National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Pinaki Pani
Student ID: 23112573

School of Computing
National College of Ireland

Supervisor: Dr Giovani Estrada

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Pinaki Pani
Student ID:	23112573
Programme:	MSc in Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Dr Giovani Estrada
Submission Due Date:	16/09/2024
Project Title:	Configuration Manual
Word Count:	628
Page Count:	24

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Pinaki Pani
Date:	16th September 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Pinaki Pani
23112573

1 Introduction

This configuration report is to show the step by step guide for conducting this research. This report contains all the details regarding the hardware and the software setups for the research. Alongside that it contains all the codes and the sample output screenshots for the codes as well demonstrating the working of the research.

2 System Configuration

The Table 1 shows the hardware configuration used to undergo this research.

2.1 Hardware Configurations

Categories	Hardware Details
Device name	ROG Strix G531GT_G531GT
Processor	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz
Installed RAM	16.0 GB (15.9 GB usable)
Disk Space	256 GB
System type	64-bit operating system, x64-based processor
GPU / GPU-Memory	NVIDIA GeForce GTX 1650 / 4 GB

Table 1: Hardware Configuration Details

2.2 Software Specifications

The below table shows the software specifications of the software used to undergo this research.

Categories	Version/Software Details
Operating System	Windows 11 Pro
Programming Language	Python 3.8.8
Framework	Jupyter-Notebook (Anaconda)
Libraries & Packages	Pytorch, Scikit-Learn, tqdm, Matplotlib, NumPy, Torch, Torchvision & others.
Report	Overleaf (LaTeX)
Browser	Microsoft Edge

Table 2: Software Specifications Details

3 Data Import

The code below shows the imports required to load the data into the jupyter notebook used to undergo the research.

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset
import torchvision.transforms as transforms
from PIL import Image
import numpy as np
import h5py
from tqdm import tqdm
import timm
import random
from torch.cuda.amp import GradScaler, autocast
from typing import List, Tuple
```

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

# random seed
random_seed = 42
np.random.seed(random_seed)
torch.manual_seed(random_seed)
random.seed(random_seed)

# Load HDF5 data
file_path = "Galaxy10_DECals.h5"
with h5py.File(file_path, 'r') as F:
    images = np.array(F['images'])
    labels = np.array(F['ans'])

# Train-validation-test split
galaxys = images.shape[0]
train_size = int(0.7 * galaxys)
valid_size = int(0.1 * galaxys)
indices = np.random.permutation(galaxys)
train_indices = indices[:train_size]
valid_indices = indices[train_size:train_size + valid_size]
test_indices = indices[train_size + valid_size:]
```

3.1 Loading Data Without Data Augmentation

In this section, the below code shows how the data is loaded without any augmentation except resizing the images as 224x224 for ease of model input.

```
# Dataset class

class GalaDataset(Dataset):
    def __init__(self, images: np.ndarray,
    labels: np.ndarray, indices: List[int], transforms) -> None:
        super().__init__()
        self.images = images
        self.labels = labels
        self.indices = indices
        self.transforms = transforms

    def __getitem__(self, index: int) -> Tuple[torch.Tensor, int]:
        idx = self.indices[index]
        image = self.images[idx]
        image = Image.fromarray(image)
        label = self.labels[idx]
        if self.transforms is not None:
            image = self.transforms(image)
        return (image, label)
```

```
    def __len__(self) -> int:
        return len(self.indices)


# Data transformations

input_size = 224

train_transforms = transforms.Compose([
    transforms.RandomResizedCrop(input_size, scale=(0.8, 1.0),
    ratio=(0.99, 1.01)),
    transforms.ToTensor()
])

valid_transforms = transforms.Compose([
    transforms.CenterCrop(input_size),
    transforms.ToTensor()
])

test_transforms = transforms.Compose([
    transforms.CenterCrop(input_size),
    transforms.ToTensor()
])


# dataloaders

train_dataset = GalaDataset(images=images, labels=labels,
indices=train_indices, transforms=train_transforms)
valid_dataset = GalaDataset(images=images, labels=labels,
indices=valid_indices, transforms=valid_transforms)
test_dataset = GalaDataset(images=images, labels=labels,
indices=test_indices, transforms=test_transforms)


# batch size

batch_size = 8
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
valid_loader = DataLoader(valid_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

3.2 Loading Data With Data Augmentation

The code below shows the transformations included for the data augmentation while loading the data.

```
    train_transforms = transforms.Compose([
    transforms.CenterCrop(input_size),
    transforms.RandomRotation(90),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomResizedCrop(input_size, scale=(0.8, 1.0), ratio=(0.99, 1.01)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

valid_transforms = transforms.Compose([
    transforms.CenterCrop(input_size),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

test_transforms = transforms.Compose([
    transforms.CenterCrop(input_size),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

4 Visualization

The below code shows the class distribution as per the number of instances available for each class within the dataset. The plot is shown in the figure 1.

```
import h5py
import numpy as np
import matplotlib.pyplot as plt

def plot_class_distribution(file_path):
    with h5py.File(file_path, 'r') as f:
        labels = f['ans'][:]

    unique, counts = np.unique(labels, return_counts=True)
    plt.figure(figsize=(10, 5))
    plt.bar(unique, counts, tick_label=unique)
    plt.xlabel('Class Labels')
    plt.ylabel('Frequency')
    plt.title("Class Distribution")
    plt.tight_layout()
    plt.show()

plot_class_distribution(file_path)
```

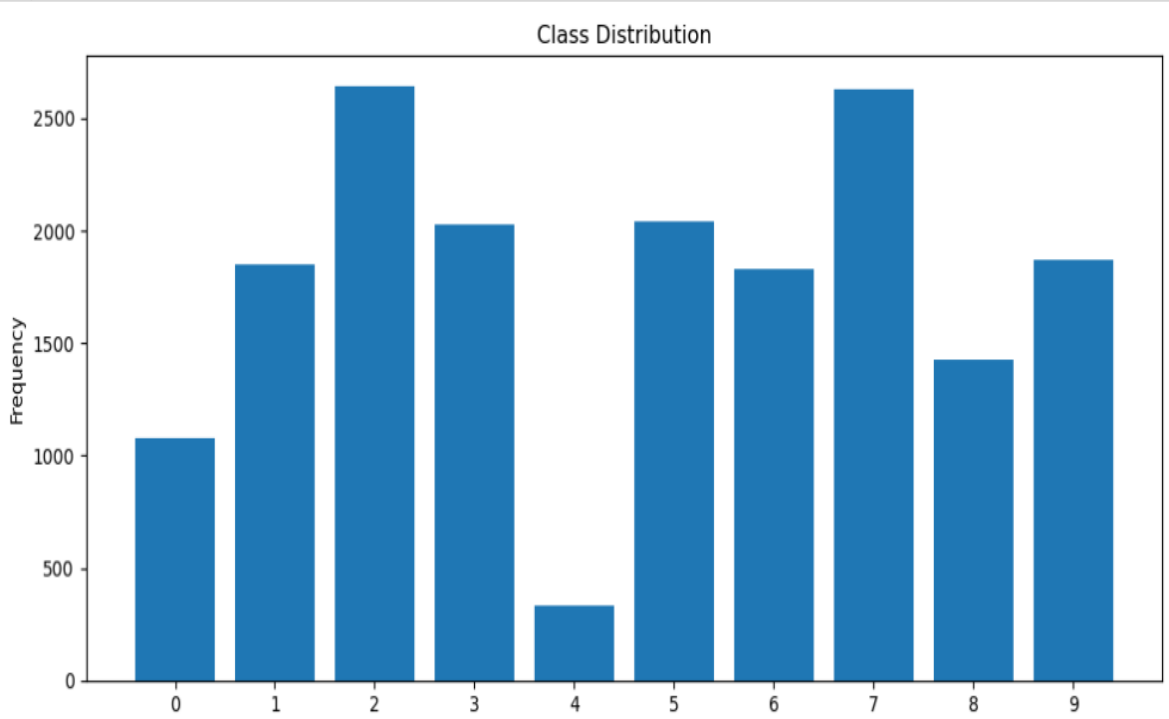

Figure 1: Class Distribution Diagram

5 Swin Transformer Model Training

The Code below shows how the Swin Transformer model is loaded and then trained for two cases - data with augmentation and data without augmentation. The figure 2 shows the sample output of one of the training among the two cases.

```
import timm
from tqdm import tqdm
from torchvision import models


model = timm.create_model('swin_base_patch4_window7_224',
        pretrained=True, num_classes=10)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
criterion = torch.nn.CrossEntropyLoss()
scaler = GradScaler()

def train_model(model, train_loader, criterion, optimizer, device,
                accumulation_steps=4):
    model.train()
    running_loss = 0.0
```

```
    correct = 0
    total = 0
    optimizer.zero_grad()

    for i, (inputs, labels) in enumerate(tqdm(train_loader)):
        inputs, labels = inputs.to(device), labels.to(device)

        with autocast():
            outputs = model(inputs)
            loss = criterion(outputs, labels)

        scaler.scale(loss).backward()

        if (i + 1) % accumulation_steps == 0:
            scaler.step(optimizer)
            scaler.update()
            optimizer.zero_grad()

        running_loss += loss.item() * inputs.size(0)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    epoch_loss = running_loss / len(train_loader.dataset)
    epoch_acc = correct / total

    return epoch_loss, epoch_acc

def validate_model(model, va, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in tqdm(test_loader):
            inputs, labels = inputs.to(device), labels.to(device)

            with autocast():
                outputs = model(inputs)
                loss = criterion(outputs, labels)

            running_loss += loss.item() * inputs.size(0)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    epoch_loss = running_loss / len(test_loader.dataset)
    epoch_acc = correct / total
```

```
    return epoch_loss, epoch_acc

# Free up memory
torch.cuda.empty_cache()

num_epochs = 10
start_time = time.time()
for epoch in range(num_epochs):


    train_loss, train_acc = train_model(model, train_loader, criterion,
                            optimizer, device)
    val_loss, val_acc = validate_model(model, valid_loader, criterion,
                        device)

    print(f"Epoch {epoch+1}/{num_epochs}")
    print(f"Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f}")
    print(f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}")


total_end_time = time.time()
total_duration = total_end_time - start_time

print("Total Time to train: ",total_duration)


# Save trained model
torch.save(model.state_dict(), "swin_transformer_galaxy10_decals_withTime.pth")
```

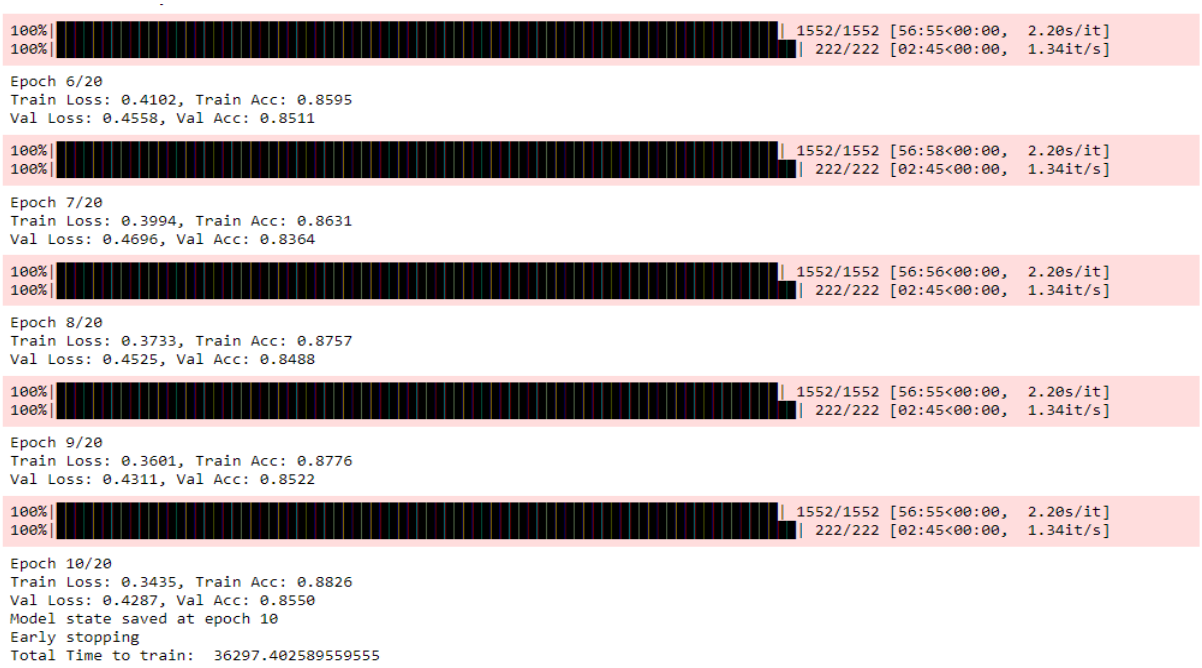

Figure 2: Swin Transformer Model training Sample Output

6 ViT Base Fine-tuned Model Training

The code below shows the implementation of the ViT Base Model which is also fine-tuned with hyperparameters along with the code for training it. The figure 3 shows the output obtained from the training.

```
import os
import time
from copy import deepcopy
import torch
import torch.nn as nn
import torch.optim as optim
from transformers import ViTModel, ViTConfig, ViTFeatureExtractor,
from transformers import ViTForImageClassification
from torch.optim.lr_scheduler import StepLR
# Hyperparameters
LR = 5e-5
STEP_SIZE = 5
GAMMA = 0.1
MAX_EPOCH = 50
class_weights = torch.FloatTensor([1., 1., 1., 1.,
                                   1., 1., 1., 1., 1., 1.]).to(device)
model_name = "ViT_without_augment"

feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-base-patch16-224'
model = ViTForImageClassification.from_pretrained('google/vit-base-patch16-224')
```

```
model.classifier = nn.Linear(in_features=768, out_features=10, bias=True)

print("Number of trainable parameters: {}".format(sum(param.numel()
      for param in model.parameters() if param.requires_grad)))

model.to(device)

criterion = nn.CrossEntropyLoss(weight=class_weights)
optimizer = optim.Adam(model.parameters(), lr=LR)
scheduler = StepLR(optimizer, step_size=STEP_SIZE, gamma=GAMMA)

def train_model(model, num_epochs, criterion, optimizer, scheduler,
                print_every=1, early_stop_epochs=10):
    best_model_weights = deepcopy(model.state_dict())
    best_train_acc = 0.0
    best_valid_acc = 0.0
    best_epoch = -1

    history_dic = {
        'train_loss': [],
        'train_acc': [],
        'valid_loss': [],
        'valid_acc': [],
        'lr': []
    }

    for epoch in range(num_epochs):
        epoch_start_time = time.time()

        # Train
        model.train()
        epoch_train_cum_loss = 0.0
        epoch_train_cum_corrects = 0

        for images, labels in train_loader:
            images = images.to(device)
            labels = labels.long().to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs.logits, labels)
            _, pred_classes = torch.max(outputs.logits.detach(),
                                        dim=1)

            epoch_train_cum_loss += loss.item() * images.size(0)
            epoch_train_cum_corrects += torch.sum(pred_classes == labels.data).detach

            loss.backward()
```

```
        optimizer.step()

    # Eval
    model.eval()
    epoch_valid_cum_loss = 0.0
    epoch_valid_cum_corrects = 0

    for images, labels in valid_loader:
        images = images.to(device)
        labels = labels.long().to(device)

        with torch.no_grad():
            outputs = model(images)
            _, pred_classes = torch.max(outputs.logits.detach(), dim=1)
            loss = criterion(outputs.logits, labels)

            epoch_valid_cum_loss += loss.item() * images.size(0)
            epoch_valid_cum_corrects += torch.sum(pred_classes ==
                           labels.data).detach().to('cpu').item()

    # metrics
    train_loss = epoch_train_cum_loss / len(train_loader.dataset)
    train_acc = epoch_train_cum_corrects / len(train_loader.dataset)
    valid_loss = epoch_valid_cum_loss / len(valid_loader.dataset)
    valid_acc = epoch_valid_cum_corrects / len(valid_loader.dataset)

    history_dic['train_loss'].append(train_loss)
    history_dic['train_acc'].append(train_acc)
    history_dic['valid_loss'].append(valid_loss)
    history_dic['valid_acc'].append(valid_acc)
    history_dic['lr'].append(scheduler.get_last_lr()[0])

    # check if best acc
    if valid_acc > best_valid_acc:
        best_train_acc = train_acc
        best_valid_acc = valid_acc
        best_epoch = epoch + 1
        best_model_weights = deepcopy(model.state_dict())
        torch.save(model.state_dict(),
                   os.path.join('./', model_name + "_cache.pth"))

    epoch_end_time = time.time()
    epoch_time_used = epoch_end_time - epoch_start_time
    mm = epoch_time_used // 60
    ss = epoch_time_used % 60

    if (epoch + 1) % print_every == 0:
        if epoch == (best_epoch - 1):
```

```
                print(f"Epoch {epoch + 1}/{num_epochs}\tTrain loss:
                {train_loss:.4f}\tTrain acc: {train_acc:.4f}\tValid loss:
                {valid_loss:.4f}\tValid acc: {valid_acc:.4f}\tTime:
                {mm:.0f}m {ss:.0f}s\t<--")
            else:
                print(f"Epoch {epoch + 1}/{num_epochs}\tTrain loss:
                {train_loss:.4f}\tTrain acc: {train_acc:.4f}\tValid loss:
                {valid_loss:.4f}\tValid acc: {valid_acc:.4f}\tTime:
                {mm:.0f}m {ss:.0f}s")

        if (epoch + 1) - best_epoch >= early_stop_epochs:
            print(f"Early stopping... (Model did not improve after
            {early_stop_epochs} epochs)")
            break

        scheduler.step()

    model.load_state_dict(best_model_weights)
    print(f"Best epoch = {best_epoch}, with training accuracy = {best_train_acc:.4f}

    return model, history_dic


trained_model, training_history = train_model(model, MAX_EPOCH, criterion,
                                              optimizer, scheduler)
```

```
Epoch 1/200     Train loss: 0.7762      Train acc: 0.7331       Valid loss: 0.6246      Valid acc: 0.7940       Time: 18m 45s
<--
Epoch 2/200     Train loss: 0.5556      Train acc: 0.8135       Valid loss: 0.5113      Valid acc: 0.8233       Time: 18m 44s
<--
Epoch 3/200     Train loss: 0.4948      Train acc: 0.8284       Valid loss: 0.5426      Valid acc: 0.8196       Time: 18m 46s
Epoch 4/200     Train loss: 0.4518      Train acc: 0.8432       Valid loss: 0.5075      Valid acc: 0.8272       Time: 18m 47s
<--
Epoch 5/200     Train loss: 0.4245      Train acc: 0.8510       Valid loss: 0.5282      Valid acc: 0.8182       Time: 18m 45s
Epoch 6/200     Train loss: 0.2976      Train acc: 0.8981       Valid loss: 0.4187      Valid acc: 0.8582       Time: 18m 47s
<--
Epoch 7/200     Train loss: 0.2676      Train acc: 0.9066       Valid loss: 0.4100      Valid acc: 0.8636       Time: 18m 46s
<--
Epoch 8/200     Train loss: 0.2448      Train acc: 0.9141       Valid loss: 0.4151      Valid acc: 0.8622       Time: 18m 48s
Epoch 9/200     Train loss: 0.2384      Train acc: 0.9153       Valid loss: 0.4131      Valid acc: 0.8610       Time: 18m 48s
Epoch 10/200    Train loss: 0.2247      Train acc: 0.9221       Valid loss: 0.4146      Valid acc: 0.8641       Time: 18m 49s
<--
Epoch 11/200    Train loss: 0.2072      Train acc: 0.9281       Valid loss: 0.4078      Valid acc: 0.8647       Time: 18m 48s
<--
Epoch 12/200    Train loss: 0.1992      Train acc: 0.9327       Valid loss: 0.4070      Valid acc: 0.8658       Time: 18m 45s
<--
Epoch 13/200    Train loss: 0.1985      Train acc: 0.9292       Valid loss: 0.4080      Valid acc: 0.8661       Time: 18m 46s
<--
Epoch 14/200    Train loss: 0.1989      Train acc: 0.9322       Valid loss: 0.4083      Valid acc: 0.8670       Time: 18m 49s
<--
Epoch 15/200    Train loss: 0.2026      Train acc: 0.9283       Valid loss: 0.4083      Valid acc: 0.8667       Time: 18m 46s
Epoch 16/200    Train loss: 0.1922      Train acc: 0.9345       Valid loss: 0.4083      Valid acc: 0.8664       Time: 18m 46s
Epoch 17/200    Train loss: 0.1968      Train acc: 0.9309       Valid loss: 0.4083      Valid acc: 0.8661       Time: 18m 46s
Epoch 18/200    Train loss: 0.1952      Train acc: 0.9324       Valid loss: 0.4082      Valid acc: 0.8664       Time: 18m 50s
Epoch 19/200    Train loss: 0.1927      Train acc: 0.9307       Valid loss: 0.4078      Valid acc: 0.8661       Time: 18m 53s
Epoch 20/200    Train loss: 0.1975      Train acc: 0.9323       Valid loss: 0.4077      Valid acc: 0.8661       Time: 18m 49s
Epoch 21/200    Train loss: 0.1946      Train acc: 0.9329       Valid loss: 0.4077      Valid acc: 0.8661       Time: 18m 51s
Epoch 22/200    Train loss: 0.1945      Train acc: 0.9332       Valid loss: 0.4077      Valid acc: 0.8661       Time: 18m 51s
Epoch 23/200    Train loss: 0.1914      Train acc: 0.9356       Valid loss: 0.4077      Valid acc: 0.8664       Time: 18m 52s
Epoch 24/200    Train loss: 0.1971      Train acc: 0.9301       Valid loss: 0.4077      Valid acc: 0.8664       Time: 18m 51s
Early stopping... (Model did not improve after 10 epochs)
Best epoch = 14, with training accuracy = 0.9322 and validation accuracy = 0.8670
```

Figure 3: ViT Base Fine-Tuned Model training Sample Output

7 DeiT Model Training

The Below code explain the way to load the Data Efficient Image Transformer Model and then training it while checking the time to train the whole model. The figure 4 shows the sample output of the model after training.

```
model = timm.create_model('deit_base_patch16_224', pretrained=True,
                          num_classes=10)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
criterion = torch.nn.CrossEntropyLoss()
scaler = GradScaler()

early_stopping = EarlyStopping(patience=5, min_delta=0.01,
                               path='deit_best_without_augment.pth')

def train_model(model, train_loader, criterion, optimizer, device,
                accumulation_steps=4):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    optimizer.zero_grad()

    for i, (inputs, labels) in enumerate(tqdm(train_loader)):
        inputs, labels = inputs.to(device), labels.to(device)

        with autocast():
            outputs = model(inputs)
            loss = criterion(outputs, labels)

        scaler.scale(loss).backward()

        if (i + 1) % accumulation_steps == 0:
            scaler.step(optimizer)
            scaler.update()
            optimizer.zero_grad()

        running_loss += loss.item() * inputs.size(0)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    epoch_loss = running_loss / len(train_loader.dataset)
    epoch_acc = correct / total
```

```
    return epoch_loss, epoch_acc

def validate_model(model, test_loader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in tqdm(test_loader):
            inputs, labels = inputs.to(device), labels.to(device)

            with autocast():
                outputs = model(inputs)
                loss = criterion(outputs, labels)

            running_loss += loss.item() * inputs.size(0)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    epoch_loss = running_loss / len(test_loader.dataset)
    epoch_acc = correct / total

    return epoch_loss, epoch_acc

torch.cuda.empty_cache()
train_losses = []
train_accuracies = []
val_losses = []
val_accuracies = []

num_epochs = 20
start_time = time.time()
for epoch in range(num_epochs):
    train_loss, train_acc = train_model(model, train_loader, criterion,
                                        optimizer, device)
    val_loss, val_acc = validate_model(model, valid_loader, criterion, device)

    train_losses.append(train_loss)
    train_accuracies.append(train_acc)
    val_losses.append(val_loss)
    val_accuracies.append(val_acc)

    print(f"Epoch {epoch+1}/{num_epochs}")
    print(f"Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f}")
    print(f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}")

    early_stopping(val_loss, model)
```

```
    if early_stopping.early_stop:
        print("Early stopping")
        break

total_end_time = time.time()
total_duration = total_end_time - start_time
print("Total Time to train in Hours: ")
print(datetime.fromtimestamp(total_duration).strftime('%H:%M:%S'))
```

```
100%|| 1552/1552 [55:29<00:00,  2.15s/it]
100%|| 222/222 [02:39<00:00,  1.39it/s]
Epoch 7/20
Train Loss: 0.3130, Train Acc: 0.8901
Val Loss: 0.5430, Val Acc: 0.8257
100%|| 1552/1552 [55:29<00:00,  2.15s/it]
100%|| 222/222 [02:39<00:00,  1.39it/s]
Epoch 8/20
Train Loss: 0.2675, Train Acc: 0.9066
Val Loss: 0.6163, Val Acc: 0.8127
100%|| 1552/1552 [55:29<00:00,  2.15s/it]
100%|| 222/222 [02:39<00:00,  1.39it/s]
Epoch 9/20
Train Loss: 0.2381, Train Acc: 0.9188
Val Loss: 0.6472, Val Acc: 0.8105
100%|| 1552/1552 [55:36<00:00,  2.15s/it]
100%|| 222/222 [02:41<00:00,  1.37it/s]
Epoch 10/20
Train Loss: 0.2110, Train Acc: 0.9288
Val Loss: 0.7219, Val Acc: 0.8116
Early stopping
Total Time to train:  34971.63448596001
```

Figure 4: DeiT Model training sample output

8 ResNet50 Model Training

The below code shows the implementation of the ResNet50 model and then training it. The figure 5 shows the output sample for the training of the model.

```
LR = 1e-5
STEP_SIZE = 10
MAX_EPOCH = 50
NUM_OF_CLASSES = 10

model_name = "resnet50_without_augment"
model = models.resnet50(pretrained=True)

# Change last fc layer
model.fc = nn.Linear(2048, NUM_OF_CLASSES)
print(model.fc)
```

```
print("============")
print("Training")
print("Number of trainable parameters: )
print("{}".format(sum(param.numel() for param in model.parameters() if
                      param.requires_grad)))
print("============")
# GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
# Loss function
criterion = nn.CrossEntropyLoss()
# Optimizer
optimizer = optim.Adam(model.parameters())
# Scheduler
scheduler = StepLR(optimizer, step_size=STEP_SIZE)

# Training function
def train_model(model, num_epochs, criterion, optimizer, scheduler,
print_every=1, early_stop_epochs=5):
    """
    Train the model
    Args:
        model: Pytorch neural model
        num_epochs: number of epochs to train
        criterion: the loss function object
        optimizer: the optimizer
        scheduler: the learning rate decay scheduler
        print_every: print the information every X epochs
        early_stop_epochs: early stopping
    """
    # Cache the best model
    best_model_weights = deepcopy(model.state_dict())
    best_train_acc = 0.0
    best_valid_acc = 0.0
    best_epoch = -1

    history_dic = {
        'train_loss': [],
        'train_acc': [],
        'valid_loss': [],
        'valid_acc': [],
        'lr': []
    }

    for epoch in range(num_epochs):
        epoch_start_time = time.time()
```

```
# Train
model.train()
epoch_train_cum_loss = 0.0
epoch_train_cum_corrects = 0

for images, labels in train_loader:
    images = images.to(device)
    labels = labels.long().to(device)

    optimizer.zero_grad()
    outputs = model(images)
    loss = criterion(outputs, labels)
    _, pred_classes = torch.max(outputs.detach(), dim=1)

    epoch_train_cum_loss += loss.item() * images.size(0)
    epoch_train_cum_corrects += torch.sum(pred_classes ==
    labels.data).detach().to('cpu').item()

    loss.backward()
    optimizer.step()

# Eval
model.eval()
epoch_valid_cum_loss = 0.0
epoch_valid_cum_corrects = 0

for images, labels in test_loader:
    images = images.to(device)
    labels = labels.long().to(device)

    with torch.no_grad():
        outputs = model(images)
        _, pred_classes = torch.max(outputs.detach(), dim=1)
        loss = criterion(outputs, labels)

        epoch_valid_cum_loss += loss.item() * images.size(0)
        epoch_valid_cum_corrects += torch.sum(pred_classes ==
        labels.data).detach().to('cpu').item()

# metrics
train_loss = epoch_train_cum_loss / len(train_loader.dataset)
train_acc = epoch_train_cum_corrects / len(train_loader.dataset)
valid_loss = epoch_valid_cum_loss / len(test_loader.dataset)
valid_acc = epoch_valid_cum_corrects / len(test_loader.dataset)

# Update history_dic
history_dic['train_loss'].append(train_loss)
history_dic['train_acc'].append(train_acc)
```

```
            history_dic['valid_loss'].append(valid_loss)
            history_dic['valid_acc'].append(valid_acc)
            history_dic['lr'].append(scheduler.get_last_lr()[0])

            # Check if best acc
            if valid_acc > best_valid_acc:
                best_train_acc = train_acc
                best_valid_acc = valid_acc
                best_epoch = epoch + 1
                best_model_weights = deepcopy(model.state_dict())
                torch.save(model.state_dict(), os.path.join('./',
                           model_name + "_new_cache.pth"))

            epoch_end_time = time.time()
            epoch_time_used = epoch_end_time - epoch_start_time
            mm = epoch_time_used // 60
            ss = epoch_time_used % 60

            if (epoch + 1) % print_every == 0:
                if epoch == (best_epoch - 1):
                    print(f"Epoch {epoch + 1}/{num_epochs}\tTrain loss:
                    {train_loss:.4f}\tTrain acc: {train_acc:.4f}\tValid loss:
                    {valid_loss:.4f}\tValid acc: {valid_acc:.4f}\tTime:
                    {mm:.0f}m {ss:.0f}s\t<--")
                else:
                    print(f"Epoch {epoch + 1}/{num_epochs}\tTrain loss:
                    {train_loss:.4f}\tTrain acc: {train_acc:.4f}\tValid loss:
                    {valid_loss:.4f}\tValid acc: {valid_acc:.4f}\tTime:
                    {mm:.0f}m {ss:.0f}s")

            if (epoch + 1) - best_epoch >= early_stop_epochs:
                print(f"Early stopping... (Model did not improve after
                {early_stop_epochs} epochs)")
                break

            scheduler.step()

        model.load_state_dict(best_model_weights)
        print(f"Best epoch = {best_epoch}, with training accuracy = {best_train_acc:.4f}
        print("validation accuracy = {best_valid_acc:.4f}")

        return model, history_dic
    resnet_start_time = time.time()

    trained_model, training_history = train_model(model, MAX_EPOCH, criterion,
                                      optimizer, scheduler)
    resnet_end_time = time.time()
```

```
Epoch 13/50     Train loss: 0.4995      Train acc: 0.8246       Valid loss: 0.5772      Valid acc: 0.8041       Time: 5m 18s
Epoch 14/50     Train loss: 0.4858      Train acc: 0.8330       Valid loss: 0.5792      Valid acc: 0.8038       Time: 5m 20s
Epoch 15/50     Train loss: 0.4727      Train acc: 0.8336       Valid loss: 0.5589      Valid acc: 0.8069       Time: 5m 18s
<--
Epoch 16/50     Train loss: 0.4599      Train acc: 0.8371       Valid loss: 0.5665      Valid acc: 0.8120       Time: 5m 18s
<--
Epoch 17/50     Train loss: 0.4508      Train acc: 0.8432       Valid loss: 0.5690      Valid acc: 0.8112       Time: 5m 28s
Epoch 18/50     Train loss: 0.4406      Train acc: 0.8459       Valid loss: 0.5624      Valid acc: 0.8098       Time: 5m 22s
Epoch 19/50     Train loss: 0.4341      Train acc: 0.8466       Valid loss: 0.5721      Valid acc: 0.8075       Time: 5m 22s
Epoch 20/50     Train loss: 0.4250      Train acc: 0.8511       Valid loss: 0.5603      Valid acc: 0.8100       Time: 5m 30s
Epoch 21/50     Train loss: 0.3925      Train acc: 0.8609       Valid loss: 0.5536      Valid acc: 0.8160       Time: 5m 22s
<--
Epoch 22/50     Train loss: 0.3880      Train acc: 0.8646       Valid loss: 0.5569      Valid acc: 0.8148       Time: 5m 22s
Epoch 23/50     Train loss: 0.3913      Train acc: 0.8627       Valid loss: 0.5526      Valid acc: 0.8134       Time: 5m 30s
Epoch 24/50     Train loss: 0.3882      Train acc: 0.8627       Valid loss: 0.5535      Valid acc: 0.8174       Time: 5m 22s
<--
Epoch 25/50     Train loss: 0.3859      Train acc: 0.8650       Valid loss: 0.5536      Valid acc: 0.8129       Time: 5m 22s
Epoch 26/50     Train loss: 0.3884      Train acc: 0.8637       Valid loss: 0.5525      Valid acc: 0.8157       Time: 5m 29s
Epoch 27/50     Train loss: 0.3853      Train acc: 0.8662       Valid loss: 0.5480      Valid acc: 0.8168       Time: 5m 22s
Epoch 28/50     Train loss: 0.3799      Train acc: 0.8648       Valid loss: 0.5516      Valid acc: 0.8185       Time: 5m 23s
<--
Epoch 29/50     Train loss: 0.3806      Train acc: 0.8698       Valid loss: 0.5536      Valid acc: 0.8162       Time: 5m 30s
Epoch 30/50     Train loss: 0.3828      Train acc: 0.8656       Valid loss: 0.5498      Valid acc: 0.8182       Time: 5m 22s
Epoch 31/50     Train loss: 0.3754      Train acc: 0.8702       Valid loss: 0.5475      Valid acc: 0.8179       Time: 5m 31s
Epoch 32/50     Train loss: 0.3735      Train acc: 0.8688       Valid loss: 0.5528      Valid acc: 0.8185       Time: 5m 23s
Epoch 33/50     Train loss: 0.3758      Train acc: 0.8677       Valid loss: 0.5470      Valid acc: 0.8176       Time: 5m 22s
Early stopping... (Model did not improve after 5 epochs)
Best epoch = 28, with training accuracy = 0.8648 and validation accuracy = 0.8185
```

Figure 5: ResNet50 Model training sample output

9 Evaluation

The below code shows the way the evaluation was done for the models. And the below figure 6 shows a sample output for the confusion matrix generated

```python
import torch
import torch.nn as nn
from torch.cuda.amp import autocast
import timm
import numpy as np
from tqdm import tqdm
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Swin Transformer
model = timm.create_model('swin_base_patch4_window7_224',
pretrained=False, num_classes=10)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
model.load_state_dict(torch.load("swin_transformer_galaxy10_decals.pth"))
model.eval()

def evaluate_model(model, test_loader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
```

```
    total = 0
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for inputs, labels in tqdm(test_loader):
            inputs, labels = inputs.to(device), labels.to(device)

            with autocast():
                outputs = model(inputs)
                loss = criterion(outputs, labels)

            running_loss += loss.item() * inputs.size(0)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    epoch_loss = running_loss / len(test_loader.dataset)
    epoch_acc = correct / total

    return epoch_loss, epoch_acc, all_labels, all_preds

test_loss, test_acc, swin_all_labels, swin_all_preds = evaluate_model(model,
                                                test_loader, criterion, device)

print(f"Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.4f}")

conf_matrix = confusion_matrix(swin_all_labels, swin_all_preds)

def plot_confusion_matrix(cm, class_names):
    cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm_normalized, annot=True, fmt='.1%', cmap='YlGnBu',
                xticklabels=class_names, yticklabels=class_names)
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.title('Swin Transformer Confusion Matrix')
    plt.show()

class_names = [
    "Disturbed Galaxies",
    "Merging Galaxies",
    "Round Smooth Galaxies",
    "In-between Round Smooth Galaxies",
    "Cigar Shaped Smooth Galaxies",
```

```
    "Barred Spiral Galaxies",
    "Unbarred Tight Spiral Galaxies",
    "Unbarred Loose Spiral Galaxies",
    "Edge-on Galaxies without Bulge",
    "Edge-on Galaxies with Bulge"
]

plot_confusion_matrix(conf_matrix, class_names)
```

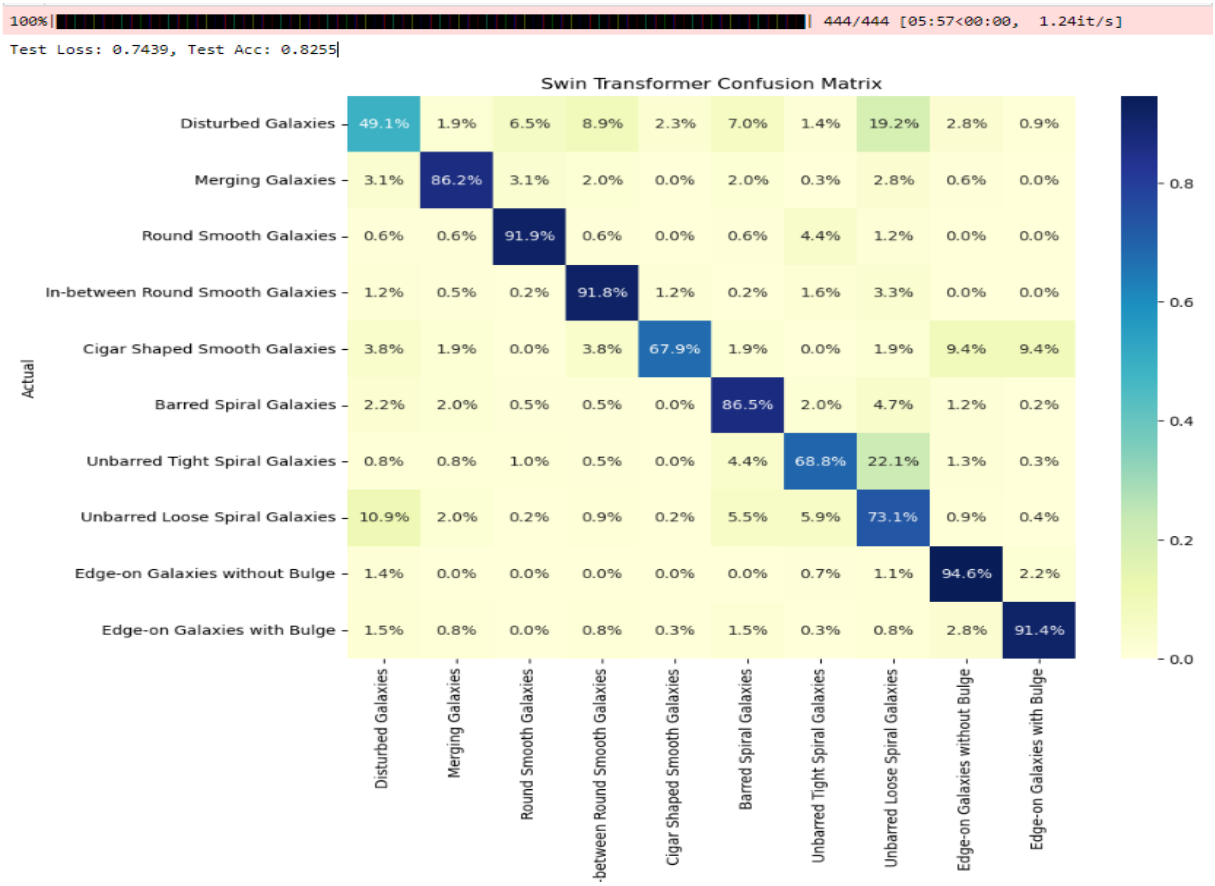

Figure 6: Sample Confusion Matrix and the output after evaluation

The below code generates the classification report as shown in figure 7.

```
print("ResNet50 Classification Report:",classification_report(all_labels,
                                        all_preds, target_names=class_names))
```

ResNet50 Classification Report:

	precision	recall	f1-score	support
Disturbed Galaxies	0.54	0.33	0.41	214
Merging Galaxies	0.87	0.86	0.87	354
Round Smooth Galaxies	0.88	0.96	0.92	496
In-between Round Smooth Galaxies	0.91	0.95	0.93	427
Cigar Shaped Smooth Galaxies	0.74	0.81	0.77	53
Barred Spiral Galaxies	0.82	0.86	0.84	401
Unbarred Tight Spiral Galaxies	0.75	0.84	0.79	385
Unbarred Loose Spiral Galaxies	0.76	0.67	0.71	543
Edge-on Galaxies without Bulge	0.90	0.90	0.90	279
Edge-on Galaxies with Bulge	0.89	0.93	0.91	396
accuracy			0.83	3548
macro avg	0.81	0.81	0.81	3548
weighted avg	0.82	0.83	0.82	3548

Figure 7: Sample Classification Report

9.1 Model Comparisons

The below code shows how the model comparisons were plotted for each and every classes based on Precision, Recall, F1 score and Accuracy. The below figure 8 shows the sample output.

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score


def compute_class_metrics(all_labels, all_preds, class_names):
    precision = precision_score(all_labels, all_preds, average=None,
    labels=np.arange(len(class_names)))
    recall = recall_score(all_labels, all_preds, average=None,
    labels=np.arange(len(class_names)))
    f1 = f1_score(all_labels, all_preds, average=None,
    labels=np.arange(len(class_names)))
    accuracy = accuracy_score(all_labels, all_preds)

    return precision, recall, f1, accuracy


def plot_metrics(models, metric_name, metric_index, class_names):
    metric_data = {model_name: compute_class_metrics(labels, press,
                                        class_names)[metric_index]
                   for model_name, (labels, preds) in models.items()}

    metrics_df = pd.DataFrame(metric_data, index=class_names)

    plt.figure(figsize=(14, 8))
```

```
    sns.heatmap(metrics_df, annot=True, cmap='YlGnBu', fmt='.2f',
                annot_kws={"size": 16})
    plt.title(f'Comparison of Models Based on {metric_name}', fontsize=20)
    plt.ylabel('Classes', fontsize=20)
    plt.xlabel('Models', fontsize=20)
    plt.xticks(rotation=45, ha="right", fontsize=16)
    plt.yticks(fontsize=16)
    plt.show()

models = {
    "Swin Transformer": (swin_all_labels, swin_all_preds),
    "ViT Base Fine-Tuned": (vit_all_labels, vit_all_preds),
    "DeiT": (deit_all_labels, deit_all_preds),
    "ResNet50": (resnet_all_labels, resnet_all_preds)
}

class_names = [
    "Disturbed Galaxies",
    "Merging Galaxies",
    "Round Smooth Galaxies",
    "In-between Round Smooth Galaxies",
    "Cigar Shaped Smooth Galaxies",
    "Barred Spiral Galaxies",
    "Unbarred Tight Spiral Galaxies",
    "Unbarred Loose Spiral Galaxies",
    "Edge-on Galaxies without Bulge",
    "Edge-on Galaxies with Bulge"
]

import pandas as pd

plot_metrics(models, "Precision", 0, class_names)

plot_metrics(models, "Recall", 1, class_names)

plot_metrics(models, "F1 Score", 2, class_names)

# Accuracy for each model
accuracies = {model_name: compute_class_metrics(labels, preds,
class_names)[3] for model_name, (labels, preds) in models.items()}
accuracy_df = pd.DataFrame.from_dict(accuracies, orient='index',
                                     columns=['Accuracy'])

# Accuracy PLot
plt.figure(figsize=(10, 6))
sns.barplot(x=accuracy_df.index, y='Accuracy', data=accuracy_df)
plt.title('Comparison of Models Based on Accuracy', fontsize=20)
plt.xlabel('Models', fontsize=20)
```

```
plt.ylabel('Accuracy', fontsize=20)
plt.xticks(rotation=45, ha="right", fontsize=16)
plt.yticks(fontsize=16)
plt.ylim(0, 1)
plt.show()
```

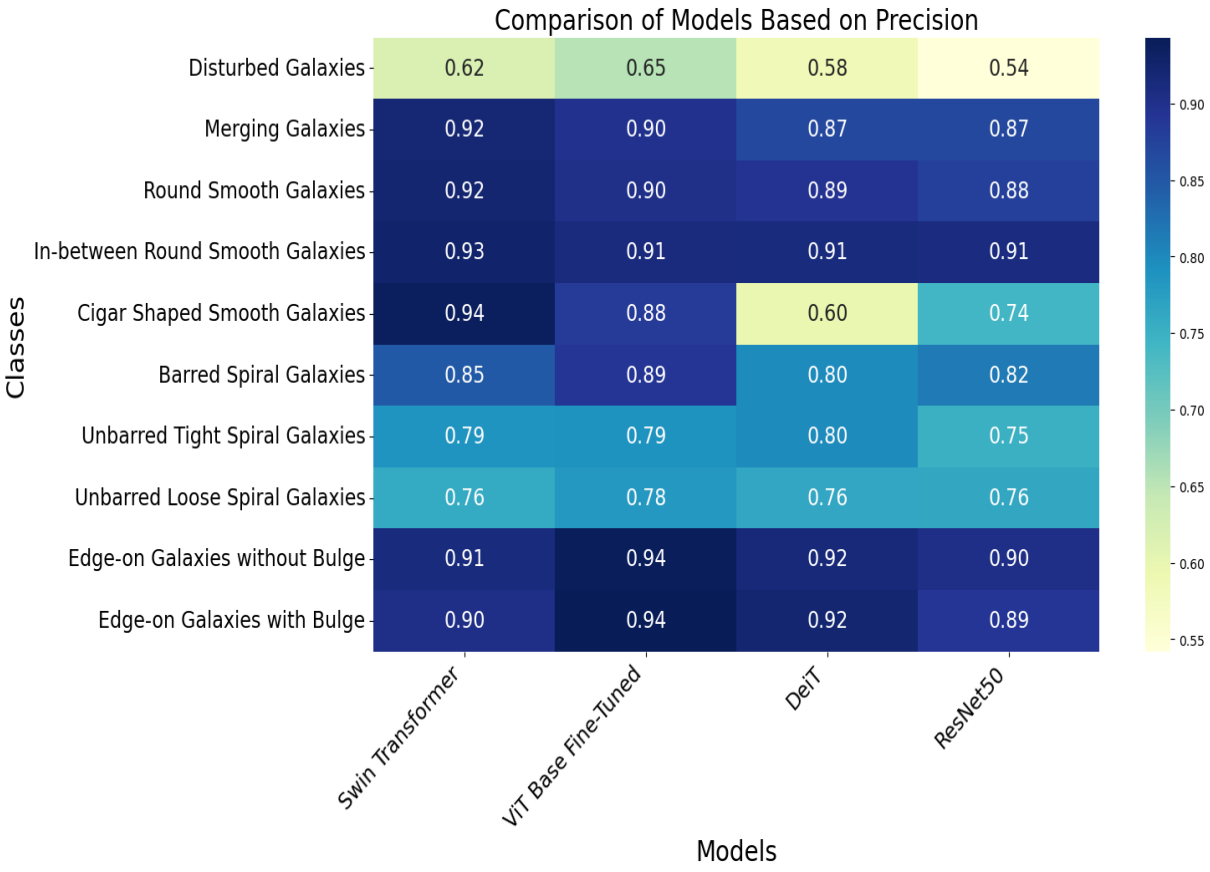

Figure 8: Sample Model Comparisons as Per Precision