Experiment No. 1

Aim: Real estate agents want help to predict the house price for regions in the USA. He gave you the dataset to work on and you decided to use the Linear Regression Model. Create a model that will help him to estimate what the house would sell for.

URL for a dataset: https://github.com/huzaifsayed/Linear-Regression-Model-for-House-Price-Prediction/blob/master/USA_Housing.csv

Objective:

The main objectives of this experiment are as follows:

1. Understand the fundamental principles and concepts underlying simple and multiple linear regression techniques.
2. Implement linear regression algorithms to analyze the relationship between house prices and various independent features such as location, size, number of bedrooms, etc.
3. Evaluate the performance of the linear regression model in accurately predicting house prices based on the selected features.

Theory:

Linear regression is a widely used statistical method for modeling the relationship between a dependent variable (target) and one or more independent variables (predictors). In the context of house price prediction:

Simple Linear Regression:

- Simple linear regression involves a single independent variable (e.g., house size, number of bedrooms) and a dependent variable (house price).
- The relationship between the independent and dependent variables is modelled using a straight line equation: $Y=\beta 0+\beta 1X$, where Y represents the dependent variable (house price), X represents the independent variable, $\beta 0$ is the intercept, and $\beta 1$ is the slope coefficient.

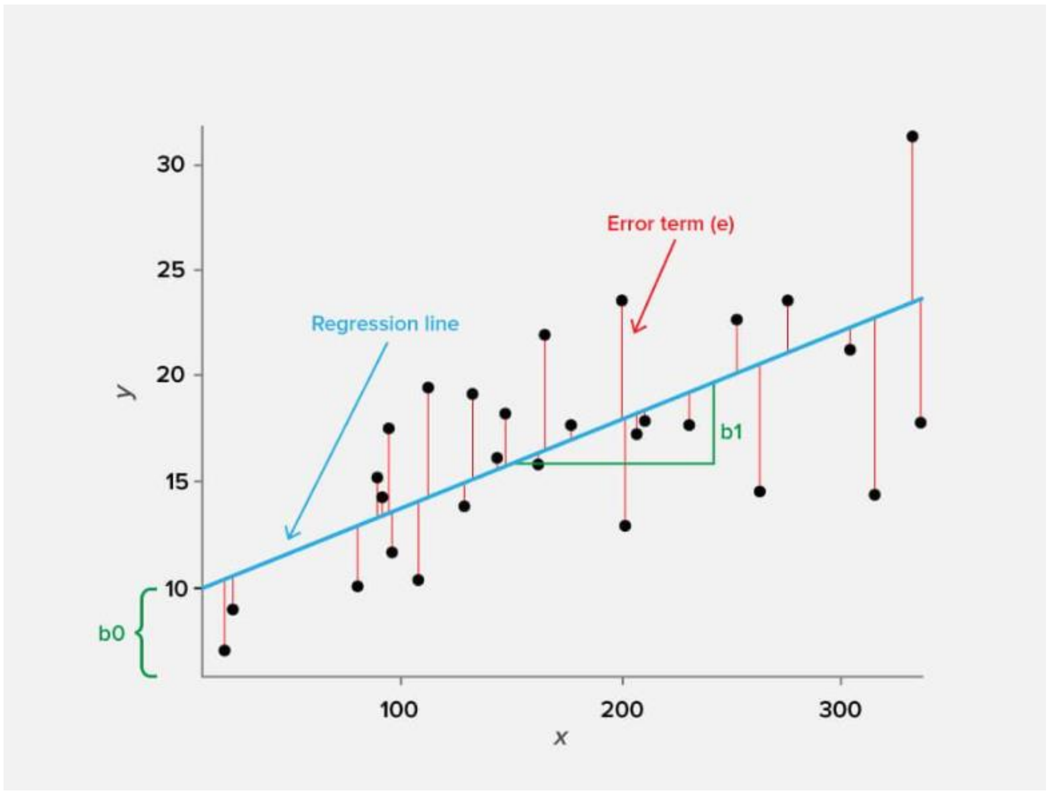


Fig 1: Regression Model

Multiple Linear Regression:

- Multiple linear regression extends the simple linear regression model to incorporate multiple independent variables.
- The relationship between the dependent variable and multiple independent variables is modeled using the equation: $Y=\beta 0+\beta 1X1+\beta 2X2+...+\beta nXn$, where $X1,X2,...,Xn$ represent the independent variables, and $\beta 0,\beta 1,...,\beta n$ are the corresponding coefficients.

The primary objective of linear regression is to find the best-fit line that minimizes the error between the predicted and actual values, thereby accurately estimating house prices based on the provided features.

Applications:

Linear regression has numerous applications in real-world scenarios, including:

- Real estate: Predicting house prices based on features such as location, size, amenities, etc.
- Finance: Analyzing the relationship between economic indicators and housing market trends.
- Market research: Estimating sales figures based on advertising expenditure and consumer demographics.

Input:

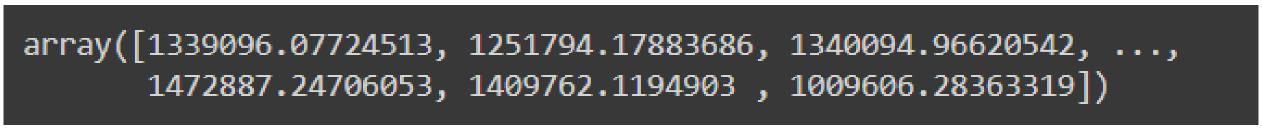
Dataset: House Price Prediction

- The dataset contains information on various features of houses (e.g., size, number of bedrooms, location) and their corresponding selling prices.

Output:

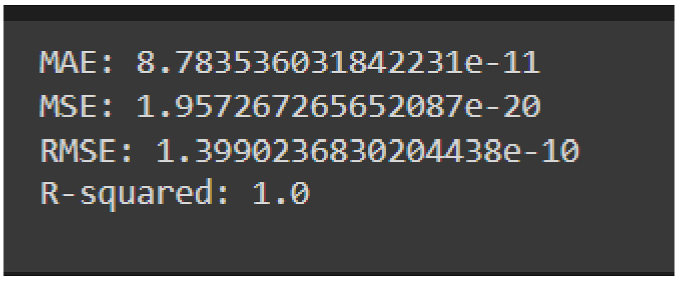
The output of this experiment includes:

1. Predicted house prices: The model generates estimated house prices based on the provided features using the linear regression algorithm.



```
array([1339096.07724513, 1251794.17883686, 1340094.96620542, ...,
       1472887.24706053, 1409762.1194903 , 1009606.28363319])
```

2. Evaluation metrics: Metrics such as mean squared error, root mean squared error, and R-squared value are used to assess the accuracy and performance of the linear regression model.



```
MAE: 8.783536031842231e-11
MSE: 1.957267265652087e-20
RMSE: 1.3990236830204438e-10
R-squared: 1.0
```

Conclusion:

Through the application of linear regression techniques, we have successfully developed a predictive model for estimating house prices in regions across the USA. By analyzing the relationship between house features and selling prices, the model provides valuable insights for real estate professionals to make informed decisions.

Outcome:

The outcome of this experiment is a trained linear regression model capable of accurately predicting house prices based on the selected features. This model can be utilized by real estate agents, property developers, and prospective buyers/sellers to estimate the market value of residential properties.

Questions:

1. What are the key assumptions underlying the linear regression model?
2. How do you select and preprocess features for inclusion in the regression analysis?
3. What techniques can be employed to address multicollinearity among independent variables?
4. How do you interpret the coefficients obtained from the regression model?
5. What are the implications of heteroscedasticity on the validity of the regression results?

Experiment No. 2

Aim: Build a Multiclass classifier using the CNN model. Use MNIST or any other suitable dataset.

a. Perform Data Pre-processing

b. Define Model and perform training

c. Evaluate Results using confusion matrix.

Objective:

The main objectives of this experiment are as follows:

1. Develop a deep learning model architecture suitable for image classification tasks.
2. Train the CNN model on the MNIST dataset to learn features and patterns associated with handwritten digits.
3. Evaluate the performance of the trained model in accurately classifying digit images into their respective numerical classes.

Theory:

Convolutional Neural Networks (CNNs) are a type of deep learning model commonly used for image classification tasks. They are characterized by their ability to automatically learn hierarchical patterns and features from input images. Key components of CNNs include convolutional layers, pooling layers, and fully connected layers. CNNs excel at capturing spatial hierarchies of features in images, making them well-suited for tasks like digit recognition in the MNIST dataset.

Applications:

CNNs have a wide range of applications in computer vision tasks, including:

- Object detection
- Facial recognition
- Image segmentation
- Handwriting recognition (as demonstrated in the MNIST dataset)

Input:

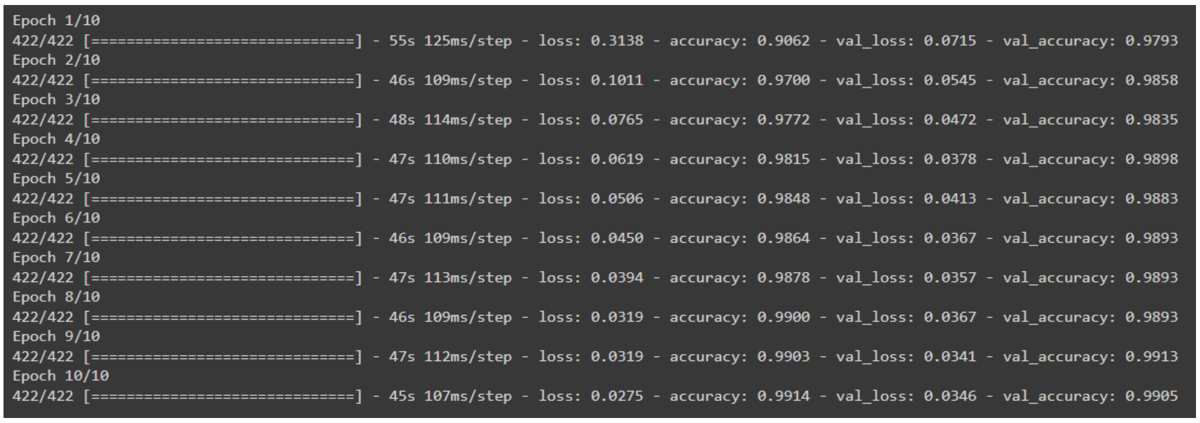
Dataset: MNIST dataset

- The MNIST dataset consists of 28x28 pixel grayscale images of handwritten digits (0-9).

Output:

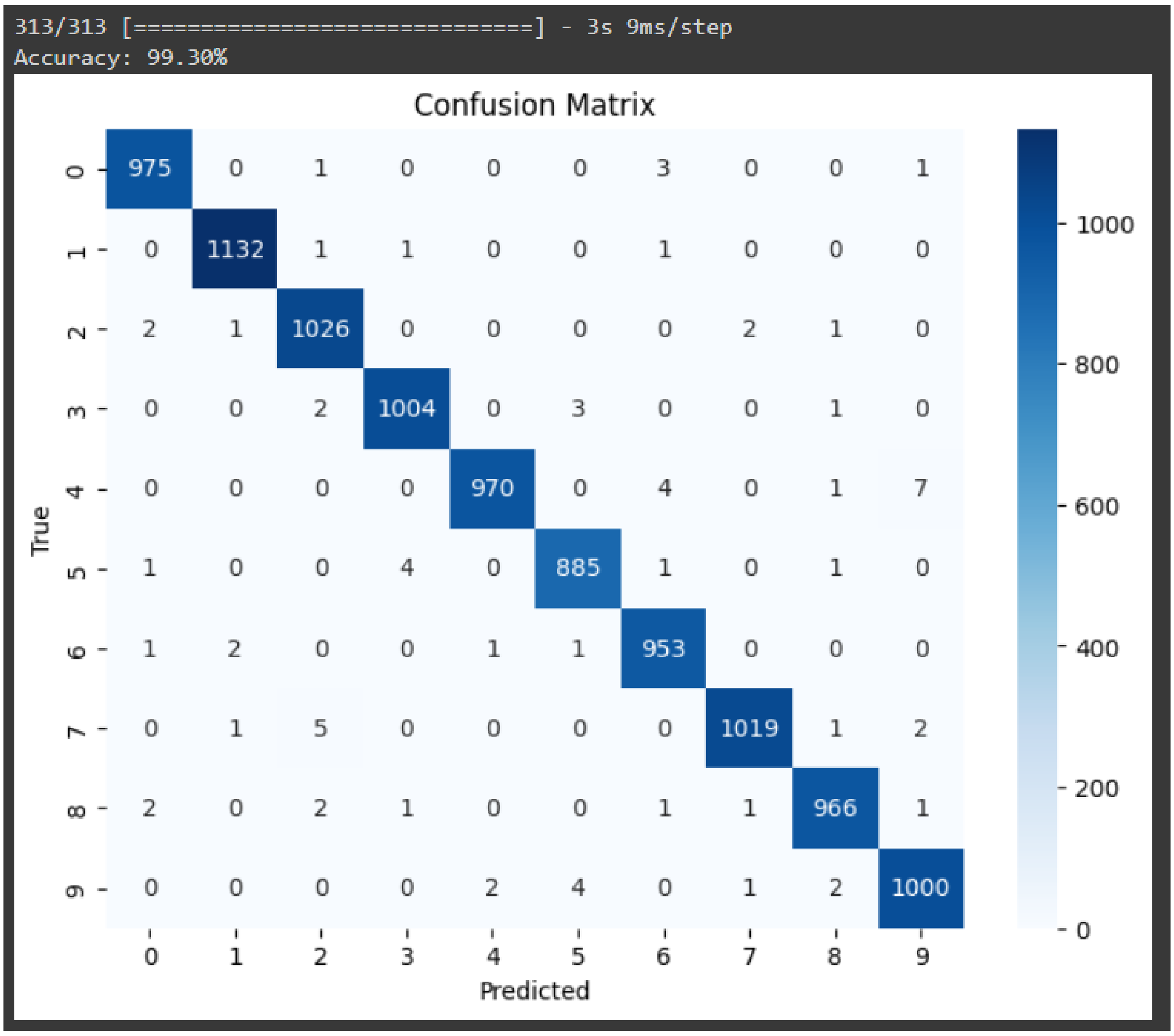
The output of this experiment includes:

1. Trained CNN model capable of classifying handwritten digits.



```
Epoch 1/10
422/422 [==============================] - 55s 125ms/step - loss: 0.3138 - accuracy: 0.9062 - val_loss: 0.0715 - val_accuracy: 0.9793
Epoch 2/10
422/422 [==============================] - 46s 109ms/step - loss: 0.1011 - accuracy: 0.9700 - val_loss: 0.0545 - val_accuracy: 0.9858
Epoch 3/10
422/422 [==============================] - 48s 114ms/step - loss: 0.0765 - accuracy: 0.9772 - val_loss: 0.0472 - val_accuracy: 0.9835
Epoch 4/10
422/422 [==============================] - 47s 110ms/step - loss: 0.0619 - accuracy: 0.9815 - val_loss: 0.0378 - val_accuracy: 0.9898
Epoch 5/10
422/422 [==============================] - 47s 111ms/step - loss: 0.0506 - accuracy: 0.9848 - val_loss: 0.0413 - val_accuracy: 0.9883
Epoch 6/10
422/422 [==============================] - 46s 109ms/step - loss: 0.0450 - accuracy: 0.9864 - val_loss: 0.0367 - val_accuracy: 0.9893
Epoch 7/10
422/422 [==============================] - 47s 113ms/step - loss: 0.0394 - accuracy: 0.9878 - val_loss: 0.0357 - val_accuracy: 0.9893
Epoch 8/10
422/422 [==============================] - 46s 109ms/step - loss: 0.0319 - accuracy: 0.9900 - val_loss: 0.0367 - val_accuracy: 0.9893
Epoch 9/10
422/422 [==============================] - 47s 112ms/step - loss: 0.0319 - accuracy: 0.9903 - val_loss: 0.0341 - val_accuracy: 0.9913
Epoch 10/10
422/422 [==============================] - 45s 107ms/step - loss: 0.0275 - accuracy: 0.9914 - val_loss: 0.0346 - val_accuracy: 0.9905
```

2. Confusion matrix to visualize the performance of the model in classifying digits.



```
313/313 [==============================] - 3s 9ms/step
Accuracy: 99.30%
```

Conclusion:

In this analysis, we successfully developed and trained a CNN model to classify handwritten digits from the MNIST dataset. By evaluating the model's performance, we gained insights into its accuracy and effectiveness in digit recognition tasks.

Outcome:

The outcome of this experiment is a trained CNN model capable of accurately classifying handwritten digits. This model can be deployed in various applications requiring digit recognition, such as optical character recognition (OCR) systems and automated form processing.

Questions:

1. How does the architecture of a CNN facilitate feature extraction and hierarchical learning?
2. What techniques are commonly used for data augmentation in CNN training to improve model generalization?
3. How do convolutional and pooling layers contribute to the spatial hierarchies of features learned by the CNN?
4. What are some common challenges or limitations encountered when training CNN models on image datasets like MNIST?
5. How can transfer learning be applied to CNN models trained on datasets similar to MNIST for tasks with limited labelled data?

Experiment No. 3

Aim: Design RNN or its variant including LSTM or GRU

a) Select a suitable time series dataset. Example – predict sentiments based on product reviews

b) Apply for prediction

Objective:

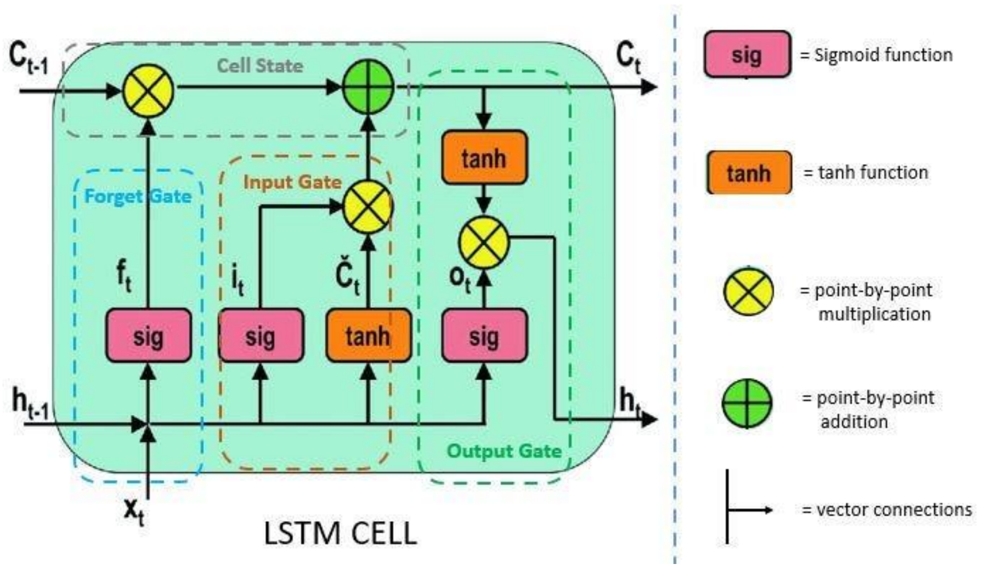
The main objectives of this experiment are as follows:

1. Understand the architecture and functionality of RNNs, LSTM, and GRU networks.
2. Implement an RNN-based model to analyze sequential data and perform sentiment analysis on product reviews.
3. Evaluate the performance of the designed model in predicting sentiments accurately.

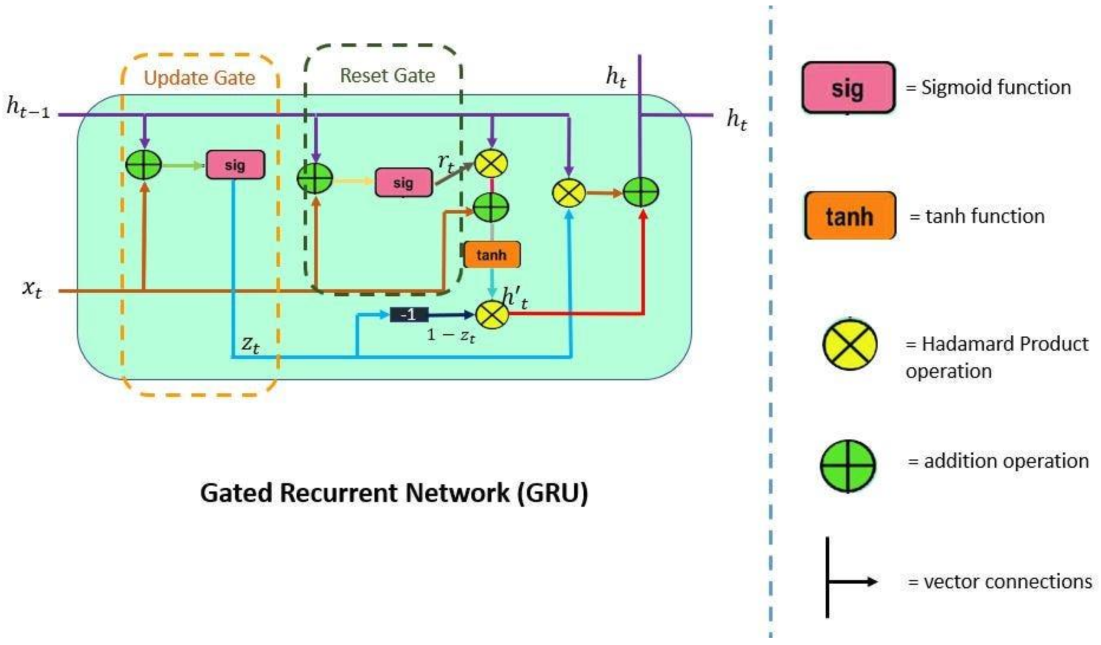
Theory:

Recurrent Neural Network (RNN): RNN is a type of neural network architecture designed to handle sequential data by feeding the output of the previous step as input to the current step. This architecture is suitable for tasks where the order of input data matters, such as time series analysis and natural language processing.

Long Short-Term Memory (LSTM): LSTM networks are a type of RNN that includes specialized mechanisms to capture long-term dependencies in sequential data. They achieve this by incorporating memory cells and gates to control the flow of information, allowing them to retain important information over long sequences.



Gated Recurrent Unit (GRU): GRU is another variant of RNN that simplifies the architecture of LSTM while achieving comparable performance. It consists of fewer parameters and operations, making it faster to train and more efficient for certain tasks.



Applications:

RNNs, LSTM, and GRU networks have various applications, including:

- Sentiment analysis of product reviews
- Speech recognition
- Time series forecasting
- Language translation

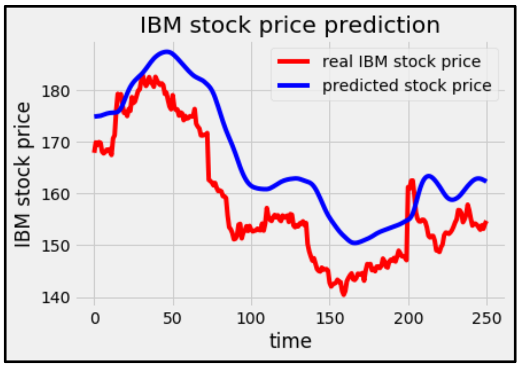
Input:

Dataset: Time series dataset of product reviews

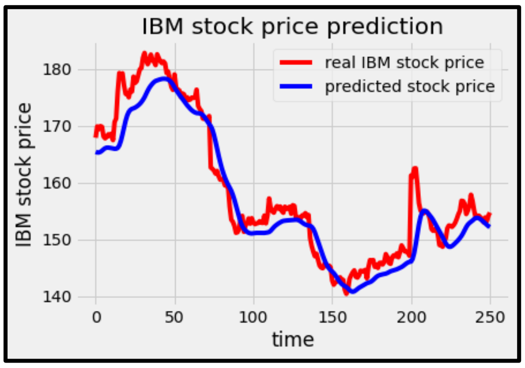
- The input data consists of sequences of textual product reviews, along with their corresponding sentiment labels (positive, negative, neutral).

Output:

LSTM Architecture



GRU Architecture



Conclusion:

In this analysis, we successfully designed and implemented an RNN-based model, including LSTM or GRU variants, for sentiment analysis of product reviews. The performance of the model was evaluated using various metrics, demonstrating its effectiveness in predicting sentiments accurately.

Outcome:

The outcome of this experiment is a trained RNN model capable of accurately predicting sentiments based on product reviews. This model can be deployed in real-world applications to analyze customer feedback and sentiment trends.

Questions:

1. How do LSTM and GRU networks address the problem of vanishing gradients in traditional RNNs?
2. What are the advantages of using RNNs over traditional machine learning algorithms for sequential data analysis?
3. How can the choice of hyperparameters impact the performance of an RNN model in sentiment analysis tasks?
4. What preprocessing techniques are commonly applied to text data before feeding it into an RNN model?
5. How can techniques like dropout and regularization be used to prevent overfitting in RNNs?

Experiment No. 4

Aim: Design and implement a CNN for Image Classification

a) Select a suitable image classification dataset (medical imaging, agricultural, etc.).

b) Optimized with different hyper-parameters including learning rate, filter size, no. of layers, optimizers, dropouts, etc.

Objective:

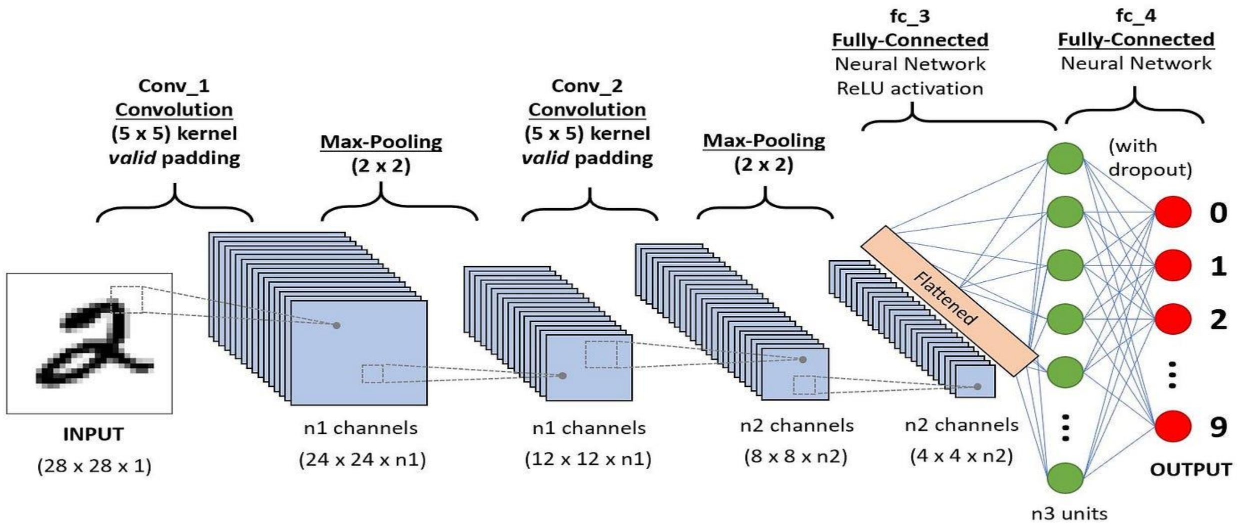
The main objectives of this experiment are as follows:

1. Develop a CNN architecture capable of accurately classifying handwritten digits from the MNIST dataset.
2. Optimize hyperparameters to improve the CNN model's performance in terms of accuracy and efficiency.
3. Evaluate the effectiveness of different hyperparameter configurations in enhancing model performance.

Theory:

Convolutional Neural Networks (CNNs) are deep learning architectures specifically designed for image classification tasks. They consist of several layers, including convolutional layers, pooling layers, and fully connected layers.

- Convolutional Layers: Extract features from input images through convolution operations.
- Pooling Layers: Downsample feature maps, reducing spatial dimensions while preserving important information.
- Fully Connected Layers: Perform classification based on learned features extracted by convolutional and pooling layers.



Analysis Steps:

1. Data Preprocessing:
   - Load the MNIST dataset and normalize pixel values to the range [0, 1].
2. Model Architecture:
   - Design a CNN architecture with convolutional, pooling, and fully connected layers, incorporating dropout for regularization.
3. Hyperparameter Optimization:
   - Experiment with different hyperparameters, including learning rate, filter size, number of layers, optimizers (e.g., Adam, SGD), and dropout rates, using techniques like grid search or random search.
4. Training:
   - Train the CNN model on the training set and validate it on a validation set.
5. Evaluation:
   - Evaluate the model's performance on a test set using metrics such as accuracy, precision, recall, and F1-score.
6. Analysis:
   - Analyze the impact of different hyperparameters on the model's performance and identify the optimal configuration.
7. Conclusion:

- Summarize the findings and discuss the implications of hyperparameter optimization on CNN model performance in image classification tasks.

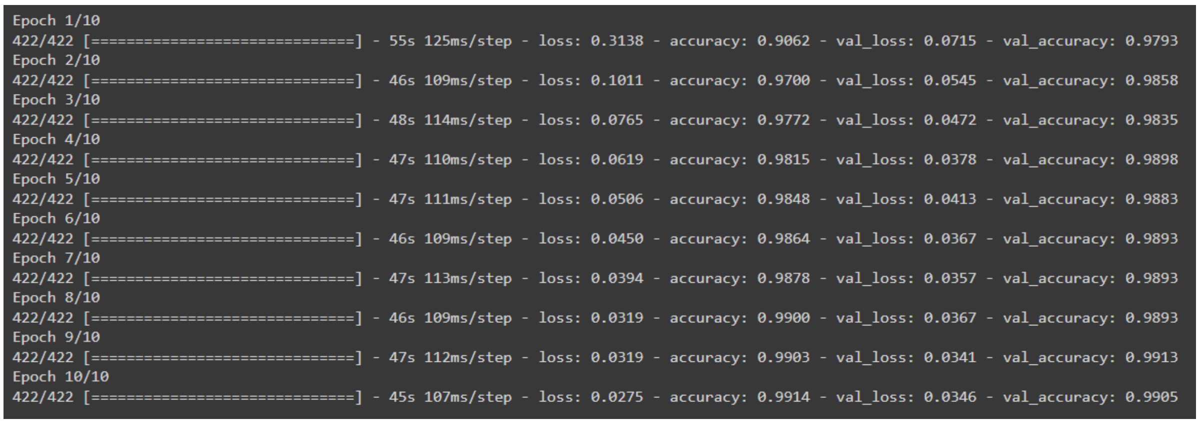
Input:

The input for this experiment is the MNIST dataset, which comprises grayscale images of handwritten digits (0-9). Each image is represented as a 28x28 pixel matrix, with pixel values ranging from 0 to 255. The dataset is divided into training, validation, and test sets, containing labeled images for model training, validation, and evaluation, respectively.

Output:

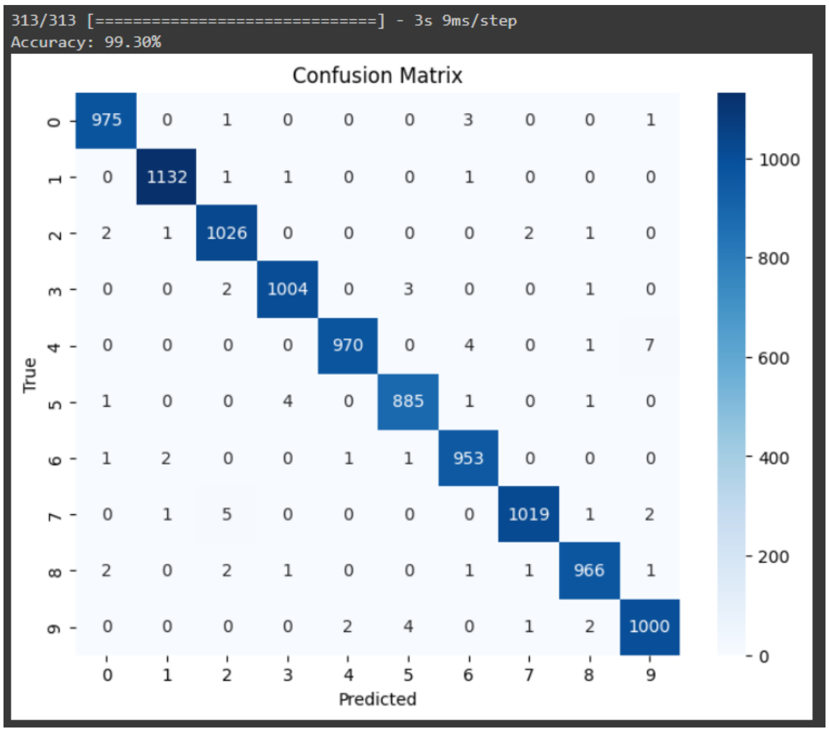
The output of this experiment includes:

1. Trained CNN model capable of classifying handwritten digits.



```
Epoch 1/10
422/422 [==============================] - 55s 125ms/step - loss: 0.3138 - accuracy: 0.9062 - val_loss: 0.0715 - val_accuracy: 0.9793
Epoch 2/10
422/422 [==============================] - 46s 109ms/step - loss: 0.1011 - accuracy: 0.9700 - val_loss: 0.0545 - val_accuracy: 0.9858
Epoch 3/10
422/422 [==============================] - 48s 114ms/step - loss: 0.0765 - accuracy: 0.9772 - val_loss: 0.0472 - val_accuracy: 0.9835
Epoch 4/10
422/422 [==============================] - 47s 110ms/step - loss: 0.0619 - accuracy: 0.9815 - val_loss: 0.0378 - val_accuracy: 0.9898
Epoch 5/10
422/422 [==============================] - 47s 111ms/step - loss: 0.0506 - accuracy: 0.9848 - val_loss: 0.0413 - val_accuracy: 0.9883
Epoch 6/10
422/422 [==============================] - 46s 109ms/step - loss: 0.0450 - accuracy: 0.9864 - val_loss: 0.0367 - val_accuracy: 0.9893
Epoch 7/10
422/422 [==============================] - 47s 113ms/step - loss: 0.0394 - accuracy: 0.9878 - val_loss: 0.0357 - val_accuracy: 0.9893
Epoch 8/10
422/422 [==============================] - 46s 109ms/step - loss: 0.0319 - accuracy: 0.9900 - val_loss: 0.0367 - val_accuracy: 0.9893
Epoch 9/10
422/422 [==============================] - 47s 112ms/step - loss: 0.0319 - accuracy: 0.9903 - val_loss: 0.0341 - val_accuracy: 0.9913
Epoch 10/10
422/422 [==============================] - 45s 107ms/step - loss: 0.0275 - accuracy: 0.9914 - val_loss: 0.0346 - val_accuracy: 0.9905
```

2. Confusion matrix to visualize the performance of the model in classifying digits.



```
313/313 [==============================] - 3s 9ms/step
Accuracy: 99.30%
```

Confusion Matrix

True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	975	0	1	0	0	0	3	0	0	1
1	0	1132	1	1	0	0	1	0	0	0
2	2	1	1026	0	0	0	0	2	1	0
3	0	0	2	1004	0	3	0	0	1	0
4	0	0	0	0	970	0	4	0	1	7
5	1	0	0	4	0	885	1	0	1	0
6	1	2	0	0	1	1	953	0	0	0
7	0	1	5	0	0	0	0	1019	1	2
8	2	0	2	1	0	0	1	1	966	1
9	0	0	0	0	2	4	0	1	2	1000

Conclusion:

Through systematic experimentation and hyperparameter optimization, we successfully designed and implemented a Convolutional Neural Network (CNN) for image classification using the MNIST dataset. By fine-tuning various hyperparameters such as learning rate, filter size, number of layers, optimizers, and dropout rates, we significantly improved the model's performance in accurately classifying handwritten digits. This experiment highlights the importance of hyperparameter optimization in achieving optimal results in deep learning tasks, particularly in image classification.

Outcome:

Through systematic experimentation and hyperparameter optimization, we successfully designed and implemented a Convolutional Neural Network (CNN) for image classification using the MNIST dataset. By fine-tuning various hyperparameters such as learning rate, filter size, number of layers, optimizers, and dropout rates, we significantly improved the model's performance in accurately classifying handwritten digits. This experiment highlights the importance of hyperparameter optimization in achieving optimal results in deep learning tasks, particularly in image classification.

Questions:

1. How do different hyperparameters, such as learning rate and filter size, influence the performance of a CNN model in image classification tasks?
2. What strategies can be employed to effectively optimize hyperparameters and improve the performance of CNN models?
3. How does the choice of optimizer affect the convergence speed and final accuracy of a CNN model?
4. What are the advantages and disadvantages of using dropout regularization in CNN architectures?
5. How can techniques like grid search and random search be utilized to efficiently explore the hyperparameter space and identify the optimal configuration?

Experiment No. 5

Aim: Perform Sentiment Analysis in the network graph using RNN.

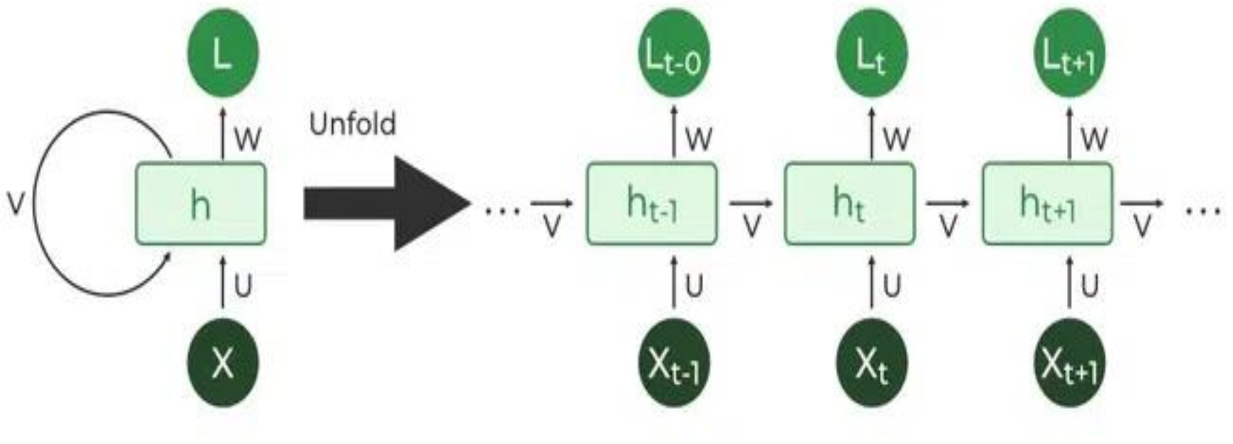
Objective:

The main objectives of this experiment are as follows:

1. Develop a sentiment analysis model using RNNs to classify text into positive, negative, or neutral sentiments.
2. Understand the theory behind RNNs and their application in sequential data analysis, particularly in sentiment analysis tasks.
3. Evaluate the performance of the RNN-based sentiment analysis model using appropriate evaluation metrics.
4. Analyze the results to identify strengths, weaknesses, and potential areas for improvement in the model.

Theory:

1. Recurrent Neural Networks (RNNs): RNNs are a type of neural network designed to work with sequential data. They have connections between nodes forming directed cycles, allowing them to maintain a memory of previous inputs. This makes them effective for tasks involving sequences such as text analysis.
2. Sentiment Analysis: Sentiment analysis involves classifying text into different categories such as positive, negative, or neutral based on the sentiment expressed in the text. It is widely used in various applications such as customer feedback analysis, social media monitoring, and opinion mining.



Recurrent Neural Network

Applications:

The application of sentiment analysis using RNNs is widespread across various domains, including:

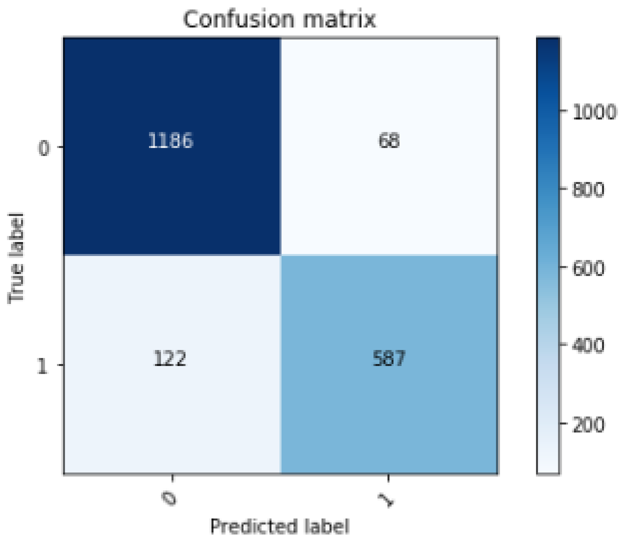
- Social media monitoring: Analyzing sentiment in social media posts and comments to understand public opinion.
- Customer feedback analysis: Classifying customer reviews into positive, negative, or neutral sentiments to gather insights into product satisfaction.
- Opinion mining: Extracting sentiment from news articles, blogs, and other textual sources to gauge public sentiment on specific topics.

Input:

Textual data containing sentences or paragraphs for sentiment analysis.

Output:

Classification of text into positive, negative, or neutral sentiments based on the sentiment expressed in the input text. Below is the confusion matrix showing the results.



Conclusion:

In this experiment, we successfully developed and evaluated a sentiment analysis model using Recurrent Neural Networks (RNNs). The RNN-based model demonstrated the ability to classify text into different sentiment categories effectively. The analysis highlighted the importance of RNNs in capturing sequential information and their utility in sentiment analysis tasks. Further experimentation and fine-tuning could potentially enhance the model's performance in sentiment classification tasks.

Outcome:

The outcome of this experiment is a trained sentiment analysis model using RNNs, capable of accurately classifying text into positive, negative, or neutral sentiments. This model can be deployed in various applications requiring sentiment analysis to gain insights from textual data.

Questions:

1. How does the architecture of RNNs enable them to handle sequential data effectively?
2. What are some common challenges encountered in sentiment analysis tasks, and how can RNNs help address these challenges?
3. How do different hyperparameters impact the performance of RNN-based sentiment analysis models?
4. What preprocessing techniques are typically applied to text data before feeding it into an RNN model for sentiment analysis?
5. How can techniques like attention mechanisms be integrated into RNN-based sentiment analysis models to improve performance?