NVLab Summer School 2023 HW3

112061517 王浩

Abstract

在本次作業裡，我們利用Pytorch 套件，建立AutoEncoder(AE), Variational AutoEncoder(VAE)，利用1476 張眼睛照片，訓練模型。此外，我們還以現有的VAE 模型為基礎，新增VAE with Skip Connection (在本篇報告簡稱SkVAE)與VAE with Deeper structure(在本篇報告簡稱DpVAE)。並且在latent space 加入Gaussian Noise 來觀察不同模型結果，並討論現象。

1 Implementation

1.1 AutoEncoder

1.1.1 Model Structure

```
# Define Autoencoder architecture
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=2,
            padding=1), # output size is (N, 16, 25, 25)
            nn.ReLU(),
            nn.Conv2d(16, 32, 3, 2, 1), # output size is (N, 32, 13, 13)
            nn.ReLU(),
            nn.Conv2d(32, 64, 3, 2, 1), # output size is (N, 64, 7, 7)
            nn.ReLU(),

            nn.Flatten(),  # Flatten the output of the convolutional layers
        )
        self.decoder = nn.Sequential(
            nn.Unflatten(1, (64, 7, 7)),  # Unflatten to 4D tensor

            nn.ConvTranspose2d(64, 32, 3, 2, 1), # output size is (N, 32, 13, 13)
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, 3, 2, 1), # output size is (N, 16, 25, 25)
            nn.ReLU(),
            nn.ConvTranspose2d(16, 3, 3, 2, 1, output_padding=1),
            # output size is (N, 3, 50, 50)
            nn.Sigmoid()
        )

    def forward(self, x, add_noise=False, noise=0.0):
        encoded = self.encoder(x)
        if add_noise:
            encoded = encoded + noise
        decoded = self.decoder(encoded)
        return decoded
```

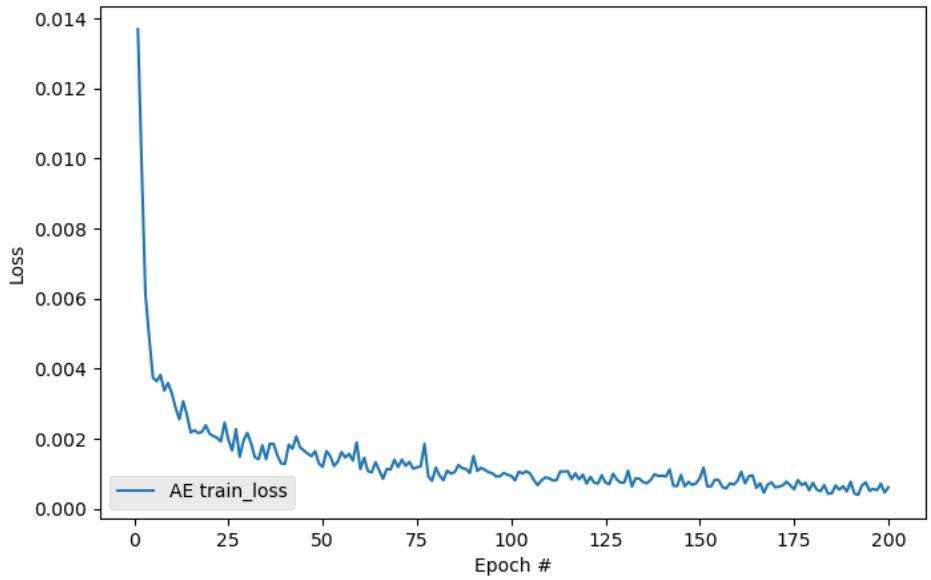


Figure 1: AE MSE loss

1.1.2 Loss and score

- 我們使用Mean square error 去計算loss，loss 隨epoch 詳細變化，可以參考圖1
- 與Ground True 相比之下，我們得到PSNR:33.64, SSIM:0.9569

1.2 VAE

Variational AutoEncoder

1.2.1 Model Structure

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

        self.encoder = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=2,
            padding=1), # output size is (N, 16, 25, 25)
            nn.ReLU(),
            nn.Conv2d(16, 32, 3, 2, 1), # output size is (N, 32, 13, 13)
            nn.ReLU(),
            nn.Conv2d(32, 64, 3, 2, 1), # output size is (N, 64, 7, 7)
            nn.ReLU(),

            nn.Flatten(),  # Flatten the output of the convolutional layers
        )
        self.fc_mu = nn.Linear(64 * 7 * 7, LATENT_DIM)
        self.fc_logvar = nn.Linear(64 * 7 * 7, LATENT_DIM)

        self.decoder = nn.Sequential(
            nn.Unflatten(1, (64, 7, 7)),  # Unflatten to 4D tensor

            nn.ConvTranspose2d(64, 32, 3, 2, 1), # output size is (N, 32, 13, 13)
            nn.ReLU(),
```

$$
\begin{aligned}
-D_{KL}((q_{\boldsymbol{\phi}}(\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{z})) &= \int q_{\boldsymbol{\theta}}(\mathbf{z}) \left(\log p_{\boldsymbol{\theta}}(\mathbf{z}) - \log q_{\boldsymbol{\theta}}(\mathbf{z})\right) d\mathbf{z} \\
&= \frac{1}{2} \sum_{j=1}^{J} \left(1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2\right)
\end{aligned}
$$

Figure 2: KL divergence loss[KW22]

```
            nn.ConvTranspose2d(32, 16, 3, 2, 1), # output size is (N, 16, 25, 25)
            nn.ReLU(),
            nn.ConvTranspose2d(16, 3, 3, 2, 1, output_padding=1),
            # output size is (N, 3, 50, 50)
            nn.Sigmoid()
        )

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def forward(self, x, add_noise=False, noise=0.0):
        encoded = self.encoder(x)
        mu = self.fc_mu(encoded)
        logvar = self.fc_logvar(encoded)
        z = self.reparameterize(mu, logvar)
        if add_noise:
            z = z +noise
        decoded = self.decoder(z)
        return decoded, mu, logvar

# Define loss function
def vae_loss(recons, originals, mu, logvar):
    recons_loss = F.mse_loss(recons, originals, reduction='sum')
    kl_loss = torch.mean(
                    -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp(), dim=1)
                    )
    return recons_loss, kl_loss, recons_loss + kl_loss
```

1.2.2 Loss and score

- 我們使用Mean square error 去計算Construction loss，除了讓經過encoder 與decoder 的圖片與label 相似，我們為了也讓encode 生成的distribution (μ and σ)與Gaussian Distribution 兩者相近，多增加了KL divergence 產生的Loss2，詳細變化參圖3
- 與Ground True 相比之下，我們得到PSNR:25.9416, SSIM:0.7840

1.3 SkVAE

Variational AutoEncoder with Skip Connection

1.3.1 Model Structure

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

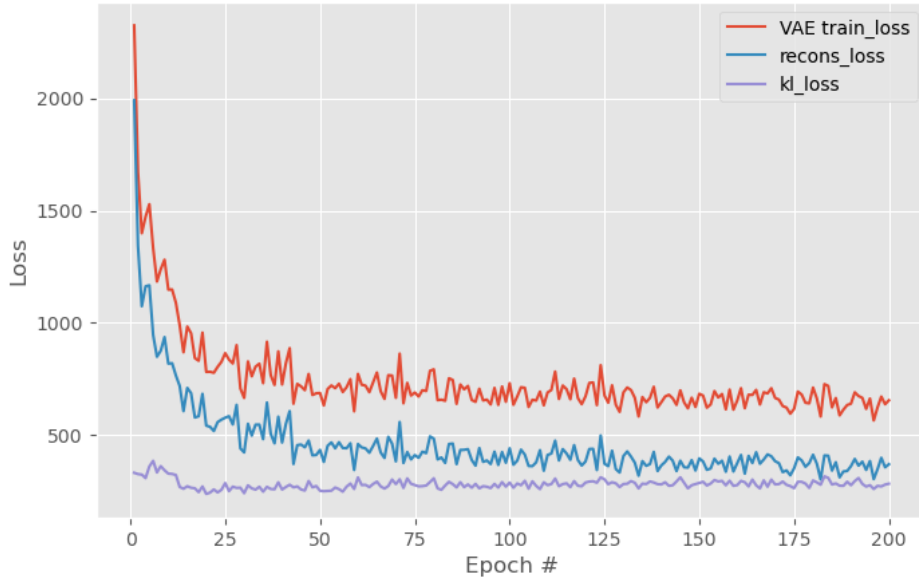


Figure 3: VAE Construction loss(MSE) and KL Divergence loss

```
        self.encoder = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=2,
            padding=1), # output size is (N, 16, 25, 25)
            nn.ReLU(),
            nn.Conv2d(16, 32, 3, 2, 1), # output size is (N, 32, 13, 13)
            nn.ReLU(),
            nn.Conv2d(32, 64, 3, 2, 1), # output size is (N, 64, 7, 7)
            nn.ReLU(),

            nn.Flatten(),  # Flatten the output of the convolutional layers
        )
        self.fc_mu = nn.Linear(64 * 7 * 7, LATENT_DIM)
        self.fc_logvar = nn.Linear(64 * 7 * 7, LATENT_DIM)

        self.decoder = nn.Sequential(
            nn.Unflatten(1, (64, 7, 7)),  # Unflatten to 4D tensor (N, 64, 7, 7)

            nn.ConvTranspose2d(64, 32, 3, 2, 1), # output size is (N, 32, 13, 13)
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, 3, 2, 1), # output size is (N, 16, 25, 25)
            nn.ReLU(),
            nn.ConvTranspose2d(16, 3, 3, 2, 1, output_padding=1),
            # output size is (N, 3, 50, 50)
            nn.Sigmoid()
        )

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def forward(self, x, add_noise=False, noise=0.0):
```

```
        encode_list = []
        for idx, layer in enumerate(self.encoder):
            x = layer(x)
            if idx in [0, 2, 4]:
                encode_list.append(x)

        mu = self.fc_mu(x)
        logvar = self.fc_logvar(x)
        x = self.reparameterize(mu, logvar)

        if add_noise:
            x = x + noise
        for idx, layer in enumerate(self.decoder):
            x = layer(x)
            if idx in [0, 1, 3]:
                x = x + encode_list.pop() # Add with skip connection

        return x, mu, logvar


# Define loss function
def skvae_loss(recons, originals, mu, logvar):
    recons_loss = F.mse_loss(recons, originals, reduction='sum')
    kl_loss = torch.mean(
                    -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp(), dim=1)
                    )
    return recons_loss, kl_loss, recons_loss + kl_loss
```

1.3.2 Loss and score

- 我們一樣使用Mean square error 去計算Construction loss，也因為VAE 講述到的原因，多增加了KL divergence 產生的Loss[KW22]，詳細變化參圖4
- 與Ground True 相比之下，我們得到PSNR:38.2399, SSIM:0.9843

1.4 DpVAE

Deeper Variational AutoEncoder

1.4.1 Model Structure

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

        self.encoder = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=2,
            padding=1), # output size is (N, 16, 25, 25)
            nn.ReLU(),
            nn.Conv2d(16, 32, 3, 2, 1), # output size is (N, 32, 13, 13)
            nn.ReLU(),
            nn.Conv2d(32, 64, 3, 2, 1), # output size is (N, 64, 7, 7)
            nn.ReLU(),
            nn.Conv2d(64, 128, 3, 2, 1), # output size is (N, 128, 4, 4)
            nn.ReLU(),
            nn.Flatten(),  # Flatten the output of the convolutional layers
```

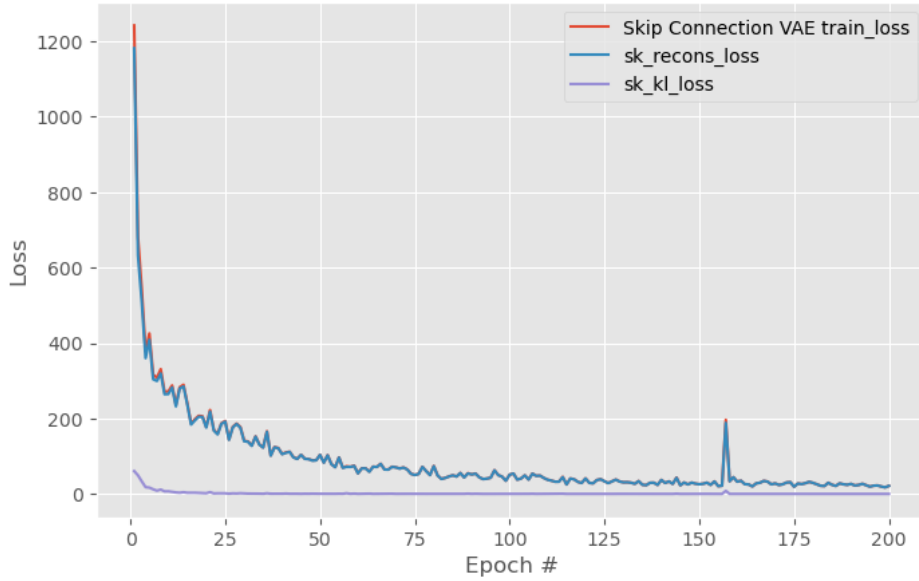


Figure 4: Skip Connection VAE loss

```
        )
        self.fc_mu = nn.Linear(128 * 4 * 4, 128 * 4 * 4)
        self.fc_logvar = nn.Linear(128 * 4 * 4, 128 * 4 * 4)

        self.decoder = nn.Sequential(
            nn.Unflatten(1, (128, 4, 4)),  # Unflatten to 4D tensor (N, 128, 4, 4)
            nn.ConvTranspose2d(128, 64, 3, 2, 1), # output size is (N, 64, 7, 7)
            nn.ReLU(),
            nn.ConvTranspose2d(64, 32, 3, 2, 1), # output size is (N, 32, 13, 13)
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, 3, 2, 1), # output size is (N, 16, 25, 25)
            nn.ReLU(),
            nn.ConvTranspose2d(16, 3, 3, 2, 1, output_padding=1),
            # output size is (N, 3, 50, 50)
            nn.Sigmoid()
        )

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def forward(self, x, add_noise=False, noise=0.0):
        encoded = self.encoder(x)
        mu = self.fc_mu(encoded)
        logvar = self.fc_logvar(encoded)
        z = self.reparameterize(mu, logvar)
        if add_noise:
            z = z +noise
        decoded = self.decoder(z)
        return decoded, mu, logvar

# Define loss function
```

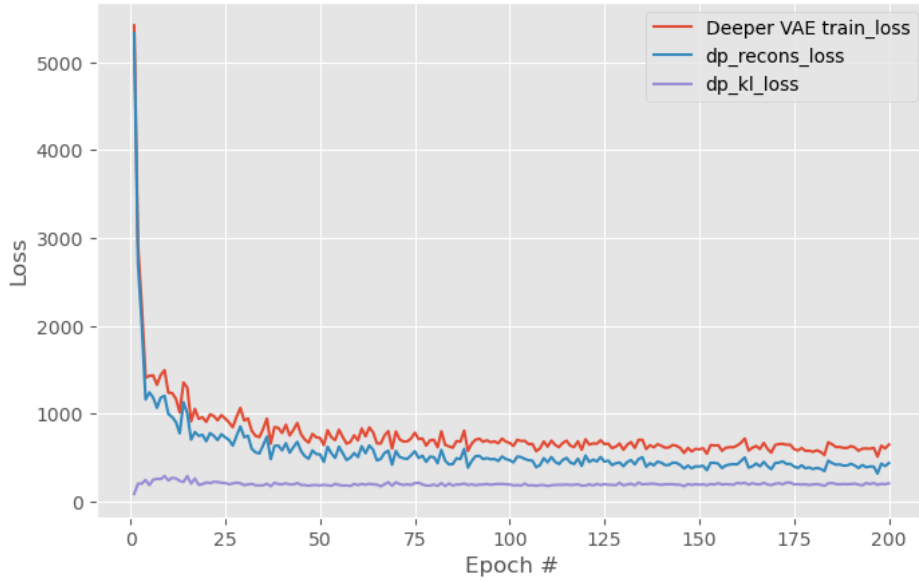


Figure 5: Deeper VAE loss

```
def dpvae_loss(recons, originals, mu, logvar):
    recons_loss = F.mse_loss(recons, originals, reduction='sum')
    kl_loss = torch.mean(
                    -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp(), dim=1)
                    )
    return recons_loss, kl_loss, recons_loss + kl_loss
```

1.4.2 Loss and Score

- 我們一樣使用Mean square error 去計算Construction loss，也因為VAE 講述到的原因，多增加了KL divergence 產生的Loss[KW22]，詳細變化參圖5
- 與Ground True 相比之下，我們得到PSNR:25.4582, SSIM:0.7588

2 Discussion

2.1 Model Architecture

在本次的AE與VAE模型設計中，我都是使用CNN based 的設計，在比較少量的情況下，利用CNN 良好的inductive bias 來提取出影像的特徵，方便訓練獲得好的結果。從圖6 看出，input 進來的image size 為$3*50*50$ 接著經過三層的convolution，feature size 經過encoder 從$16*25*25$、$32*13*13$ 到$64*7*7$，再將$64*7*7$ flatten 之後，就代表latent space 的vector，而decoder 就是使用inverse 的convolution，把encoded result 反向convolution 來得到預測的結果。

AE 和VAE 的不同，則可以從圖7和8可以看出，VAE 的output 雖然也是$64*7*7$ flatten，但不是如AE 一樣，直接接上decoder，而是接上fc_mu 與fc_logvar 來取得distribution 的性質，再從此distribution 中，隨機sample 出一個vector 來feed 入decoder 當中，以獲取更「泛化」的圖像。

而之所以要預測log variance，而不直接視output 為standard deviation，是因為log 的結果不論是正是負，在取完exponential 之後，我們得到的variance 也就不會有負的情況出現，模型不用再多做調整。此外再做output distribution 與Gaussian distribution 的Kullback-Leibler (KL) divergence 時，我們也可以用log 比較好運算的性質，來降低運算的複雜度。

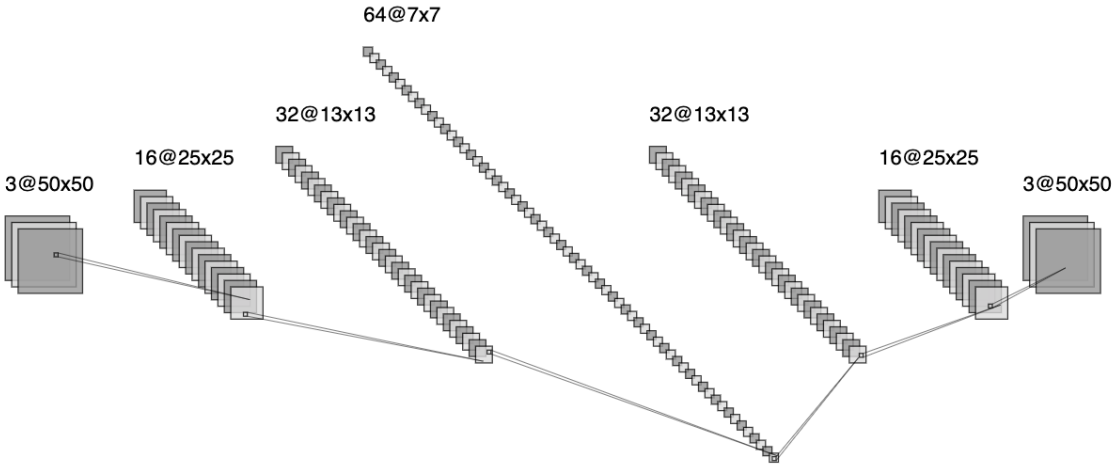


Figure 6: CNN based structure for AE and VAE

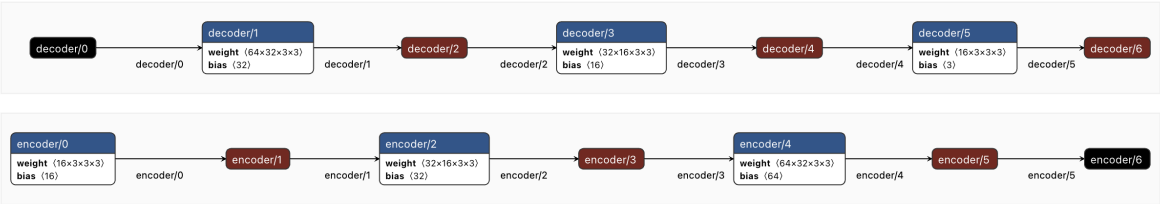


Figure 7: AE model structure

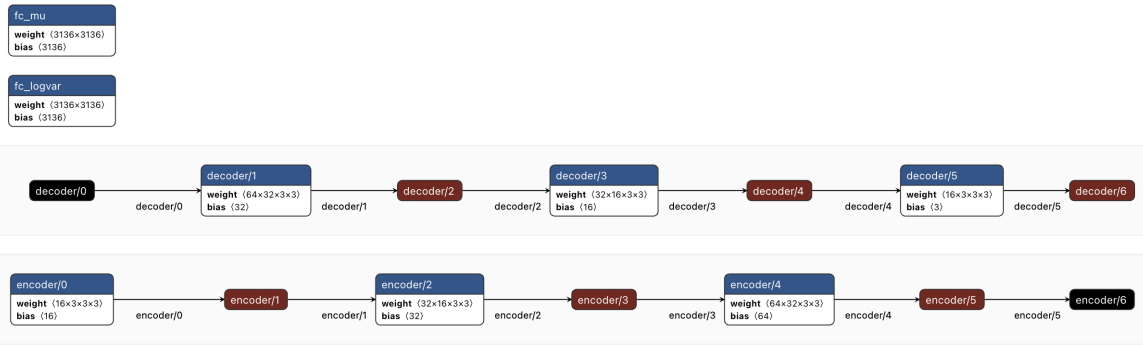


Figure 8: VAE model structure

	Value
Batch size	18
Epoch	200
Learning rate	0.001

Table 1: The hyperparameters setting of all models

	PSNR	SSIM
AE	33.6367	0.9569
VAE	25.9416	0.7840
SkVAE	38.2399	0.9843
DpVAE	25.4582	0.7588

Table 2: The scores of all models

2.2 Hyperparameter

我們本次所採用的hyperparameter 可以見表1，而batch 選擇為18，是根據本次的dataset 大小只有1476 張imgae，並不是大量的資料集，因此若要得到較好的model result，模型需要的iteration 的次數不能太低，故我選擇的batch size 為偏小的18，以得到較多次的迭代更新參數。epoch 的選擇，我從model loss plot (1, 3) 觀察，發現大約在50開始，模型的loss 就呈現緩慢下降，150之後下降情況更為緩慢，因此衡量時間與效能的情況下，我選擇epoch 為200。Learning rate 也是經過數次訓練的經驗，取效果不錯的0.001。

2.3 Result

PSNR 與SSIM 的分數計算可以參考表2，而AE與VAE的訓練圖形呈現在圖9，其中的(a)是資料集內的Ground Truth，(b)為AutoEncoder，(c)為VAE 的訓練結果。而(d)Skip Connection VAE 和(d)Deeper VAE 為更改模型，不同的VAE 版本。

另外，根據實驗，我們在latent space 加入Gaussian Noise 之後的結果，呈現圖10，(a)AE、(b)VAE 與(c)SkVAE 分別都加入了相同的Gaussian Noise。

2.4 Comparison

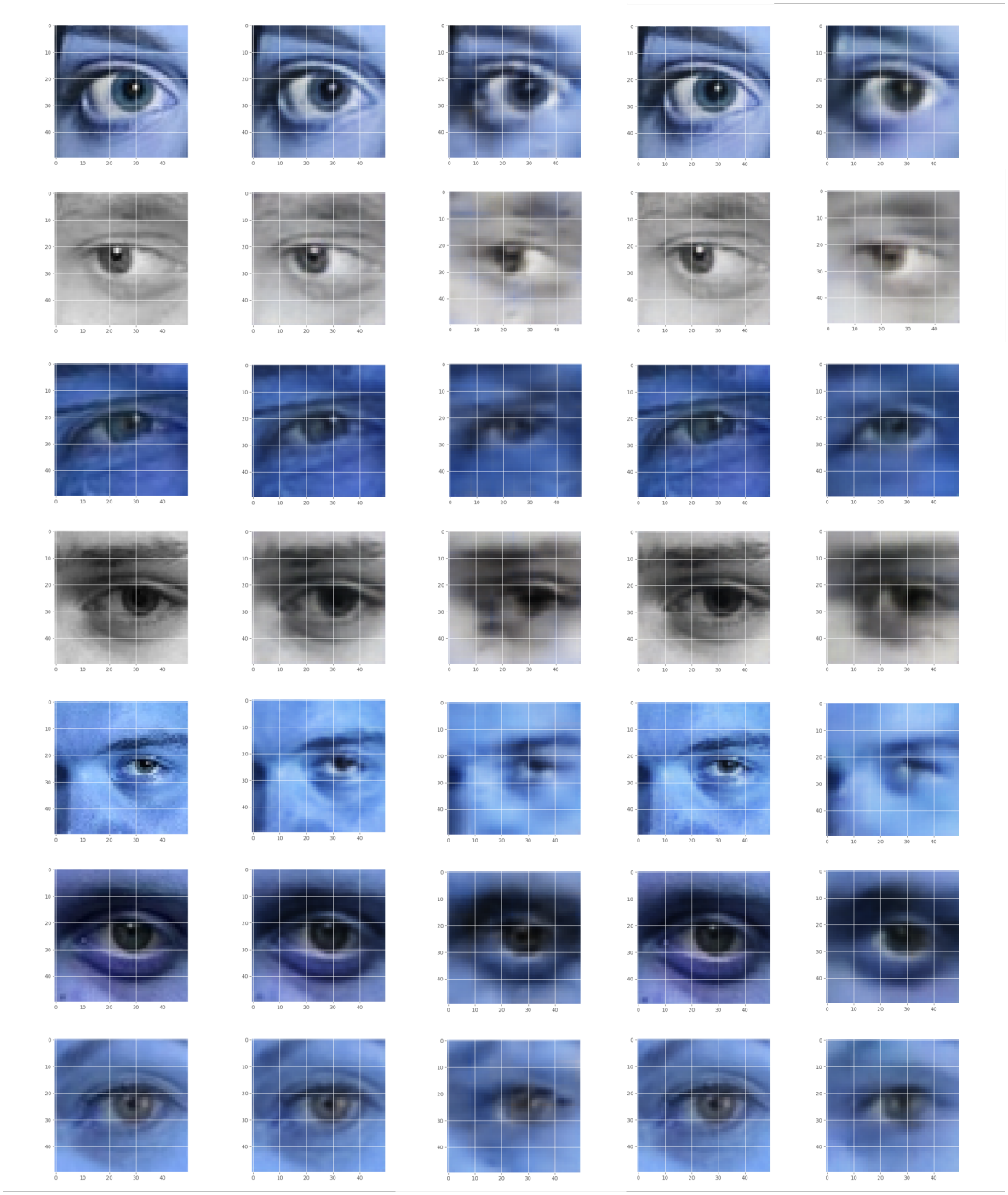
AE 的模型結構，會促使一張feed in 的圖片，以一個固定dimension 的vector 來呈現。VAE 則不同，VAE 是被設計成去學習一個latent space 去對應一張image，為了呈現一些有意義的性質，所以latent space 會是一種probability distribution，latent space sample 出來的latent variable 都希望能夠map 到有意義的圖片、正確的output。所以如果我們以經過encoder並從latent space sample 出來的vector ，再加上類似分佈的Gaussian distribution sample 出來的相同dimension 的vector(noise)，那原則上對於output image 也不會有太大的影響。

從上述可知VAE的模型相較於訓練一個vector，一個normal distribution 會比較難訓練，所以在有限的資料量下，對於Ground Truth 的訓練效果會比較差，可以在表2的分數得知，AE不管在PSNR 或SSIM 上都贏過VAE。在圖9中也可以觀察到，(b)AE 的圖像都可以比(c)VAE 還更清楚。

但在圖10，也可以看出(b)VAE 在加入相同的Gaussian Noise後，都比(a)AE 更有資料「眼睛」的輪廓，而AE的資料更像是雜訊，也可以應證VAE 訓練的distribution 面對Noise 更有泛化的能力，也凸顯出image 建立在一個space 上的重要性，而不是如AE 專注訓練在單一的vector 上。

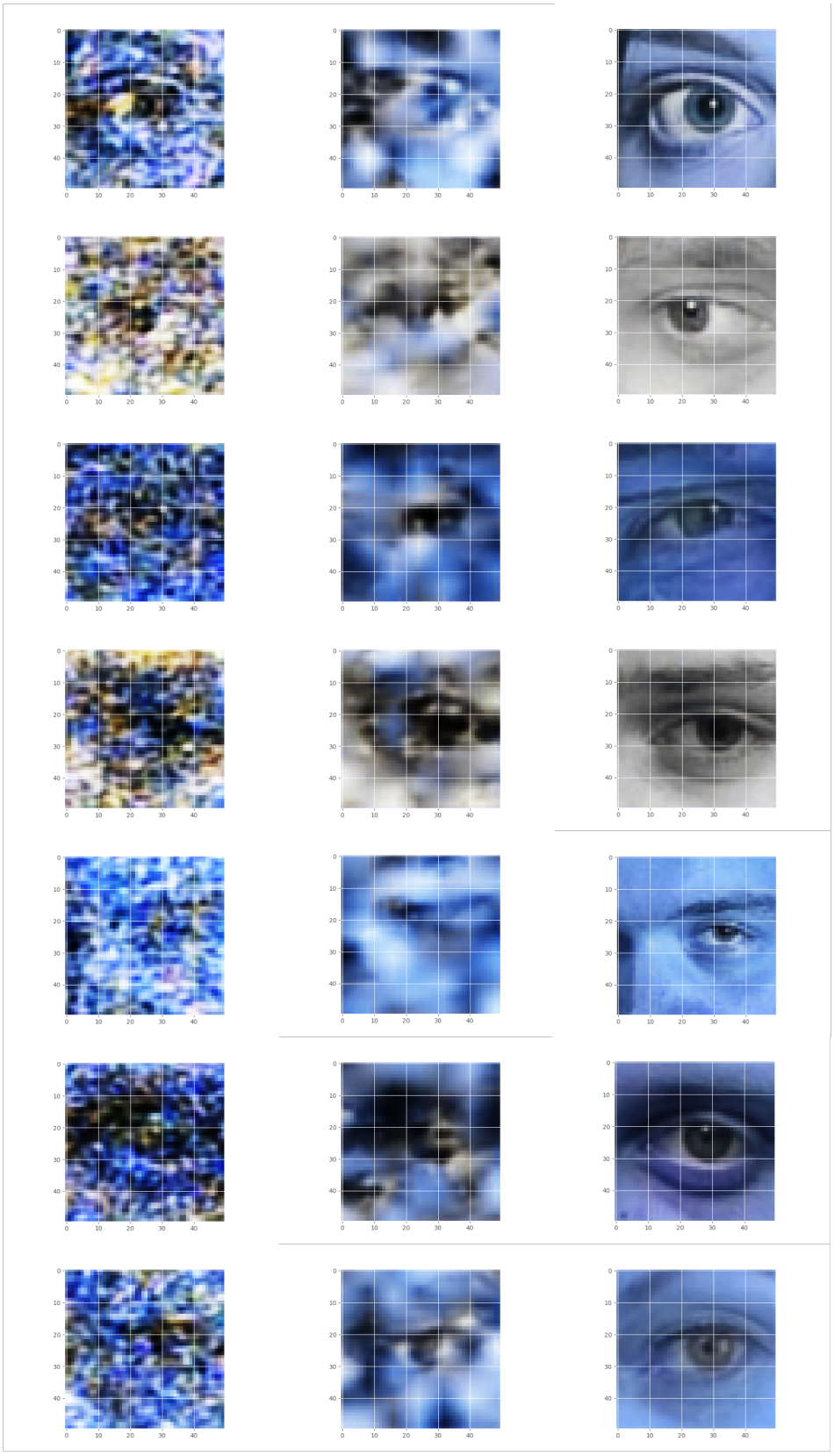
2.5 Preserve high-frequency information

為了能夠保持high-frequency 的information，我們先面臨到的是多層的CNN，可能會在訓練的過程中，忽視掉high-frequency 的內容，因此我提出可以使用到Skip Connection 的方式(圖11)，來將encoder 與decoder 資訊相連，並且用相加的方式，把encoder 的圖像資訊直接傳給decoder。



(a) GT (b) AE (c) VAE (d) SkVAE (d) DpVAE

Figure 9: The result of every model



(a) AE (b) VAE (c) SkVAE

Figure 10: The result of model adding Gaussian noise

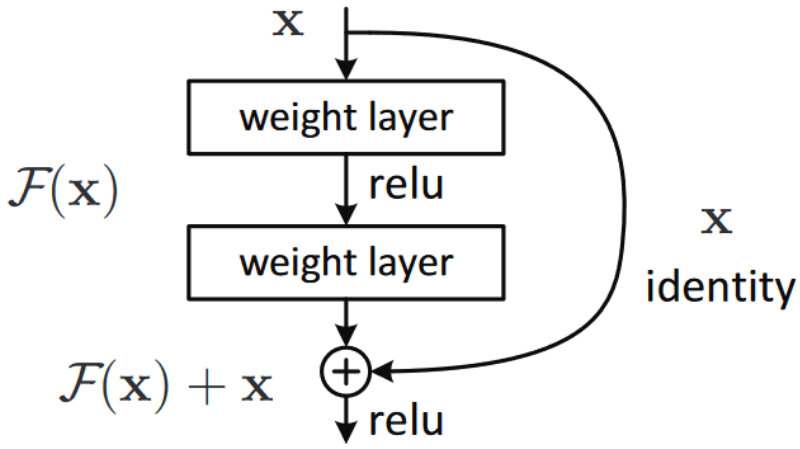


Figure 11: Skip Connection[HZRS15]

可以在表2的分數得知，Skip Connection VAE (SkVAE) 不管在PSNR 或SSIM 上都是最高的，甚至可以超越直接去學習一個vector 的AE。我們也可以從圖9 中的(d)看出SkVAE 的圖像是最接近Ground True。我們也可以從加上Gaussian Noise 的圖10，看出SkVAE 是最像一個眼睛的結果，但同時也可以發現，儘管加上Noise，結果與Ground Truth 仍然非常接近，推斷在Connection 上，所傳輸的資料可能佔比非常重，甚至模型已經忽略掉latent space 上的sample 結果，直接拿encoder 的資訊來產生結果，因此推斷此模型可能已經沒有randomness。

2.6 Deeper model

我們設計一個比VAE 更深的model，在encoder 與decoder 多加上一個Convolution layer $(128, 4, 4)$，我們可以從表2的分數得知，Deeper VAE (DpVAE)相較於比較少層的VAE，在PSNR 和SSIM 的分數都更低，也可以從圖9 中的(e) 看出DpVAE 的圖像，並沒有比較好。有可能是因為模型的深淺需要根據資料量的大小來一起設計，因為這次的資料集只有1476 張image，因此不夠適合太深的模型。也有可能是因為CNN based model 會專注extract 出於receptive field 內的資訊，而原本的layer 設計，已經獲取到的receptive field 資訊相較於新增的layer，並不會比較少。所以才導致deeper model 的分數沒有進步，甚至還多產生不必要的誤導。

References

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[KW22] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.