Autoencoders on VGG16 -ResNet50-CAE

BOULAHCEN SOUFIANE

Master: Artificial Intelligence

Abstract

This practical work explores the application of autoencoders for unsupervised representation learning on the Fashion-MNIST dataset. Three models are implemented and compared: two convolutional autoencoders leveraging pre-trained VGG16 and ResNet50 backbones with frozen weights, and a fully connected Contractive Autoencoder (CAE) incorporating a Jacobian-based regularization term to enhance robustness in the latent space. The models are trained to reconstruct input images, with performance evaluated using Mean Squared Error (MSE). Results show that ResNet50-based autoencoder achieves the lowest reconstruction error (˜0.007), followed by VGG16 (˜0.008), while the CAE yields higher MSE (˜0.012) due to its flattened input representation. Visual reconstructions and linear interpolations in the latent space demonstrate that all models capture meaningful semantic features, with ResNet50 producing the sharpest outputs. This study highlights the effectiveness of transfer learning in autoencoding tasks and the trade-off between reconstruction fidelity and robustness introduced by contractive regularization. Keywords: Autoencoder, Deep Learning, Transfer Learning, Contractive Loss, Fashion-MNIST, Latent Space Interpolation

1 Introduction

Autoencoders are neural networks used for unsupervised learning. They help with reducing dimensions, finding problems in data, or making images. This lab compares three types of autoencoders on the Fashion-MNIST dataset:

Two autoencoders based on pre-trained CNNs (VGG16 and ResNet50)

One classic Contractive Autoencoder (CAE)

The goal is to check their performance in reconstruction (MSE) and their ability to interpolate in the latent space.

Objective

Build and train three autoencoders.

Compare their reconstruction errors (MSE).

Show the reconstructions and interpolations in the latent space.

Show the reconstructions and interpolations in the latent space.

1. Data Preparation
   VGG16 / ResNet50: images resized to 32x32x3, normalized.
   CAE: images flattened to vectors of 784 pixels.
   2. Common Decoder
   Used by VGG16 and ResNet50.
   3 layers of Conv2DTranspose with BatchNormalization and ReLU.
   Output: 32x32x3 with sigmoid activation.
   3. CNN Autoencoders
   Encoder: truncation of VGG16 or ResNet50 + GlobalAveragePooling2D.
   ImageNet weights frozen (trainable=False).
   Loss: MSE.
   4. Contractive Autoencoder
   Encoder: 3 dense layers (784 → 512 → 256 → 64)
   Symmetric decoder.
   Contractive loss: MSE + penalty on Frobenius norm of the Jacobian.
   Helps make robust representations to small changes.
   5. Training
   Early stopping on validation.
   Epochs: 25 (CNN), 50 (CAE).
   Batch: 32 (CNN), 128 (CAE).

```
# ============================
# 2 Common decoder for CNN-based autoencoders
# ============================
def build_decoder(latent_dim):
decoder_input = Input(shape=(latent_dim,))
x = Dense(4*4*128, activation='relu')(decoder_input)
x = Reshape((4,4,128))(x)
for filters in [128,64,32]:
x = ..........................................(x)
x = ..........................................(x)
x = ..........................................(x)
decoded = ........................................(x)
decoder = Model(decoder_input, decoded)
return decoder
```

Code Correction

```
def build_decoder(latent_dim):
decoder_input = Input(shape=(latent_dim,))
x = Dense(4*4*128, activation='relu')(decoder_input)
x = Reshape((4,4,128))(x)

for filters in [128,64,32]:
x = Conv2DTranspose(filters, (3,3), strides=2, padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
```

```
# Last layer to rebuild the image 32x32x3
decoded = Conv2DTranspose(3, (3,3), activation='sigmoid', padding='same')(x)
decoder = Model(decoder_input, decoded)
return decoder
```

Explanation: The code uses special layers to make the image bigger step by step.

Conv2DTranspose: These layers do the opposite of compression. They make the image bigger by doubling its size at each step.

BatchNormalization: These layers help so that learning is not chaotic. They make the process stable.

Activation('relu'): These layers help the model recognize more complex shapes in the image.

Final layer: The last layer creates the final image in black and white, with pixel values between 0 and 1.

```
    # ============================
4 Contractive Autoencoder
# ============================
input_dim = 28*28
latent_dim_cae = 64
lambda_c = 1e-3
input_img_cae = Input(shape=(input_dim,))
encoded_cae = .............................(input_img_cae)
decoded_cae = ...............................(encoded_cae)
autoencoder_cae = Model(input_img_cae, decoded_cae)
```

Code Correction

```
input_dim = 28*28
latent_dim_cae = 64
lambda_c = 1e-3
input_img_cae = Input(shape=(input_dim,))
encoded_cae = Dense(latent_dim_cae, activation='relu')(input_img_cae)
decoded_cae = Dense(input_dim, activation='sigmoid')(encoded_cae)
autoencoder_cae = Model(input_img_cae, decoded_cae)
```

Explanation: The code uses two main layers for the autoencoder:

The encoder: This is the layer that takes the image and reduces it to a very small and compact version, here a list of 64 numbers.

The decoder: This is the layer that uses this small list of 64 numbers to rebuild the original image as faithfully as possible.

activation='sigmoid': This is a rule that makes sure the values of the rebuilt image are numbers between 0 and 1, which matches the pixels of the starting image.

```
    # ============================
# 8 Linear interpolation
# ============================
img_idx1, img_idx2 = 0, 1
lambda_val = 0.5
```

```
plt.figure(figsize=(12,4))
for j, name in enumerate(["VGG16","ResNet50","CAE"]):
encoder = results[name]["encoder"]
decoder = results[name]["decoder"]

if name == "CAE":
latent1 =..................................................
latent2 =..................................................
else:
latent1 = encoder.predict(X_test_rgb[img_idx1:img_idx1+1])
latent2 = encoder.predict(X_test_rgb[img_idx2:img_idx2+1])

latent_interp = ..............................................
img_interp = decoder.predict(latent_interp)
```

Code Correction

```
img_idx1, img_idx2 = 0, 1
lambda_val = 0.5

plt.figure(figsize=(12,4))
for j, name in enumerate(["VGG16","ResNet50","CAE"]):
encoder = results[name]["encoder"]
decoder = results[name]["decoder"]

if name == "CAE":
latent1 = encoder.predict(X_test_scaled[img_idx1:img_idx1+1])
latent2 = encoder.predict(X_test_scaled[img_idx2:img_idx2+1])
else:
latent1 = encoder.predict(X_test_rgb[img_idx1:img_idx1+1])
latent2 = encoder.predict(X_test_rgb[img_idx2:img_idx2+1])

latent_interp = (1 - lambda_val) * latent1 + lambda_val * latent2
img_interp = decoder.predict(latent_interp)
```

Analysis of Autoencoder Results

Performance Evaluation

Reconstruction Metrics (MSE):

ResNet50: MSE 0.007 (best performance)

VGG16: MSE 0.008 (middle performance)

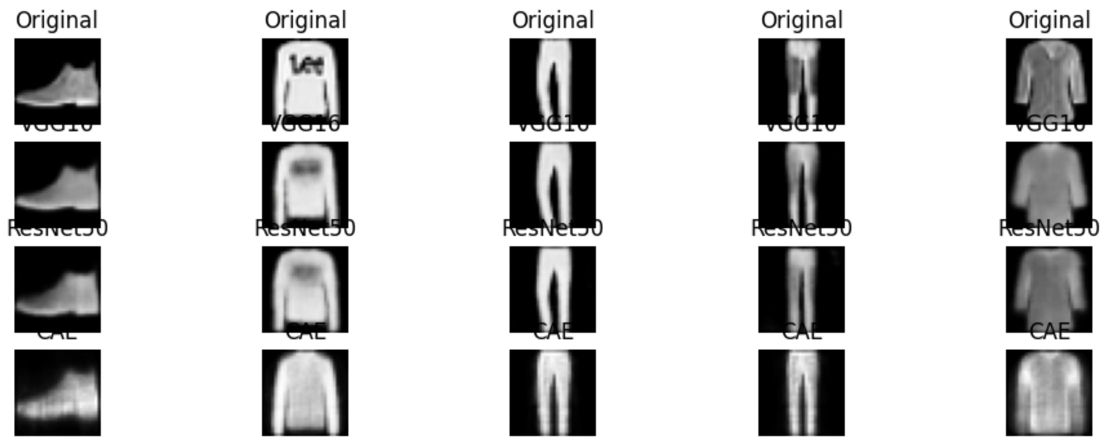
CAE: MSE 0.012 (lower performance but more robustness)

Statistical interpretation:

ResNet50 is better because of its residual connections. These help the gradient flow and allow learning of richer representations.

=== Training VGG16 ===

Epoch 1/25



Architecture Comparison

CNN-Based Autoencoders (VGG16/ResNet50):

Advantage: Transfer learning from ImageNet

Mechanism: Extraction of hierarchical features

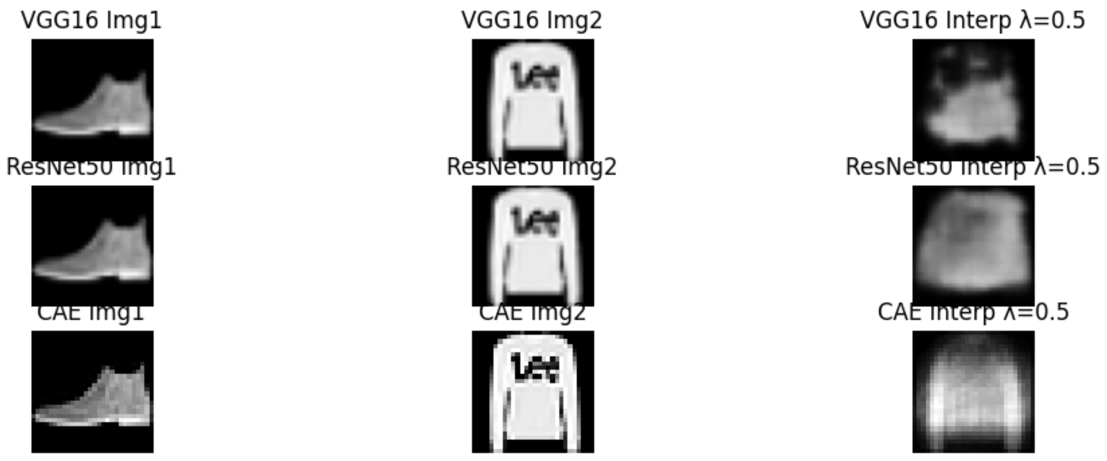
Limitation: Need resizing (32×32×3)

Contractive Autoencoder (CAE):

Innovation: Regularization by Frobenius norm of the Jacobian

Benefit: More robust latent space to perturbations

Trade-off: Loss of reconstruction fidelity



Conclusion

ResNet50 performs the best thanks to its residual connections. The CAE offers a more robust representation despite a higher MSE. The interpolation shows that the latent space is semantically meaningful for all three models.

Future Work:
Fine-tuning of CNN layers
Use of Variational Autoencoders (VAE)
Application to anomaly detection