Report for Course Project of Machine Learning

Pan Wenzheng
520030910232
pwz1121@sjtu.edu.cn

Abstract

This paper presents the results and analysis of the CS3612-Machine Learning course project at Shanghai Jiao Tong University. The project comprises two sections: a mandatory task involving multi-classification on the Fashion-MNIST dataset, and an optional task focused on image reconstruction using Variational Autoencoder (VAE) on the LFW dataset. For both tasks, Option 1 was selected, taking advantage of abundant computational resources. The source code for the mandatory task can be found at: `https://github.com/wzever/Fashion-MNIST_Classification`, and for the optional task at: `https://github.com/wzever/VAE-image-reconstruction` after the due date of the project.

1 Mandatory task: Fashion-MNIST clothing classification (Option 1)

Dataset Fashion-MNIST is a dataset of images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Fashion-MNIST serves as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. Below shows how the dataset looks.

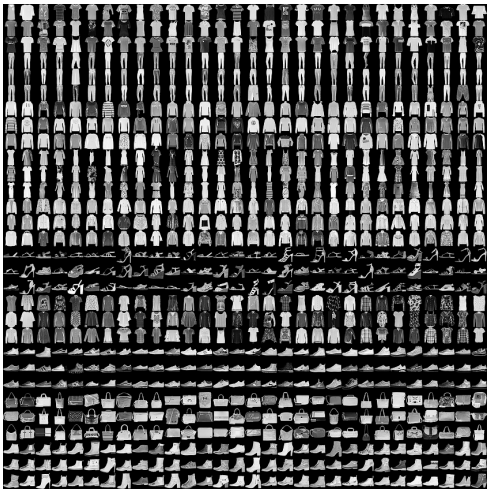

Figure 1: Fashion-MNIST dataset

Overview We developed a deep convolutional neural network (CNN) for image classification, incorporating the strengths of multiple classical networks while introducing unique architectural and hyper-parameter settings. Our network comprises three convolutional blocks and a sequence of fully connected layers, forming the backbone of our model. To expedite training, we utilized a GeForce RTX 2080Ti GPU with 12GB of memory.

The subsequent subsections provide a comprehensive analysis of our network, including its structure, the loss functions employed during training, and the achieved accuracy on both the training and test data splits. Additionally, we present visualizations of the learned features using Principal Component Analysis (PCA) and t-SNE, shedding light on the discriminative power and clustering tendencies within the network. These detailed sections aim to provide a comprehensive understanding of our approach and its performance in the image classification task.

1.1 Network architecture

The network contains 9 convolutional layers, 3 max-pooling layers, 9 batch normalization layers, 7 ReLU/LeakyRelu layers, 2 fully connected layers and 3 dropout layers in total. Figure 2 demonstrate the structure with more details of change in shape of feature maps.

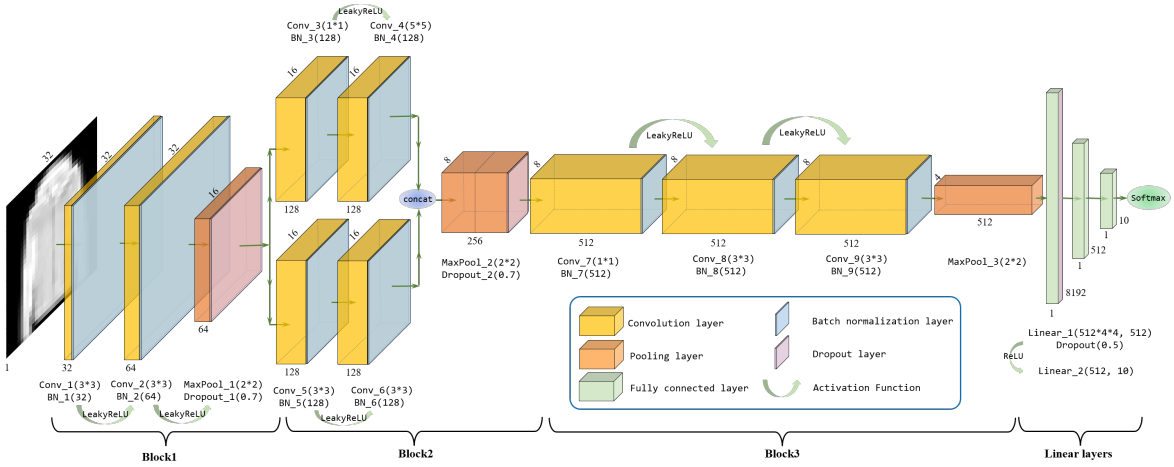

Figure 2: Architecture of proposed deep neural network

Block 1 The network accepts a gray-scale image of shape `[1,32,32]` as input and process the image through two 3*3 convolutional layers (`padding=1, stride=1, out_channels=32,64` respectively) and a 2*2 max-pooling layer. LeakyReLU is adopted as activation function for better maintenance of negative values and in case of dead neurons. Batch normalization layers exist after every convolution. A dropout with probability of 0.7 is applied after max-pooling when training as to avoid over-fitting. The shape of feature map after this module becomes `[64,16,16]`, which typically extracts low-level features of input images.

Block 2 Feature map passes two separate branches forward and get concatenated by dimension 1 (channels). Each path sequences two convolutional layers of 128 output channels. One series has two 3*3 convolutional kernels (`padding=1, stride=1`) and the other has a 1*1 kernel (`padding=0, stride=1`) and a 5*5 kernel (`padding=2, stride=1`). After convolution layers, each branch path produces a feature map of shape `[128,16,16]`, and the concatenation operation outputs an ultimate shape of `[256,16,16]`. The idea of combining features from differently parameterized convolutional filters is inspired by the Inception modules of GoogLeNet. The block also ends up with a 2*2 max-pooling layer and dropout layer, outputting a tensor of shape `[256,8,8]`. The parallel extraction of image features enriches learned characteristics and broadens the width of network, largely improving adaptation to varied scales.

Block 3 This module is comprised of three 512-channel convolution layers, one of which is 1*1 kerneled (no padding) and the other two are 3*3 kerneled (`padding=1, stride=1`), followed by a 2*2 max-pooling layer. Batch normalization is applied after every convolutional layer. Empirically, these filters extract relatively high-level information from the input image. The 1*1 convolutional kernel provides a flexible tool for dimensionality reduction and information fusion with computational efficiency. The output feature map is shaped `[512,4,4]`.

Linear layers First, the feature map is flattened to `[1,8192]` and passes through two fully connected layers (and ReLUs) with output channels of 512 and 10 respectively. So far, a 10-dimensional vector is generated, which then is transformed into predictive probabilities of the 10 classes by softmax function.

1.2 Settings and Hyper-parameters

The performance of a classification CNN is greatly influenced by the selection of hyper-parameter settings during training. The following Table 1 provides a comprehensive overview of the specifically tuned hyper-parameter values used in our experiments, which directly contributed to the reported results.

Table 1: Settings and hyper-parameters for training

Name	Description	Value/Setting
loss_func	loss function	Cross entropy loss
optimizer	Optimizer for gradient descent	Adam
lr_scheduler	Scheduler to adjust learning rate dynamically	StepLR
num_epochs	Maximum training epochs	100
lr	(Initial) Learning rate	0.0005
wd	Weight decay	0.001
lr_period	Number of epochs of learning rate decay	10
lr_decay	Multiplicative factor of learning rate decay	0.5
seed	Random seed	6
batch_size	training and testing batch size	64

1.3 Loss curves

Figure 3 demonstrates the curves for training loss and testing loss after every epoch.

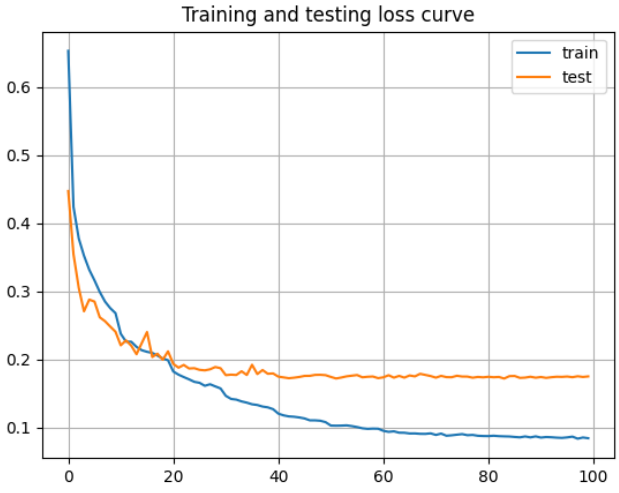

Figure 3: Loss curves

To mitigate the influence of fluctuations in the curves, we have calculated the average losses across all mini-batches within each epoch. It is important to note that both the training loss and testing loss demonstrate a rapid decrease initially. After approximately 40 epochs, the testing loss begins to level off, indicating a satisfactory convergence of the model without significant signs of overfitting.

1.4 Model performance

For such multi-classification task, accuracy is used to measure the performance of a model. Figure 4 demonstrates the curves for training and testing accuracy after every epoch.

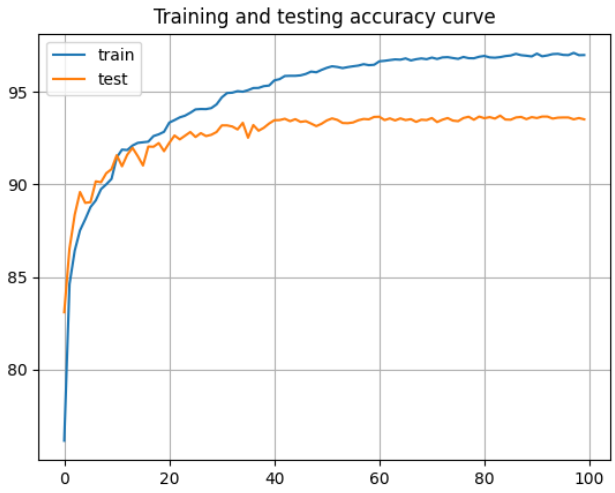

Figure 4: Accuracy curves

Under the tuned settings and parameterization in section 1.2, our proposed model reaches a reproducible best accuracy of **93.72%** on test set.

For comparative analysis, Table 2 lists the reported accuracy of several classical models on Fashion-MNIST dataset. The source data may be found at `https://github.com/zalandoresearch/fashion-mnist/tree/master`

Table 2: Benchmarks of different classifiers on Fashion-MNIST

Classifier	Preprocessing	Test accuracy
MLP 256-128-100	None	0.883
2 Conv + pooling	None	0.916
HOG+SVM	HOG	0.926
2 Conv + 3 FC	Normalization	0.932
VGG16	None	0.935
Designed model (ours)	Normalization	**0.937**
ResNet18	Normalization, horizontal/vertical flip, translation, rotation	0.949
MobileNet	Augmentation, horizontal flips	0.950

The results of our experiments indicate that our proposed network achieves comparable accuracy to that of mainstream models. This outcome suggests that our architecture design and code implementation are free from significant defects or errors. By demonstrating competitive performance, our findings reinforce the effectiveness and reliability of our approach in the task of image classification.

1.5 Feature visualization

PCA Principal Component Analysis (PCA) projects each data point onto only a few principal components to obtain lower-dimensional data, while preserving as much of the variance of data as possible.

Specifically, given n samples $X \in \mathbb{R}^{n\times 784}$ and a trained neural network $g(X)$, let $g(X) \in \mathbb{R}^{n\times p}$ denote the feature map in a specific intermediate layer of these input sample, we conducted PCA to transform $g(X)$ to $g'(X) \in \mathbb{R}^{n\times 2}$ and plot them in a diagram.

Figure 5 visualizes the feature map of a convolutional layer, a fully connected layer and the final layer of our model. Each color of point represents one of the 10 classes of the input images, and 50 images are sampled for display from each class. The selected layers are `Conv_9`, `Linear_1` and the final output layer in Figure 2.

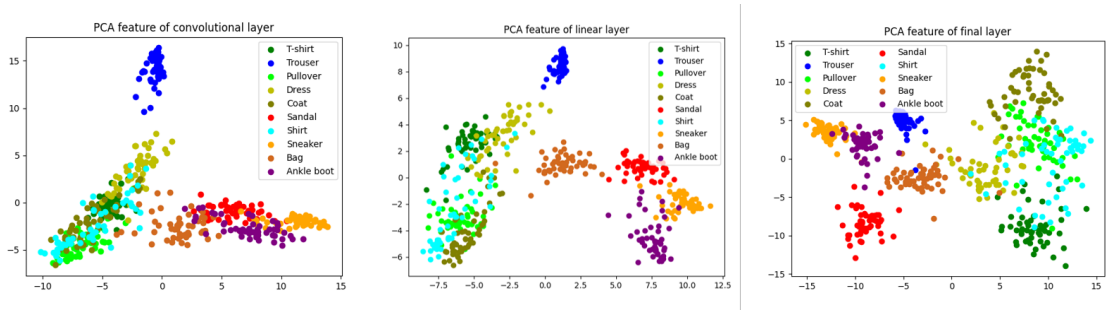

Figure 5: PCA visualization

t-SNE Stochastic Neighbor Embedding (t-SNE) uses lower dimensional vectors so as to preserve relationships that are in higher dimensional.

Specifically, given n samples $X \in \mathbb{R}^{n \times 784}$ and a trained neural network $g(X)$, let $g(X) \in \mathbb{R}^{n \times p}$ denote the feature map in a specific intermediate layer of these input sample, we conducted t-SNE to transform $g(X)$ to $g'(X) \in \mathbb{R}^{n \times 2}$ and plot them in a diagram.

Figure 6 visualizes the feature map of a convolutional layer, a fully connected layer and the final layer of our model. Each color of point represents one of the 10 classes of the input images, and 50 images are sampled for display from each class. The selected layers are `Conv_9`, `Linear_1` and the final output layer in Figure 2.

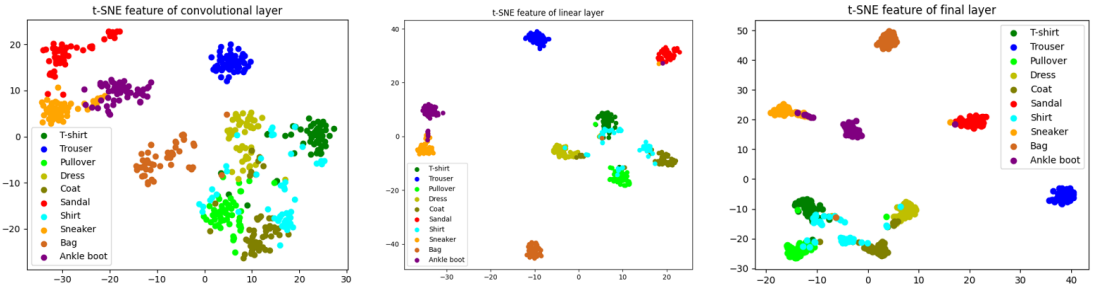

Figure 6: t-SNE visualization

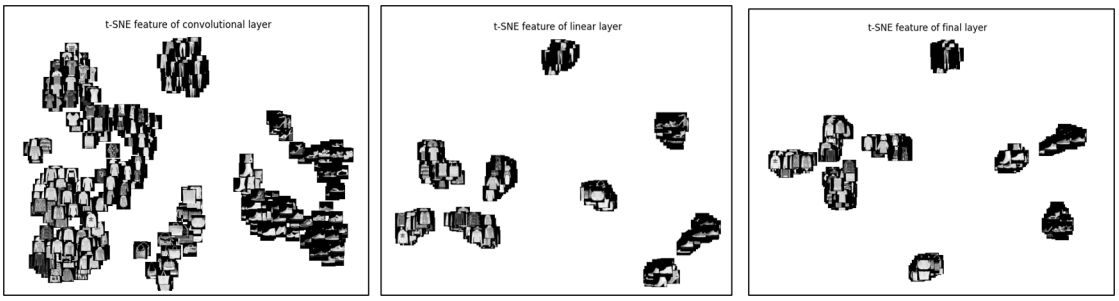

Figure 7: t-SNE visualization with real image display

Figure 7 exhibits the sampled images according to their coordinates, which presents a more intuitive view to understand how the images cluster.

Conclusion Based on the aforementioned observations, our designed neural network performs effectively for this project. The visualizations reveal that the clustering tends to improve as the layers progress towards the final output. However, it is worth noting that certain classes exhibit significant overlaps, which explains the challenges encountered in some cases. Specifically, distinguishing between shirts and t-shirts, as well as coats and pullovers, proves to be difficult, even for humans. Despite these challenges, we have successfully fulfilled all the requirements of the mandatory task, achieving the desired outcomes.

2 Optional task 1: Image reconstruction (Option 1)

2.1 VAE architecture

VAE (Variational AutoEncoder) is a type of generative model commonly used for image reconstruction and generation tasks. It is a combination of an encoder and a decoder neural network, which work together to learn a compact representation of input images and generate new images based on that representation.

Figure 8 demonstrates the hierarchical construction of the VAE network designed for this project.

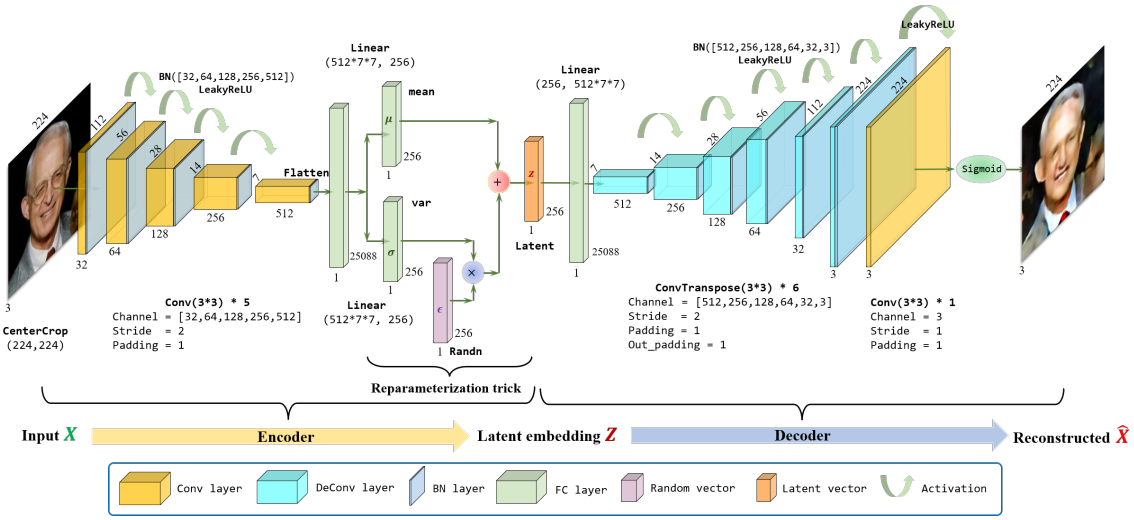

Figure 8: Architecture of proposed VAE network

Encoder The encoder network takes an input image and maps it to a lower-dimensional latent space. It consists of convolutional layers followed by fully connected layers, gradually reducing the spatial dimensions of the image and capturing its essential features. The final layers of the encoder produce the mean (μ) and variance (σ) of a multivariate Gaussian distribution that represents the latent space. To fit the distribution of original $p(X)$, a posterior distribution of latent variable Z is used

$$p(X) = \sum_{Z} p(X|Z)p(Z) \tag{1}$$

where $p(x|Z)$ is expected to approach a normal distribution during the training process.

Decoder The decoder network takes a point from the latent space and reconstructs an image from it. It mirrors the encoder architecture, gradually increasing the spatial dimensions until the output image size is reached. We adopted `ConvTranspose2d` module in Pytorch for trainable up-sampling. The output layer of the decoder uses `Sigmoid` function to squash the pixel values within $[0, 1]$ for the calculation of BCE loss.

2.2 Implementation details

Loss Function The objective of the VAE is to minimize the reconstruction error while ensuring that the latent space follows a standard Gaussian distribution which is measured by KL-divergence. Thus, the loss function consists of two parts: the reconstruction loss and the KL-divergence loss.

- Reconstruction loss: It measures the similarity between the original input image and the reconstructed output image. In experiments, we tried L1 loss, L2 loss (MSE) and BCE loss (Binary Cross Entropy) respectively and found BCE loss outperforming the others regarding reconstructing quality.

$$\mathcal{L}_{BCE} = -\frac{1}{n}\sum_{i}\big[\big(X_i \cdot \log \hat{X}_i\big) + \big((1 - X_i) \cdot \log(1 - \hat{X}_i)\big] \tag{2}$$

  where X is the input original image while $\hat{X}$ denotes the output reconstructed image tensor. This loss measures differences between X and $\hat{X}$ depending on the pixel-wise characteristics of the image.

- KL-divergence loss: Kullback-Leibler (KL) divergence here also acts as regularization, which encourages the latent space to adhere to the desired distribution $\mathcal{N}(0, 1)$. It quantifies the difference between the learned distribution and the target distribution. Minimizing the KL divergence ensures that the latent space is continuous and allows smooth interpolation between samples.

$$\mathcal{L}_{KLD} = KL\big(\mathcal{N}(\mu, \sigma^2)||\mathcal{N}(0, 1)\big) = \frac{1}{2}(-\log\sigma^2 + \mu^2 + \sigma^2 - 1) \tag{3}$$

Thus, the total loss for training is

$$\mathcal{L} = \mathcal{L}_{BCE} + \mathcal{L}_{KLD} \tag{4}$$

Reparameterization Trick The latent space is a lower-dimensional representation of the input image, often referred to as the bottleneck layer. To sample a point in the latent space during training, we use the reparameterization trick. Since sampling operation is non-differentiable, the trick involves sampling from a standard Gaussian distribution, multiplying the samples by the standard deviation (σ), and adding the mean (μ).

$$z = \sigma_x \cdot \epsilon + \mu_x \tag{5}$$

z is thus guaranteed to follow a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ if $\epsilon \sim \mathcal{N}(0, I)$. This process introduces stochasticity to the model, allowing it to compute backward propagation and generate diverse samples during training.

Hyper-parameters Table 3 lists the hyper-parameters of our training for VAE network, as well as the selection of the optimizer, scheduler for learning rate, etc.

Table 3: Settings and hyper-parameters for VAE training

Name	Description	Value/Setting
pre_process	Pre-processing of input image	CenterCrop(224)
optimizer	Optimizer for gradient descent	Adam
lr_scheduler	Scheduler to adjust learning rate dynamically	StepLR
num_epochs	Maximum training epochs	1000
lr	(Initial) Learning rate	0.0003
lr_period	Number of epochs of learning rate decay	25
lr_decay	Multiplicative factor of learning rate decay	0.95
seed	Random seed	42
batch_size	training and testing batch size	64

All the hyper-parameters above are tuned on the full lfw dataset of 13233 images with fixed random seed of 42, which rigorously guaranteed the reproducibility of our experimental results.

2.3 Reconstructed images

The figures presented below showcase randomly selected image batches and their corresponding reconstruction results obtained from the VAE decoder. Figure 9 displays a set of 64 image pairs, juxtaposing the original images alongside their reconstructed counterparts. Figure 10 provides an alternative perspective by presenting a pair-wise comparison of 96 image pairs, which allows for a more detailed assessment of the original images in comparison to their reconstructed counterparts, facilitating better understanding intelligibility.

Figure 9: Example 1: comparison between original and reconstructed images (left: original inputs; right: reconstructed outputs).

Figure 10: Example 2: pair-wise comparison between original and reconstructed images (left: original inputs; right: reconstructed outputs, for every two columns).

Note that the reconstruction basically restores the characteristics of the original input. It can be inferred that the quality of reconstructed images are affected by a number of factors. The face position, background complexity and number of appearing figures can influence the outcome.

2.4 Interpolation for image generation

To generate faces with characteristics from two distinct group of people, we first feed the images to the pretrained VAE network as to obtain the latent representative embedding encoded by the encoder.

For example, given two output features of the encoder, i.e., z_1 and z_2, the decoder takes z_1 as the input to generate a face image of a woman, and takes z_2 as the input to generate a face image of a man. Linear interpolation is utilized to produce a new feature $\tilde{z}$ with

$$\tilde{z} = \alpha \cdot z_1 + (1 - \alpha) \cdot z_2 \tag{6}$$

Then, the decoder takes $\tilde{z}$ as the input to generate a new face image. Experiments are conducted over two contrary facial features (Male and Female, Old and Young) and 10 alphas ranging from 0 to 1.

Figure 11 Figure 12 displays the result of image fusion from male/female and old/young faces, respectively.

Figure 11: Image fusion by VAE using linear interpolation of latent representation: Male and Female

2.5 Analysis and conclusion

In this task, we explored the application of a VAE network for generative image fusion, specifically targeting the integration of attributes such as gender and age. Through the use of linear interpolation in the latent space, the model generates new hidden embeddings that effectively captured the transitions between male and female, as well as old and young features. By feeding these interpolated

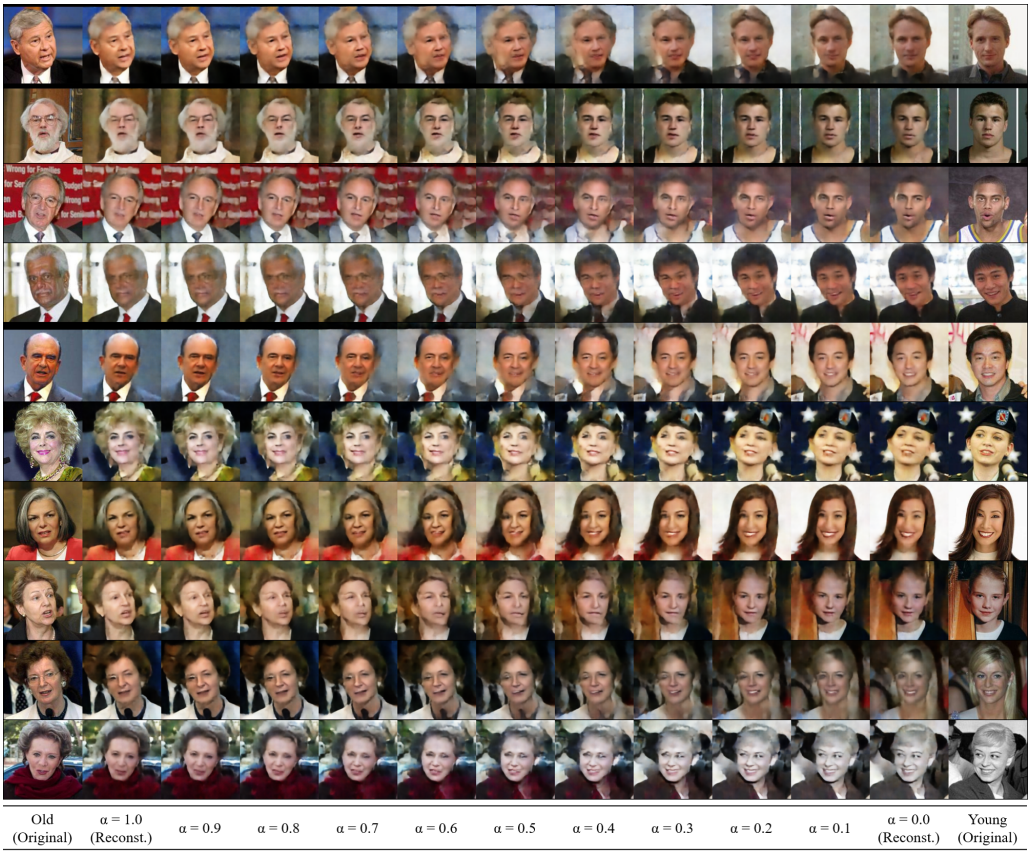

Figure 12: Image fusion by VAE using linear interpolation of latent representation: Old and Young

embeddings to the decoder, we successfully synthesized novel images that exhibit moderate and believable transformations.

However, it is important to note that not all generated faces achieved perfection. Several factors contributed to the presence of noise and blurry zones in the synthesized images. One potential cause is the complexity of the underlying data distribution. The latent space of the VAE may not fully acquire the intricate variations in facial attributes, resulting in some inconsistencies or distortions. Additionally, there is much room for improvement lying in the network architecture and model capacity designed by ourselves, which can lead to suboptimal results.

Also, the selection of interpolation techniques can influence the overall quality of the synthesized images. Linear interpolation, while providing a straightforward approach, may not always cover the subtle nuances between attributes in a highly expressive manner.

Despite these challenges, our experiments demonstrate the potential of VAE networks for image reconstruction and generation. To conclude, this project highlights the power of VAE networks in capturing and manipulating image features. While some imperfections and limitations persist, this work paves the way for further understanding in image transformation and generation using deep-learning-based methods, which offers significant foundation for future research and development in our academic career.

Acknowledgments

Sincere appreciation goes to Prof. Zhang for lecturing in this insightful course and all TAs for their conducive guidance.

References

[1] Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747.

[2] Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.

[3] Huang, G. B., Mattar, M., Berg, T., & Learned-Miller, E. (2008, October). Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. In Workshop on faces in'Real-Life'Images: detection, alignment, and recognition.

[4] Subramanian, A.K. (2020). PyTorch-VAE [GitHub repository]. GitHub. Retrieved from `https://github.com/AntixK/PyTorch-VAE`

[5] Variational AutoEncoder. (2023) In Wikipedia. Retrieved from `https://en.m.wikipedia.org/wiki/Variational_autoencoder`