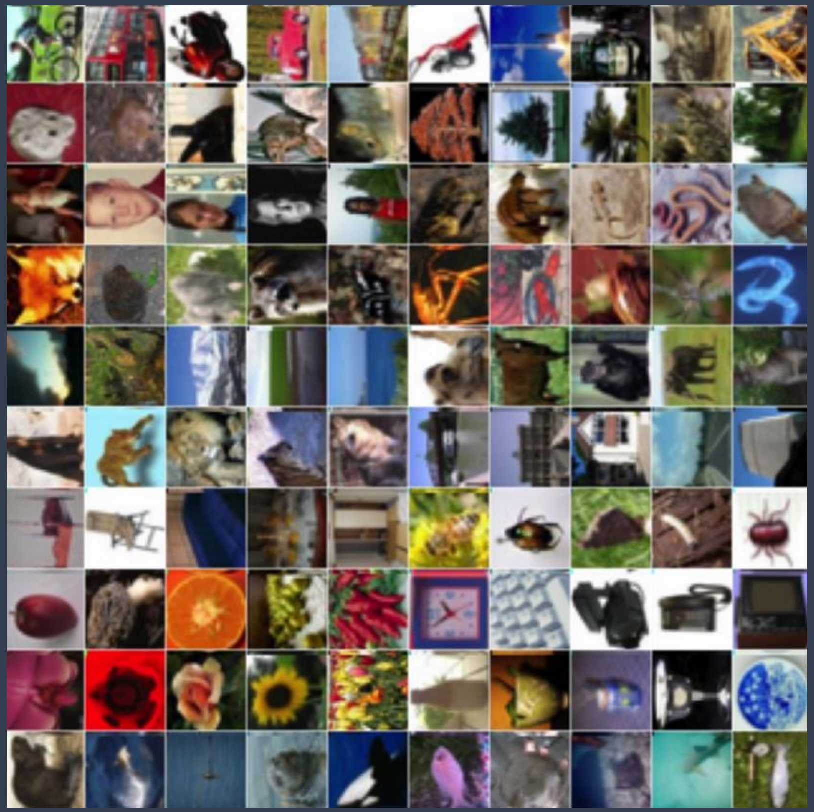
- ➢60k images in total with 100 class (balanced class distribution)
- ➢Each class: 500 training images, 100 testing images.
- ➢Images size: 32x32x3 (RGB)



# Problem description – CIFAR100

https://www.cs.toronto.edu/~kriz/cifar.html

Train a classification model with limited size, 100MB. Then test your model on testing data with report showing your method to prove the provided code (baseline).

Data: cifar100 from *Torchvision*.

Training data:

```
trainset = torchvision.datasets.CIFAR100(root='./data', train=True,
                                         download=True, transform=transform)
```

Testing data with accuracy:

```
testset = torchvision.datasets.CIFAR100(root='./data', train=False,
                                        download=True, transform=transform)
```

## Problem description

Test your model with provided test code segment:

```
correct = 0
total = 0
# since we're not training, we don't need to calculate the gradients for our outputs
with torch.no_grad():
    for data in testloader:
        images, labels = data
        images = images.to(device)
        labels = labels.to(device)
        # calculate outputs by running images through the network
        outputs = model(images)
        # the class with the highest energy is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct // total} %')
```

Then save your model to upload E3:

torch.save(model, "./saved/model.pth")

# Upload and Test

30% test ranking

## Main Component:

1. Dataset preprocessing: data augmentation
2. Model architecture: other efficient model (less than 100 MB for entire saved model)
3. Loss function: focal loss, consistency loss.
4. Optimizer: learning rate scheduler
5. Training strategy (extra data): self / semi / transfer learning, domain adaptation
6. Other

---

Improve the code and **report** your techniques

70% report

# 1. Dataset preprocessing: data augmentation

```python
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

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

Give me your mean & std in report for testing if changing them!

```python
mean = (0.5071, 0.4867, 0.4408)
std = (0.2675, 0.2565, 0.2761)
train_transform = transforms.Compose(
    [transforms.RandomHorizontalFlip(p=0.5),
     transforms.ToTensor(),
     transforms.Normalize(mean, std)]) # calculte yourself

test_transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize(mean, std)]) # calculte yourself

batch_size = 32
num_classes = 100    # check

trainset = torchvision.datasets.CIFAR100(root='./data', train=True,
                                         download=True, transform=train_transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR100(root='./data', train=False,
                                        download=True, transform=test_transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)
```

Error: CUDA out of memory!

## Supporting Code

https://pytorch.org/vision/stable/transforms.html

# 2. Model architecture: other efficient model (1/2)

## Define by your self our clone other code

```python
class Toy_CNN(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)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, num_classes)

    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
```

## off-the-shelf model

ResNet

`resnet18` ([pretrained, progress])

`resnet34` ([pretrained, progress])

`resnet50` ([pretrained, progress])

```python
model = models.resnet50(pretrained=True)
model.fc = torch.nn.Linear(2048, num_classes)
```

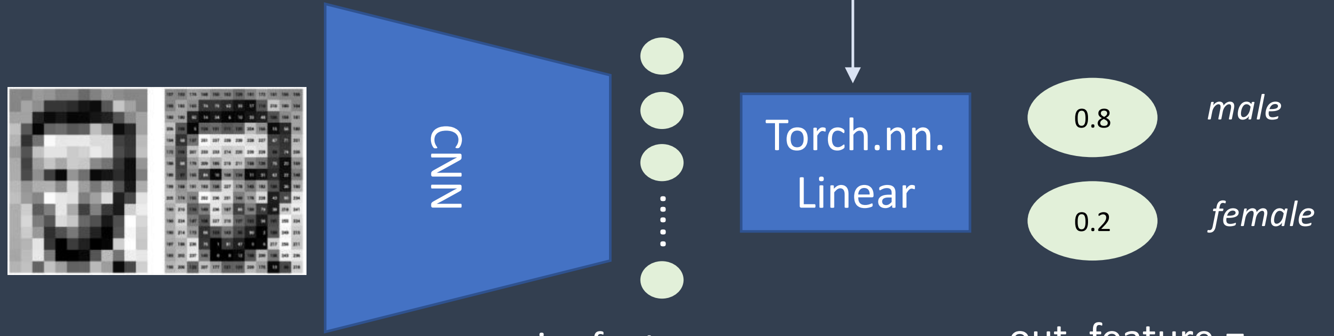
**ResNet50 is small than 100MB*

# Supporting Code

PyTorch model: https://pytorch.org/vision/stable/models.html

# 2. Model architecture: other efficient model (2/2)

```python
model = models.mobilenet_v3_small(pretrained=True)
model.classifier[3] = torch.nn.Linear(1024, num_classes)
```




in_feature =
length of input vector

out_feature =
length of output vector

```python
print(model)
```

```
    (avgpool): AdaptiveAvgPool2d(output_size=1)
    (classifier): Sequential(
      (0): Linear(in_features=576, out_features=1024, bias=True)
      (1): Hardswish()
      (2): Dropout(p=0.2, inplace=True)
      (3): Linear(in_features=1024, out_features=100, bias=True)
    )
  )
```

```python
model = models.mobilenet_v3_small(pretrained=True)
model.classifier[3] = torch.nn.Linear(1024, num_classes)
```

Change the last layer by code

## Supporting Code

torch.nn.Linear:
https://pytorch.org/docs/stable/generated/torch.nn.Linear.html

3. Loss function

4. Optimizer

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001, weight_decay=1e-4)
```

more PyTorch built-in loss function:
https://pytorch.org/docs/stable/nn.html#loss-functions

more PyTorch built-in optimizer:
https://pytorch.org/docs/stable/optim.html

## Supporting Code

# 5. Training strategy (½)

```python
total_epoch = 20
print_per_iteration = 100
save_path = './model.pth'
major_class = 10

for epoch in range(total_epoch):  # loop over the dataset multiple times
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)
        # zero the parameter gradients
        optimizer.zero_grad()
        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        if (i+1) % print_per_iteration == 0:    # print every 2000 mini-batches
            print(f'[ep {epoch + 1}][{i + 1:5d}/{len(trainloader):5d}] loss: {loss.item():.3f}')
torch.save(model, save_path)
```

Don't use

torch.save(model.state_dict(), save_path)

## Supporting Code

# 5. Training strategy (2/2)

(extra data): self / semi / transfer learning, long tail distributions technique

- FixMatch(Semi-S): https://github.com/kekmodel/FixMatch-pytorch
- BYOR(Self-S): https://github.com/lucidrains/byol-pytorch
- Pseudo long-tail techniques:

```
criterion = nn.CrossEntropyLoss(reduction='none')
optimizer = optim.Adam(model.parameters(), lr=0.0001,
                       weight_decay=1e-4)
def get_major_class(outputs):
    pass
```

Just a sample !

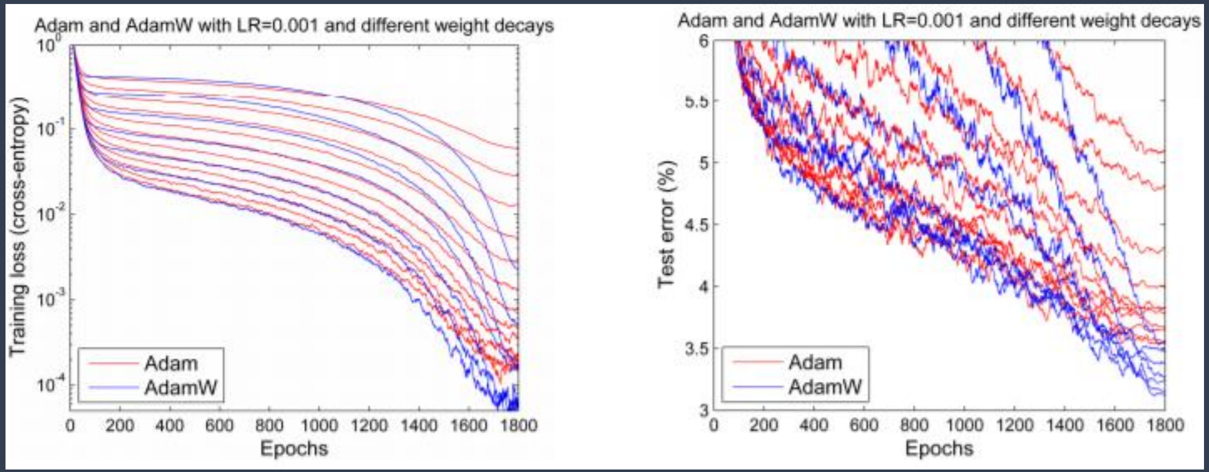
Requires more modification to work !

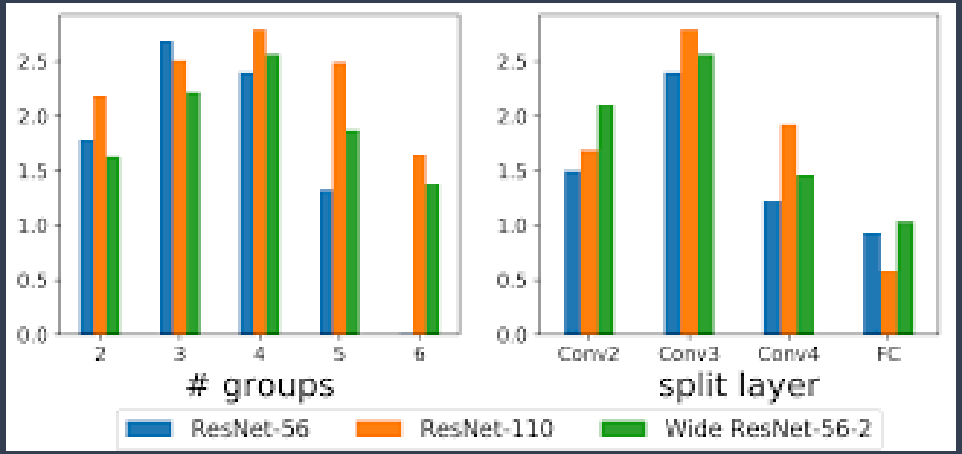
```
for epoch in range(total_epoch):  # loop over the dataset multiple times
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)
        # zero the parameter gradients
        optimizer.zero_grad()
        # forward + backward + optimize
        outputs = model(inputs)
        with torch.no_grad():
            major_class = get_major_class(outputs)  # no_grad
            mask = (torch.max(outputs, dim=1)[1] == major_class).float()
        loss = (criterion(outputs, labels) * mask).mean()
        loss.backward()
        optimizer.step()
```

## Supporting Code

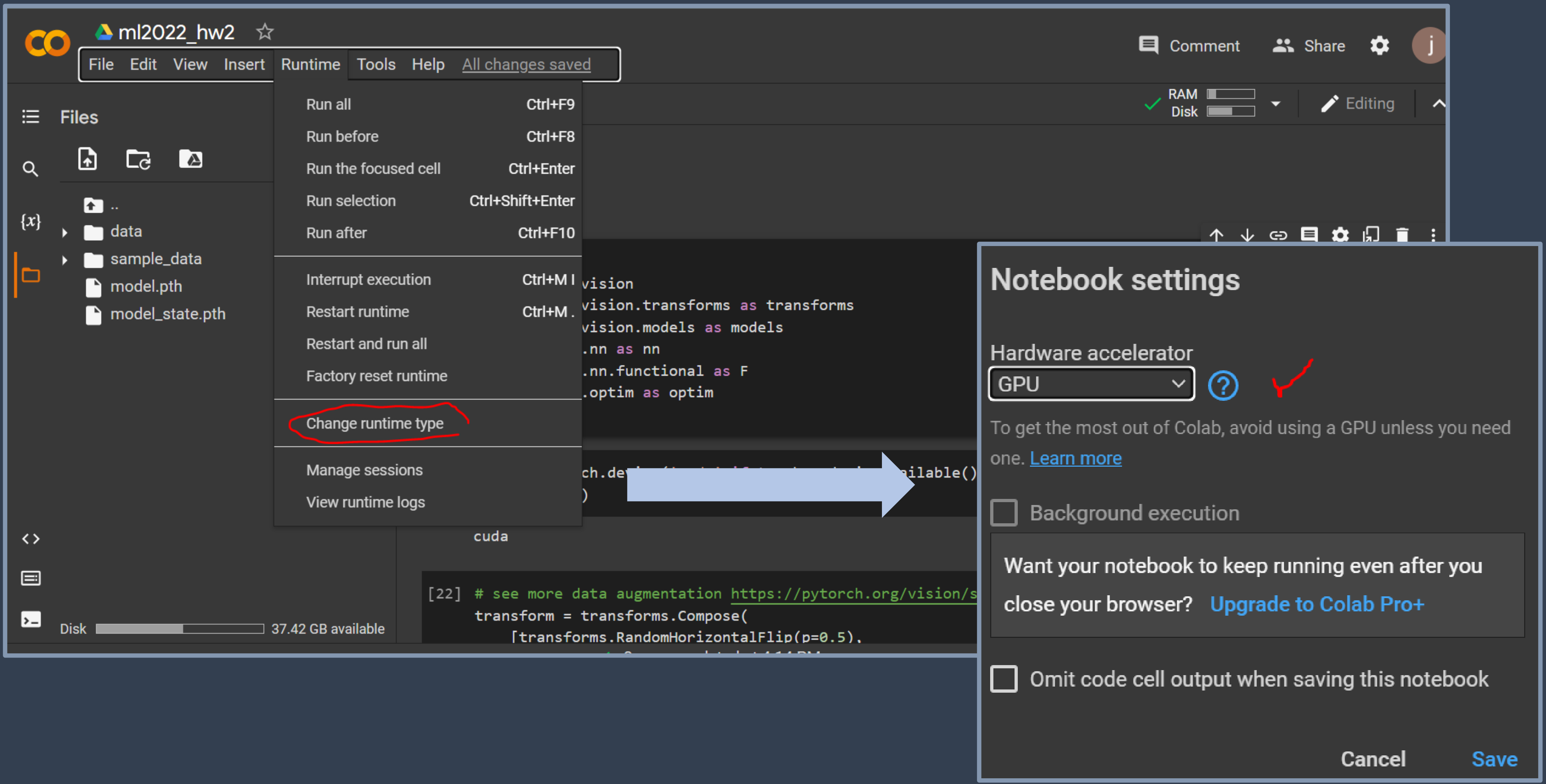
# 6. Other

調參數大師
(lr, batch_size ...)




# Supporting Code






# Colab for quick starting

https://colab.research.google.com/?utm_source=scs-index

# Upload file

1. Trained model: hw2_ID.pth
   - check the test accuracy by the code ☞
   - Less than **100MB** (otherwise will fail to upload the model.pth to E3)
2. Report: hw2_ID.pdf
3. (optional) hw2_ID.txt

```
mean = (0.5, 0.5, 0.5)
std = (0.5, 0.5, 0.5)
train_transform = transforms.Compose(
    [transforms.RandomHorizontalFlip(p=0.5),
     transforms.ToTensor(),
     transforms.Normalize(mean, std)]) # calculte yourself

test_transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize(mean, std)]) # calculte yourself

batch_size = 32
num_classes = 100    # check

trainset = torchvision.datasets.CIFAR100(root='./data', train=True,
                                        download=True, transform=train_transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR100(root='./data', train=False,
                                                                    m)

testloader = torch.utils.dat
```

hw2_309510151.txt - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
0.5 0.5 0.5
0.24 0.24 0.24
```

**Deadline 4/10 23:55**