Assignment 1: - Study of Deep Learning Packages: Tensorflow, Keras, Theano and PyTorch. Document the distinctfeatures and functionality of the packages.

Objective: comparison of four popular deep learning packages: TensorFlow, Keras, Theano, and PyTorch.

Theory:

Deep learning has emerged as a powerful tool for solving complex problems in various domains, such as image recognition, natural language processing, and autonomous driving. To harness the capabilities of deep learning effectively, researchers rely on deep learning frameworks or packages. In this study, we explore four widely used deep learning packages: TensorFlow, Keras, Theano, and PyTorch.

2. TensorFlow:

Distinct Features:

- Developed by Google Brain, TensorFlow is an open-source machine learning framework known for its flexibility and scalability.
- Offers both high-level and low-level APIs for building and training neural networks.
- TensorFlow provides a visualization toolkit called TensorBoard for model visualization and debugging.

3. Keras:

Distinct Features:

- Keras is an open-source, high-level neural networks API that runs on top of TensorFlow, Theano, or CNTK.
- Known for its user-friendliness and ease of use.
- Enables rapid prototyping of deep learning models through a simple, modular interface.
- It provides a simple API for defining and training neural networks.

4. Theano:

Distinct Features:

- Theano, although less popular today, was a explorer/discoverer in deep learning and symbolic mathematics.
- Known for its efficiency in optimizing mathematical expressions for GPU execution.
- Allowed users to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.

- It provided a foundation for early deep learning research but is no longer actively maintained.

5. PyTorch:

Distinct Features:

- PyTorch is an open-source deep learning framework developed by Facebook's AI Research lab (FAIR).
- Known for its dynamic computation graph, which offers flexibility and ease of debugging.
- Provides native support for automatic differentiation.

- PyTorch is highly suitable for natural language processing, computer vision, and reinforcement learning tasks.

Assignment 3: - Build the Image classification model by dividing the model into the following four stages:
a. Loading and preprocessing the image data
b. Defining the model's architecture
c. Training the model
d. Estimating the model's performance

Objective: The objective is to build an image classification model in a step-by-step manner by dividing the process into four stages as given

Software Requirements:

1. Python
2. TensorFlow
3. A image dataset

Hardware Requirements: A computer with a CPU or GPU.

Theory:

Image Classification: Image classification is a computer vision task in which an algorithm assigns a label or category to an input image. It is a fundamental problem in machine learning and is widely used in applications such as object recognition, face detection, and more.

Implementation Stages:

a. **Loading and Preprocessing the Image Data:**

- In this stage, we load the dataset containing images and preprocess them for training. Common preprocessing steps include resizing images to a consistent size, and splitting data into training and testing sets.

b. **Defining the Model's Architecture:**

- The architecture of a deep learning model determines how it processes input data. For image classification, Convolutional Neural Networks (CNNs) are typically used. Participants will define the layers, activations, and other model components in this stage.

c. **Training the Model:**

- Training a model involves feeding it with labelled training data, optimizing its weights using an optimization algorithm (e.g., Stochastic Gradient Descent), and iteratively improving its performance. Participants will choose hyperparameters such as learning rate, batch size, and the number of training epochs.

d. **Estimating the Model's Performance:**

- After training, it's essential to assess the model's performance. Participants will evaluate metrics such as accuracy, precision, recall, and F1-score to understand how well the model classifies unseen images. This step helps in determining if the model is suitable for practical use.

Assignment 4: - Use Autoencoder to implement anomaly detection. Build the model by using the following:
a. Import required libraries
b. Upload/access the dataset
c. The encoder converts it into a latent representation
d. Decoder networks convert it back to the original input
e. Compile the models with Optimizer, Loss, and Evaluation Metrics

Objective: The objective of this practical lab is to implement anomaly detection using autoencoders, a type of neural network architecture.

Software Requirements:

1. Python
2. TensorFlow,NumPy
3. Jupyter Notebook

Hardware Requirements: A computer with a CPU or GPU.

Theory:

Autoencoder: An autoencoder is a type of artificial neural network that is used for dimensionality reduction, feature learning, and, in this case, anomaly detection. Anomalies in data are often detected by comparing the reconstruction error.

Implementation Steps:

a. **Import Required Libraries:**

- Import libraries such as TensorFlow, NumPy for numerical operations, and Matplotlib for visualization.

b. **Upload/Access the Dataset:**

- Load the dataset that contains both normal and potentially anomalous data. This dataset should represent the type of data you want to perform anomaly detection on.

c. **The Encoder Converts it into a Latent Representation:**

- Define the architecture of the encoder, which takes input data and maps it to a lower-dimensional latent representation. The encoder typically consists of layers that reduce the dimensionality of the data.

d. **Decoder Networks Convert it Back to the Original Input:**

- Design the decoder network, which takes the latent representation and attempts to reconstruct the original input data. The decoder architecture should mirror the encoder's architecture in reverse.

e. **Compile the Models with Optimizer, Loss, and Evaluation Metrics:**

- Compile both the encoder and decoder models separately.
- Choose an optimizer (e.g., Adam or SGD), a suitable loss function (e.g., mean squared error for reconstruction), and evaluation metrics (e.g., accuracy or area under the ROC curve for anomaly detection).

Assignment 5: - Implement the Continuous Bag of Words (CBOW) Model. Stages can be:
a. Data preparation
b. Generate training data
c. Train model
d. Output

Objective: The objective of this practical lab is to implement the Continuous Bag of Words (CBOW) model.

Software Requirements:

1. Python
2. Numpy
3. TensorFlow
4. Jupyter Notebook

Hardware Requirements: A computer with a CPU is sufficient for this practical lab.

Theory:

Continuous Bag of Words (CBOW): CBOW is a neural network-based model used for word embeddings in NLP tasks. CBOW treats the context words as input and the target word as the output. By training CBOW on a large corpus of text, it learns to represent words in a continuous vector space where words with similar meanings have similar vector representations.

Implementation Stages:

a. **Data Preparation:**

- Load and preprocess a dataset containing text corpus. Tokenize the text into words, remove punctuation, and create a vocabulary of unique words.

b. **Generate Training Data:**

- Create training data for CBOW by specifying a context window size. For each target word in the corpus, select the surrounding context words within the window as input and the target word as the output.

c. **Train Model:**

- Design and train a neural network model for CBOW. The model typically consists of an embedding layer to convert words into vector representations, followed by layers like LSTM or dense layers. The training objective is to minimize the loss between the predicted word and the actual target word.

d. **Output:**

- After training, the embeddings learned by the CBOW model can be extracted for use in various NLP tasks. These embeddings represent words in a continuous vector space, capturing semantic relationships between words.

Assignment 6: - Object detection using Transfer Learning of CNN architectures
a. Load in a pre-trained CNN model trained on a large dataset
b. Freeze parameters(weights) in the model's lower convolutional layers
c. Add a custom classifier with several layers of trainable parameters to model
d. Train classifier layers on training data available for the task
e. Fine-tune hyperparameters and unfreeze more layers as needed

Objective: The objective of this practical lab is to implement object detection using transfer learning with Convolutional Neural Networks (CNNs).

Software Requirements:

1. Python
2. Deep Learning Framework
3. A pre-trained CNN model0
4. Jupyter Notebook

Hardware Requirements: A computer with a CPU and GPU.

Theory:

Transfer Learning: Transfer learning is a technique in deep learning where a pre-trained model, trained on a large dataset (e.g., ImageNet), is used as a starting point for a new task. In object detection, this approach can save time and computational resources because lower layers of the pre-trained model have already learned useful features like edges, textures, and object parts.

Implementation Stages:

a. **Load in a Pre-trained CNN Model:**

- Load a pre-trained CNN model, ResNet, or MobileNet, which has learned generic features from a large dataset.

b. **Freeze Parameters (Weights) in the Model's Lower Convolutional Layers:**

- Freeze the lower layers of the pre-trained model to prevent them from being updated during training.

c. **Train Classifier Layers on Training Data:**

- Train the custom classifier layers on task-specific data (object detection dataset). This fine-tunes the model to recognize objects relevant to the specific task.

d. **Fine-tune Hyperparameters and Unfreeze More Layers as Needed:**

- Experiment with hyperparameters such as learning rate, batch size, and optimizer.

Assignment 2:- Implementing Feed-forward neural networks with Keras and TensorFlow
a. Import the necessary packages
b. Load the training and testing data (MNIST/CIFAR10)
c. Define the network architecture using Keras
d. Train the model using SGD
e. Evaluate the network
f. Plot the training loss and accuracy

Title: Implementing Feed-Forward Neural Networks with Keras and TensorFlow

Objective: The objective of this practical lab is to implement a feed-forward neural network using Keras and TensorFlow for a classification task.

Software Requirements:

1. Python
2. TensorFlow (Deep Learning Framework)
3. Keras (Deep Learning API)
4. Jupyter Notebook (optional but recommended)

Hardware Requirements: A computer with a CPU is sufficient for this practical lab. A GPU is recommended for faster training but not mandatory.

Theory:

Feed-Forward Neural Networks (FFNN): Feed-forward neural networks, also known as multilayer perceptrons (MLPs), are a fundamental type of artificial neural network. They consist of an input layer, one or more hidden layers, and an output layer. FFNNs are widely used for various tasks, including image classification and regression.

Implementation Stages:

a. **Import the Necessary Packages:**

- Import essential libraries and packages, including TensorFlow and Keras, to facilitate neural network implementation.

b. **Load the Training and Testing Data (MNIST/CIFAR-10):**

- Load and preprocess the training and testing datasets, such as MNIST or CIFAR-10, for the chosen classification task.

c. **Define the Network Architecture Using Keras:**

- Design the neural network architecture using Keras. Define the number of layers, activation functions, and the number of neurons in each layer.

d. **Train the Model Using SGD (Stochastic Gradient Descent):**

- Compile and train the neural network model on the training data using stochastic gradient descent (SGD) as the optimization algorithm. Specify a loss function, metrics, and training parameters like batch size and epochs.

e. **Evaluate the Network:**

- Evaluate the trained model on the testing data to assess its performance. Measure metrics like accuracy, precision, recall, and F1-score, depending on the classification task.

f. **Plot the Training Loss and Accuracy:**

- Visualize the training loss and accuracy over epochs using plots or graphs. This helps in understanding how the model learns and whether it converges or overfits.