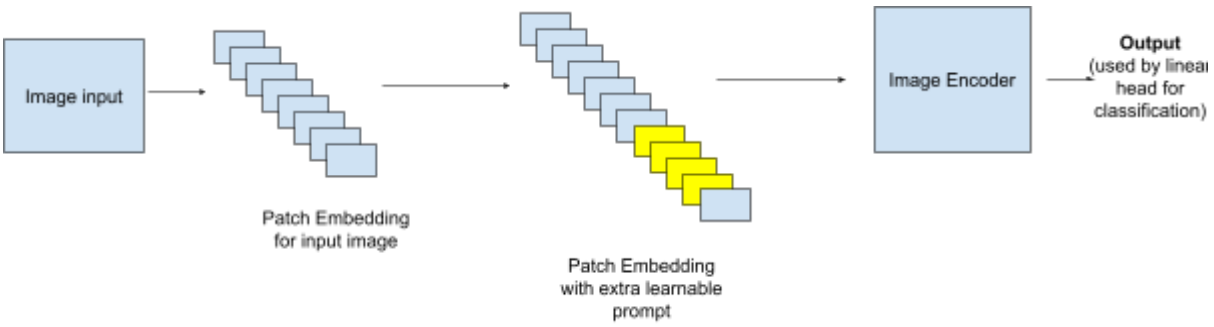
PromptedTransformer: It takes input and introduces learnable parameters after [CLS] token and before image patches embeddings. Code is as bellow:

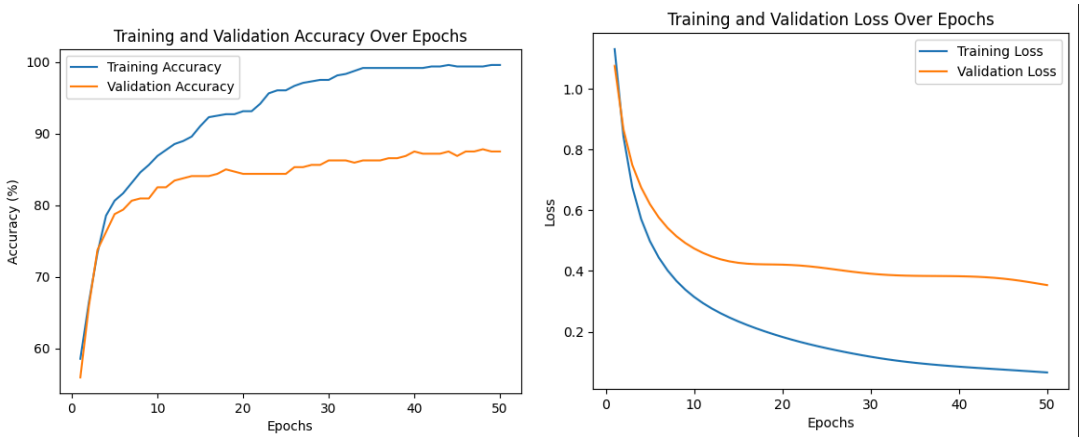
```python
# Adding learnnable prompt embeddings to the Vision Permuter Transformer
class PromptedTransformer(nn.Module):
    def __init__(self, prompt_config, config, img_size, vis):
        super(PromptedTransformer, self).__init__()

        self.prompt_config = prompt_config
        self.vit_config = config
        self.vis = vis
        img_size = (img_size, img_size)

        num_tokens = self.prompt_config["NUM_TOKENS"]
        self.num_tokens = num_tokens
        self.prompt_dropout = Dropout(self.prompt_config["DROPOUT"])

        prompt_dim = config.hidden_size if self.prompt_config["PROJECT"] == -1 else self.prompt_config["PROJECT"]
        self.prompt_proj = nn.Linear(prompt_dim, config.hidden_size) if self.prompt_config["PROJECT"] > -1 else nn.Identity()

        if self.prompt_config["INITIATION"] == "random":
            val = math.sqrt(6. / float(3 * img_size[0] * img_size[1] + prompt_dim))
            self.prompt_embeddings = nn.Parameter(torch.zeros(1, num_tokens, prompt_dim))
            nn.init.uniform_(self.prompt_embeddings.data, -val, val)

            if self.prompt_config.get("DEEP", False):
                total_d_layer = config.num_hidden_layers - 1
                self.deep_prompt_embeddings = nn.Parameter(torch.zeros(total_d_layer, num_tokens, prompt_dim))
                nn.init.uniform_(self.deep_prompt_embeddings.data, -val, val)
        else:
            raise ValueError("Only random initiation is supported")

    def incorporate_prompt(self, x):
        B = x.shape[0]
        x = torch.cat((
            x[:, :1, :],
            self.prompt_dropout(self.prompt_proj(self.prompt_embeddings).expand(B, -1, -1)),
            x[:, 1:, :]
        ), dim=1)
        return x

    def forward(self, x):
        embedding_output = self.incorporate_prompt(x)
        encoded = self.prompt_proj(embedding_output)
        return encoded
```

Later this is utilised by our modified CLIP model, named CLIP_VPT_Classifier, as shown in the code below:

```python
class CLIP_VPT_Classifier(nn.Module):
    def __init__(self, clip_model_name: str, num_classes: int, prompt_cfg):
        super(CLIP_VPT_Classifier, self).__init__()

        self.clip_model = CLIPModel.from_pretrained(clip_model_name)

        for param in self.clip_model.parameters():
            param.requires_grad = False

        self.prompted_vit = PromptedTransformer(prompt_cfg, self.clip_model.vision_model.config, img_size=224, vis=False)

        self.head = nn.Linear(self.clip_model.vision_model.config.hidden_size, num_classes)
        nn.init.kaiming_normal_(self.head.weight, a=0, mode='fan_out')

    def forward(self, pixel_values):
        x = self.clip_model.vision_model(pixel_values).last_hidden_state
        encoded = self.prompted_vit(x)

        cls_token_output = encoded[:, 0]
        logits = self.head(cls_token_output)

        return logits
```

The complete pipeline is as follows:





4. ViT w /Deep VPT

Deep implementations are similar to what is discussed in Shallow. The main difference is prompt embedding is introduced for every transformer's input. In Shallow, we were utilising the PromptedTransformer model only to add prompt embeddings but for deep, we added a prompt to every transformer block. The modified code is as follows:

```
def incorporate_prompt(self, x, layer_idx):
    B = x.shape[0]

    if self.deep_prompt_embeddings:
        prompts = self.deep_prompt_embeddings[layer_idx].expand(B, -1, -1)
    else:
        prompts = self.prompt_embeddings.expand(B, -1, -1)

    x = torch.cat((
        x[:, :1, :],  # CLS token
        self.prompt_dropout(self.prompt_proj(prompts)),  # inserting Prompts in between CLS and Patches
        x[:, 1:, :]  # Patches
    ), dim=1)
    return x

def forward(self, pixel_values, attention_mask=None, causal_attention_mask=None):
    hidden_states = self.vision_model.embeddings(pixel_values)
    B = hidden_states.shape[0]

    for layer_idx, layer in enumerate(self.vision_model.encoder.layers):
        hidden_states = self.incorporate_prompt(hidden_states, layer_idx)
        hidden_states = layer(hidden_states, attention_mask=attention_mask, causal_attention_mask=causal_attention_mask)[0]

    return hidden_states
```

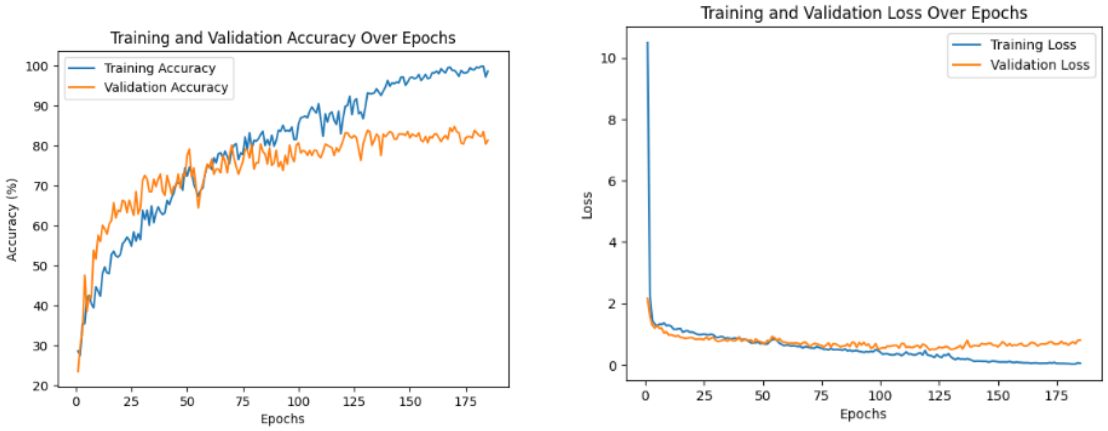
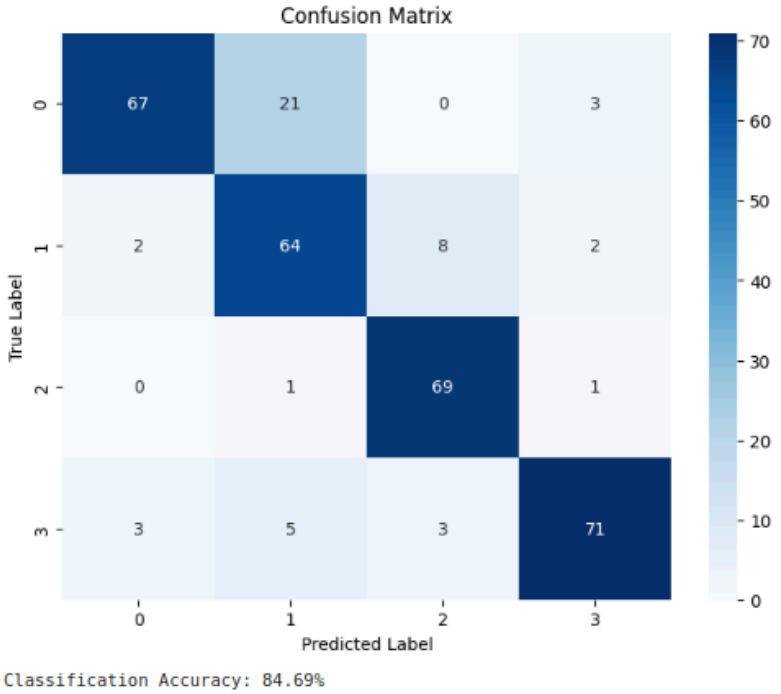
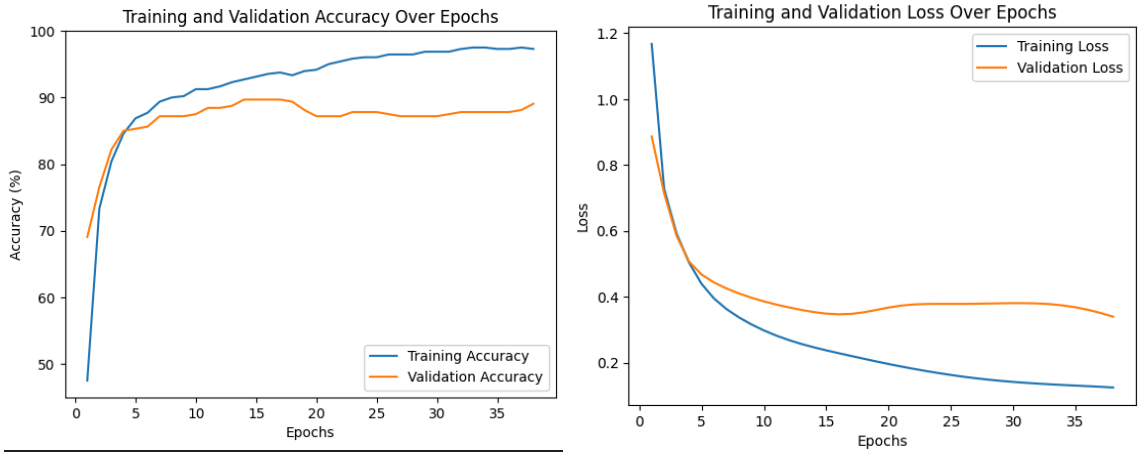


Figure : Accuracy and Loss curves



Classification Accuracy: 84.69%

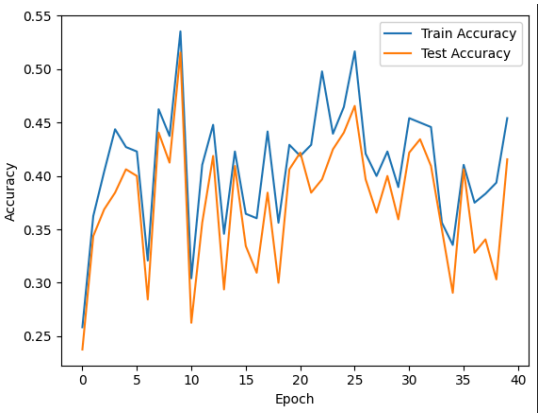
5. Full Fine Tunning



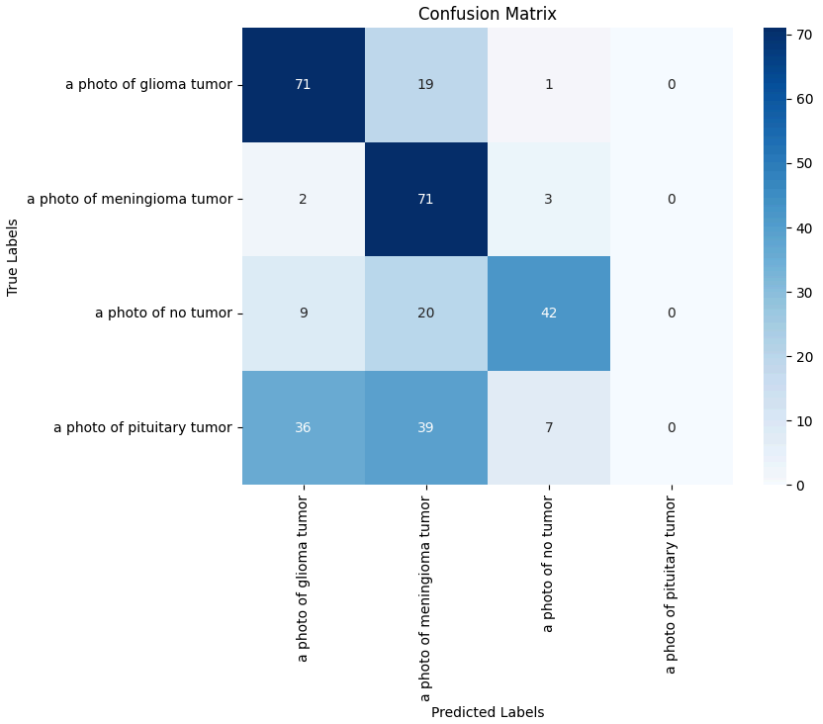
Experiment: Finetuning CLIP

```
for epoch in range(n_epochs):
    pbar = tqdm(train_dataloader, total = len(train_dataloader))
    loss=0
    model.train()
    for batch in pbar:
        optimizer.zero_grad()
        images, texts = batch
        images, texts = images.to(device), texts.to(device)

        logits_per_image, logits_per_text = model(images, texts)

        ground_truth = torch.arange(len(images), dtype=torch.long, device=device)
        total_loss = (loss_img(logits_per_image, ground_truth) + loss_txt(logits_per_text, ground_truth))/2

        total_loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm = 5)
        optimizer.step()
```

Clip finetuning both image_encoder and text_encoder without VPT using contrastive learning as shown in this figure. The training was not stable as you can see by following figure:



Test Accuracy achieved: 57.50%



Results

Model	Test Accuracy	Train Accuracy	Comments
Zero-shot	29.69	-----	No training so number of updated parameter=0, training time is also 0.
Linear Head	90.00	93.12	Takes 15 epochs to reach the best testing accuracy and starts to overfit after that.
Swallow	87.81	99.38	Saturation achieved near 50 epochs. The time taken to train per epoch was also greater than Linear Head. and convergence speed is also low compared to the linear head. Training parameters = parameters in linear head + 5*prompt_size
Deep	84.69	98.75	Saturation achieved near 50 epochs. The time taken to train per epoch was also greater than Shallow, and the convergence speed is also low as compared to Shallow. Training parameters = parameters in linear head+ 5* prompt_size*number_of_transformer_layers
full	89.69	93.65	The model starts overfitting similarly to the linear head after 17 epochs. Training time is greater than the linear head, and time taken

			per epoch is also greater than the linear head. But compared to shallow and deep, it takes less time to train. The number of trained parameters is larger than all. Because it is fully fine-tuned.
CLIP Exp	57.50	56.00	Trained using contrastive losses. Unstable training.