|  |
| --- |
|  |
| 1. a) Develop a code for simple vector addition, subtraction, multiplication, and division using TensorFlow.   b) Explain Deep Learning Life cycle and Performance metrics.  **Ex.No. 1.a) Develop a code for simple vector addition, subtraction, multiplication, and division using TensorFlow.**  **Aim:** To learn the implementation of simple vector addition in TensorFlow.  **Requirements:**  Software: Python with Tensorflow Library, Numpy library, data modeling library Scikit learn.  Hardware: Modern Operating System,x86 64-bit CPU (Intel / AMD architecture),4 GB RAM,  5 GB free disk space.  **Suggested Reading(Theoretical Background):**  Tensorflow is an effective open-source framework for numerical computation and machine learning is called TensorFlow. It functions using a computational graph that shows the arrangement of data (edges) and processes (nodes). An outline of a typical TensorFlow workflow follows the Creation of Tensors, Definition of Operations, Initialization Variables, Execution the Graph, Training a Model, and Evaluation the of Model.  The operations to be done on the tensor flow are Addition, Subtraction, Multiplication , Division, Matrix Multiplication,Transpose, Reshape, Concatenate, Reduce Sum, Reduce Mean, Reduce Max,Argmax, Greater,Logical AND, Sigmoid Activation, ReLUActivation,SoftmaxActivation,Standard Deviation ,Variance ,Cast ,Slicing ,Tile , Expand Dimensions, Squeeze , One Hot Encoding. Here , only simple vector addition is implemented.  **Algorithm:**   1. Start the process 2. Import Library : Import TensorFlow to utilize its functions and operations. 3. Define Vectors: Create constant tensors to represent the vectors to be added. Also check the dimension of the vector inputed. 4. Add Vectors: Use TensorFlow's add function to perform element-wise addition of the vectors. (Here other operations such as multiplication, division, transpose etc. can also performed). 5. Run the Computation: Execute the addition operation and convert the result to a NumPy array for easy interpretation. 6. Display the result. 7. Stop the process   **Sample Output/Result:**  Input: [3 6]  [9, 12]  Output [12 18]  The implementation of vector addition using TensorFlow is done.  **Inferences:**  This experiment demonstrates how to perform basic vector addition using tensor flow.  **Viva Questions:**   1. What is the functionof numpy function in tensorflow? 2. What is the significance of the tensorflow in this program? 3. Which method do you use in this program to add matrix elements elementwise ? 4. Which are the different functions to perform mathematical operations? 5. List the different functions available in tensorflow.   **Ex. No. 1.b) Explain Deep Learning Life cycle and Performance metrics.**  **Aim:** To study of Deep Learning life cycle and its Performance metrics  **Requirements:**  **Software**: Python with Tensorflow Library, Numpy library, data modeling library Scikit learn.  **Hardware:** Modern Operating System,x86 64-bit CPU (Intel / AMD architecture),4 GB RAM,  5 GB free disk space.  **Suggested Reading(Theoretical Background):**  Deep Learning is a vast domain with a variety of tools and technologies. The toolset is evolving rapidly. Multiple courses exist for concepts and implementations of deep learning including concepts and implementations of deep learning including concepts, libraries and tools, implementations.  One of the key components of deep learning is the math involved in it. Some topics cover this math in depth and some ignore them as the tool takes care of its implementation. The same applies to various tools used for deep learning as they take care of the implementation of the algorithms and techniques. Students are expected to be perfect with machine learning and techniques. Hands-on experience is also preferred with Python programming. Suggest to get the knowledge of Keras and TensorFlow, sci-kit-learn, text processing with NLTK libraries  **Life cycle of Deep Learning** **1. Problem Definition**  * Problem Statement: Define the problem statement to be solved. * Define objectives: identify the goals and success critira * Identify constraints: Figure out any constraints, including those related to time, computational power, and data accessibility.  **2. Data Collection and Preparation**  * Data Acquisition: Gather the data needed for the project from various sources like databases, APIs, web scraping, or predefined datasets. * Data Exploration: Analyze the data to understand its characteristics and identify patterns. This includes summary statistics, visualization, and anomaly detection. * Data Cleaning: Handle missing values, remove duplicates, and correct errors. * Data Transformation: Normalize or standardize data, create new features, and encode categorical variables. * Data Splitting: Split the dataset into training, validation, and test sets.  **3. Model Building**  * Model Selection: Choose the appropriate model architecture based on the problem type (e.g., CNNs for image data, RNNs for sequence data). * Define Model Architecture: Design the network layers, activation functions, and how layers are connected. * Initialization: Initialize model parameters (weights and biases).  **4. Model Training**  * Compilation: Compile the model by specifying the loss function, optimizer, and evaluation metrics. * Training: Train the model on the training dataset. This involves:   + Forward Propagation: Compute predictions.   + Loss Calculation: Measure the error between predictions and true values.   + Backward Propagation: Update model weights using gradients to minimize the loss. * Hyperparameter Tuning: Adjust hyperparameters (learning rate, batch size, number of epochs) to optimize performance. * Validation: Use the validation set to monitor the model’s performance during training and prevent overfitting.  **5. Model Evaluation**  * Performance Metrics: Evaluate the model on the test set using relevant metrics (accuracy, precision, recall, F1 score, etc.). * Error Analysis: Analyze model errors to understand failure cases and improve the model.  **6. Model Deployment**  * Model Export: Save the trained model in a suitable format (e.g., TensorFlow SavedModel, ONNX). * Integration: Integrate the model into the application or system where it will be used. * Scalability: Ensure the model can handle the expected load in a production environment.  **7. Monitoring and Maintenance**  * Performance Monitoring: Continuously monitor the model’s performance in the real world to ensure it meets the desired objectives. * Model Updating: Periodically update the model with new data to maintain its performance. This involves retraining the model or performing incremental learning. * Error Logging: Keep track of errors and performance degradation to identify when the model needs retraining or fine-tuning.  **8. Documentation and Reporting**  * Document Process: Document each step of the deep learning project lifecycle, including data preprocessing, model building, training, and evaluation. * Reporting: Create reports and visualizations to communicate findings and model performance to stakeholders.  **9. Feedback Loop**  * Collect Feedback: Gather feedback from users and stakeholders to understand the model's impact and areas for improvement. * Iterate: Use the feedback to make necessary adjustments and improvements to the model or the overall deep learning pipeline.  **Visual Representation** Here's a visual representation of the deep learning life cycle:  Evaluating a deep learning model involves using various performance metrics to understand how well the model is performing on unseen data. The choice of metrics depends on the specific task (e.g., classification, regression, etc.). Here, we'll focus on common metrics used for classification tasks:  **2 Performance Metrics of Deep Learning:** **1. Accuracy** Definition: Accuracy is the ratio of correctly predicted instances to the total instances.   * Formula:   Accuracy=[Total Number of Predictions] /[Number of Correct Predictions]  Example: If a model correctly predicts 90 out of 100 instances, the accuracy is 90%. **2. Precision** Definition: Precision is the ratio of correctly predicted positive observations to the total predicted positives.   * Formula: *Precision=[True Positives]/[True Positives + False Positives\*   Example: If a model predicts 10 instances as positive and 8 are actually positive, precision is 80%. **3. Recall (Sensitivity or True Positive Rate)** Definition: Recall is the ratio of correctly predicted positive observations to the all observations in actual class.   * Formula: *Recall=[True Positives]/[True Positives + False Negatives]*   Example: If a model identifies 8 out of 10 actual positive cases, recall is 80%. **4. F1 Score** Definition: The F1 Score is the weighted average of Precision and Recall. It considers both false positives and false negatives.   * Formula: *F1 Score=[Precision\*Recall]/[Precision + Recall]*   Example: If a model has a precision of 80% and recall of 70%, the F1 score is approximately 74.83%. **5. Confusion Matrix** Definition: A confusion matrix is a table used to describe the performance of a classification model by comparing actual vs. predicted values.   |  |  |  | | --- | --- | --- | |  | Predicted Positive | Predicted Negative | | Actual Positive | True Positive (TP) | False Negative (FN) | | Actual Negative | False Positive (FP) | True Negative (TN) |   This gives a breakdown of correct and incorrect classification **6. ROC Curve (Receiver Operating Characteristic)** Definition: A graphical plot that illustrates the diagnostic ability of a binary classifier as its discrimination threshold is varied.   * Axes: True Positive Rate (Recall) vs. False Positive Rate (1 - Specificity). * AUC (Area Under Curve): A single scalar value summarizing the performance. Higher AUC indicates a better model.   Use Case: Useful for evaluating models where you need to balance between sensitivity and specificity. **7. Precision-Recall Curve** Definition: A plot that shows the trade-off between precision and recall for different thresholds.   * Axes: Precision vs. Recall. * AUC-PR: Area Under the Precision-Recall Curve.   This curve is more informative than ROC curve while dealing with imbalanced datasets. **8. Logarithmic Loss (Log Loss)** Definition: Log Loss measures the performance of a classification model where the prediction is a probability value between 0 and 1.   * Formula: *Log Loss=−1N[∑i=1N[yi log⁡(pi)+(1−yi)log⁡(1−pi)]]*   *Where*   * + *N* : Number of samples.   + *yi* : True label (0 or 1).   + *pi* : Predicted probability.  **9. Specificity (True Negative Rate)** Definition: Specificity is the ratio of correctly predicted negative observations to all the observations in the actual negative class.   * Formula: *Specificity = [True Negatives]/[True Negatives + False Positives]*   Example: If a model correctly identifies 95 out of 100 actual negatives, the specificity is 95%.  **Inferences:**  The study of the life cycle of Deep Learning is done.    **Viva Questions:**   * 1. Which are the steps involved int he life cycle method of deep learning method?   2. Which are performance metrics of Deep learning method?   3. Differentiate between ROC and Precision Call Curve.   4. Which are performance metrics of Deep learning method?   5. Differentiate between ROC and Precision Call Curve.  1. **Develop a python code for single and muti layer perceptron in TensorFlow/Keras Environment using your own data or predefined**.   **Exp.No.2 Implement single and multi-layer perceptron using Keras dataset**  **Date:**    **Aim:** To learn the implementation of single and multi-layer perceptron using TensorFlow.  **Requirements:**  **Software:** Python with Tensorflow Library, Numpy library, data modeling library Scikit learn.  **Hardware:** Modern Operating System,x86 64-bit CPU (Intel / AMD architecture),4 GB RAM, 5 GB free disk space.  Dataset : keras .mnist,npz  A screenshot of a white sheet  Description automatically generated  **Suggested Reading(Theoretical Background):**  To work on current experiment the knowledge of python Programming and tensorflow operations are essential. This knowledge is essential to complete the lab as well. **Single-Layer Perceptron** A single-layer perceptron is the simplest form of a neural network, consisting of just one layer of output nodes connected to the input features.  **Components:**   * Input Layer: Takes input features. * Weights: Each input feature is assigned a weight that is adjusted during the training process. * Bias: An additional parameter that helps the model fit the data better. * Activation Function: A function applied to the weighted sum of inputs to produce the output.  **Multi-Layer Perceptron (MLP)** A multi-layer perceptron (MLP) is a more complex neural network that consists of multiple layers, including an input layer, one or more hidden layers, and an output layer.  **Components:**   * Input Layer: Takes the input features. * Hidden Layers: One or more layers where each neuron is connected to the neurons in the previous and next layers. These layers enable the network to learn complex patterns. * Output Layer: Produces the final output of the network. * Weights and Biases: Each connection between neurons has an associated weight and bias. * Activation Functions: Applied to the weighted sums in each layer to introduce non-linearity.   **Architecture:**   * The input layer receives the input features. * Each hidden layer receives inputs from the previous layer, applies weights, biases, and an activation function, and passes the output to the next layer. * The output layer produces the final prediction.   A diagram of a neural network  Description automatically generated  **Algorithm:**  Single layer Perceptron and Multi Layer Perceptoron:   1. Start the process 2. Import Tensor - To Import the TensorFlow library to access its machine learning functionalities. 3. Load the dataset – To load the MNIST dataset, which contains images of handwritten digits and their labels. 4. Preprocess the data – To normalize the pixel values to be between 0 and 1 and flatten the 28x28 images into 784-element vectors. 5. Build the single-layer perceptron model / Build the multi-layer perceptron model – To define a sequential model with an input layer, one or more hidden layers with activation functions, and an output layer with a softmax activation. 6. Compile the model – To configure the model with a loss function, optimizer, and metrics for evaluation. 7. Train the model – To train the model using the training data, specifying the number of epochs and the batch size. 8. Evaluate the model – To assess the model's performance on the test data to see its accuracy and loss. 9. Display the output – To print the test accuracy to see how well the model performs on unseen data. 10. Stop the process   **Sample Output/Result:**  **Single Layer Output**  Single-Layer Perceptron Test Accuracy: 0.9222999811172485  **Multi Layer Output**  Multi-Layer Perceptron Test Accuracy: 0.9702000021934509  **Inferences:**  The implementation of single and multi layer perceptron is demonstrated.  **Viva Questions:**   1. What is a single-layer perceptron, and how does it differ from a multi-layer perceptron? 2. Why is the softmax activation function often used in the output layer of a single-layer perceptron for multi-class classification problems? 3. How does a single-layer perceptron learn from the training data? 4. What are some practical applications where a single-layer perceptron might be used effectively? 5. What are some practical applications where a multi-layer perceptron might be used effectively? |
| 1. Develop a python code for Feed-Forward & Back propagation Network in TensorFlow/Keras.   **Exp.No.3 Implement feed-forward & Back propagation Network inTensorflow**  **Date:**    **Aim**: To learn the Implement feed-forward & Back propagation Network in TensorFlow.  **Requirements:**  Software: Python with Tensorflow Library, Numpy library, data modeling library Scikit learn.  Hardware: Modern Operating System,x86 64-bit CPU (Intel / AMD architecture),4 GB RAM,  5 GB free disk space.  Dataset: mnist.npz Converted to csv  **Inputs:**  X1,X2,X3,target  0.1,0.5, 0.8,0  0.2,0.4,0.7,1  0.3,0.3,0.6,0  0.4,0.2,0.5,1  0.5,0.1,0.4,1  0.6,0.9,0.3,0  0.7,0.8,0.2,1  0.8,0.7,0.1,0  0.9,0.6,0.0,1    **Suggested Reading(Theoretical Background):**  Feed-forward & Back propagation Network:  A feed – forward neural network is one of the basic types of artificial neural network. Here information travels only in forward direction from the input nodes to the output nodes, passing through any hidden nodes that may exist. The input layer is made up of nodes that receive the initial features of the data. Every node in the supplied data represents a feature. The hidden layer in which nodes make up these layers compute the input data. To these input weights and biases are applied to from previous layer. Each neuron sends the output to the layer. Hidden layer is used to identify intricate patterns in the data.    **Algorithm:** Start the processImport necessary libraries : The libraries such as mnist, sequential,dense,flatten, These are essential to run the mode.Define Neural Architecture Model: With help of import of libraries and architecture is built. Need to set the path for the directories.  1. Load and preprocess the dataset: The dataset mnist is loaded and normalized. These dataset are one-hot encode.  Build the feed-forward neural network model: Constructs the neural network with an input layer, two hidden layers, and an output layer.Compile the model: During this model it specifies optimizer, metrics and loss function.Train the model : The dataset is of two part one is train dataset and other is test dataset. With use of train model is trained.Evaluate the model : With metrics parameter evaluates the model’s performance on the test data.Display the results: This phase displays calculated output such as accuracy etc,.  1. Stop the process     **Sample Output/Result**  *Single-Layer Perceptron Test Accuracy: 0.9222999811172485*  *Multi-Layer Perceptron Test Accuracy: 0.9702000021934509*  **Inferences:**  The feed forward and back propogation network is implemented.  **Viva Questions:**   1. In the neural network what is feed farward? 2. Explain working of feed farward network. 3. What is back propogation? 4. Explain the process sof back propogation. 5. Explain significance of loss function in Back propogation. |

|  |
| --- |
| **Exp. No. 4. Build and Train a simple neural network (e.g. MNIST digit classification) using Keras, TensorFlow with backend and evaluate the model’s performance and visualize the result.**  **Aim:** To make use Keras with TensorFlow backend to build and train a simple neural network (e.g., MNIST digit classification). Evaluate the model's performance and visualize the results. TensorFlow / Keros  **Requirements:**  Software: Python with Tensorflow Library, Numpy library, data modeling library Scikit learn.  Hardware: Modern Operating System,x86 64-bit CPU (Intel / AMD architecture),4 GB RAM,  5 GB free disk space.  Data set : <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>  A screenshot of a computer  Description automatically generated  **Suggested Reading(Theoretical Background):**  Understanding of Python is required since Keras and TensorFlow are both Python libraries, and a solid understanding of Python programming is essential. The fundamental Machine Learning Concepts to understand the basic concepts such as supervised learning, training, and testing datasets, and model evaluation metrics. Knowledge of Neural Networks to get the knowledge of the structure and function of neural networks, including neurons, layers, activation functions, and backpropagation. Data Preprocessing Techniques such as normalization to understand the importance of normalizing data, such as scaling pixel values to a range of 0 to 1. One-Hot Encoding knowledge to convert categorical labels into a format that can be used for classification tasks. Model Evaluation and Performance Metrics, Training, and Validation must be studied. **Performance Evaluation:** **Metrics:**   * 1. **Accuracy:** The ratio of correctly predicted instances to the total instances.   2. **Precision:** The ratio of true positive predictions to the total positive predictions (true positives + false positives).   3. **Recall (Sensitivity):** The ratio of true positive predictions to the total actual positives (true positives + false negatives).   4. **F1 Score:** The harmonic mean of precision and recall, providing a balance between the two.   5. **ROC-AUC:** The area under the Receiver Operating Characteristic curve, indicating the model’s ability to distinguish between classes.   6. **Loss:** A measure of how well the model's predictions match the actual outcomes, commonly used during training.   **Confusion Matrix:**   * 1. A table that summarizes the performance of a classification model by showing the true positives, true negatives, false positives, and false negatives.  **Visualization** **Learning Curves:**  Plots showing the model's performance on the training and validation datasets over time (epochs). They help in understanding if the model is overfitting or underfitting.  **ROC Curve:**  A plot of the true positive rate (recall) against the false positive rate for different threshold values.  **Precision-Recall Curve:**  A plot of precision against recall for different threshold values.  **Confusion Matrix:**  Visual representation (heatmap) of the confusion matrix to easily identify the number of correct and incorrect predictions.  **Feature Importance:**  Bar plots or other visualizations to show the importance of different features in the model’s predictions.  **Activation Maps (for CNNs):**  Visualizations of the activations of different layers in a convolutional neural network to understand which parts of the input image are being emphasized.  **t-SNE or PCA Plots:**  Visualization techniques to reduce the dimensionality of high-dimensional data and visualize the data points in 2D or 3D space, useful for understanding the data distribution and clustering. **Example Workflow** **Training the Model:**   * 1. Split the data into training and validation sets.   2. Train the model on the training set.   3. Evaluate the model on the validation set using the chosen metrics.   **Visualizing Performance:**   * 1. Plot learning curves to observe the training and validation loss/accuracy over epochs.   2. Plot the confusion matrix to understand the model's prediction errors.   3. Plot the ROC and Precision-Recall curves.   4. Visualize feature importance if applicable.   5. For CNNs, visualize activation maps to interpret the model’s focus areas.   **Improving the Model:**  Based on the visualizations and metrics, decide on steps to improve the model, such as adjusting hyper parameters, using different architectures, adding regularization, or augmenting data.  **Algorithm:**   1. Start the process 2. Import Library: Import TensorFlow to utilize its functions and operations. 3. Define Vectors: Create constant tensors to represent the vectors to be added. Also check the dimension of the vector input. 4. Add Vectors: Use TensorFlow's add function to perform element-wise addition of the vectors. (Here other operations such as multiplication, division, transpose etc. can also performed). 5. Run the Computation: Execute the addition operation and convert the result to a NumPy array for easy interpretation. 6. Display the result. 7. Stop the Process   **Sample Output/Result:**  A group of white letters in black squares  Description automatically generated  **Inferences:**  This experiment demonstrates how to perform basic vector addition using tensor flow.  **Viva Questions:**   1. What is the MNIST dataset, and why is it commonly used for neural network training and evaluation? 2. How do you pre-process the MNIST dataset before feeding it into a neural network? 3. What are the main components of a simple neural network model used for digit classification? 4. What are some techniques to visualize the results of the neural network's performance on the MNIST dataset? 5. How does using Keras with a TensorFlow backend simplify the process of building and training neural networks? |
| **Exp.No.5 Implement Image Classifier using CNN in TensorFlow/Keras Date:**  **Aim:** To implement an Image Classifier using CNN in TensorFlow/Keras.  **Requirements:**  Software: Python with Tensorflow Library, Numpy library, data modeling library Scikit learn.  Hardware: Modern Operating System, x86 64-bit CPU (Intel / AMD architecture),4 GB RAM,  5 GB free disk space.  Dataset : keras.datasets.cif10    **Suggested Reading (Theoretical Background):**  To get the knowledge of this program candidate should have knowledge of CNN. Understanding of basic neural network architecture components like layers, activation functions (e.g., ReLU, Sigmoid), and how information flows through a network. Familiarity with CNN concepts such as convolutional layers, pooling layers (e.g., MaxPooling), padding, and strides. Understanding how these layers extract features from images hierarchically.Knowledge of gradient descent variants (e.g., SGD, Adam) used for optimizing neural network parameters.  **CNN Architecture:**    · **Input Layer:**   * The input layer holds the raw pixel values of the image. For example, an RGB image of size 32x32 pixels will have a shape of (32, 32, 3).   · **Convolutional Layer (Conv Layer):**   * The convolutional layer applies a set of filters (kernels) to the input image to create feature maps. Each filter detects a specific feature (like edges, textures, etc.). * Parameters:   + Number of filters: Determines the depth of the output volume.   + Filter size (kernel size): Common sizes are 3x3, 5x5, etc.   + Stride: Determines how the filter moves across the input.   + Padding: 'Same' (output size same as input) or 'Valid' (no padding, reduces size).   · **Activation Function:**   * Non-linear activation functions like ReLU (Rectified Linear Unit) are applied after each convolution to introduce non-linearity into the model.   · **Pooling Layer:**   * Pooling layers reduce the spatial dimensions (width and height) of the feature maps, keeping the depth unchanged. This helps in reducing the computational load and controlling overfitting. * Types:   + Max Pooling: Takes the maximum value in each patch.   + Average Pooling: Takes the average value in each patch. * Common pool size is 2x2 with a stride of 2.   · **Fully Connected Layer (Dense Layer):**   * Fully connected layers are used towards the end of the network to combine features learned by convolutional layers and predict the final output. Each neuron in a fully connected layer is connected to every neuron in the previous layer.   · **Output Layer:**   * The output layer provides the final predictions. For a classification task, it typically uses a softmax activation function to output probabilities for each class.   **Algorithm:**   1. Start the process 2. Import Library : Import TensorFlow to utilize its functions and operations. 3. Define Vectors: Create constant tensors to represent the vectors to be added. Also check the dimension of the vector inputed. 4. Add Vectors: Use TensorFlow's add function to perform element-wise addition of the vectors. (Here other operations such as multiplication, division, transpose etc. can also performed). 5. Run the Computation: Execute the addition operation and convert the result to a NumPy array for easy interpretation. 6. Display the result. 7. Stop the process   **Sample Output/Result:**  accuracy: 0.7010999917984009  313/313 [==============================] - 4s 12ms/step    **Inferences:**  This experiment demonstrates implement an Image Classifier using CNN in TensorFlow/Keras.  **Viva Questions:**   1. What is a Convolutional Neural Network (CNN) and how does it differ from a fully connected neural network? 2. Explain the role of convolutional layers and pooling layers in a CNN. 3. How do they contribute to feature extraction? 4. Describe the process of building and compiling a CNN model using TensorFlow/Keras. 5. What are the typical layers included in a CNN architecture? |
|  |
|  |

**Exp.No 5 Implement an LSTM-based Autoencoder in TensorFlow/Kera Date:**

**AIM**

To implement an LSTM-based model show that one-way time-series data can be efficiently encoded to lower dimensions and used in non-time-series models.

**Requirements**

Software

* Python: Pycharm, Jupyter, Google Colab
* Packages: sklearn, TensorFlow, keras:layer, optimizer, sequential, Model
* Import, Json, codec, glob
* Python Pre-processing packages: feature\_scaling, drop.na [MOOC-1,2]

Hardware:

* Hardware: Modern Operating System,x86 64-bit CPU (Intel / AMD architecture)
* 4 GB RAM

**Suggested Reading**

* Long Short-Term Memory is an improved version of recurrent neural network designed by Hochreiter & Schmidhuber.
* A traditional[RNN](https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/) has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies. LSTMs model address this problem by introducing a memory cell, which is a container that can hold information for an extended period.
* LSTM architectures are capable of learning long-term dependencies in sequential data, which makes them well-suited for tasks such as [language translation](https://www.geeksforgeeks.org/language-translator-using-google-api-in-python/), speech recognition, and [time series forecasting](https://www.geeksforgeeks.org/time-series-forecasting-using-recurrent-neural-networks-rnn-in-tensorflow/).
* LSTMs can also be used in combination with other neural network architectures, such as [Convolutional Neural Networks](https://www.geeksforgeeks.org/introduction-convolution-neural-network/) (CNNs) for image and video analysis.

## **LSTM Architecture**

The LSTM architectures involves the memory cell which is controlled by three gates: the input gate, the forget gate, and the output gate. These gates decide what information to add to, remove from, and output from the memory cell.

* The input gate controls what information is added to the memory cell.
* The forget gate controls what information is removed from the memory cell.
* The output gate controls what information is output from the memory cell.

A diagram of a flowchart

Description automatically generated

**Algorithm:**

1. Start the process
2. Importing the required image dataset to identify the object in images.

*from IPython.display import Image*

*Image("../input/object.jpg ")*

A street with many signs

Description automatically generated

### Pre-processing of the images

* *x\_train: Numpy arrays of the images of the training dataset*
  + *x\_train[y\_train == 7]*
* *y\_train: Labels of the training dataset*
  + *y\_train[y\_train == 7]*
* *x\_test: Numpy arrays of the images of the testing dataset*
  + *set x\_test[y\_test == 7]*
* *y\_test: Labels of the testing dataset*
  + *sety\_test[y\_test == 7]*

1. Set seeds to make the experiment more reproducible.
   1. seed\_everything(seed=0): random.seed(seed)
   2. np.random.seed(seed) , tf.random.set\_seed(seed)
   3. os.environ['PYTHONHASHSEED'] = str(see environ['TF\_DETERMINISTIC\_OPS'] = '1'
2. Initialize LSTM Autoencoder:
   1. encoder\_decoder = Sequential()
   2. encoder\_decoder.add(L.LSTM(serie\_size, activation='relu',
   3. input\_shape=(serie\_size, n\_features), return\_sequences=True))
   4. adam = optimizers.Adam(lr)
   5. encoder\_decoder.compile(loss='mse', optimizer=adam)
3. Building LSTM:MLP with LSTM encoded feature
4. mlp\_model.add(L.Dense(1))
5. mlp\_model.summary()

### Stop the process

Sample Input

A screenshot of a calculator

Description automatically generated

Sample Output

A number on a white background

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Inferences:**

This experiment demonstrates implement LSTM-based Autoencoder in TensorFlow/Keras.

Viva Questions:

1. What is a Convolutional Neural Network (CNN) and how does it differ from a fully connected neural network?
2. Explain the role of convolutional layers and pooling layers in a LSTM.
3. How do they contribute to feature extraction?
4. Describe the process of building and compiling a LSTM model using TensorFlow/Keras.
5. What are the typical layers included in a LSTN architecture?

**Exp.No.6 Implement Object Detection using CNN Date:**

**AIM**

To detect the object, present it in the given input image using CNN models.

**Requirements**

Software

1. Python: Pycharm, Jupyter, Google Colab
2. Packages: sklearn, TensorFlow, CV2, Conv2D, Conv2DTranspose, Matplotlib [MOOC-1,2]
3. Import, Json, codec, glob
4. Importing the image\_detection Dataset
   1. Data source link- <https://www.kaggle.com/code/kavyasmallika/object-detection-cnn/input>
5. Python Pre-processing packages [MOOC-1,2]

Hardware:

1. Hardware: Modern Operating System,x86 64-bit CPU (Intel / AMD architecture)
2. 4 GB RAM

**Suggested Reading:**

1. Python: sklearn, TensorFlow, CV2, Conv2D, Conv2DTranspose, Matplotlib [MOOC-1,2]
2. Importing the Image dataset Dataset
   1. Data source link- <https://www.kaggle.com/code/kavyasmallika/object-detection-cnn/input>
3. Pre-processing : import Rescaling or Normalization using Keras [MOOC-1,2]
4. Creating or Setting object detection CNN or YOLO [MOOC-1,2]
   1. Set model.classifier 🡪 import Conv2DTranspose from keras
   2. Image Classifier 🡪 define discriminator in function
5. Training: initialize epoch, custome\_model.fit
6. Model evaluation: np.argmax, predict(test)

**(refer CNN concepts and Architecture in the Exp No: 7)**

**Algorithm:**

1. Start the process
2. Importing the required image dataset to identify the object in images.

*from IPython.display import Image*

*Image("../input/object.jpg ")*

A person leading a group of sheep

Description automatically generated

1. Pre-processing of the images

* *x\_train: Numpy arrays of the images of the training dataset*
  + *x\_train[y\_train == 7]*
* *y\_train: Labels of the training dataset*
  + *y\_train[y\_train == 7]*
* *x\_test: Numpy arrays of the images of the testing dataset*
  + *set x\_test[y\_test == 7]*
* *y\_test: Labels of the testing dataset*
  + *sety\_test[y\_test == 7]*

1. Creating or Setting image classifier[MOOC-1,2]
   1. Set classifier 🡪 import Conv2DTranspose from keras
      1. define models.Sequential()
      2. set optimizer=adam
      3. initialize activation = 'relu
      4. model.add(layers.Dense(y\_train.shape[-1], activation = 'relu'))
   2. Initialize YOLO model or general CNN 🡪 define discriminator in function
      1. define the Res.Classifier():
      2. Set model = tf.keras.Sequential()
      3. Set model = flattern
      4. Set res\_classifier,summary()
2. Defining Training: initialize epoch, custome\_model.fit
   * 1. define train\_steps(images):
     2. set noise = np.random.normal(0,1,(batch\_size,latent\_dim))
     3. set gen\_loss = generator\_loss(fake\_output)

### Model evaluation

### set np.argmax=predict.x\_test

### set np.argmax=predict.y\_test

### predict(x\_test, y\_test)

### y\_test\_pred = best\_model.predict(X\_test)

Sample Input:

A person leading a group of sheep

Description automatically generated

Sample Output:

A collage of several images of sheep

Description automatically generated

**Inferences:**

This experiment demonstrates object selection using CNN in TensorFlow/Keras.

**Viva Questions:**

1. What is a YOLO and how does it differ from a CNN?
2. Explain the role of convolutional layers, flatten, dense and pooling layers in a YOLO.
3. How do they contribute to feature extraction?
4. Describe the process of building and compiling a YOLO model using TensorFlow/Keras.
5. What are the typical layers included in a YOLO architecture?

**Exp. No. 7 Implement basic Convolutional Neural Network from scratch using Keras and TensorFlow. Train it on a simple image dataset (e.g. CIFAR-10 and visualize the filters learned by the convolutional layers.**

**AIM**

To implement an animal input image classification model using Transfer learning.

**Requirements**

Softwares:

1. Python libraries: numpy, pandas, matplotlib, seabon
2. Python module libraries: keras: backend, sequential, optimizers
3. Collect flower dataset
   1. Data resource: <https://www.kaggle.com/c/cifar-10>
4. Python Pre-processing packages [MOOC-1,2]
5. Python standard data science libraries: psutil, keras pretrained model

Hardwares:

1. Hardware: Modern Operating System,x86 64-bit CPU (Intel / AMD architecture),
2. 4 GB RAM,
3. 5 GB free disk space.

**Suggested Reading**

Refer the exp. No 7 for the concepts of CNN model and its Architecture

* 1. Data source link- <https://www.kaggle.com/c/cifar-10>

A collage of images of animals and cars

Description automatically generated

* 1. tensorflow.keras.datasets import cifar10
  2. The CIFAR-10 data consists of 60,000 32x32 color images in 10 classes, with 6000 images per class. There are 50,000 training images and 10,000 test images in the official data. We have preserved the train/test split from the original dataset. The provided files are:
  3. train.7z - a folder containing the training images in png format
  4. test.7z - a folder containing the test images in png format

**Procedure:**

1. Import Necessary packages related to Transfer Learning

First and foremost, the step is to load the data.

* Then search for "Kera’s Pretrained Models" and add it to your environment.
* Last import necessary files into "Keras" directory to ease the environment.
* from *keras.models import Sequential, Convolution, Dropout, pre-trained model*

1. Preparing the Dataset

*from IPython.display import Image*

*Image("../input/animages/1.png")*

A collage of images of animals and cars

Description automatically generated

1. import dependencies

* from keras.model, Sequential, Flatten, decode\_prediction, layers\_score
* from keras.model, droput, json, metrics, decode\_prediction

1. Creating or Setting pooling value [MOOC-1,2]

* Set random.seed(SEED\_VALUE)
* Set np.random.seed(SEED\_VALUE)
* Set tf.random.set\_seed(SEED\_VALUE)
* Initialize inception v3🡪 define inception v3 parameter

1. Training: initialize epoch, custome\_model.fit

* (X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()
* (X\_train.shape), (X\_test.shape)
* class DatasetConfig:

NUM\_CLASSES: int = 10

IMG\_HEIGHT: int = 32

IMG\_WIDTH: int = 32

NUM\_CHANNELS: int = 3

* class TrainingConfig:

EPOCHS: int = 31

BATCH\_SIZE: int = 256

LEARNING\_RATE: float = 0.001

1. Model evaluation: np.argmax, predict(test), classification\_report, confusion matrix
2. Repeat the process till all images are trained and tested

**Sample Input:**

A collage of images of animals and cars

Description automatically generated

**Sample Output:**

A collage of images of different animals

Description automatically generated A white paper with black text

Description automatically generated with medium confidence

**Inferences:**

This experiment demonstrates CNN model visualization in TensorFlow/Keras.

**Viva Questions:**

1. What is the difference between Conv2D, Conv2DTranspose?
2. Explain the role of decode\_prediction and layers\_score function.
3. How do they contribute to feature extraction?
4. Describe the process of linear\_kernal, cosine similiarity.
5. What are the typical layers included in a CNN Model, and data config, training config?

**Exp. No. 8 Implement Transfer Learning Concept in Image Classification using UGG-16 and ResNet.**

**8a) Implement Transfer Learning Concept in Image Classification using UGG-16**

**AIM**

To implement image classification using VGG-16.

**Requirements**

Software

1. Python: Pycharm, Jupyter
2. Packages: sklearn, TensorFlow, CV2, Conv2D, Conv2DTranspose, Matplotlib [MOOC-1,2]
3. Importing the MNIST Dataset
   1. Data source link- <https://www.kaggle.com/code/apurvayadav29/image-generation-using-gan/notebook>
4. Python Pre-processing packages [MOOC-1,2]

Hardware:

1. Hardware: Modern Operating System,x86 64-bit CPU (Intel / AMD architecture)
2. 4 GB RAM

**Suggested Reading**

Image classification using VGG-16 involves using a pre-trained convolutional neural network (CNN) model that has been widely used for various computer vision tasks, including image classification. VGG-16 (Visual Geometry Group - 16 layers) is known for its simplicity and effectiveness, consisting of 16 layers (13 convolutional layers and 3 fully connected layers).

* Importing the MNIST Dataset
  1. Data source link- <https://www.kaggle.com/code/apurvayadav29/image-generation-using-gan/notebook>

A number on a black background

Description automatically generated

**VGG-16 Architecture:**

A green and blue squares with black numbers

Description automatically generated with medium confidence

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual competition in computer vision where teams tackle tasks including object localization and image classification. VGG16, proposed by Karen Simonyan and Andrew Zisserman in 2014, achieved top ranks in both tasks, detecting objects from 200 classes and classifying images into 1000 categories.

**Algorithm:**

1. Start the Process
2. Importing the MNIST Dataset to classify the given digits

*from IPython.display import Image*

*Image("../input/MNIST.jpg ")*

A number on a black background

Description automatically generated A table with numbers and letters

Description automatically generated

### Pre-processing of the images

* x\_train: Numpy arrays of the images of the training dataset
  + x\_train[y\_train == 7]
* y\_train: Labels of the training dataset
  + y\_train[y\_train == 7]
* x\_test: Numpy arrays of the images of the testing dataset
  + set x\_test[y\_test == 7]
* y\_test: Labels of the testing dataset
  + sety\_test[y\_test == 7]

1. Creating or Setting image classifier[MOOC-1,2]

Set classifier 🡪 import Conv2DTranspose from keras

* define classifier.model.VGG16():
* Set model = tf.keras.Sequential()

Initialize ResNet.classifier 🡪 define discriminator in function

* define the Res.Classifier():
* Set model = tf.keras.Sequential()
* Set model = flattern
* Set res\_classifier,summary()

1. Defining Training: initialize epoch, custome\_model.fit

* define train\_steps(images):
* set noise = np.random.normal(0,1,(batch\_size,latent\_dim))
* set gen\_loss = generator\_loss(fake\_output)

### Model evaluation

* set np.argmax=predict.x\_test
* set np.argmax=predict.y\_test
* predict(x\_test, y\_test)
* y\_test\_pred = best\_model.predict(X\_test)

### Stop the process

**Sample Input:**

A number on a black background

Description automatically generated A group of rainbow colored lines

Description automatically generated A group of rainbow colored arrows

Description automatically generated A group of colored lines

Description automatically generated with medium confidence

**Sample Output:**

A number set of numbers

Description automatically generated with medium confidenceA group of rainbow colored lines

Description automatically generatedA group of colorful letters

Description automatically generated with medium confidence A group of colorful lines

Description automatically generated with medium confidence

**Inferences:**

This experiment demonstrates Image Classification using VGG-16 in TensorFlow/Keras.

**Viva Questions:**

1. What is the difference between Conv2D, Conv2DTranspose?
2. Explain the role of decode\_prediction and layers\_score function.
3. Define Epoch?
4. Describe the process of linear\_kernal, cosine similiarit, random\_scaling.
5. What are step\_size, gen\_loss?

**8b) Implement Transfer Learning Concept in Image Classification using ResNet.**

**AIM**

To implement a Transfer Learning concept inImage Classification using ResNet

**Requirements**

Softwares:

1. Python libraries: numpy, pandas, matplotlib, seabon
2. Python module libraries: keras: backend, sequential, optimizers
3. Collect flower dataset
   1. Data resource: <https://www.kaggle.com/alessiocorrado99/animals10>
   2. or https://www.kaggle.com/code/min4tozaki/animal-classification
4. Python Pre-processing packages [MOOC-1,2]
5. Python standard data science libraries: psutil, keras pretrained model

Hardwares:

1. Hardware: Modern Operating System,x86 64-bit CPU (Intel / AMD architecture),
2. 4 GB RAM, 5 GB free disk space.

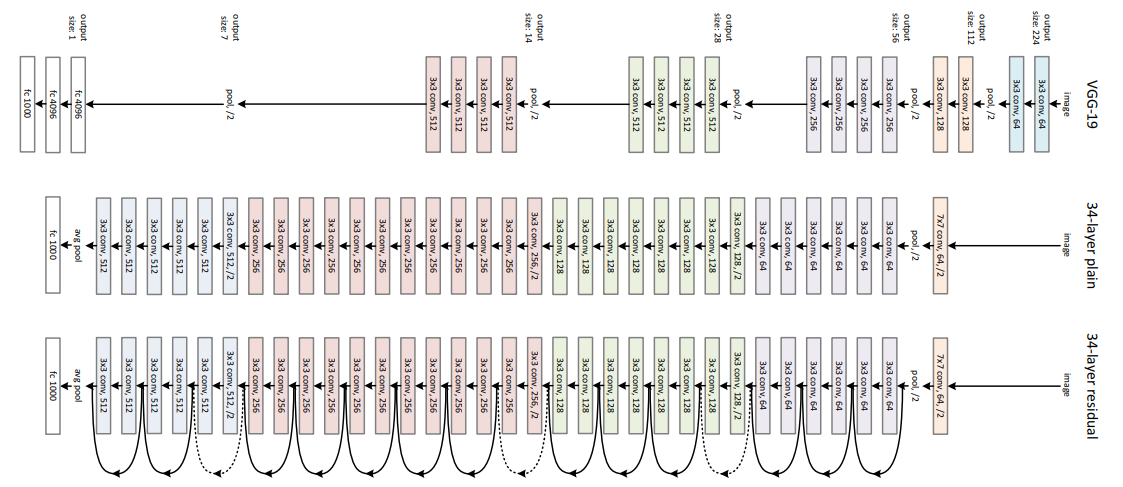
**Suggested Reading**

Residual Network: In order to solve the problem of the vanishing/exploding gradient, this architecture introduced the concept called Residual Blocks. In this network, we use a technique called skip connections. The skip connection connects activations of a  layer to further layers by skipping some layers in between. This forms a residual block. Resnets are made by stacking these residual blocks together.

A diagram of a diagram

Description automatically generated

**Network Architecture:** This network uses a 34-layer plain network architecture inspired by VGG-19 in which then the shortcut connection is added. These shortcut connections then convert the architecture into a residual network.  

[](https://media.geeksforgeeks.org/wp-content/uploads/20200424011138/ResNet.PNG)

*ResNet -34 architecture*

image dataset

* 1. Data source link- <https://www.kaggle.com/code/min4tozaki/animal-classification>
  2. Or <https://www.kaggle.com/alessiocorrado99/animals10>
  3. A collage of animals

     Description automatically generated

**Algorithm:**

1. Import Necessary packages related to Transfer Learning

We first need to download the pre-trained model and its weights to transfer learning.

* First and foremost, the step is to load the data.
* Then search for "Kera’s Pretrained Models" and add it to your environment.
* Last import necessary files into "Keras" directory to ease the environment.
* from *keras.models import Sequential, Convolution, Dropout, pre-trained model*

1. Preparing the dataset

*from IPython.display import Image*

*Image("../input/animages/1.png")*

A collage of animals

Description automatically generatedA white rectangular box with black text

Description automatically generated

1. Pre-processing: import Rescaling or Normalization using ResNet [MOOC-1,2]
2. Creating or Setting pooling value [MOOC-1,2]

* Set zeropaddingr 🡪 import Conv2DTranspose from keras
* Initialize inception v3🡪 define inception v3 parameter

1. Training: initialize epoch, custome\_model.fit
2. Model evaluation: np.argmax, predict(test), classification\_report, confusion matrix

**Sample input**

A collage of animals

Description automatically generated

**Sample output**

A number of numbers and letters

Description automatically generated with medium confidence

A collage of images of a mouse and a dog

Description automatically generated

A comparison of elephants and horses

Description automatically generated

A collage of pictures of chickens

Description automatically generated

**Inferences:**

This experiment demonstrates transfer learning using ResNet in TensorFlow/Keras.

**Viva Questions:**

1. What is the difference between Conv2D, Conv2DTranspose in transfer learning?
2. Explain the role of decode\_prediction and layers\_score function.
3. Define flatten, decode prediction?
4. Describe the process of linear\_kernal, cosine similiarit, random\_scaling.
5. What are step\_size, gen\_loss, sequential flattern?