// Cover page

Task

Importing libraries:

1. **NumPy (np):** For working with numbers and arrays.
2. **Matplotlib (plt):** For plotting.
3. **PdfPages:** Saves Matplotlib plots into a neat and tidy PDF.
4. **scikit-learn (GaussianMixture):** Unveils hidden patterns in data using Gaussian Mixture Models

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.mixture import GaussianMixture
```

Task1:

This function (task1) simulates data corruption. It focuses on the data and creates a random mask with the same dimensions. This mask has 90% of its elements set to 0 (effectively masking out those pixels), simulating missing or unreliable data, and the remaining 10% are set to 1 (unmasked pixels). The function then returns this mask, which can be used later to corrupt the test data.

```python
def task1():
    # Load data from .npz file
    data = np.load('data.npz')

    # Extract train and test data
    train_data = data['train_data']
    test_data = data['test_data']

    # Generate corruption mask (90% pixel masking)
    S, H, W = test_data.shape
    mask = np.random.choice([0, 1], size=(S, H, W), p=[0.9, 0.1])

    # Return the corruption mask for further usage if needed
    return mask
```

fit_gmm()

This function (fit_gmm) focuses on learning the underlying structure of your training data. It does this by fitting a Gaussian Mixture Model (GMM) with a specified number of components (clusters) to the flattened training data. The function takes two arguments:

- Flattened training data: This represents training data where each data point is a single row vector.
- Number of components (K): This specifies the number of Gaussian distributions (clusters) used in the GMM.

```
def fit_gmm(x_train_flattened, K):
    # Fit a Gaussian Mixture Model (GMM) to the flattened training data
    gmm = GaussianMixture(n_components=K)
    gmm.fit(x_train_flattened)
    return gmm
```

Task2:

This function (task2) analyzes and visualizes the structure of training data using a Gaussian Mixture Model (GMM)

Here's the overall process:

- **Data Reshaping:** It reshapes the training data into a 2D array, flattening each image into a single row vector.
- **GMM Fitting and Visualization:** It calls the fit_gmm function to fit a GMM with K components to the flattened data. It then visualizes the means and covariances of each Gaussian component in the GMM.
- **GMM Parameter Retrieval:** It retrieves the means, covariances, and weights of the fitted GMM model.

- **Visualization Setup:** It creates a figure with subplots to display the means and covariances of each component.
- **Visualization Loop:** It iterates through each GMM component and displays its mean (as an image) and covariance matrix (as a grayscale image) in the corresponding subplots.
- **Overall, Title and Layout:** It adds a title indicating the number of components (K) used in the GMM and adjusts the layout for better presentation.
- **Output:** Finally, it displays the visualization and returns the fitted GMM model and the generated figure.

```python
def task2(train_data, K):
    # Reshape train data to 2D array (flatten each image)
    S, H, W = train_data.shape
    train_data_flattened = train_data.reshape(S, H * W)

    # Fit a GMM to the training data and visualize GMM components
    gmm = fit_gmm(train_data_flattened, K)

    # Obtain GMM parameters (means, covariances, and weights)
    means = gmm.means_
    covariances = gmm.covariances_
    weights = gmm.weights_

    # Determine image dimensions (assuming grayscale images)
    feature_dim = H * W

    # Visualize GMM components (means and covariances)
    num_components = len(means)
    fig, ax = plt.subplots(2, num_components, figsize=(2*num_components, 4))

    for k in range(num_components):
        ax[0, k].imshow(means[k].reshape(H, W), cmap='gray')
        ax[0, k].set_title(f'Mean {k+1}')

        # Reshape covariance matrix properly for visualization
        cov_matrix = covariances[k].reshape(feature_dim, feature_dim)
        ax[1, k].imshow(cov_matrix, cmap='gray')
        ax[1, k].set_title(f'Covariance {k+1}')

    plt.suptitle(f'GMM Components (K={K})', fontsize=16)
    plt.tight_layout()
    plt.show()

    return gmm, fig
```

conditional_gmm()

This function (conditional_gmm) utilizes a pre-trained GMM model (gmm) to analyze unseen test data. Here's the overall process:

- **Data Reshaping:** It reshapes the test data into a 2D array, flattening each image into a single row vector.
- **Posterior Probabilities:** It utilizes the fitted GMM model (gmm) to calculate the posterior probabilities for each test data point. These probabilities represent the likelihood of each test data point belonging to each Gaussian component in the GMM.
- **Posterior Expectations:** It computes the posterior expectations for the test data. This involves using the posterior probabilities and the means of each GMM component. The specific meaning of "posterior expectation" depends on your application, but it generally combines the information about the test data with the information captured by the GMM components.

```python
def conditional_gmm(gmm, test_data):
    # Reshape test data to 2D array (flatten each image)
    S, H, W = test_data.shape
    test_data_flattened = test_data.reshape(S, H * W)

    # Compute posterior probabilities for the test data
    posteriors = gmm.predict_proba(test_data_flattened)

    # Compute posterior expectations (using the means of GMM components)
    posterior_expectations = np.dot(posteriors, gmm.means_)

    return posterior_expectations
```

Task3:

This function (task3) focuses on applying a pre-trained GMM model to analyze and potentially recover corrupted test data. Here's the overall breakdown:

- **GMM Model Extraction:** It retrieves the pre-trained GMM model from the provided gmm_params.
- **Posterior Expectations:** It uses the conditional_gmm function to compute posterior expectations for the test data using the GMM model.
- **Visualization Setup:** It creates a figure with subplots to display (for each sample in the test data):
    1. The corrupted version of the test image (simulated with a mask).
    2. The computed posterior expectation for that image.
    3. The original, uncorrupted test image (assuming it's available).
- **Visualization Loop:** It iterates through each test image and displays the corrupted image, posterior expectation, and ground truth (if available) in the corresponding subplots of each row.
- **Cleaning Up the Plots:** It removes axis labels for a cleaner presentation.
- **Overall Layout and Output:** It adjusts the layout for better visualization and displays the combined plots. Finally, it returns the generated figure.

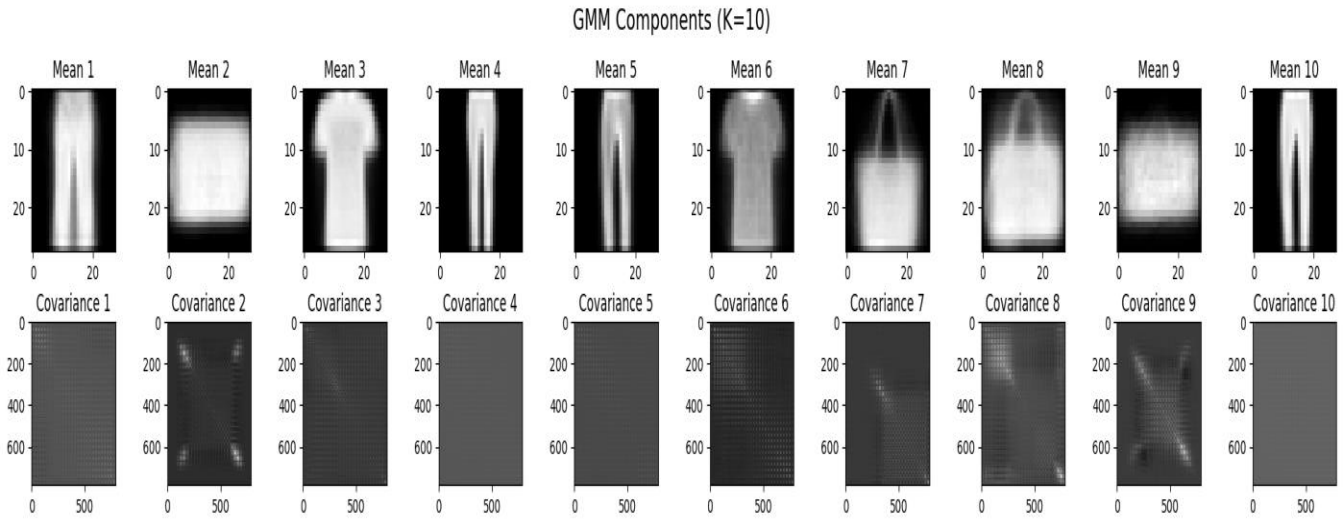
```python
def task3(test_data, gmm_params):
    # Unpack GMM parameters obtained from fitting
    gmm = gmm_params  # gmm_params should be the trained GMM model

    # Conditioned GMM: compute posterior expectations for the test data
    S, H, W = test_data.shape  # Get dimensions of test data
    posterior_expectations = conditional_gmm(gmm, test_data)

    # Plotting the results (corrupted, restored, ground truth if available)
    fig, ax = plt.subplots(S, 3, figsize=(12, 4*S))

    data = np.load('data.npz')
    test_data = data['test_data']

    # Generate corruption mask (90% pixel masking)
    S, H, W = test_data.shape
    mask = np.random.choice([0, 1], size=(S, H, W), p=[0.9, 0.1])

    # Apply mask to each test image (for demonstration)
    corrupted_images = test_data * mask

    for s in range(S):
        ax[s, 0].imshow(corrupted_images[s], cmap='gray')
        ax[s, 0].set_title('Condition')
        ax[s, 1].imshow(posterior_expectations[s].reshape(H, W), cmap='gray')
        ax[s, 1].set_title('Posterior Expectation')
        ax[s, 2].imshow(test_data[s], cmap='gray')
        ax[s, 2].set_title('Ground Truth ')

        for a in ax[s]:
            a.axis('off')

    plt.tight_layout()
    plt.show()

    return fig
```

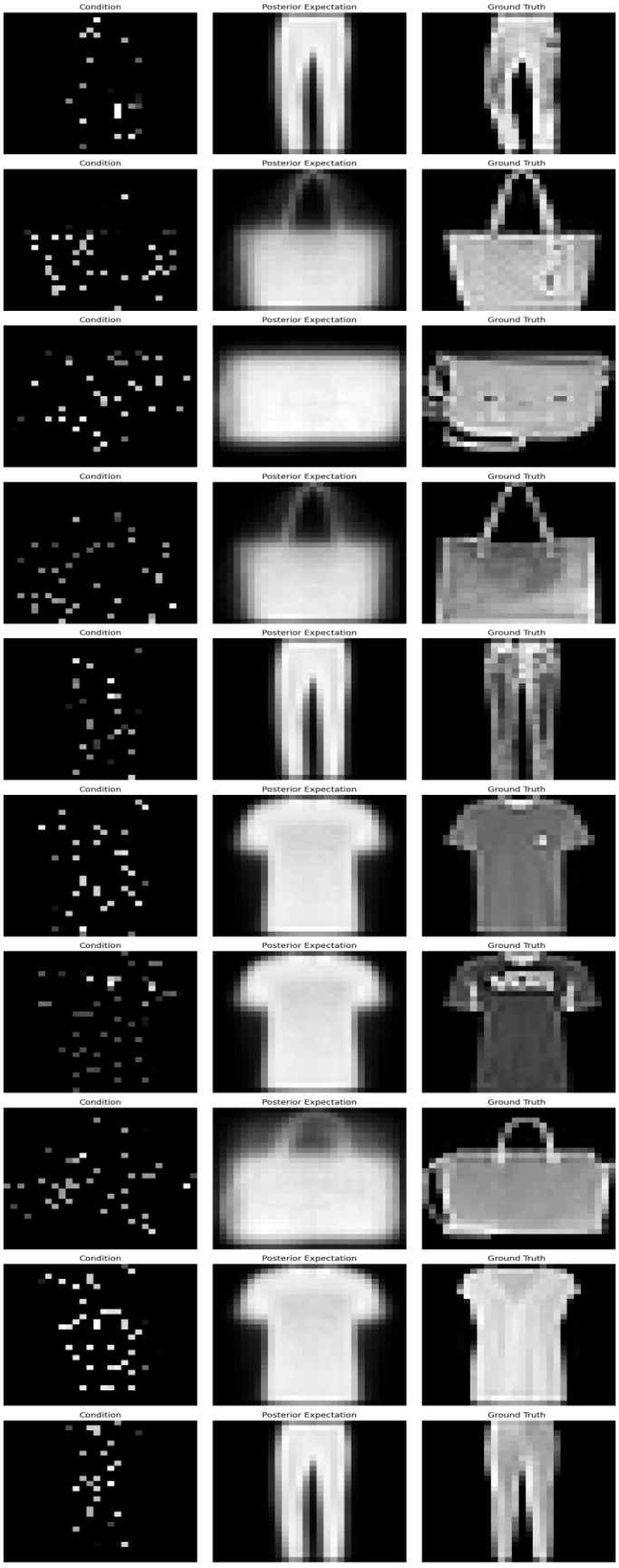
main():

- Simulating data corruption with a mask.
- Learning underlying patterns in training data using a GMM.
- Potentially restoring information in corrupted test data using the learned GMM.
- Visualizing the results (potentially including original, corrupted, and restored images).
- Saving the visualizations to a PDF document.

```python
if __name__ == '__main__':
    pdf = PdfPages('figures.pdf')

    # Task 1: Generate corruption mask
    mask = task1()

    # Load train and test data (assuming train_data and test_data are keys in data.npz)
    try:
        with np.load("data.npz") as data:
            if "train_data" in data and "test_data" in data:
                train_data = data["train_data"]
                test_data = data["test_data"]
            else:
                raise ValueError("Missing 'train_data' or 'test_data' in data.npz")
    except Exception as e:
        print(f"Error loading data.npz: {e}")
        train_data = None
        test_data = None

    if train_data is not None and test_data is not None:
        # Task 2: Fit GMM to training data and visualize GMM components
        K = 10  # Number of GMM components (adjust as needed)
        gmm_params, fig1 = task2(train_data, K)

        # Task 3: Image restoration using conditional GMM
        fig2 = task3(test_data, gmm_params)

        # Save figures to PDF
        pdf.savefig(fig1)
        pdf.savefig(fig2)
        pdf.close()
    else:
        print("Data loading failed. Check the content of data.npz and the keys.")
```

Output:


GMM Components (K=10)
Mean 1
Mean 2
Mean 3
Mean 4
Mean 5
Mean 6
Mean 7
Mean 8
Mean 9
Mean 10
Covariance 1
Covariance 2
Covariance 3
Covariance 4
Covariance 5
Covariance 6
Covariance 7
Covariance 8
Covariance 9
Covariance 10


Condition
Posterior Expectation
Ground Truth
Condition
Posterior Expectation
Ground Truth
Condition
Posterior Expectation
Ground Truth
Condition
Posterior Expectation
Ground Truth
Condition
Posterior Expectation
Ground Truth
Condition
Posterior Expectation
Ground Truth
Condition
Posterior Expectation
Ground Truth
Condition
Posterior Expectation
Ground Truth
Condition
Posterior Expectation
Ground Truth
Condition
Posterior Expectation
Ground Truth