data_gathering_transfer_learning

April 1, 2022

[1]:

```
import torch
import torchvision
import torchvision.transforms as transforms
import torchvision.models as models
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import time
from itertools import count
import natsort
import datetime
import numpy as np
import os
import math
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler
import albumentations as A
from albumentations.pytorch import ToTensorV2
import cv2
import glob
import numpy
import random
import pandas as pd
import tqdm
torch.manual_seed(10)
```

[1]:

```
<torch._C.Generator at 0x21fe907b150>
```

[47]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
import time
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,␣
 ↪classification_report, roc_curve, auc
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
import seaborn as sns
```

[3]:

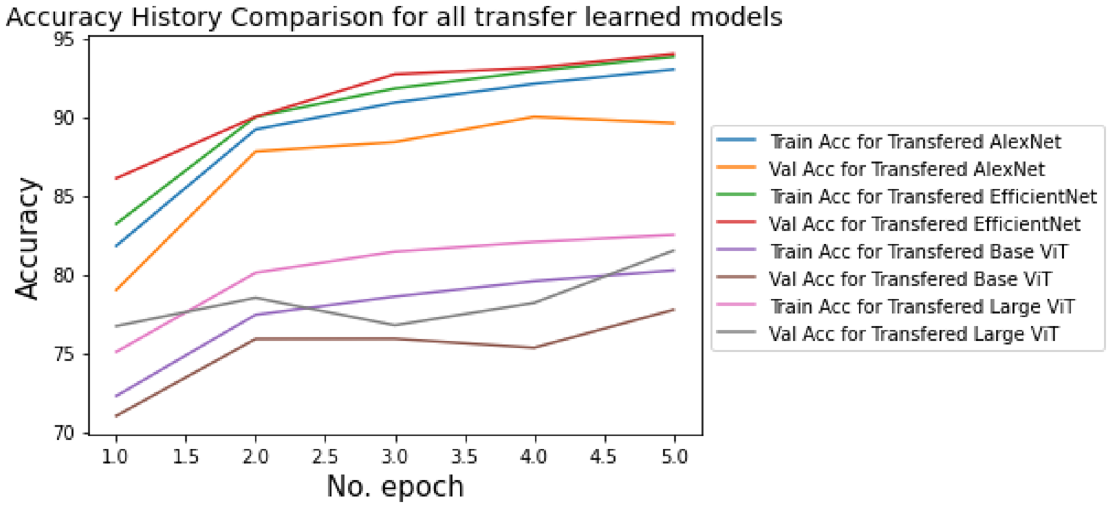
```
print(f"Is CUDA supported by this system? {torch.cuda.is_available()}")
print(f"CUDA version: {torch.version.cuda}")
# Storing ID of current CUDA device
cuda_id = torch.cuda.current_device()
print(f"ID of current CUDA device: {torch.cuda.current_device()}")
print(f"Name of current CUDA device: {torch.cuda.get_device_name(cuda_id)}")

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
Is CUDA supported by this system? True
CUDA version: 11.3
ID of current CUDA device: 0
Name of current CUDA device: NVIDIA GeForce RTX 2070 Super
cuda:0
```

[4]:

```
# Plot history: Accuracy
history_alex_train = [81.8, 89.2, 90.9, 92.1, 93.0]
history_alex_val = [79.0, 87.8, 88.4, 90.0, 89.6]

history_eff_train = [83.2, 90.0, 91.8, 92.89, 93.82]
history_eff_val = [86.1, 90.0, 92.7, 93.1, 94.0]

history_vit_b_train = [72.26, 77.42, 78.58, 79.56, 80.24]
history_vit_b_val = [71.0,75.88, 75.89, 75.32, 77.74]

history_vit_l_train = [75.06, 80.09, 81.43, 82.05, 82.50]
history_vit_l_val = [76.7, 78.5, 76.75, 78.18, 81.498]

epochs = [1, 2, 3, 4, 5]

plt.plot(epochs, history_alex_train, label='Train Acc for Transfered AlexNet')
plt.plot(epochs, history_alex_val, label='Val Acc for Transfered AlexNet')

plt.plot(epochs, history_eff_train, label='Train Acc for Transfered␣
 ↪EfficientNet')
plt.plot(epochs, history_eff_val, label='Val Acc for Transfered EfficientNet')

plt.plot(epochs, history_vit_b_train, label='Train Acc for Transfered Base ViT')
plt.plot(epochs, history_vit_b_val, label='Val Acc for Transfered Base ViT')

plt.plot(epochs, history_vit_l_train, label='Train Acc for Transfered Large␣
 ↪ViT')
plt.plot(epochs, history_vit_l_val, label='Val Acc for Transfered Large ViT')

plt.title('Accuracy History Comparison for all transfer learned models',␣
 ↪fontsize=14)
```

```
plt.ylabel('Accuracy', fontsize=15)
plt.xlabel('No. epoch', fontsize=15)
plt.legend(bbox_to_anchor=(1,0.8))
plt.show()
```



1 Run First

[5]:
```
def get_transform(model_name):

    if model_name == 'alexnet':
        transform = A.Compose([
            A.Resize(227, 227),
            A.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
            ToTensorV2(),
        ])

    elif model_name == 'effinet':
        transform = A.Compose([
            A.Resize(224, 224),
            A.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
            ToTensorV2(),
        ])

    elif model_name == 'TransferViT':

        transform = A.Compose([
            A.Resize(224, 224),
            A.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
            ToTensorV2(),
        ])

    return transform
```

[6]:

```
class SurgicalDataset(Dataset):
    def __init__(self, image_paths, labels, transform=False):
        super(SurgicalDataset, self).__init__()
        self.image_paths = image_paths
        self.labels = labels    #.astype(dtype='int')
        self.transform = transform

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

    def __getitem__(self, idx):
        image_filepath = self.image_paths[idx]
        image = cv2.imread(image_filepath)
        label = self.labels[idx]
        if self.transform is not None:
            image = self.transform(image=image)["image"]

        return image, label
```

[7]:

```
# Preparing the datasets
# Get images
train_image_paths = []
train_data_path = r"C:
 ↪\Users\panji\EECS6691_Advanced_DL\Assignment2\training_data_images"
train_image_paths.append(glob.glob(train_data_path + '/*'))
# unpack the listed list
train_image_paths1 = [item for sublist in train_image_paths for item in sublist]
train_image_paths1 = natsort.natsorted(train_image_paths1)
print('len(train_image_paths1)', len(train_image_paths1))

# Get labels
df = pd.read_csv("Processed_data.csv")
df1 = df.loc[:,"Phases"].to_numpy()
df2 = df1.tolist()
print('len(df2)', len(df2))

# Preparing the datasets (images and labels)
dataset_train = pd.DataFrame(
    {'Link': train_image_paths1,
     'Label': df2,
    })
dataset_train1 = dataset_train.sample(frac=1, random_state=1)
train_image_paths = dataset_train1.loc[:,"Link"].to_numpy().tolist()
```

```
labels = dataset_train1.loc[:,"Label"].to_numpy().tolist()

# manually split the dataset
train_image_paths, valid_image_paths = train_image_paths[:int(0.
 ↪8*len(train_image_paths))], train_image_paths[int(0.
 ↪8*len(train_image_paths)):]
train_labels, valid_labels = labels[:int(0.8*len(labels))], labels[int(0.
 ↪8*len(labels)):]
print('train_labels', len(train_labels))
print('train_image_paths', len(train_image_paths))
print('label distribution in the training data', np.bincount(train_labels))
```

```
len(train_image_paths1) 215057
len(df2) 215057
train_labels 172045
train_image_paths 172045
label distribution in the training data [  243  8681 22901 41140   952 22305
666 10930   896  2308 44928 12987
  1789  1246    73]
```

[8]:

```
class TransferViT_l_32(nn.Module):
    def __init__(self):
        super().__init__()
        self.vit = models.vit_l_32(pretrained=True)
        #self.conv_layer = self.get_conv_proj()
        self.vit.heads = self.get_fc_layers()
        #self.vit = self.get_ViT_encoder()
        #self.fc_model = self.get_fc_layers()
        self.activate_training_layers()

    def activate_training_layers(self):
#         for name, param in self.conv_layer.named_parameters():
#             # for all of these layers set param.requires_grad as True
#             param.requires_grad = False

        for name, param in self.vit.named_parameters():
            number = name.split('.')
            # for all layers except the last conv layer, set param.
 ↪requires_grad = False
            if number[0] == 'heads':
#                 if number[1].split('_')[2] == 11 and number[2] == 'mlp':
#                     param.requires_grad = True
#                 else:
                param.requires_grad = True
                print('required_grad = True', number)
            else:
                param.requires_grad = False
                print('required_grad = False', number)

        #for name, param in self.vit.heads.named_parameters():
            # for all of these layers set param.requires_grad as True

    def get_fc_layers(self):
        return nn.Sequential(
            nn.Dropout(p=0.5, inplace=False),
            nn.Linear(in_features=1024, out_features=512, bias=True),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5, inplace=False),
            nn.Linear(in_features=512, out_features=128, bias=True),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=128, out_features=15, bias=True),
        )

    def forward(self, x):
        #x = self.conv_layer(x)
        x = self.vit(x)
        #x = torch.flatten(x, 1)
        #x = self.fc_model(x)  #call fully connected layers

        return x


class TransferViT(nn.Module):
    def __init__(self):
        super().__init__()
        self.vit = models.vit_b_32(pretrained=True)
        #self.conv_layer = self.get_conv_proj()
        self.vit.heads = self.get_fc_layers()
        #self.vit = self.get_ViT_encoder()
        #self.fc_model = self.get_fc_layers()
        self.activate_training_layers()

    def activate_training_layers(self):
#         for name, param in self.conv_layer.named_parameters():
#             # for all of these layers set param.requires_grad as True
#             param.requires_grad = False

        for name, param in self.vit.named_parameters():
            number = name.split('.')
            # for all layers except the last conv layer, set param.
 ↪requires_grad = False
            if number[0] == 'heads':
#                 if number[1].split('_')[2] == 11 and number[2] == 'mlp':
#                     param.requires_grad = True
#                 else:
                param.requires_grad = True
                print('required_grad = True', number)
            else:
                param.requires_grad = False
                print('required_grad = False', number)

        #for name, param in self.vit.heads.named_parameters():
            # for all of these layers set param.requires_grad as True

    def get_fc_layers(self):
        return nn.Sequential(
            nn.Dropout(p=0.5, inplace=False),
            nn.Linear(in_features=768, out_features=512, bias=True),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5, inplace=False),
            nn.Linear(in_features=512, out_features=128, bias=True),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=128, out_features=15, bias=True),
        )

    def forward(self, x):
        #x = self.conv_layer(x)
        x = self.vit(x)
        #x = torch.flatten(x, 1)
        #x = self.fc_model(x)  #call fully connected layers

        return x


class TransferEffiNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.base_effi_net = models.efficientnet_b7(pretrained=True)
        self.conv_model = self.get_conv_layers()
        self.avg_pool = self.transition_layer()
        self.fc_model = self.get_fc_layers()
        self.activate_training_layers()

    def activate_training_layers(self):
        for name, param in self.conv_model.named_parameters():
            number = int(name.split('.')[1])
            # for all layers except the last conv layer, set param.
 ↪requires_grad = False
            if number == 8:
                param.requires_grad = True
            else:
                param.requires_grad = False

        for name, param in self.fc_model.named_parameters():
            # for all of these layers set param.requires_grad as True
            param.requires_grad = True

    def get_conv_layers(self):
        return self.base_effi_net.features

    def transition_layer(self):
        return self.base_effi_net.avgpool

    def get_fc_layers(self):
        return nn.Sequential(
            nn.Dropout(p=0.5, inplace=False),
            nn.Linear(in_features=2560, out_features=1024, bias=True),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5, inplace=False),
            nn.Linear(in_features=1024, out_features=512, bias=True),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=512, out_features=15, bias=True),
        )

    def forward(self, x):
        x = self.conv_model(x)   #call the conv layers
        x = self.avg_pool(x)  #call the avg pool layer
        x = torch.flatten(x, 1)
        x = self.fc_model(x)  #call fully connected layers

        return x


class TransferAlexNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.base_alex_net = models.alexnet(pretrained=True)
        self.conv_model = self.get_conv_layers()
        self.final_max_pool = self.final_pool_layer()
        self.avg_pool = self.transition_layer()
        self.fc_model = self.get_fc_layers()
        self.activate_training_layers()

    def activate_training_layers(self):
        for name, param in self.conv_model.named_parameters():
            number = int(name.split('.')[0])
            # for all layers except the last layer set param.requires_grad =␣
 ↪False
```

```
            if number < 10:
                param.requires_grad = False
            else:
                param.requires_grad = True

        for name, param in self.fc_model.named_parameters():
            # for all of these layers set param.requires_grad as True
            param.requires_grad = True

    def get_conv_layers(self):
        return self.base_alex_net.features[:12]

    def final_pool_layer(self):
        return nn.MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,␣
↪ceil_mode=False)

    def transition_layer(self):
        return nn.AdaptiveAvgPool2d(output_size=(6, 6))

    def get_fc_layers(self):
        return nn.Sequential(
            nn.Dropout(p=0.5, inplace=False),
            nn.Linear(in_features=9216, out_features=4096, bias=True),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5, inplace=False),
            nn.Linear(in_features=4096, out_features=4096, bias=True),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=4096, out_features=1000, bias=True),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=1000, out_features=15, bias=True),
        )

    def forward(self, x):
        x = self.conv_model(x)   #call the conv layers
        x = self.final_max_pool(x)  #call the max pool layer
        x = self.avg_pool(x)  #call the avg pool layer
        x = torch.flatten(x, 1)
        x = self.fc_model(x)  #call fully connected layers

        return x
```

[13]:

```
def get_pred(model, train_transforms, batch_size, use_cuda=True):
    val_dataset = SurgicalDataset(valid_image_paths, valid_labels,␣
↪train_transforms)
    valid_loader = DataLoader(val_dataset, batch_size, shuffle = False)
    model = model.to('cuda' if use_cuda else 'cpu')
    pr = []
    pred = []
    l = []
    # again no gradients needed
    t = time.time()
    negative_examples = []
    model.eval()
    with torch.no_grad():
        for data in valid_loader:
            images, labels = data[0].to(device), data[1].to(device)
            l.append(labels)
            outputs = model(images)
            _, predictions = torch.max(outputs, 1)
            m = F.softmax(outputs, dim=1)
            # collect the correct predictions for each class
            for label, prediction in zip(labels, predictions):
                pred.append(prediction)
            for p in m:
                pr.append(p)

    processtime = time.time()-t
    print('processtime', processtime)
    return l, pred, pr, processtime
```

[37]:

```
def get_to_cpu(l, pred, pr):
    for i in range(len(l)):
        l[i] = l[i].cpu()
    for i in range(len(l)):
        l[i] = l[i].data.numpy()
    l = [item for sublist in l for item in sublist]
    for i in range(len(l)):
        pred[i] = pred[i].cpu().data.numpy()
    for i in range(len(l)):
        pr[i] = pr[i].cpu().data.numpy()
    return l, pred, pr

def get_class_names(y_true, y_predicted, classes):
    yt = [classes[i] for i in y_true]
    yp = [classes[i] for i in y_predicted]
    return yt, yp
```

2 Data Gathering for the transfered AlexNet

[23]:

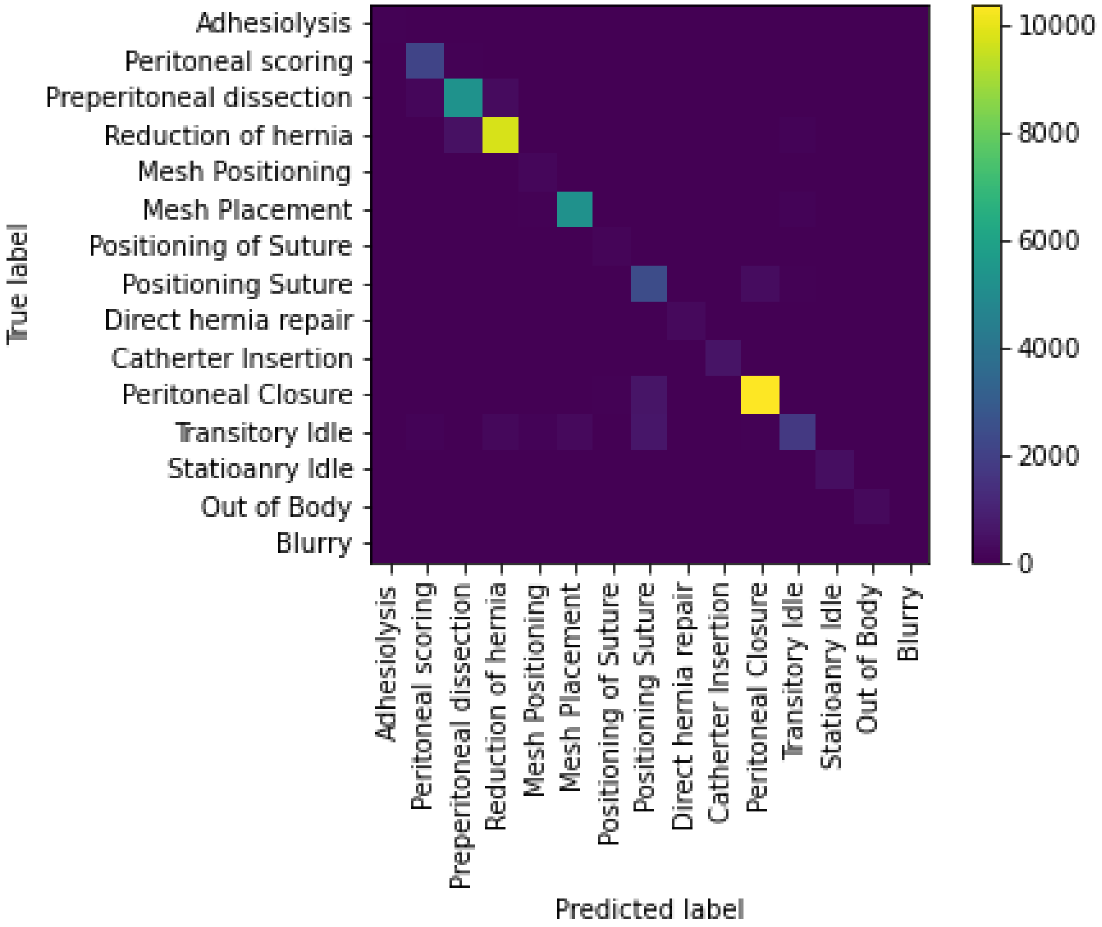
```
# for validation set only
```

```
classes = ['Adhesiolysis', 'Peritoneal scoring', 'Preperitoneal dissection',␣
 ↪'Reduction of hernia', 'Mesh Positioning', 'Mesh Placement', 'Positioning of␣
 ↪Suture', 'Positioning Suture', 'Direct hernia repair',
           'Catherter Insertion', 'Peritoneal Closure', 'Transitory Idle',␣
 ↪'Statioanry Idle', 'Out of Body', 'Blurry']
# correct_predictions = {0: 49, 1: 2117, 2: 5491, 3: 9229, 4: 163, 5: 5288, 6:␣
 ↪136, 7: 2370, 8: 243, 9: 561, 10: 10284, 11: 1953, 12: 399, 13: 281, 14: 17}
# total_predictions =  {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470, 6:␣
 ↪158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
# correct = [correct_predictions[i] for i in correct_predictions]
# total = [total_predictions[i] for i in total_predictions]
save_path = os.path.join(os.getcwd(), 'models', 'TransferAlexNet')
best_alex = TransferAlexNet()
best_alex.load_state_dict(torch.load(os.path.join(save_path, 'best.pt')))
transforms = get_transform('alexnet')
batch_size = 32
y_test_true, y_test_predicted, pr, time = get_pred(best_alex, transforms,␣
 ↪batch_size)
y_test_true, y_test_predicted, pr = get_to_cpu(y_test_true, y_test_predicted,␣
 ↪pr)
y_true, y_predicted = get_class_names(y_test_true, y_test_predicted, classes)
```

```
processtime 304.6354877948761
```

[44]:

```
cm = confusion_matrix(y_true, y_predicted, labels=classes)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
disp.plot(include_values=False, xticks_rotation = 'vertical')
plt.show()
```

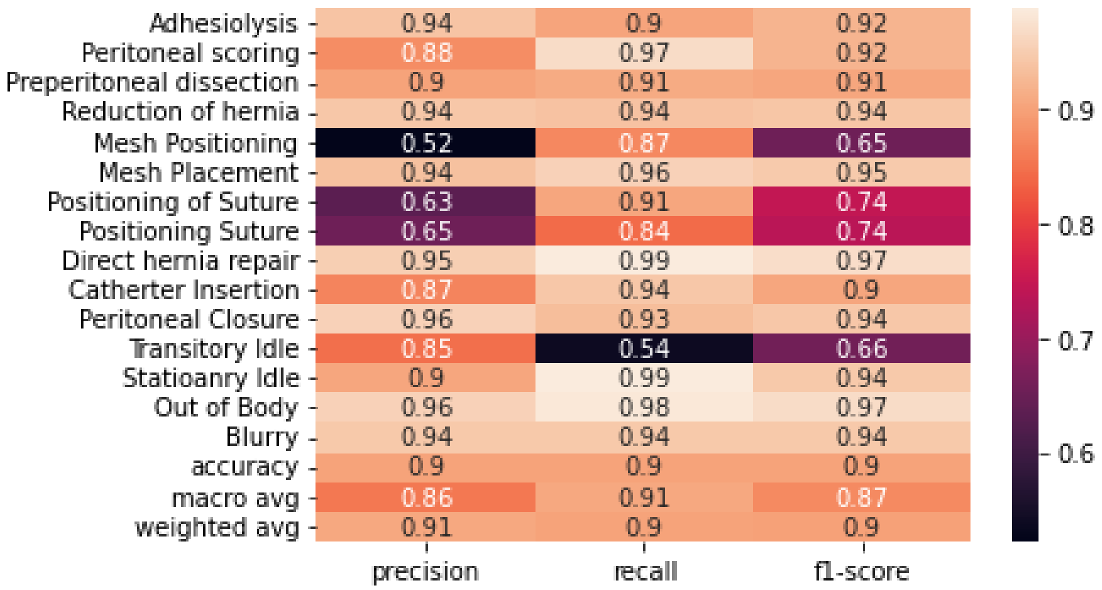


```
[48]: target_names = classes
      c_report = classification_report(y_true, y_predicted, labels=classes,␣
       ↪target_names=target_names, output_dict=True)
      print(c_report)
      basic_report = classification_report(y_true, y_predicted, labels=classes)
      print(basic_report)
      sns.heatmap(pd.DataFrame(c_report).iloc[:-1, :].T, annot=True)
```

```
{'Adhesiolysis': {'precision': 0.94, 'recall': 0.9038461538461539, 'f1-score':
0.9215686274509804, 'support': 52}, 'Peritoneal scoring': {'precision':
0.8780183180682765, 'recall': 0.9687643546164446, 'f1-score':
0.9211618257261411, 'support': 2177}, 'Preperitoneal dissection': {'precision':
0.9012809564474807, 'recall': 0.9099844800827729, 'f1-score':
0.9056118071048567, 'support': 5799}, 'Reduction of hernia': {'precision':
0.9438691570695829, 'recall': 0.9373378183565594, 'f1-score':
0.9405921496769215, 'support': 10405}, 'Mesh Positioning': {'precision':
0.5224719101123596, 'recall': 0.8732394366197183, 'f1-score':
0.6537785588752197, 'support': 213}, 'Mesh Placement': {'precision':
```

```
0.9361473797174029, 'recall': 0.956855575868373, 'f1-score': 0.9463882108308471,
'support': 5470}, 'Positioning of Suture': {'precision': 0.6327433628318584,
'recall': 0.9050632911392406, 'f1-score': 0.7447916666666667, 'support': 158},
'Positioning Suture': {'precision': 0.6543816543816544, 'recall':
0.8431234611326064, 'f1-score': 0.7368582846603136, 'support': 2843}, 'Direct
hernia repair': {'precision': 0.953125, 'recall': 0.9878542510121457,
'f1-score': 0.970178926441352, 'support': 247}, 'Catherter Insertion':
{'precision': 0.8683385579937304, 'recall': 0.9437819420783645, 'f1-score':
0.9044897959183672, 'support': 587}, 'Peritoneal Closure': {'precision':
0.9566660520007376, 'recall': 0.9318365514144589, 'f1-score':
0.9440880760656931, 'support': 11135}, 'Transitory Idle': {'precision':
0.8476237138657521, 'recall': 0.5379353233830846, 'f1-score':
0.6581700589689937, 'support': 3216}, 'Statioanry Idle': {'precision':
0.9047619047619048, 'recall': 0.9876237623762376, 'f1-score': 0.944378698224852,
'support': 404}, 'Out of Body': {'precision': 0.956081081081081, 'recall':
0.9826388888888888, 'f1-score': 0.9691780821917807, 'support': 288}, 'Blurry':
{'precision': 0.9444444444444444, 'recall': 0.9444444444444444, 'f1-score':
0.9444444444444444, 'support': 18}, 'accuracy': 0.9008881242443969, 'macro avg':
{'precision': 0.8559968995184178, 'recall': 0.9076219823506331, 'f1-score':
0.8737119475498286, 'support': 43012}, 'weighted avg': {'precision':
0.9062971269660363, 'recall': 0.9008881242443969, 'f1-score': 0.899694880646224,
'support': 43012}}
                          precision    recall  f1-score   support

            Adhesiolysis       0.94      0.90      0.92        52
      Peritoneal scoring       0.88      0.97      0.92      2177
Preperitoneal dissection       0.90      0.91      0.91      5799
     Reduction of hernia       0.94      0.94      0.94     10405
        Mesh Positioning       0.52      0.87      0.65       213
          Mesh Placement       0.94      0.96      0.95      5470
   Positioning of Suture       0.63      0.91      0.74       158
      Positioning Suture       0.65      0.84      0.74      2843
    Direct hernia repair       0.95      0.99      0.97       247
     Catherter Insertion       0.87      0.94      0.90       587
      Peritoneal Closure       0.96      0.93      0.94     11135
         Transitory Idle       0.85      0.54      0.66      3216
         Statioanry Idle       0.90      0.99      0.94       404
             Out of Body       0.96      0.98      0.97       288
                  Blurry       0.94      0.94      0.94        18

                accuracy                           0.90     43012
               macro avg       0.86      0.91      0.87     43012
            weighted avg       0.91      0.90      0.90     43012
```

```
[48]: <AxesSubplot:>
```



[49]:
```
print(metrics.roc_auc_score(y_test_true, pr, multi_class = 'ovr'))
print(metrics.roc_auc_score(y_test_true, pr, multi_class = 'ovo'))
```

```
0.991774780210564
0.993675663933648
```

3 Data Gathering for the transfered EfficientNet

[50]:
```
correct_predictions = {0: 51, 1: 2094, 2: 5631, 3: 9911, 4: 176, 5: 5285, 6:␣
 ↪145, 7: 2457, 8: 245, 9: 567, 10: 10674, 11: 2494, 12: 400, 13: 286, 14: 17}
total_predictions = {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470, 6:␣
 ↪158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
```

[51]:
```
# for validation set only
classes = ['Adhesiolysis', 'Peritoneal scoring', 'Preperitoneal dissection',␣
 ↪'Reduction of hernia', 'Mesh Positioning', 'Mesh Placement', 'Positioning of␣
 ↪Suture', 'Positioning Suture', 'Direct hernia repair',
           'Catherter Insertion', 'Peritoneal Closure', 'Transitory Idle',␣
 ↪'Statioanry Idle', 'Out of Body', 'Blurry']
# correct_predictions = {0: 49, 1: 2117, 2: 5491, 3: 9229, 4: 163, 5: 5288, 6:␣
 ↪136, 7: 2370, 8: 243, 9: 561, 10: 10284, 11: 1953, 12: 399, 13: 281, 14: 17}
# total_predictions =  {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470, 6:␣
 ↪158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
# correct = [correct_predictions[i] for i in correct_predictions]
# total = [total_predictions[i] for i in total_predictions]
save_path = os.path.join(os.getcwd(), 'models', 'TransferEffiNet')
```

```
best_effi = TransferEffiNet()
best_effi.load_state_dict(torch.load(os.path.join(save_path, 'best.pt')))
transforms = get_transform('effinet')
batch_size = 32
y_test_true, y_test_predicted, pr, time = get_pred(best_effi, transforms,␣
 ↪batch_size)
y_test_true, y_test_predicted, pr = get_to_cpu(y_test_true, y_test_predicted,␣
 ↪pr)
```

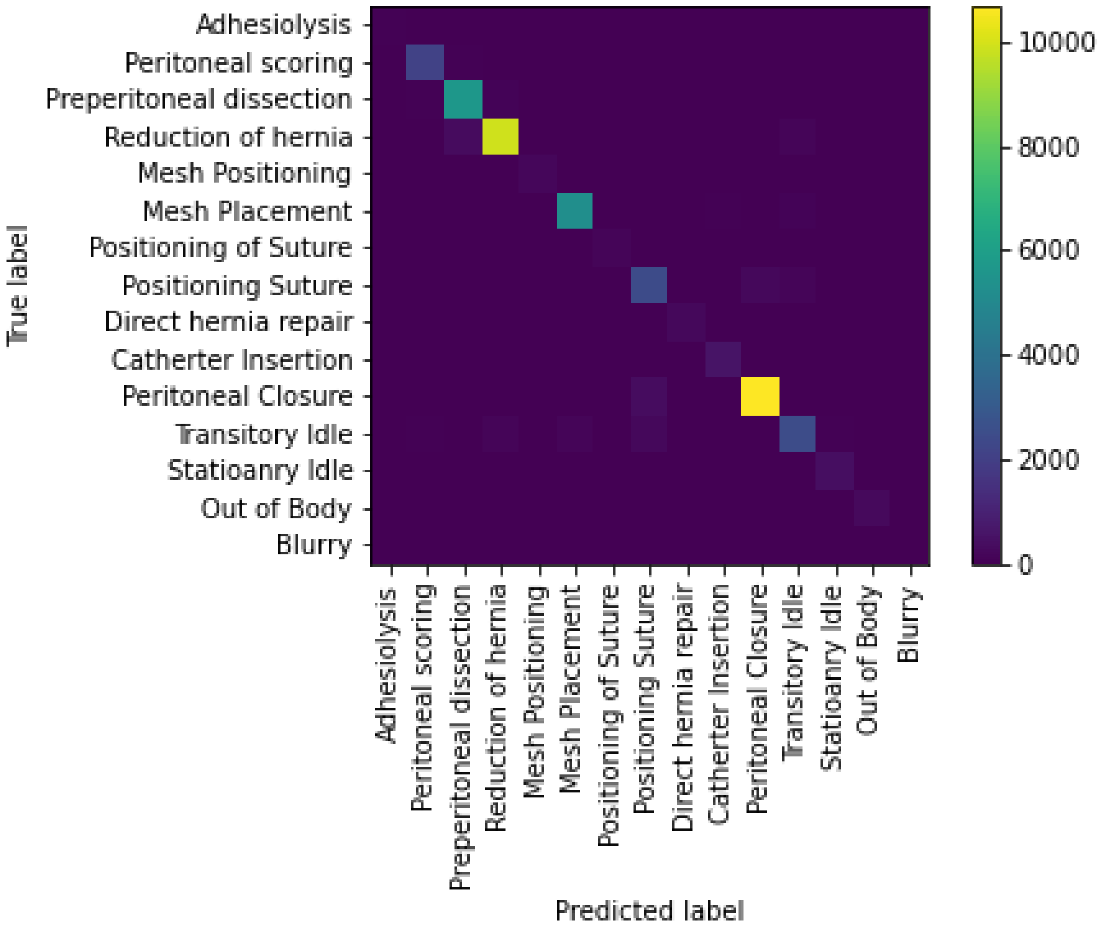
```
processtime 351.8050129413605
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [51], in <cell line: 15>()
     13 y_test_true, y_test_predicted, pr, time = get_pred(best_effi,␣
 ↪transforms, batch_size)
     14 y_test_true, y_test_predicted, pr = get_to_cpu(y_test_true,␣
 ↪y_test_predicted, pr)
---> 15 y_true, y_predicted =␣
 ↪get_class_names(y_test_true, y_test_predicted, classes)
     17 cm = confusion_matrix(y_true, y_predicted, labels=classes)
     18 disp = ConfusionMatrixDisplay(confusion_matrix=cm,␣
 ↪display_labels=classes)

Input In [37], in get_class_names(y_true, y_predicted, classes)
     13 def get_class_names(y_true, y_predicted, classes):
---> 14     yt = [classes[i] for i in y_true]
     15     yp = [classes[i] for i in y_predicted]
     16     return yt, yp

Input In [37], in <listcomp>(.0)
     13 def get_class_names(y_true, y_predicted, classes):
---> 14     yt = [classes[i] for i in y_true]
     15     yp = [classes[i] for i in y_predicted]
     16     return yt, yp

TypeError: 'set' object is not subscriptable
```

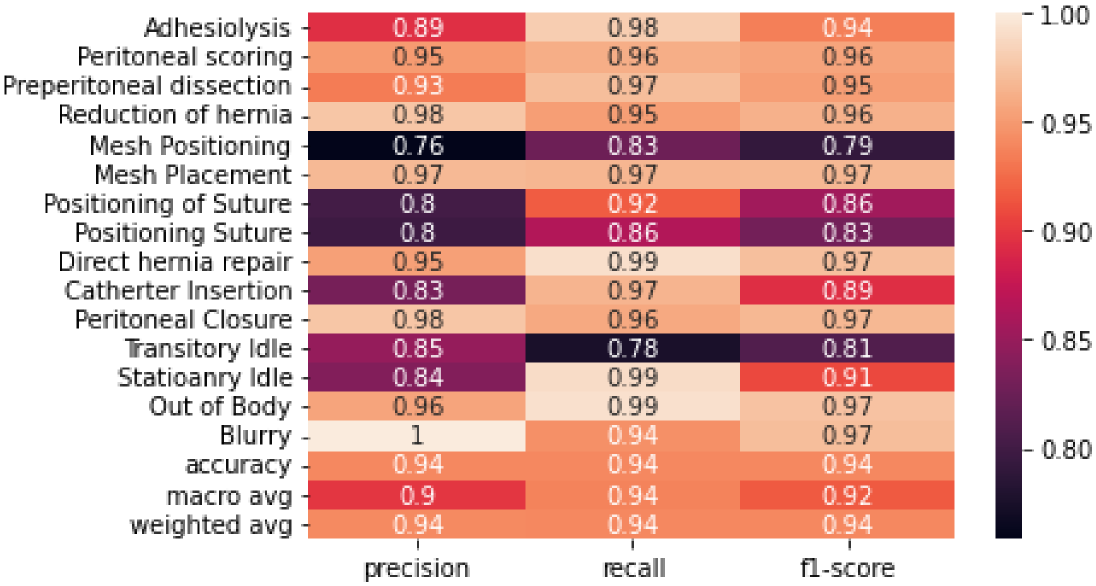
[55]:

```
y_true, y_predicted = get_class_names(y_test_true, y_test_predicted, classes)
cm = confusion_matrix(y_true, y_predicted, labels=classes)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
disp.plot(include_values=False, xticks_rotation = 'vertical')
plt.show()

target_names = classes
```

```
c_report = classification_report(y_true, y_predicted, labels=classes,␣
 ↪target_names=target_names, output_dict=True)
print(c_report)
basic_report = classification_report(y_true, y_predicted, labels=classes)
print(basic_report)
sns.heatmap(pd.DataFrame(c_report).iloc[:-1, :].T, annot=True)

print(metrics.roc_auc_score(y_test_true, pr, multi_class = 'ovr'))
print(metrics.roc_auc_score(y_test_true, pr, multi_class = 'ovo'))
```



```
{'Adhesiolysis': {'precision': 0.8947368421052632, 'recall': 0.9807692307692307,
'f1-score': 0.9357798165137614, 'support': 52}, 'Peritoneal scoring':
{'precision': 0.9509536784741145, 'recall': 0.9618741387230133, 'f1-score':
0.9563827357844258, 'support': 2177}, 'Preperitoneal dissection': {'precision':
0.9339857356112125, 'recall': 0.9710294878427315, 'f1-score':
0.9521474467365573, 'support': 5799}, 'Reduction of hernia': {'precision':
0.975588148439807, 'recall': 0.9525228255646324, 'f1-score': 0.9639175257731958,
'support': 10405}, 'Mesh Positioning': {'precision': 0.7586206896551724,
'recall': 0.8262910798122066, 'f1-score': 0.7910112359550561, 'support': 213},
'Mesh Placement': {'precision': 0.9681260304084998, 'recall':
0.9661791590493601, 'f1-score': 0.9671516149693477, 'support': 5470},
'Positioning of Suture': {'precision': 0.8011049723756906, 'recall':
0.9177215189873418, 'f1-score': 0.855457227138643, 'support': 158}, 'Positioning
Suture': {'precision': 0.7974683544303798, 'recall': 0.8642279282448119,
'f1-score': 0.8295070898041864, 'support': 2843}, 'Direct hernia repair':
{'precision': 0.953307392996109, 'recall': 0.9919028340080972, 'f1-score':
0.9722222222222222, 'support': 247}, 'Catherter Insertion': {'precision':
0.8313782991202346, 'recall': 0.9659284497444633, 'f1-score':
0.8936170212765957, 'support': 587}, 'Peritoneal Closure': {'precision':
0.9756855575868373, 'recall': 0.9585990121239335, 'f1-score': 0.967066817667044,
'support': 11135}, 'Transitory Idle': {'precision': 0.8482993197278912,
'recall': 0.775497512437811, 'f1-score': 0.8102664067576348, 'support': 3216},
'Statioanry Idle': {'precision': 0.8385744234800838, 'recall':
0.9900990099009901, 'f1-score': 0.9080590238365492, 'support': 404}, 'Out of
Body': {'precision': 0.9565217391304348, 'recall': 0.9930555555555556,
'f1-score': 0.9744463373083475, 'support': 288}, 'Blurry': {'precision': 1.0,
'recall': 0.9444444444444444, 'f1-score': 0.9714285714285714, 'support': 18},
'accuracy': 0.9400399888403236, 'macro avg': {'precision': 0.8989567455694487,
'recall': 0.9373428124805748, 'f1-score': 0.9165640728781423, 'support': 43012},
'weighted avg': {'precision': 0.9412043327733812, 'recall': 0.9400399888403236,
'f1-score': 0.940151731033028, 'support': 43012}}
                          precision    recall  f1-score   support

            Adhesiolysis       0.89      0.98      0.94        52
      Peritoneal scoring       0.95      0.96      0.96      2177
Preperitoneal dissection       0.93      0.97      0.95      5799
     Reduction of hernia       0.98      0.95      0.96     10405
        Mesh Positioning       0.76      0.83      0.79       213
          Mesh Placement       0.97      0.97      0.97      5470
   Positioning of Suture       0.80      0.92      0.86       158
      Positioning Suture       0.80      0.86      0.83      2843
    Direct hernia repair       0.95      0.99      0.97       247
     Catherter Insertion       0.83      0.97      0.89       587
      Peritoneal Closure       0.98      0.96      0.97     11135
         Transitory Idle       0.85      0.78      0.81      3216
         Statioanry Idle       0.84      0.99      0.91       404
             Out of Body       0.96      0.99      0.97       288
                  Blurry       1.00      0.94      0.97        18

                accuracy                           0.94     43012
               macro avg       0.90      0.94      0.92     43012
            weighted avg       0.94      0.94      0.94     43012

0.9962277793123141
0.9968360295296173
```



4 Data Gathering for the transfered ViT base model

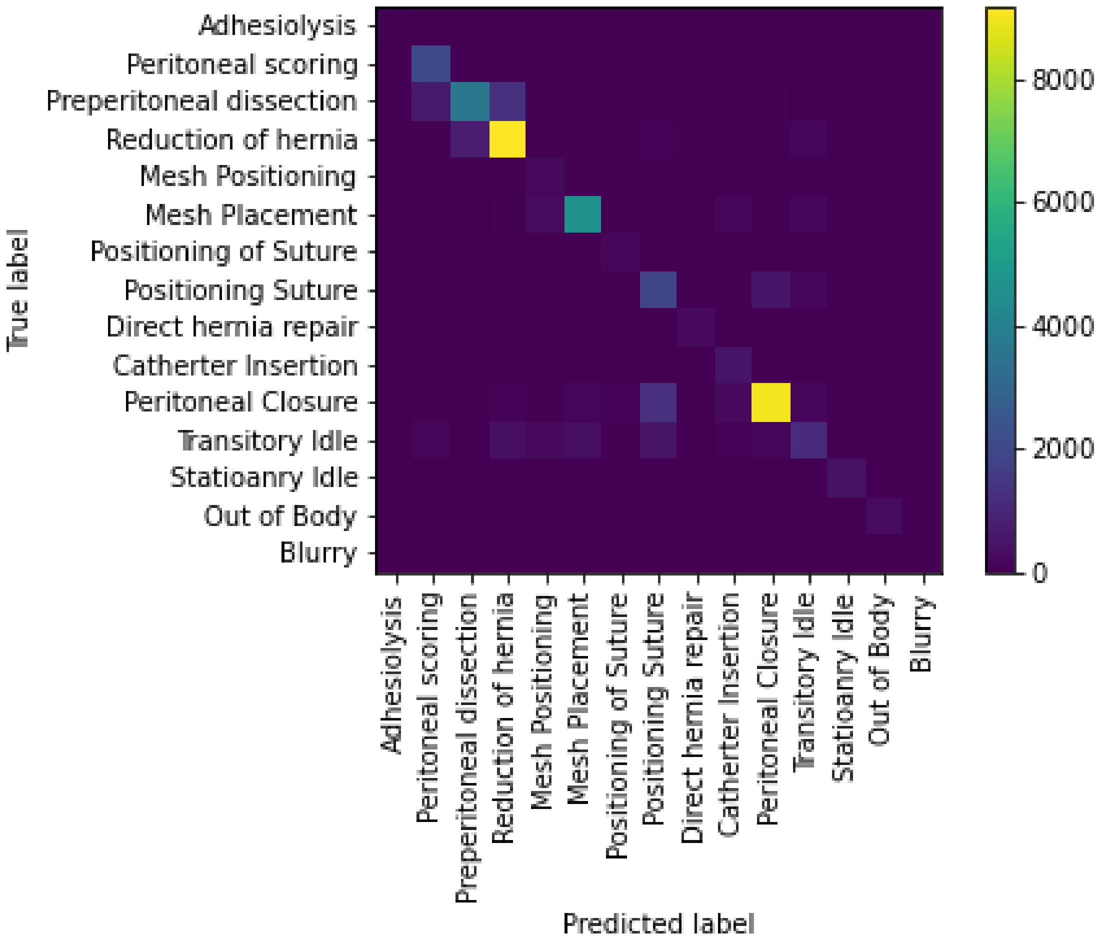
```
[ ]: correct_predictions = {0: 51, 1: 2044, 2: 3659, 3: 9166, 4: 186, 5: 4695, 6:␣
      ↪131, 7: 1904, 8: 244, 9: 530, 10: 8993, 11: 1135, 12: 399, 13: 284, 14: 17}
     total_predictions = {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470, 6:␣
      ↪158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
```

```
[58]: # for validation set only
      import time
      classes = ['Adhesiolysis', 'Peritoneal scoring', 'Preperitoneal dissection',␣
       ↪'Reduction of hernia', 'Mesh Positioning', 'Mesh Placement', 'Positioning of␣
       ↪Suture', 'Positioning Suture', 'Direct hernia repair',
                 'Catherter Insertion', 'Peritoneal Closure', 'Transitory Idle',␣
       ↪'Statioanry Idle', 'Out of Body', 'Blurry']
      # correct_predictions = {0: 49, 1: 2117, 2: 5491, 3: 9229, 4: 163, 5: 5288, 6:␣
       ↪136, 7: 2370, 8: 243, 9: 561, 10: 10284, 11: 1953, 12: 399, 13: 281, 14: 17}
      # total_predictions =  {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470, 6:␣
       ↪158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
      # correct = [correct_predictions[i] for i in correct_predictions]
      # total = [total_predictions[i] for i in total_predictions]
      save_path = os.path.join(os.getcwd(), 'models', 'TransferViT')
      best_vitb = TransferViT()
      best_vitb.load_state_dict(torch.load(os.path.join(save_path, 'best.pt')))
      transforms = get_transform('TransferViT')
      batch_size = 32
```

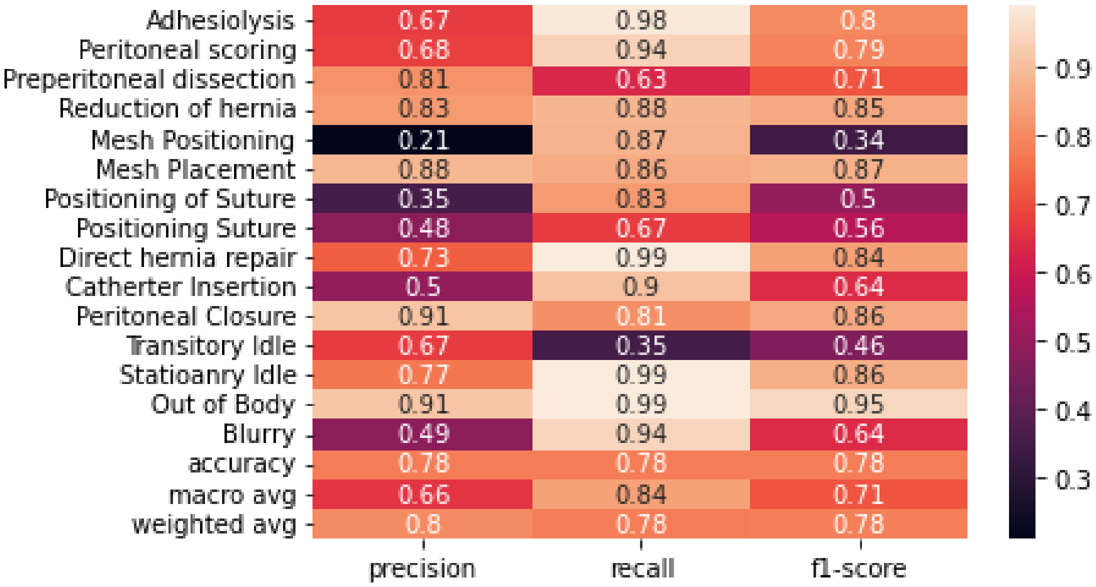
```
y_test_true, y_test_predicted, pr, time = get_pred(best_vitb, transforms,␣
 ↪batch_size)
y_test_true, y_test_predicted, pr = get_to_cpu(y_test_true, y_test_predicted,␣
 ↪pr)
y_true, y_predicted = get_class_names(y_test_true, y_test_predicted, classes)

cm = confusion_matrix(y_true, y_predicted, labels=classes)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
disp.plot(include_values=False, xticks_rotation = 'vertical')
plt.show()

target_names = classes
c_report = classification_report(y_true, y_predicted, labels=classes,␣
 ↪target_names=target_names, output_dict=True)
print(c_report)
basic_report = classification_report(y_true, y_predicted, labels=classes)
print(basic_report)
sns.heatmap(pd.DataFrame(c_report).iloc[:-1, :].T, annot=True)

print(metrics.roc_auc_score(y_test_true, pr, multi_class = 'ovr'))
print(metrics.roc_auc_score(y_test_true, pr, multi_class = 'ovo'))
```

```
required_grad = False ['class_token']
required_grad = False ['conv_proj', 'weight']
required_grad = False ['conv_proj', 'bias']
required_grad = False ['encoder', 'pos_embedding']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'ln_2',
'weight']
```

```
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'ln', 'weight']
required_grad = False ['encoder', 'ln', 'bias']
required_grad = True ['heads', '1', 'weight']
required_grad = True ['heads', '1', 'bias']
required_grad = True ['heads', '4', 'weight']
required_grad = True ['heads', '4', 'bias']
required_grad = True ['heads', '6', 'weight']
required_grad = True ['heads', '6', 'bias']
processtime 322.493004322052
```



```
{'Adhesiolysis': {'precision': 0.6710526315789473, 'recall': 0.9807692307692307,
'f1-score': 0.7968749999999999, 'support': 52}, 'Peritoneal scoring':
{'precision': 0.6774941995359629, 'recall': 0.9389067524115756, 'f1-score':
0.7870619946091645, 'support': 2177}, 'Preperitoneal dissection': {'precision':
0.8127498889382496, 'recall': 0.630970857044318, 'f1-score': 0.71041646442093,
'support': 5799}, 'Reduction of hernia': {'precision': 0.8286773347798572,
'recall': 0.8809226333493513, 'f1-score': 0.8540016770707164, 'support': 10405},
'Mesh Positioning': {'precision': 0.21064552661381652, 'recall':
0.8732394366197183, 'f1-score': 0.3394160583941606, 'support': 213}, 'Mesh
Placement': {'precision': 0.8838478915662651, 'recall': 0.8583180987202925,
'f1-score': 0.8708959376739009, 'support': 5470}, 'Positioning of Suture':
{'precision': 0.35405405405405405, 'recall': 0.8291139240506329, 'f1-score':
0.4962121212121212, 'support': 158}, 'Positioning Suture': {'precision':
0.4846016798167473, 'recall': 0.6697150896939852, 'f1-score':
0.5623154164205552, 'support': 2843}, 'Direct hernia repair': {'precision':
0.7261904761904762, 'recall': 0.9878542510121457, 'f1-score': 0.83704974271012,
'support': 247}, 'Catherter Insertion': {'precision': 0.4967197750702905,
'recall': 0.9028960817717206, 'f1-score': 0.6408706166868199, 'support': 587},
'Peritoneal Closure': {'precision': 0.9084756035963228, 'recall':
0.8076335877862595, 'f1-score': 0.8550917562042407, 'support': 11135},
'Transitory Idle': {'precision': 0.6704075605434141, 'recall':
0.3529228855721393, 'f1-score': 0.46241597066612344, 'support': 3216},
'Statioanry Idle': {'precision': 0.7673076923076924, 'recall':
0.9876237623762376, 'f1-score': 0.8636363636363636, 'support': 404}, 'Out of
Body': {'precision': 0.9102564102564102, 'recall': 0.9861111111111112,
'f1-score': 0.9466666666666667, 'support': 288}, 'Blurry': {'precision':
0.4857142857142857, 'recall': 0.9444444444444444, 'f1-score':
0.6415094339622641, 'support': 18}, 'accuracy': 0.7774109550823026, 'macro avg':
{'precision': 0.6592130007041862, 'recall': 0.8420961431155443, 'f1-score':
0.7109623480222764, 'support': 43012}, 'weighted avg': {'precision':
0.8016888680180408, 'recall': 0.7774109550823026, 'f1-score':
0.7788114874703039, 'support': 43012}}
                          precision    recall  f1-score   support

            Adhesiolysis       0.67      0.98      0.80        52
      Peritoneal scoring       0.68      0.94      0.79      2177
Preperitoneal dissection       0.81      0.63      0.71      5799
     Reduction of hernia       0.83      0.88      0.85     10405
        Mesh Positioning       0.21      0.87      0.34       213
          Mesh Placement       0.88      0.86      0.87      5470
   Positioning of Suture       0.35      0.83      0.50       158
      Positioning Suture       0.48      0.67      0.56      2843
    Direct hernia repair       0.73      0.99      0.84       247
     Catherter Insertion       0.50      0.90      0.64       587
      Peritoneal Closure       0.91      0.81      0.86     11135
         Transitory Idle       0.67      0.35      0.46      3216
         Statioanry Idle       0.77      0.99      0.86       404
             Out of Body       0.91      0.99      0.95       288
                  Blurry       0.49      0.94      0.64        18

                accuracy                           0.78     43012
               macro avg       0.66      0.84      0.71     43012
            weighted avg       0.80      0.78      0.78     43012

0.9788159404747322
0.9870761309204674
```



5 Data Gathering for the transfered ViT large model

[ ]:

```
correct_predictions = {0: 51, 1: 2032, 2: 3500, 3: 9769, 4: 186, 5: 4936, 6:␣
 ↪133, 7: 1903, 8: 245, 9: 527, 10: 9819, 11: 1256, 12: 398, 13: 282, 14: 17}
total_predictions = {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470, 6:␣
 ↪158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
```

[59]:

```
# for validation set only
import time
classes = ['Adhesiolysis', 'Peritoneal scoring', 'Preperitoneal dissection',␣
 ↪'Reduction of hernia', 'Mesh Positioning', 'Mesh Placement', 'Positioning of␣
 ↪Suture', 'Positioning Suture', 'Direct hernia repair',
          'Catherter Insertion', 'Peritoneal Closure', 'Transitory Idle',␣
 ↪'Statioanry Idle', 'Out of Body', 'Blurry']
# correct_predictions = {0: 49, 1: 2117, 2: 5491, 3: 9229, 4: 163, 5: 5288, 6:␣
 ↪136, 7: 2370, 8: 243, 9: 561, 10: 10284, 11: 1953, 12: 399, 13: 281, 14: 17}
# total_predictions =  {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470, 6:␣
 ↪158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
# correct = [correct_predictions[i] for i in correct_predictions]
# total = [total_predictions[i] for i in total_predictions]
save_path = os.path.join(os.getcwd(), 'models', 'TransferViT_l_32')
best_ViTL = TransferViT_l_32()
best_ViTL.load_state_dict(torch.load(os.path.join(save_path, 'best.pt')))
transforms = get_transform('TransferViT')
batch_size = 32
```

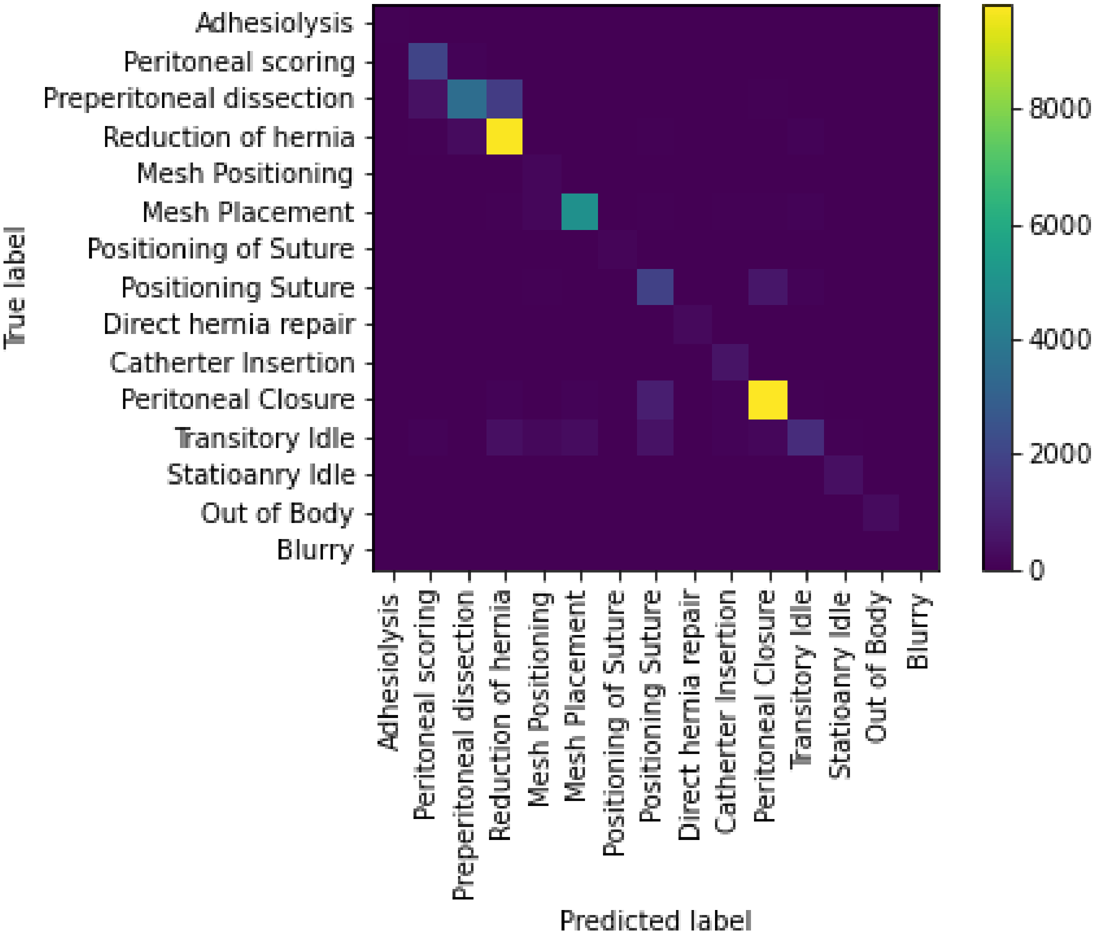
```
y_test_true, y_test_predicted, pr, time = get_pred(best_ViTL, transforms,␣
 ↪batch_size)
y_test_true, y_test_predicted, pr = get_to_cpu(y_test_true, y_test_predicted,␣
 ↪pr)
y_true, y_predicted = get_class_names(y_test_true, y_test_predicted, classes)

cm = confusion_matrix(y_true, y_predicted, labels=classes)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
disp.plot(include_values=False, xticks_rotation = 'vertical')
plt.show()

target_names = classes
c_report = classification_report(y_true, y_predicted, labels=classes,␣
 ↪target_names=target_names, output_dict=True)
print(c_report)
basic_report = classification_report(y_true, y_predicted, labels=classes)
print(basic_report)
sns.heatmap(pd.DataFrame(c_report).iloc[:-1, :].T, annot=True)

print(metrics.roc_auc_score(y_test_true, pr, multi_class = 'ovr'))
print(metrics.roc_auc_score(y_test_true, pr, multi_class = 'ovo'))
```

```
required_grad = False ['class_token']
required_grad = False ['conv_proj', 'weight']
required_grad = False ['conv_proj', 'bias']
required_grad = False ['encoder', 'pos_embedding']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'ln_2',
'weight']
```

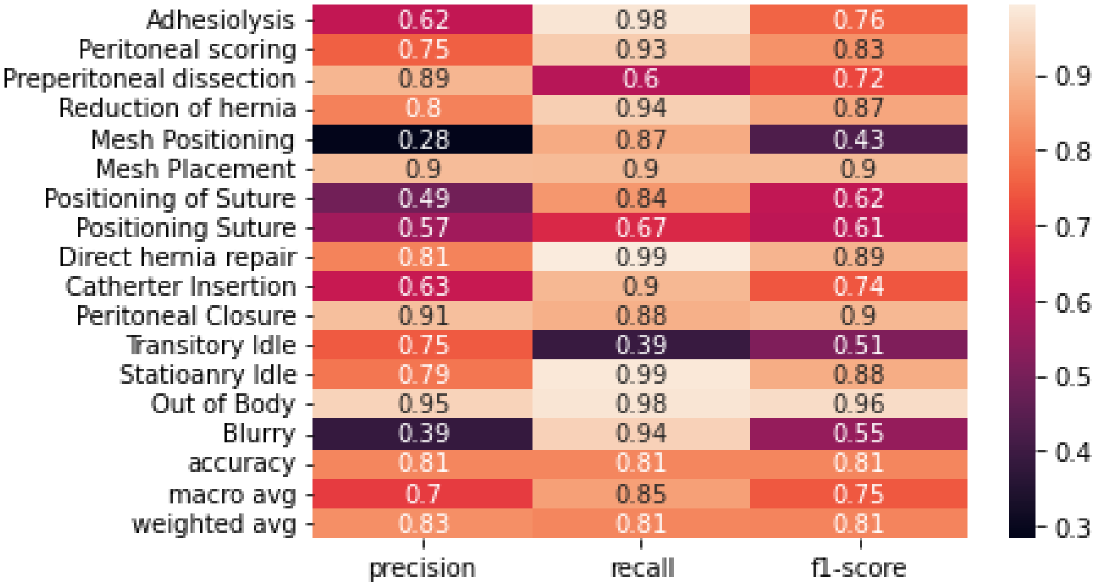
```
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_12', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_12', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_12',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_12',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_12',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_12',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_12', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_12', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_12', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_12', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_12', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_12', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_13', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_13', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_13',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_13',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_13',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_13',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_13', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_13', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_13', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_13', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_13', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_13', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_14', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_14', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_14',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_14',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_14',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_14',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_14', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_14', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_14', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_14', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_14', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_14', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_15', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_15', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_15',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_15',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_15',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_15',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_15', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_15', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_15', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_15', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_15', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_15', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_16', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_16', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_16',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_16',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_16',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_16',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_16', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_16', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_16', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_16', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_16', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_16', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_17', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_17', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_17',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_17',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_17',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_17',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_17', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_17', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_17', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_17', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_17', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_17', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_18', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_18', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_18',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_18',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_18',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_18',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_18', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_18', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_18', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_18', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_18', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_18', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_19', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_19', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_19',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_19',
'self_attention', 'in_proj_bias']
```

```
required_grad = False ['encoder', 'layers', 'encoder_layer_19',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_19',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_19', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_19', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_19', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_19', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_19', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_19', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_20', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_20', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_20',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_20',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_20',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_20',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_20', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_20', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_20', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_20', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_20', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_20', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_21', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_21', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_21',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_21',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_21',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_21',
'self_attention', 'out_proj', 'bias']
```

```
required_grad = False ['encoder', 'layers', 'encoder_layer_21', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_21', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_21', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_21', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_21', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_21', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_22', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_22', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_22',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_22',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_22',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_22',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_22', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_22', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_22', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_22', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_22', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_22', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_23', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_23', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_23',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_23',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_23',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_23',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_23', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_23', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_23', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_23', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_23', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_23', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'ln', 'weight']
required_grad = False ['encoder', 'ln', 'bias']
required_grad = True ['heads', '1', 'weight']
required_grad = True ['heads', '1', 'bias']
required_grad = True ['heads', '4', 'weight']
required_grad = True ['heads', '4', 'bias']
required_grad = True ['heads', '6', 'weight']
required_grad = True ['heads', '6', 'bias']
processtime 342.0810308456421
```



```
{'Adhesiolysis': {'precision': 0.6219512195121951, 'recall': 0.9807692307692307,
'f1-score': 0.7611940298507464, 'support': 52}, 'Peritoneal scoring':
{'precision': 0.7542687453600594, 'recall': 0.9333945796968305, 'f1-score':
0.8343256004927119, 'support': 2177}, 'Preperitoneal dissection': {'precision':
0.8939974457215837, 'recall': 0.6035523366097603, 'f1-score':
0.7206094296891086, 'support': 5799}, 'Reduction of hernia': {'precision':
0.8045626750123538, 'recall': 0.9388755406054782, 'f1-score':
0.8665454384175278, 'support': 10405}, 'Mesh Positioning': {'precision':
0.28440366972477066, 'recall': 0.8732394366197183, 'f1-score':
0.4290657439446367, 'support': 213}, 'Mesh Placement': {'precision':
0.9041949074922147, 'recall': 0.9023765996343693, 'f1-score':
0.9032848385030653, 'support': 5470}, 'Positioning of Suture': {'precision':
0.4907749077490775, 'recall': 0.8417721518987342, 'f1-score': 0.62004662004662,
'support': 158}, 'Positioning Suture': {'precision': 0.5672131147540984,
'recall': 0.6693633485754484, 'f1-score': 0.6140690545337205, 'support': 2843},
'Direct hernia repair': {'precision': 0.8085808580858086, 'recall':
0.9919028340080972, 'f1-score': 0.8909090909090909, 'support': 247}, 'Catherter
Insertion': {'precision': 0.6311377245508982, 'recall': 0.8977853492333902,
'f1-score': 0.7412095639943742, 'support': 587}, 'Peritoneal Closure':
{'precision': 0.911106987102162, 'recall': 0.8818140996856758, 'f1-score':
0.8962212486308871, 'support': 11135}, 'Transitory Idle': {'precision':
0.7480643240023823, 'recall': 0.39054726368159204, 'f1-score':
0.5131767109295199, 'support': 3216}, 'Statioanry Idle': {'precision':
0.7896825396825397, 'recall': 0.9851485148514851, 'f1-score':
0.8766519823788547, 'support': 404}, 'Out of Body': {'precision':
0.9463087248322147, 'recall': 0.9791666666666666, 'f1-score': 0.962457337883959,
'support': 288}, 'Blurry': {'precision': 0.38636363636363635, 'recall':
0.9444444444444444, 'f1-score': 0.5483870967741935, 'support': 18}, 'accuracy':
0.8149818655258998, 'macro avg': {'precision': 0.7028407653297329, 'recall':
0.8542768264653946, 'f1-score': 0.7452102524652678, 'support': 43012}, 'weighted
avg': {'precision': 0.8287565209123339, 'recall': 0.8149818655258998,
'f1-score': 0.810318599384176, 'support': 43012}}
                          precision    recall  f1-score   support

            Adhesiolysis       0.62      0.98      0.76        52
      Peritoneal scoring       0.75      0.93      0.83      2177
Preperitoneal dissection       0.89      0.60      0.72      5799
     Reduction of hernia       0.80      0.94      0.87     10405
        Mesh Positioning       0.28      0.87      0.43       213
          Mesh Placement       0.90      0.90      0.90      5470
   Positioning of Suture       0.49      0.84      0.62       158
      Positioning Suture       0.57      0.67      0.61      2843
    Direct hernia repair       0.81      0.99      0.89       247
     Catherter Insertion       0.63      0.90      0.74       587
      Peritoneal Closure       0.91      0.88      0.90     11135
         Transitory Idle       0.75      0.39      0.51      3216
         Statioanry Idle       0.79      0.99      0.88       404
             Out of Body       0.95      0.98      0.96       288
                  Blurry       0.39      0.94      0.55        18
                accuracy                           0.81     43012
               macro avg       0.70      0.85      0.75     43012
            weighted avg       0.83      0.81      0.81     43012

0.9846040426947692
0.989946427611651
```



```
[ ]:
```