Report

**Convolution Neural Network: Done by Vijay(se22ucse305)**

First neural network we are using is Convolution Neural network for classification of 10 different images (“Top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle Boot”) .My Neural is

model = Sequential([

model = Sequential([

Conv2D(32, (3, 3), activation='elu', padding = 'same', input\_shape=(28, 28, 1)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='elu', padding = 'same'),

Conv2D(64, (3, 3), activation='elu', padding = 'same'),

MaxPooling2D((2, 2)),

Conv2D(128, (3, 3), activation='elu',padding = 'same'),

Conv2D(128, (3, 3), activation='elu',),

MaxPooling2D((2, 2)),

Flatten(),

Dense(128, activation='elu'),

Dropout(0.2),

Dense(10, activation='softmax')

])

Rationale:

(3 × 3) is chosen because STL10 Dataset images are not too noisy.

The network is designed based on my intuition.

The network commences with a single 3×3 convolutional layer employing 32 filters and ELU activation. This initial layer is intended to extract fundamental features—such as edges and simple textures—while maintaining a modest parameter count. A subsequent 2×2 max-pooling operation reduces the spatial dimensions to 14×14,(Yeah! Max- pooling reduces important feature but not necessarily all) thereby concentrating representational capacity on salient regions and reducing computational overhead.

The second block comprises two consecutive 3×3 convolutional layers with 64 filters each, enabling the model to refine basic features into more complex mid-level patterns. A further 2×2 max-pooling(!same) step halves the feature map size to 7×7, focusing the network’s capacity on increasingly abstract representations.

In the third block, the filter depth is increased to 128 for two additional 3×3 convolutions. At this stage, the network’s receptive field encompasses a substantial portion of the original input, justifying a higher channel dimension to capture intricate, high-level features that differentiate the ten target classes. A final 2×2 max-pool reduces the feature maps to 3×3, distilling spatial information into a compact set of activations.

Following flattening, a fully connected layer of 128 units integrates the learned hierarchical features into a concise representation. The final softmax layer produces class probabilities for the ten categories. Overall, this architecture transitions systematically from low to high capacity—as filter depth increases and spatial resolution decreases—thereby achieving an effective balance between parameter efficiency, hierarchical feature extraction, and discriminative power.

**Key Hyper parameters:**

1. **Filter Progression (32 → 64 → 128)**  
   Increasing the number of convolutional filters with depth allows the network to learn progressively more complex features. Early layers focus on simple patterns with fewer filters (32), while deeper layers require greater capacity (128) to capture high-level abstractions.
2. **Kernel Size (3×3)**  
   The 3×3 kernel is the industry standard and computational cost manageable. Stacking multiple 3×3 convolutions also increases receptive field without the expense of larger kernels.
3. **Activation Function (ELU)**  
   ELU (Exponential Linear Unit) is chosen for its ability to produce negative outputs, which centres activations around zero. This behaviour speeds up convergence and stabilises training by reducing bias shifts compared to ReLU.(Since the number of images are 60,000)
4. **Pooling Size (2×2)**  
   A 2×2 max-pool halves each spatial dimension, reducing feature-map size and computational load while preserving the strongest activations. This coarse reduction encourages spatial invariance and prevents overfitting.
5. **Dropout** should be enabledif the number epochs are higH because training for too many epochs without dropout our model overfits for the test data and gives higher accuracy for training.Test accuracy will be bad.(I hashes out because I trained for 5 epochs only)

final Thoughts :

This network classifies with accuracy of .8911 for 5 Epochs.Epoch 1/5

**1875/1875** ━━━━━━━━━━━━━━━━━━━━ **201s** 107ms/step - accuracy: 0.6779 - loss: 0.9598

Epoch 2/5

**1875/1875** ━━━━━━━━━━━━━━━━━━━━ **201s** 106ms/step - accuracy: 0.8415 - loss: 0.4338

Epoch 3/5

**1875/1875** ━━━━━━━━━━━━━━━━━━━━ **197s** 105ms/step - accuracy: 0.8678 - loss: 0.3597

Epoch 4/5

**1875/1875** ━━━━━━━━━━━━━━━━━━━━ **198s** 106ms/step - accuracy: 0.8797 - loss: 0.3257

Epoch 5/5

**1875/1875** ━━━━━━━━━━━━━━━━━━━━ **201s** 107ms/step - accuracy: 0.8911 - loss: 0.2963

<keras.src.callbacks.history.History at 0x78b4d4604910>

If we want to improve accuracy we have to increase number of epochs getting trained a 15 epoch training would led to accuracy of 96 and loss of .15(Approx). My model predictions is out of the ordinary for its accuracy and haven't seen one wrong prediction.So my model can be trusted.(hahaha)

One more thing test accuracy :**313/313** ━━━━━━━━━━━━━━━━━━━━ **10s** 32ms/step - accuracy: 0.8854 - loss: 0.3259!> 0.0001

HelmHoltz Machine: Done by Vijay(se22ucse305)

Rationale:

Disclaimer : I took significant help from Generative Artificial Intelligence (A.I) to write this code (!Help).New to PyTorch

Now come back to the report.

Setup : Performed it on MNIST ( 0 to 9 numbers)

Used key tools :PyTorch, torch.nn, torch.optim and torch vision.

torch.manual\_seed(42) ensures random weight initialisations.

**Hyper Parameters:**

input\_size = 784: Each MNIST image is 28×28 pixels. We flatten it into a 784‑dim vector.

hidden\_size = 200: The number of latent (hidden) units the model uses to explain each image.

num\_classes = 10: classified digits 0–9.

batch\_size, num\_epochs, learning\_rate: Control the mini‑batch size, how many full passes through the data, and how big our gradient steps are.

dropout\_prob = 0.5: I randomly drop half of the hidden units during training for regularisation.

**Recognition Network (Bottom‑Up Model):**

**Purpose**: Given a real image, infer two things:

1. A binary latent code (size 200) that “explains” the image.
2. A classification into one of 10 digit classes.

**Architecture**:

1. A fully‑connected layer (784 to 200) + ReLU + dropout.
2. Another fully‑connected layer (200 to 200) whose outputs are passed through a sigmoid to give probabilities for each latent bit. We sample those bits during training (Bernoulli).
3. A fully‑connected classification head (200 to 10), producing logits for the 10 digit classes.

**Generative Network (Top‑Down Model)**

* **Purpose**: Take a sampled latent code (200‑dim binary vector) and reconstruct an image.
* **Architecture**:
  1. Fully‑connected (200 to 200) + ReLU + dropout.
  2. Fully‑connected (200 to 784) + sigmoid to produce pixel intensities between 0 and 1.

**Helmholtz Machine Wrapper**

* Combines the recognition and generative networks.
* forward(x): Runs bottom‑up inference (to get z and class logits) then top‑down reconstruction (x\_recon).
* generate(batch\_size): Samples random latent codes from a simple prior (here, each bit = 1 with 50% chance) and runs them through the generator to produce “dreamed” images.

**Wake–Sleep algorithm**

**Wake Phase**

* **Goal**: Teach the *generator* to reconstruct real images.
* **Steps**:
  + Infer latent bits z and class logits from a real image batch.
  + Compute reconstruction loss (binary cross‑entropy between x\_recon and the input) and classification loss (cross‑entropy on the digit label).
  + Back propagate through both recognition & generative networks (via the combined “wake loss”) and update weights.

**Sleep Phase**

* **Goal**: Teach the *recogniser* to correctly infer latents from generated images.
* **Steps**:
  + Sample random latent codes z\_dream from the prior and generate fake images x\_dream.
  + Run these dreams through the recognition network to get inferred probabilities z\_dream\_recon\_probs.
  + Compute a sleep loss (binary cross‑entropy between inferred probabilities and the original sampled bits) and update only the recognition network.

**What I got after Training:** Better Helmholtz Machine and the below results.

Epoch 1/10:

Wake Loss: 0.6618, Sleep Loss: 0.6931, Class Loss: 0.4095, Train Acc: 88.66%

Test Loss: 0.1821, Test Acc: 94.63%

Epoch 2/10:

Wake Loss: 0.4212, Sleep Loss: 0.6861, Class Loss: 0.2047, Train Acc: 94.00%

Test Loss: 0.1313, Test Acc: 96.16%

Epoch 3/10:

Wake Loss: 0.3712, Sleep Loss: 0.6859, Class Loss: 0.1631, Train Acc: 95.25%

Test Loss: 0.1089, Test Acc: 96.72%

Epoch 4/10:

Wake Loss: 0.3420, Sleep Loss: 0.6857, Class Loss: 0.1380, Train Acc: 95.91%

Test Loss: 0.0930, Test Acc: 97.33%

Epoch 5/10:

Wake Loss: 0.3242, Sleep Loss: 0.6852, Class Loss: 0.1230, Train Acc: 96.28%

Test Loss: 0.0851, Test Acc: 97.30%

Generated 10 samples at epoch 5

Epoch 6/10:

Wake Loss: 0.3118, Sleep Loss: 0.6850, Class Loss: 0.1134, Train Acc: 96.64%

Test Loss: 0.0833, Test Acc: 97.42%

Epoch 7/10:

Wake Loss: 0.2995, Sleep Loss: 0.6845, Class Loss: 0.1033, Train Acc: 96.86%

Test Loss: 0.0763, Test Acc: 97.64%

Epoch 8/10:

Wake Loss: 0.2898, Sleep Loss: 0.6846, Class Loss: 0.0946, Train Acc: 97.03%

Test Loss: 0.0764, Test Acc: 97.82%

Epoch 9/10:

Wake Loss: 0.2841, Sleep Loss: 0.6844, Class Loss: 0.0905, Train Acc: 97.19%

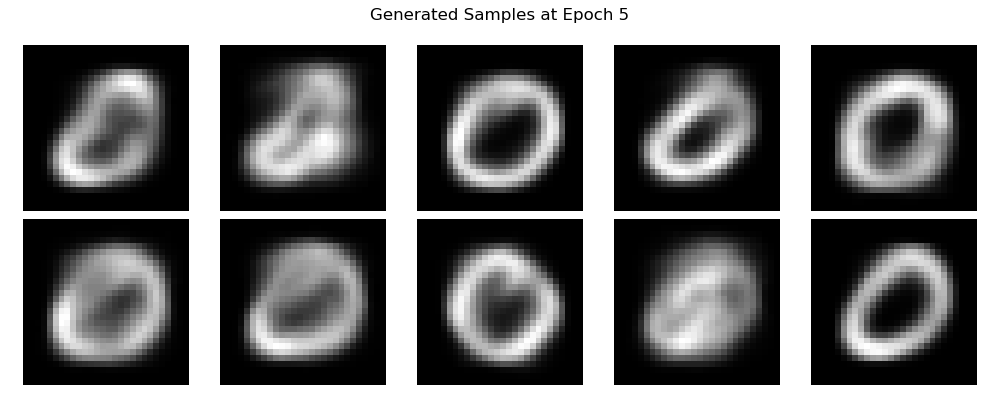
Test Loss: 0.0725, Test Acc: 97.83%

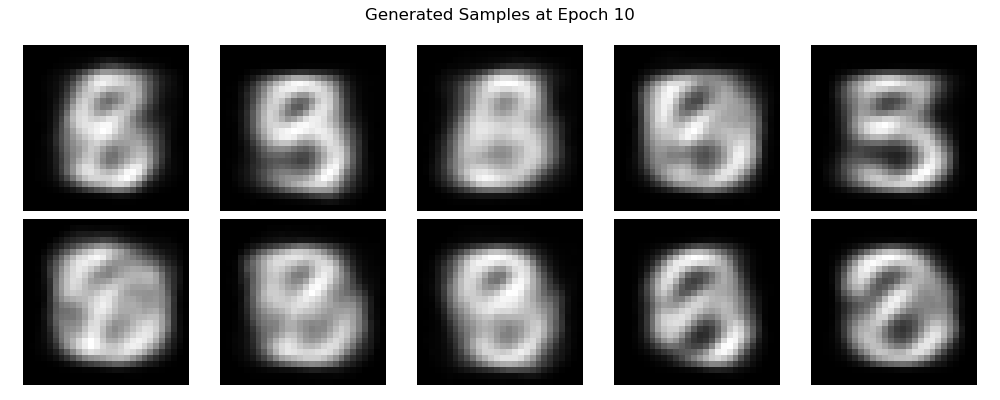
Epoch 10/10:

Wake Loss: 0.2778, Sleep Loss: 0.6841, Class Loss: 0.0854, Train Acc: 97.32%

Test Loss: 0.0734, Test Acc: 97.70%

10 random latent codes of images from 5th and 10th epoch:

Pasted here :

Tada

Ok I like number 9 not 5 or 8.