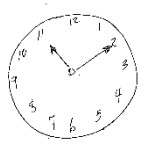
應用深度學習於畫時鐘測試

presenter：虎冠廷

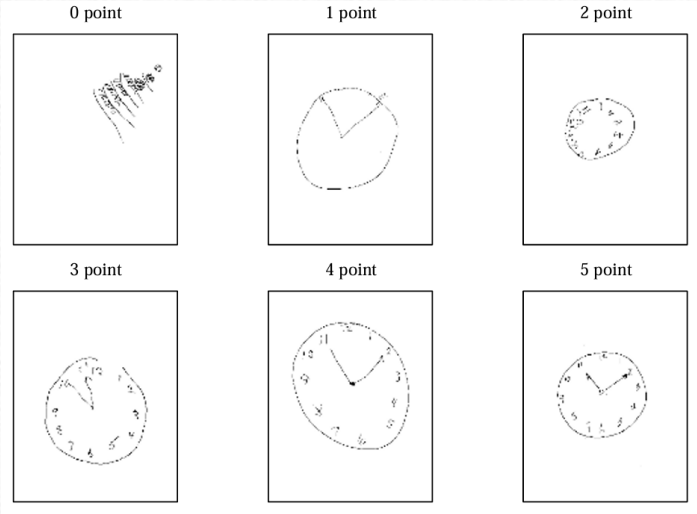


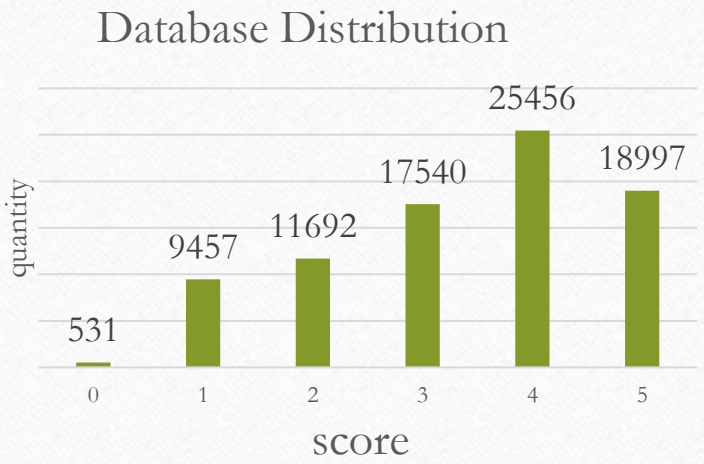
DATASET

- An association sponsored by National Institutes of Health (United States)

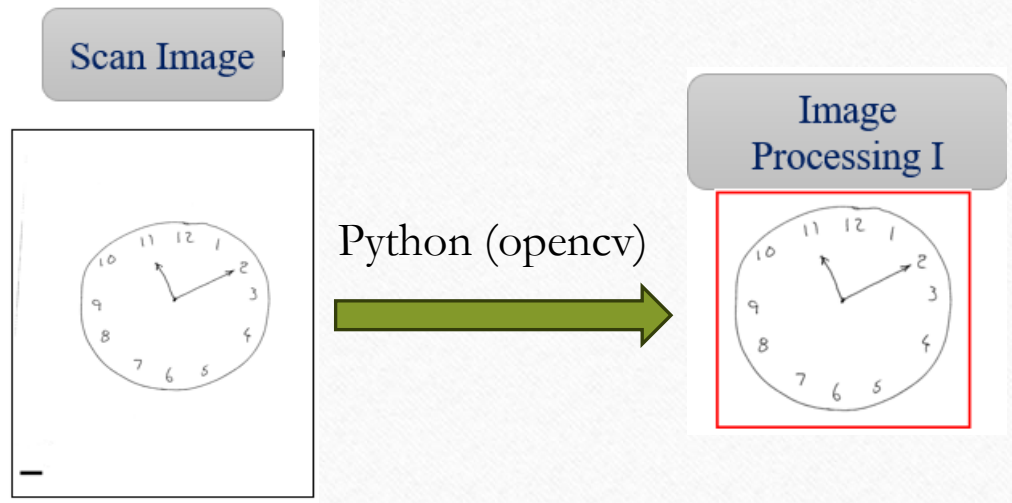
nhats National Health & Aging Trends Study

how daily life changes as we age



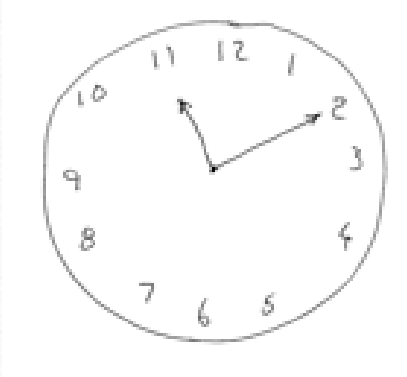


- Objective
  - 3000 samples for each class



Deal with data imbalance

- up-sample
  - data augmentation
  - Generative AI
- down-sample

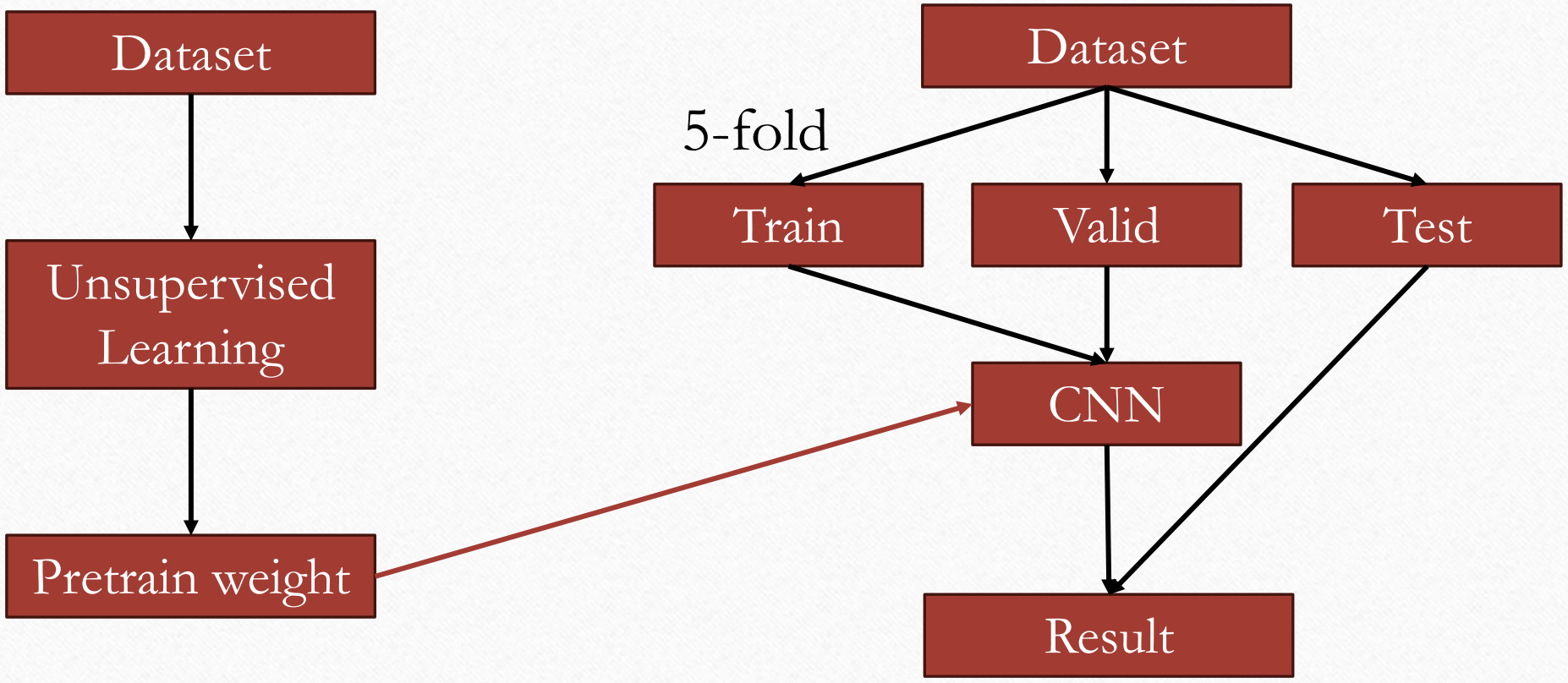


Small Dataset

VS

Big Dataset

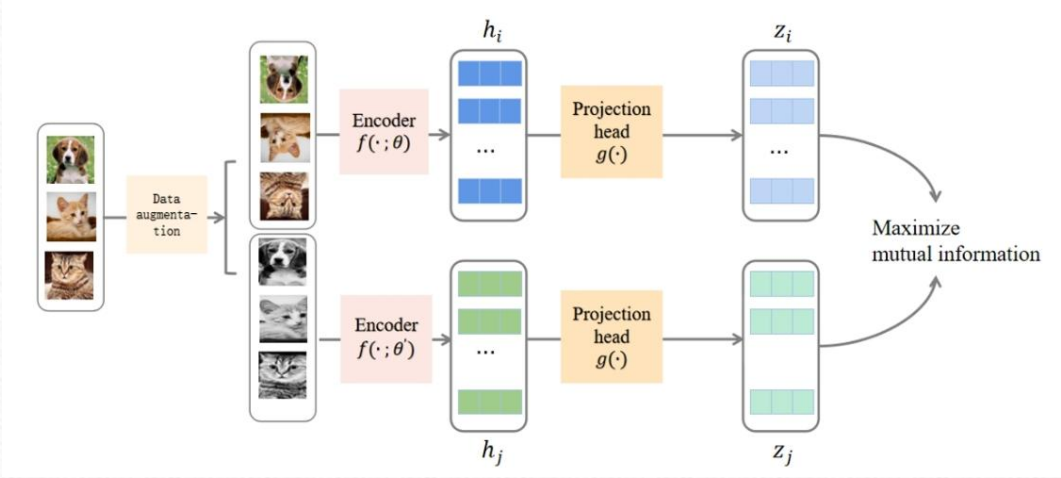
Pretrain weight

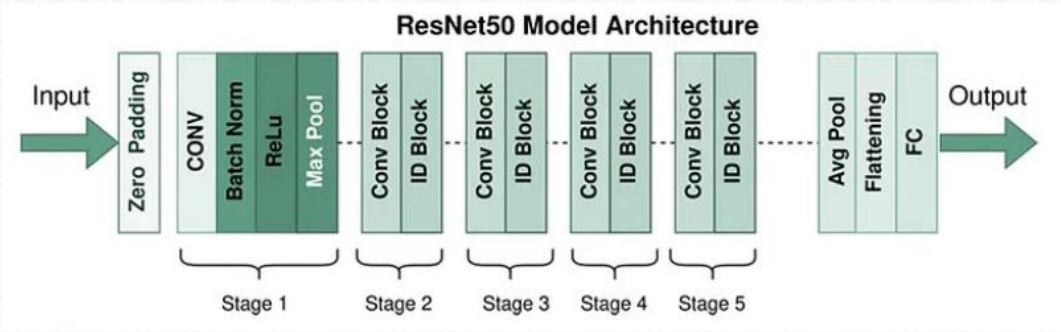

Dataset
Unsupervised Learning
Pretrain weight
Dataset
5-fold
Train
Valid
Test
CNN
Result

Model

Part I : Simclr

Part II : Resnet50





Experiment Process

1. Create data through data augmentation to balance the dataset

```python
def augment(image: tf.Tensor) -> tf.Tensor:

    random_rotation = tf.random.uniform(shape=[], minval=1, maxval=3, dtype=tf.int32)

    z = tf.image.rot90(image, k=random_rotation)
    z = tf.image.random_flip_left_right(z, seed=None)
    z = tf.image.random_flip_up_down(z, seed=None)
    z = random_crop_and_resize(z, seed=42, max_border=50)

    z = tf.image.random_hue(image, 0.1)
    z = tf.image.random_saturation(z, 0.8, 1.2)
    z = tf.image.random_contrast(z, 0.7, 1.3)
    z = tf.image.random_brightness(z, max_delta=0.35)

    return z
```

2. transform

```
from torchvision import transforms
train_transform = transforms.Compose([
                transforms.Resize((32, 32)),
                transforms.RandomHorizontalFlip(),
                transforms.RandomVerticalFlip(),
                transforms.RandomRotation(30),
                transforms.ToTensor(),
                transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        ])

valid_transform = transforms.Compose([
                transforms.Resize((32, 32)),
                transforms.ToTensor(),
                transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        ])
```

3. hyperparameter

- batch size = 128
- loss function = nn.CrossEntropyLoss()
- learning rate = 5e-4
- optimizer = optim.Adam(model.parameters(), lr=learning rate)
- scheduler = lr scheduler.ExponentialLR(optimizer, gamma=0.9)
- epochs = 50

4. Model Structure

- ResNet50

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()

        # Load the ResNet-50 model
        resnet = models.resnet50(pretrained=False)

        # Remove the fully connected layers at the end
        resnet = nn.Sequential(*list(resnet.children())[:-2])

        # Add it to your model
        self.resnet = resnet

        # Define your own classifier
        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d((1, 1)),
            nn.Flatten(),
            nn.Linear(2048, 2048),
            nn.ReLU(inplace=True),
            nn.Dropout(0.7),
            nn.Linear(2048, 1024),
            nn.ReLU(inplace=True),
            nn.Dropout(0.7),
            nn.Linear(1024, 6)
        )

    def forward(self, x):
        x = self.resnet(x)
        x = self.classifier(x)
        return x
```

- with / without pretrained weight

```
if pretrained_weights_path:
    pretrained_dict = torch.load(pretrained_weights_path)
    model_dict = self.state_dict()
    pretrained_dict = {k: v for k, v in pretrained_dict.items() if k in model_dict}
    model_dict.update(pretrained_dict)
    self.load_state_dict(model_dict)
```

- SimCLR model

base_encoder=resnet50
out_dim=128

```
class SimCLR(pl.LightningModule):
    def __init__(self, base_encoder, out_dim):
        super(SimCLR, self).__init__()
        self.encoder = base_encoder(num_classes=out_dim)
        dim_mlp = self.encoder.fc.in_features
        self.encoder.fc = nn.Sequential(
            nn.Linear(dim_mlp, dim_mlp),
            nn.ReLU(),
            nn.Linear(dim_mlp, out_dim)
        )
        self.criterion = nn.CrossEntropyLoss()

    def forward(self, x):
        return self.encoder(x)

    def training_step(self, batch, batch_idx):
        (x1, x2), _ = batch
        z1, z2 = self.encoder(x1), self.encoder(x2)
        logits, labels = self.info_nce_loss(z1, z2)
        loss = self.criterion(logits, labels)
        return loss

    def info_nce_loss(self, z1, z2):
        z1 = nn.functional.normalize(z1, dim=1)
        z2 = nn.functional.normalize(z2, dim=1)
        logits = torch.mm(z1, z2.t())
        labels = torch.arange(len(logits)).long().to(logits.device)
        return logits, labels

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=1e-3)
        return optimizer
```

- SimCLR hyperparameter

batch_size=256
epochs = 200

```python
transform = transforms.Compose([
        transforms.Resize((64, 64)),
        transforms.RandomResizedCrop(32),
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.RandomRotation(30),
        transforms.ToTensor(),
])
```

```python
def training_step(self, batch, batch_idx):
        (x1, x2), _ = batch
        z1, z2 = self.encoder(x1), self.encoder(x2)
        logits, labels = self.info_nce_loss(z1, z2)
        loss = self.criterion(logits, labels)
        return loss

def info_nce_loss(self, z1, z2):
        z1 = nn.functional.normalize(z1, dim=1)
        z2 = nn.functional.normalize(z2, dim=1)
        logits = torch.mm(z1, z2.t())
        labels = torch.arange(len(logits)).long().to(logits.device)
        return logits, labels

def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=1e-3)
        return optimizer
```

Result

- without pretrained weight

Resnet50

Dataset :

train, valid, test (each class)= 2320, 580, 100

- with pretrained weight

Resnet50

Dataset :

train, valid, test (each class)= 1520, 380, 100

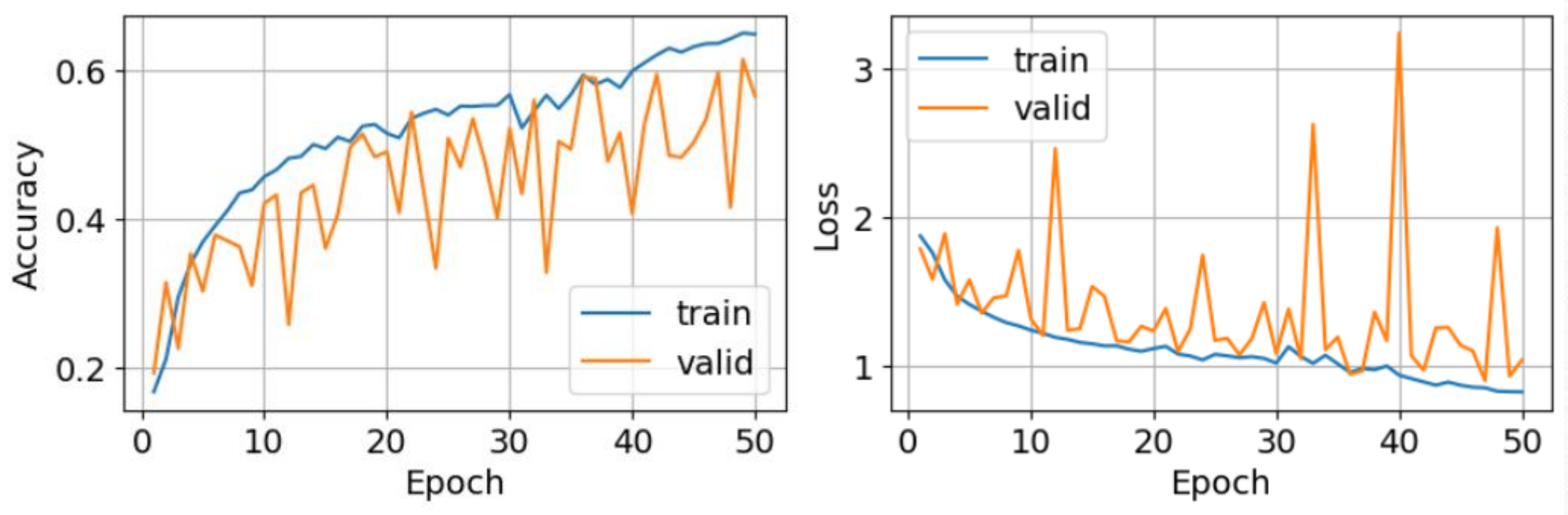
SimCLR

Dataset :

train = 11400

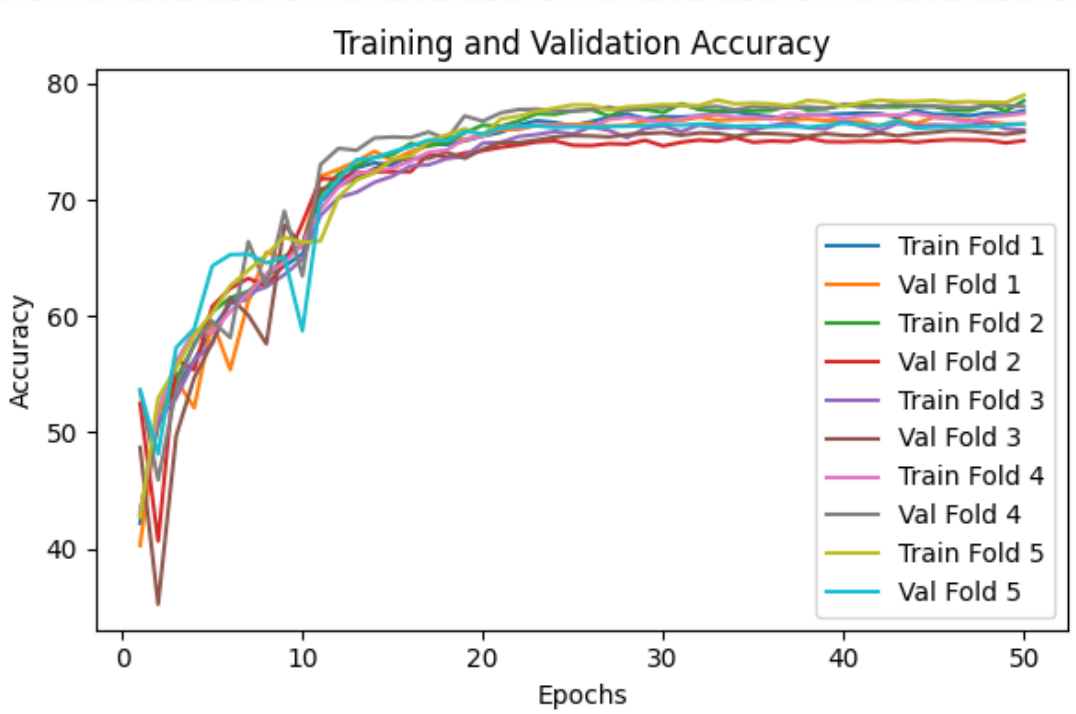
- without pretrained weight

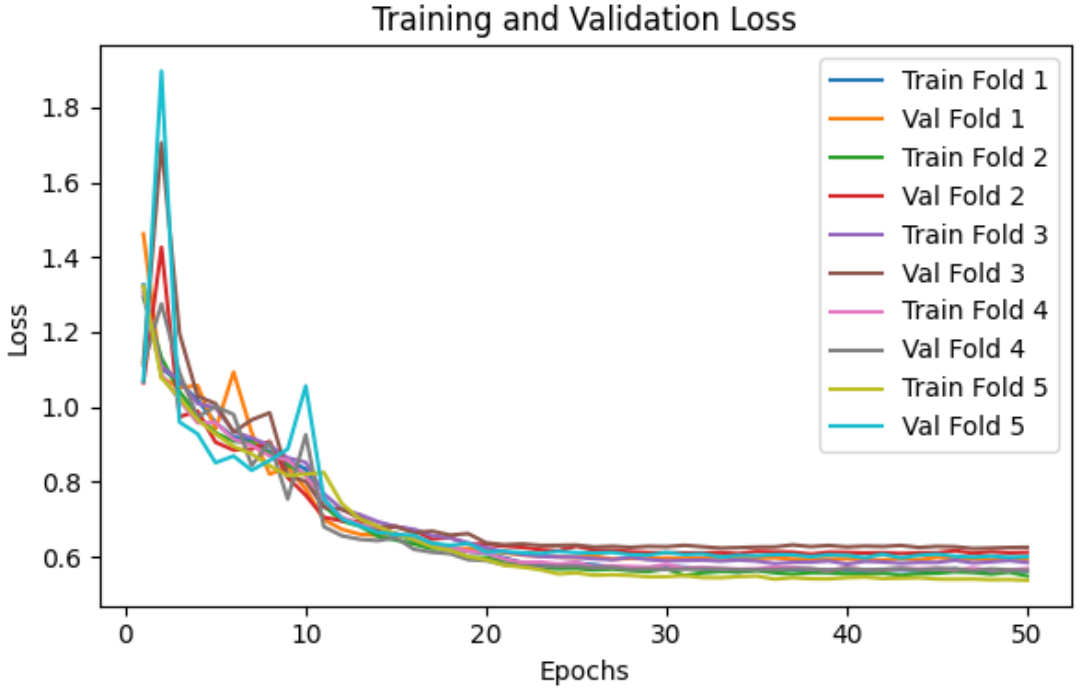
```
Accuracy: 0.5822
Precision: 0.6450
Recall: 0.5822
F1 Score: 0.5676
```



- with pretrained weight

```
Accuracy: 0.7483
Precision: 0.7501
Recall: 0.7483
F1 Score: 0.7488
```





Additional try :

DCGAN: