inference-capsule-vision

October 23, 2024

1 Directory

[7]:
```
import os
for dirname, _, filenames in os.walk('/kaggle/input/capsule-vision-2024-models/
 ↪pytorch/updated/1'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/capsule-
vision-2024-models/pytorch/updated/1/SwinTransformer_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/ResNeXt_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/WideResNet_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/ResNet_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/ViT_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/RegNet_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/BEiT_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/TwinsSVT_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/SEResNet50_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/MobileNetV3_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/MNASNet_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/CaiT_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/DeiT_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/DenseNet_best.pth
/kaggle/input/capsule-
vision-2024-models/pytorch/updated/1/EfficientFormer_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/InceptionV3_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/ConvNeXt_best.pth
/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/EfficientNet_best.pth
```

2 Imports

[8]:
```
# Imports
import os
import json
import random

from typing import Dict, List, Tuple
```

```
from datetime import datetime
from pathlib import Path
from logging import getLogger, Logger, INFO, StreamHandler, FileHandler,␣
 ↪Formatter
from tqdm.auto import tqdm

import pandas as pd
import numpy as np
from PIL import Image

import timm

import torch
from torch import nn, optim
from torch.optim.lr_scheduler import CosineAnnealingLR
from torch.utils.data import Dataset, DataLoader
from torch.utils.data.sampler import WeightedRandomSampler
from torch.nn import functional as F

import torchvision
from torchvision import transforms, models

import torchmetrics
print("Libraries Imported Successfuly!\n\n")
```

```
Libraries Imported Successfuly!
```

3 Models

[9]:
```
import torch
import torch.nn as nn
import timm
import warnings

warnings.filterwarnings('ignore')

# 1. ViT (Vision Transformer)
def model_vit(pretrained=True, num_classes=10):
    model = timm.create_model('vit_base_patch16_224', pretrained=pretrained)
    model.head = nn.Linear(model.head.in_features, num_classes)
    return model

# 2. Swin Transformer
def model_swin(pretrained=True, num_classes=10):
```

```
    model = timm.create_model('swin_base_patch4_window7_224',␣
 ↪pretrained=pretrained)
    model.head.fc = nn.Linear(model.head.fc.in_features, num_classes)
    return model

# 3. DeiT (Data-efficient Image Transformers)
def model_deit(pretrained=True, num_classes=10):
    model = timm.create_model('deit_base_patch16_224', pretrained=pretrained)
    model.head = nn.Linear(model.head.in_features, num_classes)
    return model

# 4. ConvNeXt
def model_convnext(pretrained=True, num_classes=10):
    model = timm.create_model('convnext_base', pretrained=pretrained)
    model.head.fc = nn.Linear(model.head.fc.in_features, num_classes)
    return model

# 5. EfficientNet
def model_efficientnet(pretrained=True, num_classes=10):
    model = timm.create_model('tf_efficientnetv2_s_in21ft1k',␣
 ↪pretrained=pretrained)
    model.classifier = nn.Linear(model.classifier.in_features, num_classes)
    return model

# 6. ResNet
def model_resnet(pretrained=True, num_classes=10):
    model = timm.create_model('resnet50', pretrained=pretrained)
    model.fc = nn.Linear(model.fc.in_features, num_classes)
    return model

# 7. MobileNetV3
def model_mobilenetv3(pretrained=True, num_classes=10):
    model = timm.create_model('mobilenetv3_large_100', pretrained=pretrained)
    model.classifier = nn.Linear(model.classifier.in_features, num_classes)
    return model

# 8. RegNet
def model_regnet(pretrained=True, num_classes=10):
    model = timm.create_model('regnetx_032', pretrained=pretrained)
    model.head.fc = nn.Linear(model.head.fc.in_features, num_classes)
    return model

# 9. DenseNet
def model_densenet(pretrained=True, num_classes=10):
    model = timm.create_model('densenet121', pretrained=pretrained)
    model.classifier = nn.Linear(model.classifier.in_features, num_classes)
    return model
```

```python
# 10. Inception v3
def model_inception_v3(pretrained=True, num_classes=10):
    model = timm.create_model('inception_v3', pretrained=pretrained)
    model.fc = nn.Linear(model.fc.in_features, num_classes)
    return model

# 11. ResNeXt
def model_resnext(pretrained=True, num_classes=10):
    model = timm.create_model('resnext50_32x4d', pretrained=pretrained)
    model.fc = nn.Linear(model.fc.in_features, num_classes)
    return model

# 12. Wide ResNet
def model_wide_resnet(pretrained=True, num_classes=10):
    model = timm.create_model('wide_resnet50_2', pretrained=pretrained)
    model.fc = nn.Linear(model.fc.in_features, num_classes)
    return model

# 13. MNASNet
def model_mnasnet(pretrained=True, num_classes=10):
    model = timm.create_model('mnasnet_100', pretrained=pretrained)
    model.classifier = nn.Linear(model.classifier.in_features, num_classes)
    return model

# 14. SEResNet50 (Replaces SqueezeNet)
def model_seresnet50(pretrained=True, num_classes=10):
    model = timm.create_model('seresnet50', pretrained=pretrained)
    model.fc = nn.Linear(model.fc.in_features, num_classes)
    return model

# 15. BEiT (Bidirectional Encoder Representation from Image Transformers)
def model_beit(pretrained=True, num_classes=10):
    model = timm.create_model('beit_base_patch16_224', pretrained=pretrained)
    model.head = nn.Linear(model.head.in_features, num_classes)
    return model

# 16. CaiT (Class-Attention in Image Transformers)
def model_cait(pretrained=True, num_classes=10):
    model = timm.create_model('cait_s24_224', pretrained=pretrained)
    model.head = nn.Linear(model.head.in_features, num_classes)
    return model

# 17. Twins-SVT (Spatially Separable Vision Transformer)
def model_twins_svt(pretrained=True, num_classes=10):
    model = timm.create_model('twins_svt_base', pretrained=pretrained)
    model.head = nn.Linear(model.head.in_features, num_classes)
```

```
    return model

# 18. EfficientFormer
def model_efficientformer(pretrained=True, num_classes=10):
    model = timm.create_model('efficientformerv2_s0', pretrained=pretrained,␣
 ↪num_classes=num_classes)

    # Ensure the classifier is set to the correct number of classes
    if hasattr(model, 'head'):
        in_features = model.head.in_features
        model.head = nn.Linear(in_features, num_classes)
    elif hasattr(model, 'classifier'):
        in_features = model.classifier.in_features
        model.classifier = nn.Linear(in_features, num_classes)
    else:
        raise AttributeError("Model doesn't have a 'head' or 'classifier'␣
 ↪attribute")

    return model

# if __name__ == "__main__":
#     # Test the models with random input
#     input_tensor = torch.randn(1, 3, 224, 224)  # Batch size of 1, 3 color␣
 ↪channels, 224x224 image size

#     models_to_test = [
#         model_vit, model_swin, model_deit, model_convnext, model_efficientnet,
#         model_resnet, model_mobilenetv3, model_regnet, model_densenet,␣
 ↪model_inception_v3,
#         model_resnext, model_wide_resnet, model_mnasnet,
#         model_seresnet50,
#         model_beit, model_cait,
#         model_twins_svt, model_pnasnet,
#         model_xcit
#     ]

#     expected_shape = (1, 10)  # Expected output shape

#     for model_func in models_to_test:
#         model = model_func()
#         output = model(input_tensor)
#         if output.shape != expected_shape:
#             print(f"Model {model_func.__name__} failed with output shape:␣
 ↪{output.shape}")
#             break
#         print(f"{model_func.__name__} Output Shape:", output.shape)
```

4 Utils and Initial Setup

[10]:
```
# Save predictions to Excel
def save_predictions_to_excel(image_paths, y_pred: torch.Tensor, output_path:␣
↪str):
    class_columns = ['Angioectasia', 'Bleeding', 'Erosion', 'Erythema', 'Foreign␣
↪Body', 'Lymphangiectasia', 'Normal', 'Polyp', 'Ulcer', 'Worms']

    # Convert logits to class predictions
    y_pred_classes = y_pred.argmax(dim=1).cpu().numpy()

    # Create a DataFrame to store image paths, predicted class, and prediction␣
↪probabilities
    df = pd.DataFrame({
        'image_path': image_paths,
        'predicted_class': [class_columns[i] for i in y_pred_classes],
        **{col: y_pred[:, i].cpu().numpy() for i, col in␣
↪enumerate(class_columns)}
    })

    # Save to Excel file
    df.to_excel(output_path, index=False)
    print(f"Predictions saved to {output_path}")
```

[11]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Model paths
model_paths = [
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/
↪SwinTransformer_best.pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/ResNeXt_best.
↪pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/WideResNet_best.
↪pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/ResNet_best.pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/ViT_best.pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/RegNet_best.pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/BEiT_best.pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/TwinsSVT_best.
↪pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/SEResNet50_best.
↪pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/MobileNetV3_best.
↪pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/MNASNet_best.
↪pth',
```

```
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/CaiT_best.pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/DeiT_best.pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/DenseNet_best.
↪pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/
↪EfficientFormer_best.pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/InceptionV3_best.
↪pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/ConvNeXt_best.
↪pth',
    '/kaggle/input/capsule-vision-2024-models/pytorch/updated/1/
↪EfficientNet_best.pth'
]

# Model classes
model_classes = [
    model_swin,          # Swin Transformer
    model_resnext,       # ResNeXt
    model_wide_resnet,   # Wide ResNet
    model_resnet,        # ResNet
    model_vit,           # Vision Transformer
    model_regnet,        # RegNet
    model_beit,          # BEiT
    model_twins_svt,     # Twins-SVT
    model_seresnet50,    # SEResNet50
    model_mobilenetv3,   # MobileNetV3
    model_mnasnet,       # MNASNet
    model_cait,          # CaiT
    model_deit,          # DeiT
    model_densenet,      # DenseNet
    model_efficientformer, # EfficientFormer
    model_inception_v3, # Inception v3
    model_convnext,      # ConvNeXt
    model_efficientnet   # EfficientNet
]
```

[12]:
```
import os
import torch
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms
from PIL import Image
import pandas as pd
from tqdm import tqdm

# Function to load PyTorch model
def load_model(model_class, model_path, device):
    model = model_class()
```

```
    model.load_state_dict(torch.load(model_path, map_location=device))
    model.eval()
    model.to(device)
    return model

# Preprocess the image as per PyTorch model requirements
def load_and_preprocess_image(full_path, target_size=(224, 224)):
    img = Image.open(full_path).convert('RGB')
    transform = transforms.Compose([
        transforms.Resize(target_size),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
 ↪225])
    ])
    return transform(img)

# Custom Dataset for loading test data
class TestDataset(Dataset):
    def __init__(self, image_paths, image_size=(224, 224)):
        self.image_paths = image_paths
        self.image_size = image_size

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

    def __getitem__(self, idx):
        image_path = self.image_paths[idx]
        img = load_and_preprocess_image(image_path, self.image_size)
        return img, image_path

# Function to load test data
def load_test_data(test_dir, image_size=(224, 224)):
    full_image_paths = [os.path.join(test_dir, fname) for fname in os.
 ↪listdir(test_dir) if fname.lower().endswith(('jpg'))]
    dataset = TestDataset(full_image_paths, image_size)
    dataloader = DataLoader(dataset, batch_size=32, shuffle=False, num_workers=4)

    # Return the dataset loader and image file names (without full path)
    image_paths = [os.path.basename(path) for path in full_image_paths]
    return dataloader, image_paths

# Ensemble Inference on the test set
def ensemble_test_inference(models, dataloader, device):
    all_predictions = []
    all_image_paths = []

    # Set all models to eval mode once before inference
```

```
    for model_idx, model in enumerate(models):
        model.eval()
        print(f"Model {model_idx + 1} set to eval mode.")

    with torch.no_grad():
        # Initialize the progress bar with total number of batches
        with tqdm(total=len(dataloader), desc="Ensemble Inference",␣
↪unit="batch", leave=True, miniters=1, smoothing=0) as pbar:
            for batch_idx, (X, image_paths) in enumerate(dataloader):

                # Move data to device (GPU or CPU)
                X = X.to(device)

                # Calculate predictions for all models and average them
                ensemble_preds = torch.stack([
                    torch.softmax(model(X), dim=1)  # Softmax for probabilities
                    for model_idx, model in enumerate(models)
                ]).mean(dim=0)  # Average over the models

                # Append predictions to list (move them to CPU for easier␣
↪processing)
                all_predictions.append(ensemble_preds.cpu())

                # Extract and save only the file names (not full paths)
                all_image_paths.extend([os.path.basename(path) for path in␣
↪image_paths])

                # Update the progress bar for each batch
                pbar.update(1)

    # Concatenate all predictions
    predictions = torch.cat(all_predictions, dim=0)

    return predictions, all_image_paths


def main():
    # Load ensemble models
    models = [load_model(cls, path, device) for cls, path in zip(model_classes,␣
↪model_paths)]

    # Directory containing test images
#     test_path = "../capsule-vision-2024/data/Testing set/Images"
    test_path = "/kaggle/input/capsule-vision-2020-test/Testing set/Images"  #␣
↪Update with actual path
    dataloader, image_paths = load_test_data(test_path)
```

```
    # Run ensemble inference on the test set
    predictions, image_paths = ensemble_test_inference(models, dataloader,␣
 ↪device)

    # output_test_predictions = "../capsule-vision-2024/reports/test_excel.xlsx"
    output_test_predictions = "/kaggle/working/Seq2Cure.xlsx"
    save_predictions_to_excel(image_paths, predictions, output_test_predictions)

    print(f"Predictions saved to {output_test_predictions}")

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

```
Model 1 set to eval mode.
Model 2 set to eval mode.
Model 3 set to eval mode.
Model 4 set to eval mode.
Model 5 set to eval mode.
Model 6 set to eval mode.
Model 7 set to eval mode.
Model 8 set to eval mode.
Model 9 set to eval mode.
Model 10 set to eval mode.
Model 11 set to eval mode.
Model 12 set to eval mode.
Model 13 set to eval mode.
Model 14 set to eval mode.
Model 15 set to eval mode.
Model 16 set to eval mode.
Model 17 set to eval mode.
Model 18 set to eval mode.

Ensemble Inference: 100%|| 138/138 [04:25<00:00,  1.92s/batch]

Predictions saved to /kaggle/working/Seq2Cure.xlsx
Predictions saved to /kaggle/working/Seq2Cure.xlsx
```

[ ]:

[ ]:

[ ]: