FOR SOFTWARE STUDENTS

Lab: Deploying an Image Recognition Model to a Local Production Environment

Objective:

1. Guide students in deploying an image recognition model in a local production environment.
2. Teach the process of serving the model via a REST API.
3. Containerize the application using Docker to ensure portability and ease of deployment.
4. Introduce best practices for local deployment, including model updates and monitoring.

Skills Developed:

- Deploying image recognition models.
- Serving models through REST APIs.
- Containerizing applications using Docker.
- Model monitoring and logging in production.
- Managing model updates in production environments.

Required Tools & Technologies:

- **Python Development Environment** (IDE like VSCode, Jupyter Notebook, or PyCharm).
- **Pre-trained Image Recognition Model** (e.g., a convolutional neural network like ResNet, VGG, or custom-trained model).
- **Flask** or **FastAPI** to serve the model via a REST API.
- **Docker** for containerizing the application.
- **OpenCV** or **Pillow** for image preprocessing.
- **Local server** or **VM** for deployment simulation.

Dataset:

- Use the pre-trained model of **Lab 4**

Lab Steps:

Step 1: Prepare the Image Recognition Model

1. **Train or Use a Pre-Trained Model**:
    - If you have a pre-trained model, save it to a file. If not, use a model like ResNet or VGG, and train it on a suitable image dataset.
    - Save the model in a format that can be loaded later (e.g., .pth, .h5).

Step 2: Create the API Using Flask/FastAPI

1. **Set Up a Web Framework**:
    - Choose between Flask or FastAPI to build the API for serving the model.
    - Install the necessary libraries like Flask/FastAPI, image processing libraries, and PyTorch/TensorFlow (depending on the model used).
2. **Load the Pre-trained Model**:
    - In the API, write the logic to load the saved model and ensure it's in evaluation mode (not training mode).
3. **Create an Endpoint for Image Prediction**:
    - Create an API endpoint (e.g., /predict) that accepts POST requests with an image file.
    - Implement image preprocessing (e.g., resizing, normalization) to convert the input image into the format expected by the model.
    - Use the loaded model to predict the image class and return the result as a JSON response.

Step 3: Dockerize the Application

1. **Create a Dockerfile**:
    - Write a Dockerfile that sets up the environment for your API.
    - The Dockerfile should install the necessary Python dependencies, copy the app code, and specify how to run the app inside the container.
2. **Build the Docker Image**:
    - Build the Docker image from the Dockerfile to create a portable containerized version of the application.
3. **Run the Docker Container**:
    - Once the image is built, run the container, ensuring that the application is accessible on a specified port.

Step 4: Deploy the Application Locally

1. **Deploy on a Local Machine or Virtual Machine**:
    - Deploy the Docker container on a local machine or VM that simulates a production environment.
    - Ensure that Docker is installed and running on the target machine.
    - Test the deployed model by sending requests to the API from any local machine.
2. **Test the Model API**:
    - Use tools like **Postman** or **curl** to send POST requests to the model's API endpoint and verify that the model is correctly predicting the class of the images.

Step 5: Monitoring and Logging

1. **Add Logging to the Application**:
    - Implement basic logging to track errors or important events (e.g., model prediction times, error handling).

   ○ Log incoming requests and predictions made by the model to ensure traceability and transparency in production.
2. **Monitor API Usage**:
   ○ In a real production environment, set up tools to monitor API usage, such as logging response times and tracking failures.

Step 6: Model Updates and Versioning

1. **Model Versioning**:
   ○ Keep different versions of your model (e.g., model_v1.pth, model_v2.pth).
   ○ When updating the model, replace the old model with the new one and restart the server to apply the changes.
2. **Deploying Model Updates**:
   ○ After updating the model, ensure that the new model is correctly loaded into the API.
   ○ Perform a test to confirm that the new model works as expected before fully transitioning to production.

Expected Outcomes:

- Students will deploy an image recognition model in a local production environment.
- They will create a REST API for image recognition and test it locally.
- The application will be containerized using Docker and deployed locally for testing.
- Students will learn how to monitor and log the application in a production environment and manage model updates.

Assessment Criteria:

- Successful deployment of the API with a working image recognition model.
- The model should be served via a REST API and accessible from a local environment.
- The application should be containerized using Docker.
- Basic logging and monitoring should be implemented.

Lab Duration:

The lab will take approximately **4-5 hours**, depending on the students' familiarity with Docker and model deployment

FOR NETWORKING STUDENTS

Lab: Deploying and Testing an Anomaly Detection Model in a Network Simulation

Objective:

1. Guide students in deploying an anomaly detection model that identifies abnormal behavior in network traffic.
2. Implement the model in a simulated network environment to detect network anomalies such as intrusion attempts, malicious traffic, or unusual patterns.
3. Teach the process of testing the model in a controlled network simulation environment.

Skills Developed:

- Network traffic simulation and analysis.
- Building and deploying anomaly detection models for network security.
- Model evaluation and performance testing in simulated environments.
- Use of unsupervised learning for anomaly detection in cybersecurity.

Required Tools & Technologies:

- **Python Development Environment** (IDE like VSCode, Jupyter Notebook, or PyCharm).
- **Scikit-learn** for machine learning.
- **PyShark** or **Scapy** for network traffic capture and simulation.
- **Anomaly detection algorithm** (e.g., Isolation Forest, DBSCAN, One-Class SVM, or Autoencoders).
- **Local network simulation tools** like **GNS3**, **Mininet**, or **Cisco Packet Tracer**.
- **Network traffic datasets** (e.g., **NSL-KDD**, **CICIDS**, or **UNSW-NB15**).
- **Flask/FastAPI** for serving the anomaly detection model if needed.

Lab Steps:

Step 1: Prepare the Anomaly Detection Model

1. **Train the Anomaly Detection Model**:(Use the model of Lab 4)
   -

Step 2: Simulate Network Traffic

1. **Set Up a Network Simulation Environment**:
   - Use a local network simulation tool like **Mininet**, **GNS3**, or **Cisco Packet Tracer** to create a simulated network with multiple devices (e.g., routers, switches, servers, clients).
   - Set up a few different types of network traffic, such as normal traffic (e.g., HTTP, FTP, DNS) and abnormal traffic (e.g., DDoS, port scanning, SQL injection attempts).
2. **Generate Traffic Data**:
   - Use traffic generation tools like **Iperf**, **Ostinato**, or **Scapy** to generate synthetic network traffic.
   - Ensure that there is a mix of normal and abnormal traffic to test the model's ability to identify anomalies.
3. **Capture Network Traffic**:
   - Use tools like **Wireshark** or **Tcpdump** to capture the network packets from the simulated network.
   - Export the captured network traffic to a file (e.g., pcap or CSV format).

Step 3: Deploy the Anomaly Detection Model

1. **Load the Pre-Trained Anomaly Detection Model**:
   - If the model is already trained, load the model in the script or API.
   - If needed, write a Flask or FastAPI service to serve the trained model as a REST API for testing.
2. **Run Anomaly Detection on Captured Traffic**:
   - Feed the captured network traffic data into the anomaly detection model.
   - Use the model to classify network traffic as normal or abnormal. The model should output an anomaly score or label indicating whether the traffic is suspicious.
3. **Evaluate the Model**:
   - Compare the model's predictions against known anomalies in the test dataset.
   - Evaluate its performance based on metrics like precision, recall, and F1 score. Adjust the model if necessary to reduce false positives and negatives.

Step 4: Test and Visualize Results

1. **Test the Model on New Traffic**:
   - Use new, unseen network traffic to test the model's ability to detect anomalies in real-time.

  - Generate additional synthetic traffic, introducing new types of attacks (e.g., SYN floods, DNS tunneling, malware traffic) to see if the model can adapt.
2. **Visualize the Results**:
  - Plot the results of the anomaly detection, such as the number of anomalies detected, and any trends in the traffic data (e.g., volume of traffic over time).
  - Use visualization tools like **Matplotlib** or **Seaborn** to show the distribution of normal vs. abnormal traffic.

Step 5: Monitor and Improve the Model

1. **Monitor Model Performance in the Simulation**:
  - Monitor the model's performance on real-time network traffic.
  - Track the number of true positives, false positives, true negatives, and false negatives over time.
2. **Improve the Model**:
  - Based on performance metrics, consider fine-tuning the model by adjusting hyperparameters or retraining with new data (e.g., including more diverse attack types).
  - If the model's performance is subpar, try different anomaly detection algorithms, such as Autoencoders or k-Means clustering.
3. **Deploy the Model to a Real Environment** (Optional):
  - For students who are ready, discuss the steps for deploying the anomaly detection model to a real network environment (e.g., integrating it into a security monitoring system).

Expected Outcomes:

- Students will deploy an anomaly detection model that can identify abnormal network behavior.
- They will test the model's performance in a simulated network environment with various types of traffic, including attacks.
- Students will visualize and analyze the model's results, gaining hands-on experience with anomaly detection in cybersecurity.

Assessment Criteria:

- The successful deployment of the model in a simulated network environment.
- The model should be able to detect and classify anomalies with a reasonable degree of accuracy.
- Clear understanding of the performance of the model, including its ability to distinguish normal traffic from attack traffic.
- Well-documented results with visualizations demonstrating model performance.

Lab Duration:

The lab will take approximately **5-6 hours**, depending on the complexity of the network simulation and the dataset size.