

Transformer Models

Lecture 05

Prepared by

Dr. Hager Saleh

```
import warnings
warnings.simplefilter("ignore")
import os
import numpy as np
import pandas as pd
import torch
import torchvision
from torchvision import datasets, transforms as T
from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision import models
import timm
from timm.loss import LabelSmoothingCrossEntropy
import matplotlib.pyplot as plt
from tqdm import tqdm
import time
import copy
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score, precision_score, recall_score, f1_score,
cohen_kappa_score, matthews_corrcoef
from sklearn.metrics import ConfusionMatrixDisplay
import torch.optim as optim
from torchvision.models import vit_b_32
```

Transformer

A Transformer is a deep learning model introduced in the paper "Attention is All You Need" (Vaswani et al., 2017).

Key Components of the Transformer Model

- Self-Attention Mechanism
The Transformer attends to all input tokens simultaneously.
The Scaled Dot-Product Attention mechanism allows the model to focus on the most relevant parts of the input.
- Multi-Head Attention (MHA)
The model applies multiple self-attention mechanisms in parallel.
Each head learns different aspects of the relationships between tokens.

Types of Transformer models

1. Vision Transformer (ViT)
 - Introduced by Google in 2020.
 - Uses patch embedding: Instead of convolution, it splits an image into fixed-size patches (e.g., 16×16 pixels) and treats them as tokens.
 - Applies self-attention mechanisms over these tokens to capture global dependencies.
2. Swin Transformer (Shifted Window Transformer)
 - Introduced by Microsoft in 2021.
 - Improves ViT by using a hierarchical structure and shifted window attention to reduce computational complexity.
 - Instead of using global self-attention on all patches, it processes local windows and shifts them between layers to capture both local and global information.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Data augmentation

```
def get_classes(data_dir):  
    return datasets.ImageFolder(data_dir).classes  
  
def get_data_loaders(data_dir, batch_size, train=False):  
    transform = T.Compose([  
        T.RandomHorizontalFlip() if train else T.Lambda(lambda x: x),  
        T.RandomVerticalFlip() if train else T.Lambda(lambda x: x),  
        T.Resize(256),  
        T.CenterCrop(224),  
        T.ToTensor(),  
        T.Normalize(timm.data.IMAGENET_DEFAULT_MEAN,  
                    timm.data.IMAGENET_DEFAULT_STD),  
        T.RandomErasing(p=0.1, value='random') if train else
```

```

T.Lambda(lambda x: x)
    ])

    dataset_path = os.path.join(data_dir, "train" if train else
"test")
    dataset = datasets.ImageFolder(dataset_path, transform=transform)
    dataloader = DataLoader(dataset, batch_size=batch_size,
shuffle=True, num_workers=0)
    return dataloader, len(dataset)

data_dir = "new_directories"
train_loader, train_data_len = get_data_loaders(data_dir, 124,
train=True)
test_loader, test_data_len = get_data_loaders(data_dir, 32,
train=False)

classes = get_classes(os.path.join(data_dir, "train"))

def train_model(model, criterion, optimizer, scheduler, dataloader,
dataset_size, num_epochs=10):
    since = time.time()
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print("-" * 10)
        model.train()
        running_loss = 0.0
        running_corrects = 0

        for inputs, labels in tqdm(dataloader):
            inputs = inputs.to(device)
            labels = labels.to(device)
            optimizer.zero_grad()

            with torch.set_grad_enabled(True):
                outputs = model(inputs)
                _, preds = torch.max(outputs, 1)
                loss = criterion(outputs, labels)
                loss.backward()
                optimizer.step()

            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)

        scheduler.step()
        epoch_loss = running_loss / dataset_size
        epoch_acc = running_corrects.double() / dataset_size
        print(f'Train Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')

```

```

        if epoch_acc > best_acc:
            best_acc = epoch_acc
            best_model_wts = copy.deepcopy(model.state_dict())
        print()

    time_elapsed = time.time() - since
    print(f'Training complete in {time_elapsed // 60:.0f}m
{time_elapsed % 60:.0f}s')
    print(f'Best Train Acc: {best_acc:.4f}')

    model.load_state_dict(best_model_wts)
    return model

def evaluate_model(model, dataloader):
    y_pred, y_true = [], []
    with torch.no_grad():
        for inputs, labels in tqdm(dataloader):
            inputs = inputs.to(device)
            preds = torch.argmax(model(inputs), axis=1).cpu().numpy()
            y_pred.extend(preds)
            y_true.extend(labels.numpy())

    return y_true, y_pred

```

ViT model

```

model = vit_b_32(pretrained=True)
n_inputs = model.heads.head.in_features
model.heads.head = nn.Sequential(
    nn.Linear(n_inputs, 512),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(512, len(classes))
)
model = model.to(device)
print(model.heads.head )

criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)

scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.97)

# Train the model
ViT_model = train_model(model, criterion, optimizer, scheduler,
train_loader, train_data_len, num_epochs=5)

```

```
Sequential(  
  (0): Linear(in_features=768, out_features=512, bias=True)  
  (1): ReLU()  
  (2): Dropout(p=0.3, inplace=False)  
  (3): Linear(in_features=512, out_features=4, bias=True)  
)
```

Epoch 0/4

100%|██████████| 24/24 [00:27<00:00, 1.16s/it]

Train Loss: 1.4057 Acc: 0.2897

Epoch 1/4

100%|██████████| 24/24 [00:24<00:00, 1.01s/it]

Train Loss: 1.3590 Acc: 0.2948

Epoch 2/4

100%|██████████| 24/24 [00:23<00:00, 1.01it/s]

Train Loss: 1.3153 Acc: 0.3517

Epoch 3/4

100%|██████████| 24/24 [00:23<00:00, 1.04it/s]

Train Loss: 1.1752 Acc: 0.4300

Epoch 4/4

100%|██████████| 24/24 [00:24<00:00, 1.01s/it]

Train Loss: 1.0548 Acc: 0.5046

Training complete in 2m 3s

Best Train Acc: 0.5046

Evaluate best model

```
y_true, y_pred= evaluate_model(ViT_model, test_loader)
```

```
print("Classification Report:")
```

```
print(classification_report(y_true, y_pred))
```

100%|██████████| 27/27 [00:07<00:00, 3.53it/s]

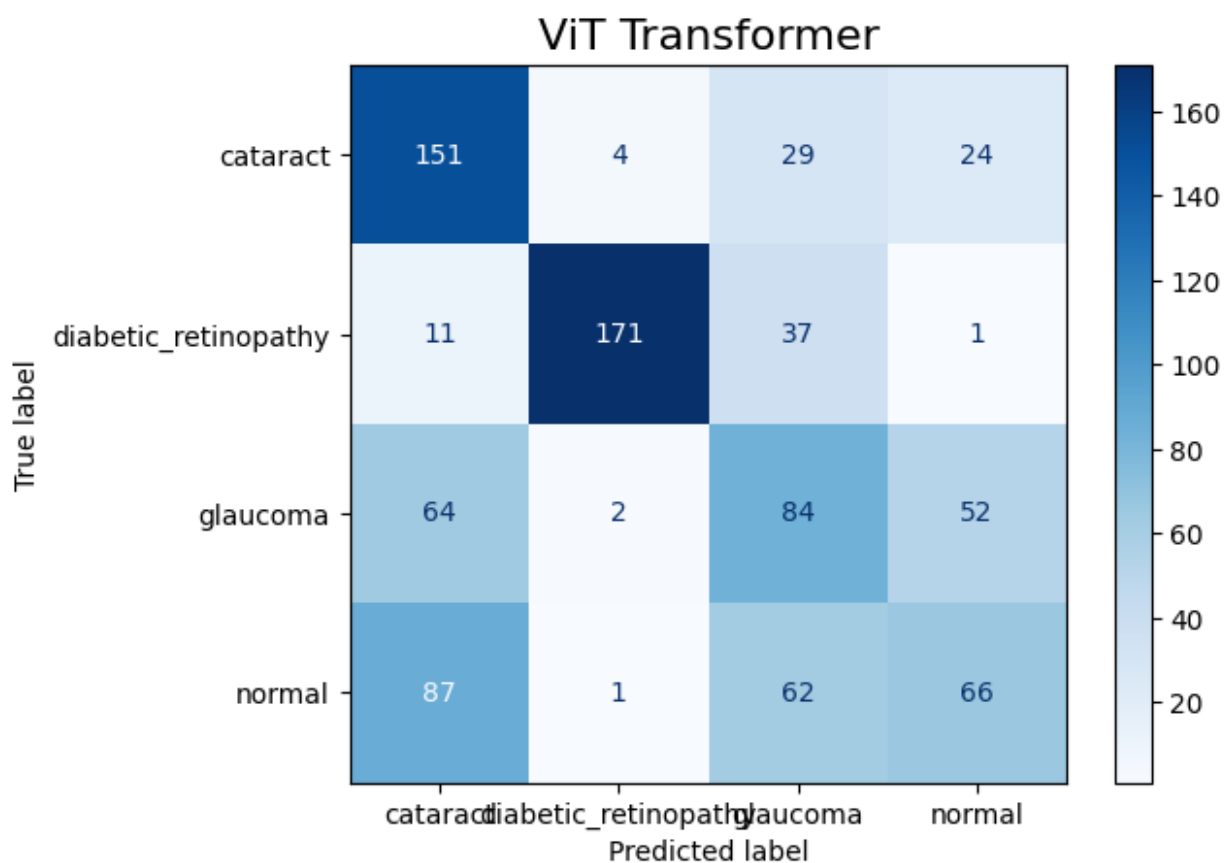
Classification Report:

	precision	recall	f1-score	support
0	0.48	0.73	0.58	208
1	0.96	0.78	0.86	220
2	0.40	0.42	0.41	202
3	0.46	0.31	0.37	216
accuracy			0.56	846
macro avg	0.58	0.56	0.55	846
weighted avg	0.58	0.56	0.56	846

```

disp =
ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_true,
y_pred), display_labels=classes)
disp.plot(cmap="Blues", values_format='')
plt.title("ViT Transformer", fontsize=16)
plt.show()

```



```

results = pd.DataFrame([{'Accuracy': round(accuracy_score(y_true,
y_pred) * 100, 3), 'Precision': round(precision_score(y_true, y_pred,
average='weighted') * 100, 3), 'Recall': round(recall_score(y_true,
y_pred, average='weighted') * 100, 3), 'F1 Score':
round(f1_score(y_true, y_pred, average='weighted') * 100, 3), 'Kappa
Statistic': round(cohen_kappa_score(y_true, y_pred) * 100, 3), 'MCC':
round(matthews_corrcoef(y_true, y_pred) * 100, 3) }])
results

```

	Accuracy	Precision	Recall	F1 Score	Kappa Statistic	MCC
0	55.792	58.088	55.792	55.674	41.16	41.874

Swin Model

```

HUB_URL = "SharanSMenon/swin-transformer-hub:main"
MODEL_NAME = "swin_tiny_patch4_window7_224"
model = torch.hub.load(HUB_URL, MODEL_NAME, pretrained=True)
model = model.to(device)

criterion = LabelSmoothingCrossEntropy().to(device)
optimizer = optim.AdamW(model.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.97)

Swin_model = train_model(model, criterion, optimizer, scheduler,
train_loader, train_data_len, num_epochs=5)

Using cache found in /home/hager/.cache/torch/hub/SharanSMenon_swin-
transformer-hub_main

Epoch 0/4
-----
100%|██████████| 24/24 [04:23<00:00, 10.98s/it]
Train Loss: 2.5674 Acc: 0.5557

Epoch 1/4
-----
100%|██████████| 24/24 [04:05<00:00, 10.23s/it]
Train Loss: 1.4831 Acc: 0.8102

Epoch 2/4
-----
100%|██████████| 24/24 [04:06<00:00, 10.27s/it]
Train Loss: 1.3594 Acc: 0.8645

```

Epoch 3/4

100%|██████████| 24/24 [04:17<00:00, 10.72s/it]

Train Loss: 1.2708 Acc: 0.9027

Epoch 4/4

100%|██████████| 24/24 [04:17<00:00, 10.72s/it]

Train Loss: 1.2313 Acc: 0.9187

Training complete in 21m 10s

Best Train Acc: 0.9187

Evaluate best model

y_true, y_pred= evaluate_model(Swin_model, test_loader)

print("Classification Report:")

print(classification_report(y_true, y_pred))

100%|██████████| 27/27 [00:10<00:00, 2.52it/s]

Classification Report:

	precision	recall	f1-score	support
0	0.91	0.94	0.92	208
1	0.96	0.98	0.97	220
2	0.92	0.77	0.84	202
3	0.84	0.92	0.88	216
accuracy			0.91	846
macro avg	0.91	0.90	0.90	846
weighted avg	0.91	0.91	0.91	846

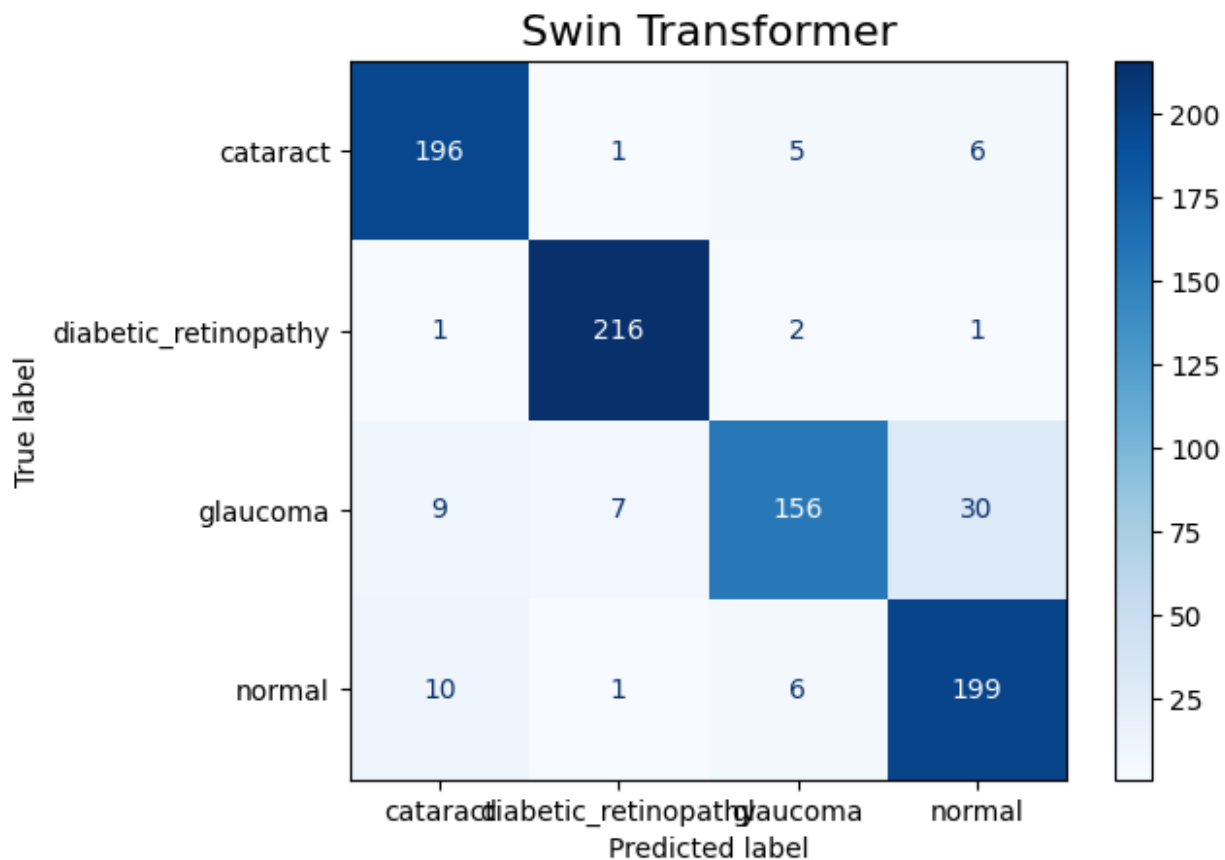
disp =

ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_true,
y_pred), display_labels=classes)

disp.plot(cmap="Blues", values_format='')

plt.title("Swin Transformer", fontsize=16)

plt.show()



```

results = pd.DataFrame([{'Accuracy': round(accuracy_score(y_true,
y_pred) * 100, 3), 'Precision': round(precision_score(y_true, y_pred,
average='weighted') * 100, 3), 'Recall': round(recall_score(y_true,
y_pred, average='weighted') * 100, 3), 'F1 Score':
round(f1_score(y_true, y_pred, average='weighted') * 100, 3), 'Kappa
Statistic': round(cohen_kappa_score(y_true, y_pred) * 100, 3), 'MCC':
round(matthews_corrcoef(y_true, y_pred) * 100, 3) }])
results

```

	Accuracy	Precision	Recall	F1 Score	Kappa Statistic	MCC
0	90.662	90.844	90.662	90.537	87.535	87.664