Creating Illusions:

Synthesizing Dynamic Movements for Non-Experts

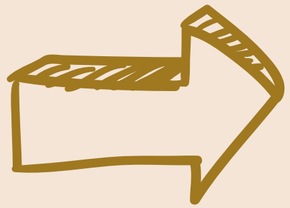
AI 2023 Final Project
Group 28 : 110704031賴怡 110263029楊芊華 109612007吳宜靜

Introduction

Developing an AI model to convert human motion to skeleton and generate an image of another human performing the same motion.

Non-Expert

Expert



Non-Expert
but
performing well

Related Work

-Image-to-Image Translation with Conditional Adversarial Networks

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros CVPR, 2017

-PatchedGAN

Introduced by Isola et al.

-Keypoint R-CNN

Kaiming He Georgia Gkioxari Piotr Dollar Ross Girshick ´
Facebook AI Research (FAIR)

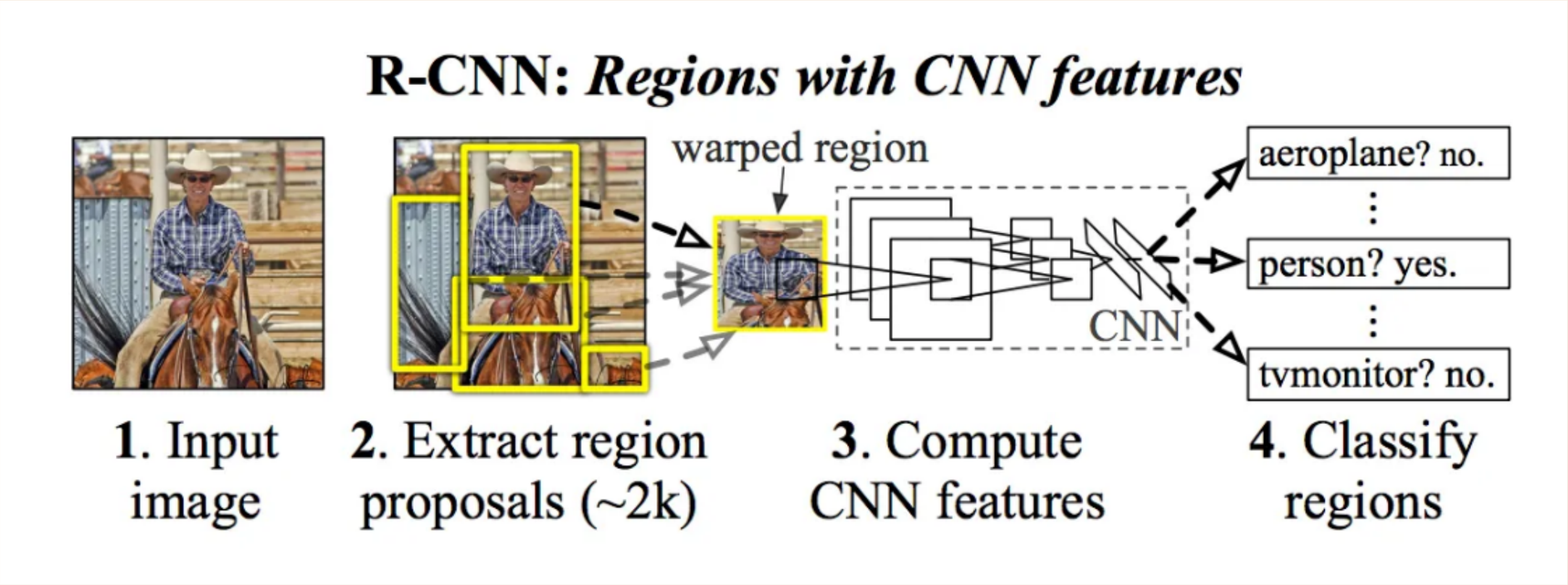
Dataset

- Keypoint Detection - COCO
  - pros:
    -wide range of coverage
    -high-quality annotations
  - dataset selection:
    -get 100,000+ keypoint data
    -remove data with less than 13 keypoints and randomly select 10,000 samples

- Input Video - dancing videos on youtube
  - video segmentation: extract 2,500+ images

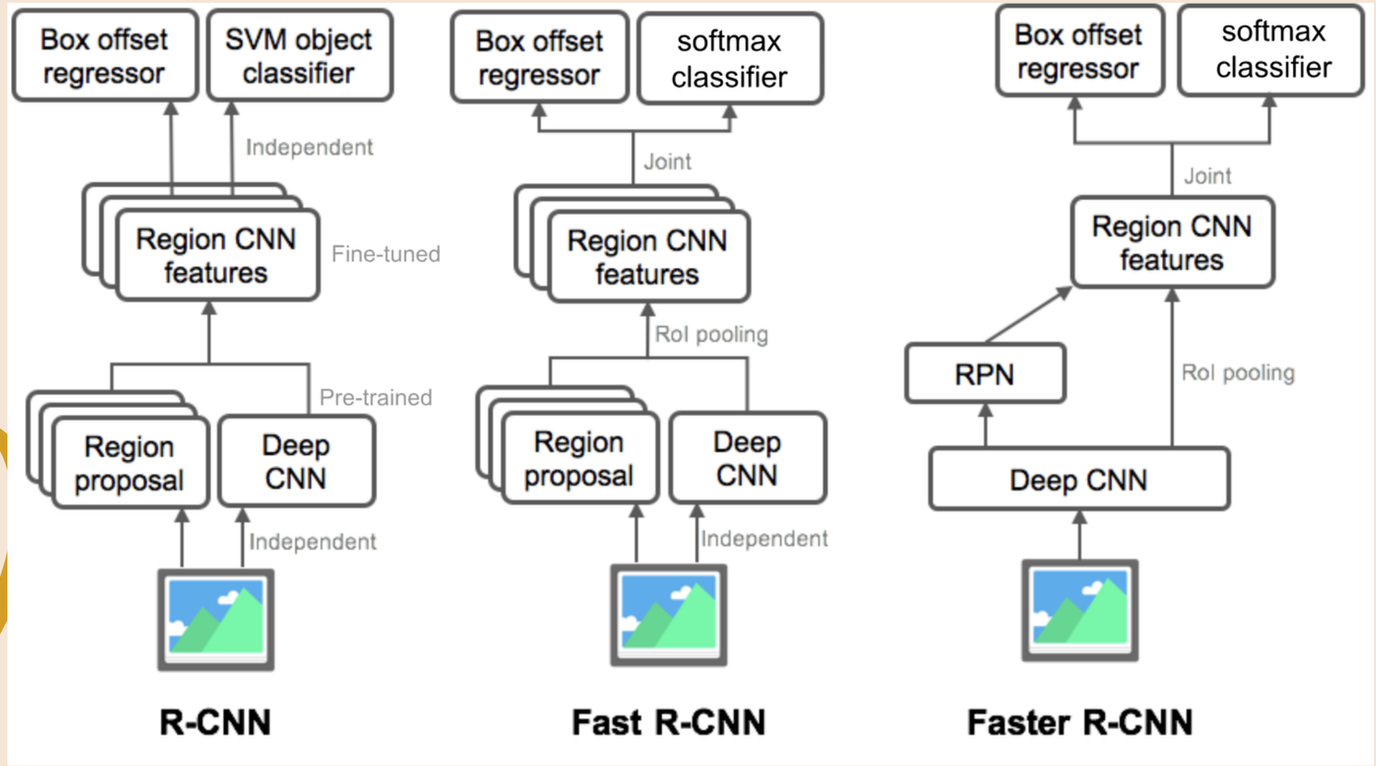
Baseline - image to skeleton

-basis: RCNN

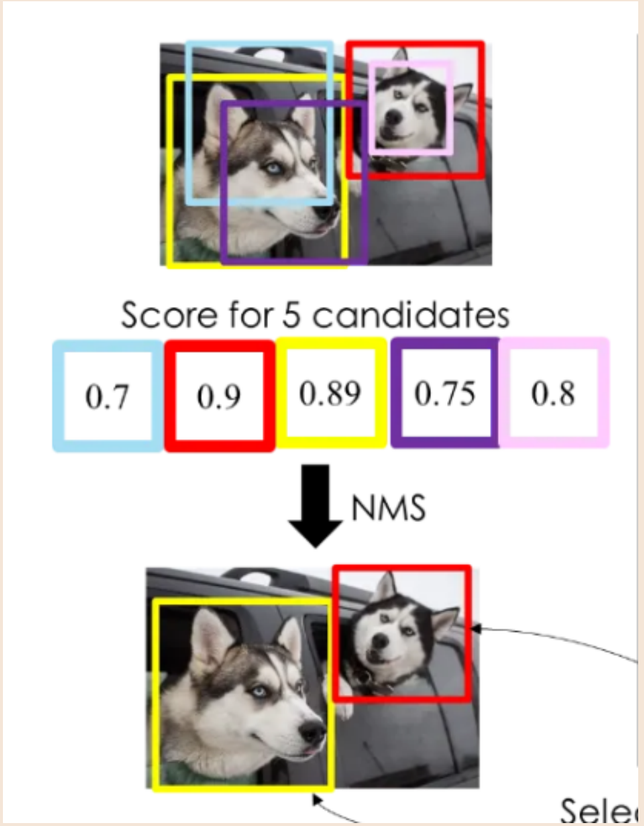


Baseline - image to skeleton

-optimized: Faster RCNN



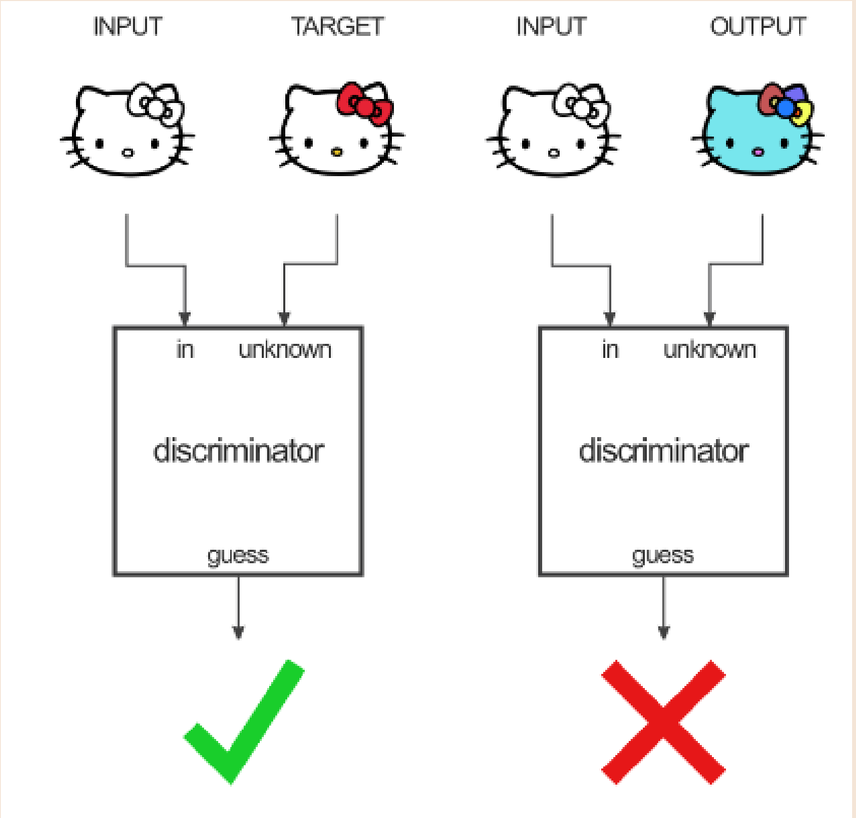
choose the most possible anchor



Baseline - image to image

-Conditional Adversarial Network

- generator produces image
- discriminator identifies whether the image is real or fake(generated)



Baseline - image to image

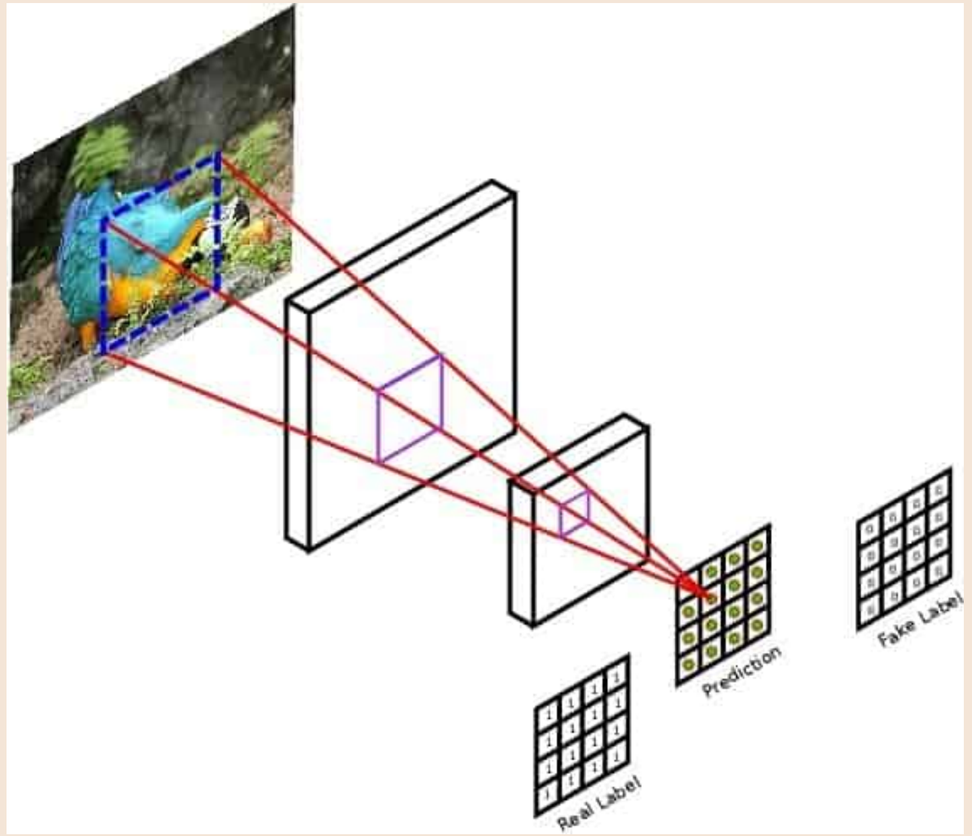
Conditional Adversarial Networks

generator

```
self.inc = Inconv(input_nc, ngf, norm_layer, use_bias)  # input convolutional layer(64 feature maps)
self.down1 = Down(ngf, ngf * 2, norm_layer, use_bias)
self.down2 = Down(ngf * 2, ngf * 4, norm_layer, use_bias)

model = []
# 在model內建立n個殘差塊(resblock)
for i in range(n_blocks):
    model += [ResBlock(ngf * 4, padding_type=padding_type, norm_layer=norm_layer, use_dropout=use_dropo
self.resblocks = nn.Sequential(*model)

self.up1 = Up(ngf * 4, ngf * 2, norm_layer, use_bias)
self.up2 = Up(ngf * 2, ngf, norm_layer, use_bias)

self.outc = Outconv(ngf, output_nc)
```

Baseline - image to image

-discriminator: patchGAN



- like a convolutional network
- chop the image to patches and process them independently

Baseline - image to image

Conditional Adversarial Networks

discriminator

```
sequence = [
    # input channel, output channel(number of feature maps), kernel size, stride, padding(補0)
    nn.Conv2d(input_nc, ndf, kernel_size=kw, stride=2, padding=padw),
    # 上一層神經元給下一層非線性函數轉換 (助於學習)
    nn.LeakyReLU(0.2, True)
]

nf_mult = 1
nf_mult_prev = 1
for n in range(1, n_layers):
    nf_mult_prev = nf_mult
    nf_mult = min(2**n, 8)
    sequence += [
        nn.Conv2d(ndf * nf_mult_prev, ndf * nf_mult,
                  kernel_size=kw, stride=2, padding=padw, bias=use_bias),
        norm_layer(ndf * nf_mult),
        nn.LeakyReLU(0.2, True)
    ]
```

MAIN APPROACH

Keypoint Detection

- Model - based on FasterRCNN
- Train - with some optimizers
- Visualize the result

Image to Image

- Model - GAN
- Train - with some optimizers
- Generate image

MAIN APPROACH

Keypoint Detection

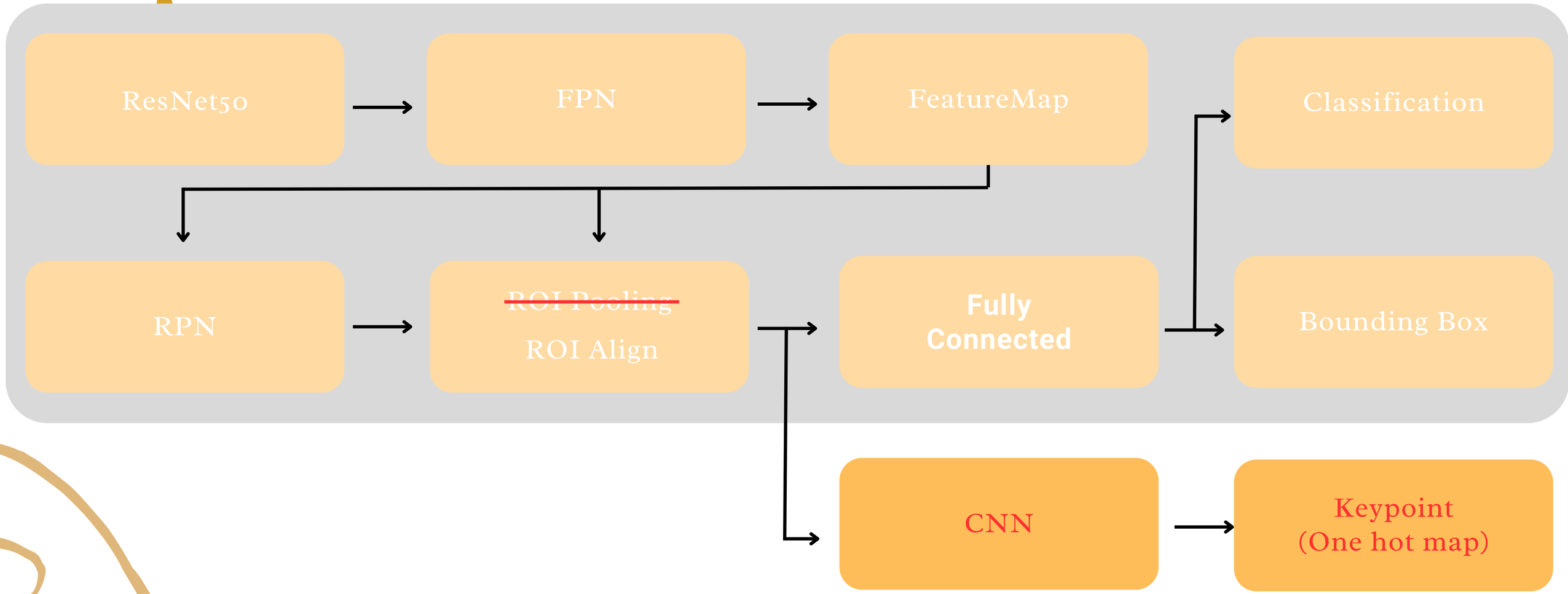
- Model - based on FasterRCNN
- Train - with some optimizers
- Visualize the results

Image to Image

- Model - GAN
- Train - with some optimizers
- Generate image

Model

Faster RCNN



Model-CNN

```python
def __init__(self, in_channels, layers):
    super(KeypointRCNNHeads, self).__init__()
    self.model = nn.Sequential(
        nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),
        nn.ReLU(inplace=True),
        nn.BatchNorm2d(512),
        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
        nn.ReLU(inplace=True),
        nn.BatchNorm2d(512),
        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
        nn.ReLU(inplace=True),
        nn.BatchNorm2d(512),
        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
        nn.ReLU(inplace=True),
        nn.BatchNorm2d(512),
        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
        nn.ReLU(inplace=True),
        nn.BatchNorm2d(512),
        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
        nn.ReLU(inplace=True),
        nn.BatchNorm2d(512),
        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
        nn.ReLU(inplace=True),
        nn.BatchNorm2d(512),
        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
        nn.ReLU(inplace=True),
        nn.BatchNorm2d(512),
    )
```

```python
def __init__(self):
    super(KeypointRCNNPredictor, self).__init__()
    self.kps_score_lowres = nn.ConvTranspose2d(512, 17, kernel_size=4, stride=2, padding=1)
```

Batch Normalization: normalizes the inputs of each mini-batch

- accelerate convergence
- keep inputs distribution stable at each layer
- improve stability and accuracy

ConvTranspose2d: increases the size of a feature map

- convert low-resolution feature maps to high-resolution ones
- aiding in detail restoration and improving spatial accuracy.

Train

STEP1: Loading data

- Transform: increasing the diversity of training data
- Using RandomSampler to randomly samples elements from the dataset
- Using BatchSampler to divide the samples into batches then using data loader
- Finally we can use the dataset from the data loader to train the model

```
def get_transform(train):
    transforms = []
    transforms.append(T.ToTensor())
    if train:
        transforms.append(T.RandomHorizontalFlip(0.5))
    return T.Compose(transforms)
```

```
num = list(range(1, len(dataset_test), 2))
dataset_test = torch.utils.data.Subset(dataset_test, num)

train_sampler = torch.utils.data.RandomSampler(dataset)
test_sampler = torch.utils.data.RandomSampler(dataset_test)
train_batch_sampler = torch.utils.data.BatchSampler(train_sampler, args.batch_size, drop_last=True)
```

Train

STEP2: Optimize and train the model

- Using SGD + momentum: the update step for the model parameters is determined by both the current gradient and the accumulated momentum from previous iterations.
- learning rate scheduler: multiplies the learning rate by a specified factor at predetermined steps -> lr = lr * gamma
- Training the model

```
# optimize the model
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=args.lr, momentum=args.momentum, weight_decay=args.weight_decay)

#adjust learning rate every step_size lr = lr*gamma
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=args.lr_step_size, gamma=args.lr_gamma)
```

```
for epoch in range(args.epochs):
    train_one_epoch(model, optimizer, data_loader, device, epoch, args.print_freq)
    lr_scheduler.step()
    evaluate(model, data_loader_test, device=device) # evaluate after every epoch
    torch.save(model.state_dict(), 'training_weights'+str(i)+'.pth') # save model /epoch
```

Visualize

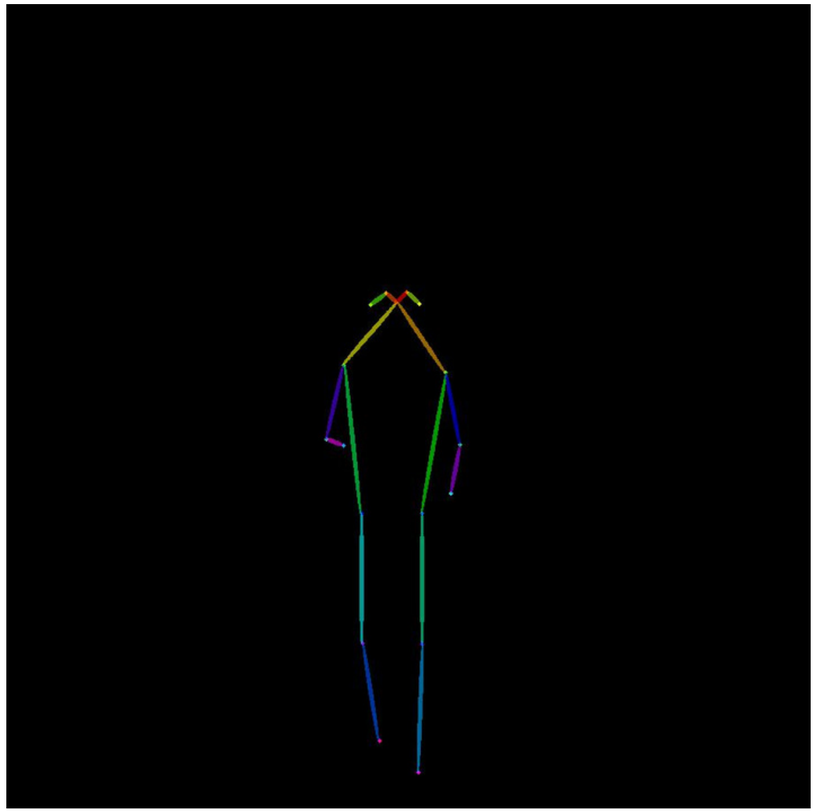
STEP3: Plot the results

- Load the training weight for model
- Using detect_threshold, keypoint_score_threshold parameters to filter out individuals with low scores and keypoints with low scores,

```
model = get_model(train_weight='training_weights.pth')
```

```
detect_threshold = 0.7
keypoint_score_threshold = 2
with torch.no_grad():
    for i in range(10):
        print(i)
        img, _ = dataset_test[i]
        prediction = model([img.to(device)])
        keypoints = prediction[0]['keypoints'].cpu().numpy()
        scores = prediction[0]['scores'].cpu().numpy()
        keypoints_scores = prediction[0]['keypoints_scores'].cpu().numpy()
        idx = np.where(scores>detect_threshold)
        keypoints = keypoints[idx]
        keypoints_scores = keypoints_scores[idx]
        for j in range(keypoints.shape[0]):
            for num in range(17):
                if keypoints_scores[j][num]<keypoint_score_threshold:
                    keypoints[j][num]=[0,0,0]
        img = img.mul(255).permute(1, 2, 0).byte().numpy()
        plot_poses(img,keypoints,save_name='./result/'+str(i)+'.jpg')
print('Finish!')
```

Visualize



MAIN APPROACH

Keypoint Detection

- Model - based on FasterRCNN
- Train - with some optimizers
- Visualize the result

Image to Image

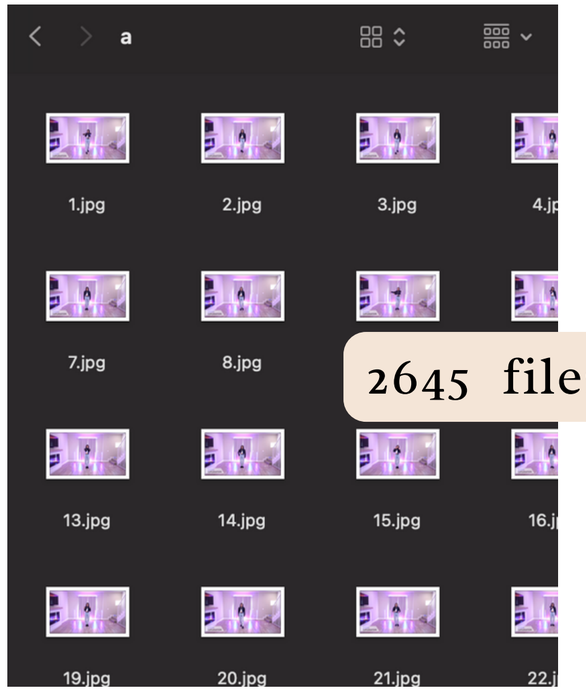
- Model - GAN
- Train - with some optimizers
- Generate image

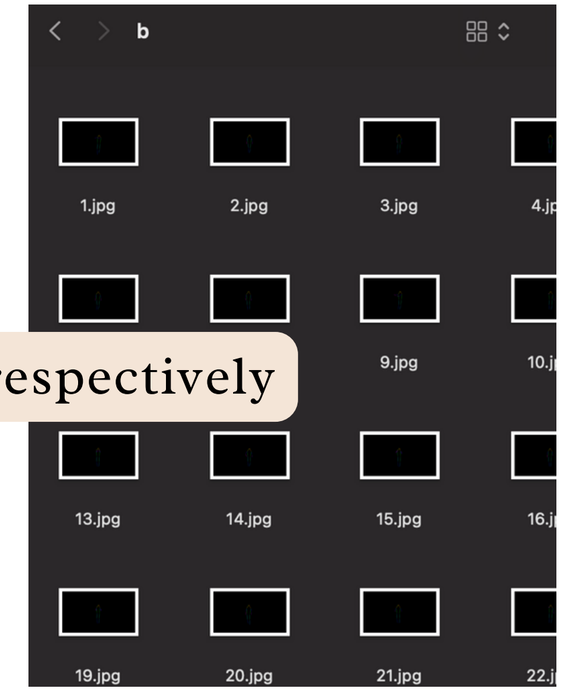
Train GAN model

STEP 1 : Loading Dataset

- training data : 2645 files, testing data : 6 files

```
print('===> Loading datasets')
root_path = "dataset1/"
train_set = get_training_set(root_path + opt.dataset, opt.direction)
test_set = get_test_set(root_path + opt.dataset, opt.direction)
training_data_loader = DataLoader(dataset=train_set, num_workers=opt.threads, batch_size=opt.batch_size, shuffle=True)
testing_data_loader = DataLoader(dataset=test_set, num_workers=opt.threads, batch_size=opt.test_batch_size, shuffle=False)
```





2645 files respectively

6 files respectively

Train GAN model

STEP 2 : Model Definition

- Define Generator, Discriminator, GAN loss function

```
print('===> Building models')
net_g = define_G(opt.input_nc, opt.output_nc, opt.ngf, 'batch', False, 'normal', 0.02, gpu_id=device)
net_d = define_D(opt.input_nc + opt.output_nc, opt.ndf, 'basic', gpu_id=device)

criterionGAN = GANLoss().to(device)
criterionL1 = nn.L1Loss().to(device)
criterionMSE = nn.MSELoss().to(device)
```

- Setup Optimizer and learning rate scheduler

```
optimizer_g = optim.Adam(net_g.parameters(), lr=opt.lr, betas=(opt.beta1, 0.999))
optimizer_d = optim.Adam(net_d.parameters(), lr=opt.lr, betas=(opt.beta1, 0.999))
net_g_scheduler = get_scheduler(optimizer_g, opt)
net_d_scheduler = get_scheduler(optimizer_d, opt)
```

Train GAN model

Introduction to optimizer

SGD (stochastic gradient decent)	calculate weight : gradient of the loss function by differentiation update parameters : opposite direction of gradient and learning rate $W \leftarrow W - \eta \frac{\partial L}{\partial W}$
Momentum SGD	allowing for faster updates in the same direction $V_t \leftarrow \beta V_{t-1} - \eta \frac{\partial L}{\partial W}$
Adam	Keeps the exponentially decaying average of the past gradient keeps the squared decaying average of the past gradient

Train GAN model

STEP3 : Training Loop

- Updates the discriminator

```
for iteration, batch in enumerate(training_data_loader, 1): # forward
    real_a, real_b = batch[0].to(device), batch[1].to(device)
    fake_b = net_g(real_a)

    ######################
    # (1) Update D network
    ######################

    optimizer_d.zero_grad()

    # train with fake
    fake_ab = torch.cat((real_a, fake_b), 1)
    pred_fake = net_d.forward(fake_ab.detach())
    loss_d_fake = criterionGAN(pred_fake, False)

    # train with real
    real_ab = torch.cat((real_a, real_b), 1)
    pred_real = net_d.forward(real_ab)
    loss_d_real = criterionGAN(pred_real, True)

    # Combined D loss
    loss_d = (loss_d_fake + loss_d_real) * 0.5

    loss_d.backward()

    optimizer_d.step()
```

generator

loss calculations are performed for the generated fake images and real images and the model weights are updated based on these losses.

```
    ######################
    # (2) Update G network
    ######################

    optimizer_g.zero_grad()

    # First, G(A) should fake the discriminator
    fake_ab = torch.cat((real_a, fake_b), 1)
    pred_fake = net_d.forward(fake_ab)
    loss_g_gan = criterionGAN(pred_fake, True)

    # Second, G(A) = B
    loss_g_l1 = criterionL1(fake_b, real_b) * opt.lamb

    loss_g = loss_g_gan + loss_g_l1

    loss_g.backward()

    optimizer_g.step()

    print("===> Epoch[{}]({}/{}): Loss_D: {:.4f} Loss_G: {:.4f}".format(
        epoch, iteration, len(training_data_loader), loss_d.item(), loss_g.item()))
```

Train GAN model

STEP4 : Model preservation and testing

Filename	Filesize	Filetype
..		
netG_model_epoch_20.pth	45637225	pth-file
netD_model_epoch_20.pth	11091427	pth-file
netG_model_epoch_10.pth	45637225	pth-file
netD_model_epoch_10.pth	11091427	pth-file
netG_model_epoch_200.pth	45637225	pth-file
netD_model_epoch_200.pth	11091427	pth-file
netG_model_epoch_190.pth	45637225	pth-file
netD_model_epoch_190.pth	11091427	pth-file
netG_model_epoch_180.pth	45637225	pth-file
netD_model_epoch_180.pth	11091427	pth-file
netG_model_epoch_170.pth	45637225	pth-file
netD_model_epoch_170.pth	11091427	pth-file
netG_model_epoch_160.pth	45637225	pth-file
netD_model_epoch_160.pth	11091427	pth-file
netG_model_epoch_150.pth	45637225	pth-file
netD_model_epoch_150.pth	11091427	pth-file
netG_model_epoch_140.pth	45637225	pth-file
netD_model_epoch_140.pth	11091427	pth-file
netG_model_epoch_130.pth	45637225	pth-file
netD_model_epoch_130.pth	11091427	pth-file
netG_model_epoch_120.pth	45637225	pth-file

120 files. Total size: 3403719120 bytes

```python
# test
avg_psnr = 0
for batch in testing_data_loader:
    input, target = batch[0].to(device), batch[1].to(device)

    prediction = net_g(input)
    mse = criterionMSE(prediction, target)
    psnr = 10 * log10(1 / mse.item())
    avg_psnr += psnr
print("===> Avg. PSNR: {:.4f} dB".format(avg_psnr / len(testing_data_loader)))
```

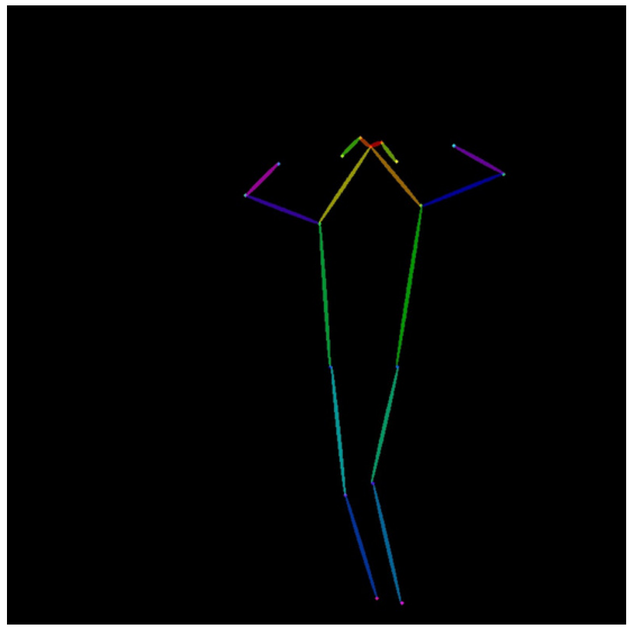
Performance metric : PSNR (peak signal-to-noise ratio)
a common measure of image quality.

```
===> Epoch[15](2643/2645): Loss_D: 0.0014 Loss_G: 2.1399
===> Epoch[15](2644/2645): Loss_D: 0.0487 Loss_G: 2.1273
===> Epoch[15](2645/2645): Loss_D: 0.0292 Loss_G: 2.2286
learning rate = 0.0008000
learning rate = 0.0008000
===> Avg. PSNR: 6.9454 dB
===> Epoch[16](1/2645): Loss_D: 0.0153 Loss_G: 2.0304
===> Epoch[16](2/2645): Loss_D: 0.0111 Loss_G: 1.9949
===> Epoch[16](3/2645): Loss_D: 0.0031 Loss_G: 1.9677
```

Image Generation

STEP : Image Transform

Test Dataset



Result

Evaluation Metric

DATA ACCURACY

- IoU = 0.5
- Precision
- Compare the accuracy at each epoch

RUNNING TIME

- End time -Start time

Results & Analysis

Keypoint Detection:

- Learning rate
- Optimizer: SGD, SGD+momentum, Adagrade
- add BatchNorm2d

Image to Image

- find best optimizer
- find optimal learning rate
- find optimal epoch

Results & Analysis

Keypoint Detection:

- Learning rate
- Optimizer: SGD, SGD+momentum, Adagrade
- add BatchNorm2d

Image to Image

- find best optimizer
- find optimal learning rate
- find optimal epoch

Results & Analysis

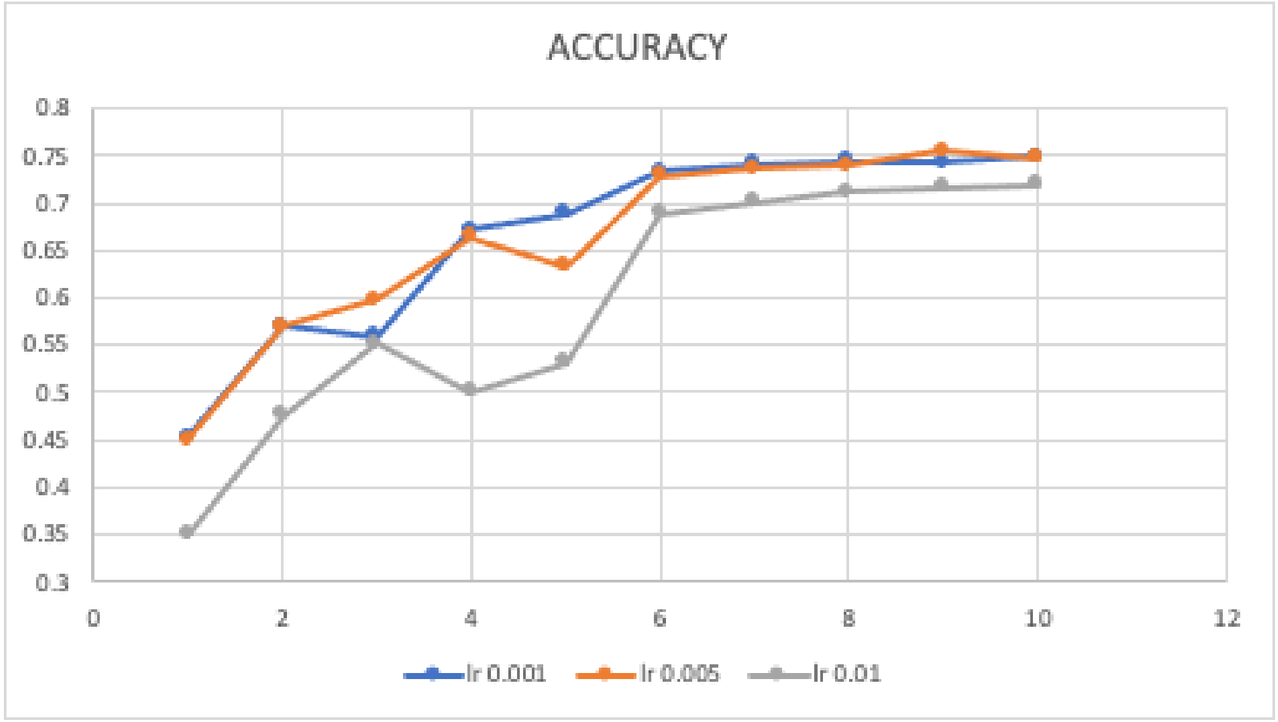
OPTIMAL LEARNING RATE

Result

Best performance:
learning rate = 0.001, 0.005

Analysis

With learning rates 0.001 and 0.005, the accuracies are relatively high and show less fluctuation across iterations.
In contrast, the accuracy is lower for the learning rate of 0.01, and there is more significant fluctuation across iterations.



Results & Analysis

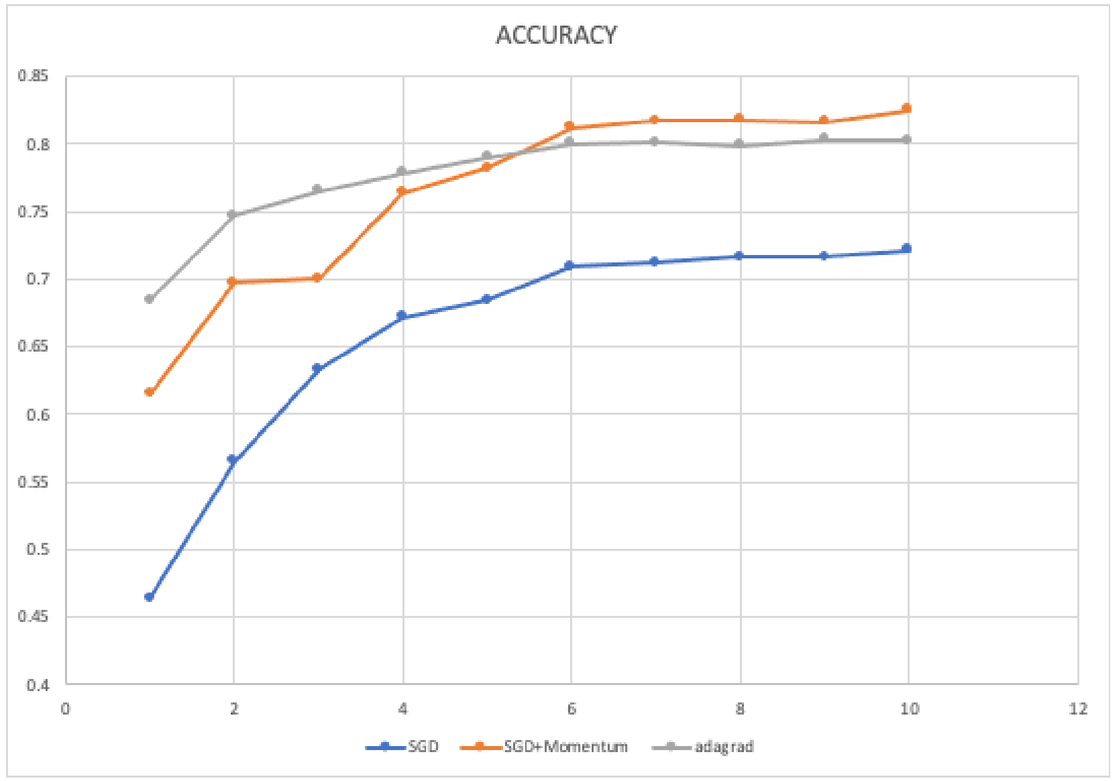
OPTIMIZER: Adagrad v.s SGD v.s SGD+Momentum

Result

Best performance:
SGD+Momentum

Analysis

- Adagrad's characteristic is to adaptively adjust the learning rate based on the historical gradient information of parameters.it may cause the learning rate to decrease excessively
- SGD+Momentum optimizer incorporates the concept of momentum, which helps it better handle the direction and magnitude of gradients during optimization

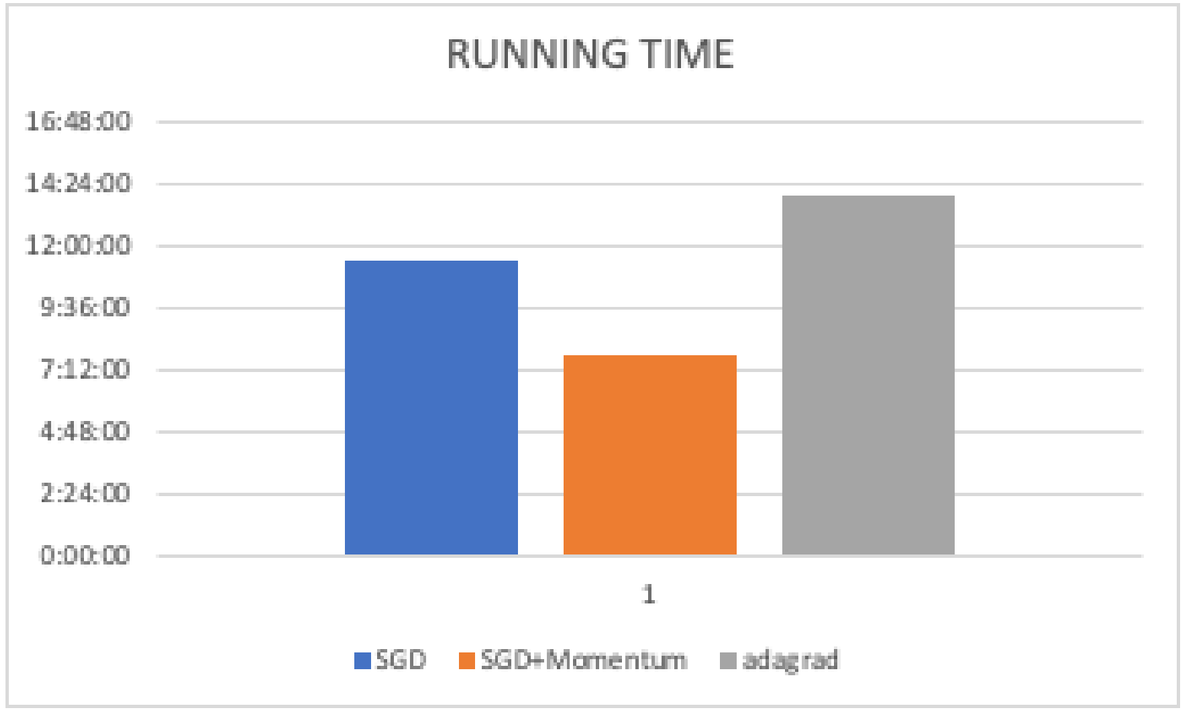


Results & Analysis

OPTIMIZER: Adagrad v.s SGD v.s SGD+Momentum

Result

SGD+Momentum has the shortest training time



Analysis

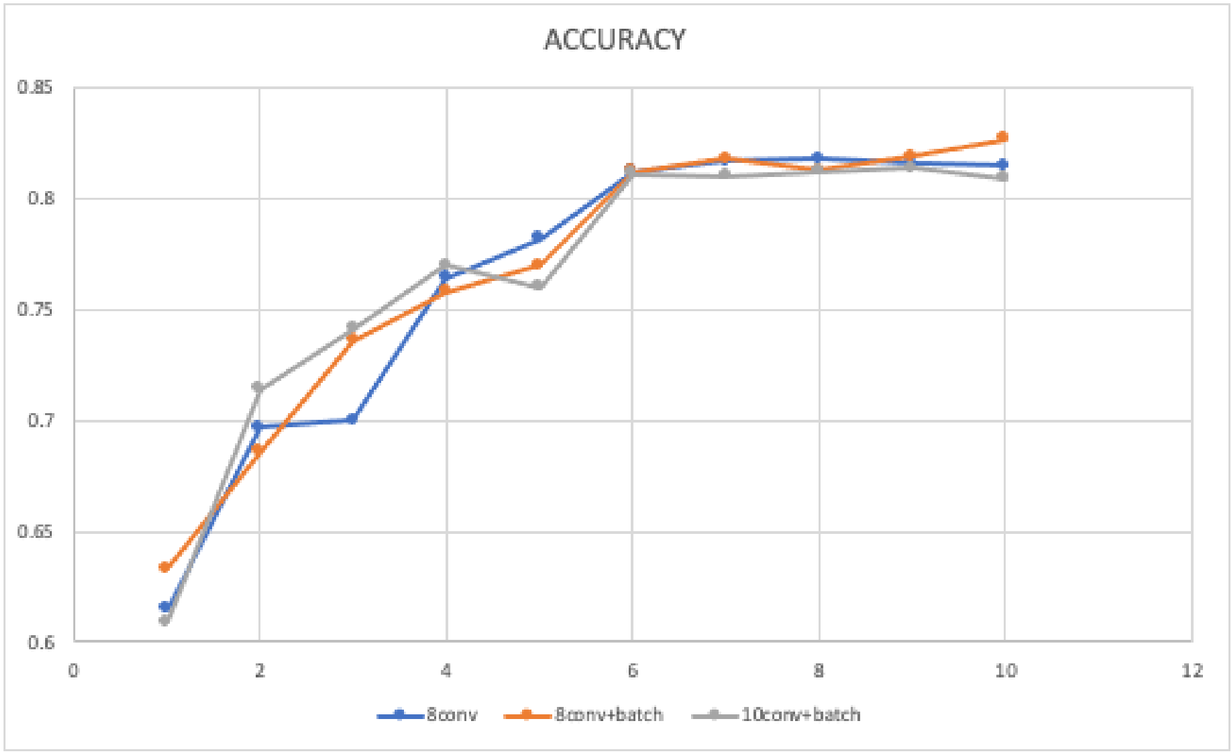
- SGD+momentum could be due to the introduction of the momentum term.
- adagrad: lead to excessive learning rate decay and slower convergence

Results & Analysis

Layer: 8*conv2d v.s 8*conv2d + BatchNorm2d layer v.s 10*conv2+BatchNorm2d

Result

8*conv2d+batchNorm2d›
8*conv2d›
10*conv2d+batchNorm2d



Analysis

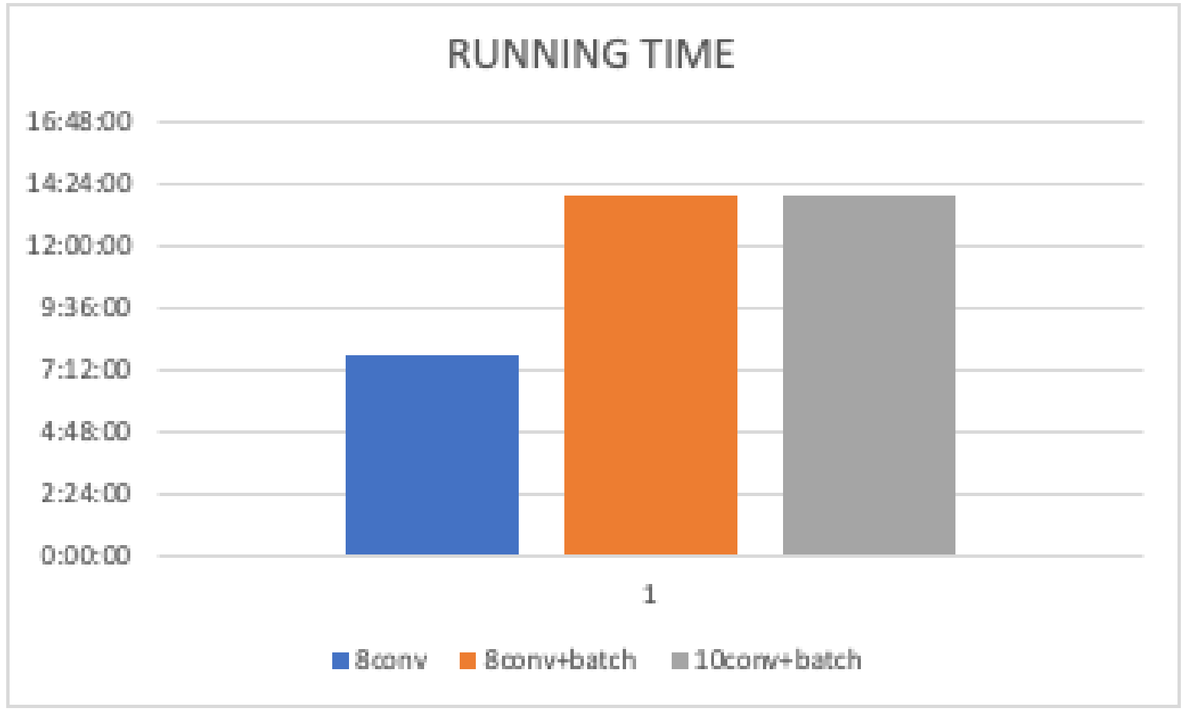
8*conv2d+batchNorm2d: adding batch normalization improves the model's performance.

Results & Analysis

Layer: 8*conv2d v.s 8*conv2d + BatchNorm2d layer v.s 10*conv2+BatchNorm2d

Result

8*conv2d has the shortest training time



Analysis

- The 8conv model has a shorter training time due to its fewer convolutional layers and parameters.
- The 8conv+batch and 10conv+batch models have longer training times due to their higher number of convolutional layers and parameters.

Results & Analysis

Keypoint Detection:

- Learning rate
- Optimizer: SGD, SGD+momentum, Adagrade
- add BatchNorm2d

Image to Image

- find best optimizer
- find optimal learning rate
- find optimal epoch

Results & Analysis

Task 1: Optimizer

	Momentum SGD	Adam
2 Epoch		
40 Epoch		

test data : 488.jpg
learning rate : 0.0002

Results & Analysis

Task 2: Learning Rate

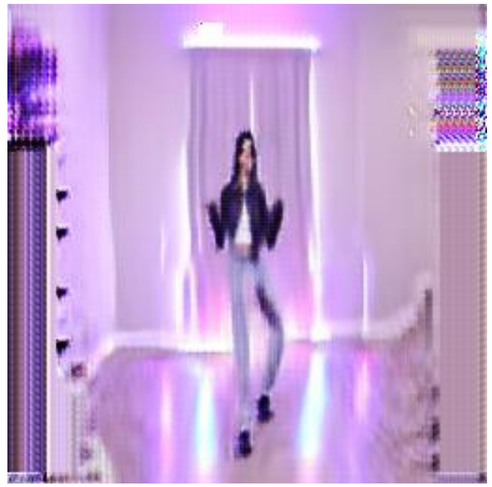
20 Epoch

0.0004

0.001

0.002

0.004



test data : 1088.jpg
optimizer : Adam

Results & Analysis

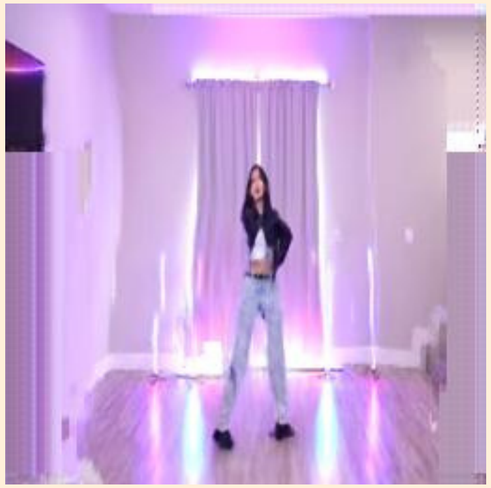
Task 2: Learning Rate

	0.0008	0.001	0.0012	0.0015
20 Epoch				

test data : 1088.jpg
optimizer : Adam

Results & Analysis

Task 3: Epoch

	40	100	150	200 E
lr 0.001				

test data : 688.jpg
optimizer : Adam

References

Our Github Repo

https://github.com/Joannaaaaaa/Synthesizing-Dynamic-Movements-for-Non-Experts

Contribution of each member

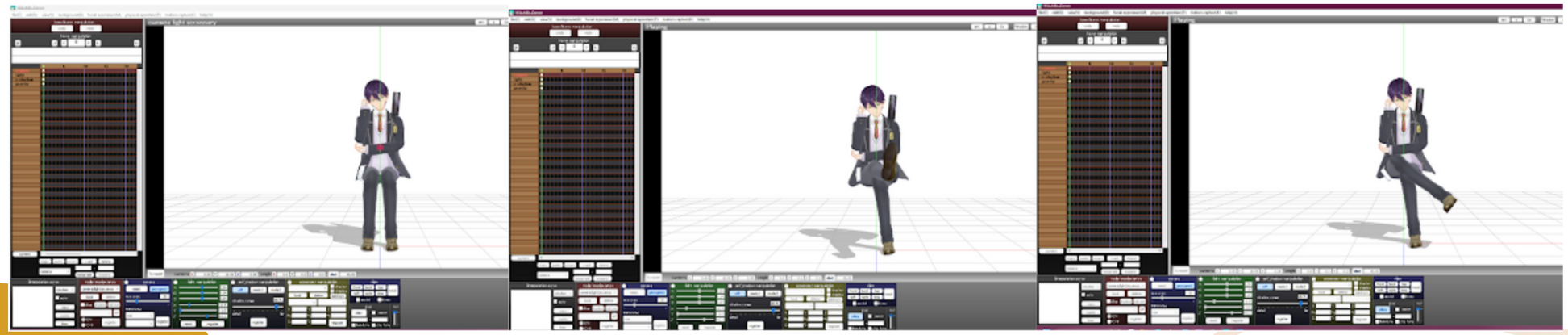
image to image - 楊芊華(35%)、賴怡暄(30%)

video to image、keypoint detection - 吳宜靜(35%)

Future

Tote bag with durable leather handle for heavier things

Presentations are tools that can be used as lectures, speeches, reports, and more. It is mostly presented before an audience. It serves a variety of purposes, making presentations powerful tools for convincing and teaching.



Thank You