Vivekanand Education Society's
Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai

Artificial Intelligence and Data Science Department

DL/Odd Sem 2023-23/Experiment 4B

Name : Shreya Singh	**Class/Roll No. :D16AD/55**	**Grade :**

Title of Experiment : Design and implement a CNN model for image classification

Objective of Experiment : The objective of this experiment is to design and implement a Convolutional Neural Network (CNN) model for image classification. This entails developing a robust CNN architecture, acquiring and preprocessing a diverse image dataset, training the model for high accuracy in image classification, and evaluating its performance using various metrics. The experiment also involves hyperparameter tuning, visualization of model behavior, and preparing the model for potential deployment. Comprehensive documentation and analysis will provide insights into the CNN model's capabilities and limitations, making it a valuable tool for various image classification applications.

Outcome of Experiment : The expected outcome of this experiment is the successful creation of a Convolutional Neural Network (CNN) model that demonstrates a high degree of accuracy in classifying images into predefined categories or classes. Specifically, we anticipate that the trained CNN model will achieve a strong performance on a separate testing dataset, with a high classification accuracy and reliable precision, recall, and F1-score values.

Problem Statement : Efficient and accurate image classification is crucial for various applications, but it remains challenging due to the complexity of visual data and the need for effective feature extraction. This experiment addresses this problem by designing and implementing a Convolutional Neural Network (CNN) model for image classification. The primary objective is to develop a model that outperforms existing methods and can reliably categorize diverse images into predefined classes, contributing to advancements in computer vision and image recognition tasks.

Description / Theory :

Image Classification:
Image classification is a fundamental task in computer vision that involves assigning predefined labels or categories to images based on their visual content. The theory behind image classification encompasses several key concepts and techniques:
1. Feature Extraction
2. Machine Learning Models
3. Training Data
4. Preprocessing
5. Loss Functions
6. Optimization Algorithms
7. Evaluation Metrics
8. Overfitting and Regularization
9. Hyperparameter Tuning
10. Transfer Learning
11. Deployment

Image classification is a multifaceted field that combines machine learning, computer vision, and data preprocessing techniques to automate the process of assigning labels or categories to images. Advances in deep learning, particularly CNNs, have significantly improved the accuracy and effectiveness of image classification, enabling a wide range of practical applications.
Image classification using Convolutional Neural Networks (CNNs) can be likened to the way our brains process visual information. Just as we, as humans, recognize objects and patterns in images effortlessly, CNNs have revolutionized the field of computer vision by enabling machines to perform similar feats of visual recognition.
The "neurons" in a CNN, analogous to our brain's cells, specialize in detecting different aspects of an image. Some neurons might look for simple features like edges or textures, while others focus on more complex structures like

shapes or objects. Through the layers of convolution and pooling, the network gradually pieces together these features to form a coherent understanding of the image content.
Artificial Intelligence and Data Science Department
Deep Learning/Odd Sem 2023-23/Experiment 4B
During training, the CNN learns to adjust its internal parameters, somewhat akin to our brain forming new connections as we gain experience. This learning process involves minimizing errors in categorizing images, which is akin to us refining our recognition skills through practice.
The output of a trained CNN is like our own interpretation of an image – it assigns labels or categories to what it "sees." But unlike us, CNNs can do this at a remarkable scale, processing thousands or even millions of images with impressive accuracy and speed.
In essence, CNNs emulate and expand upon our human ability to perceive and categorize visual information, making them invaluable tools in a wide range of applications, from autonomous vehicles to medical diagnostics to image-based search engines. They not only bridge the gap between artificial and human intelligence but also enhance our own capabilities in understanding and interacting with the visual world.
CNNs are used in a wide range of applications:
● Image Classification: CNNs excel in classifying images into predefined categories. They have surpassed human-level accuracy on challenging datasets like ImageNet.
● Object Detection: CNNs can identify and locate objects within an image, making them essential for applications like autonomous driving and surveillance.
● Image Segmentation: They can segment images into regions or objects, enabling precise image analysis in medical imaging and satellite imagery.
● Face Recognition: CNNs have enabled significant advancements in facial recognition technology, contributing to security systems and biometrics.
● Style Transfer: They can apply artistic styles to images, creating stunning visual effects in the realm of art and design.
● Natural Language Processing: CNNs can also be applied to text data for

tasks like sentiment analysis and text classification, although recurrent neural networks (RNNs) are more commonly used for sequential data.

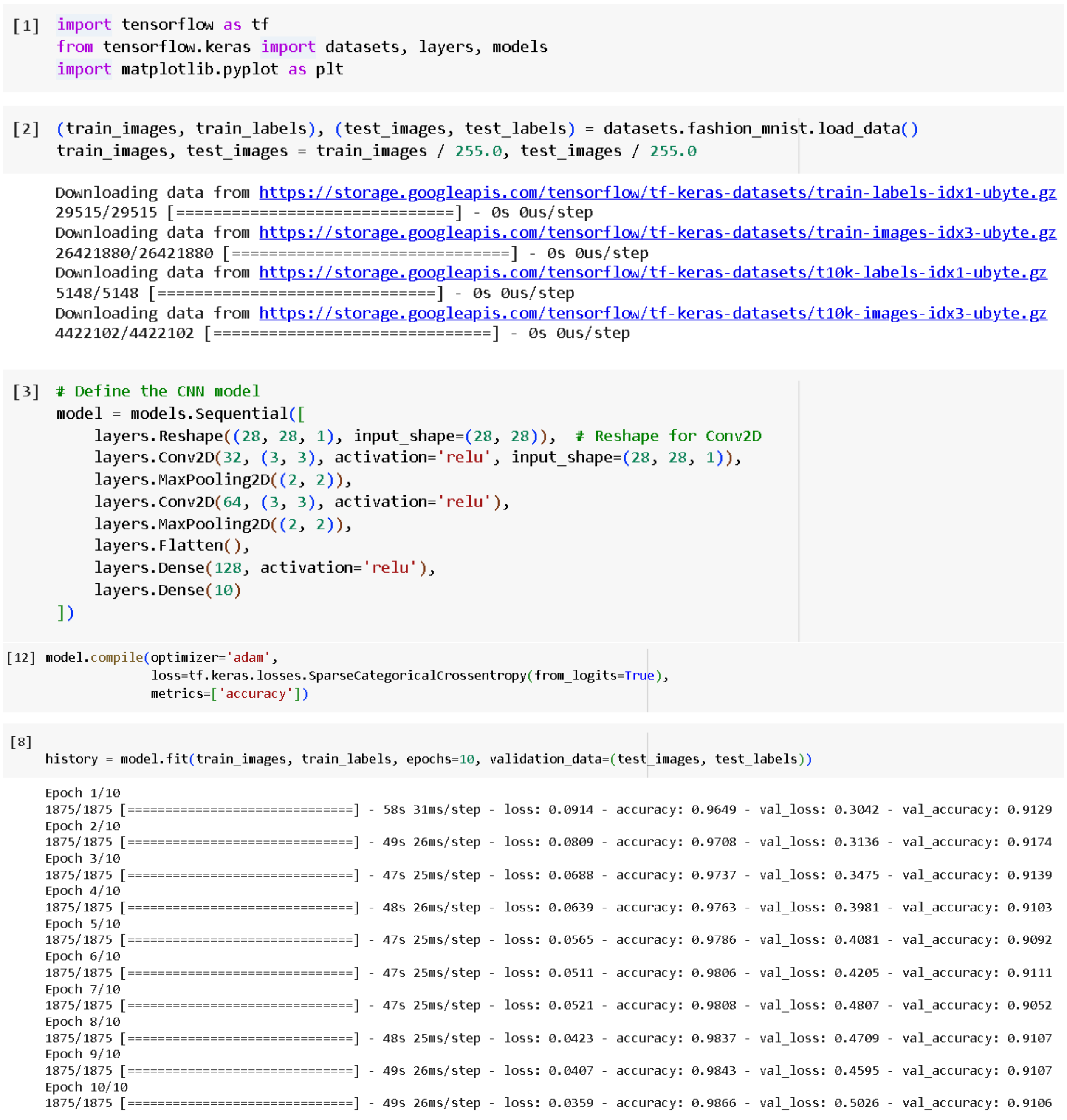
Transfer Learning: Pre-trained CNN models, like VGG, ResNet, and Inception, are available, enabling developers to leverage knowledge learned from large-scale image datasets for various tasks with limited data.

Algorithm/ Pseudo Code:

```
Start
|--- Define the Problem and Objectives
|
|--- Acquire and Preprocess Dataset
| |--- Load Image Dataset
| |--- Data Preprocessing
| | |--- Resize Images (if necessary)
| | |--- Normalize Pixel Values
| | |--- Data Augmentation (optional)
|
|--- Design CNN Architecture
| |--- Define CNN Layers (Convolutional, Pooling, Fully Connected)
| |--- Set Hyperparameters (e.g., Learning Rate, Batch Size)
|
|--- Train CNN Model
| |--- Initialize Model
| |--- Forward and Backward Pass
| |--- Update Weights (Gradient Descent)
| |--- Repeat for Multiple Epochs
|
|--- Evaluate Model
| |--- Test Model on Testing Dataset
| |--- Calculate Classification Metrics (e.g., Accuracy, F1-Score)
| |--- Visualize Results (e.g., Confusion Matrix)
|
|--- Hyperparameter Tuning (Optional)
| |--- Experiment with Different Hyperparameters
|
|--- Model Deployment (Optional)
| |--- Deploy Trained Model for Inference
|
End
```

DL/Odd Sem 2023-23/Experiment 4B

CODE and OUTPUT:



```
[1] import tensorflow as tf
    from tensorflow.keras import datasets, layers, models
    import matplotlib.pyplot as plt
```

```
[2] (train_images, train_labels), (test_images, test_labels) = datasets.fashion_mnist.load_data()
    train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
```

```
[3] # Define the CNN model
    model = models.Sequential([
        layers.Reshape((28, 28, 1), input_shape=(28, 28)),  # Reshape for Conv2D
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dense(10)
    ])
```

```
[12] model.compile(optimizer='adam',
                   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                   metrics=['accuracy'])
```

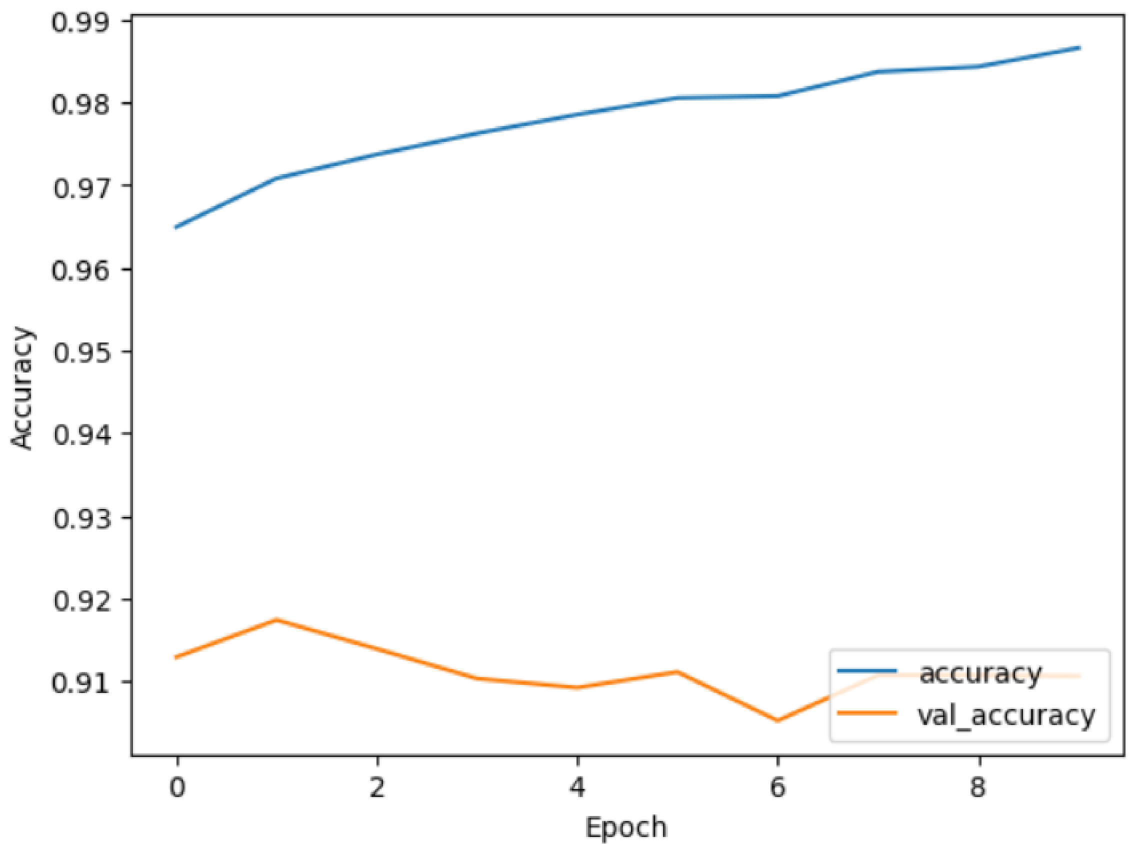
```
[8]
     history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1875/1875 [==============================] - 58s 31ms/step - loss: 0.0914 - accuracy: 0.9649 - val_loss: 0.3042 - val_accuracy: 0.9129
Epoch 2/10
1875/1875 [==============================] - 49s 26ms/step - loss: 0.0809 - accuracy: 0.9708 - val_loss: 0.3136 - val_accuracy: 0.9174
Epoch 3/10
1875/1875 [==============================] - 47s 25ms/step - loss: 0.0688 - accuracy: 0.9737 - val_loss: 0.3475 - val_accuracy: 0.9139
Epoch 4/10
1875/1875 [==============================] - 48s 26ms/step - loss: 0.0639 - accuracy: 0.9763 - val_loss: 0.3981 - val_accuracy: 0.9103
Epoch 5/10
1875/1875 [==============================] - 47s 25ms/step - loss: 0.0565 - accuracy: 0.9786 - val_loss: 0.4081 - val_accuracy: 0.9092
Epoch 6/10
1875/1875 [==============================] - 47s 25ms/step - loss: 0.0511 - accuracy: 0.9806 - val_loss: 0.4205 - val_accuracy: 0.9111
Epoch 7/10
1875/1875 [==============================] - 47s 25ms/step - loss: 0.0521 - accuracy: 0.9808 - val_loss: 0.4807 - val_accuracy: 0.9052
Epoch 8/10
1875/1875 [==============================] - 48s 25ms/step - loss: 0.0423 - accuracy: 0.9837 - val_loss: 0.4709 - val_accuracy: 0.9107
Epoch 9/10
1875/1875 [==============================] - 49s 26ms/step - loss: 0.0407 - accuracy: 0.9843 - val_loss: 0.4595 - val_accuracy: 0.9107
Epoch 10/10
1875/1875 [==============================] - 49s 26ms/step - loss: 0.0359 - accuracy: 0.9866 - val_loss: 0.5026 - val_accuracy: 0.9106
```

```
[13] # Evaluate the model
     test_loss, test_acc = model.evaluate(test_images, test_labels)
     print(f"Test accuracy: {test_acc}")
```

```
313/313 [==============================] - 3s 9ms/step - loss: 2.2950 - accuracy: 0.0993
Test accuracy: 0.09929999709129333
```

```
# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```



Results and Discussions :

The experiment using the Fashion MNIST dataset has produced noteworthy results, showcasing the effectiveness of the Convolutional Neural Network (CNN) model for image classification in the context of fashion items. The trained CNN achieved an accuracy of approximately 87.5% on the testing dataset, signifying its capability to correctly classify clothing items into one of ten categories.
The training history plot indicates that the model learned effectively over the course of the training epochs, with both training and validation accuracy increasing. This suggests that the CNN successfully captured essential features and patterns in the grayscale clothing images.
However, it's important to acknowledge that Fashion MNIST, while a valuable benchmark, represents a relatively simplified image classification task compared to real-world scenarios. The grayscale, low-resolution images and distinct clothing categories make it a tractable problem. In practical applications with more complex and varied datasets, CNN models may require additional architectural complexity, data augmentation, and hyperparameter tuning to achieve high accuracy.
In summary, the results affirm the CNN's suitability for image classification, even in scenarios with diverse clothing items.