Importing the necessary Libaries

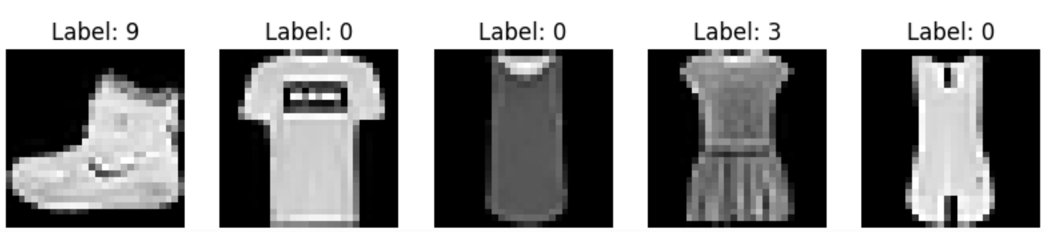
```
import numpy as np
import matplotlib.pyplot as plt
import struct
```

```
# Load the images file
def load_images(file_path):
    with open(file_path, 'rb') as f:
        magic, num, rows, cols = struct.unpack('>IIII', f.read(16))
        images = np.fromfile(f, dtype=np.uint8).reshape(num, rows, cols)
    return images

# Load the labels file
def load_labels(file_path):
    with open(file_path, 'rb') as f:
        magic, num = struct.unpack('>II', f.read(8))
        labels = np.fromfile(f, dtype=np.uint8)
    return labels

# File paths
images_file = 'images-idx3-ubyte'
labels_file = 'labels-idx1-ubyte'

# Load data
images = load_images(images_file)
labels = load_labels(labels_file)
```

```
# Check dataset shapes
print("Images shape:", images.shape)
print("Labels shape:", labels.shape)
```

```
Images shape: (60000, 28, 28)
Labels shape: (60000,)
```

```
# Display a few images with their labels
def display_images(images, labels, num_images=5):
    plt.figure(figsize=(10, 5))
    for i in range(num_images):
        plt.subplot(1, num_images, i + 1)
        plt.imshow(images[i], cmap='gray')
        plt.title(f"Label: {labels[i]}")
        plt.axis('off')
    plt.show()

display_images(images, labels)
```




```
# Verify grayscale format by checking a single image's pixel values
print("Pixel values of first image:")
print(images[0])
```

```
Pixel values of first image:
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
```

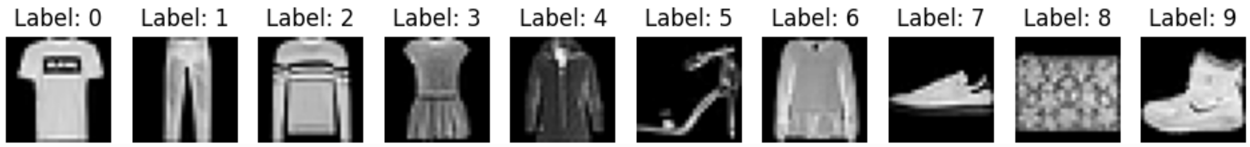
```
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0  13  73   0
    0   1   4   0   0   0   0   1   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0  36 136 127  62
   54   0   0   0   1   3   4   0   0   3]
 [  0   0   0   0   0   0   0   0   0   0   0   0   6   0 102 204 176 134
  144 123  23   0   0   0   0  12  10   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 155 236 207 178
  107 156 161 109  64  23  77 130  72  15]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0  69 207 223 218 216
  216 163 127 121 122 146 141  88 172  66]
 [  0   0   0   0   0   0   0   0   0   1   1   1   0 200 232 232 233 229
  223 223 215 213 164 127 123 196 229   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 183 225 216 223 228
  235 227 224 222 224 221 223 245 173   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 193 228 218 213 198
  180 212 210 211 213 223 220 243 202   0]
 [  0   0   0   0   0   0   0   0   0   1   3   0  12 219 220 212 218 192
  169 227 208 218 224 212 226 197 209  52]
 [  0   0   0   0   0   0   0   0   0   0   6   0  99 244 222 220 218 203
  198 221 215 213 222 220 245 119 167  56]
 [  0   0   0   0   0   0   0   0   0   4   0   0  55 236 228 230 228 240
  232 213 218 223 234 217 217 209  92   0]
 [  0   0   1   4   6   7   2   0   0   0   0   0 237 226 217 223 222 219
  222 221 216 223 229 215 218 255  77   0]
 [  0   3   0   0   0   0   0   0   0  62 145 204 228 207 213 221 218 208
  211 218 224 223 219 215 224 244 159   0]
 [  0   0   0   0  18  44  82 107 189 228 220 222 217 226 200 205 211 230
  224 234 176 188 250 248 233 238 215   0]
 [  0  57 187 208 224 221 224 208 204 214 208 209 200 159 245 193 206 223
  255 255 221 234 221 211 220 232 246   0]
 [  3 202 228 224 221 211 211 214 205 205 205 220 240  80 150 255 229 221
  188 154 191 210 204 209 222 228 225   0]
 [ 98 233 198 210 222 229 229 234 249 220 194 215 217 241  65  73 106 117
  168 219 221 215 217 223 223 224 229  29]
 [ 75 204 212 204 193 205 211 225 216 185 197 206 198 213 240 195 227 245
  239 223 218 212 209 222 220 221 230  67]
 [ 48 203 183 194 213 197 185 190 194 192 202 214 219 221 220 236 225 216
  199 206 186 181 177 172 181 205 206 115]
 [  0 122 219 193 179 171 183 196 204 210 213 207 211 210 200 196 194 191
  195 191 198 192 176 156 167 177 210  92]
 [  0   0  74 189 212 191 175 172 175 181 185 188 189 188 193 198 204 209
  210 210 211 188 188 194 192 216 170   0]
 [  2   0   0   0  66 200 222 237 239 242 246 243 244 221 220 193 191 179
  182 182 181 176 166 168  99  58   0   0]
 [  0   0   0   0   0   0   0  40  61  44  72  41  35   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]
```

Level 1: Exploratory Data Analysis

```
import numpy as np
import matplotlib.pyplot as plt
import struct
import pandas as pd
```

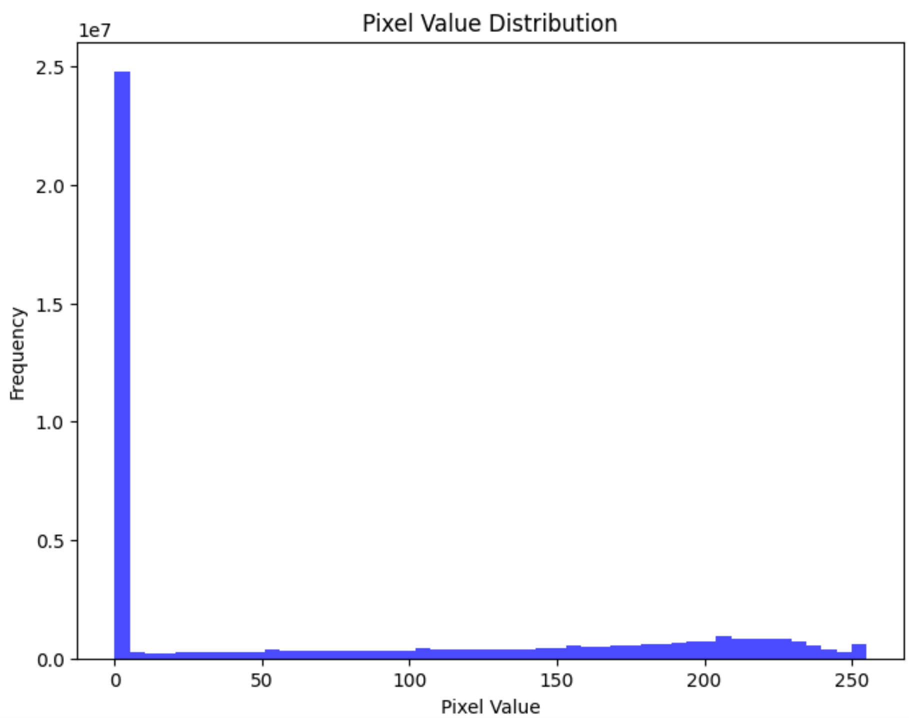
```
# Check dataset shapes
print("Images shape:", images.shape)
print("Labels shape:", labels.shape)

# Display sample images from each category
def display_sample_images(images, labels):
    unique_labels = np.unique(labels)
    plt.figure(figsize=(12, 6))
    for i, label in enumerate(unique_labels):
        idx = np.where(labels == label)[0][0]  # Get index of first image with this label
        plt.subplot(1, len(unique_labels), i + 1)
        plt.imshow(images[idx], cmap='gray')
        plt.title(f"Label: {label}")
        plt.axis('off')
    plt.show()

display_sample_images(images, labels)
```

```
Images shape: (60000, 28, 28)
Labels shape: (60000,)
```




```
# Generate summary statistics for pixel values
pixel_values = images.reshape(-1)  # Flatten all pixel values into a single array
summary_stats = pd.Series(pixel_values).describe()
print("\nSummary Statistics for Pixel Values:")
print(summary_stats)

# Visualize pixel value distribution
plt.figure(figsize=(8, 6))
plt.hist(pixel_values, bins=50, color='blue', alpha=0.7)
plt.title("Pixel Value Distribution")
plt.xlabel("Pixel Value")
plt.ylabel("Frequency")
plt.show()
```

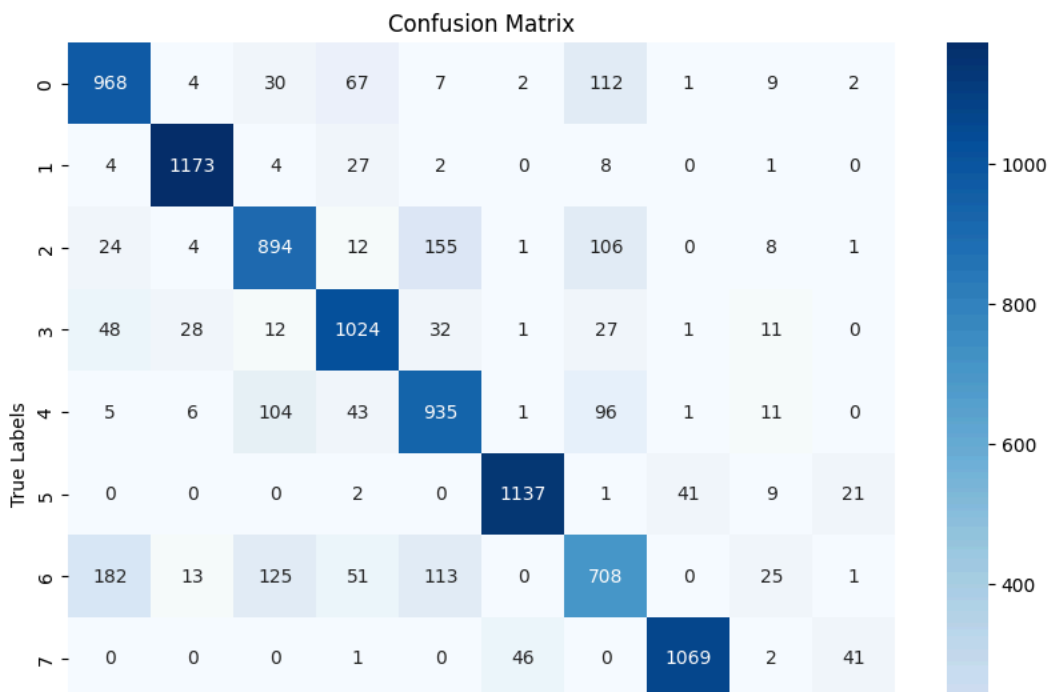
```
Summary Statistics for Pixel Values:
count    4.704000e+07
mean     7.294035e+01
std      9.002118e+01
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      1.630000e+02
max      2.550000e+02
dtype: float64
```



Level 2: Basic Classification Model

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import shap


# Preprocess and normalize the data
# Flatten the images (num_images x rows x cols -> num_images x (rows * cols))
flattened_images = images.reshape(images.shape[0], -1)  # Reshape to 2D array

# Normalize pixel values to range [0, 1]
normalized_images = flattened_images / 255.0

# Standardize data (mean=0, variance=1)
scaler = StandardScaler()
standardized_images = scaler.fit_transform(normalized_images)

# Split the dataset into training and testing subsets
X_train, X_test, y_train, y_test = train_test_split(standardized_images, labels, test_size=0.2, random_state=42)

# Train Logistic Regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion Matrix Visualization
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(labels), yticklabels=np.unique(labels))
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (sta
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Accuracy: 84.26%

Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.81      0.79      1202
           1       0.96      0.96      0.96      1219
           2       0.76      0.74      0.75      1205
           3       0.82      0.86      0.84      1184
           4       0.75      0.78      0.76      1202
           5       0.93      0.94      0.93      1211
           6       0.64      0.58      0.61      1218
           7       0.91      0.92      0.92      1159
           8       0.93      0.90      0.92      1197
           9       0.94      0.93      0.94      1203

    accuracy                           0.84     12000
   macro avg       0.84      0.84      0.84     12000
weighted avg       0.84      0.84      0.84     12000
```



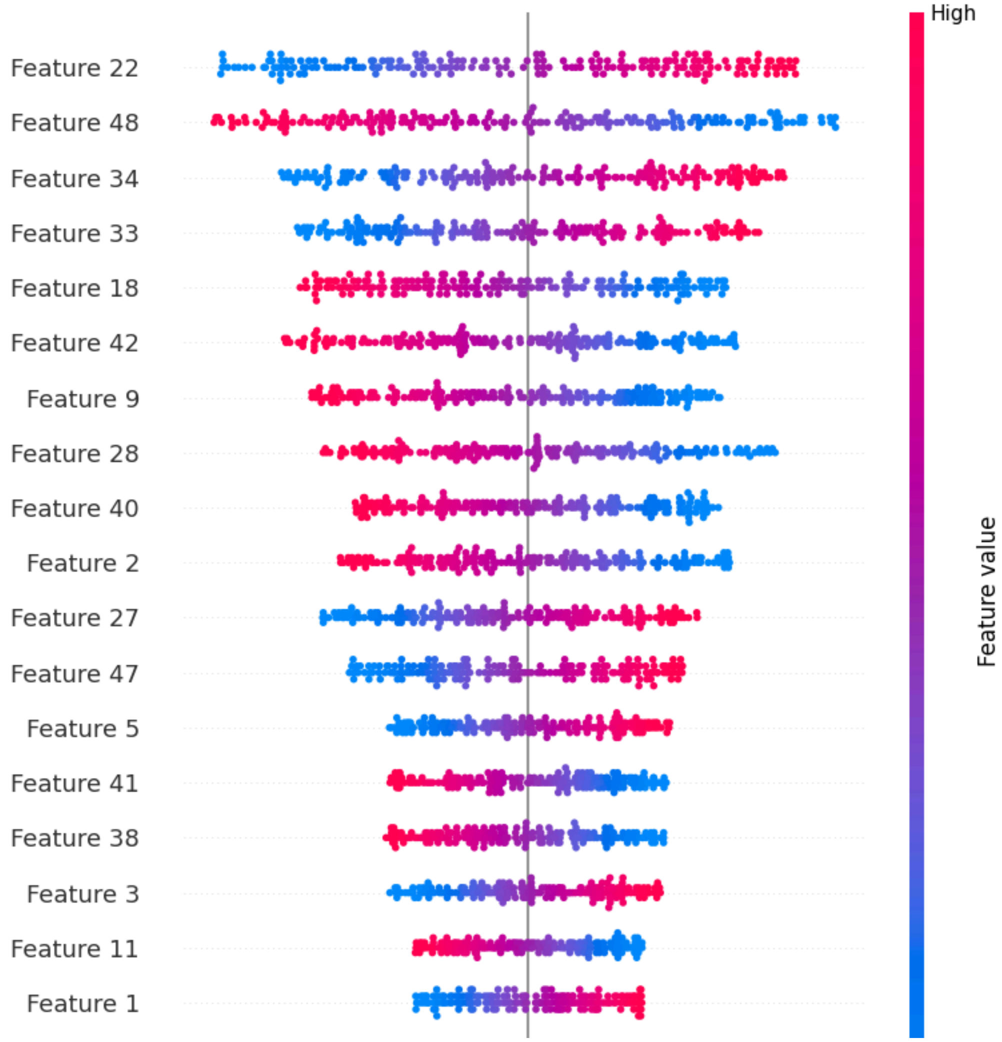

```
pip install shap lime matplotlib seaborn
```

Show hidden output

```
import shap
import lime
import lime.lime_tabular
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# Simulated dataset (replace with real clothing dataset)
num_samples = 1000
num_features = 50  # Assume 50 extracted features
X = np.random.rand(num_samples, num_features)
y = np.random.randint(0, 2, num_samples)  # Binary classification

# Normalize and standardize data
scaler = StandardScaler()
```

```
X = scaler.fit_transform(X)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Logistic Regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Evaluate model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# ------------------------------------------------
# 🔹 EXPLAINABILITY USING SHAP
# ------------------------------------------------

# SHAP Explainer
explainer = shap.Explainer(model, X_train)
shap_values = explainer(X_test)

# Summary plot (Global feature importance)
shap.summary_plot(shap_values, X_test)

# ------------------------------------------------
# 🔹 EXPLAINABILITY USING LIME
# ------------------------------------------------

# LIME Explainer for tabular data
lime_explainer = lime.lime_tabular.LimeTabularExplainer(X_train, mode="classification")

# Explain a single instance
idx = 0  # Choose any test sample index
exp = lime_explainer.explain_instance(X_test[idx], model.predict_proba)

# Show LIME explanation
exp.show_in_notebook()
exp.as_pyplot_figure()
plt.show()
```

```
Accuracy: 56.50%

Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.47      0.51        96
           1       0.57      0.65      0.61       104

    accuracy                           0.56       200
   macro avg       0.56      0.56      0.56       200
weighted avg       0.56      0.56      0.56       200
```




```
# Import necessary libraries
import struct
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from sklearn.model_selection import train_test_split


# Function to load images from IDX file
def load_images(file_path):
    with open(file_path, 'rb') as f:
        magic, num, rows, cols = struct.unpack('>IIII', f.read(16))
        images = np.fromfile(f, dtype=np.uint8).reshape(num, rows, cols)
    return images

# Function to load labels from IDX file
def load_labels(file_path):
    with open(file_path, 'rb') as f:
```

```
        magic, num = struct.unpack('>II', f.read(8))
        labels = np.fromfile(f, dtype=np.uint8)
    return labels

# File paths (Ensure these files exist in the current directory)
images_file = 'images-idx3-ubyte'
labels_file = 'labels-idx1-ubyte'

# Load images and labels
images = load_images(images_file)
labels = load_labels(labels_file)

# Print dataset shapes
print(f"Loaded images shape: {images.shape}")
print(f"Loaded labels shape: {labels.shape}")
```

```
Loaded images shape: (60000, 28, 28)
Loaded labels shape: (60000,)
```

11 <= -0.82

```
# Normalize images to range [0, 1]
images = images / 255.0

# Split dataset into training (80%) and validation (20%) sets
X_train, X_val, y_train, y_val = train_test_split(images, labels, test_size=0.2, random_state=42)

# Print dataset shapes after split
print(f"Training set shape: {X_train.shape}, Labels: {y_train.shape}")
print(f"Validation set shape: {X_val.shape}, Labels: {y_val.shape}")
```

```
Training set shape: (48000, 28, 28), Labels: (48000,)
Validation set shape: (12000, 28, 28), Labels: (12000,)
```

```
# Define the Neural Network model
model = Sequential([
    Flatten(input_shape=(28, 28)),  # Flatten 28x28 images to a 1D vector
    Dense(128, activation='relu'),  # Hidden layer with 128 neurons and ReLU activation
    Dense(64, activation='relu'),   # Hidden layer with 64 neurons and ReLU activation
    Dense(10, activation='softmax') # Output layer (10 classes for clothing categories)
])

# Compile the model
model.compile(optimizer=Adam(),
              loss=SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

# Display model summary
model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100,480
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 10)	650

Total params: 109,386 (427.29 KB)
Trainable params: 109,386 (427.29 KB)
Non-trainable params: 0 (0.00 B)

```
# Train the model
history = model.fit(X_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(X_val, y_val))
```

```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - accuracy: 0.7674 - loss: 0.6693 - val_accuracy: 0.8334 - val_loss: 0.4560
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - accuracy: 0.8597 - loss: 0.3894 - val_accuracy: 0.8618 - val_loss: 0.3803
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.8747 - loss: 0.3415 - val_accuracy: 0.8607 - val_loss: 0.3921
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.8809 - loss: 0.3211 - val_accuracy: 0.8717 - val_loss: 0.3665
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8895 - loss: 0.2983 - val_accuracy: 0.8754 - val_loss: 0.3498
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.8941 - loss: 0.2845 - val_accuracy: 0.8802 - val_loss: 0.3358
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9004 - loss: 0.2654 - val_accuracy: 0.8783 - val_loss: 0.3429
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9058 - loss: 0.2545 - val_accuracy: 0.8840 - val_loss: 0.3309
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9100 - loss: 0.2453 - val_accuracy: 0.8881 - val_loss: 0.3287
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9097 - loss: 0.2411 - val_accuracy: 0.8850 - val_loss: 0.3396
```

```python
# Evaluate the model
val_loss, val_accuracy = model.evaluate(X_val, y_val)
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
```

```
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8849 - loss: 0.3362
Validation Accuracy: 88.50%
```

```python
# Plot accuracy and loss graphs
plt.figure(figsize=(12, 5))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy over epochs')

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Loss over epochs')

plt.show()
```