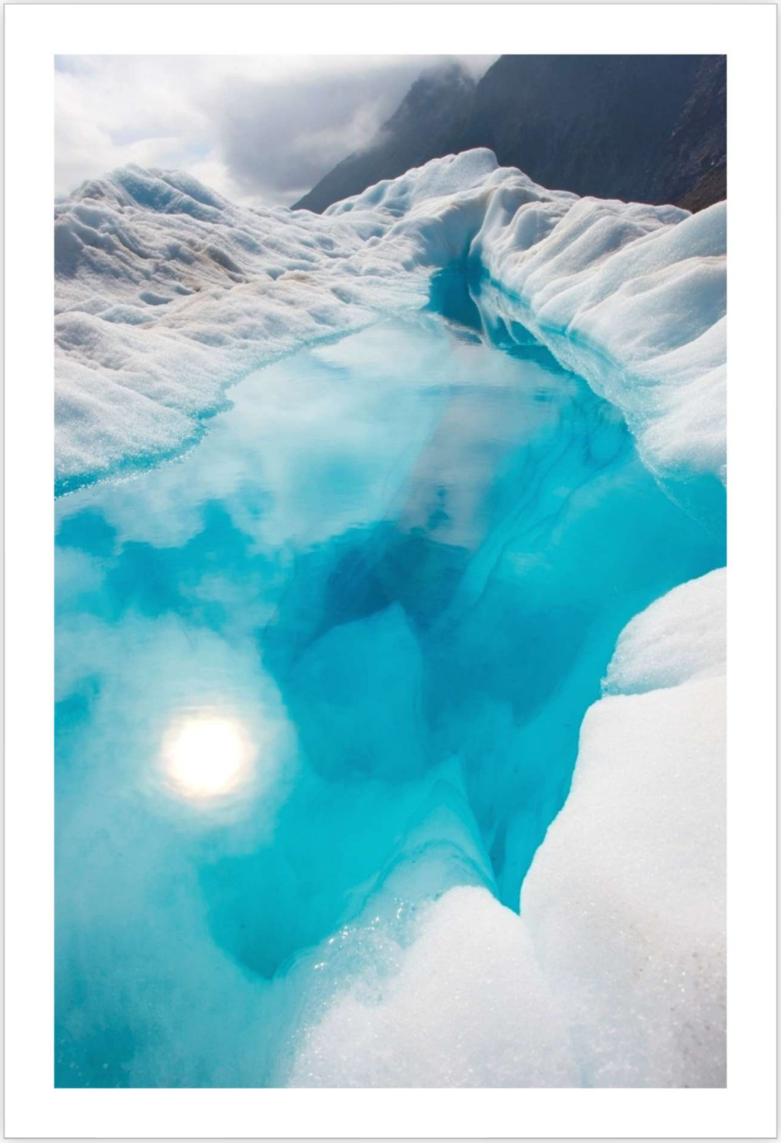

ViT for Image Classification

Implementing and Evaluating Vision Transformer for Image Classification

Ankit 23BAI1379 | Amlan Sarkar 23BAI1247 |Kanishq Tiwari 23BAI1150

BCSE306L

March 13, 2025

Preparation

Check CUDA compatibility

Install Cudnn, CUDA dev toolkit, Nvidia drivers

Set up a new Conda env

Install relevant packages

- Torch (PyTorch)
- Torchvision
- Transformers
- SkLearn
- Numpy

Create a new notebook in the created env

Start the jupyter server and kernel

GITHUB: HTTPS://GITHUB.COM/AMLAN131/DA2/TREE/MAIN

CHECK IF GPU IS AVAILABLE

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)
```

Code

```
import os
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader
from transformers import ViTForImageClassification, ViTFeatureExtractor,
TrainingArguments, Trainer
from PIL import Image
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

-------- Step 1: Load Dataset --------

```
class BreastCancerDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.image_paths = []
        self.labels = []
```

Read all images and assign labels based on filenames

```
        for img_name in os.listdir(root_dir):
```

```
            if img_name.endswith(".png"):
                self.image_paths.append(os.path.join(root_dir, img_name))
                self.labels.append(0 if "SOB_B" in img_name else 1)  # 0=Benign, 1=Malignant

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

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        image = Image.open(img_path).convert("RGB")
        label = self.labels[idx]

        if self.transform:
            image = self.transform(image)

        return {"pixel_values": image, "labels": torch.tensor(label)}
```

Define Size

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])
```

Load datasets

```
train_dataset = BreastCancerDataset(r"D:\01 STUDY MATERIAL\ai project\mkfold\combined fold 1\train_40x", transform=transform)
test_dataset = BreastCancerDataset(r"D:\01 STUDY MATERIAL\ai project\mkfold\combined fold 1\test_40x", transform=transform)
```

DataLoaders

```
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
```

Load ViT feature extractor

```
feature_extractor = ViTFeatureExtractor.from_pretrained("google/vit-base-patch16-224-in21k")
```

-------- Step 2: Define ViT Model --------

```
model = ViTForImageClassification.from_pretrained(
    "google/vit-base-patch16-224-in21k",
    num_labels=2,  # Binary classification
    id2label={0: "Benign", 1: "Malignant"},
    label2id={"Benign": 0, "Malignant": 1},
)
model.to(device)
```

-------- Step 3: Define Evaluation Metrics --------

```
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
```

```
    accuracy = accuracy_score(labels, predictions)
    precision = precision_score(labels, predictions)
    recall = recall_score(labels, predictions)
    f1 = f1_score(labels, predictions)

    return {"accuracy": accuracy, "precision": precision, "recall": recall, "f1": f1}


-------- Step 4: Define Training Arguments --------
training_args = TrainingArguments(
    output_dir="./vit_cancer_detection",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=5,
    save_total_limit=2,
    logging_dir="./logs",
    report_to="none",
    load_best_model_at_end=True,
    push_to_hub=False,
    fp16=True,
)


-------- Step 5: Train Model --------
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    tokenizer=feature_extractor,
    compute_metrics=compute_metrics,
)

trainer.train()


-------- Step 6: Save Model --------
trainer.save_model("vit_binary_model_40x")
print("Model saved successfully!")


-------- Step 7: Load Model and Evaluate --------
model = ViTForImageClassification.from_pretrained("vit_binary_model_40X")
model.to(device)

trainer.model = model
metrics = trainer.evaluate()
print("Final Evaluation Metrics:", metrics)


-------- Step 8: Make Predictions on Random Image --------
import random
import os
```

```
def predict_random_image(test_dir):
    random_class = random.choice(['benign', 'malignant'])
    random_image = random.choice(os.listdir(os.path.join(test_dir, random_class)))
    image_path = os.path.join(test_dir, random_class, random_image)
    image = Image.open(image_path).convert("RGB")
    inputs = feature_extractor(image, return_tensors="pt").to(device)

    with torch.no_grad():
        logits = model(**inputs).logits

    predicted_class = torch.argmax(logits, dim=-1).item()
    return "Benign" if predicted_class == 0 else "Malignant"
```

Example prediction

```
test_dir = 'D:/01 STUDY MATERIAL/ai project/Breast-Splitted/test'
print("Prediction for random test image:", predict_random_image(test_dir))
```

Adjusments

Change input dataset parameter to get results and metrics at different magnification levels

Output

Using device: cuda

C:\Users\AMLAN\anaconda3\envs\tf2\lib\site-packages\transformers\models\vit\feature_extraction_vit.py:28: FutureWarning: The class ViTFeatureExtractor is deprecated and will be removed in version 5 of Transformers. Please use ViTImageProcessor instead.

warnings.warn(

Some weights of ViTForImageClassification were not initialized from the model checkpoint at google/vit-base-patch16-224-in21k and are newly initialized: ['classifier.bias', 'classifier.weight']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

C:\Users\AMLAN\anaconda3\envs\tf2\lib\site-packages\transformers\training_args.py:1594: FutureWarning: `evaluation_strategy` is deprecated and will be removed in version 4.46 of 😭 Transformers. Use `eval_strategy` instead

warnings.warn(

C:\Users\AMLAN\AppData\Local\Temp\ipykernel_5208\2259406091.py:97: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead.

trainer = Trainer(

[395/395 05:42, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	No log	0.470654	0.829530	0.794165	1.000000	0.885276
2	No log	0.268625	0.920805	0.938900	0.940816	0.939857
3	No log	0.296465	0.922148	0.933735	0.948980	0.941296
4	No log	0.314285	0.926174	0.909605	0.985714	0.946131
5	No log	0.328231	0.928859	0.909944	0.989796	0.948192

Model saved successfully!

[47/47 00:22]

Final Evaluation Metrics: {

'eval_loss': 0.26865100860595703,

'eval_accuracy': 0.9208053691275168,

'eval_precision': 0.9389002036659878,

'eval_recall': 0.9408163265306122,

'eval_f1': 0.9398572884811417,

'eval_runtime': 24.2592,

'eval_samples_per_second': 30.71,

'eval_steps_per_second': 1.937, 'epoch': 5.0}

Prediction for random test image: **Benign**