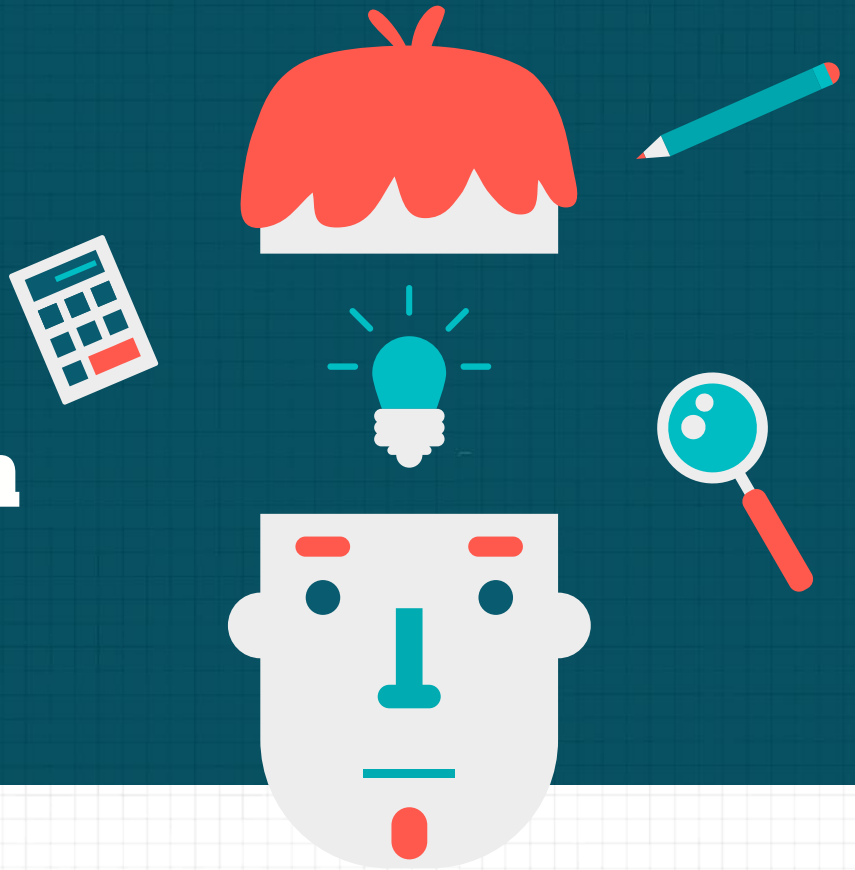
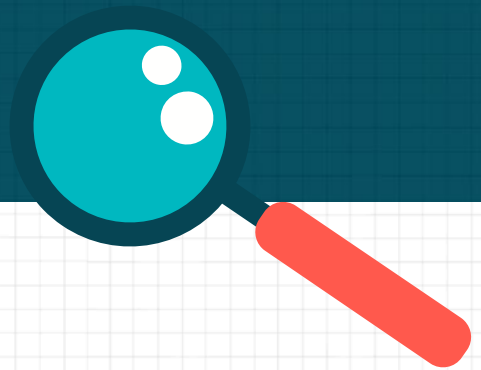
GROUP 3

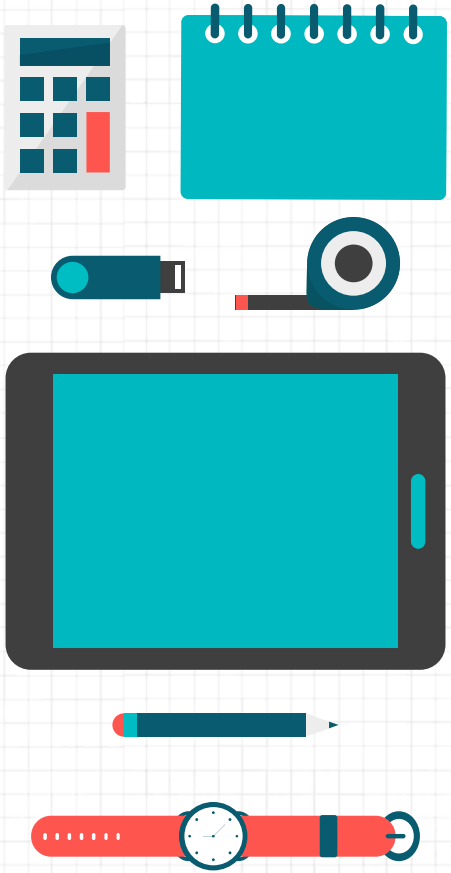
Deep Learning Model for Image Classification



DB927491 Mega, ANG JIAN HWEE
DB927482 EMILY CHOO HUE CHE

Contents





01

Preprocessing

1.1 Import Dataset and Train Test Split

Import Library and set a constant seed

```
import os
import shutil
import random
import time

import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from torchvision import transforms, datasets
import torchvision.models as models

# Set SEED
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

1.1 Import Dataset and Train Test Split

Import Library and set a constant seed

```
import os
import shutil
import random
import time

import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from torchvision import transforms, datasets
import torchvision.models as models

# Set SEED
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

1.1 Import Dataset and Train Test Split

Transform data using PyTorch

```
train_ratio = 0.8
test_ratio = 0.2
# validation_ratio_within_train = 0.2 # validation data: 0.2*0.8

train_path = r'./train/'
test_path = r'./test/'
data_location = r'/kaggle/input/satellite-image-classification/data'
```

Get class name

```
class_name = os.listdir(data_location)
print(f"Class: {class_name}\n")
```

```
Class: ['cloudy', 'desert', 'green_area', 'water']
```

Create Folder for training and testing data

```
try:
  os.mkdir(train_path)
  os.mkdir(test_path)
except FileExistsError:
  pass

for x in class_name:
  try:
    os.mkdir(os.path.join(train_path,x))
    os.mkdir(os.path.join(test_path,x))
  except FileExistsError:
    pass

for dirname, _, filenames in os.walk('./'):
    print(dirname)
```

```
./
./.virtual_documents
./test
./test/green_area
./test/cloudy
./test/desert
./test/water
./train
./train/green_area
./train/cloudy
./train/desert
./train/water
```

1.1 Import Dataset and Train Test Split

Transform data using PyTorch

```
train_ratio = 0.8
test_ratio = 0.2
# validation_ratio_within_train = 0.2 # validation data: 0.2*0.8

train_path = r'./train/'
test_path = r'./test/'
data_location = r'/kaggle/input/satellite-image-classification/data'
```

Get class name

```
class_name = os.listdir(data_location)
print(f"Class: {class_name}\n")
```

```
Class: ['cloudy', 'desert', 'green_area', 'water']
```

Create Folder for training and testing data

```
try:
  os.mkdir(train_path)
  os.mkdir(test_path)
except FileExistsError:
  pass

for x in class_name:
  try:
    os.mkdir(os.path.join(train_path,x))
    os.mkdir(os.path.join(test_path,x))
  except FileExistsError:
    pass

for dirname, _, filenames in os.walk('./'):
    print(dirname)
```

```
./
./.virtual_documents
./test
./test/green_area
./test/cloudy
./test/desert
./test/water
./train
./train/green_area
./train/cloudy
./train/desert
./train/water
```

1.1 Import Dataset and Train Test Split

Transform data using PyTorch

```
train_ratio = 0.8
test_ratio = 0.2
# validation_ratio_within_train = 0.2 # validation data: 0.2*0.8

train_path = r'./train/'
test_path = r'./test/'
data_location = r'/kaggle/input/satellite-image-classification/data'
```

Get class name

```
class_name = os.listdir(data_location)
print(f"Class: {class_name}\n")
```

```
Class: ['cloudy', 'desert', 'green_area', 'water']
```

Create Folder for training and testing data

```
try:
  os.mkdir(train_path)
  os.mkdir(test_path)
except FileExistsError:
  pass

for x in class_name:
  try:
    os.mkdir(os.path.join(train_path,x))
    os.mkdir(os.path.join(test_path,x))
  except FileExistsError:
    pass

for dirname, _, filenames in os.walk('./'):
    print(dirname)
```

```
./
./.virtual_documents
./test
./test/green_area
./test/cloudy
./test/desert
./test/water
./train
./train/green_area
./train/cloudy
./train/desert
./train/water
```

1.1 Import Dataset and Train Test Split

Transform data using PyTorch

```
train_ratio = 0.8
test_ratio = 0.2
# validation_ratio_within_train = 0.2 # validation data: 0.2*0.8

train_path = r'./train/'
test_path = r'./test/'
data_location = r'/kaggle/input/satellite-image-classification/data'
```

Get class name

```
class_name = os.listdir(data_location)
print(f"Class: {class_name}\n")
```

```
Class: ['cloudy', 'desert', 'green_area', 'water']
```

Create Folder for training and testing data

```
try:
  os.mkdir(train_path)
  os.mkdir(test_path)
except FileExistsError:
  pass

for x in class_name:
  try:
    os.mkdir(os.path.join(train_path,x))
    os.mkdir(os.path.join(test_path,x))
  except FileExistsError:
    pass

for dirname, _, filenames in os.walk('./'):
    print(dirname)
```

```
./
./.virtual_documents
./test
./test/green_area
./test/cloudy
./test/desert
./test/water
./train
./train/green_area
./train/cloudy
./train/desert
./train/water
```

1.1 Import Dataset and Train Test Split

Random Split and move images (Shuffle)

```
for x in class_name:
  cur_dir = os.path.join(data_location,x)
  print(f"{len(os.listdir(cur_dir))} image - {cur_dir}")

  train, test = train_test_split(os.listdir(cur_dir),train_size = 0.8,shuffle = True)

  for train_ in train:
    shutil.copyfile(os.path.join(cur_dir, train_),os.path.join(train_path, x, train_))

  for test_ in test:
    shutil.copyfile(os.path.join(cur_dir, test_),os.path.join(test_path, x, test_))

  print(f"train : {len(os.listdir(os.path.join(train_path,x)))} images")
  print(f"test  : {len(os.listdir(os.path.join(test_path,x)))} images")
  print()
```

```
1500 image - /kaggle/input/satellite-image-classification/data/cloudy
train : 1200 images
test  : 300 images

1131 image - /kaggle/input/satellite-image-classification/data/desert
train : 904 images
test  : 227 images

1500 image - /kaggle/input/satellite-image-classification/data/green_area
train : 1200 images
test  : 300 images

1500 image - /kaggle/input/satellite-image-classification/data/water
train : 1200 images
test  : 300 images
```

Use the shutil.copyfile() method to copy the split images to it relative folder

1.1 Import Dataset and Train Test Split

Random Split and move images (Shuffle)

```
for x in class_name:
  cur_dir = os.path.join(data_location,x)
  print(f"{len(os.listdir(cur_dir))} image - {cur_dir}")

  train, test = train_test_split(os.listdir(cur_dir),train_size = 0.8,shuffle = True)

  for train_ in train:
    shutil.copyfile(os.path.join(cur_dir, train_),os.path.join(train_path, x, train_))

  for test_ in test:
    shutil.copyfile(os.path.join(cur_dir, test_),os.path.join(test_path, x, test_))

  print(f"train : {len(os.listdir(os.path.join(train_path,x)))} images")
  print(f"test  : {len(os.listdir(os.path.join(test_path,x)))} images")
  print()
```

```
1500 image - /kaggle/input/satellite-image-classification/data/cloudy
train : 1200 images
test  : 300 images

1131 image - /kaggle/input/satellite-image-classification/data/desert
train : 904 images
test  : 227 images

1500 image - /kaggle/input/satellite-image-classification/data/green_area
train : 1200 images
test  : 300 images

1500 image - /kaggle/input/satellite-image-classification/data/water
train : 1200 images
test  : 300 images
```

Use the shutil.copyfile() method to copy the split images to it relative folder

1.2 Normalization & DataLoader

Normalize data

```
batch_size = 64
target_image_dim = 224
data_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize(target_image_dim),
    transforms.Normalize(mean=(0.4914, 0.4822, 0.4465),
                         std=(0.2023, 0.1994, 0.2010))
  ])
```

Prepare the dataset with datasets.ImageFolder() and use DataLoader() to load the data.

```
print(train_path)

train_dataset = datasets.ImageFolder(root=train_path,
                                     transform=data_transform)

train_dataset_loader = torch.utils.data.DataLoader(train_dataset,
                                                   batch_size=batch_size, shuffle=True,
                                                   num_workers=4)

print(train_dataset_loader.dataset.classes)
```

Output classes for training data and test data

```
./train/
['cloudy', 'desert', 'green_area', 'water']
./test/
['cloudy', 'desert', 'green_area', 'water']
```

1.2 Normalization & DataLoader

Normalize data

```
batch_size = 64
target_image_dim = 224
data_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize(target_image_dim),
    transforms.Normalize(mean=(0.4914, 0.4822, 0.4465),
                         std=(0.2023, 0.1994, 0.2010))
 ])
```

Pre-calculated:
mean = (0.4914, 0.4822, 0.4465)
std. dev. = (0.2023, 0.1994, 0.2010).

Prepare the dataset with datasets.ImageFolder() and use DataLoader() to load the data.

```
print(train_path)

train_dataset = datasets.ImageFolder(root=train_path,
                                     transform=data_transform)

train_dataset_loader = torch.utils.data.DataLoader(train_dataset,
                                                   batch_size=batch_size, shuffle=True,
                                                   num_workers=4)

print(train_dataset_loader.dataset.classes)
```

Output classes for training data and test data

```
./train/
['cloudy', 'desert', 'green_area', 'water']
./test/
['cloudy', 'desert', 'green_area', 'water']
```

1.2 Normalization & DataLoader

Normalize data

```
batch_size = 64
target_image_dim = 224
data_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize(target_image_dim),
    transforms.Normalize(mean=(0.4914, 0.4822, 0.4465),
                         std=(0.2023, 0.1994, 0.2010))
 ])
```

Pre-calculated:
mean = (0.4914, 0.4822, 0.4465)
std. dev. = (0.2023, 0.1994, 0.2010).

Prepare the dataset with datasets.ImageFolder() and use DataLoader() to load the data.

```
print(train_path)

train_dataset = datasets.ImageFolder(root=train_path,
                                     transform=data_transform)

train_dataset_loader = torch.utils.data.DataLoader(train_dataset,
                                                   batch_size=batch_size, shuffle=True,
                                                   num_workers=4)

print(train_dataset_loader.dataset.classes)
```

We repeat the same for testing data

Output classes for training data and test data

```
./train/
['cloudy', 'desert', 'green_area', 'water']
./test/
['cloudy', 'desert', 'green_area', 'water']
```

1.2 Normalization & DataLoader

Normalize data

```
batch_size = 64
target_image_dim = 224
data_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize(target_image_dim),
    transforms.Normalize(mean=(0.4914, 0.4822, 0.4465),
                         std=(0.2023, 0.1994, 0.2010))
 ])
```

Pre-calculated:
mean = (0.4914, 0.4822, 0.4465)
std. dev. = (0.2023, 0.1994, 0.2010).

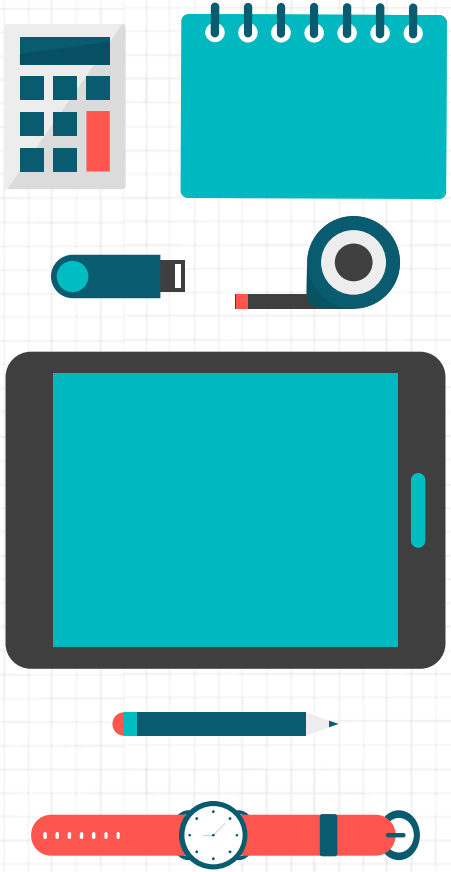
Prepare the dataset with datasets.ImageFolder() and use DataLoader() to load the data.

```
print(train_path)

train_dataset = datasets.ImageFolder(root=train_path,
                                     transform=data_transform)

train_dataset_loader = torch.utils.data.DataLoader(train_dataset,
                                                   batch_size=batch_size, shuffle=True,
                                                   num_workers=4)

print(train_dataset_loader.dataset.classes)
```

We repeat the same for testing data

Output classes for training data and test data

```
./train/
['cloudy', 'desert', 'green_area', 'water']
./test/
['cloudy', 'desert', 'green_area', 'water']
```



02

Model

2.1
Custom CNN (Small Model)

2.1 Custom CNN

Model Architecture

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)

        # Size of 224*224 become 212*212
        self.fc1 = nn.Linear(212*212, 120)
        self.fc2 = nn.Linear(120, 84)

        # Only 4 class at the end
        self.fc3 = nn.Linear(84, 4)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters())
```

2.1 Custom CNN

Model Architecture

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)

        # Size of 224*224 become 212*212
        self.fc1 = nn.Linear(212*212, 120)
        self.fc2 = nn.Linear(120, 84)

        # Only 4 class at the end
        self.fc3 = nn.Linear(84, 4)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

5 x 5 conv

features extraction module

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters())
```

2.1 Custom CNN

Model Architecture

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)

        # Size of 224*224 become 212*212
        self.fc1 = nn.Linear(212*212, 120)
        self.fc2 = nn.Linear(120, 84)

        # Only 4 class at the end
        self.fc3 = nn.Linear(84, 4)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

5 x 5 conv

3 FC layer

features extraction module

classifier module

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters())
```

2.1 Custom CNN

Model Architecture

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)

        # Size of 224*224 become 212*212
        self.fc1 = nn.Linear(212*212, 120)
        self.fc2 = nn.Linear(120, 84)

        # Only 4 class at the end
        self.fc3 = nn.Linear(84, 4)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

5 x 5 conv

3 FC layer

features extraction module

classifier module

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters())
```

Loss and Optimizer (Same across all model)

2.1 Custom CNN

Model Training phase

```
log = ""
for epoch in range(50):  # 50 Epoches

    running_loss = 0.0
    for i, data in enumerate(train_dataset_loader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # Cumulative loss
        running_loss += loss.item()
        if i % 16 == 0:    # print every 16 mini-batches

            # Logging
            log += (f'Epoch {epoch + 1} | batch {i:5d} | cumulative loss within epoch: {running_loss / 20:.3f}\n')

            # Only print last 5 epoch
            if epoch >=45:
                print(f'Epoch {epoch + 1} | batch {i:5d} | cumulative loss within epoch: {running_loss / 20:.3f}')


    # Logging
    log += (f'Saving model as "./model_{epoch+1:05d}_loss_{running_loss / 20:.3f}.pt"\n\n')

    # Only print last 5 epoch
    if epoch >=45:
        print(f'Saving model as "./model_{epoch+1:05d}_loss_{running_loss / 20:.3f}.pt"\n')

    # Save Model
    torch.save(net.state_dict(), f'./model_{epoch+1:05d}_loss_{running_loss / 20:.3f}.pt')
```

2.1 Custom CNN

Model Training phase

Feed Forward and Back Propagation

```
log = ""
for epoch in range(50):  # 50 Epoches

    running_loss = 0.0
    for i, data in enumerate(train_dataset_loader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # Cumulative loss
        running_loss += loss.item()
        if i % 16 == 0:    # print every 16 mini-batches

            # Logging
            log += (f'Epoch {epoch + 1} | batch {i:5d} | cumulative loss within epoch: {running_loss / 20:.3f}\n')

            # Only print last 5 epoch
            if epoch >=45:
                print(f'Epoch {epoch + 1} | batch {i:5d} | cumulative loss within epoch: {running_loss / 20:.3f}')


    # Logging
    log += (f'Saving model as "./model_{epoch+1:05d}_loss_{running_loss / 20:.3f}.pt"\n\n')

    # Only print last 5 epoch
    if epoch >=45:
        print(f'Saving model as "./model_{epoch+1:05d}_loss_{running_loss / 20:.3f}.pt"\n')

    # Save Model
    torch.save(net.state_dict(), f'./model_{epoch+1:05d}_loss_{running_loss / 20:.3f}.pt')
```

2.1 Custom CNN

Model Training phase

Feed Forward and Back Propagation

logging

```
log = ""
for epoch in range(50):  # 50 Epoches

    running_loss = 0.0
    for i, data in enumerate(train_dataset_loader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # Cumulative loss
        running_loss += loss.item()
        if i % 16 == 0:    # print every 16 mini-batches

            # Logging
            log += (f'Epoch {epoch + 1} | batch {i:5d} | cumulative loss within epoch: {running_loss / 20:.3f}\n')

            # Only print last 5 epoch
            if epoch >=45:
                print(f'Epoch {epoch + 1} | batch {i:5d} | cumulative loss within epoch: {running_loss / 20:.3f}')


    # Logging
    log += (f'Saving model as "./model_{epoch+1:05d}_loss_{running_loss / 20:.3f}.pt"\n\n')

    # Only print last 5 epoch
    if epoch >=45:
        print(f'Saving model as "./model_{epoch+1:05d}_loss_{running_loss / 20:.3f}.pt"\n')

    # Save Model
    torch.save(net.state_dict(), f'./model_{epoch+1:05d}_loss_{running_loss / 20:.3f}.pt')
```

2.1 Custom CNN

Model Training phase

Feed Forward and Back Propagation

logging

Save Model

```
log = ""
for epoch in range(50):  # 50 Epoches

    running_loss = 0.0
    for i, data in enumerate(train_dataset_loader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # Cumulative loss
        running_loss += loss.item()
        if i % 16 == 0:    # print every 16 mini-batches

            # Logging
            log += (f'Epoch {epoch + 1} | batch {i:5d} | cumulative loss within epoch: {running_loss / 20:.3f}\n')

            # Only print last 5 epoch
            if epoch >=45:
                print(f'Epoch {epoch + 1} | batch {i:5d} | cumulative loss within epoch: {running_loss / 20:.3f}')


    # Logging
    log += (f'Saving model as "./model_{epoch+1:05d}_loss_{running_loss / 20:.3f}.pt"\n\n')

    # Only print last 5 epoch
    if epoch >=45:
        print(f'Saving model as "./model_{epoch+1:05d}_loss_{running_loss / 20:.3f}.pt"\n')

    # Save Model
    torch.save(net.state_dict(), f'./model_{epoch+1:05d}_loss_{running_loss / 20:.3f}.pt')
```

2.1 Custom CNN

Model Training phase

```
Epoch 46 | batch     0 | cumulative loss within epoch: 0.001
Epoch 46 | batch    16 | cumulative loss within epoch: 0.035
Epoch 46 | batch    32 | cumulative loss within epoch: 0.072
Epoch 46 | batch    48 | cumulative loss within epoch: 0.125
Epoch 46 | batch    64 | cumulative loss within epoch: 0.175
Saving model as "./model_00046_loss_0.190.pt"

Epoch 47 | batch     0 | cumulative loss within epoch: 0.011
Epoch 47 | batch    16 | cumulative loss within epoch: 0.095
Epoch 47 | batch    32 | cumulative loss within epoch: 0.157
Epoch 47 | batch    48 | cumulative loss within epoch: 0.215
Epoch 47 | batch    64 | cumulative loss within epoch: 0.254
Saving model as "./model_00047_loss_0.263.pt"

Epoch 48 | batch     0 | cumulative loss within epoch: 0.001
Epoch 48 | batch    16 | cumulative loss within epoch: 0.078
Epoch 48 | batch    32 | cumulative loss within epoch: 0.164
Epoch 48 | batch    48 | cumulative loss within epoch: 0.212
Epoch 48 | batch    64 | cumulative loss within epoch: 0.259
Saving model as "./model_00048_loss_0.284.pt"

Epoch 49 | batch     0 | cumulative loss within epoch: 0.001
Epoch 49 | batch    16 | cumulative loss within epoch: 0.058
Epoch 49 | batch    32 | cumulative loss within epoch: 0.104
Epoch 49 | batch    48 | cumulative loss within epoch: 0.134
Epoch 49 | batch    64 | cumulative loss within epoch: 0.183
Saving model as "./model_00049_loss_0.196.pt"

Epoch 50 | batch     0 | cumulative loss within epoch: 0.006
Epoch 50 | batch    16 | cumulative loss within epoch: 0.034
Epoch 50 | batch    32 | cumulative loss within epoch: 0.073
Epoch 50 | batch    48 | cumulative loss within epoch: 0.090
Epoch 50 | batch    64 | cumulative loss within epoch: 0.132
Saving model as "./model_00050_loss_0.140.pt"
```

2.1 Custom CNN

Model Training phase

```
Epoch 46 | batch     0 | cumulative loss within epoch: 0.001
Epoch 46 | batch    16 | cumulative loss within epoch: 0.035
Epoch 46 | batch    32 | cumulative loss within epoch: 0.072
Epoch 46 | batch    48 | cumulative loss within epoch: 0.125
Epoch 46 | batch    64 | cumulative loss within epoch: 0.175
Saving model as "./model_00046_loss_0.190.pt"

Epoch 47 | batch     0 | cumulative loss within epoch: 0.011
Epoch 47 | batch    16 | cumulative loss within epoch: 0.095
Epoch 47 | batch    32 | cumulative loss within epoch: 0.157
Epoch 47 | batch    48 | cumulative loss within epoch: 0.215
Epoch 47 | batch    64 | cumulative loss within epoch: 0.254
Saving model as "./model_00047_loss_0.263.pt"

Epoch 48 | batch     0 | cumulative loss within epoch: 0.001
Epoch 48 | batch    16 | cumulative loss within epoch: 0.078
Epoch 48 | batch    32 | cumulative loss within epoch: 0.164
Epoch 48 | batch    48 | cumulative loss within epoch: 0.212
Epoch 48 | batch    64 | cumulative loss within epoch: 0.259
Saving model as "./model_00048_loss_0.284.pt"

Epoch 49 | batch     0 | cumulative loss within epoch: 0.001
Epoch 49 | batch    16 | cumulative loss within epoch: 0.058
Epoch 49 | batch    32 | cumulative loss within epoch: 0.104
Epoch 49 | batch    48 | cumulative loss within epoch: 0.134
Epoch 49 | batch    64 | cumulative loss within epoch: 0.183
Saving model as "./model_00049_loss_0.196.pt"

Epoch 50 | batch     0 | cumulative loss within epoch: 0.006
Epoch 50 | batch    16 | cumulative loss within epoch: 0.034
Epoch 50 | batch    32 | cumulative loss within epoch: 0.073
Epoch 50 | batch    48 | cumulative loss within epoch: 0.090
Epoch 50 | batch    64 | cumulative loss within epoch: 0.132
Saving model as "./model_00050_loss_0.140.pt"
```

Saved Filename

2.1 Custom CNN

Model Training phase

```
Epoch 46 | batch     0 | cumulative loss within epoch: 0.001
Epoch 46 | batch    16 | cumulative loss within epoch: 0.035
Epoch 46 | batch    32 | cumulative loss within epoch: 0.072
Epoch 46 | batch    48 | cumulative loss within epoch: 0.125
Epoch 46 | batch    64 | cumulative loss within epoch: 0.175
Saving model as "./model_00046_loss_0.190.pt"

Epoch 47 | batch     0 | cumulative loss within epoch: 0.011
Epoch 47 | batch    16 | cumulative loss within epoch: 0.095
Epoch 47 | batch    32 | cumulative loss within epoch: 0.157
Epoch 47 | batch    48 | cumulative loss within epoch: 0.215
Epoch 47 | batch    64 | cumulative loss within epoch: 0.254
Saving model as "./model_00047_loss_0.263.pt"

Epoch 48 | batch     0 | cumulative loss within epoch: 0.001
Epoch 48 | batch    16 | cumulative loss within epoch: 0.078
Epoch 48 | batch    32 | cumulative loss within epoch: 0.164
Epoch 48 | batch    48 | cumulative loss within epoch: 0.212
Epoch 48 | batch    64 | cumulative loss within epoch: 0.259
Saving model as "./model_00048_loss_0.284.pt"

Epoch 49 | batch     0 | cumulative loss within epoch: 0.001
Epoch 49 | batch    16 | cumulative loss within epoch: 0.058
Epoch 49 | batch    32 | cumulative loss within epoch: 0.104
Epoch 49 | batch    48 | cumulative loss within epoch: 0.134
Epoch 49 | batch    64 | cumulative loss within epoch: 0.183
Saving model as "./model_00049_loss_0.196.pt"

Epoch 50 | batch     0 | cumulative loss within epoch: 0.006
Epoch 50 | batch    16 | cumulative loss within epoch: 0.034
Epoch 50 | batch    32 | cumulative loss within epoch: 0.073
Epoch 50 | batch    48 | cumulative loss within epoch: 0.090
Epoch 50 | batch    64 | cumulative loss within epoch: 0.132
Saving model as "./model_00050_loss_0.140.pt"
```

Saved Filename

Detailed Performance Analysis in Part 3

2.2
Transfer Learning using ResNet18 (Medium Model)

2.2 Transfer Learning using ResNet18

Model Architecture

```
# Download Weights (this will initiate a download if not cached)
resnet_model = models.resnet18(pretrained=True)

# Set all layer to be frozen
for param in resnet_model.parameters():
    param.requires_grad = False

# add another fc layer at the end
resnet_model = nn.Sequential(resnet_model, nn.Linear(in_features=1000, out_features=4))

# requires_grad to be True only for last layer
# Check, last 2 should be 1
print([int(x.requires_grad) for x in resnet_model.parameters()])
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
```

2.2 Transfer Learning using ResNet18

Model Architecture

```
# Download Weights (this will initiate a download if not cached)
resnet_model = models.resnet18(pretrained=True)
```

Initial Weights

```
# Set all layer to be frozen
for param in resnet_model.parameters():
    param.requires_grad = False

# add another fc layer at the end
resnet_model = nn.Sequential(resnet_model, nn.Linear(in_features=1000, out_features=4))

# requires_grad to be True only for last layer
# Check, last 2 should be 1
print([int(x.requires_grad) for x in resnet_model.parameters()])
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
```

2.2 Transfer Learning using ResNet18

Model Architecture

```
# Download Weights (this will initiate a download if not cached)
resnet_model = models.resnet18(pretrained=True)
```

Initial Weights

```
# Set all layer to be frozen
for param in resnet_model.parameters():
    param.requires_grad = False
```

Frozen Layers

```
# add another fc layer at the end
resnet_model = nn.Sequential(resnet_model, nn.Linear(in_features=1000, out_features=4))

# requires_grad to be True only for last layer
# Check, last 2 should be 1
print([int(x.requires_grad) for x in resnet_model.parameters()])
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
```

2.2 Transfer Learning using ResNet18

Model Architecture

```
# Download Weights (this will initiate a download if not cached)
resnet_model = models.resnet18(pretrained=True)
```

Initial Weights

```
# Set all layer to be frozen
for param in resnet_model.parameters():
    param.requires_grad = False
```

Frozen Layers

```
# add another fc layer at the end
resnet_model = nn.Sequential(resnet_model, nn.Linear(in_features=1000, out_features=4))
```

Add FC layer

```
# requires_grad to be True only for last layer
# Check, last 2 should be 1
print([int(x.requires_grad) for x in resnet_model.parameters()])
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
```

2.2 Transfer Learning using ResNet18

Model Architecture

```
# Download Weights (this will initiate a download if not cached)
resnet_model = models.resnet18(pretrained=True)

# Set all layer to be frozen
for param in resnet_model.parameters():
    param.requires_grad = False

# add another fc layer at the end
resnet_model = nn.Sequential(resnet_model, nn.Linear(in_features=1000, out_features=4))

# requires_grad to be True only for last layer
# Check, last 2 should be 1
print([int(x.requires_grad) for x in resnet_model.parameters()])
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
```

2.2 Transfer Learning using ResNet18

Model Architecture

```
# Download Weights (this will initiate a download if not cached)
resnet_model = models.resnet18(pretrained=True)

# Set all layer to be frozen
for param in resnet_model.parameters():
    param.requires_grad = False

# add another fc layer at the end
resnet_model = nn.Sequential(resnet_model, nn.Linear(in_features=1000, out_features=4))
```

```
# requires_grad to be True only for last layer
# Check, last 2 should be 1
print([int(x.requires_grad) for x in resnet_model.parameters()])
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(resnet_model.parameters())
```

Same loss function and optimizer

2.2 Transfer Learning using ResNet18

Training Result

```
Epoch 1 | batch     0 | cumulative loss within epoch: 0.088
Epoch 1 | batch    16 | cumulative loss within epoch: 0.428
Epoch 1 | batch    32 | cumulative loss within epoch: 0.520
Epoch 1 | batch    48 | cumulative loss within epoch: 0.578
Epoch 1 | batch    64 | cumulative loss within epoch: 0.623
Saving model as "./resnet_model_00001_loss_0.634.pt"

Epoch 2 | batch     0 | cumulative loss within epoch: 0.001
Epoch 2 | batch    16 | cumulative loss within epoch: 0.039
Epoch 2 | batch    32 | cumulative loss within epoch: 0.088
Epoch 2 | batch    48 | cumulative loss within epoch: 0.125
Epoch 2 | batch    64 | cumulative loss within epoch: 0.161
Saving model as "./resnet_model_00002_loss_0.173.pt"

Epoch 3 | batch     0 | cumulative loss within epoch: 0.001
Epoch 3 | batch    16 | cumulative loss within epoch: 0.023
Epoch 3 | batch    32 | cumulative loss within epoch: 0.043
Epoch 3 | batch    48 | cumulative loss within epoch: 0.071
Epoch 3 | batch    64 | cumulative loss within epoch: 0.097
Saving model as "./resnet_model_00003_loss_0.109.pt"

Epoch 4 | batch     0 | cumulative loss within epoch: 0.001
Epoch 4 | batch    16 | cumulative loss within epoch: 0.021
Epoch 4 | batch    32 | cumulative loss within epoch: 0.042
Epoch 4 | batch    48 | cumulative loss within epoch: 0.054
Epoch 4 | batch    64 | cumulative loss within epoch: 0.087
Saving model as "./resnet_model_00004_loss_0.096.pt"

Epoch 5 | batch     0 | cumulative loss within epoch: 0.001
Epoch 5 | batch    16 | cumulative loss within epoch: 0.020
Epoch 5 | batch    32 | cumulative loss within epoch: 0.041
Epoch 5 | batch    48 | cumulative loss within epoch: 0.062
Epoch 5 | batch    64 | cumulative loss within epoch: 0.080
Saving model as "./resnet_model_00005_loss_0.087.pt"
```

2.3
Transfer Learning using VGG19 (Large Model)

2.3 Transfer Learning using VGG19

Model Architecture

```
# Download Weights (this will initiate a download if not cached)
vgg_model = models.vgg19(pretrained=True)

# Set all layer to be frozen
for param in vgg_model.parameters():
    param.requires_grad = False

# add another fc layer at the end
vgg_model = nn.Sequential(vgg_model, nn.Linear(in_features=1000, out_features=4))

# requires_grad to be True only for last layer
# Check, last 2 should be 1
print([int(x.requires_grad) for x in vgg_model.parameters()])
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
```

2.3 Transfer Learning using VGG19

Model Architecture

```
# Download Weights (this will initiate a download if not cached)
vgg_model = models.vgg19(pretrained=True)
```

Initial Weights

```
# Set all layer to be frozen
for param in vgg_model.parameters():
    param.requires_grad = False
```

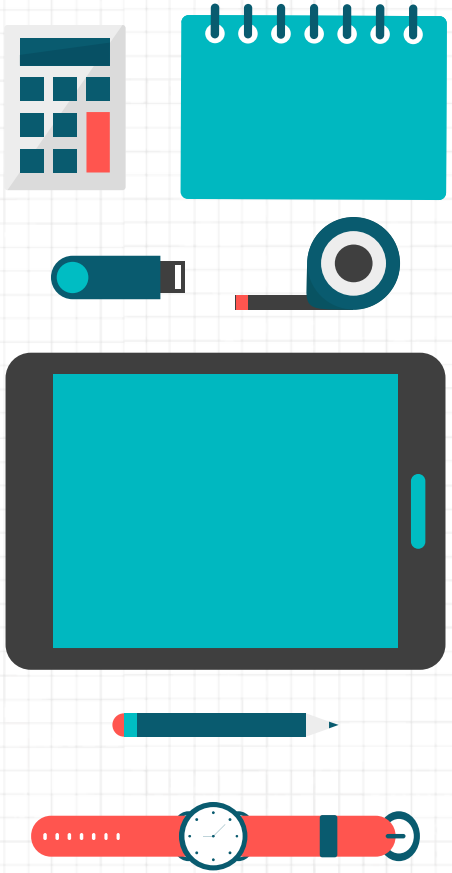
Frozen layer

```
# add another fc layer at the end
vgg_model = nn.Sequential(vgg_model, nn.Linear(in_features=1000, out_features=4))
```

Add FC layer

```
# requires_grad to be True only for last layer
# Check, last 2 should be 1
print([int(x.requires_grad) for x in vgg_model.parameters()])
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
```



03

Performance Analysis & Model Selection

3. Performance Analysis & Model Selection

	type	epoch	loss	test_accuracy_score	test_recall_score	test_precision_score	test_f1_score	test_prediction_time_taken
0	CNN	1	2.040.	0.5138	0.5423	0.5813	0.4813	5.561126
1	CNN	2	1.843.	0.8571	0.8642	0.8657	0.8643	5.582082
2	CNN	3	1.245.	0.8589	0.8659	0.8719	0.8644	5.622688
3	CNN	4	1.222.	0.8625	0.8687	0.8744	0.8683	5.573508
4	CNN	5	1.100.	0.8713	0.8776	0.8817	0.878	5.559339
5	CNN	6	1.068.	0.89	0.8951	0.9054	0.8947	5.558045
6	CNN	7	0.965.	0.8882	0.8926	0.9033	0.8928	5.528205
7	CNN	8	1.072.	0.8802	0.8859	0.9013	0.8847	5.64043
8	CNN	9	0.975.	0.874	0.8801	0.8872	0.8792	5.618529

(... epoch 10 to 42)

43	CNN	44	0.341.	0.9122	0.9156	0.9267	0.916	5.471462
44	CNN	45	0.270.	0.9636	0.9645	0.9651	0.9647	5.383117
45	CNN	46	0.190.	0.9255	0.9287	0.9405	0.9283	5.431942
46	CNN	47	0.263.	0.9556	0.9575	0.9566	0.9568	5.533185
47	CNN	48	0.284.	0.9796	0.9806	0.9796	0.9799	5.449032
48	CNN	49	0.196.	0.9627	0.9602	0.966	0.9625	5.513555
49	CNN	50	0.140.	0.9645	0.9651	0.9674	0.966	5.465121

3. Performance Analysis & Model Selection

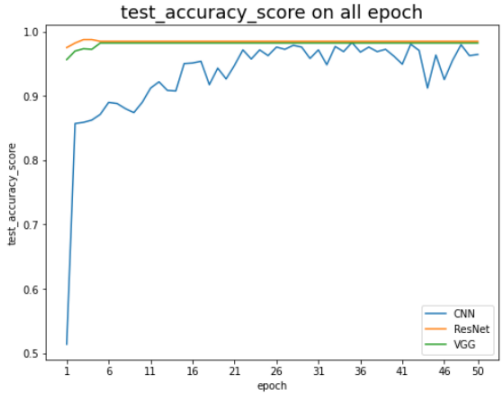
	type	epoch	loss	test_accuracy_score	test_recall_score	test_precision_score	test_f1_score	test_prediction_time_taken
0	CNN	1	2.040.	0.5138	0.5423	0.5813	0.4813	5.561126
1	CNN	2	1.843.	0.8571	0.8642	0.8657	0.8643	5.582082
2	CNN	3	1.245.	0.8589	0.8659	0.8719	0.8644	5.622688
3	CNN	4	1.222.	0.8625	0.8687	0.8744	0.8683	5.573508
4	CNN	5	1.100.	0.8713	0.8776	0.8817	0.878	5.559339
5	CNN	6	1.068.	0.89	0.8951	0.9054	0.8947	5.558045
6	CNN	7	0.965.	0.8882	0.8926	0.9033	0.8928	5.528205
7	CNN	8	1.072.	0.8802	0.8859	0.9013	0.8847	5.64043
8	CNN	9	0.975.	0.874	0.8801	0.8872	0.8792	5.618529

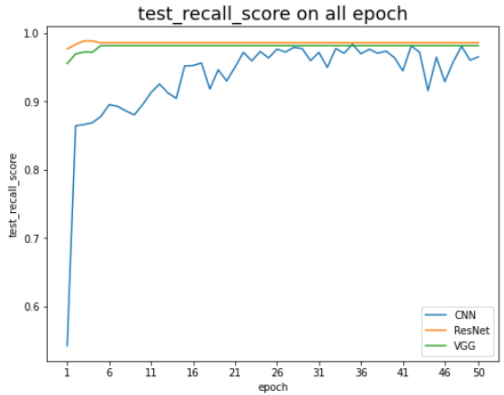
(... epoch 10 to 42)

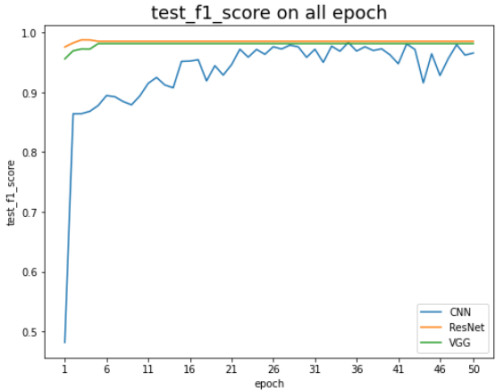
43	CNN	44	0.341.	0.9122	0.9156	0.9267	0.916	5.471462
44	CNN	45	0.270.	0.9636	0.9645	0.9651	0.9647	5.383117
45	CNN	46	0.190.	0.9255	0.9287	0.9405	0.9283	5.431942
46	CNN	47	0.263.	0.9556	0.9575	0.9566	0.9568	5.533185
47	CNN	48	0.284.	0.9796	0.9806	0.9796	0.9799	5.449032
48	CNN	49	0.196.	0.9627	0.9602	0.966	0.9625	5.513555
49	CNN	50	0.140.	0.9645	0.9651	0.9674	0.966	5.465121

96% accuracy after 50 epoch

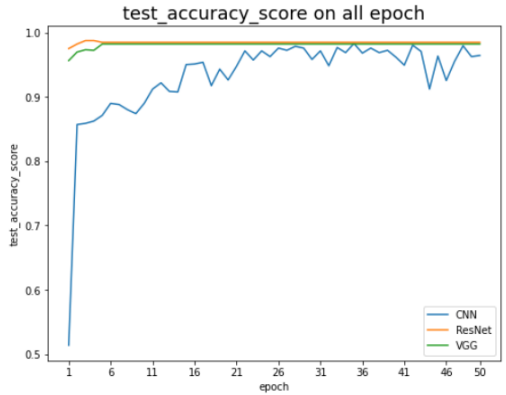
3. Performance Analysis & Model Selection

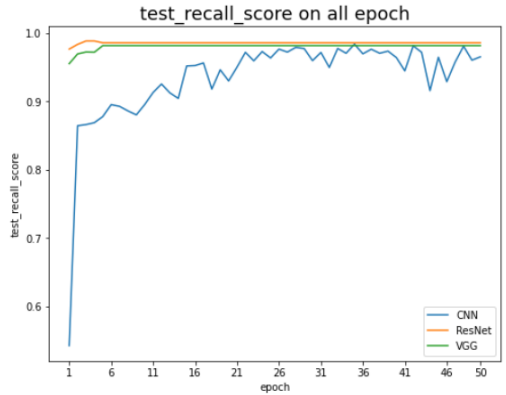


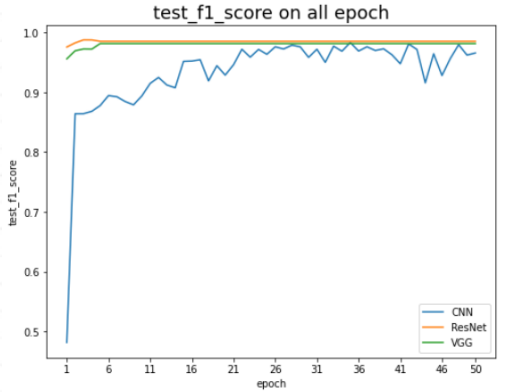




3. Performance Analysis & Model Selection

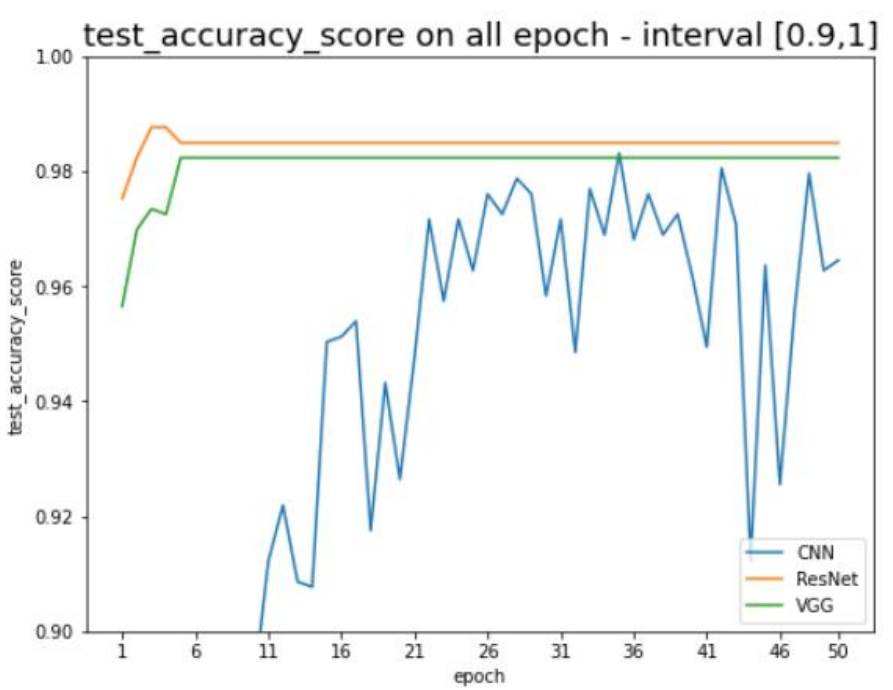




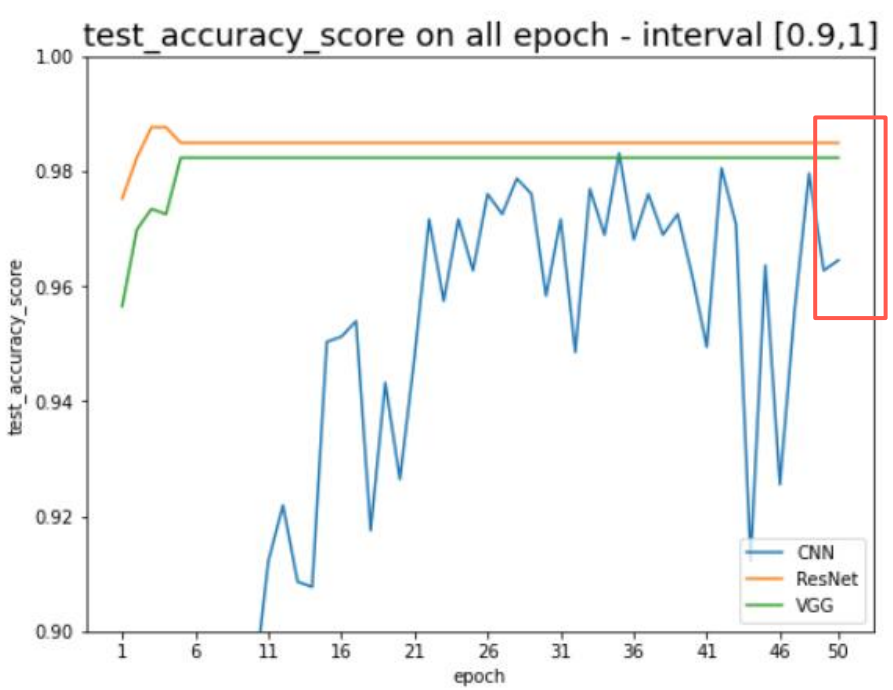


curve behave the same, look at accuracy in details

3. Performance Analysis & Model Selection

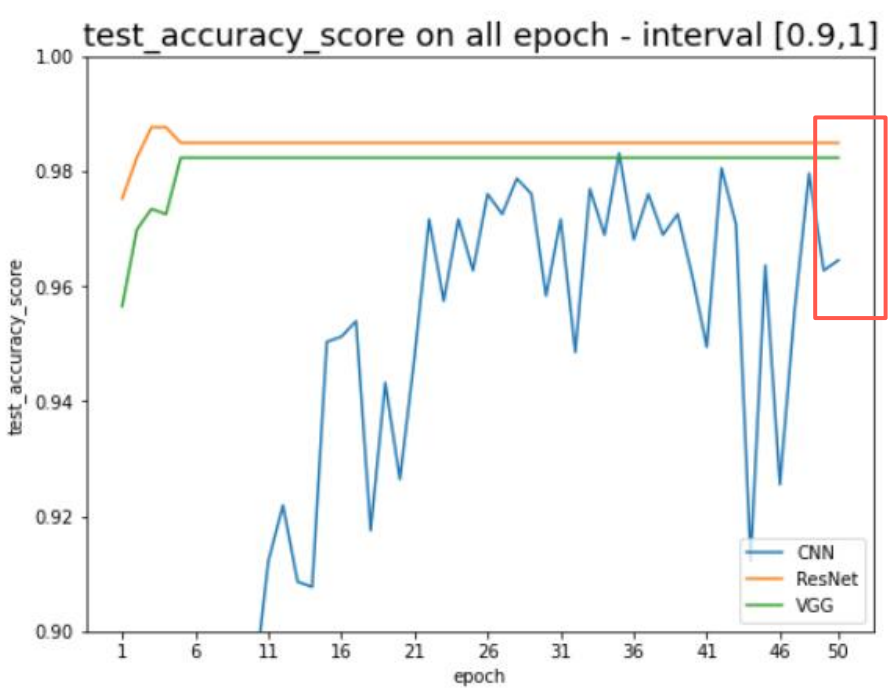


3. Performance Analysis & Model Selection



CNN - 96.45%
ResNet - 98.49%
VGG - 98.23 %

3. Performance Analysis & Model Selection

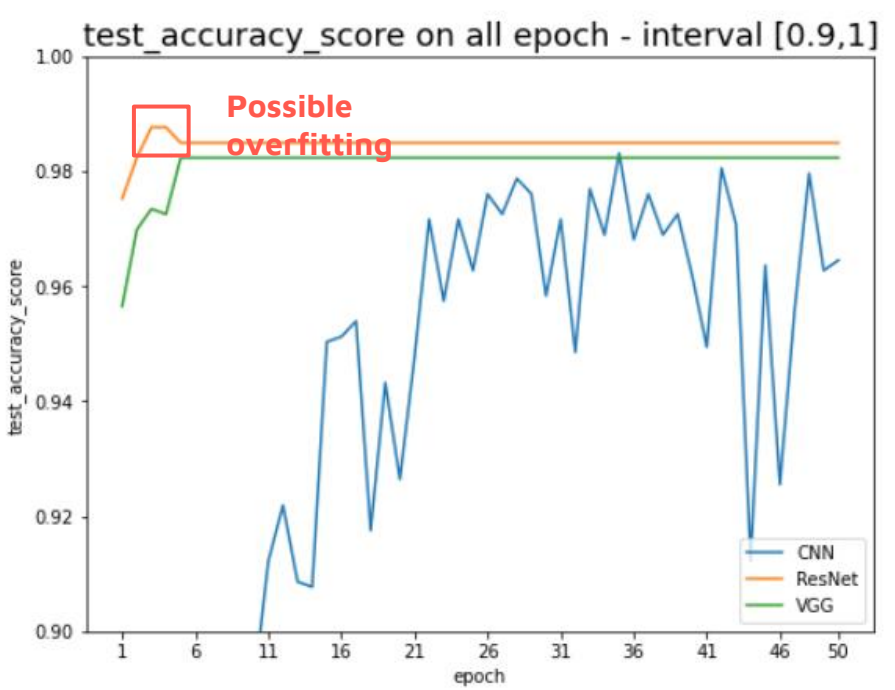


CNN - 96.45%
ResNet - 98.49%
VGG - 98.23 %

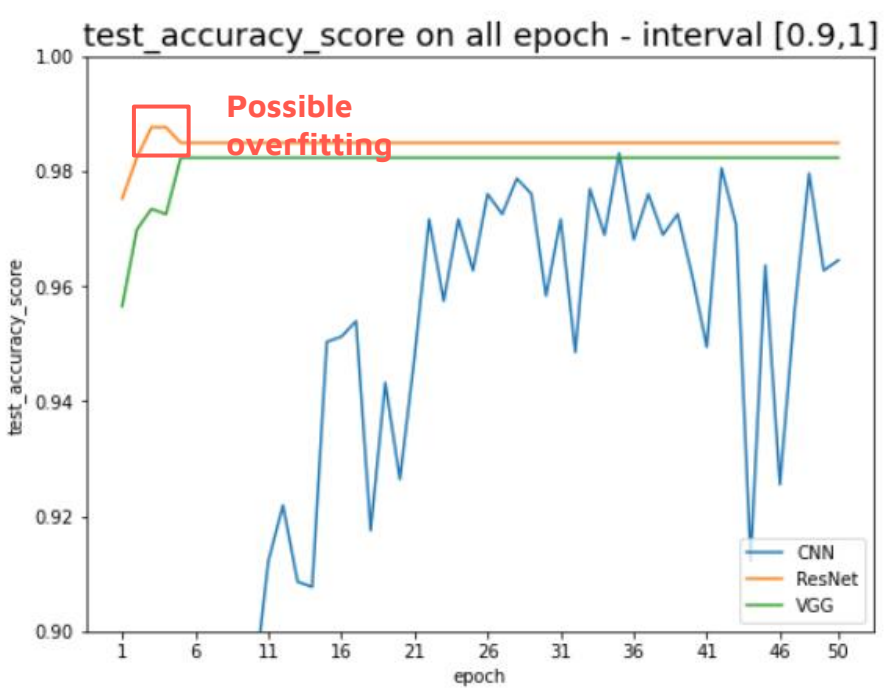
CNN Significantly lower
VGG take longer training time

Choose ResNet best model

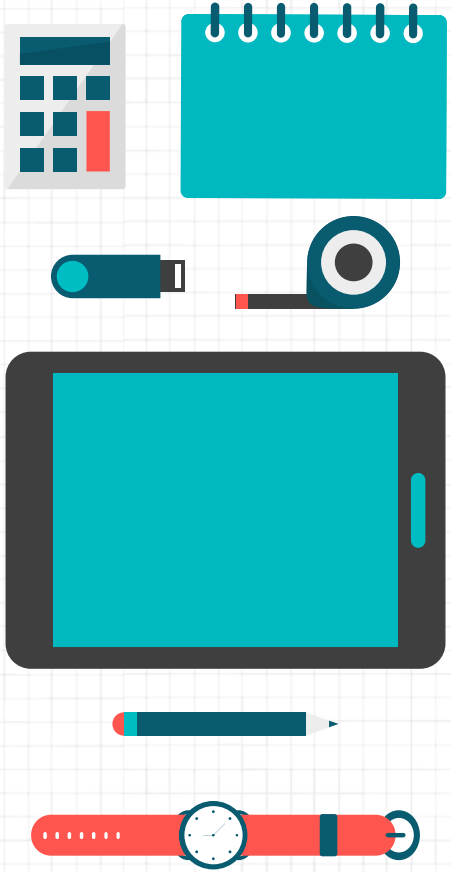
3. Performance Analysis & Model Selection



3. Performance Analysis & Model Selection



Choose ResNet Epoch 3 (98.76%) as final model



04

Conclusion & Future Work Suggestion

Conclusion

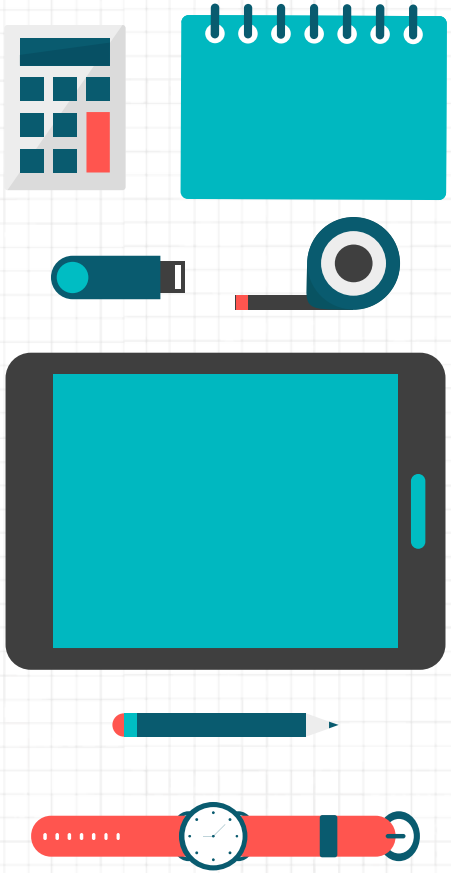
- Pretrain model is better in image classification task
- Larger model is not always better (ResNet 50 MB, VGG 500MB)

Conclusion

- Pretrain model is better in image classification task
- Larger model is not always better (ResNet 50 MB, VGG 500MB)

Future Work Suggestion

- Increase Dataset Size
- Deal with data imbalance problem (eg: SMOTE)



05

Reference & Contribution

Reference

- He, K., Zhang, X., Ren, S., & Sun, J. (2016).
  Deep residual learning for image recognition. 2016
  IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
- Karen, S., Andrew, Z.(2014).
  Very Deep Convolutional Networks for Large-Scale Image Recognition.
- https://pytorch.org/tutorials/

Contribution

Emily Choo Hue Che	Preprocessing, Custom CNN
Ang Jian Hwee	ResNet, VGG, Performance Analysis