

DEEP LEARNING

LAB ASSIGNMENT

AI&ML BATCH- 4 (NH)

PART – B

Assignment 6: Synthetic Data Generation for Fashion items with GAN’s.

Submitted By- Submitted To-

Name – Harsh Bhartia Mr. Mausam Chouksey

SAP ID – 500106274

Roll No. – R2142221221

**About Dataset**

The **Fashion-MNIST dataset** is a widely used benchmark dataset for machine learning and deep learning models, particularly in image classification and generation tasks. It consists of 70,000 grayscale images, each of size 28x28 pixels, representing 10 different classes of fashion items such as T-shirts, trousers, dresses, and shoes. The dataset is divided into 60,000 training images and 10,000 test images.

Fashion-MNIST serves as a more challenging alternative to the classic MNIST dataset of handwritten digits, offering a greater variety of complex patterns and textures. The images are labelled and pre-processed, making the dataset suitable for rapid prototyping and experimentation. Its small size and visual diversity make it ideal for tasks such as generative modelling, as demonstrated in this project.

**Objective**

The objective of this assignment is to design and train a Generative Adversarial Network (GAN) to generate synthetic images of fashion items resembling those in the Fashion-MNIST dataset. This involves developing and fine-tuning a generator and discriminator to create realistic images while distinguishing them from actual data, ultimately evaluating the GAN's ability to mimic the characteristics of the original dataset.

1. Train a Generative Adversarial Network (GAN) to generate realistic fashion item images.

2. Utilize the Fashion-MNIST dataset as the training source.

3. Develop a generator to produce synthetic images and a discriminator to evaluate them.

4. Fine-tune the model for better realism in generated outputs.

**Data Preprocessing**

The data preprocessing for the Fashion-MNIST dataset in this assignment involved the following steps to ensure the data is suitable for training a Generative Adversarial Network (GAN):

1. Loading the Dataset:
   * The Fashion-MNIST dataset was loaded using keras.datasets.fashion\_mnist.
   * The dataset contains 60,000 training images and 10,000 testing images, each of size 28x28 pixels and labelled with one of 10 fashion item classes.
2. Normalizing Pixel Values:
   * The pixel values, originally in the range of 0–255, were normalized to a range of 0–1 by dividing by 255.
   * This step ensures faster convergence during training by standardizing the input data scale.
3. Reshaping the Images:
   * The images were reshaped to include a channel dimension, converting them from (28, 28) to (28, 28, 1) to match the input requirements of the GAN's convolutional layers.
   * The shape (28, 28, 1) represents a single grayscale channel for each image.
4. Data Type Conversion:
   * The image data was converted to the float32 data type to ensure compatibility with TensorFlow's backend and the GAN's architecture.
5. Batching and Shuffling:
   * The pre-processed training data was divided into batches using the specified batch-size.
   * The data was shuffled to ensure the model sees diverse examples during each epoch, preventing overfitting to specific patterns.
6. Latent Space Preparation:
   * A latent space was defined with a dimensionality of 100, used to sample random noise vectors for the generator. These noise vectors serve as the input for generating synthetic images.

This preprocessing pipeline ensured that the dataset was clean, properly scaled, and efficiently formatted for training the GAN model. It played a crucial role in stabilizing the training process and improving the quality of the generated images.

**Model Architecture**

The Generative Adversarial Network (GAN) used in this assignment consists of two main components: the **Generator** and the **Discriminator**, each with its specific architecture designed for generating and evaluating fashion item images.

**1. Generator Architecture**

The **Generator** is responsible for transforming random noise vectors from the latent space into realistic images resembling those in the Fashion-MNIST dataset.

**Input**: A random noise vector of dimension 100 from the latent space.  
**Architecture**:

**Dense Layer**: Fully connected layer with 7x7x256 units, reshaped into a 3D tensor to serve as the starting point for deconvolution.

*Activation*: Leaky ReLU.

*Output Shape*: (7, 7, 256).

**Batch Normalization**: Ensures stable training by normalizing activations and speeding up convergence.

**Transpose Convolution Layers**:

Multiple Conv2DTranspose layers progressively upsample the feature maps.

*Filters*: 128 → 64 → 1 (grayscale).

*Kernel Size*: 3x3 for all layers.

*Strides*: 2x2 for upsampling.

*Padding*: "same" to maintain spatial dimensions.

*Activation*: Leaky ReLU for hidden layers, Tanh for the output layer (to scale outputs to [-1, 1]).  
**Output**: A 28x28x1 grayscale image.

**2. Discriminator Architecture**

The **Discriminator** evaluates the authenticity of an input image, distinguishing between real images from the dataset and synthetic images generated by the Generator.

**Input**: A 28x28x1 grayscale image (real or generated).  
**Architecture**:

**Convolutional Layers**:

Multiple Conv2D layers downsample the input.

*Filters*: 64 → 128.

*Kernel Size*: 3x3 for all layers.

*Strides*: 2x2 for downsampling.

*Padding*: "same" to preserve spatial information.

*Activation*: Leaky ReLU for all layers to prevent dying neurons.

**Dropout Layers**:

Dropout of 0.3 is applied after each convolutional layer to reduce overfitting.

**Flatten Layer**: Converts the 3D feature maps into a 1D vector.

**Dense Layer**: Fully connected layer with 1 output neuron.

*Activation*: Sigmoid to output a probability score (real or fake).

**Output**: A single scalar value (0: fake, 1: real).

**3. GAN (Combined Model) Architecture**

The **GAN** combines the Generator and the Discriminator to train the Generator to produce realistic images.

The **Generator** takes random noise as input and generates synthetic images.

The **Discriminator** evaluates these images but is kept **non-trainable** during the GAN's training process to ensure only the Generator is updated.

**Input**: Random noise vector.  
**Output**: Probability score indicating how real the generated image is.

**Model Summary**

**Generator**:

Total Parameters: ~1,177,345.

**Discriminator**:

Total Parameters: ~211,393.

**GAN**:

Combines the above components and trains them jointly.

This architecture ensures the GAN effectively learns the data distribution of Fashion-MNIST and generates realistic synthetic fashion images.

**Training Details**

The training process of the GAN involves alternating updates to the **Generator** and **Discriminator** to achieve a balance between the two models. The details of the training procedure used in the assignment are as follows:

**1. Training Strategy**

* The GAN was trained using the **adversarial learning approach**, where:
  + The **Generator** aims to create realistic images that can fool the Discriminator.
  + The **Discriminator** works to distinguish between real and fake images.
* The models were updated iteratively to improve both their performances simultaneously.

**2. Training Dataset**

* **Dataset**: Fashion-MNIST, containing 60,000 grayscale images for training.
* **Preprocessing**: Images were normalized to the range [0, 1] and reshaped to include a channel dimension (28, 28, 1).
* **Latent Space**: A 100-dimensional latent space was defined, where random noise vectors were sampled as input for the Generator.

**3. Training Process**

The training loop alternates between updating the Discriminator and Generator:

1. **Training the Discriminator**:
   * A batch of real images was sampled from the dataset.
   * A batch of fake images was generated by passing random noise through the Generator.
   * The Discriminator was trained on both real and fake images:
     + **Real images** were labeled as 1.
     + **Fake images** were labeled as 0.
   * Loss function: Binary Crossentropy Loss.
   * Goal: Maximize accuracy in distinguishing real from fake images.
2. **Training the Generator**:
   * Random noise was passed through the Generator to create fake images.
   * The Discriminator’s feedback on these fake images was used to compute the Generator’s loss.
   * Loss function: Binary Crossentropy Loss (with flipped labels, aiming for a label of 1 to "fool" the Discriminator).
   * Goal: Minimize the Discriminator's ability to distinguish fake images from real ones.
3. **Iterative Updates**:
   * Both models were updated iteratively using the Adam optimizer with:
     + Learning Rate: 0.0002.
     + Beta1: 0.5.

**4. Key Parameters**

* **Epochs**: 100.
* **Batch Size**: 64.
* **Save Interval**: Every 500 epochs, generated images were saved to track progress.

**5. Metrics Monitored**

* **Discriminator Loss**: Evaluates how well the Discriminator distinguishes between real and fake images.
* **Discriminator Accuracy**: Indicates the percentage of correct classifications made by the Discriminator.
* **Generator Loss**: Evaluates how well the Generator creates images that can fool the Discriminator.

**6. Outputs Generated During Training**

* **Synthetic Images**:
  + Images were generated at different stages of training to visualize the improvement in quality.
  + These outputs showed the Generator's progression from producing random noise-like patterns to realistic fashion items.
* **Loss and Accuracy Values**:
  + Displayed for both models to monitor convergence and stability.

**7. Hardware and Runtime**

* **Hardware Used**: GPU-enabled environment for accelerated training.
* **Runtime**: Approximately 10–15 minutes for 100 epochs, depending on the computational power available.

**Observations**

* Early in training, generated images appeared noisy and unrealistic.
* As training progressed, the Generator produced increasingly realistic images resembling the Fashion-MNIST items.
* Balancing the Generator and Discriminator was crucial to prevent mode collapse or overfitting.

This training process effectively leveraged adversarial learning to produce high-quality synthetic images while maintaining stable training dynamics.

**Explanation of Libraries Used**

The final code for the assignment leverages several powerful libraries from the Python ecosystem, each serving specific roles in implementing and training the Generative Adversarial Network (GAN). Below is a detailed explanation of the libraries used:

**1. TensorFlow/Keras**

* **Purpose**: TensorFlow is a deep learning framework, and Keras (a high-level API within TensorFlow) simplifies the design and training of neural networks.
* **Key Functions and Modules Used**:
  + **tensorflow.keras.models**: To build the Sequential models for the Generator and Discriminator.
  + **tensorflow.keras.layers**:
    - **Dense**: Fully connected layers used in both Generator and Discriminator.
    - **Conv2D and Conv2DTranspose**: Used for convolutional and transposed convolutional operations, essential for downsampling and upsampling image data.
    - **LeakyReLU**: Activation function used to prevent dying neurons in GAN models.
    - **BatchNormalization**: Normalizes activations to speed up convergence and stabilize training.
    - **Flatten and Reshape**: To reshape data between layers as required.
    - **Dropout**: Regularization to prevent overfitting in the Discriminator.
    - **Activation**: To specify activations like Sigmoid and Tanh in layers.
  + **tensorflow.keras.optimizers**: Adam optimizer, used to train both Generator and Discriminator with stable convergence.

**2. NumPy**

* **Purpose**: NumPy is used for efficient numerical computations and data manipulation.
* **Key Functions**:
  + **numpy.random.normal**: To generate random noise vectors from a normal distribution, which are used as input to the Generator.
  + **numpy.zeros and numpy.ones**: To create labels (real or fake) for training the Discriminator.

**3. Matplotlib**

* **Purpose**: Matplotlib is used for visualizing the results of GAN training, including generated images.
* **Key Functions**:
  + **matplotlib.pyplot**: Provides functions for creating plots and displaying generated images at various training stages.
  + **imshow**: Displays images, such as real and generated images, in a grid format.

**4. Fashion-MNIST Dataset via TensorFlow/Keras**

* **Purpose**: The dataset is included in the tensorflow.keras.datasets module, eliminating the need for external dataset downloads.
* **Key Features**:
  + Provides pre-labeled grayscale images of fashion items, ideal for generative tasks.
  + Built-in functionality to load and split the dataset into training and testing sets.

**Additional Details**

* **Integration of Libraries**:
  + TensorFlow/Keras served as the core library for building and training the GAN.
  + NumPy supplemented TensorFlow by handling random noise generation and array manipulations.
  + Matplotlib ensured the results were visually represented, making it easy to evaluate the GAN's progress.

By combining these libraries, the code achieves a seamless workflow for training, evaluating, and visualizing the GAN for fashion item generation. Each library plays a distinct role, contributing to the success of the assignment.