Exercises Week 5

Task 1

```python
def __getitem__(self, index):
    current_path = self.triplets[index] # e.g. triplet triplet_000010

    # Make sure you understand the __init__ function and the structure of

    # TASK 1: Read the triplet and make sure you use the self.transform
    frame_1 = os.path.join(self.triplets_path, current_path, "1.png")
    frame_2 = os.path.join(self.triplets_path, current_path, "2.png")
    frame_3 = os.path.join(self.triplets_path, current_path, "3.png")

    frame_1 = Image.open(frame_1)
    frame_2 = Image.open(frame_2)
    frame_3 = Image.open(frame_3)

    frame_1 = self.transform(frame_1)
    frame_2 = self.transform(frame_2)
    frame_3 = self.transform(frame_3)

    return frame_1, frame_2, frame_3
```

Task 2

```python
os.makedirs('weights', exist_ok=True)
for epoch in tqdm(range(NUM_EPOCHS), desc=f'Training for conf {conf}'):
    model.train()
    total_train_loss = 0
    for f1, f2, f3 in train_loader:
        # TASK 2: Implement the training loop
        output = model(f1, f3)

        batch_loss = loss_fn(output, f2)
        optimizer.zero_grad()
        batch_loss.backward()
        optimizer.step()
        total_train_loss += batch_loss.item()

    total_train_loss /= len(train_loader)
    logger.add_scalar('Train loss',total_train_loss, global_step=epoch+1)
torch.save(model.state_dict(), os.path.join('weights', f'conf-{conf}.pth'))
```

Task 3

```python
def forward(self, input, target):
    """
    Input is the prediction (output of the model)
    Target is the target image.
    """
    total_loss = 0
    for loss_func, weight in self.losses_dict.items():
        if loss_func == 'rec_loss': # Reconstruction loss
            tmp_loss = self.l1_loss(input['output_im'], target) # TASK 3
```

Task 4

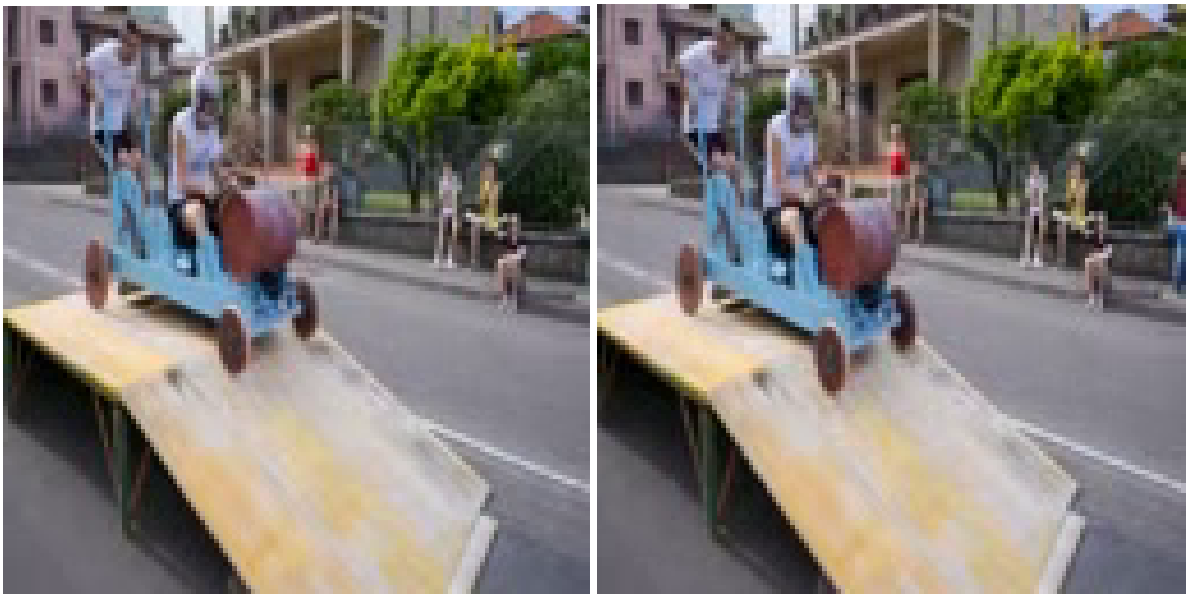
```python
def forward(self, input, target):
    """
    Input is the prediction (output of the model)
    Target is the target image.
    """
    total_loss = 0
    for loss_func, weight in self.losses_dict.items():
        if loss_func == 'rec_loss': # Reconstruction loss
            tmp_loss = self.l1_loss(input['output_im'], target) # TASK 3

        elif loss_func == 'bidir_rec_loss': # Bidirectional reconstruction loss
            tmp_loss = self.l1_loss(input['interp0'], target) + self.l1_loss(input['interp2'], target) # TASK 4
```

Task 5

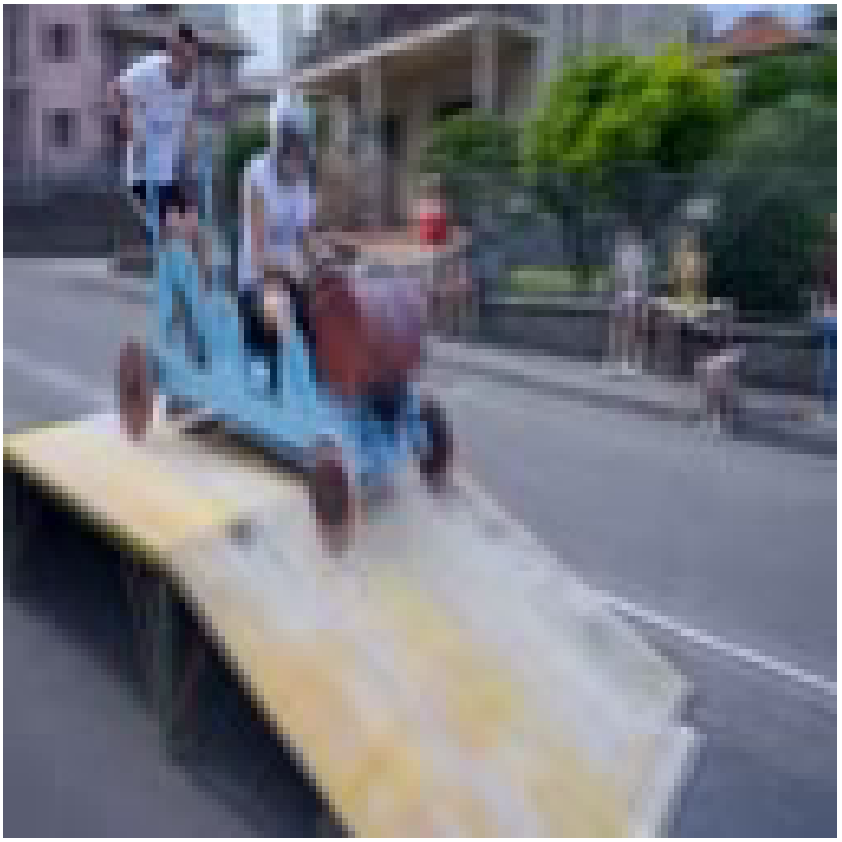
```python
def forward(self, input, target):
    """
    Input is the prediction (output of the model)
    Target is the target image.
    """
    total_loss = 0
    for loss_func, weight in self.losses_dict.items():
        if loss_func == 'rec_loss': # Reconstruction loss
            tmp_loss = self.l1_loss(input['output_im'], target) # TASK 3

        elif loss_func == 'bidir_rec_loss': # Bidirectional reconstruction loss
            tmp_loss = self.l1_loss(input['interp0'], target) + self.l1_loss(input['interp2'], target)

        elif loss_func == 'feature_loss': # Feature Loss
            input_features = self.vgg(input['output_im'])
            target_features = self.vgg(target)
            tmp_loss = self.l2_loss(input_features, target_features)
```

Evaluation

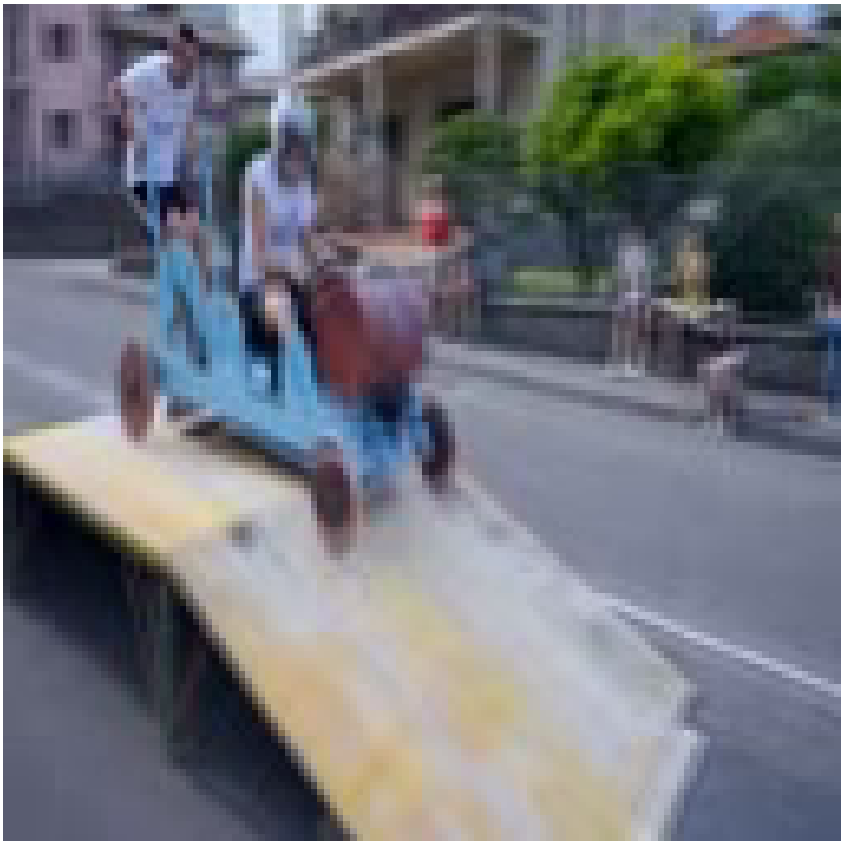
The three losses produced *very* similar looking slowmotion videos. Below, we will compare an interpolated frame for each of the three loss-type trained models.
Firstly, we have the original frames 0 and 2 for the given interpolation here:



Now we see the three loss-types.
First, only reconstruction loss, which looks like its the two frames merely added together.



Second, reconstruction loss and bidirectional loss, which I deem is almost identical to the one above:



Third, reconstruction loss, bidirectional loss and feature loss, which looks a lot smoother than the first two, being more a motion blur between frames than a combination of the two: