DLBDSMTP01 - PROJECT: FROM MODEL TO PRODUCTION

ORAL PROJECT REPORT

Task 2:

Image classification for a refund department

(spotlight: Batch processing)

Course of Study:

Bachelor of Science Applied Artificial Intelligence

Tutor Name:

Dr. Frank Passing

Student Name:

Achshah R M

Matriculation No.:

92125572

Submission Date: 17/08/2024

Table of Contents

1. Introduction

1.1 Project Overview

As e-commerce platforms expand, the volume of returned items increases, making efficient categorization essential for managing inventory, processing refunds, and maintaining customer satisfaction. Traditional manual sorting is labor-intensive, error-prone, and lacks scalability. To address this, the project leverages machine learning (ML) techniques, specifically Convolutional Neural Networks (CNNs), to automate the classification of returned items in an online fashion store.

CNNs are highly effective for image processing tasks due to their ability to capture spatial hierarchies in visual data (Krizhevsky, Sutskever, & Hinton, 2012). They have been widely adopted in various applications, including inventory management and quality control, due to their high accuracy and efficiency (He et al, 2016). This project integrates a CNN-based image classification model into a batch processing system, automating the categorization of returned items and reducing the need for manual intervention.

1.2 Objectives

The primary objective of this project is to develop an automated system for classifying images of returned items in an online fashion store, thereby reducing the manual effort and cost associated with the categorization process. Specific objectives include:

1. **Model Development**: Create a CNN-based model capable of accurately classifying images of returned items into predefined categories.
2. **System Integration**: Seamlessly connect the model with the API for batch processing .
3. **System Automation**: Automate the classification process using cron jobs.

1.3 Scope

The scope of this project includes the development, deployment, and testing of an image classification system for a specific use case within the e-commerce domain. This project is confined to the following aspects:

- **Image Classification**: Focus on classifying images of fashion items (e.g., clothes, shoes, accessories) using a CNN.
- **Batch Processing**: Implementation of a batch processing system to automate the classification process at specified times.
- **Local Deployment**: Deployment of the model on a local machine, with automation managed through a cron job.

Excluded:

- **Cloud Deployment**: This project does not include deploying the system to a cloud environment or integrating cloud-based services.
- **Multi-Platform Support**: The system is designed to run on a local machine and is not optimized for deployment across different operating systems or platforms.
- **Real-Time Processing**: The project focuses on batch processing rather than real-time image classification.

This project aims to demonstrate the effectiveness of machine learning in automating repetitive tasks in e-commerce, providing a framework that can be expanded or adapted for broader applications in the future.

2. System Architecture

2.1 Architecture Overview

The system architecture for this project is designed to automate the classification of returned items in an online fashion store, leveraging a Convolutional Neural Network (CNN) model, as shown figure 1. The architecture is composed of three primary modules: data ingestion, model serving, and batch processing. These modules work together to ensure that images of returned items are automatically processed and categorized with minimal manual intervention.

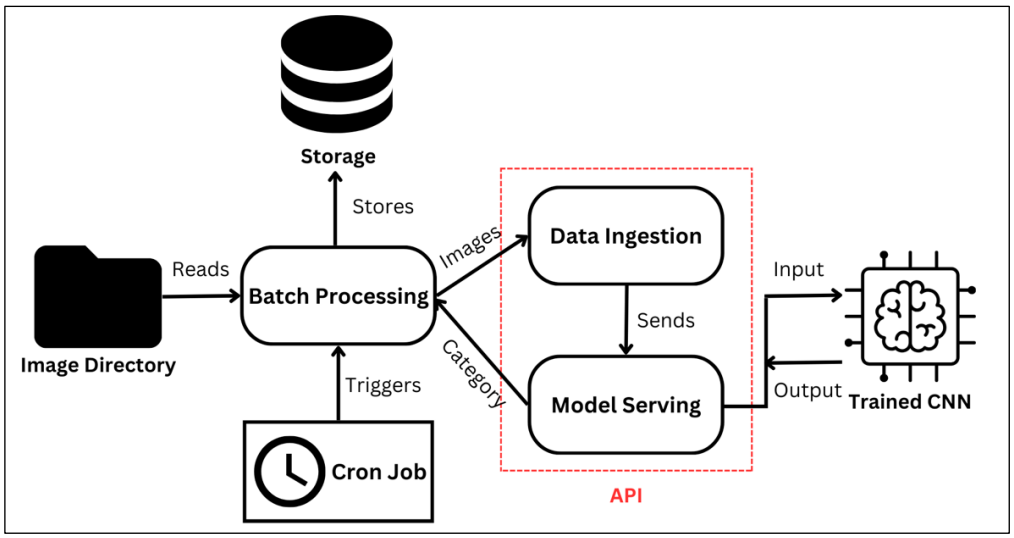

Figure 1: System Architecture

2.2 Components Description

2.2.1 Data Ingestion

The Data Ingestion Module is responsible for receiving image data from the Batch Processing system. In this project, images are first read by the Batch Processing script and then sent to the API. The Data Ingestion Module within the API normalizes, resizes, and reshapes the images, preparing them for classification by the model.

2.2.2 Model Serving

The Model Serving Module deploys the CNN model and makes it accessible through a RESTful API. This module is implemented using Flask, a lightweight Python web framework. The trained model is loaded and served through an API endpoint, which accepts image data, processes it through the CNN model, and returns the predicted category.

2.2.3 Batch Processing

The Batch Processing System automates the classification task by scheduling the processing of images at specified intervals. A cron job is set up to trigger a Python script (batch_request.py) at a scheduled time (e.g., 7:00 PM daily). This script reads the images from the directory, sends them to the API for classification, and saves the predictions in a CSV file. The batch processing system ensures that the image classification task is performed automatically without requiring manual intervention.

2.3 Data Flow

The data flow in the system is designed for efficiency and accuracy, ensuring smooth processing and minimal delays:

1. **Image Directory**: Image files are stored in a designated directory, serving as the source for batch processing.
2. **Batch Processing**: At the scheduled time, the batch processing script reads the images from the directory and sends them to the API.
3. **Data Ingestion (API Module)**: The API's Data Ingestion Module receives the images, normalizes, resizes, and reshapes them to the required format for model processing.
4. **Model Serving (API Module)**: The processed images are passed to the CNN model for classification, and the model generates predictions for each image.
5. **Prediction Return**: The predictions are returned from the Model Serving Module to the Batch Processing script.
6. **Output**: The Batch Processing script saves the predictions into a CSV file, which can be used for further analysis or reporting.

This structured data flow ensures that each component interacts seamlessly, allowing the system to perform accurate and timely classifications.

3. Model Design

3.1 Model Selection

For this project, a Convolutional Neural Network (CNN) was chosen as the model architecture due to its proven effectiveness in image classification tasks. CNNs are specifically designed to process

data that have a grid-like topology, such as images. They excel at capturing spatial hierarchies in visual data through convolutional layers that automatically learn feature representations, such as edges, textures, and shapes, making them ideal for tasks like classifying fashion items. Given that we're working with the Fashion MNIST dataset, which consists of grayscale images of clothing items, a CNN is well-suited to identify and categorize these items based on their visual features.

3.2 Model Architecture

The CNN architecture for this project is designed to efficiently process the 28x28 grayscale images in the Fashion MNIST dataset and classify them into one of ten categories. The architecture is composed of several layers, each serving a specific role in the feature extraction and classification process, as shown in figure 2 and 3.

The architecture of the Convolutional Neural Network (CNN) designed for this project begins with an Input Layer that accepts grayscale images with a shape of (28, 28, 1). This layer serves as the entry point for the image data, preparing it for subsequent processing by the network.

The network then consists of a series of Convolutional Layers, each followed by a MaxPooling layer to progressively detect features at different levels of abstraction. The first convolutional layer employs 32 filters with a 3x3 kernel size and uses the ReLU activation function. This layer outputs a feature map of shape (26, 26, 32) and is responsible for detecting basic features, such as edges, from the input image. Following this, a MaxPooling layer with a 2x2 pool size reduces the spatial dimensions of the feature map to (13, 13, 32), retaining the most important features while reducing the computational complexity.

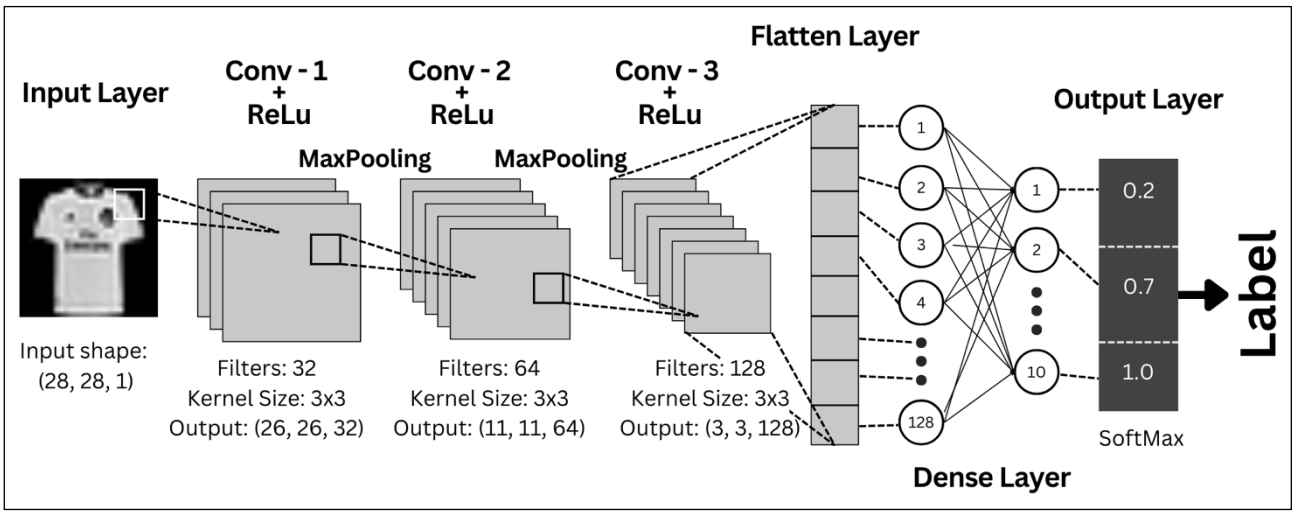

Figure 2: Convolutional Neural Network (CNN) Model Architecture

The second convolutional layer increases the filter count to 64, maintaining a 3x3 kernel size with ReLU activation. This layer outputs a feature map of shape (11, 11, 64) and focuses on detecting more complex features like textures. Another MaxPooling layer with a 2x2 pool size further reduces the dimensions to (5, 5, 64). The third convolutional layer, with 128 filters and a 3x3 kernel size,

captures even more complex patterns such as shapes and motifs. The output of this layer is a feature map of shape (3, 3, 128), representing the high-level abstractions learned by the network.

Following the convolutional layers, the network includes a Flatten Layer, which converts the 3D feature map from the final convolutional layer into a 1D vector of shape (1152). This flattening process prepares the data for the fully connected layers that follow, enabling the network to combine the learned features and make a classification decision.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 128)	73,856
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 128)	147,584
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Total params: 241,546 (943.54 KB)
Trainable params: 241,546 (943.54 KB)
Non-trainable params: 0 (0.00 B)

Figure 3: Summary of the Convolutional Neural Network (CNN) Model Architecture

The Fully Connected Layers consist of a dense layer with 128 units, using the ReLU activation function to serve as the decision-making component of the network. This layer integrates the features learned in the convolutional layers to make a more informed prediction. To prevent overfitting, a dropout layer with a 0.5 dropout rate is applied after the dense layer. This dropout layer randomly sets half of the input units to zero at each update during training, helping to ensure that the network generalizes well to new data.

Finally, the network concludes with an Output Layer, which is a dense layer with 10 units, corresponding to the 10 categories in the Fashion MNIST dataset. This layer uses the softmax activation function to produce a probability distribution over the 10 categories, enabling the model to classify the input image into one of the predefined classes.

3.2.1 Activation Functions

ReLU (Rectified Linear Unit) is used in the convolutional and dense layers, ReLU is a popular activation function that introduces non-linearity into the model, allowing it to learn complex patterns. It outputs the input directly if positive, otherwise, it will output zero. Moreover, Softmax is used in the output layer, the softmax activation function converts the logits into probabilities, ensuring that the sum of all output probabilities equals 1. This is ideal for multi-class classification tasks like this one.

3.2.2 Optimizer and Loss Function

The Adam optimizer was chosen for this project due to its adaptive learning rate and efficient handling of sparse gradients. Adam combines the advantages of two other popular optimizers: AdaGrad and RMSProp, making it well-suited for tasks like image classification. Categorical cross-entropy was used as the loss function. This function is appropriate for multi-class classification problems, as it calculates the loss by comparing the predicted probability distribution with the true distribution (i.e., the one-hot encoded labels).

3.3 Training Process

3.3.1 Dataset

The Fashion MNIST dataset (Zalando, 2017) was used for training and evaluating the model, shown in figure 4. This dataset consists of 60,000 grayscale images, each of size 28x28 pixels, representing 10 categories of fashion items. The dataset was divided into three sets:

- **Training Set**: 36,000 images used to train the model.
- **Validation Set**: 12,000 images used to validate the model during training and to tune hyperparameters.
- **Test Set**: 12,000 images reserved for evaluating the model after training is complete.

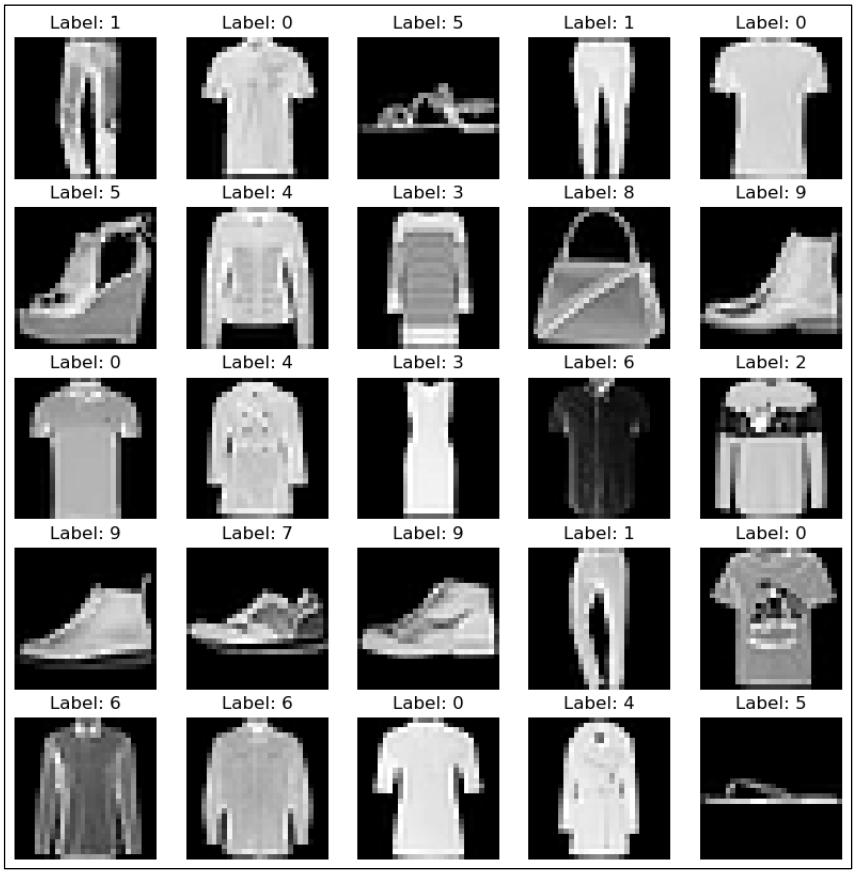

Figure 4: Random Sample of 25 Images from the Fashion-MNIST Training Data

Before proceeding with training, the distribution of labels across the dataset was checked to ensure balance. The label distribution was found to be approximately equal, meaning that each category was well-represented in the dataset, shown in figure 5. This balanced distribution helps in training the model effectively, ensuring that no single class dominates, which could otherwise lead to biased predictions.

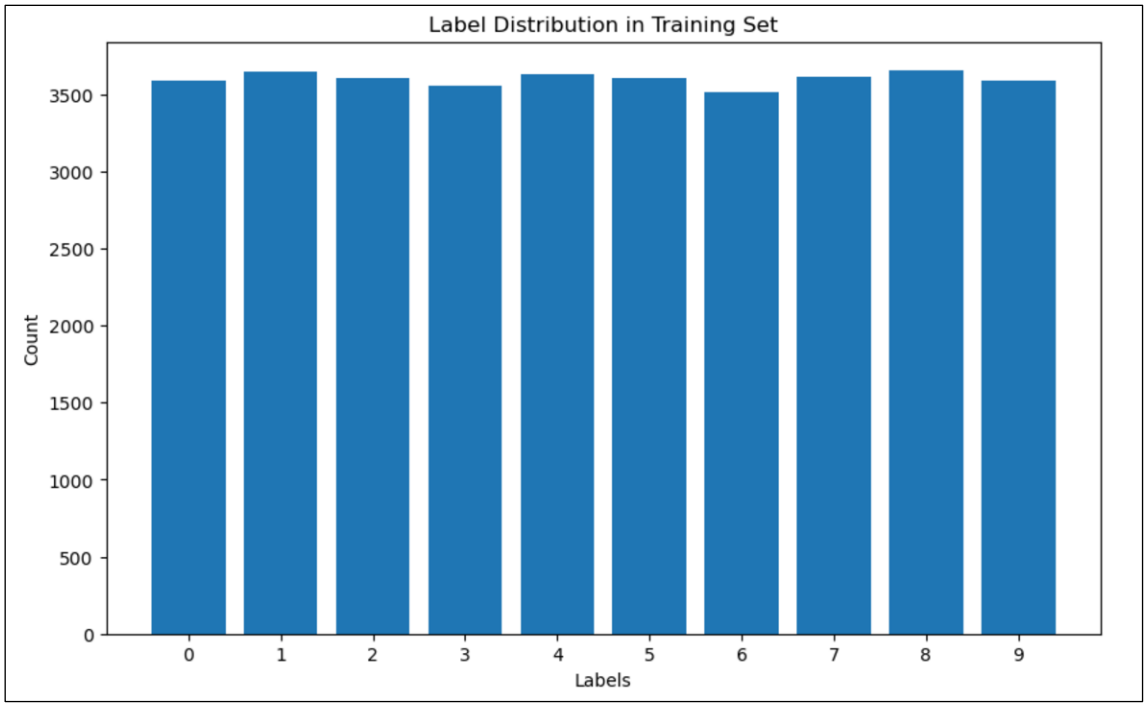

Figure 5: Label Distribution in the Training Dataset

3.3.2 Training Parameters

The model was trained over 10 epochs, with the training data fed into the network in batches of 64 images. This batch size ensures that the model updates its weights after every 64 images, balancing computational efficiency with learning performance. The learning rate, a critical parameter for adjusting the model's weights, was managed automatically by the Adam optimizer, which dynamically adjusts the learning rate throughout the training process. Each epoch represents a complete pass through the entire training dataset, allowing the model to iteratively refine its ability to classify the images accurately.

3.3.3 Evaluation Metrics

To evaluate the performance of the model, several key metrics were monitored throughout the training and validation process. Accuracy was the primary metric used to assess the proportion of correctly classified images across both the training and validation sets. This metric provided a straightforward measure of the model’s ability to generalize to unseen data.

In addition to accuracy, categorical cross-entropy loss was tracked as an indicator of the model's learning effectiveness. This loss function measures the difference between the true label

distributions and the predicted probabilities. A steady decrease in loss across epochs suggested that the model was progressively improving its predictions.

To gain a more detailed understanding of the model's performance across different categories, a confusion matrix was generated. This matrix provided a visual representation of the number of correct and incorrect predictions for each class, highlighting areas where the model excelled and where it encountered challenges. Furthermore, a classification report was produced, offering a comprehensive overview of precision, recall, and F1-score for each category, thus providing deeper insights into the model's performance on a per-class basis.

4. API Design and Usage

4.1 API Overview

The API is designed to facilitate interaction with the trained Convolutional Neural Network (CNN) model, enabling the automated classification of images in batch mode. The primary purpose of the API is to provide a standardized interface through which image data can be sent to the model, processed, and classified into predefined categories. This allows the model to be used as a service, capable of handling batch image classification requests and returning predictions in a structured format.

4.2 /process_batch

The /process_batch endpoint is the core endpoint of the API, handling batch processing requests. It is a **POST** request endpoint that expects a JSON payload containing a list of images, encoded in base64 format. Upon receiving the request, the API decodes each image, preprocesses it to the appropriate format required by the model, and then uses the CNN model to predict the category of each image. The predictions are then returned as a JSON response.

4.3 Request Format

The API expects requests to be sent in JSON format. An array of base64-encoded strings, each representing an image. These images are preprocessed to the appropriate size (28x28 pixels for the Fashion MNIST model) before being encoded. No specific headers are required beyond the default Content-Type header for JSON requests.

4.4 Response Format

The API returns a JSON response containing the predictions for each image. An array of integers, where each integer represents the predicted class label for the corresponding image in the request. The prediction labels correspond to the categories in the Fashion MNIST dataset (e.g., 0 for T-shirts, 1 for trousers, etc.).

4.5 Error Handling

The API includes basic error handling mechanisms to manage different types of errors:

- **400 Bad Request**: If the request is malformed or missing the required images field, the API will return a 400 status code. This typically indicates that the client needs to correct the request format or content.
- **500 Internal Server Error**: If there is an issue on the server side, such as a failure to process the images or load the model, the API will return a 500 status code. This error is accompanied by a log message indicating the nature of the issue, which can be used for debugging.
- **JSON Decoding Error**: If the server fails to decode the JSON response (e.g., if the response is not valid JSON), an error message will be printed, and the predictions array will be empty. This is handled in the batch_request.py script to ensure that the system can gracefully manage such errors.

These error handling mechanisms ensure that the API provides meaningful feedback to the user, helping to identify and resolve issues during the interaction with the model.

5. Batch Processing Setup

5.1 Overview

In the context of this project, batch processing is essential due to the nature of the task—automatically classifying large volumes of returned items based on their images. Rather than processing each image individually, which could be resource-intensive and time-consuming, batch processing allows for the efficient classification of images in groups, scheduled to run at specific times. This approach is particularly beneficial for an online shopping platform where returned items accumulate throughout the day and need to be processed in bulk, typically during off-peak hours. By automating this process, the system reduces manual effort and ensures that the classification is done consistently and efficiently.

5.2 Cron Job Setup

To automate the batch processing, a cron job was set up to trigger the execution of the batch_request.py script at a scheduled time. For this experiment, the cron job was scheduled for 7:00 PM to allow for real-time monitoring and validation. In a real-world scenario, the job would typically be scheduled to run during off-peak hours, such as 2:00 AM, to minimize the impact on system resources and ensure that the processing of returns does not interfere with other operational tasks.

5.3 Script Functionality

The batch_request.py script is responsible for orchestrating the batch processing of images. The script performs the following tasks:

1. **Image Fetching**: The script scans a specified directory for images that need to be processed. It reads each image, converts it to base64 encoding, and stores it in a list.

2. **Sending Requests to the API**: The script sends the base64-encoded images to the Flask API endpoint (/process_batch) via a POST request. The API processes the images using the CNN model and returns the predicted class labels.
3. **Saving Predictions**: Upon receiving the predictions from the API, the script saves the results to a CSV file. This file includes the filenames of the processed images and their corresponding predicted categories, providing a record of the classification results.

This script is designed to run automatically at the specified time, ensuring that the batch processing is performed without requiring manual intervention.

6. Results and Analysis

6.1 Model Performance

During the training process, the model's performance was evaluated primarily through two key metrics: accuracy and loss. The training and validation accuracy steadily increased across the 10 epochs, while the training and validation loss decreased correspondingly, indicating that the model was learning effectively, shown in figure 6. The final evaluation on the test dataset yielded an accuracy of 90.61%, which signifies that the model correctly classified 90.61% of the images in the test set. Categorical cross-entropy was used as the loss function, and the model achieved a loss of 0.27. This high accuracy and low loss suggest that the model is capable of generalizing well to unseen data, making it suitable for deployment in a real-world scenario.

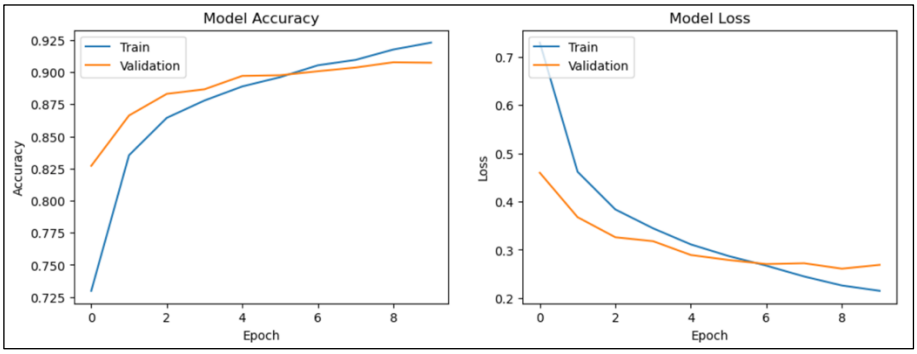

Figure 6: Training and Validation Accuracy and Loss Curves

Figure 7 illustrates sample predictions made by the model on a set of six random images from the test dataset. For each image, the figure displays both the model's predicted label (Pred) and the true label (True). As seen in the figure, the model correctly identified the labels for four out of the six images, including categories such as a coat (True label 4, Predicted label 4) and an ankle boot (True label 9, Predicted label 9). However, the model misclassified two images: it predicted the label 6 (Shirt) for images that actually belonged to the category 0 (T-shirt/top), and similarly, it predicted label 3 (Dress) for another image with the true label 0 (T-shirt/top). This highlights the model's challenges in distinguishing between certain categories that may have visually similar features.

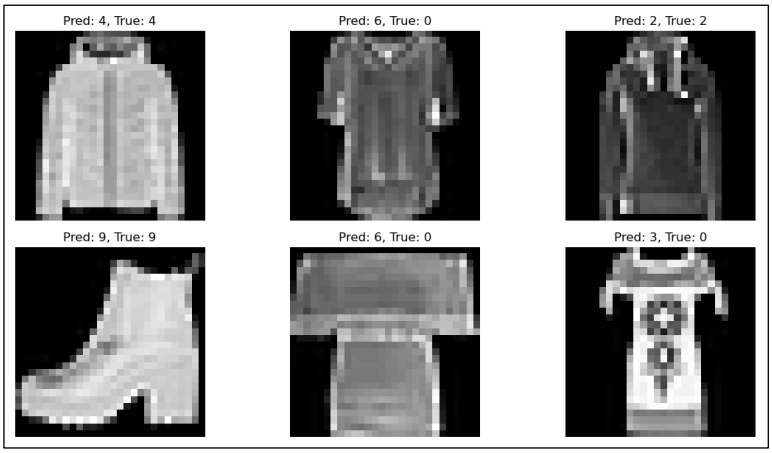

Figure 7: Sample Predictions with True and Predicted Labels

To gain a deeper understanding of the model's performance across different categories, a confusion matrix was generated, represented in figure 8. The confusion matrix provided insight into where the model performed well and where it struggled. For instance, the model showed strong performance in categories such as labels 1 (trousers) and 9 (ankle boots), where it achieved near-perfect precision and recall. However, it encountered challenges with certain categories like label 6 (shirts) and label 2 (pullover), where the precision and recall were lower, indicating some difficulty in distinguishing these items from other similar categories.

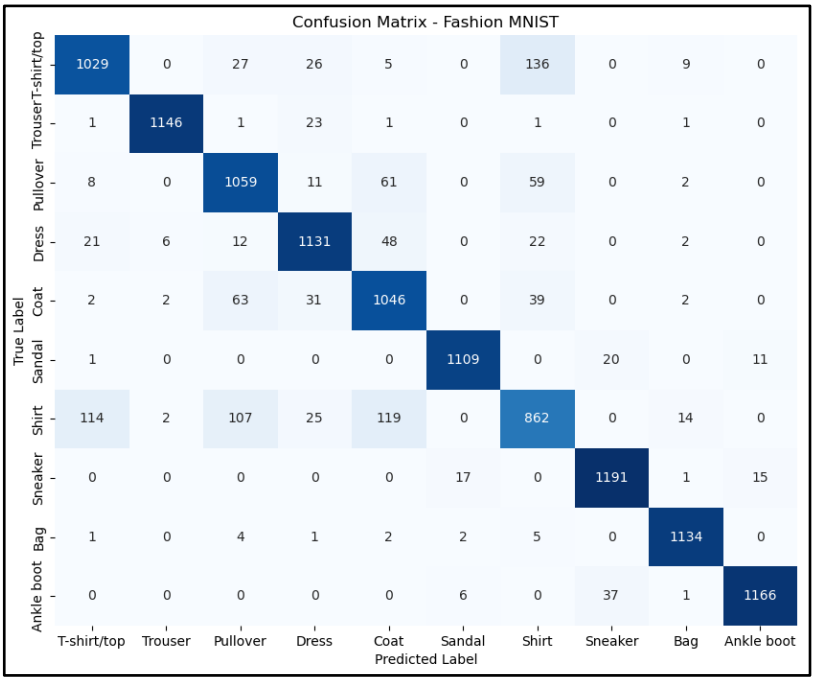

Figure 8: Confusion Matrix for Model Predictions on the Test Dataset

Furthermore, a classification report was produced to provide a detailed breakdown of the model's performance in terms of precision, recall, and F1-score for each category. The report confirmed that while the overall performance was strong, specific categories could benefit from further fine-tuning or additional training data to improve the model’s accuracy in those areas.

Classification Report:

	precision	recall	f1-score	support
0	0.87	0.84	0.85	1232
1	0.99	0.98	0.98	1174
2	0.83	0.88	0.86	1200
3	0.91	0.91	0.91	1242
4	0.82	0.88	0.85	1185
5	0.98	0.97	0.97	1141
6	0.77	0.69	0.73	1243
7	0.95	0.97	0.96	1224
8	0.97	0.99	0.98	1149
9	0.98	0.96	0.97	1210
accuracy			0.91	12000
macro avg	0.91	0.91	0.91	12000
weighted avg	0.91	0.91	0.91	12000

Figure 9: Classification Report for Model Predictions on the Test Datasets

6.2 System Performance

Beyond the model's predictive capabilities, the overall system performance was evaluated, focusing on the efficiency and responsiveness of the API and the batch processing setup. The Flask API demonstrated quick response times when handling batch requests, efficiently processing large volumes of images and returning predictions in a timely manner. This responsiveness is crucial for ensuring that the batch processing system operates smoothly, particularly during overnight runs when the volume of data may be substantial.

The batch processing system, managed through a cron job, was tested by scheduling the task at 7:00 PM daily. The system successfully fetched the images from the specified directory, sent them to the API for classification, and saved the predictions in a CSV file. This test run confirmed the system’s ability to handle the end-to-end process reliably. In a real-world application, the cron job could be scheduled to run at off-peak hours, such as 2:00 AM, to further optimize resource usage while ensuring that returned items are classified and ready for further processing by the start of the next business day.

By analyzing both the model's predictive performance and the system's operational efficiency, this project demonstrated that the deployed CNN model and its surrounding infrastructure are well-equipped to handle the task of automating the classification of returned items in an e-commerce setting.

7. Conclusion

This project successfully developed and deployed a Convolutional Neural Network (CNN) model to automate the classification of returned items in an online fashion store. The model demonstrated strong performance, achieving an accuracy of 90.61% on the test set, which is indicative of its ability to generalize well to unseen data. The system was further enhanced by integrating a batch processing setup, ensuring that large volumes of images could be classified efficiently and automatically, with minimal manual intervention.

While the results are promising, there are several areas where the system could be improved. One potential enhancement is the integration of cloud storage and processing capabilities, which would allow for greater scalability and flexibility, particularly as the volume of returned items increases. Additionally, expanding the model to handle more categories beyond the 10 provided by the Fashion MNIST dataset would make the system more versatile and applicable to a wider range of e-commerce platforms.

Looking ahead, the next steps could involve fine-tuning the model to improve its performance on categories where it currently struggles, such as differentiating between similar clothing items. Further, exploring advanced techniques like transfer learning or ensembling multiple models could enhance the model's accuracy and robustness. Finally, deploying the system in a production environment, with real-time data ingestion and continuous model updates, would be a valuable extension of this project.

Reflecting on the project, it has provided valuable insights into the practical challenges of deploying machine learning models in a real-world setting. The experience of integrating model training, API development, and batch processing into a cohesive system has underscored the importance of considering both the technical and operational aspects of machine learning projects. This project not only demonstrated the power of CNNs in image classification but also highlighted the potential of automation in improving operational efficiency in e-commerce. The skills and knowledge gained through this project are directly applicable to a wide range of real-world scenarios, where similar approaches can be employed to tackle complex challenges in various industries.

For more details and to access the full project repository, visit: https://github.com/Achshah-RM/CNN-Based-Image-Classification-for-Returned-Products

References:

- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770-778. https://doi.org/10.1109/CVPR.2016.90
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 1097-1105. https://doi.org/10.1145/3065386
- Zalando SE. (2017). Fashion-MNIST dataset [kaggle datasets]. Kaggle. https://www.kaggle.com/datasets/zalando-research/fashionmnist