JHANVI PAREKH
60009210033
CSE(DS)

Experiment 9
(Data Science Project Architecture)

Aim: To implement Data Science Project Architecture.

Theory:

DevOps (Development Operations):

DevOps is a set of practices, principles, and cultural philosophies that aim to enhance collaboration and communication between software development (Dev) and IT operations (Ops) teams. The goal is to automate the processes of software development, testing, and infrastructure management to deliver high-quality software more quickly and reliably.

- Code Building: Managing the process of compiling and building code into executable files.
- Continuous Integration (CI): Integrating code changes into a shared repository frequently and automatically.
- Continuous Deployment (CD): Automating the deployment of code changes to production environments.
- Infrastructure as Code (IaC): Managing and provisioning infrastructure through code to ensure consistency.
- Configuration Management: Maintaining and updating system configurations across different environments.
- Monitoring and Logging: Continuous monitoring of application and system performance with logging and alerting.
- Collaboration: Encouraging collaboration between development and operations teams to enhance communication.

MLOps (Machine Learning Operations):

- Data Management: Ensuring efficient data acquisition, cleaning, and storage for machine learning models.
- Model Training: Automating the training of machine learning models using diverse datasets.
- Model Deployment: Deploying models into production environments and managing their lifecycle.
- Monitoring and Logging: Continuous monitoring of model performance, logging, and alerting for any anomalies.
- Version Control: Managing versions of both code and machine learning models to track changes.
- Collaboration: Facilitating collaboration between data scientists, engineers, and other stakeholders in the ML workflow.
- Automation: Automating repetitive tasks to streamline the ML pipeline and reduce manual errors.

Department of Computer Science and Engineering (Data Science)

Difference between MLOps and DevOps:

- **Focus:**

MLOps: Primarily focuses on the machine learning lifecycle, including model development, training, and deployment.
DevOps: Primarily focuses on the overall software development lifecycle, from code writing to deployment and operations.

- **Nature of Artifacts:**

MLOps: Involves artifacts like machine learning models, datasets, and experimentation logs.
DevOps: Involves artifacts like source code, binaries, and configuration files.

- **Workflow:**

MLOps: Involves specialized steps for data pre-processing, feature engineering, and model evaluation.
DevOps: Involves steps like code compilation, testing, and deployment.
Tools:
MLOps: Uses tools specific to machine learning, such as TensorFlow, PyTorch, and specialized model deployment tools.
DevOps: Uses a broader set of tools for version control, CI/CD, infrastructure management, like Git, Jenkins, Docker, and Kubernetes.

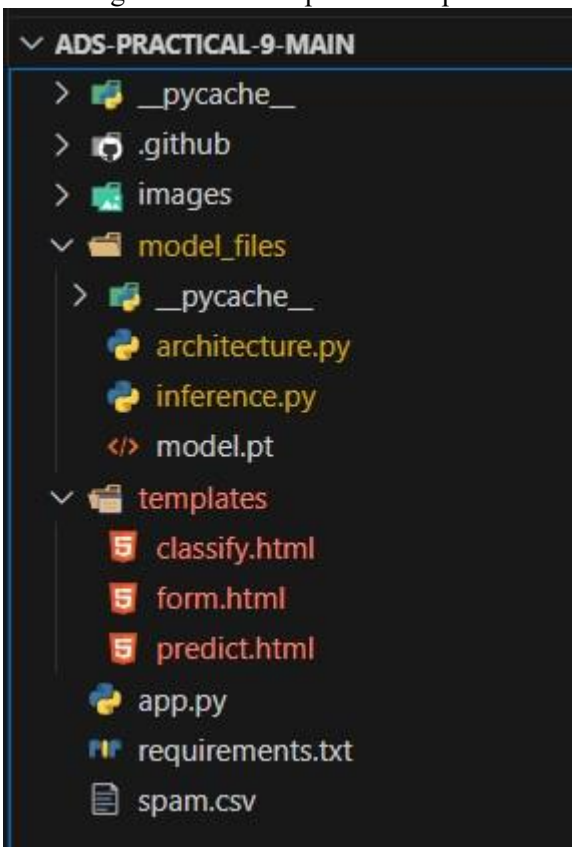
Collaboration:
Both MLOps and DevOps emphasize collaboration between different teams involved in the software development and deployment processes. Collaboration ensures better communication, faster feedback loops, and more effective problem-solving.

Scalability:
MLOps: Focuses on the scalability of machine learning workflows, ensuring that models can handle larger datasets and diverse environments.
DevOps: Focuses on the scalability of software applications, infrastructure, and deployment processes to handle increased loads and user demands.

Reusability:
MLOps: Encourages the reuse of machine learning components, models, and workflows to save time and resources.
DevOps: Encourages the reuse of code, configurations, and deployment scripts to maintain consistency across different environments and applications.

Lab Assignment:
Using any dataset relevant to data science project implement in data preprocessing, feature engineering, and model training on.
The chosen machine learning model will be serialized and deployed through a Flask or FastAPI REST API endpoint, containerized using Docker for consistency. The assignment emphasizes MLOps practices by integrating the project into a CI/CD pipeline (e.g., GitHub Actions).

Furthermore, students will implement basic monitoring and logging functionalities to track the model's performance in a production-like environment. Clear and comprehensive documentation covering development environment setup, model training, deployment, and monitoring is an integral part of the evaluation. This assignment offers a hands-on, practical introduction to MLOps concepts while addressing the real-world problem of predictive maintenance.



Department of Computer Science and Engineering (Data Science)

architecture.py 1 | inference.py 1 | app.py | classify.html 1 | form.html 1 | predict.html 1

model_files > architecture.py > ...

```python
import io
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import transforms
from PIL import Image
from pathlib import Path

# architecture
class Net(nn.Module):
    """Contains model architecture class and two helper functions.
        get_model(): Loads the trainded model
        get_tensor(): Persorms transforms on the input image
    """
    def __init__(self):
        super(Net, self).__init__()
        #convolutional layer
        self.conv1 = nn.Conv2d(1, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        # max pooling layer
        self.pool = nn.MaxPool2d(2, 2)
        # fully connected layers
        self.fc1 = nn.Linear(64 * 7 * 7, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 64)
        self.fc4 = nn.Linear(64, 10)
        # dropout
        self.dropout = nn.Dropout(p=.5)

    def forward(self, x):
        # add sequence of convolutional and max pooling layers
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        # flatten image input
        x = x.view(-1, 64 * 7 * 7)
        # add hidden layer, with relu activation function
        x = self.dropout(F.relu(self.fc1(x)))
        x = self.dropout(F.relu(self.fc2(x)))
        x = self.dropout(F.relu(self.fc3(x)))
        x = F.log_softmax(self.fc4(x), dim=1)

        return x


def get_model():
    """Returns loaded model."""
    chechpoint = Path('model_files/model.pt')
    model = Net()
    model.load_state_dict(torch.load(chechpoint, map_location='cpu'), strict=False)
    model.eval()
    return model

def get_tensor(image_bytes):
    """Returns transformed image."""
    transform = transforms.Compose([transforms.Resize((28,28)),
                                    transforms.ToTensor()])
    image = Image.open(io.BytesIO(image_bytes)).convert('L') # image_bytes are what we get from web request then grays the image
    return transform(image).unsqueeze(0) # sends a single image
```

Department of Computer Science and Engineering (Data Science)

architecture.py 1 | inference.py 1 | app.py | classify.html 1 | form.html 1 | predict.html 1

model_files > inference.py > get_image_label

```python
import io
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import transforms
from PIL import Image
from pathlib import Path
# architecture
class Net(nn.Module):
    """Contains model architecture class and two helper functions.
        get_model(): Loads the trainded model
        get_tensor(): Persorms transforms on the input image
    """
    def __init__(self):
        super(Net, self).__init__()
        #convolutional layer
        self.conv1 = nn.Conv2d(1, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        # max pooling layer
        self.pool = nn.MaxPool2d(2, 2)
        # fully connected layers
        self.fc1 = nn.Linear(64 * 7 * 7, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 64)
        self.fc4 = nn.Linear(64, 10)
        # dropout
        self.dropout = nn.Dropout(p=.5)

    def forward(self, x):
        # add sequence of convolutional and max pooling layers
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        # flatten image input
        x = x.view(-1, 64 * 7 * 7)
        # add hidden layer, with relu activation function
        x = self.dropout(F.relu(self.fc1(x)))
        x = self.dropout(F.relu(self.fc2(x)))
        x = self.dropout(F.relu(self.fc3(x)))
        x = F.log_softmax(self.fc4(x), dim=1)
        return x

def get_model():
    """Returns loaded model."""
    chechpoint = Path('model_files/model.pt')
    model = Net()
    model.load_state_dict(torch.load(chechpoint, map_location='cpu'), strict=False)
    model.eval()
    return model

def get_tensor(image_bytes):
    """Returns transformed image."""
    transform = transforms.Compose([transforms.Resize((28,28)),
                                    transforms.ToTensor()])
    image = Image.open(io.BytesIO(image_bytes)).convert('L') # image_bytes are what we get from web request then grays the image
    return transform(image).unsqueeze(0) # sends a single image

def get_image_label(image_bytes):
    tensor = get_tensor(image_bytes)
    output = model.forward(tensor)
    _, pred = torch.max(output, 1)

    return pred.item()
model = get_model()
```

Department of Computer Science and Engineering (Data Science)

architecture.py 1 | inference.py 1 | app.py | classify.html 1 | form.html 1 | predict.html 1

app.py > predict

```python
import pandas as pd
from flask import Flask, request, render_template
from model_files.inference import get_image_label

app = Flask(__name__)

@app.route("/")
def home():
    return """You redirect to following:
            <br>
            /form to access form
            <br>
            /predict to access spam classifier
            <br>
            /classify to access digit classifier"""

@app.route("/form", methods=["GET", "POST"])
def form():
    if request.method == "GET":
        return render_template("form.html")

    name = request.form.get("name")
    mail = request.form.get("mail")
    pass_ = "".join(["*" for _ in range(len(request.form.get("pass")))])
    return render_template("form.html", name=name, mail=mail, pass_=pass_)


@app.route("/predict", methods=["GET", "POST"])
def predict():
    if request.method == "GET":
        return render_template("predict.html")

    df = pd.read_csv("spam.csv", encoding="latin-1")
    df.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1, inplace=True)
    df["label"] = df["type"].map({"ham": 0, "spam": 1})
    X = df["text"]
    y = df["label"]

    from sklearn.feature_extraction.text import CountVectorizer
    from sklearn.model_selection import GridSearchCV

    cv = CountVectorizer()
    X = cv.fit_transform(X)
    from sklearn.model_selection import train_test_split

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=42
    )

    from sklearn.naive_bayes import MultinomialNB

    clf = MultinomialNB(alpha = 0.001, force_alpha=True)
    clf.fit(X_train, y_train)
    clf.score(X_test, y_test)

    message = request.form["message"]
    data = [message]
    vect = cv.transform(data).toarray()
    my_prediction = clf.predict(vect)
```

```
62
63  @app.route("/classify", methods=['GET', 'POST'])
64  def classify():
65      if request.method == 'GET':
66          return render_template('classify.html')
67
68      if request.method == 'POST':
69          file = request.files['image_file']
70          print(file)
71          image = file.read()
72          label = get_image_label(image_bytes=image)
73          label = str(label)
74          print(label)
75          return render_template('classify.html', label=label,)
76
77  if __name__ == "__main__":
78      app.run(debug=True)
79
```

architecture.py 1 | inference.py 1 | app.py | classify.html 1 | form.html 1 | predict.html 1

templates > classify.html > ...

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>MNIST CLASSIFIER</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
        integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
</head>

<body>
    <br>
    <h3 align="center">MNIST CLASSIFIER</h3>
    <br>
    <br>
    <div class="container" align="center">
        <h2>Upload Digit Image</h2>
        <br>
        <div class="container">
            <form method=POST enctype=multipart/form-data class="form-group">
                <input type="file" name="image_file" required>
                <button type="submit" class="btn btn-primary">Upload</button>
            </form>
        </div>
    </div>
    {% if label%}
    <br>
    <div class="container" align="center">
        <h4>Predicted Label: {{ label }}</h4>
    </div>
    {% endif %}

</body>

</html>
```

architecture.py 1 | inference.py 1 | app.py | classify.html 1 | form.html 1 | predict.html 1

templates > form.html > html > body > form

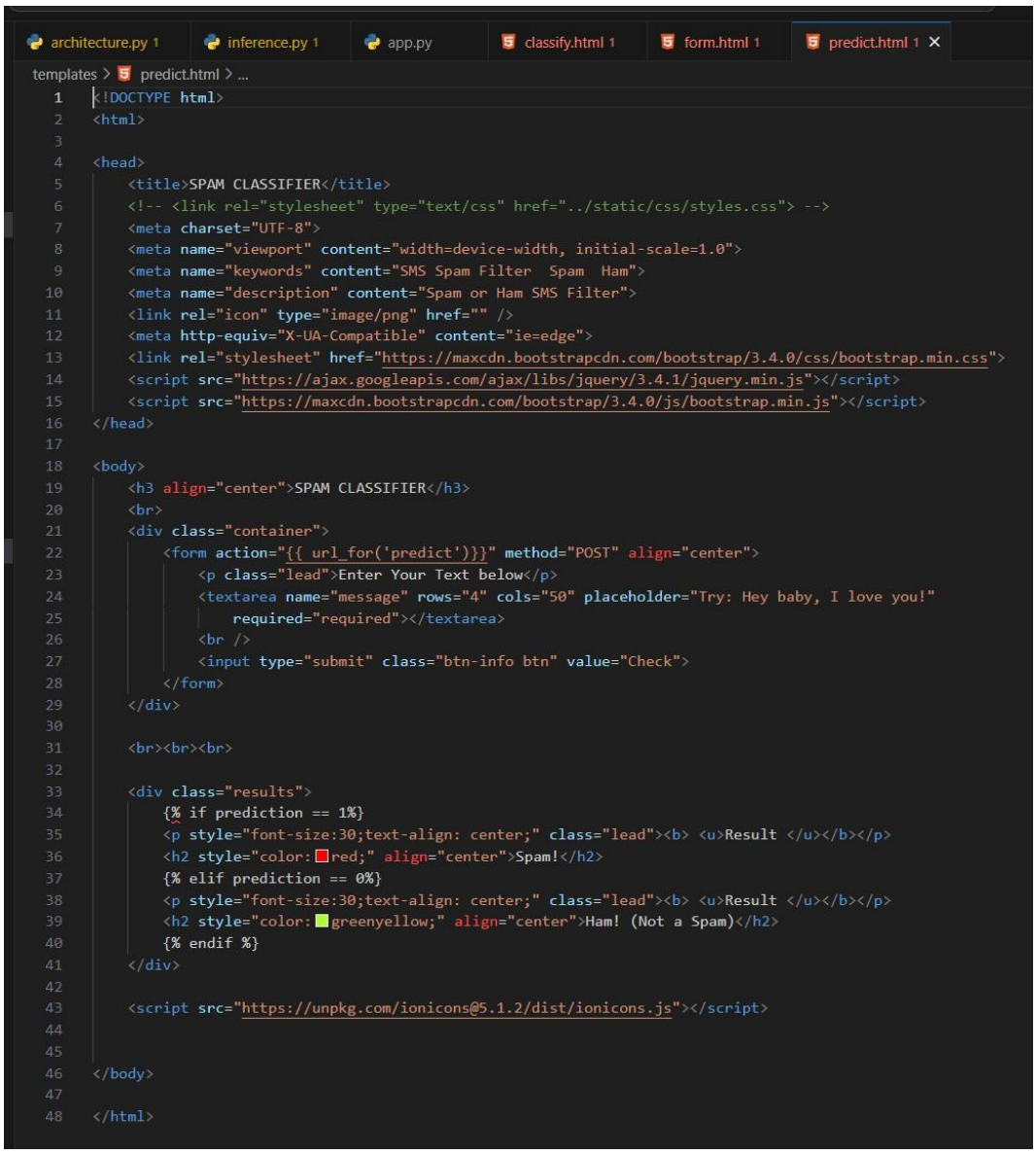
```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>FORM</title>
</head>

<body>
    <h3>LOGIN FORM</h3>
    <form action="/form" method="post">
        <label for="name">Name: </label>
        <input type="text" name="name" required>
        <br>
        <label for="mail">Email: </label>
        <input type="email" name="mail" required>
        <br>
        <label for="pass">Password: </label>
        <input type="password" name="pass" required>
        <br>
        <button type="submit">Submit</button>
    </form>

    {% if name %}
        <h4>Name: {{name}}</h4>
        <h4>Email: {{mail}}</h4>
        <h4>Password: {{pass_}}</h4>
    {%endif%}
</body>

</html>
```



architecture.py 1 | inference.py 1 | app.py | classify.html 1 | form.html 1 | predict.html 1

templates > predict.html > ...

```
<!DOCTYPE html>
<html>

<head>
    <title>SPAM CLASSIFIER</title>
    <!-- <link rel="stylesheet" type="text/css" href="../static/css/styles.css"> -->
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="keywords" content="SMS Spam Filter  Spam  Ham">
    <meta name="description" content="Spam or Ham SMS Filter">
    <link rel="icon" type="image/png" href="" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
</head>

<body>
    <h3 align="center">SPAM CLASSIFIER</h3>
    <br>
    <div class="container">
        <form action="{{ url_for('predict')}}" method="POST" align="center">
            <p class="lead">Enter Your Text below</p>
            <textarea name="message" rows="4" cols="50" placeholder="Try: Hey baby, I love you!"
                required="required"></textarea>
            <br />
            <input type="submit" class="btn-info btn" value="Check">
        </form>
    </div>

    <br><br><br>

    <div class="results">
        {% if prediction == 1%}
        <p style="font-size:30;text-align: center;" class="lead"><b> <u>Result </u></b></p>
        <h2 style="color: red;" align="center">Spam!</h2>
        {% elif prediction == 0%}
        <p style="font-size:30;text-align: center;" class="lead"><b> <u>Result </u></b></p>
        <h2 style="color: greenyellow;" align="center">Ham! (Not a Spam)</h2>
        {% endif %}
    </div>

    <script src="https://unpkg.com/ionicons@5.1.2/dist/ionicons.js"></script>


</body>

</html>
```