**DEEP LEARNING ASSIGNMENT**

**Question 1 -**

Implement 3 different CNN architectures with a comparison table for the MNSIT dataset using the Tensorflow library. Note - 1. The model parameters for each architecture should not be more than 8000 parameters 2. Code comments should be given for proper code understanding. 3. The minimum accuracy for each accuracy should be at least 96%

**Answer 1 DL:-**

**Question 2 -**

Implement 5 different CNN architectures with a comparison table for CIFAR 10 dataset using the PyTorch library Note - 1. The model parameters for each architecture should not be more than 10000 parameters 2 Code comments should be given for proper code understanding.

**Answer 2 DL:-**

**Question 3 -**

Train a Pure CNN with less than 10000 trainable parameters using the MNIST Dataset having minimum validation accuracy of 99.40% Note - 1. Code comments should be given for proper code understanding. 2. Implement in both PyTorch and Tensorflow respectively.

**Answer 3 DL:-**

**Question 4 -**

Design an end-to-end solution with diagrams for object detection use cases leveraging AWS cloud services and open-source tech Note -

1. You need to use both AWS cloud services and open-source tech to design the entire solution

2. The pipeline should consist of a data pipeline, ml pipeline, deployment pipeline, and inference pipeline.

3. In the data pipeline, you would be designing how to get the data from external or existing sources and tech used for the same

4. In the ml pipeline, you would be designing how to train the model, and what all algorithms, techniques, etc. would you be using. Again, tech used for the same

5. Since this is a deep learning project, the use of GPUs, and how effectively are you using them to optimize for cost and training time should also be taken into consideration.

6. In the deployment pipeline, you would be designing how effectively and efficiently you are deploying the model in the cloud,

7. In the inference pipeline, consider the cost of inference and its optimization related to computing resources and handling external traffic

8. You can use any tool to design the architecture

9. Do mention the pros and cons of your architecture and how much further it can be optimized and its tradeoffs.

10. Do include a retraining approach as well.

11. Try to include managed AWS resources for deep learning like AWS Textract, AWS Sagemaker, etc., and not just general-purpose compute resources like S3, EC2, etc. Try to mix the best of both services.

**Answer 4 DL:-**

To design an end-to-end solution for object detection use cases leveraging AWS cloud services and open-source tech, we can create a comprehensive architecture that includes a data pipeline, ML pipeline, deployment pipeline, and inference pipeline. Here's an overview of the solution architecture:

Let's go through each pipeline component and the technologies used:

1. Data Pipeline:

- External or existing data sources: This can include datasets from various sources like open datasets, custom labeled datasets, or data generated from IoT devices.

- Data storage: Use AWS S3 to store the input data in a scalable and durable manner. S3 provides high availability, durability, and supports integrations with other AWS services.

- Data preprocessing: Utilize open-source tech like OpenCV or TensorFlow's data preprocessing capabilities to perform data augmentation, resizing, normalization, and other necessary preprocessing steps.

- Data annotation: Use open-source annotation tools like LabelImg or RectLabel to manually label the objects in the images and generate bounding box annotations.

- Annotation storage: Store the annotations in a structured format, such as JSON or XML, in S3 or a database for easy access during training.

2. ML Pipeline:

- Training infrastructure: Utilize AWS SageMaker for training. SageMaker provides managed Jupyter notebooks, distributed training capabilities, and supports popular deep learning frameworks like TensorFlow and PyTorch.

- Model training: Train the object detection model using a deep learning framework like TensorFlow or PyTorch, utilizing popular architectures like YOLO, SSD, or Faster R-CNN. Leverage GPUs available in SageMaker instances to accelerate training.

- Model evaluation: Evaluate the trained model using metrics like mean Average Precision (mAP), Intersection over Union (IoU), or any other appropriate evaluation metrics.

- Model versioning: Keep track of different versions of the trained models and store them in S3 or a model registry for easy retrieval and retraining.

3. Deployment Pipeline:

- Model deployment: Use AWS SageMaker's model hosting capabilities to deploy the trained model as an API endpoint. This allows for scalable and serverless inference.

- Auto scaling: Configure the SageMaker endpoint to automatically scale up or down based on the incoming request traffic, optimizing cost and resource utilization.

- Monitoring and logging: Utilize AWS CloudWatch to monitor the health and performance of the deployed model. Capture and analyze logs for troubleshooting and performance optimization.

4. Inference Pipeline:

- Inference infrastructure: Utilize AWS Lambda and API Gateway to create a serverless API endpoint for handling inference requests.

- Cost optimization: Implement caching mechanisms to reduce redundant inference requests and utilize Amazon CloudFront to cache and serve static assets like images and models, reducing the overall cost of inference.

- Load balancing: Implement Elastic Load Balancing (ELB) to distribute incoming inference requests across multiple instances, ensuring high availability and scalability.

Retraining Approach:

- Retraining trigger: Implement a retraining trigger mechanism based on predefined conditions like performance degradation, new data availability, or a scheduled retraining interval.

- Incremental learning: Utilize transfer learning techniques to speed up retraining by initializing the model with the previous weights and fine-tuning it on the new data.

- Automated pipeline: Automate the retraining process by triggering the data pipeline, ML pipeline, and deployment pipeline sequentially to update the model and deploy the latest version seamlessly.

Pros, Cons, and Trade-offs:

- Pros:

- Utilizes scalable and managed AWS services, reducing the need for infrastructure management.

- Supports efficient data storage, preprocessing, annotation, and training.

- Enables cost optimization through resource scaling, caching, and load balancing.

- Provides a

serverless and scalable inference pipeline.

- Offers flexibility to incorporate the best open-source deep learning frameworks and algorithms.

- Cons and trade-offs:

- AWS services may come with costs, so careful optimization and cost management are required.

- Integration of multiple services may require additional effort for setup and configuration.

- Real-time inference with low-latency may require further optimization and caching strategies.

Further Optimization:

- Use AWS AutoML services like Amazon Rekognition to explore pre-trained models for object detection and fine-tuning with custom datasets.

- Implement distributed training with multiple instances or distributed frameworks like Horovod to accelerate training and reduce training time.

- Implement model quantization or compression techniques to reduce model size and improve inference performance on edge devices.

- Utilize AWS Lambda and AWS Batch for serverless batch inference on large datasets.

Remember that the architecture and its optimization depend on specific use cases, requirements, and constraints. Adjustments may be required based on factors such as dataset size, complexity, real-time requirements, and cost considerations.

**Question 5 -**

In Question 4, you have designed the architecture for an object detection use case leveraging AWS Cloud, similarly, here you will be designing for Document Classification use case leveraging Azure Cloud services. Note - 1. Most of the points are the same as in Question 4, just cloud services will change.

**Answer 5 DL:-**

Certainly! Here's a design for a Document Classification use case leveraging Azure Cloud services. The architecture consists of a data pipeline, ML pipeline, deployment pipeline, and inference pipeline. Let's dive into the details:

1. Data Pipeline:

- Data ingestion: Fetch data from external or existing sources, such as file systems, databases, or APIs.

- Data storage: Utilize Azure Blob Storage to store the document data securely and durably.

- Data preprocessing: Perform text preprocessing tasks, such as tokenization, normalization, and removal of stop words, using open-source libraries like NLTK or SpaCy.

- Data labeling: Manually label the documents or leverage pre-existing labeled datasets for supervised training.

- Annotation storage: Store the document labels or annotations in Azure Blob Storage or Azure Table Storage.

2. ML Pipeline:

- Training infrastructure: Utilize Azure Machine Learning service (AML) to manage the end-to-end ML workflow. AML provides capabilities for data preparation, model training, and deployment.

- Model training: Train the document classification model using machine learning algorithms like Naive Bayes, Support Vector Machines (SVM), or deep learning models like LSTM or Transformer.

- Model evaluation: Evaluate the trained model using metrics such as accuracy, precision, recall, or F1 score.

- Model versioning: Utilize AML to track different versions of the trained models and manage their lifecycle.

3. Deployment Pipeline:

- Model deployment: Deploy the trained model using Azure Container Instances (ACI) or Azure Kubernetes Service (AKS). ACI is suitable for low-scale deployments, while AKS provides scalability and flexibility for larger workloads.

- Autoscaling: Configure AKS to automatically scale up or down based on the incoming inference traffic, optimizing cost and resource utilization.

- Monitoring and logging: Utilize Azure Monitor and Azure Log Analytics to monitor the deployed model's performance, health, and resource usage.

4. Inference Pipeline:

- Inference infrastructure: Utilize Azure Functions to create serverless functions for handling inference requests.

- Cost optimization: Implement caching mechanisms to reduce redundant inference requests and utilize Azure Content Delivery Network (CDN) for caching and serving static assets, reducing overall inference costs.

- Load balancing: Leverage Azure Traffic Manager or Azure Front Door to distribute incoming inference requests across multiple instances, ensuring high availability and scalability.

Retraining Approach:

- Retraining trigger: Implement a retraining trigger mechanism based on predefined conditions like performance degradation, new labeled data availability, or a scheduled retraining interval.

- Incremental learning: Use transfer learning techniques to speed up retraining by initializing the model with previous weights and fine-tuning on new labeled data.

- Automated pipeline: Automate the retraining process using Azure Data Factory or Azure Logic Apps to trigger the data pipeline, ML pipeline, and deployment pipeline sequentially for seamless updates.

This architecture leverages Azure services like Azure Blob Storage for data storage, Azure Machine Learning for model training and deployment, Azure Container Instances or Azure Kubernetes Service for model deployment, and Azure Functions for serverless inference.

Pros, Cons, and Trade-offs:

- Pros:

- Utilizes scalable and managed Azure services, reducing infrastructure management overhead.

- Provides a seamless end-to-end ML workflow with Azure Machine Learning service.

- Offers flexibility to incorporate various ML algorithms and techniques for document classification.

- Enables cost optimization through autoscaling, caching, and load balancing mechanisms.

- Cons and trade-offs:

- Azure services may come with costs, so careful optimization and cost management are required.

- Integration of multiple services may require additional effort for setup and configuration.

- Real-time inference with

low-latency may require further optimization and caching strategies.

Further Optimization:

- Utilize Azure Cognitive Services like Azure Text Analytics or Azure Form Recognizer for document classification and extraction tasks, leveraging pre-trained models.

- Implement distributed training using Azure Machine Learning service with multiple nodes or Azure Databricks to accelerate training and reduce training time.

- Implement model quantization or compression techniques to reduce model size and improve inference performance on edge devices.

- Utilize Azure Logic Apps or Azure Data Factory to automate the retraining process based on specific triggers.

Remember to adapt the architecture based on your specific use case requirements, dataset size, complexity, real-time requirements, and cost considerations.