Seminar and Progress Report (EC-456)

On

Technology Used in "HealthSure"

Submitted in partial fulfillment of the requirements
for the degree of

Bachelor of Technology
in
Electronics and Communication Engineering

By

Aman Sagar
Enrollment No. – 00416412819

Priyanshu Gautam
Enrollment No. – 06216412819

Under the provision of

Dr. Virendra Prasad Vishwakarma
(Professor, USIC&T)

Signature

DECLARATION

I hereby declare that matter embodied in the Seminar and Progress report entitled "Technology used in HealthSure" submitted to **USICT, GGSIPU** is a record of original work done by **Aman Sagar,** under the guidance and supervision of **Dr. Virendra Prasad Vishwakarma** and it has not formed the basis for the award of any Degree/Diploma/Associateship/Fellowship or other similar title to any candidate of any University.

CERTIFICATE

This is certify that **Mr. Aman Sagar** registered in the **B.Tech (ECE)** programme of **USICT, GGSIPU** is permitted to carry out the proposed work presented in this seminar and progress report under my guidance. The matter emboided in this proposal has not been submitted earlier.

DATE :

Signature of Supervisor
(Dr. Virendra Prasad Vishwakarma)

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude and appreciation for your guidance and support througout the development of my seminar and progress report on HealthSure website build using python and flask. Your expert knowledge and insights have been instrumental in helping me understand the complexities of building HealthSure website and implementing various features. I also appreciate your patience and encouragement as we navigated through various challenges and learned new concepts. Your commitment to my learning and development as aspiring software developers is greatly appreciated. Thank you for being an exceptional mentor and for helping me achieve the successfull completion of my major project.

Sincerely,
Aman Sagar

Table of Contents

Abstract

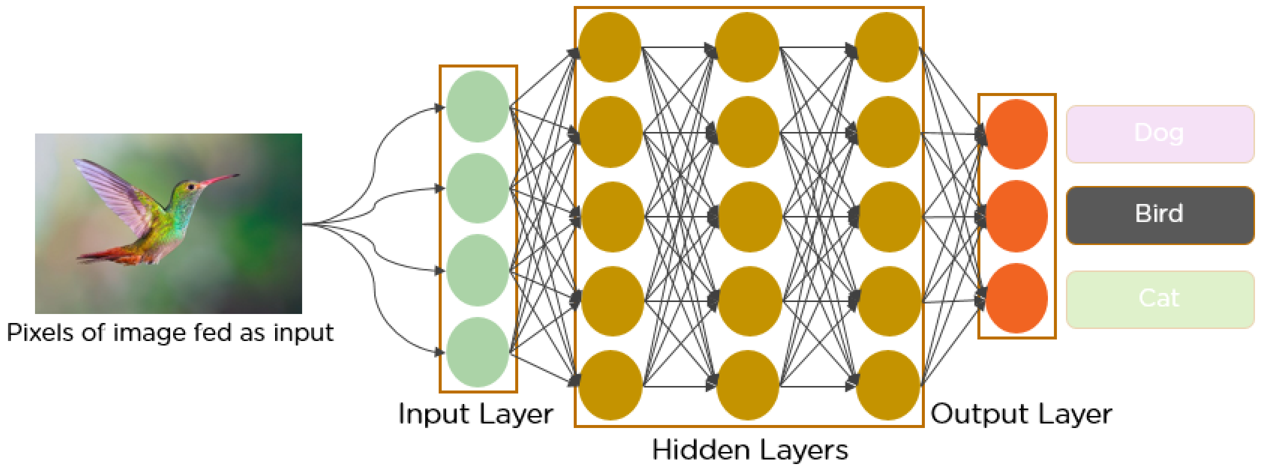
Convolutional Neural Network is deep learning algorithms for image processing. This report provides comprehensive knowledge about deep understanding of CNN, how to implement, how can we increase it's accuracy and what are its application and limitations. On the other hand we have used various python libraries for building our project. Flask for routing our application and host our model, predict the ouput. The integration of CNNs and Flask involves several steps, including model training, model serialization, and incorporating the model into a Flask application. Flask's routing system facilitates the creation of endpoints that receive image data, pass it through the CNN model, and return the processed results to the user. Flask's templating engine enables the dynamic generation of web pages, making it possible to display CNN predictions, visualizations, or interactive elements.

This abstract provides a brief overview of the capabilities and integration of CNNs and Flask. By leveraging the power of CNNs for image processing and combining it with the flexibility of Flask for web development, developers can create intelligent and interactive web applications that can handle visual data effectively. The integration of CNNs and Flask holds great potential for various domains, including e-commerce, healthcare, content moderation, and more, where intelligent web applications are becoming increasingly important.

Introduction to Convolutional neural network

The field of Deep Learning has materialized a lot over the past few decades due to efficiently tackling massive datasets and making computer systems capable enough to solve computational problems

Hidden layers have ushered in a new era, with the old techniques being non-efficient, particularly when it comes to problems like Pattern Recognition, Object Detection, Image Segmentation, and other image processing-based problems. CNN is one of the most deployed deep learning neural networks.

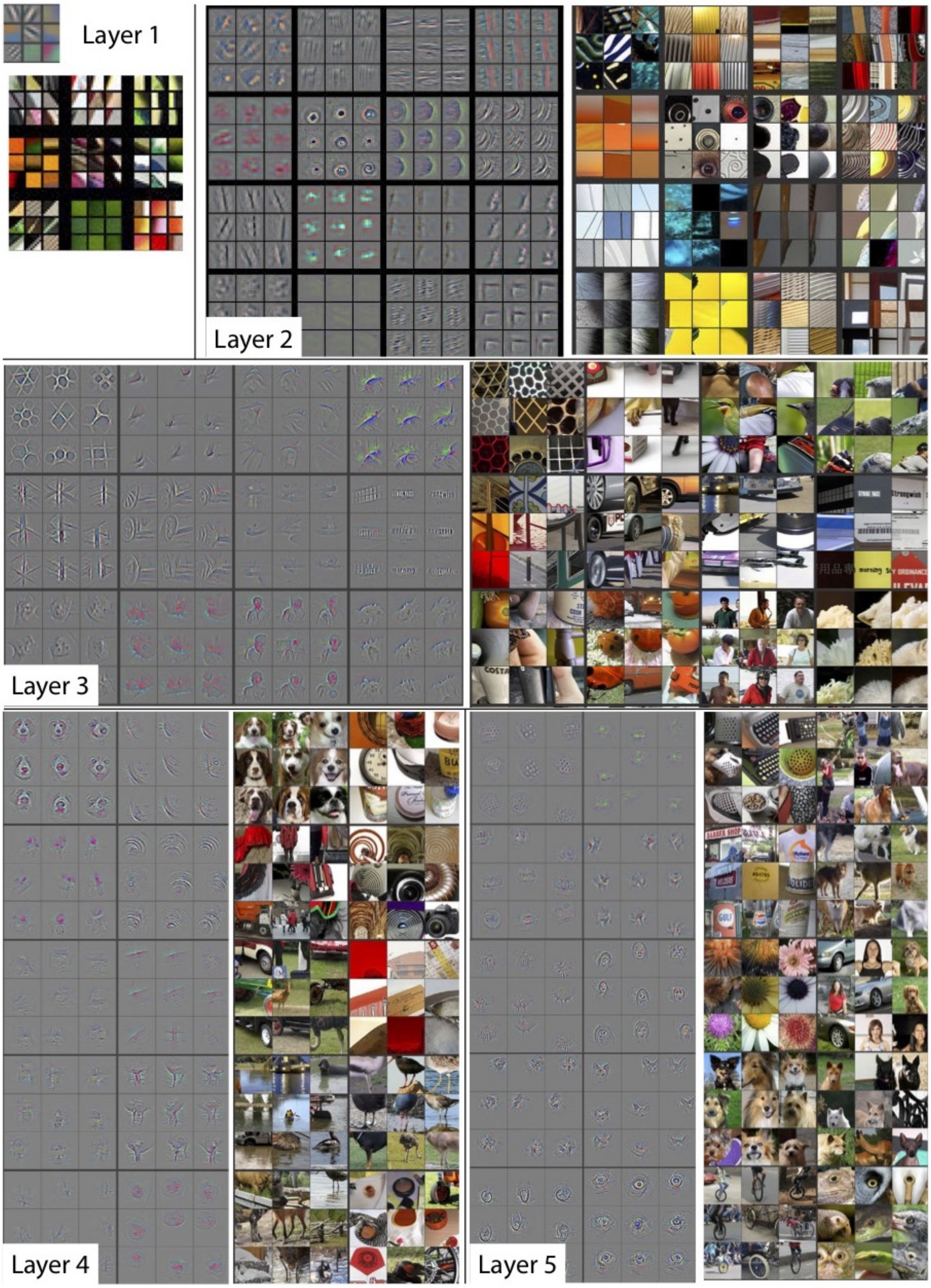


What is CNN?

In the field of deep learning, convolutional neural network (CNN) is among the class of deep neural networks, which was being mostly deployed in the field of analyzing/image recognition.

Convolutional Neural uses a very special kind of method which is being known as Convolution.

The mathematical definition of convolution is a mathematical operation being applied on the two functions that give output in a form of a third function that shows how the shape of one function is being influenced, modified by the other function.


Layer 1
Layer 2
Layer 3
Layer 4
Layer 5

The Convolutional neural networks(CNN) consists of various layers of artificial neurons. Artificial neurons, similar to that neuron cells that are being used by the human brain for passing various sensory input signals and other responses, are mathematical functions that are being used for calculating the sum of various inputs and giving output in the form of an activation value.

The behaviour of each CNN neuron is being defined by the value of its weights. When being fed with the values (of the pixel), the artificial neurons of a CNN recognizes various visual features and specifications.

When we give an input image into a CNN, each of its inner layers generates various activation maps. Activation maps point out the relevant features of the given input image. Each of the CNN neurons generally takes input in the form of a group/patch of the pixel, multiplies their values(colours) by the value of its weights, adds them up, and input them through the respective activation function.

The first (or maybe the bottom) layer of the CNN usually recognizes the various features of the input image such as edges horizontally, vertically, and diagonally.

The output of the first layer is being fed as an input of the next layer, which in turn will extract other complex features of the input image like corners and combinations of edges.

The deeper one moves into the convolutional neural network, the more the layers start detecting various higher-level features such as objects, faces, etc

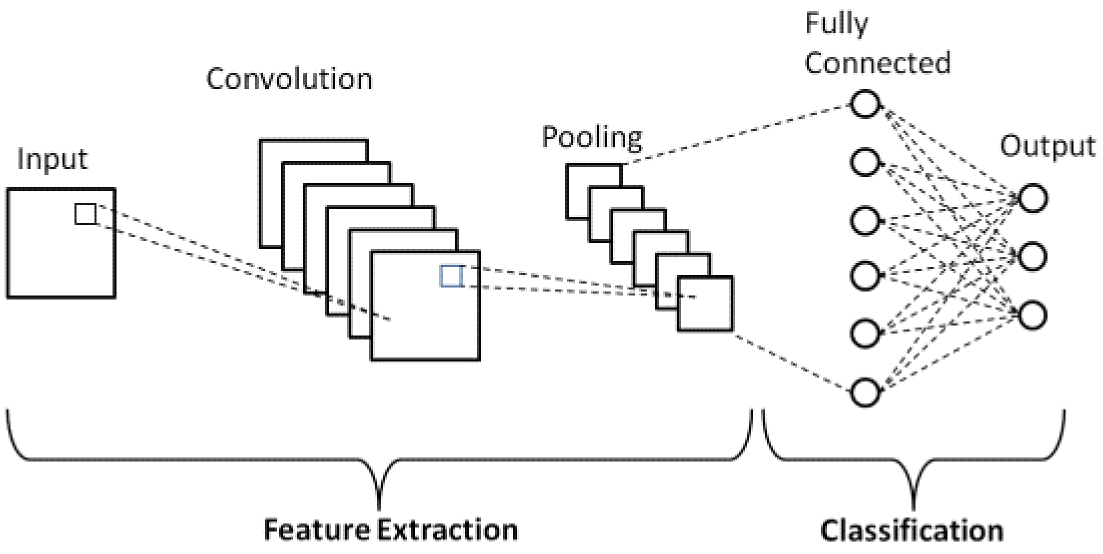
CNN’s Basic Architecture

A CNN architecture consists of two key components:

• A convolution tool that separates and identifies the distinct features of an image for analysis in a process known as Feature Extraction

• A fully connected layer that takes the output of the convolution process and predicts the image's class based on the features retrieved earlier.

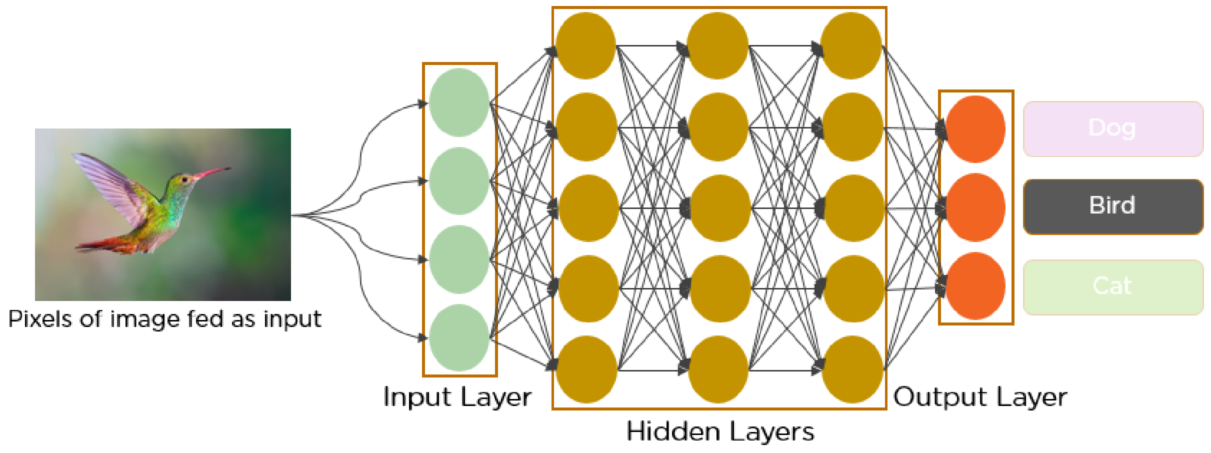
The CNN is made up of three types of layers: convolutional layers, pooling layers, and fully-connected (FC) layers.



This article was published as a part of the Data Science Blogathon.

The field of Deep Learning has materialized a lot over the past few decades due to efficiently tackling massive datasets and making computer systems capable enough to solve computational problems

Hidden layers have ushered in a new era, with the old techniques being non-efficient, particularly when it comes to problems like Pattern Recognition, Object Detection, Image Segmentation, and other image processing-based problems. CNN is one of the most deployed deep learning neural networks.



Python Code implementation for Implementing CNN for classification

Importing Relevant Libraries

```
import NumPy as np
%matplotlib inline
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import TensorFlow as tf
tf.compat.v1.set_random_seed(2019)
```

Loading MNIST Dataset

```
(X_train,Y_train),(X_test,Y_test) = keras.datasets.mnist.load_data()
```

Scaling The Data

```
X_train = X_train / 255
X_test = X_test / 255


#flatenning
```

```
X_train_flattened = X_train.reshape(len(X_train), 28*28)

X_test_flattened = X_test.reshape(len(X_test), 28*28)
```

Designing The Neural Network

```
model = keras.Sequential([

     keras.layers.Dense(10, input_shape=(784,), activation='sigmoid')

])

model.compile(optimizer='adam',

                    loss='sparse_categorical_crossentropy',

                    metrics=['accuracy'])

model.fit(X_train_flattened, Y_train, epochs=5)
```

Applications of CNN

Image Classification: CNNs are widely used for image classification tasks. They can learn to recognize patterns and features in images, enabling them to classify objects, scenes, or people with high accuracy.

1. Object Detection: CNNs can be used for object detection tasks, where they not only classify objects but also locate their positions within an image. This is commonly used in autonomous vehicles, surveillance systems, and robotics.
2. Image Segmentation: CNNs can segment images into different regions or objects. This is useful in medical imaging for identifying and analyzing specific areas of interest, such as tumors, organs, or blood vessels.

3. Facial Recognition: CNNs have been applied to facial recognition tasks, enabling systems to identify and verify individuals based on their facial features. This technology is used in security systems, biometric authentication, and social media tagging.
4. Natural Language Processing (NLP): CNNs have been adapted for text classification tasks in NLP. They can process word embeddings or character-level representations of text, allowing them to perform sentiment analysis, spam detection, or topic classification.
5. Video Analysis: CNNs can analyze videos by processing frames sequentially. They can be used for action recognition, video summarization, and video captioning. CNNs have also been employed in surveillance systems to detect abnormal activities or track objects in real-time.
6. Medical Diagnosis: CNNs are utilized in medical diagnosis tasks, such as identifying diseases from medical images (e.g., X-rays, MRIs), analyzing histopathology slides, or predicting the progression of diseases based on patient data.
7. Autonomous Driving: CNNs play a crucial role in autonomous driving systems, including tasks such as lane detection, object detection and tracking, traffic sign recognition, and pedestrian detection. They help vehicles perceive the surrounding environment and make informed decisions.
8. Robotics: CNNs are used in robotics for tasks like object recognition, grasping and manipulation, and robot navigation. They enable robots to perceive and understand the environment, enhancing their autonomy and interaction capabilities.
9. Artistic Style Transfer: CNNs have been employed for artistic style transfer, allowing users to transform the style of an image or video based on the characteristics of famous artworks. This application has gained popularity in the field of digital art and creative expression.

Limitations of CNN

Translation Invariance: CNNs are designed to be translation invariant, meaning they can recognize objects regardless of their position in the input image. However, this property can also be a limitation when the spatial relationships between objects are important. CNNs may struggle to capture fine-grained details or precise object boundaries.

1. Lack of Global Context: CNNs typically have a local receptive field, meaning they only consider a small neighborhood of pixels at a time. This local processing may

result in limited access to global contextual information, making it challenging to understand the overall context of an image or scene.

2. Parameter Sensitivity: CNNs have a large number of parameters, and they are sensitive to their values. Small changes in the parameters can significantly affect the network's performance, making them sensitive to hyperparameter tuning and requiring extensive training on large datasets.
3. Limited Robustness to Variations: CNNs are sensitive to variations in input data, such as changes in scale, rotation, illumination, or occlusions. While techniques like data augmentation can mitigate some of these issues, CNNs may struggle to generalize well to unseen variations that differ significantly from the training data.
4. High Computational Requirements: CNNs can be computationally intensive, especially for deeper architectures with a large number of layers. Training and evaluating CNN models require significant computational resources, limiting their deployment on resource-constrained devices or in real-time applications.
5. Data Dependency: CNNs require a large amount of labeled training data to learn representative features and generalize well. Acquiring and annotating large-scale datasets can be time-consuming, expensive, and may not always be feasible, especially for niche or specialized domains.
6. Limited Interpretability: CNNs are often referred to as "black box" models because they lack interpretability. Understanding how and why CNNs make certain predictions can be challenging, hindering their use in critical applications where interpretability and transparency are essential.
7. Domain Specificity: CNNs trained on one domain may not generalize well to other domains. Transfer learning and domain adaptation techniques can help address this limitation, but still, there may be a need for significant fine-tuning or retraining on target domain data.

Increasing Accuracy

Increase the depth of the network by adding more convolutional and fully connected layers.

1. Increase the width of the network by increasing the number of filters in each convolutional layer.
2. Use more advanced architectures like ResNet, Inception, or DenseNet, which have demonstrated improved performance in various tasks.
3. Implement skip connections or residual connections to enable better flow of gradients and alleviate the vanishing gradient problem.

4. Use techniques like batch normalization to improve the stability and convergence of the model.
5. Experiment with different activation functions like ReLU, Leaky ReLU, or ELU to improve the model's non-linearity.
6. Incorporate regularization techniques like dropout or L1/L2 regularization to prevent overfitting and improve generalization.
7. Perform data augmentation by applying transformations such as rotations, translations, and flips to artificially increase the diversity of the training data.
8. Collect and incorporate more labeled training data to provide a larger and more diverse set of examples for the model to learn from.
9. Use transfer learning by initializing the CNN with pre-trained weights from a model that has been trained on a large dataset and then fine-tune it on your specific task.
10. Optimize hyperparameters such as learning rate, batch size, and optimizer choice through grid search or random search to find the best combination for your specific problem.
11. Train the model for more epochs to allow it to converge to a better solution, but be cautious of overfitting and monitor the performance on a validation set.

Training Model of CNN

1.Data Preprocessing:

- Load and preprocess your dataset. This may involve tasks such as resizing images, normalizing pixel values, and splitting the data into training and validation sets.

2. Design the CNN Architecture:

- Choose the appropriate CNN architecture for your task, such as VGG, ResNet, or Inception. Consider the depth, number of layers, filter sizes, and activation functions based on your problem and available computational resources.

3. Initialize the Model:

- Initialize the CNN model using the chosen architecture. This involves creating the layers, defining the connections between them, and setting the appropriate hyperparameters.

4. Define the Loss Function:

- Select a suitable loss function based on your task, such as categorical cross-entropy for multi-class classification or mean squared error for regression. The loss function quantifies the difference between the predicted and actual outputs.

5. Choose an Optimization Algorithm:
    - Select an optimization algorithm, such as Stochastic Gradient Descent (SGD), Adam, or RMSprop, to update the model's weights during training. The optimizer minimizes the loss function by adjusting the network's parameters.
6. Training Loop:
    - Iterate over the training data for a fixed number of epochs (passes through the entire dataset). Within each epoch:
        - Forward Propagation: Pass the input data through the network to obtain predicted outputs.
        - Compute Loss: Compare the predicted outputs with the ground truth labels and calculate the loss using the chosen loss function.
        - Backward Propagation: Calculate the gradients of the loss with respect to the model's parameters, using techniques such as backpropagation.
        - Update Parameters: Use the optimization algorithm to update the model's weights and biases, based on the computed gradients.
        - Repeat the above steps until all training samples have been processed.
7. Evaluate Model Performance:
    - After training, evaluate the model's performance on the validation set or a separate test set. Calculate metrics such as accuracy, precision, recall, or mean squared error to assess the model's performance.
8. Fine-tuning and Hyperparameter Tuning:
    - If necessary, fine-tune the model by adjusting hyperparameters, such as learning rate, batch size, regularization techniques, or dropout rates. Use techniques like cross-validation to find the optimal values.
9. Deployment and Prediction:
    - Once you are satisfied with the model's performance, deploy it for making predictions on new, unseen data. Preprocess the input data in the same way as during training and use the trained model to predict the outputs.

INTRODUCTION TO FLASK AND IT"S FEATURES

Flask is a lightweight web framework in Python that is commonly used for building web applications and APIs. It provides a simple and flexible way to handle HTTP requests and responses, making it a popular choice for deploying machine learning models.

When it comes to model deployment, Flask can be used to create a web server that exposes an API endpoint for your model. This allows users to send data to the server, which then passes the data to the model for prediction. Flask handles the communication between the client and the server, making it easy to integrate your model into a web application or make it accessible over the internet.

Flask provides a set of features that are beneficial for model deployment, such as:

1. Routing: Flask allows you to define routes to handle different types of requests. For example, you can create a route that accepts POST requests and sends the data to your model for prediction.
2. Request Handling: Flask provides tools to handle incoming requests, allowing you to extract data and perform any necessary preprocessing before passing it to the model.
3. Response Generation: Flask makes it easy to generate responses to client requests. You can format the model's predictions as JSON, HTML, or any other desired format before sending it back to the client.
4. Integration: Flask can be easily integrated with other Python libraries and frameworks, making it straightforward to combine your model with other functionality, such as data persistence or visualization.
5. Scalability: Flask's lightweight nature allows for easy deployment to a variety of hosting options, including cloud platforms. It can handle multiple simultaneous requests efficiently, making it suitable for production environments.

Overall, Flask simplifies the process of deploying machine learning models by providing a framework for building the server-side components necessary for model inference. It allows you to create a RESTful API that can be accessed by clients to interact with your model, making it an excellent choice for deploying CNN models or any other type of machine learning model.

FLASK ROUTING

Prepare the model: Train your CNN model using a suitable framework like TensorFlow or PyTorch. Save the trained model along with its weights to disk. You can use the model's respective APIs to accomplish this.

1. Set up Flask: Create a new Flask project by setting up a new virtual environment and installing Flask using pip. Initialize the Flask application and create a new route to handle requests.
2. Load the model: In your Flask application, load the saved model and its weights. This can be done by using the appropriate framework's APIs to load the model from the saved files.
3. Preprocess the input: Define a preprocessing function to preprocess the input data before feeding it to the CNN model. This may include resizing the image, normalizing pixel values, or any other required preprocessing steps.
4. Create an API endpoint: In your Flask application, define an API endpoint to receive the input data for prediction. This can be done by creating a new route using the @app.route decorator. Inside the route function, preprocess the input data and pass it to the loaded model for prediction.
5. Handle the prediction: Receive the preprocessed data in the route function and pass it to the CNN model for prediction. Get the predictions from the model and format the output as desired.
6. Return the response: Return the predictions as a response to the API request. You can choose to return the results as JSON, HTML, or any other suitable format.
7. Run the Flask application: Finally, run the Flask application locally or deploy it to a server. You can use a web server like Gunicorn or uWSGI to serve the Flask application.

These steps provide a general overview of deploying a CNN model using Flask. You may need to adapt them based on your specific model and requirements. Flask provides a flexible framework to build web applications, and you can add more functionality and features as needed.

what is flask in terms of model deployment

Flask is a lightweight web framework in Python that is commonly used for building web applications and APIs. It provides a simple and flexible way to handle HTTP requests and responses, making it a popular choice for deploying machine learning models.

When it comes to model deployment, Flask can be used to create a web server that exposes an API endpoint for your model. This allows users to send data to the server, which then passes the data to the model for prediction. Flask handles the communication between the client and the server, making it easy to integrate your model into a web application or make it accessible over the internet.

Flask provides a set of features that are beneficial for model deployment, such as:

1. Routing: Flask allows you to define routes to handle different types of requests. For example, you can create a route that accepts POST requests and sends the data to your model for prediction.
2. Request Handling: Flask provides tools to handle incoming requests, allowing you to extract data and perform any necessary preprocessing before passing it to the model.
3. Response Generation: Flask makes it easy to generate responses to client requests. You can format the model's predictions as JSON, HTML, or any other desired format before sending it back to the client.
4. Integration: Flask can be easily integrated with other Python libraries and frameworks, making it straightforward to combine your model with other functionality, such as data persistence or visualization.
5. Scalability: Flask's lightweight nature allows for easy deployment to a variety of hosting options, including cloud platforms. It can handle multiple simultaneous requests efficiently, making it suitable for production environments.

Overall, Flask simplifies the process of deploying machine learning models by providing a framework for building the server-side components necessary for model inference. It allows you to create a RESTful API that can be accessed by clients to interact with your model, making it an excellent choice for deploying CNN models or any other type of machine learning model.

References

- Introduction to CNN
  https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-convolutional-neural-network-in-deep-learning/
- Training CNN Model
  https://towardsdatascience.com/training-a-convolutional-neural-network-from-scratch-2235c2a25754
- Deployment using Flask
  https://towardsdatascience.com/model-deployment-using-flask-c5dcbb6499c9
- https://www.analyticsvidhya.com/blog/2021/10/applications-of-convolutional-neural-networkscnn/
- https://anjanisuman.medium.com/limitation-of-nn-and-cnn-ee21a4cdc9eb