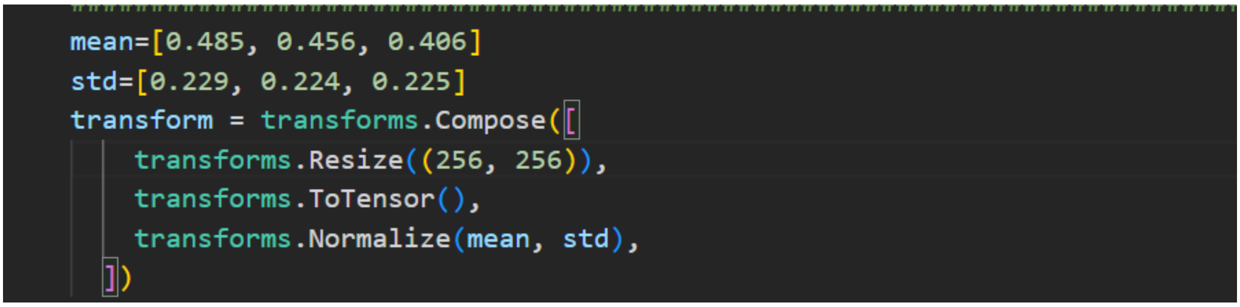
Assignment 6 Part 1: Image Captioning

## Garbage Output_1

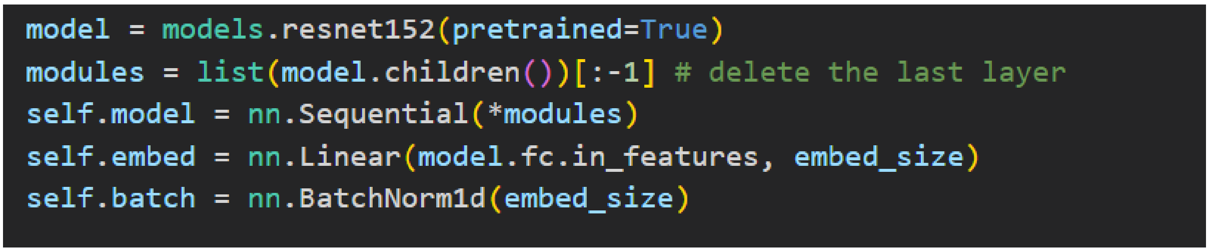
- Data Augmentation



```
mean=[0.485, 0.456, 0.406]
std=[0.229, 0.224, 0.225]
transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize(mean, std),
])
```

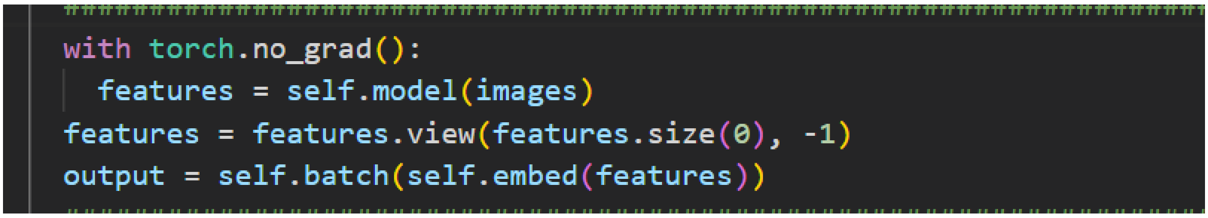
- Encoder Model

__init__()



```
model = models.resnet152(pretrained=True)
modules = list(model.children())[:-1] # delete the last layer
self.model = nn.Sequential(*modules)
self.embed = nn.Linear(model.fc.in_features, embed_size)
self.batch = nn.BatchNorm1d(embed_size)
```

forward()

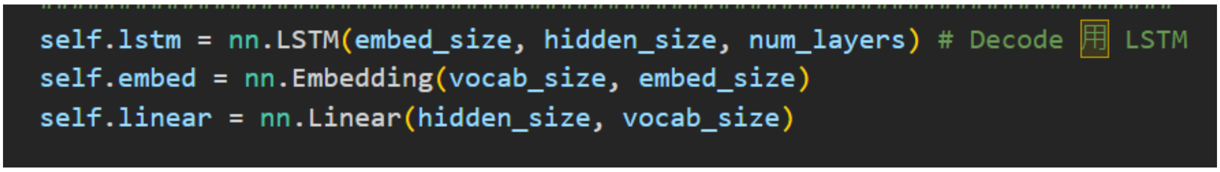


```
with torch.no_grad():
    features = self.model(images)
features = features.view(features.size(0), -1)
output = self.batch(self.embed(features))
```

- 這裡我使用的 pretrained model 是 **Resnet152**

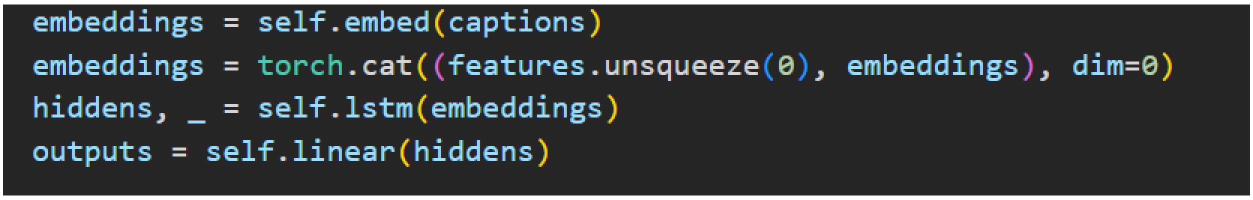
- Decoder Model

__init__()



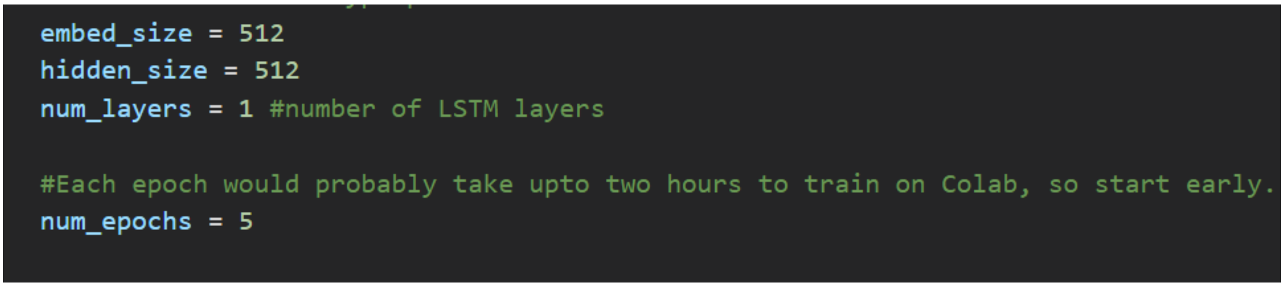
```
self.lstm = nn.LSTM(embed_size, hidden_size, num_layers) # Decode 用 LSTM
self.embed = nn.Embedding(vocab_size, embed_size)
self.linear = nn.Linear(hidden_size, vocab_size)
```

forward()

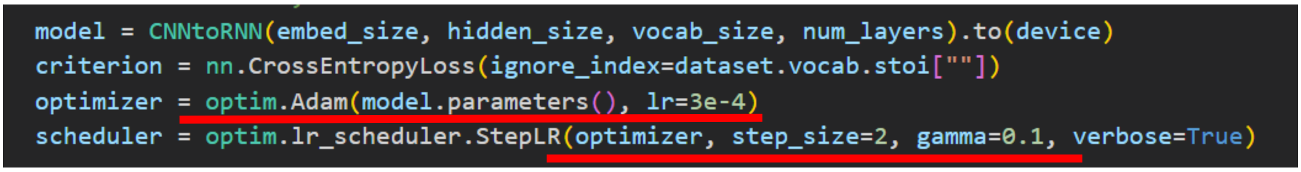


```
embeddings = self.embed(captions)
embeddings = torch.cat((features.unsqueeze(0), embeddings), dim=0)
hiddens, _ = self.lstm(embeddings)
outputs = self.linear(hiddens)
```

- Training Settings

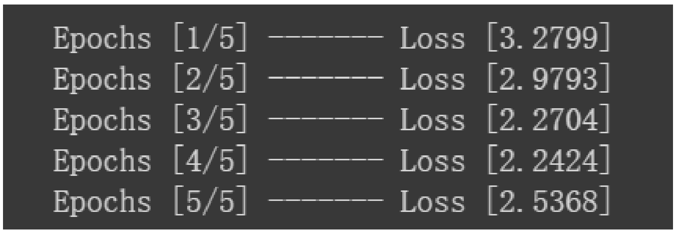


```
embed_size = 512
hidden_size = 512
num_layers = 1 #number of LSTM layers

#Each epoch would probably take upto two hours to train on Colab, so start early.
num_epochs = 5
```



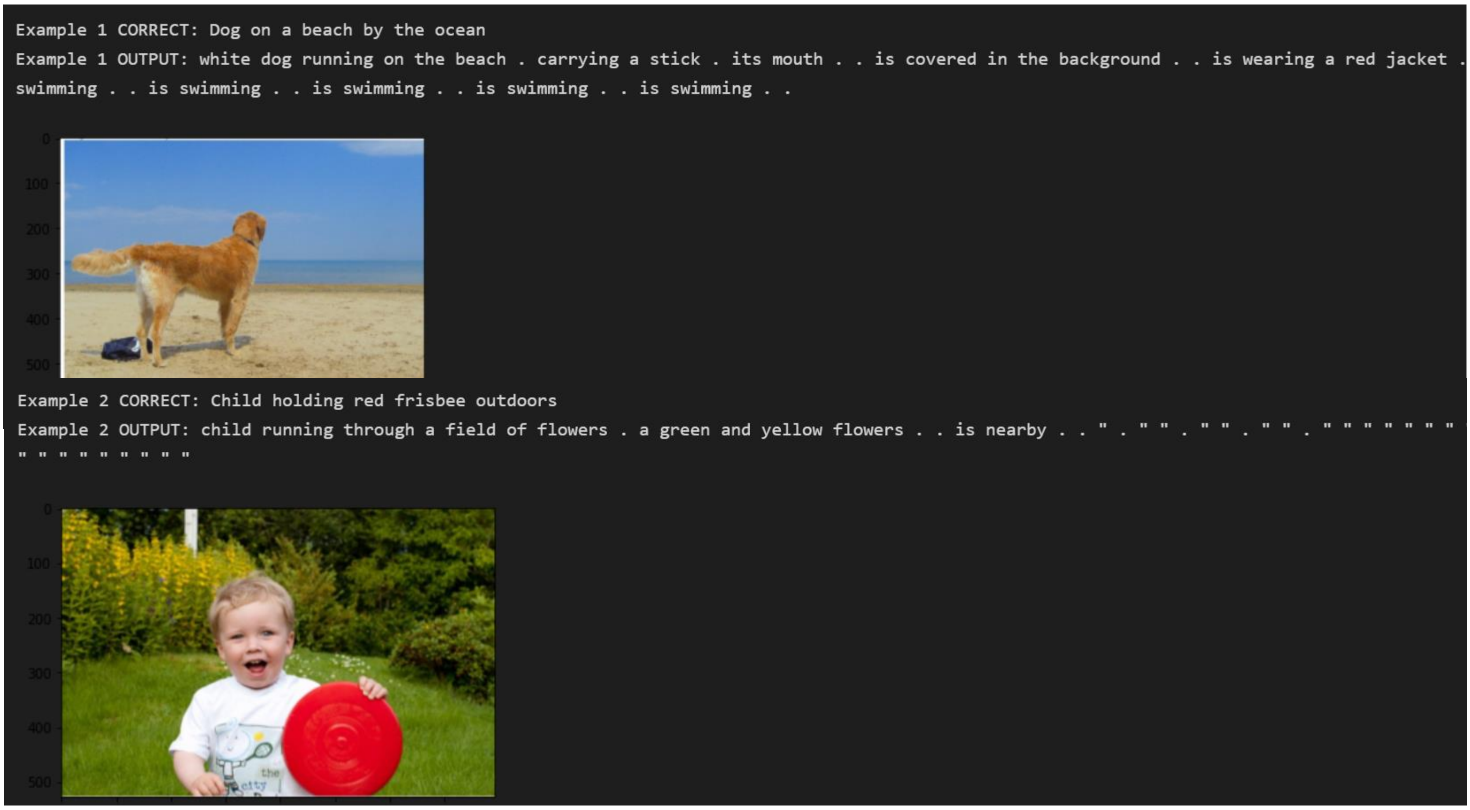
```
model = CNNtoRNN(embed_size, hidden_size, vocab_size, num_layers).to(device)
criterion = nn.CrossEntropyLoss(ignore_index=dataset.vocab.stoi[""])
optimizer = optim.Adam(model.parameters(), lr=3e-4)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=2, gamma=0.1, verbose=True)
```

- Training Result



```
Epochs [1/5] ------- Loss [3.2799]
Epochs [2/5] ------- Loss [2.9793]
Epochs [3/5] ------- Loss [2.2704]
Epochs [4/5] ------- Loss [2.2424]
Epochs [5/5] ------- Loss [2.5368]
```

- 這邊**光是一個 Epoch 就要跑 2.5 小時**，所以決定只開 5 個 Epochs



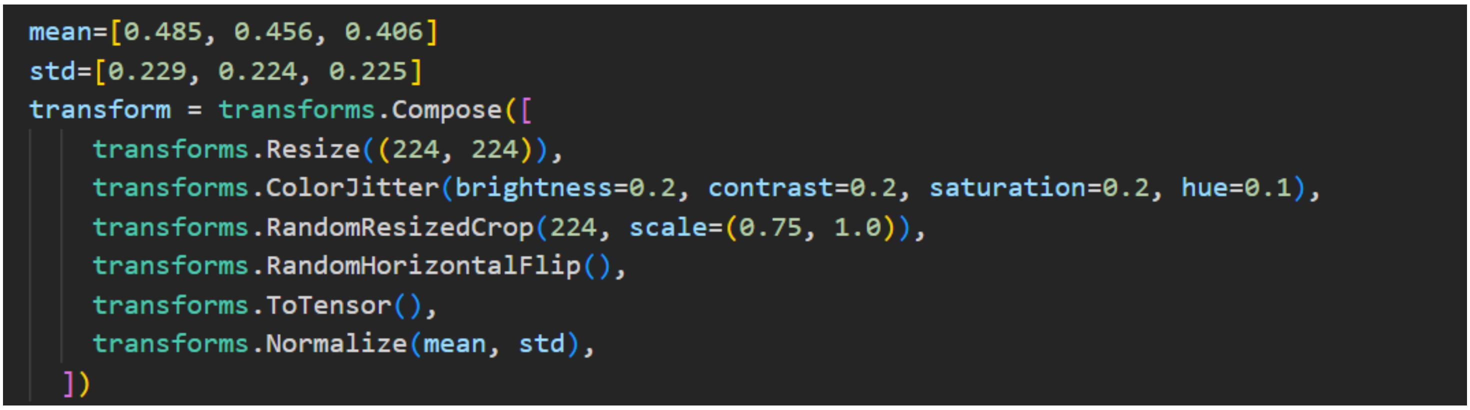
```
Example 1 CORRECT: Dog on a beach by the ocean
Example 1 OUTPUT: white dog running on the beach . carrying a stick . its mouth . . is covered in the background . . is wearing a red jacket .
swimming . . is swimming . . is swimming . . is swimming . . is swimming . .
```

```
Example 2 CORRECT: Child holding red frisbee outdoors
Example 2 OUTPUT: child running through a field of flowers . a green and yellow flowers . . is nearby . . " . " " . " " . " " . " " " " " " "
" " " " " " " " "
```

- 在這之前我嘗試了很多參數的設定，怎麼樣都是這種又臭又長且不合理的 output，所以**推測是 Epoch 不夠**，因此我還是乖乖地拿出錢包儲值了 Colab Pro

## Garbage Output_2

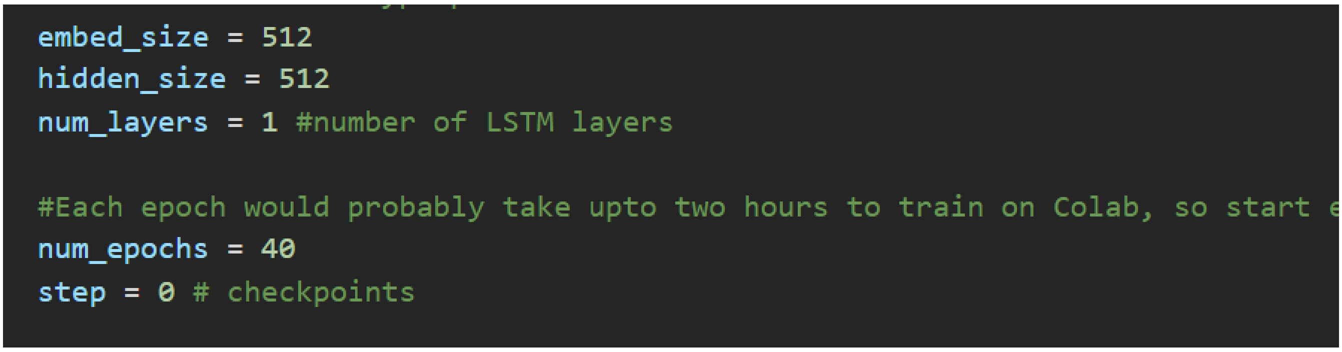
- Data Augmentation
  - 在更之前的作業，我曾經有在 “Data Augmentation” 加入 “**ColorJitter**” 使 Loss 大幅下降過，又我看了網路上其他人的做法，當 Loss 降不下來，參數調來調去都是差不多的結果時也會加入“ColorJitter”，因此我在這次也決定加入參數調整



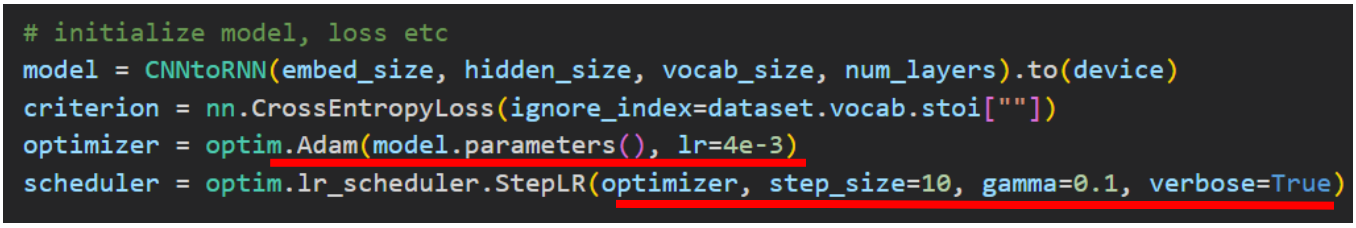
```
mean=[0.485, 0.456, 0.406]
std=[0.229, 0.224, 0.225]
transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
        transforms.RandomResizedCrop(224, scale=(0.75, 1.0)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean, std),
    ])
```

- Encoder Model & Decoder Model
  - Pretrained model 採用跟先前一樣的 **Resnet152**

- Training Settings



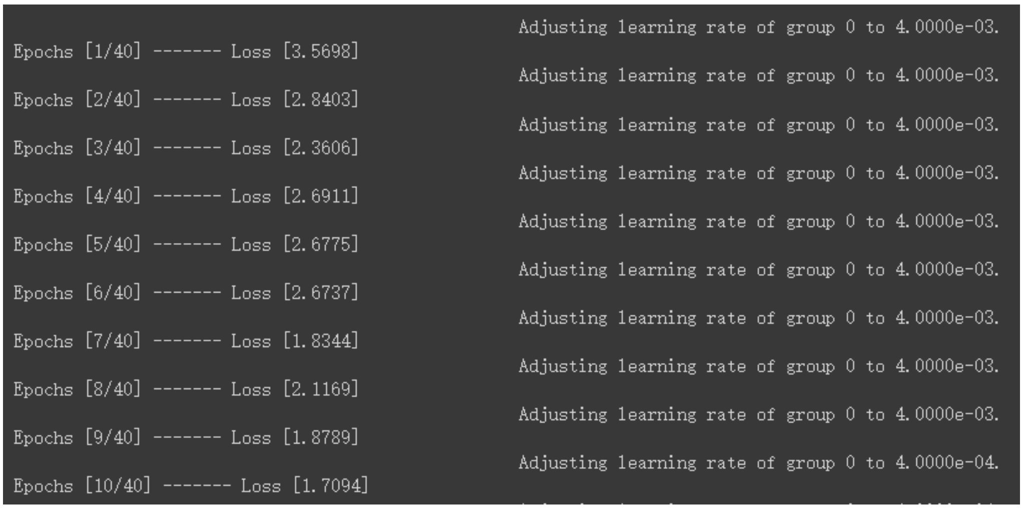
```
embed_size = 512
hidden_size = 512
num_layers = 1 #number of LSTM layers

#Each epoch would probably take upto two hours to train on Colab, so start e
num_epochs = 40
step = 0 # checkpoints
```



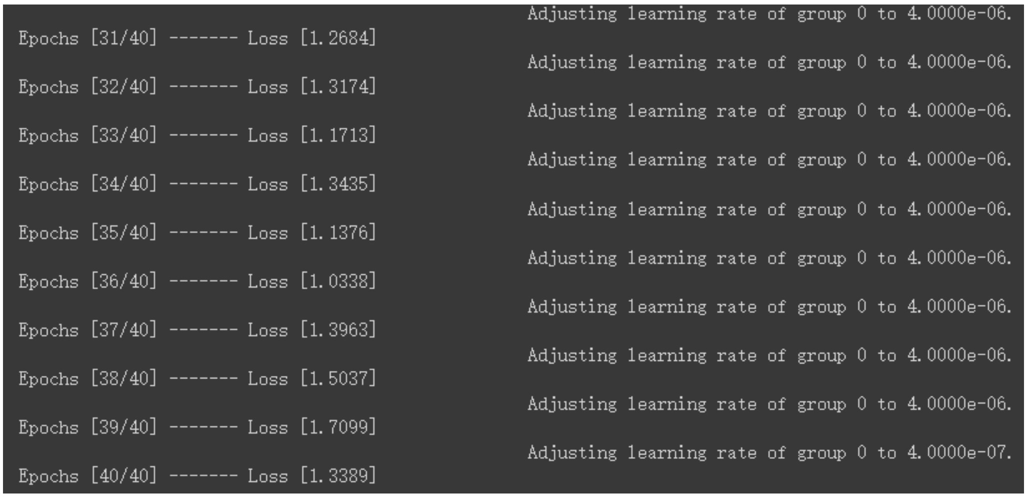
```
# initialize model, loss etc
model = CNNtoRNN(embed_size, hidden_size, vocab_size, num_layers).to(device)
criterion = nn.CrossEntropyLoss(ignore_index=dataset.vocab.stoi[""])
optimizer = optim.Adam(model.parameters(), lr=4e-3)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1, verbose=True)
```

  - 因為前面推測是 Epoch 數不夠，所以我在 train 時設到 **60 epochs**，並加入 scheduler 使 learning rate 每 10 step 就下降 10 倍，但在我 train 到第 10 小時(第 50 epoch)的時後它自行中斷了☹，我看了一下 Loss 有出現 0.9 開頭了，但是實在是跑太久又怕失敗，所以我再次 train 就設 **40 epochs**，總共耗費了 7 小時

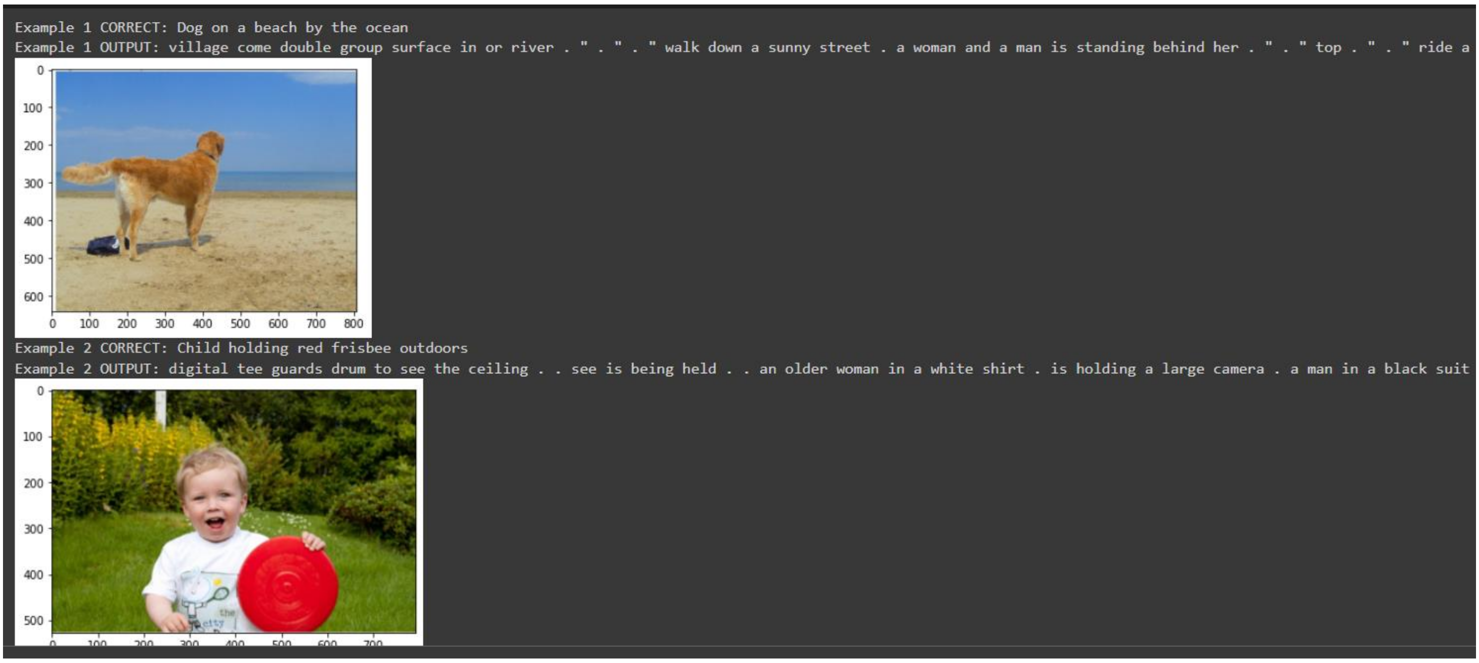
- Training Result
  - 經過了漫長的等待，看到 Loss 下降還興奮了一下，但看起來還是沒 train 成功☹ (這邊只截圖前後各 10 epochs 的 Loss)



```
                                              Adjusting learning rate of group 0 to 4.0000e-03.
Epochs [1/40] ------- Loss [3.5698]
                                              Adjusting learning rate of group 0 to 4.0000e-03.
Epochs [2/40] ------- Loss [2.8403]
                                              Adjusting learning rate of group 0 to 4.0000e-03.
Epochs [3/40] ------- Loss [2.3606]
                                              Adjusting learning rate of group 0 to 4.0000e-03.
Epochs [4/40] ------- Loss [2.6911]
                                              Adjusting learning rate of group 0 to 4.0000e-03.
Epochs [5/40] ------- Loss [2.6775]
                                              Adjusting learning rate of group 0 to 4.0000e-03.
Epochs [6/40] ------- Loss [2.6737]
                                              Adjusting learning rate of group 0 to 4.0000e-03.
Epochs [7/40] ------- Loss [1.8344]
                                              Adjusting learning rate of group 0 to 4.0000e-03.
Epochs [8/40] ------- Loss [2.1169]
                                              Adjusting learning rate of group 0 to 4.0000e-03.
Epochs [9/40] ------- Loss [1.8789]
                                              Adjusting learning rate of group 0 to 4.0000e-04.
Epochs [10/40] ------- Loss [1.7094]
```



```
                                              Adjusting learning rate of group 0 to 4.0000e-06.
Epochs [31/40] ------- Loss [1.2684]
                                              Adjusting learning rate of group 0 to 4.0000e-06.
Epochs [32/40] ------- Loss [1.3174]
                                              Adjusting learning rate of group 0 to 4.0000e-06.
Epochs [33/40] ------- Loss [1.1713]
                                              Adjusting learning rate of group 0 to 4.0000e-06.
Epochs [34/40] ------- Loss [1.3435]
                                              Adjusting learning rate of group 0 to 4.0000e-06.
Epochs [35/40] ------- Loss [1.1376]
                                              Adjusting learning rate of group 0 to 4.0000e-06.
Epochs [36/40] ------- Loss [1.0338]
                                              Adjusting learning rate of group 0 to 4.0000e-06.
Epochs [37/40] ------- Loss [1.3963]
                                              Adjusting learning rate of group 0 to 4.0000e-06.
Epochs [38/40] ------- Loss [1.5037]
                                              Adjusting learning rate of group 0 to 4.0000e-06.
Epochs [39/40] ------- Loss [1.7099]
                                              Adjusting learning rate of group 0 to 4.0000e-07.
Epochs [40/40] ------- Loss [1.3389]
```

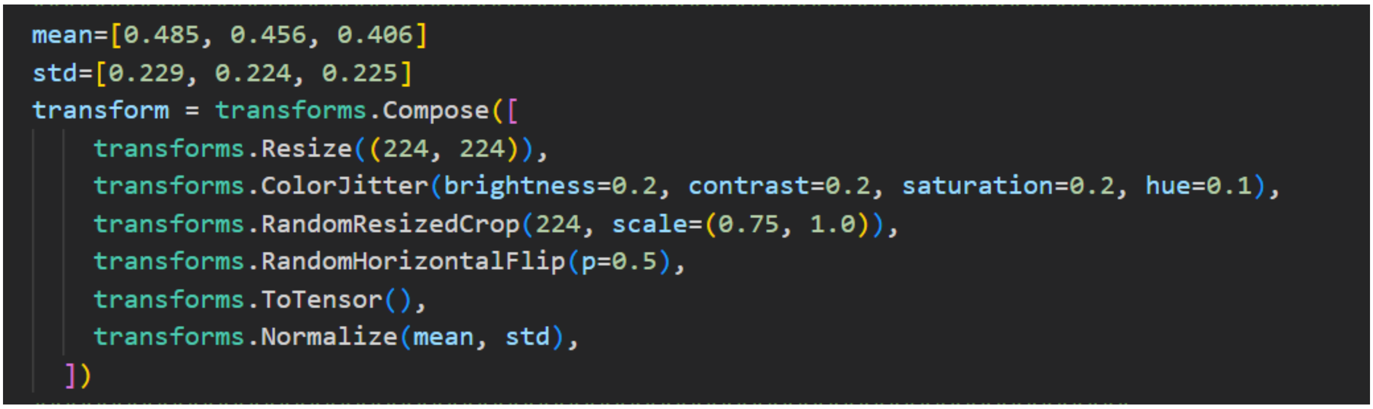


```
Example 1 CORRECT: Dog on a beach by the ocean
Example 1 OUTPUT: village come double group surface in or river . " . " . " walk down a sunny street . a woman and a man is standing behind her . " . " top . " . " ride a
```

```
Example 2 CORRECT: Child holding red frisbee outdoors
Example 2 OUTPUT: digital tee guards drum to see the ceiling . . see is being held . . an older woman in a white shirt . is holding a large camera . a man in a black suit
```

## Excellent Output

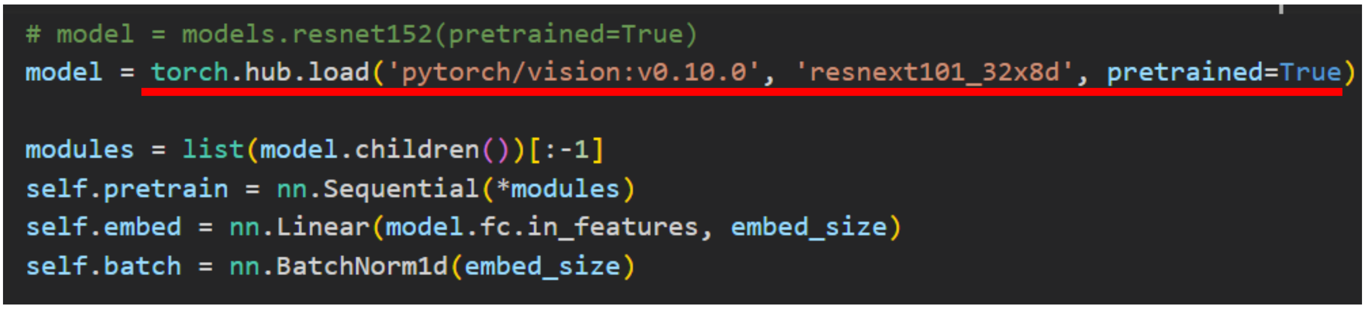
- Data Augmentation



```
mean=[0.485, 0.456, 0.406]
std=[0.229, 0.224, 0.225]
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.RandomResizedCrop(224, scale=(0.75, 1.0)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize(mean, std),
  ])
```

- 跟第二個版本一樣，有加入 **ColorJitter** 來增強

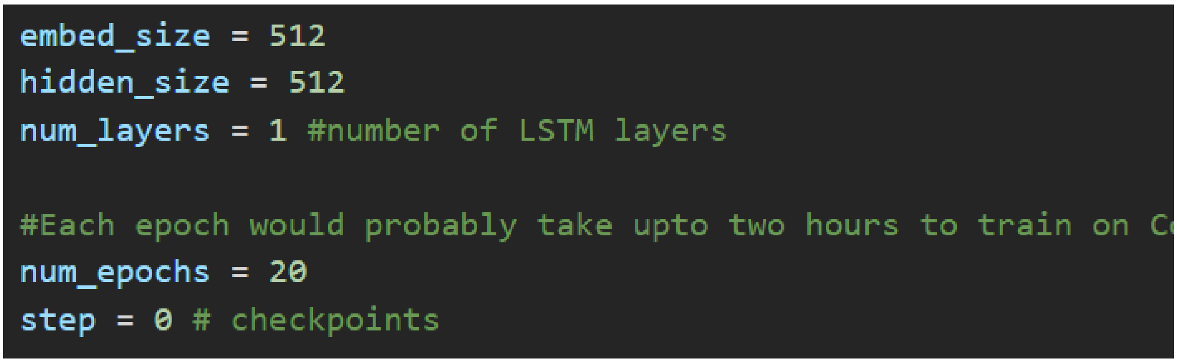
- Encoder Model



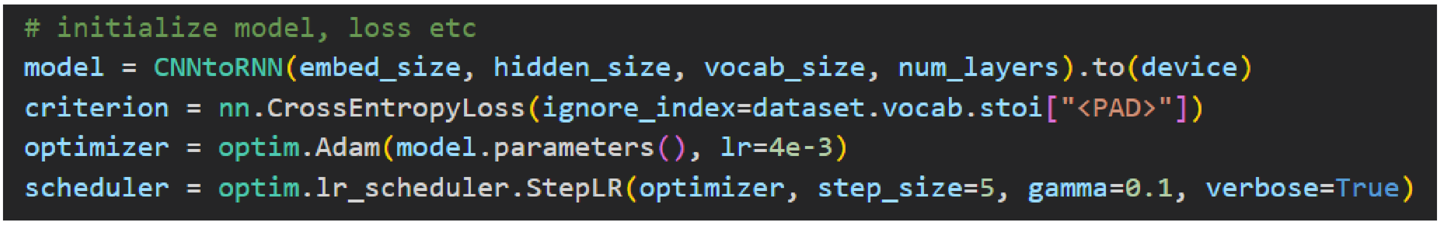
```
# model = models.resnet152(pretrained=True)
model = torch.hub.load('pytorch/vision:v0.10.0', 'resnext101_32x8d', pretrained=True)

modules = list(model.children())[:-1]
self.pretrain = nn.Sequential(*modules)
self.embed = nn.Linear(model.fc.in_features, embed_size)
self.batch = nn.BatchNorm1d(embed_size)
```

- 因為先前 train 了多次的 Resnet152，效果並沒有見好，所以決定改使用 **Resnext101_32x8d**

- Training Settings



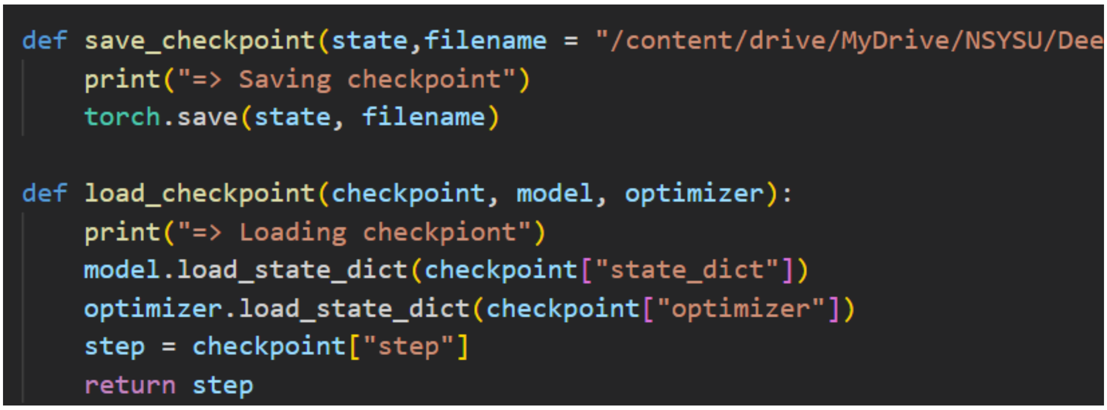
```
embed_size = 512
hidden_size = 512
num_layers = 1 #number of LSTM layers

#Each epoch would probably take upto two hours to train on C
num_epochs = 20
step = 0 # checkpoints
```



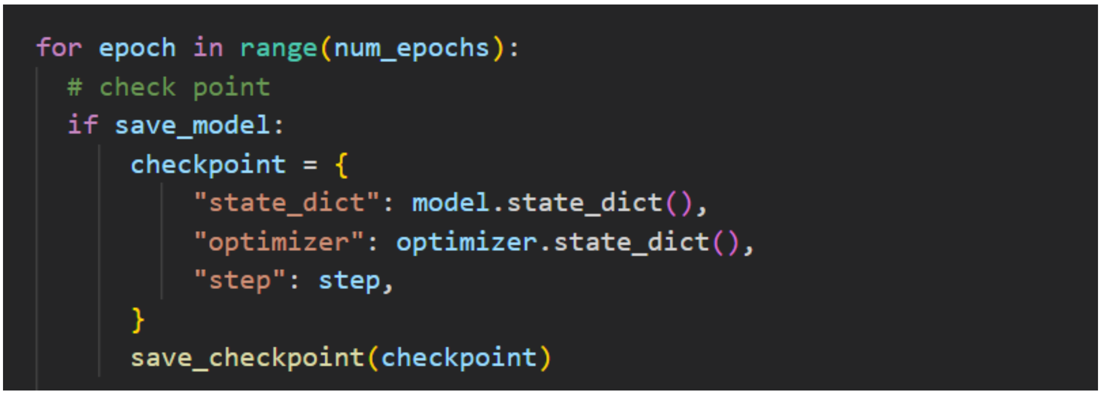
```
# initialize model, loss etc
model = CNNtoRNN(embed_size, hidden_size, vocab_size, num_layers).to(device)
criterion = nn.CrossEntropyLoss(ignore_index=dataset.vocab.stoi["<PAD>"])
optimizer = optim.Adam(model.parameters(), lr=4e-3)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1, verbose=True)
```

- 有了上一個版本的 loss 結果，這次就只 train **20 epochs**，並且將 **scheduler 的 step_size 調為 5**

- 比較特別的是因為怕模型在 train 的過程中又突然中斷，我有使用 **checkpoints** 來記錄每一個 epoch 的 loss，以利後面中斷可以接續著 train，就不用一直盯著模型看

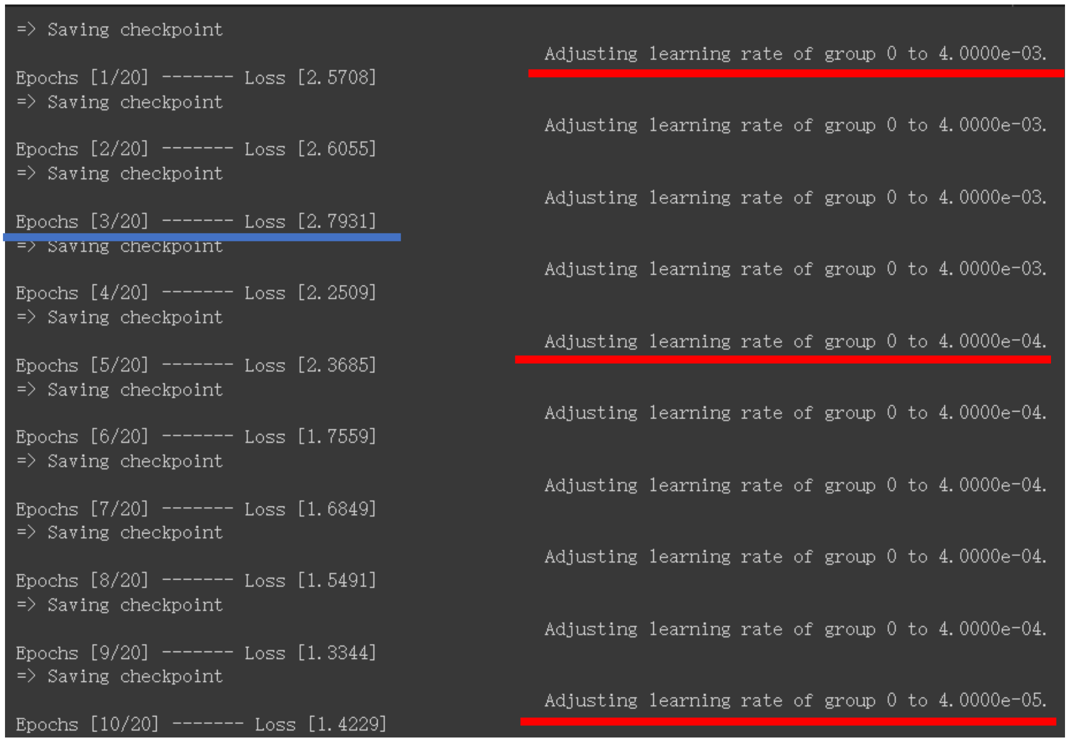


```python
def save_checkpoint(state,filename = "/content/drive/MyDrive/NSYSU/Dee
    print("=> Saving checkpoint")
    torch.save(state, filename)

def load_checkpoint(checkpoint, model, optimizer):
    print("=> Loading checkpiont")
    model.load_state_dict(checkpoint["state_dict"])
    optimizer.load_state_dict(checkpoint["optimizer"])
    step = checkpoint["step"]
    return step
```

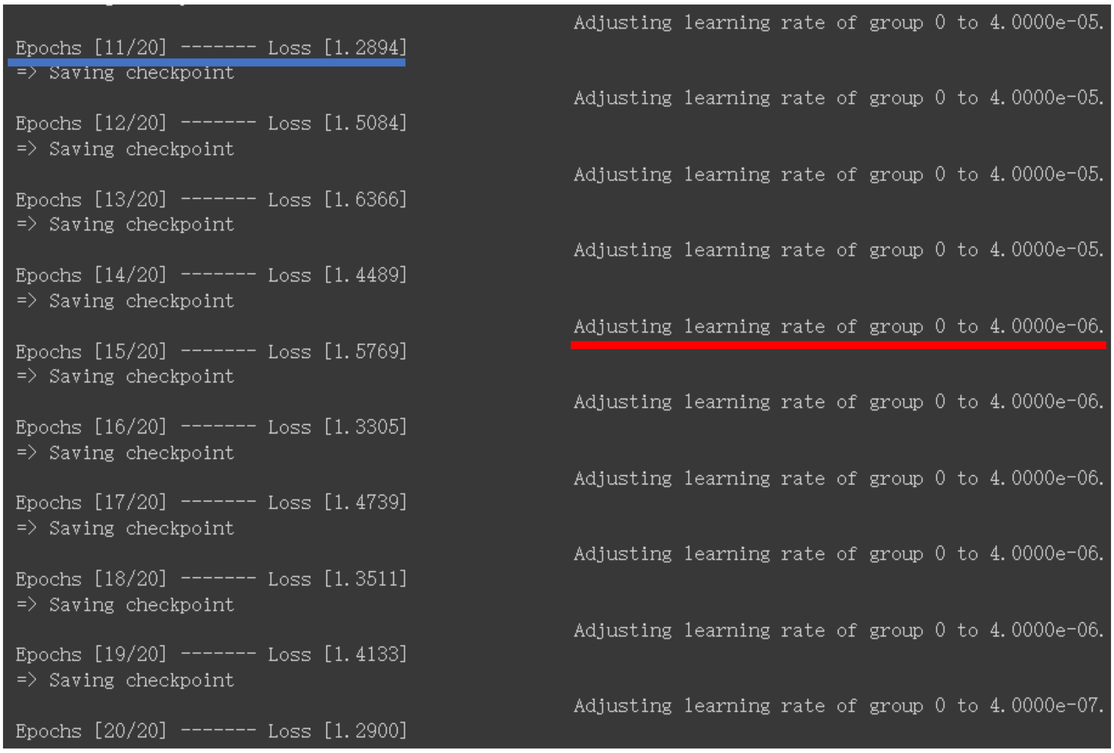


```python
for epoch in range(num_epochs):
    # check point
    if save_model:
        checkpoint = {
            "state_dict": model.state_dict(),
            "optimizer": optimizer.state_dict(),
            "step": step,
        }
        save_checkpoint(checkpoint)
```

- Training Result
  - 這次因為 epoch 數量減半，模型也有改變，總共耗費了 3 個多小時

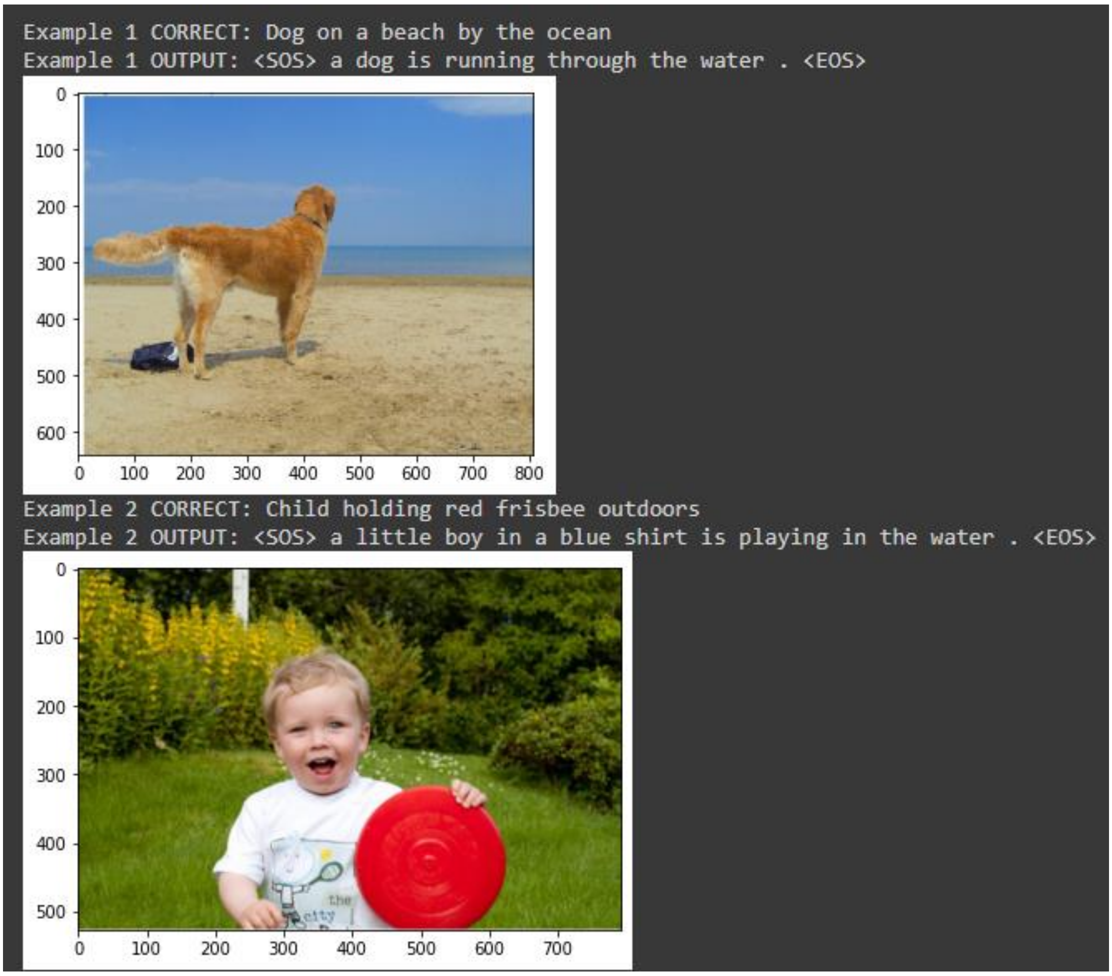


```
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-03.
Epochs [1/20] ------- Loss [2.5708]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-03.
Epochs [2/20] ------- Loss [2.6055]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-03.
Epochs [3/20] ------- Loss [2.7931]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-03.
Epochs [4/20] ------- Loss [2.2509]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-04.
Epochs [5/20] ------- Loss [2.3685]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-04.
Epochs [6/20] ------- Loss [1.7559]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-04.
Epochs [7/20] ------- Loss [1.6849]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-04.
Epochs [8/20] ------- Loss [1.5491]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-04.
Epochs [9/20] ------- Loss [1.3344]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-05.
Epochs [10/20] ------- Loss [1.4229]
```



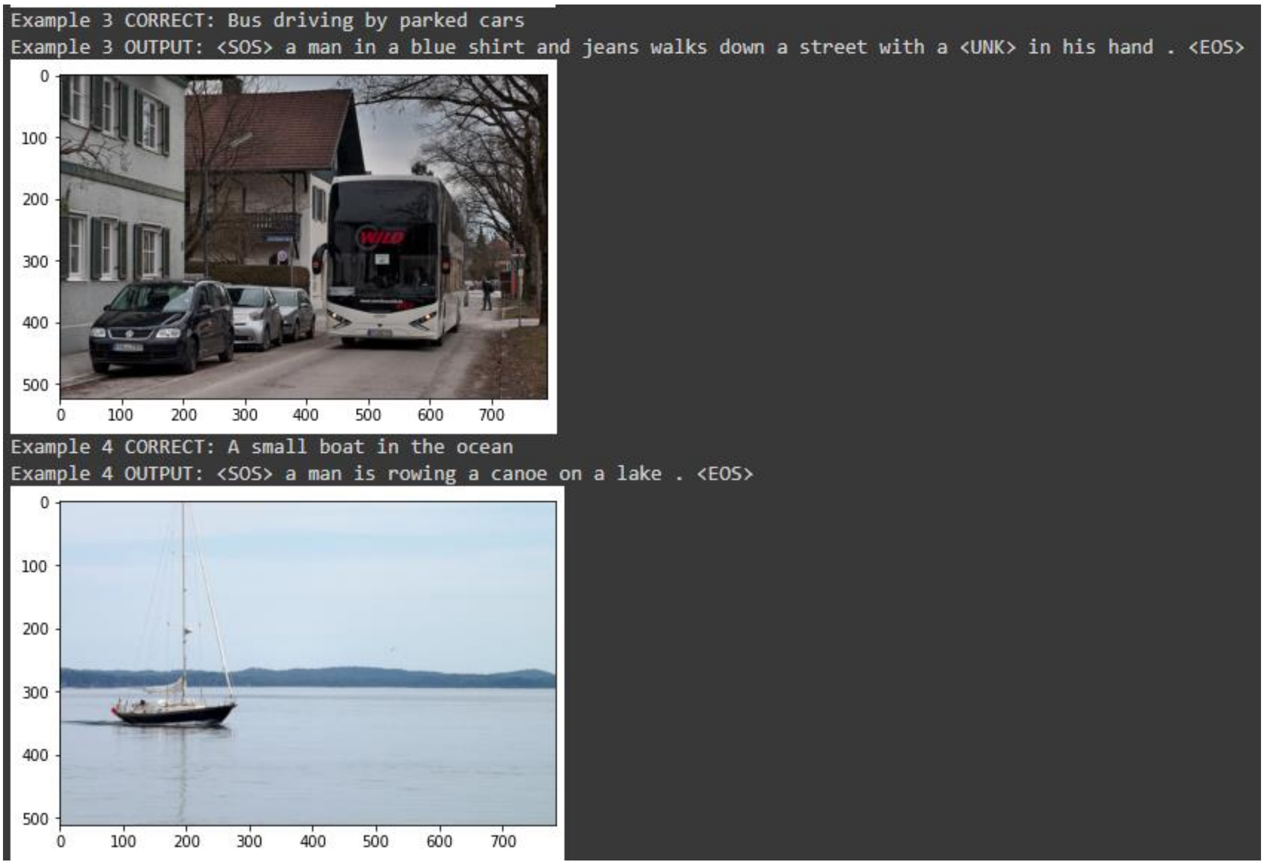
```
                                              Adjusting learning rate of group 0 to 4.0000e-05.
Epochs [11/20] ------- Loss [1.2894]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-05.
Epochs [12/20] ------- Loss [1.5084]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-05.
Epochs [13/20] ------- Loss [1.6366]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-05.
Epochs [14/20] ------- Loss [1.4489]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-06.
Epochs [15/20] ------- Loss [1.5769]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-06.
Epochs [16/20] ------- Loss [1.3305]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-06.
Epochs [17/20] ------- Loss [1.4739]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-06.
Epochs [18/20] ------- Loss [1.3511]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-06.
Epochs [19/20] ------- Loss [1.4133]
=> Saving checkpoint
                                              Adjusting learning rate of group 0 to 4.0000e-07.
Epochs [20/20] ------- Loss [1.2900]
```

- Loss 最高在第 3 epoch：2.7931，Loss 最低在第 11 epoch：1.2894



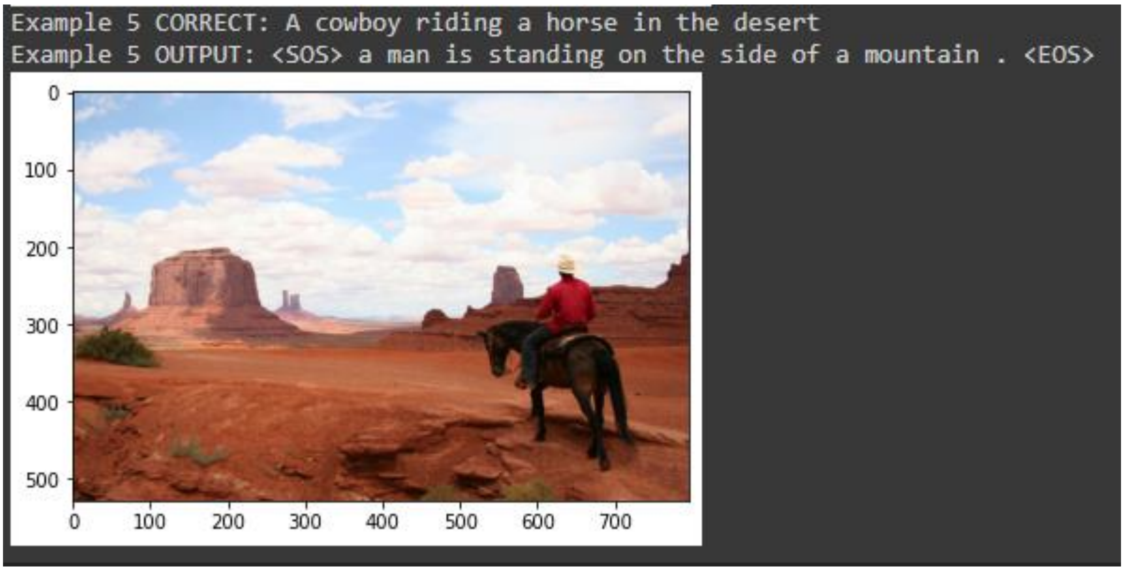
```
Example 1 CORRECT: Dog on a beach by the ocean
Example 1 OUTPUT: <SOS> a dog is running through the water . <EOS>
```

```
Example 2 CORRECT: Child holding red frisbee outdoors
Example 2 OUTPUT: <SOS> a little boy in a blue shirt is playing in the water . <EOS>
```



```
Example 3 CORRECT: Bus driving by parked cars
Example 3 OUTPUT: <SOS> a man in a blue shirt and jeans walks down a street with a <UNK> in his hand . <EOS>
```

```
Example 4 CORRECT: A small boat in the ocean
Example 4 OUTPUT: <SOS> a man is rowing a canoe on a lake . <EOS>
```

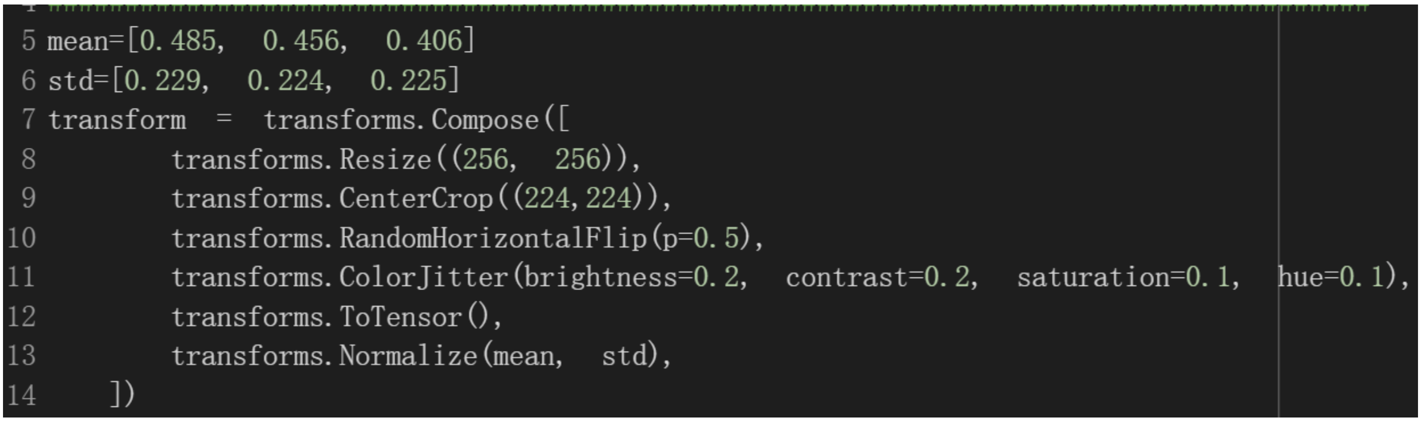


```
Example 5 CORRECT: A cowboy riding a horse in the desert
Example 5 OUTPUT: <SOS> a man is standing on the side of a mountain . <EOS>
```

- 看起來是有 train 起來啦😊，終於~

Assignment 6 Part 2: Image Captioning with Attention

- Data Augmentation

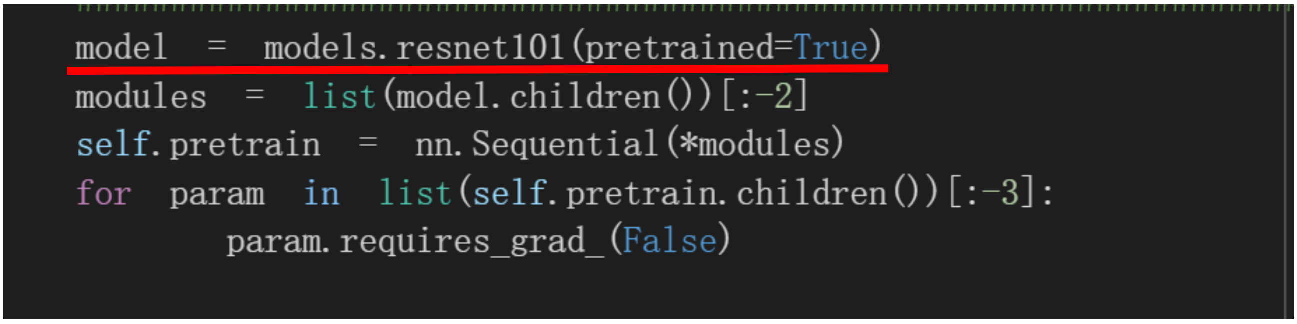


```
mean=[0.485, 0.456, 0.406]
std=[0.229, 0.224, 0.225]
transform = transforms.Compose([
        transforms.Resize((256, 256)),
        transforms.CenterCrop((224,224)),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.1, hue=0.1),
        transforms.ToTensor(),
        transforms.Normalize(mean, std),
    ])
```

- Resize：256
- CenterCrop：224
- 其他調整的功能除了使用 RandomHorizontalFlip 外還有加入 **ColorJitter** 去對圖片進行調整，使 data 更加複雜

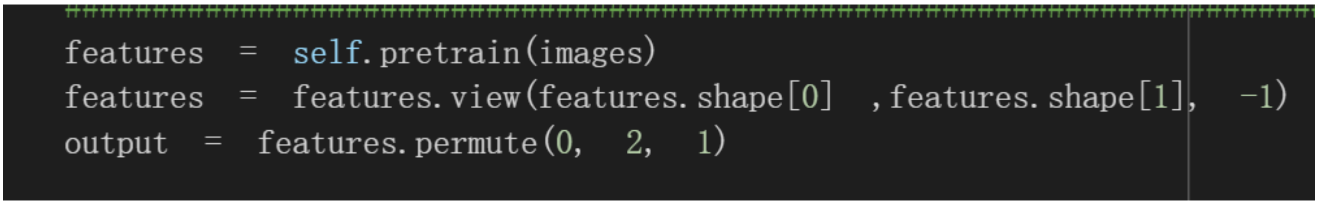
- Encoder Model

__init_()



```
model = models.resnet101(pretrained=True)
modules = list(model.children())[:-2]
self.pretrain = nn.Sequential(*modules)
for param in list(self.pretrain.children())[:-3]:
        param.requires_grad_(False)
```

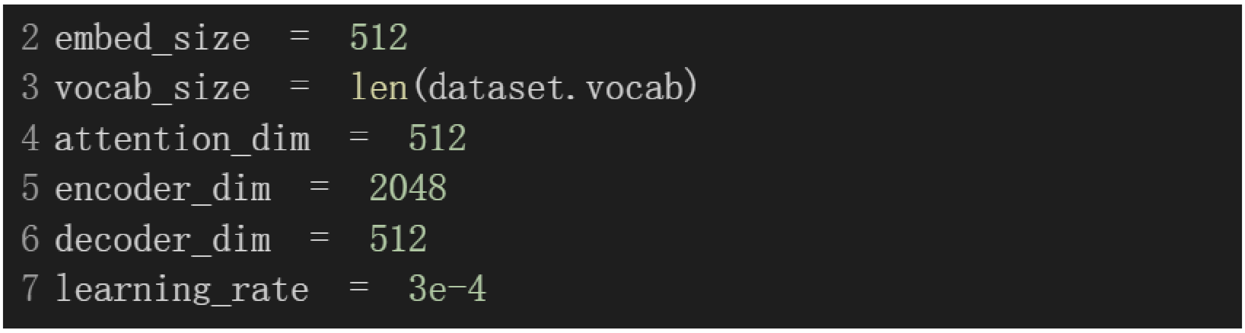
forward()



```
features = self.pretrain(images)
features = features.view(features.shape[0] ,features.shape[1], -1)
output = features.permute(0, 2, 1)
```

- 這次使用的 pretrained model 是 **Resnet101**，選擇用這個 model 是因為我在網路上查相關資料時，大多都是看到使用 Resnet，又使用 Resnet101 的範本最多，因此我也選擇這個 model

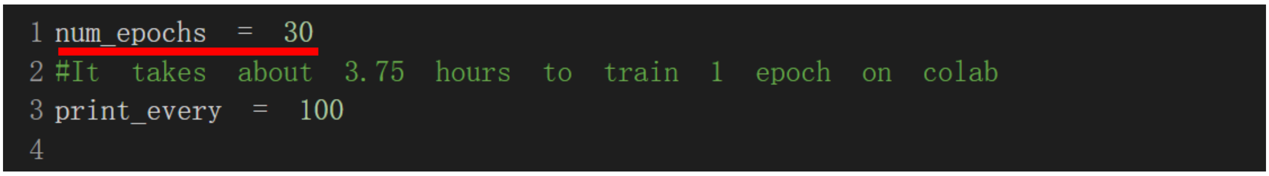
- Training Hyperparams Setting



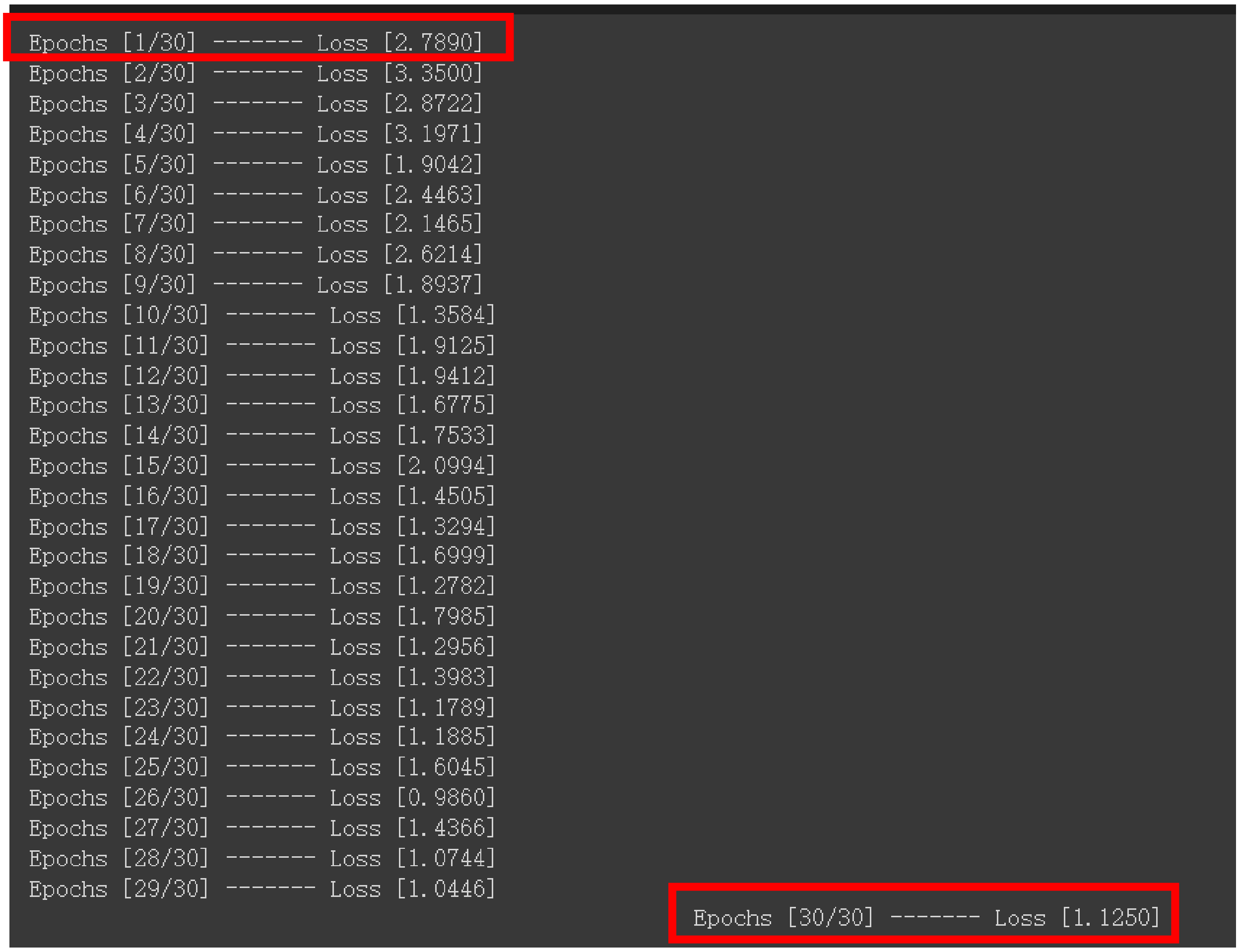
```
embed_size = 512
vocab_size = len(dataset.vocab)
attention_dim = 512
encoder_dim = 2048
decoder_dim = 512
learning_rate = 3e-4
```

  - 以上參數也是我參考網路上作法再自己邊 train 邊調的結果

- Training



```
num_epochs = 30
#It takes about 3.75 hours to train 1 epoch on colab
print_every = 100

```

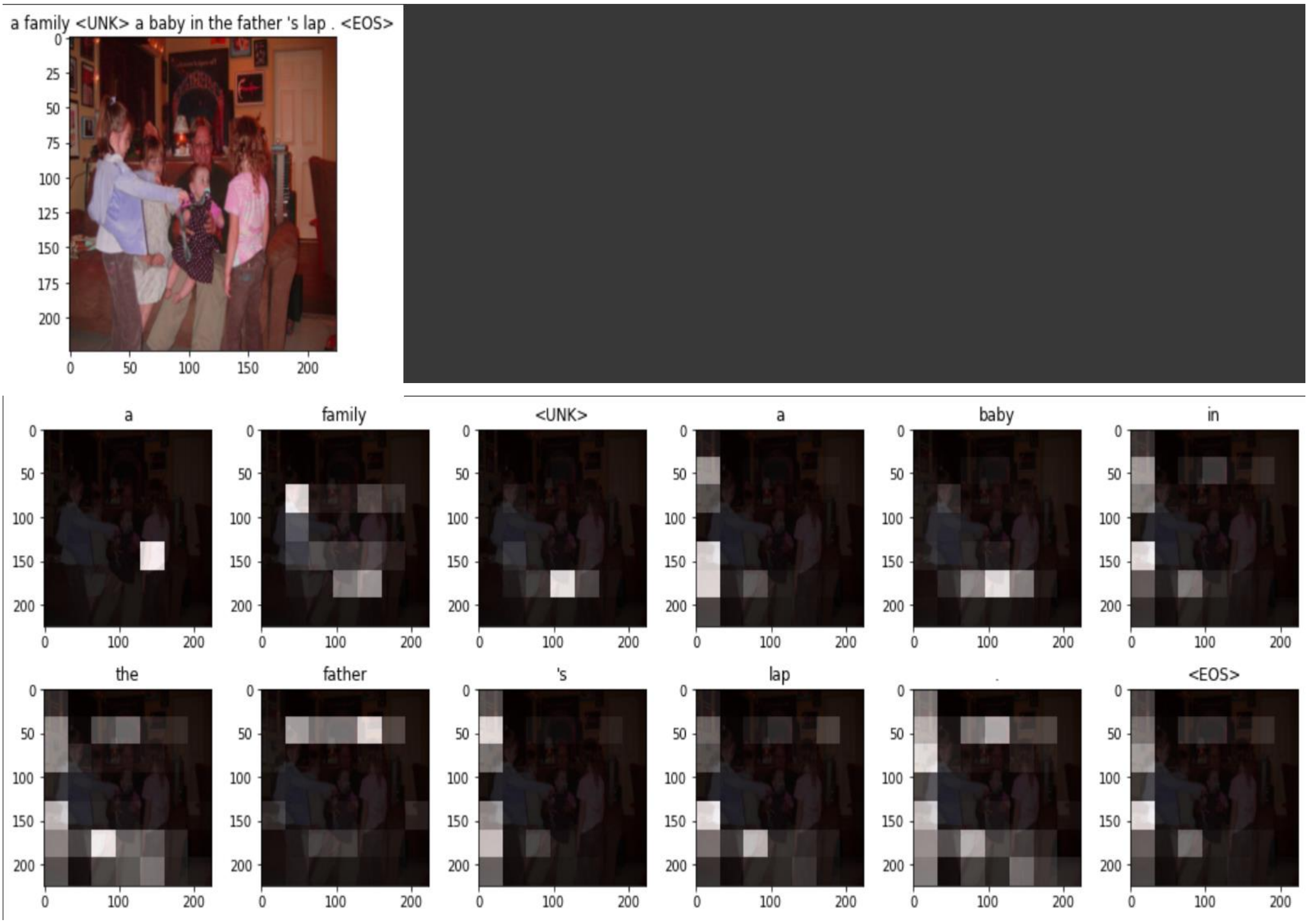


```
Epochs [1/30] ------- Loss [2.7890]
Epochs [2/30] ------- Loss [3.3500]
Epochs [3/30] ------- Loss [2.8722]
Epochs [4/30] ------- Loss [3.1971]
Epochs [5/30] ------- Loss [1.9042]
Epochs [6/30] ------- Loss [2.4463]
Epochs [7/30] ------- Loss [2.1465]
Epochs [8/30] ------- Loss [2.6214]
Epochs [9/30] ------- Loss [1.8937]
Epochs [10/30] ------- Loss [1.3584]
Epochs [11/30] ------- Loss [1.9125]
Epochs [12/30] ------- Loss [1.9412]
Epochs [13/30] ------- Loss [1.6775]
Epochs [14/30] ------- Loss [1.7533]
Epochs [15/30] ------- Loss [2.0994]
Epochs [16/30] ------- Loss [1.4505]
Epochs [17/30] ------- Loss [1.3294]
Epochs [18/30] ------- Loss [1.6999]
Epochs [19/30] ------- Loss [1.2782]
Epochs [20/30] ------- Loss [1.7985]
Epochs [21/30] ------- Loss [1.2956]
Epochs [22/30] ------- Loss [1.3983]
Epochs [23/30] ------- Loss [1.1789]
Epochs [24/30] ------- Loss [1.1885]
Epochs [25/30] ------- Loss [1.6045]
Epochs [26/30] ------- Loss [0.9860]
Epochs [27/30] ------- Loss [1.4366]
Epochs [28/30] ------- Loss [1.0744]
Epochs [29/30] ------- Loss [1.0446]
                                                  Epochs [30/30] ------- Loss [1.1250]
```

  - 起初一個 epoch 都要跑**將近 3 小時!!** 而升級了 Colab Pro 後一 epoch 只需要 13 分鐘左右

- 一開始我都是開 5、10 epochs 在跑，但 train 出來的結果很差，所以我看了一下所剩的 colab pro 單元，估算後決定開到 30 epoch
- **30 epochs** 總共花了 6 個多小時在 train，所幸 **Loss 有從 2.7 降到 1.0**
  (Loss 最高在第 2 Epoch：3.3500，Loss 最低在第 26 Epoch：0.9860)

- Test Result



- 我個人覺得最後是有 train 起來的，如果我還有更多單元可用，我想我會再往上增加 epoch 的數量，來嘗試讓 loss 降到 1.0 以下