CE301 – Individual Capstone Project Challenge

Oluwatosin Oluwafemi Oluwole

1902549

Fashion-Stylist Application using Deep Learning

BSc Computer Science

Supervisor: **Dr Xiaojun Zhai**

Second Assessor: **Dr Yunfei Long**

Acknowledgements

I am grateful to Dr. Xiaojun Zhai for his exceptional guidance, insights, and support throughout my Capstone project. My family and friends also deserve my appreciation for their constant belief, motivation, and love that helped me succeed. Lastly, I thank God for His blessings, opportunities, and guidance in my academic journey.

Contents

Abstract

Fashion Recommendation Systems (FRS) can enhance the shopping experience by providing personalised recommendations, improving customer engagement, and increasing sales. This project aims to research and develop an FRS that allows users to upload an outfit and get recommendations for similar outfits.

This system utilises the DeepFashion dataset, which contains over 800,000 images and detailed annotations of fashion items such as clothing type, colour, texture, and attributes. In addition to this, deep learning techniques such as the ResNet18/50, a convolutional neural network (CNN), approximate nearest neighbours, and embeddings centroid detection have been used to train the model. This project involves using classification techniques to detect the type of fashion item in the image and clustering techniques to group similar fashion items.

This approach is different from other FRS as it focuses on outfit-based recommendations instead of item-based recommendations. By analysing entire outfits, this model can provide more accurate and personalised recommendations that consider the user's style and preferences. The deep learning framework PyTorch was used to train the model using and transfer learning was incorporated to fine-tune the pre-trained models for this task. The evaluation results showed that the model can successfully recommend similar fashion items to an inputted outfit with a high accuracy.

My research contributes to the field of fashion/e-commerce and demonstrates the potential of deep learning techniques and outfit-based recommendations. I also provide an iPython notebook that documents the entire process from data preprocessing to model training and evaluation, making it easy for others to reproduce and build upon work in the future.

1. Introduction

1.1 Overview

In this digital era, the convergence of Computer Vision (CV) and Deep Learning (DL) is altering the way visual data is understood and interacted with. This project exemplifies the merging of these two areas, demonstrating how CV interprets visual elements and how DL extracts intricate patterns. These technologies are essential to recommender systems and have revolutionised user engagement with visual material. However, the vastness and complexity of visual data often leads to issues in accurately understanding user preferences and providing relevant recommendations.

To tackle this challenge, I have developed a Fashion Recommender System (FRS) that utilises advanced DL techniques. The FRS excels in understanding the subtle attributes of fashion items, ensuring accurate categorisation. The integration of CV enhances its ability to recognise visual cues, such as patterns, textures, and styles, which are crucial in fashion image analysis. The FRS encapsulates the project's principle and functions as an intelligent tool that seamlessly integrates CV and DL to manage and improve the analysis and recommendation of fashion items.

Through the FRS, users gain the ability to accurately classify fashion items and access personalised outfit recommendations. Users can upload images of desired fashion items and gain insights into their attributes, styles, and categories. Moreover, the system enhances users explore fashion by suggesting visually similar outfits, thereby providing personalised and visually captivating experiences.

1.2 Motivation

The motivation behind the development of the FRS was based on a combination of personal interests and aspirations. I have an interest in Artificial Intelligence (AI) and a fondness for online shopping, therefore, the concept of an intelligent system that could suggest outfits based on individual preferences and shopping history immediately caught my attention.

While searching through the Project Proposal Database on Moodle, I noticed there was a lack of fashion-oriented initiatives and recommendation systems within the AI/Machine Learning category. Thereby, strengthening my determination to contribute something innovative and impactful to the database.

Beyond simply developing a functional system, my objective was to channel my enthusiasm into an avenue that aligned with my personal and professional aspirations, shaping an endeavour that merges AI and fashion.

1.3 Research Objectives

1. Develop an Image Classification Model

To design and train an effective image classification model. By employing transfer learning, the model should be initialised with pre-trained weights on a large-scale fashion dataset to capture general features from diverse images. The model will be trained to accurately classify fashion images into various clothing categories.

2. Evaluate Model Performance

To assess the efficiency and reliability of the image classification model, this project aims to evaluate its performance using standard metrics. The model's ability to correctly predict the most likely clothing category and its performance in providing reasonable alternative predictions will be thoroughly analysed.

3. Investigate Image Embeddings

To investigate the concept of image embeddings and their effectiveness in representing fashion images. The objective is to transform the images into low-dimensional embeddings that contain the semantic data about the clothes. This project aims to facilitate efficient similarity search and retrieval tasks by generating image embeddings.

4. Implement Outfit Recommendation System

To develop an outfit recommendation system which is built upon the generated image embeddings. Given a user's input image of an outfit, the system should utilise the Annoy Index to find similar outfits from the dataset. The objective is to provide users with personalised recommendations of outfits that share similar visual characteristics.

5. Visualise and Interpret Results

To gain insights into the image classification model's performance and understand the outfit recommendation process, this project aims to visualise and interpret the results. Visualisations will be generated to aid in the analysis and understanding of the model's behaviour.

6. Assess Project Management

To assess the project management approach. The project's timeline, milestones, and any challenges encountered during the development process will be evaluated to highlight successful strategies and areas for improvement in future projects.

2. Literature Review and Background Research

2.1 Image Classification & Deep Learning Models

Image classification, a fundamental task in CV, involves assigning a label or category to an input image based on its visual content [1]. This task has vast applications, ranging from medical diagnosis to autonomous vehicles and e-commerce. Traditionally, image classification was approached using hand-engineered features and conventional Machine Learning (ML) algorithms. However, the emergence of DL models, particularly Convolutional Neural Networks (CNNs), has assisted in a fundamental change, revolutionising image classification, and significantly enhancing its capabilities.

2.1.1. Deep Learning and CNNs

DL, a subset of ML, has demonstrated extraordinary success in a multitude of applications, with image classification being a prominent area. CNNs have emerged as a keystone of this success, marked by their ability to automatically learn, and extract complex hierarchical features from raw image data [2]. Inspired by the human visual system, CNNs have demonstrated exceptional expertise in understanding and categorising images.

2.1.2. CNN Architecture: Key Elements

The architecture of CNNs is characterised by several key elements that contribute to their effectiveness in image classification. Convolutional layers, the core components of CNNs, employ filters that slide over the input image, capturing spatial features and patterns [3]. These filters learn to detect edges, textures, and more intricate visual elements through the process of convolution. This process can be described as "feature mapping" where high-level features are built upon lower-level ones, ultimately leading to the network's ability to discern intricate patterns and shapes within the image.

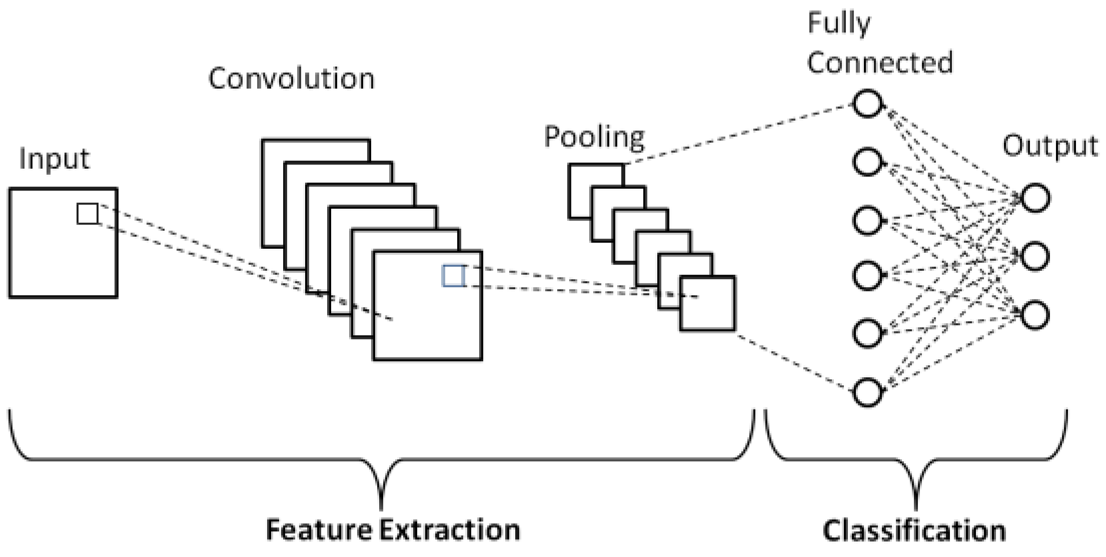


Figure 1: CNN Architecture Diagram

2.1.3. Pooling for Robustness and Efficiency

Pooling layers, typically dispersed between convolutional layers, help the network handle image shifts and computational efficiency [4]. These layers reduce feature map dimensions, preserving important details while minimising the impact of shifts, rotations, and distortions. This technique assists CNNs in identifying objects regardless of their position or orientation in the image.

2.1.4. Hierarchical Learning and Deep Attributes

The exceptional performance of CNN’S is attributed to their hierarchical structure. Deep CNNs consist of multiple layers stacked upon each other, allowing the model to learn increasingly abstract representations as data flows through the network [2]. This architecture imitates the brain's processing hierarchy, enabling the network to capture intricate details and higher-level semantics. In the words of LeCun et al., "Deep architectures allow composition of intermediate features of increasing complexity" [2].

2.1.5. CNN’s Impact and Applications

CNNs’ influence on image classification has been demonstrated through the ImageNet Large Scale Visual Recognition Challenge, an influential competition in CV. The winning models have consistently been CNN-based models, showcasing their supremacy [1]. Additionally, CNNs excel in numerous real-world applications. For instance, in medical image analysis, CNNs aid in identifying anomalies and diseases from X-rays, MRIs, and CT scans [4]. This transformative capability holds potential for improving healthcare diagnostics.

In autonomous vehicles, CNNs contribute to object detection, lane recognition, and pedestrian tracking [5]. Their ability to rapidly process visual information in real-time scenarios is crucial for safe and efficient autonomous navigation. Furthermore, CNNs enhance user experience and engagement on e-commerce platforms through image-based search and recommendation systems.

2.1.6. Conclusion

In summary, the introduction of DL models, specifically CNNs, has led to a significant transformation in image classification. Their ability to automatically learn and extract hierarchical features from raw image data has pushed CV into new and unexplored areas. The combination of convolutional layers, pooling mechanisms, and deep architectures has proven to be a winning formula, enabling CNNs to understand the intricate details and semantics within images. The implications of CNNs extend beyond classification, spreading throughout diverse sectors and revolutionising how we interact with visual data. As DL continues to evolve, CNNs stand as a testament to the remarkable synergy between technological advancement and human-inspired design.

2.2 Transfer Learning

Transfer learning has emerged as a pivotal technique in revolutionising the field of image classification within DL. This approach leverages pre-trained models, particularly CNNs, to transfer knowledge learned from one task or dataset to another. This method accelerates the training process, improves convergence, and enables accurate predictions, particularly when dealing with insufficient labelled data [6].

2.2.1. Understanding Transfer Learning

Transfer learning is based on the concept that knowledge gained from solving one problem can be applied to another related problem [7]. Within image classification, a pre-trained model, often trained on a large dataset, acts as a feature extractor. The model's initial layers learn fundamental features such as edges, textures, and basic shapes, which are generally applicable across various image classification tasks. Consequently, the pre-trained model acquires a comprehensive understanding of visual data, establishing itself as a valuable foundation for tackling new tasks [8].

Within Transfer Learning, two primary strategies are encompassed, fine-tuning and feature extraction. Fine-tuning modifies the weights of certain layers in the pre-trained model during the target dataset's training. This process adapts the model's acquired features to the distinct characteristics of a new dataset. Conversely, feature extraction involves leveraging the pre-trained model as a fixed feature extractor. While the model's convolutional layers are frozen, the focus is directed towards training the classifier layers using the target dataset. Both strategies facilitate the transfer of knowledge and contribute to enhanced performance [9].

2.2.2. Mechanisms of Transfer Learning

Overfitting: where the model memorises the training data instead of learning meaningful patterns

Transfer learning efficacy can be attributed several fundamental mechanisms. Firstly, the acquisition of low-level features. As the pre-trained model learns to detect edges, textures, and basic shapes, it develops a foundational understanding of visual patterns. When applied to a new dataset, these learned features offer an advantage in recognising fundamental visual components.

Secondly, domain adaptation. Images sourced from different datasets can display variations in lighting, viewpoint, and background. Transfer learning mitigates the challenge of domain shift by enabling the model to adapt to the specific attributes of the target dataset. This adaptability results in improved generalisation and robustness, especially in cases where the new dataset differs significantly from the model's original training data.

Furthermore, transfer learning capitalises on the idea of knowledge compression. Pre-trained models have undergone extensive training to refine their parameters and capture intricate features. This compressed knowledge is transferred to the new task, expediting convergence during training, and mitigating the risk of overfitting, which is particularly beneficial when dealing with limited target datasets [10].

2.2.3. Impact on Image Classification

The impact of transfer learning on image classification is profound and has extensive implications. One of its notable contributions is making DL accessible to a wider range of people. Training deep neural networks from scratch demands vast amounts of labelled data, significant computational resources, and time. Transfer learning overcomes these challenges by utilising existing knowledge, enabling practitioners to achieve remarkable results, especially with limited datasets.

Moreover, transfer learning accelerates innovation across diverse domains. In medical imaging, transfer learning expedites the development of diagnostic tools by leveraging models trained on expansive medical datasets. This approach contributes to accurate disease detection and prognosis assessment, ultimately enhancing patient care [11].

Within natural language processing (NLP), transfer learning has been extended to image-text tasks. Multimodal models leverage pre-trained image and text embeddings, enabling tasks such as image captioning and visual question answering. Transfer learning facilitates a deeper understanding of relationships between different types of data [12].

2.2.4. Conclusion

Transfer learning has revolutionised image classification. Through the utilisation of pre-trained models and techniques like feature extraction and fine-tuning, transfer learning accelerates model convergence, improves generalisation, and empowers applications across domains. Its impact extends beyond image classification, facilitating innovation in various fields and cultivating a close relationship between knowledge sharing and technological advancement.

2.3 Data Preprocessing and Augmentation

Data preprocessing and augmentation are highly significant in precise image classification. These crucial steps lay the foundation for training robust and effective DL models, particularly CNNs [13].

2.3.1. Data Preprocessing: A Prerequisite for Model Success

Data preprocessing encompasses a series of transformation that convert raw image data into a suitable format for training DL models. This initial phase involves tasks such as resizing images to a consistent resolution, normalizing pixel values, and handling missing or corrupted data [14]. Proper data preprocessing is a key principle for several reasons.

Firstly, consistent image sizes streamline model training and inference. CNNs rely on fixed input dimensions. Resizing images to a consistent resolution ensures compatibility with the network architecture and accelerates computation. Secondly, pixel normalization enhances model convergence by standardising pixel values to a common scale. This step mitigates the influence of varying intensity levels across images, preventing certain features from dominating the learning process.

Furthermore, data preprocessing addresses data quality issues. Handling missing or corrupted data prevents model instability and inaccurate predictions. Techniques such as data augmentation can be employed to fill gaps left by missing data, further enhancing the dataset's robustness.

2.3.2. Data Augmentation: Amplifying Dataset Diversity

Data augmentation amplifies the diversity of a dataset by creating variations of existing images through transformations. These transformations introduce controlled processes, such as rotations, flips, and translations, without altering the image's fundamental content [15]. Data augmentation serves multiple pivotal purposes in image classification.

Augmentation enhances a model's ability to generalise by exposing it to a wider array of scenarios. Presenting the model with slightly altered images during augmentation tackles overfitting. Through augmentation, models become more robust, proficient at handling variations in lighting, orientation, and other real-world conditions.

Furthermore, data augmentation significantly enlarges the dataset, effectively increasing its size. This is particularly valuable when working with limited data, as it artificially enhances the pool of available training samples. Larger datasets contribute to better model generalisation and prevent models from becoming excessively specialised to the training data.

Augmentation also addresses class imbalance which is a common challenge in image classification where certain classes have significantly fewer samples than others. Through generating variations of the minority classes, augmentation balances the class distribution, enabling the model to learn each class more effectively.

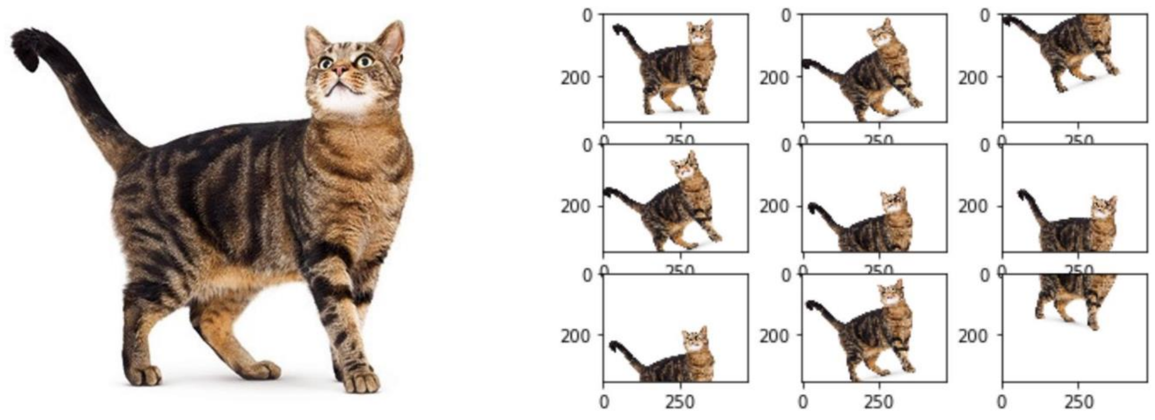


Figure 2: Data Augmentation Technique on Image of Cat

2.3.3. Strategies for Effective Data Augmentation

Several augmentation techniques have proven effective in enhancing image classification performance. Horizontal flips and vertical flips create mirror images, increasing dataset diversity. Random rotations introduce varying angles to images, simulating changes in orientation. Gaussian noise injections add a layer of randomness, simulating imperfections in data capture.

Colour transformations, such as adjusting brightness, contrast, and saturation, further enrich the dataset. These transformations expose the model to a wider range of colour distributions, making it more adaptable to different lighting conditions.

Moreover, augmentation strategies tailored to specific domains can be employed. For medical images, elastic deformations mimic the elasticity of human tissues, introducing realistic distortions. Geometric transformations, like affine transformations, simulate variations in perspective and viewing angles [16].

2.3.4. Impact on Image Classification Performance

The impact of data preprocessing and augmentation on image classification performance is profound. A thoroughly pre-processed and augmented dataset lays the foundation for training DL models that excel in real-world scenarios. Models trained on carefully pre-processed data exhibit improved convergence, higher accuracy, and greater resilience to variations in input data [17].

Furthermore, data augmentation acts as a controller, preventing models from becoming overly confident in the training data. Regularisation enhances model generalisation, enabling it to make accurate predictions on previously unseen images.

2.3.5. Conclusion

Data preprocessing and augmentation are crucial within image classification. Efficiently preprocessing data ensures that models are equipped to handle variations in pixel values, while augmentation diversifies the dataset, enabling models to generalise better and tackle overfitting. The collaboration of these techniques cultivates datasets that are more representative of real-world conditions, setting the stage for robust and high-performing deep learning models in image classification.

As DL continues to evolve, data preprocessing and augmentation remain integral components in the pursuit of accurate and reliable image classification. Optimising data quality and expanding dataset diversity enhances the capabilities of CNNs and contributes to advancements in various domains, such as healthcare and autonomous vehicles.

2.4 Model Evaluation Metrics

Assessing image classification models is a critical endeavour that guides the understanding of their refinement and effectiveness. Model evaluation metrics provide a quantitative perspective through which the performance of these models can be analysed.

2.4.1. Significance of Model Evaluation Metrics

Model evaluation metrics serve as the guiding tools within image classification. They encapsulate a model's performance essence, allowing for a comprehensive assessment of its strengths and weaknesses [18]. Without these metrics, the true impact of a model's predictions on real-world tasks remains obscure. This uncertainty can make the entire classification process ambiguous.

Model evaluation metrics play a pivotal role in promoting clarity and accountability in ML endeavours. They provide a standardised language through which the quality of different models can be compared objectively [19]. This objectivity is crucial for making informed decisions, whether in research, application, or deployment scenarios.

2.4.2. Types of Model Evaluation Metrics

An abundance of model evaluation metrics is available, each designed to emphasise different aspects of model performance. Frequently used metrics include accuracy, precision, recall, F1-score, and area under the Receiver Operating Characteristic curve (AUC-ROC).

Accuracy measures the percentage of correctly categorised instances out of the total. However, accuracy can be deceptive when addressing imbalanced datasets, where certain classes vastly outnumber others.

Precision measures the proportion of correctly predicted positive instances among all instances predicted as positive. Whereas Recall quantifies the ratio of correctly predicted positive instances among all actual positive instances. F1-score finds a balance between precision and recall, providing a single metric to assesses a model's overall performance.

AUC-ROC evaluates the ability of a model to distinguish between different classes. It plots the true positive (TP) rate against the false positive (FP) rate, creating a visual depiction of a model's ability to differentiate.

2.4.3. The Impact of Imbalanced Datasets

Imbalanced datasets frequently occur in real-world scenarios. For instance, in medical diagnosis, the number of healthy patients is often significantly higher than those with a specific disease, leading to an imbalanced distribution. Therefore, it's necessary for the careful consideration of evaluation metrics. Imbalanced datasets, where certain classes have significantly fewer samples than others, bring challenges to traditional metrics like accuracy. A model may appear deceptively accurate by classifying the majority class correctly while neglecting the minority class.

In situations involving imbalanced datasets, metrics like precision, recall, and F1-score become particularly valuable [20]. These metrics provide a nuanced understanding into a model's performance by focusing on positive instances, which is crucial when dealing with minority classes. This is especially relevant in applications like disease diagnosis, where accurate identification of rare cases holds substantial importance.

2.4.4. A Comprehensive Approach: Confusion Matrices

Confusion matrices provide a visual and quantitative summary of a model's classification performance. Confusion matrices provide a comprehensive perspective on a model's strengths and areas for improvement by displaying the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) for each class.

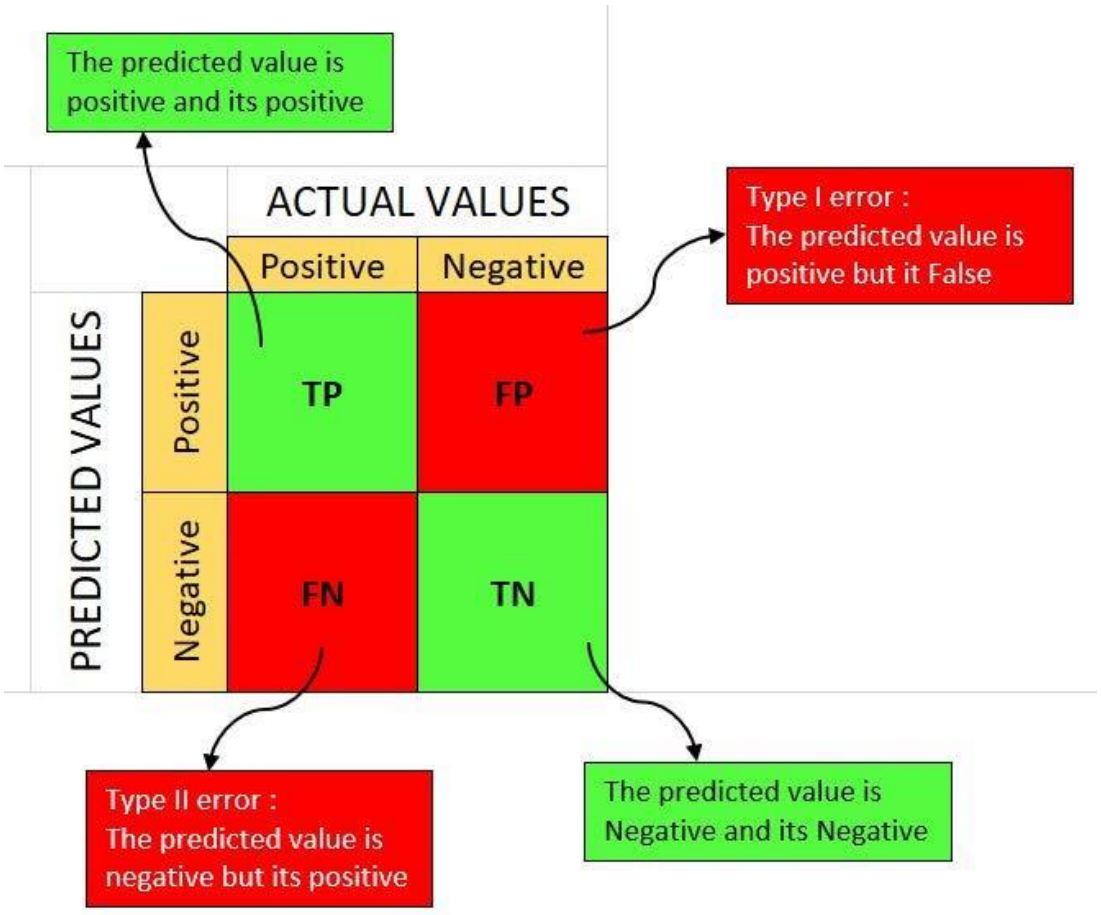

The predicted value is positive and its positive
ACTUAL VALUES
Positive
Negative
PREDICTED VALUES
Positive
Negative
TP
FP
FN
TN
Type I error : The predicted value is positive but it False
Type II error : The predicted value is negative but its positive
The predicted value is Negative and its Negative

Figure 3: Confusion Matrix

2.4.5. Real-world Implications and Considerations

The choice of model evaluation metrics is not applicable in all situations. It depends on the specific context, goals, and consequences of classification errors. In medical diagnoses, where FNs could have severe repercussions, recall might be prioritised. Whereas, in spam email detection, precision could be essential to avoid false positives.

2.4.6. Applications in Real-world Scenarios

In medical imaging, accurately diagnosing diseases like cancer is of high importance. Metrics like sensitivity (recall) and specificity assess a model's ability to identify TPs and TNs, respectively [21].

In autonomous vehicles safety is key and metrics like precision play a crucial role. Precision ensures that potential hazards are identified with high confidence, minimising false alarms, and ensuring passenger safety [22].

2.4.7. Conclusion

Model evaluation metrics stand as the guide that navigates the effectiveness of image classification. They provide a standardised language to assess performance, offering insights into models' strengths and weaknesses. These metrics empower informed decisions, promote transparency, and select models tailored to the specific context. As image classification progresses, the set of evaluation metrics will also evolve, ensuring that the search for accurate and effective models is backed by observed evidence.

3. Implementation

3.1 Data Loading and Preparation

The process of image classification begins with data loading and preparation, a critical phase that sets the foundation for model training and evaluation.

3.1.1. Dataset Description and Acquisition

This implementation utilises the DeepFashion dataset, which contains a wide range of images showcasing different clothing styles, patterns, and types. The categories within the dataset are used as labels for the image classification task. The dataset was chosen because of its relevance to fashion and its ability to enhance accurate image classification. Within this dataset, the Category and Attribute Prediction Benchmark stands as an important subset. This benchmark acts as a crucial subset, providing a way to assess the performance of predicting clothing categories and attributes. This prediction task is complex but essential in the field of analysing fashion images.

The Dataset Highlights:

- 289,222 Images: The DeepFashion dataset contains an extensive compilation of 289,222 images portraying diverse clothing articles, captured within varied real-world scenarios.
- 50 Clothing Categories: Encompassing a broad spectrum of clothing, the dataset features 50 distinct clothing categories. These categories encompass a diverse range, spanning from

formal ensembles to casual attire, thereby facilitating robust training and comprehensive testing.

- 1,000 Clothing Attributes: The dataset boasts an array of 1,000 clothing attributes, further enhancing its depth and variety. These attributes offer detailed descriptors, spanning colour, pattern, texture, and more.
- Bounding Boxes and Clothing Type: Each image is annotated with bounding boxes, precisely outlining the region occupied by the clothing. Additionally, the categorisation of clothing type associated with each image serves as a vital component of the dataset's labelling methodology.

DeepFashion is a fundamental element that shapes the quality and potential of the final trained model.

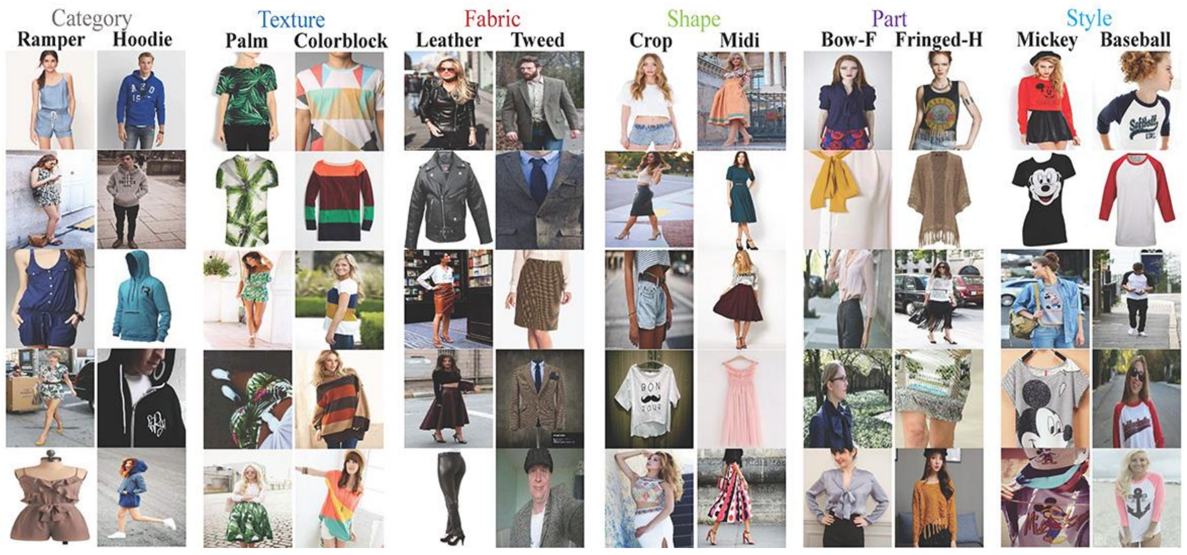


Figure 4: Category and Attribute Prediction Benchmark

The Google Drive API is utilised to acquire the DeepFashion dataset, streamlining the process of downloading and managing the dataset files. Four key files are involved in this dataset: **list_category_cloth.txt, list_category_img.txt, list_eval_partition.txt**, and **img.zip**. These files contain essential information about category labels, image paths, data partitions (train, validation, test), and the actual image files.

3.1.2. Data Preprocessing and Augmentation

Using raw, unprocessed image data often doesn't meet the quality standards needed when working with DL tasks. Occasionally, the initial data lacks the necessary qualities required for effective DL. Therefore, preprocessing steps are important, as they help refine the data so that it becomes more suitable and useful for the DL model.

Resizing for Uniformity: The images undergo resizing to establish a consistent and identical sizes. This is uniformity which serves a dual purpose: it aligns the images with the input specifications (size) of the selected pre-trained ResNet model, and it also optimises computational efficiency during the training procedure.

Enhancing Diversity with Augmentations: Data augmentation techniques are strategically used to increase the dataset's diversity and prevent overfitting. Random flips, rotations, and controlled variations in brightness and contrast are introduced. These adjustments mirror the variability encountered in real-world scenarios.

3.1.3. Handling Category Labels and Data Types

The dataset's category labels play a major role in model training and evaluation. To facilitate accurate classification, the category labels are extracted from the **list_category_cloth.txt** file. These labels are mapped to numerical indices, which act as the reference basis for the model's predictions.

Additionally, the data types (train, validation, or test) are read from the **list_eval_partition.txt** file and associated with each image. This information is crucial for splitting the dataset into appropriate subsets for training, validation, and testing.

3.1.4. Testing

In the domain of technical evaluation, a crucial aspect of the data loading and preparation process involves comprehensive testing. This step ensures that the methods applied to select, organise, and enhance the dataset function as intended. The testing phase is essential to confirm that the dataset is correctly structured for subsequent model training and evaluation. This is achieved by using the capabilities of the Fastai library and the DeepFashion dataset as a testing ground.

For instance, when resizing images for uniformity, the testing process checks whether the resizing operation successfully generates images of consistent dimensions. The code snippet below demonstrates how this works:

```
image_tensor = transform(image)
```

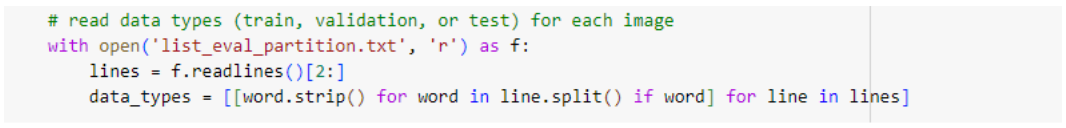
Listing 1: Uniformity Code Extract

The accuracy of extracting category labels is cross verified by comparing the extracted labels with the entries in the **'list_category_cloth.txt'** file:

```
# read category names
with open('list_category_cloth.txt', 'r') as f:
    lines = f.readlines()[2:]
    categories = [line.split()[0] for line in lines]
```

Listing 2: Reading Category Names Code Extract

Furthermore, the effectiveness of data partitioning into training, validation, and test sets is validated by comparing the distribution of dataset entries across these partitions:



```
# read data types (train, validation, or test) for each image
with open('list_eval_partition.txt', 'r') as f:
    lines = f.readlines()[2:]
    data_types = [[word.strip() for word in line.split() if word] for line in lines]
```

Listing 3: Reading and Parsing Image Data Types Code Extract

Employing these testing practices ensures that the data loading and preparation process aligns with its intended goals. This validation process provides confidence that the data will be accurately processed, which in establishes a solid foundation for the subsequent stages of model training, evaluation, and interpretation.

3.1.5. Summary

In summary, the data loading and preparation phase is a foundational step that lays the groundwork for successful image classification. Category labels and data types are extracted and associated, ensuring proper organisation and access to the data. Thoroughly selecting and preparing the dataset establishes a strong foundation for the later stages of model training, evaluation, and interpretation.

3.2 DataBlock and DataLoaders

The Fastai library's DataBlock and DataLoaders functionalities significantly streamline the intricate task of converting raw data into a usable dataset. This involves a combination of data processing, augmentation, and organisation.

3.2.1. Organising Data for Model Consumption

To enable efficient and organised access to the dataset during model training, the data is structured using the DataBlock API provided by the Fastai library.

Data Blocks: The complexities involved in managing inputs (images) and outputs (categories) were encapsulated within data blocks, which outlined the transformation processes. This ensured a logical and systematic approach to data handling.

DataLoaders: The use of DataLoaders further simplifies the process of data management. DataLoaders handle the logic of creating batches of data, shuffling, and applying necessary transformations. These DataLoader instances serve as the bridge between the data and the model, facilitating a smooth and uninterrupted exchange of information during training and evaluation processes.

Partitioning with Precision: The dataset was partitioned into training, validation, and test sets, a critical step to ensure proper model evaluation and validation. The DataBlock facilitated this partitioning, enhancing the model's ability to generalise.

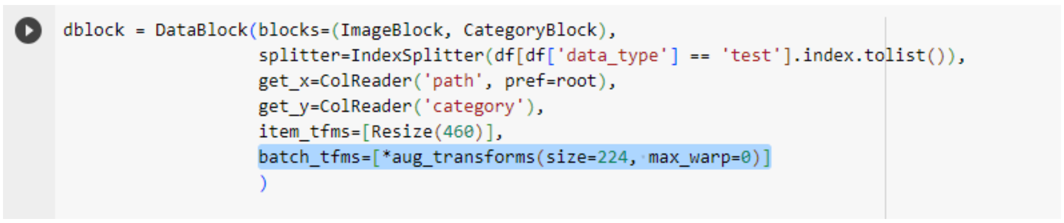
3.2.2. Defining DataBlock: Blueprint for Data Processing

Fundamental to the data transformation process is the DataBlock API. This versatile tool plays a crucial role in facilitating the conversion of raw images into a format suitable for DL tasks. DataBlock is like a blueprint, detailing the sequence of steps required to shape images into tensors (a structure that perfectly aligns with the model's framework). Within this blueprint, data blocks (modular units tailored to specific data attributes) play a crucial role. For the task of image classification, two foundational blocks emerge: the **ImageBlock** and the **CategoryBlock**.

The ImageBlock assumes a major role in handling images by carefully converting them into tensors that can be used with the selected DL architecture. Meanwhile, the CategoryBlock manages categorical labels, which are essential for classification. These two blocks cooperate to create a well-structured foundation within DataBlock. This foundation facilitates the connection of input images to their respective categorical labels, establishing the groundwork for successful model training.

3.2.3. Data Augmentation and Transformation

Augmentation plays a vital role in **transforming** data. Augmentation adds variety and resilience to the dataset. Within the DataBlock framework, augmentation seamlessly integrates into the data processing pipeline, strengthening the dataset's defences against overfitting, and enhancing its capacity to handle diverse scenarios. For instance, *Listing 4* showcases the use of augmentation techniques within the DataBlock definition:



```
dblock = DataBlock(blocks=(ImageBlock, CategoryBlock),
                   splitter=IndexSplitter(df[df['data_type'] == 'test'].index.tolist()),
                   get_x=ColReader('path', pref=root),
                   get_y=ColReader('category'),
                   item_tfms=[Resize(460)],
                   batch_tfms=[*aug_transforms(size=224, max_warp=0)]
                   )
```

Listing 4: Batch-Level Data Augmentation Configuration Code Extract

This snippet demonstrates how augmentation techniques are seamlessly integrated into the data loading process, allowing the model to encounter a range of image variations during training.

3.2.4. Structuring DataLoaders: Data Flow Management

In data flow management, DataLoaders emerge as a crucial component, seamlessly linking processed data with to the DL model. This pipeline integrates raw data with the model’s architecture, which ensures carefully prepared and optimised data consumption.

Within this project, DataLoaders use **partitioning with precision.** This segmentation aligns with real-world training conditions, enabling the model to generalise effectively beyond its initial training phase. Consequently, the model becomes better equipped to handle the diverse array of clothing images it during deployment.

3.2.5. Adapting DataBlock and DataLoaders to the Project

In this project, the versatility of DataBlock and DataLoaders played a pivotal role, seamlessly adapting to its distinct requirements. During the implementation process, these functionalities were carefully customised to align with the specific demands of image classification within the DeepFashion dataset. Specifically, the DataBlock blueprint was optimised to ensure the integration of images and categorical labels, resulting in a cohesive way of dataset representation. Augmentation techniques were selected and incorporated, enhancing the dataset by introducing valuable variations and complexity.

Additionally, the construction of DataLoaders were implemented to achieve an optimal splitting of the dataset. By aligning the balance between training and validation subsets, a data distribution that closely resembles real-world scenarios was established. This strategic arrangement facilitated robust model evaluation and enabled the model to generalise effectively to unseen data.

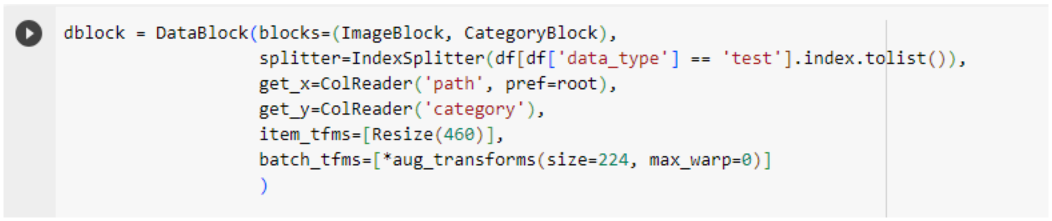
3.2.6. Leveraging DeepFashion's Potential

The DeepFashion dataset incorporates the Category and Attribute Prediction Benchmark which introduces an additional layer of complexity. This benchmark serves as both an evaluative metric and a guiding framework for the model's training route. By integrating clothing attributes and categories seamlessly, the model enhances its ability to differentiate and recognise even the subtle characteristics. This refinement process sharpens the model's capacity to comprehensively understand fashion, enabling it to navigate the intricate landscape of clothing features with precision.

3.2.7. Testing

As part of the evaluation process, thorough testing of the DataBlock and DataLoaders implementation was conducted to ensure their effectiveness in preparing and managing the dataset. This involved assessing the integration of augmentation techniques, the seamless connection between images and categorical labels, and the accurate splitting of the dataset into training and validation subsets.

The use of augmentation techniques within the DataBlock definition is evident in *Listing 5:*

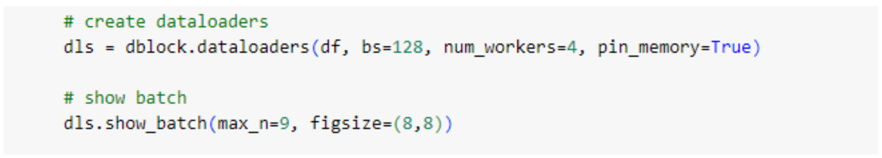


```
dblock = DataBlock(blocks=(ImageBlock, CategoryBlock),
                   splitter=IndexSplitter(df[df['data_type'] == 'test'].index.tolist()),
                   get_x=ColReader('path', pref=root),
                   get_y=ColReader('category'),
                   item_tfms=[Resize(460)],
                   batch_tfms=[*aug_transforms(size=224, max_warp=0)]
                   )
```

Listing 5: DataBlock definition for Data Augmentation and Data Loading Code Extract

Listing 5 showcases how augmentation strategies are integrated into the data loading process. By employing techniques like resizing and applying a maximum warp transformation, the model is exposed to diverse variations during training, enhancing its robustness and adaptability to real-world scenarios.

To validate the DataLoader creation process, *Listing 6* is used:



```
# create dataloaders
dls = dblock.dataloaders(df, bs=128, num_workers=4, pin_memory=True)

# show batch
dls.show_batch(max_n=9, figsize=(8,8))
```

Listing 6: Data Preparation and Batch Visualisation Code Extract

By defining appropriate batch sizes, specifying the number of workers, and enabling memory pinning, the DataLoaders were optimised for efficient data flow during model training and evaluation.

In conclusion, the testing of the DataBlock and DataLoaders implementation confirmed their effectiveness in seamlessly transforming raw data into a structured, augmented, and well-prepared dataset. Through careful integration of augmentation techniques and precision in dataset partitioning, the model's ability to recognise intricate clothing attributes and categories was strengthened. These essential components have paved the way for the subsequent stages of model training, evaluation, and interpretation within this project.

3.3 Model Training

3.3.1. ResNet-50 Architecture: a CNN

In this implementation, ResNet-50 plays a crucial component in model training. Renowned for its deep architecture and effectiveness in diverse CV, ResNet-50 features a sophisticated structure featuring 50 layers and specialised residual blocks. These blocks tackle challenges such as vanishing gradients and enhance feature extraction.

ResNet-50 operates various training parameters, encompassing learning rates and weight decay. By utilising gradient descent, this mechanism adjusts the model weights during training iterations, thereby refining its parameters. As a result, the model gains the ability to distinguish intricate clothing attributes and categories. This highlights the synergy between the model's architecture and its proficiency in fashion image analysis.

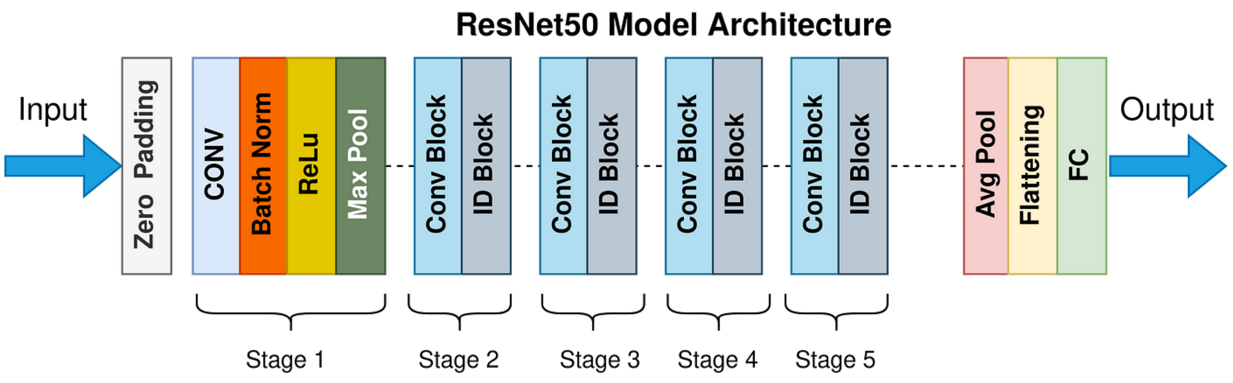
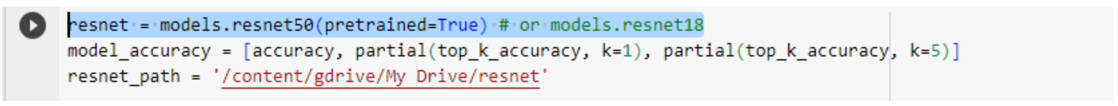


Figure 5: ResNet40 Model Architecture

3.3.2. Pre-Trained Initialisation

Within the scope of this FRS, the initialisation of ResNet-50 with pre-trained weights is accomplished through the utilisation of the **models.resnet50 (pretrained=True)** function.



```
resnet = models.resnet50(pretrained=True) # or models.resnet18
model_accuracy = [accuracy, partial(top_k_accuracy, k=1), partial(top_k_accuracy, k=5)]
resnet_path = '/content/gdrive/My Drive/resnet'
```

Listing 7: Pre-defined ResNet50 model architecture Code Extract

This technique utilises knowledge extracted from the dataset. This procedure equips the model with a fundamental understanding of essential features, enabling it to logically adapt to the dataset's nuances. This concept aligns with the principles of transfer learning, where the model leverages on its prior understanding while refining its abilities for outfit classification.

The **cnn_learner** function, an integral component of the Fastai library, plays a pivotal role in this process. It simplifies the creation of a learner object that encompasses the entire training, evaluation, and prediction pipeline. This function attends to intricate details, such as configuring the architecture, loss function, optimisation method, and metrics.

In the call to the cnn_learner function, ResNet-50 is seamlessly integrated as the backbone architecture. Employing the model's layers for feature extraction and transformation, this process ensures an efficient flow of image data throughout the neural network. Subsequently, the cnn_learner function constructs the learner object **learn** encapsulating the model in a format ready for training.

By strategically embedding the pre-trained ResNet-50 into the cnn_learner function, the model's capability to extract significant image features is harnessed. This is particularly crucial when training with fashion images. This approach streamlines training while significantly enhancing the model's potential to excel in the intricate task of fashion classification, showcasing the synergy between transfer learning and model architecture.

3.3.3. Transfer Learning Paradigm

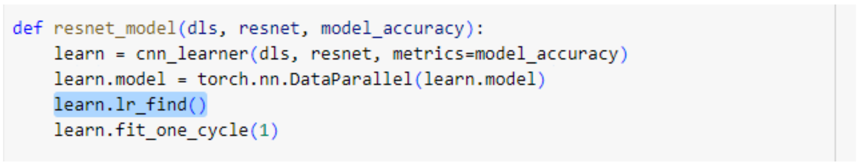
In this implementation, transfer learning is key, leveraging on the pre-trained ResNet-50 model's knowledge acquired from datasets like ImageNet. As a result, the model has already possessed a wide understanding of diverse images features across various categories. This strategy accelerates the model's understanding of basic image features such as edges and textures. Notably, this technique excels when applied to the DeepFashion dataset, abundant with fashion-focused images.

3.3.4. Training Regimen and Hyperparameters

The model's training process is guided by a systematic approach and carefully selected settings.

1. Learning Rate Strategy

One crucial aspect of training is the learning rate schedule, determining the speed at which the model adjusts to the data. Learning rates play a critical role in guiding the optimisation process of model during training. An appropriate learning rate is essential for achieving effective convergence and avoiding issues like slow convergence or overshooting.



```
def resnet_model(dls, resnet, model_accuracy):
    learn = cnn_learner(dls, resnet, metrics=model_accuracy)
    learn.model = torch.nn.DataParallel(learn.model)
    learn.lr_find()
    learn.fit_one_cycle(1)
```

Listing 8: Learning Rate Finder Plot Code Extract

The **learn.lr_find()** function is utilised to explore and identify a suitable learning rate for this model. It discovers an optimal learning rate by plotting the loss against a range of learning rates. This plot helps to identify the range of learning rates where the loss decreases most rapidly, indicating that the model is learning efficiently.

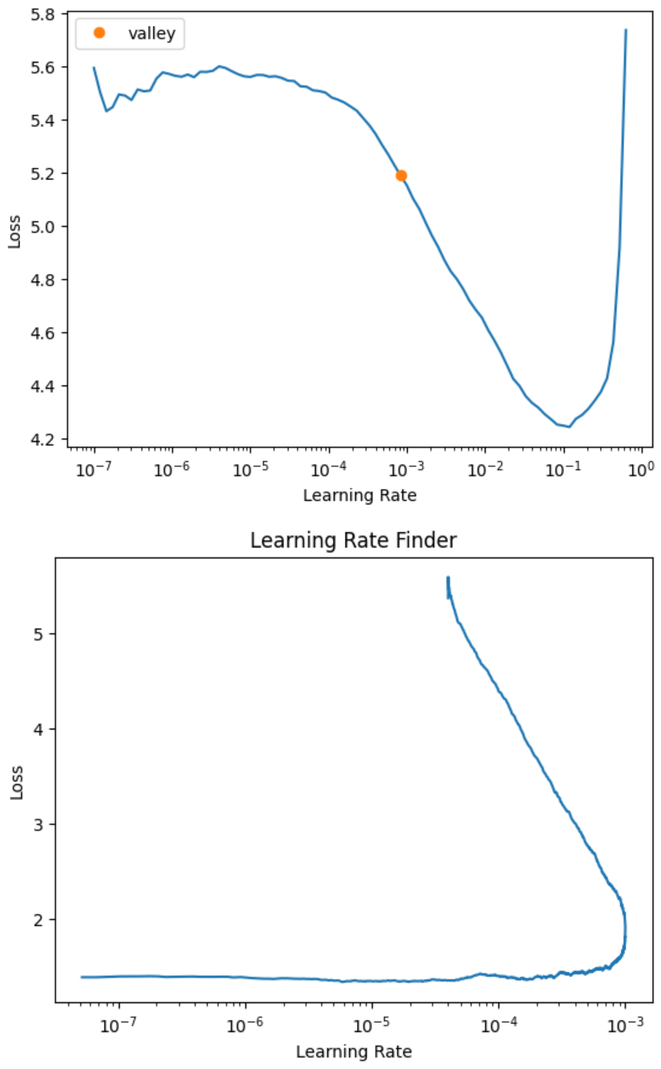


Figure 6: Visual Learning Rate Plots

This plot illustrates how different learning rates impact our model's performance.

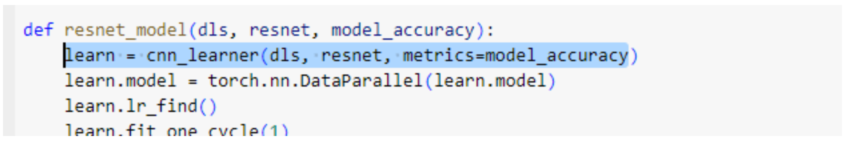
Observations from the graph include:

- Low Learning Rate Fluctuation (10^-7 to 10^-4): At very low rates, the loss fluctuates between 5.6 and 5.4, suggesting instability and slow learning.
- Decrease in Loss (10^-4 to 10^-1): The loss decreases as the learning rate goes from 10^-4 to 10^-1. The valley point, around 5.2, occurs at a rate of 10^-3. This is where the model learns effectively.
- Loss Increase at Higher Rates (10^-1 to 10^0): Beyond 10^-1, the loss rises to about 5.8. High rates lead to divergent training and lower accuracy.

In conclusion, a learning rate between 10^-4 and 10^-1 is ideal for stable and effective model training. This balance ensures improved convergence, lower loss, and enhanced performance for this FRS.

2. Monitoring Progress

During training, it is crucial to monitor how well the model is learning. Accuracy is primarily used as a metric to understand classification performance. The metrics parameter in the **cnn_learner** function help track this progress:



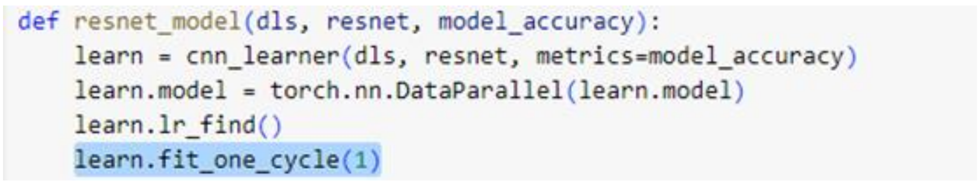
```
def resnet_model(dls, resnet, model_accuracy):
    learn = cnn_learner(dls, resnet, metrics=model_accuracy)
    learn.model = torch.nn.DataParallel(learn.model)
    learn.lr_find()
    learn.fit_one_cycle(1)
```

Listing 9: Model Initialisation with Fastai's CNN Learner Code Extract

3. Training Iterations

Underfitting: when a model is too simple to capture the underlying patterns in the data.

Determining the number of training epochs significantly impacts the model's learning. Achieving a suitable equilibrium between avoiding underfitting and preventing overfitting is crucial. In this scenario, the model is trained for a total of four epochs.

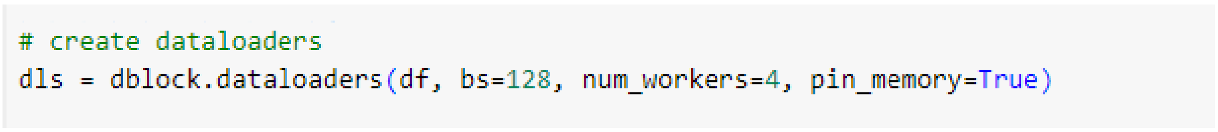


```
def resnet_model(dls, resnet, model_accuracy):
    learn = cnn_learner(dls, resnet, metrics=model_accuracy)
    learn.model = torch.nn.DataParallel(learn.model)
    learn.lr_find()
    learn.fit_one_cycle(1)
```

Listing 10: Model Training using the One-Cycle Learning Rate Schedule Code Extract

4. Efficient Batch Processing

Batch size, which defines the number of training samples processed per iteration, plays a role in efficient training. The batch size is set when creating **dataloaders** using the DataBlock API. Specifically, in *Listing 11*:



```
# create dataloaders
dls = dblock.dataloaders(df, bs=128, num_workers=4, pin_memory=True)
```

Listing 11: Data Loaders Initialisation Code Extract

Here, **bs=128** specifies the batch size. The batch size of 128 indicates that each training iteration will process 128 images at once. The choice of batch size can impact both training efficiency and model performance.

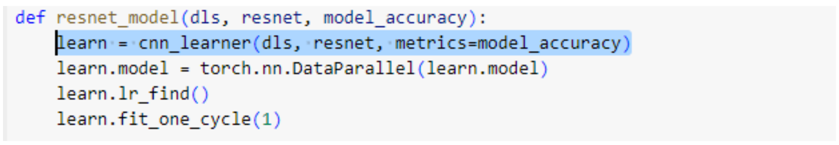
While larger batch sizes can expedite training, overly large batches might lead to memory issues. On the other hand, a smaller batch size can potentially help the model generalise better, as it updates weights more frequently based on fewer examples. However, training might be slower due to the increased number of iterations.

Upon research, a batch size of 128 is a common choice and is often used as a starting point for many DL tasks. For larger datasets, such as DeepFashion, and models, a batch size of 128 can work well. It provides a good balance between training efficiency and model generalisation. By fine-tuning these training parameters and adhering to a structured approach, the learning process is optimised, achieving reliable and accurate results on the DeepFashion dataset.

3.3.5. Validation

In the pursuit of a reliable FRS, it's crucial to ensure that the model can effectively handle new and unseen data during validation. Validation metrics measure how accurately the model predicts unseen clothing categories. For this purpose, accuracy as a validation metric is prioritised, quantifying the proportion of correct predictions made by the model. Throughout training, accuracy is continuously monitored to ensure that the FRS maintains a strong predictive performance.

Validation provides insight into how well the model performs on unseen data. By prioritising accuracy as the validation metric, the proportion of correctly predicted labels can be obtained. The metrics parameter in the **cnn_learner** function enables continuous monitoring of accuracy during training:



```python
def resnet_model(dls, resnet, model_accuracy):
    learn = cnn_learner(dls, resnet, metrics=model_accuracy)
    learn.model = torch.nn.DataParallel(learn.model)
    learn.lr_find()
    learn.fit_one_cycle(1)
```

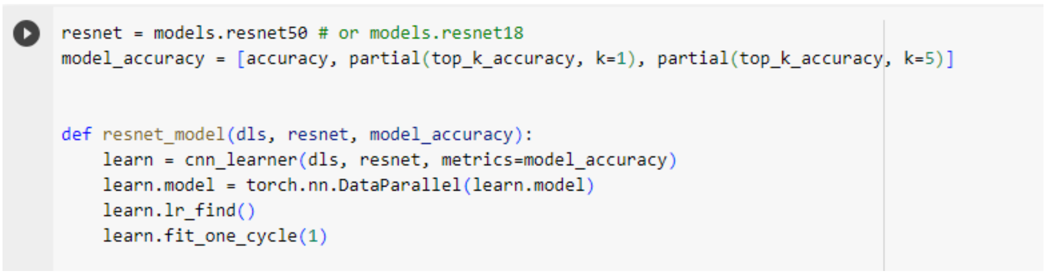
Listing 12: Model Initialisation with CNN Learner Code Extract

By monitoring accuracy as the model trains, how the model’s predictive performance evolves over time can be observed. This information is crucial for making decisions about model adjustments, fine-tuning, and potentially avoiding overfitting or underfitting issues.

3.3.6. Fine-Tuning: Tailoring to Fashion

With the appropriate learning rate established, fine-tuning is initiated. This process unfolds over multiple epochs, which are training cycles. In each epoch, the model's internal settings undergo continuous adjustments to learn and adapt to the intricacies within fashion data. This iterative refinement is carried out through a carefully designed optimisation technique.

During every epoch, the model progressively hones its ability to identify the intricate characteristics that define various clothing attributes and categories. This ongoing adaptation process significantly enhances the model's proficiency in recognising even the subtlest details within fashion images. The result is a heightened capability to accurately categorise and understand diverse fashion elements, ultimately enabling more precise and personalised outfit recommendations.



```python
resnet = models.resnet50 # or models.resnet18
model_accuracy = [accuracy, partial(top_k_accuracy, k=1), partial(top_k_accuracy, k=5)]


def resnet_model(dls, resnet, model_accuracy):
    learn = cnn_learner(dls, resnet, metrics=model_accuracy)
    learn.model = torch.nn.DataParallel(learn.model)
    learn.lr_find()
    learn.fit_one_cycle(1)
```

Listing 13: ResNet Model Training Function Code Extract

Listing 13 shows the **fit_one_cycle** function, which orchestrates the fine-tuning by refining the model for 1 epoch. Initially, I fine-tuned the model by running it for 5 epochs. As each epoch unfolds, the model becomes attuned to various characteristics of fashion, like unique patterns or textures. However, due to the time it took to load and train the model for 5 epochs, I decided to reduce the training time.

By using these techniques together, the FRS becomes better at understanding intricate fashion aspects. This results in more accurate recommendations that match users' individual style preferences.

3.4 Model Evaluation and Interpretation

3.4.1. Chosen Evaluation Metrics and Rationale

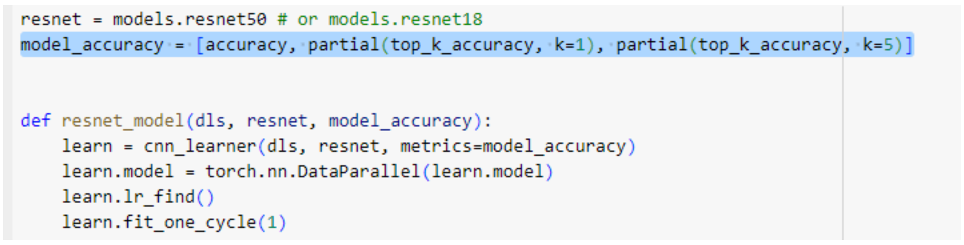
In assessing the performance of the FRS, key metrics that capture its accuracy and predictive capability are relied on:

1. **Accuracy:** This metric quantifies the proportion of accurately categorised fashion ensembles within the DeepFashion dataset. It offers an overall measure of the system's correctness by reflecting its ability to assign the accurate category label to each outfit image.

2. **Top-k Accuracy:** This metric assesses the system by verifying if the accurate category label is present among the top-k predicted categories. Top-k accuracy holds significance in outfit recommendation, accommodating scenarios where an outfit may belong to multiple relevant categories. In this implementation, when k is defined as 5, the model's accuracy is affirmed if the correct category is among the top 5 predictions.

3.4.2. Computation of Evaluation Metrics

The accuracy and top-k accuracy metrics are calculated using functions provided by the Fastai library. These functions process the FRS's predictions along with the actual labels to quantitatively evaluate both the system's correctness and its ability to effectively rank possible outfit categories.



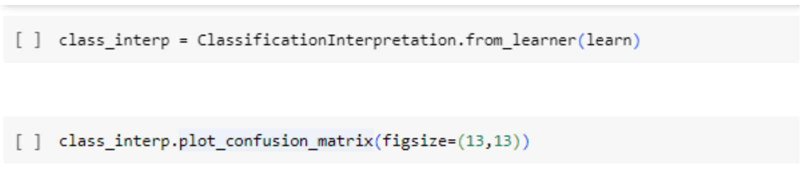
```python
resnet = models.resnet50 # or models.resnet18
model_accuracy = [accuracy, partial(top_k_accuracy, k=1), partial(top_k_accuracy, k=5)]


def resnet_model(dls, resnet, model_accuracy):
    learn = cnn_learner(dls, resnet, metrics=model_accuracy)
    learn.model = torch.nn.DataParallel(learn.model)
    learn.lr_find()
    learn.fit_one_cycle(1)
```

Listing 14: Accuracy Metrics Configuration for Evaluation Code Extract

3.4.3. Interpretation Using Confusion Matrices

In this implementation, confusion matrices are utilised to further understand the FRS. These matrices visually represent the alignment between predicted and actual labels, offering insights into the FRS's prediction patterns. Each row corresponds to an actual category, while each column corresponds to a predicted category. The diagonal entries represent correct predictions, and off-diagonal entries indicate misclassifications.



```python
[ ]  class_interp = ClassificationInterpretation.from_learner(learn)

[ ]  class_interp.plot_confusion_matrix(figsize=(13,13))
```

Listing 15: Confusion Matrix Visualisation Code Extract

Listing 15 employs the **ClassificationInterpretation** class to generate the confusion matrix, followed by visualisation using the **plot_confusion_matrix** method.

Confusion matrices provide a detailed assessment of the FRS's performance strengths and areas for enhancement. They unveil both the system's successes and instances of misclassifications, offering insights into potential biases or difficulties. A thorough review of the matrix assists developers in identifying categories where the model could face challenges, thereby directing efforts for improvement.

The **plot_confusion_matrix** function demonstrates how this approach is effective in analysing the system's performance. This makes it a valuable tool for improving the system's overall accuracy and dependability.

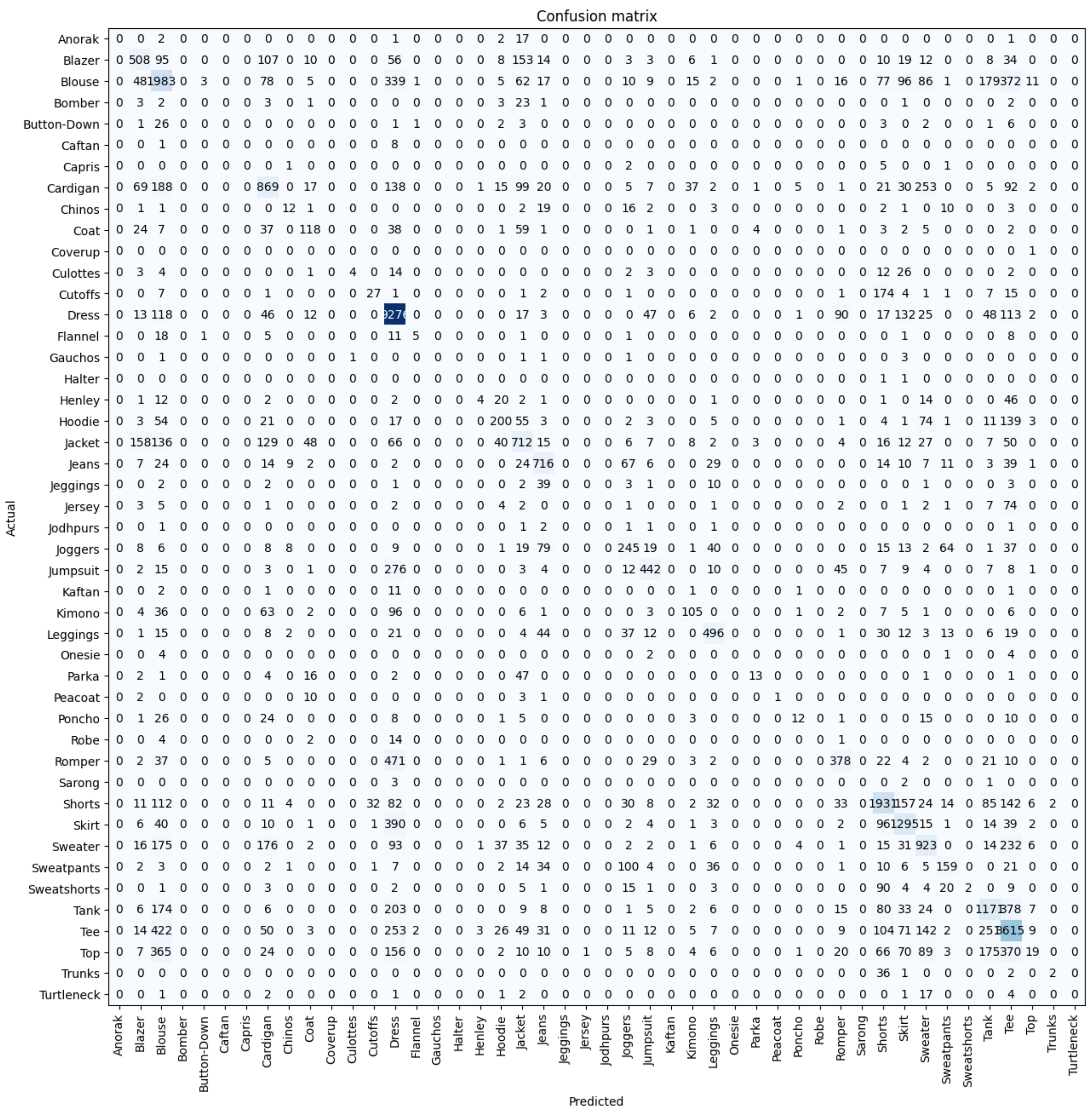


Figure 7: Confusion Matrix Visualisation

The dual highlighting of the "dress" category in both the actual and predicted axes emphasises that the accuracy of predictions for this category is significant and important for the overall success of the model. This emphasis indicates that the model's predictions for "dress" are being carefully examined.

Any discrepancies or misclassifications in this category can be analysed more deeply to understand potential challenges the model faces, whether related to image characteristics, patterns, textures, or other factors.

3.4.4. Interpretation Using Top Losses Visualisation

The visualisation of top losses is a pivotal resource for interpretation. It offers insight into images where the FRS encountered the highest prediction errors. These images often point out categories where the system faced difficulties. Analysing these instances and their corresponding predictions identified challenging categories, enabling a focused effort on enhancing accuracy within those areas.

The implementation of this visualisation demonstrates its ability to reveal regions where the FRS struggles. For instance, if the system misclassifies a striped shirt as a dress, it highlights a challenge in recognising intricate patterns. This technique bridges the gap between model predictions and fashion items, helping in identifying limitations and refining the system's accuracy. Through the **plot_top_losses** function, developers can efficiently assess these instances, enabling focused adjustments to enhance the system's performance in recommending accurate fashion outfits.

```
class_interp.plot_top_losses(9, figsize=(15,11), largest=False)
```

Listing 16: Top Losses Visualisation Code Extract

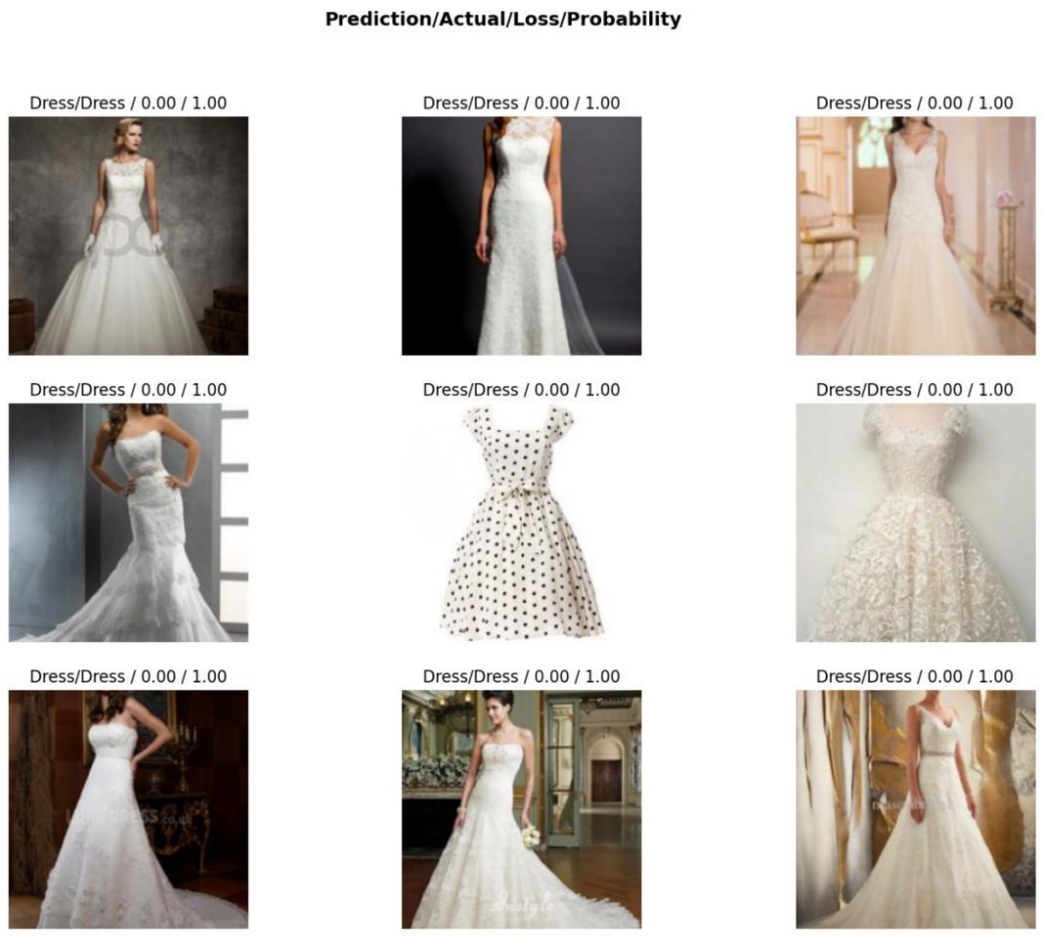

Prediction/Actual/Loss/Probability
Dress/Dress / 0.00 / 1.00
Dress/Dress / 0.00 / 1.00
Dress/Dress / 0.00 / 1.00
Dress/Dress / 0.00 / 1.00
Dress/Dress / 0.00 / 1.00
Dress/Dress / 0.00 / 1.00
Dress/Dress / 0.00 / 1.00
Dress/Dress / 0.00 / 1.00
Dress/Dress / 0.00 / 1.00

Figure 8: Top Losses Visualisation

The output "Dress/Dress/0.00/1.00" obtained from the **class_interp.plot_top_losses(9, figsize=(15,11), largest=False)** function signifies the following:

1. **Dress/Dress:** This indicates that the model predicted the true class label as "dress" for the outfit in question.

2. **0.00:** The first number represents the predicted probability/confidence score assigned by the model to the predicted class, which in this case, is "dress." A value of "0.00" suggests

that the model assigned a probability of 0% to the predicted class label, indicating that it was very confident in its prediction.

3. **1.00:** The second number represents the true probability/confidence score of the actual class label, which is also "dress." A value of "1.00" indicates that the model assigned a probability of 100% to the true class label, reaffirming its high confidence in the correctness of the true label.

In essence, the output "Dress/Dress/0.00/1.00" reveals that the model correctly predicted the class label "dress," with an extremely low confidence score (0.00%) assigned to the prediction. Simultaneously, the model assigned a perfect confidence score (100%) to the true class label "dress." This suggests that the model was confident in its prediction and correctly identified the outfit as a "dress."

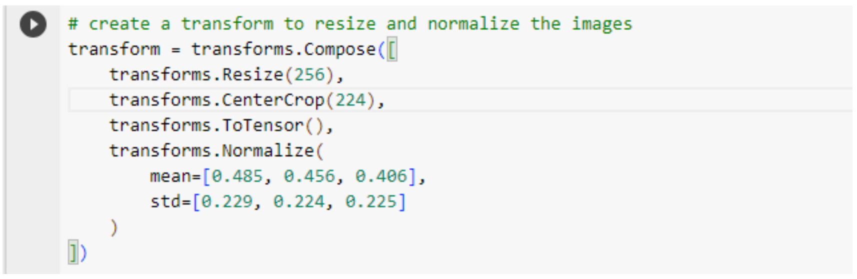
3.5 Image Embeddings

3.5.1. Transformation

The transformation process is a critical step in generating image embeddings using the trained ResNet model. Its purpose is to preprocess input images appropriately before they are passed to the model for embedding extraction.

In the implementation, the transformation pipeline is created using the **transforms.Compose** class. This pipeline includes resizing the images to dimensions of 256x256 pixels, followed by central cropping to 224x224 pixels (to align with ResNet's requirements). Subsequently, the images are then converted into tensors and their pixel values are normalized based on the mean and standard deviation.

This transformation process ensures that images are standardised and consistently represented, facilitating the ResNet model's accurate extraction of essential fashion attributes.



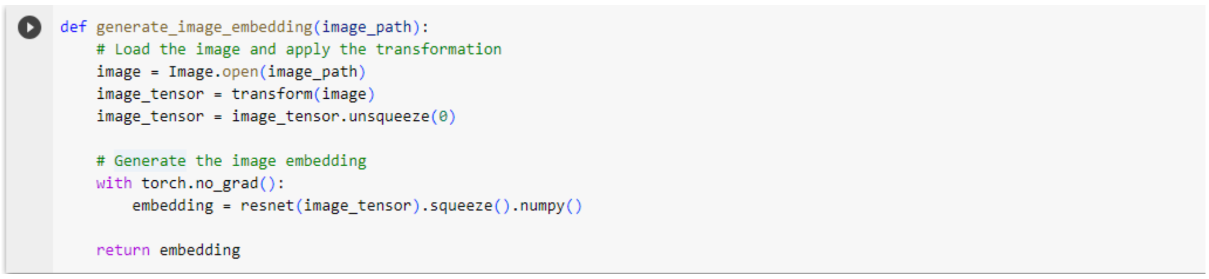
```
# create a transform to resize and normalize the images
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])
```

Listing 17: Image Transformation and Normalization Configuration Code Extract

3.5.2. Embedding generation

The generation of image embeddings is a pivotal phase within the FRS leveraging the trained ResNet model's abilities to capture intricate image characteristics. During this process, images undergo a carefully designed transformation pipeline involving resizing, cropping, and normalization to align with the model's input requirements. The **generate_image_embedding** function arranges this process by generating concise numerical embeddings, obtained by processing transformed images through the

ResNet model. These embeddings serve as condensed representations of fashion outfits, capturing vital visual attributes.



```
def generate_image_embedding(image_path):
    # Load the image and apply the transformation
    image = Image.open(image_path)
    image_tensor = transform(image)
    image_tensor = image_tensor.unsqueeze(0)

    # Generate the image embedding
    with torch.no_grad():
        embedding = resnet(image_tensor).squeeze().numpy()

    return embedding
```

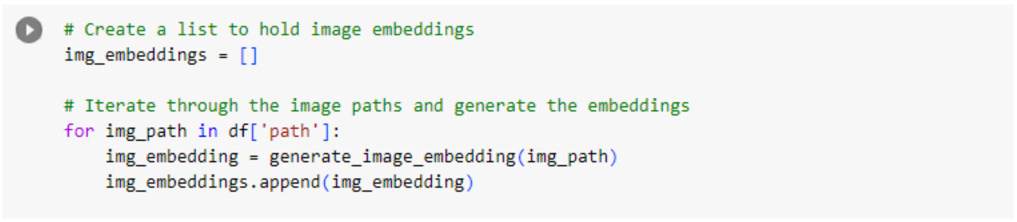
Listing 18: Image Embedding Generation Function Code Extract

Listing 18 exemplifies the image embedding generation process, where the **transform** pipeline is applied to the input image. The transformed image is then passed through the ResNet model, and the resulting embedding is obtained.

The significance of generating image embeddings using the trained ResNet model lies in its ability to encapsulate the essence of fashion items within condensed numerical vectors. These embeddings facilitate efficient similarity-based searches, enabling the system to better understand and match intricate fashion patterns. This functionality equips users with personalised and relevant outfit recommendations, enhancing the FRS's overall efficacy.

3.5.3. Embedding Indexing

The generated image embeddings are stored in a list, **img_embeddings**. These embeddings clearly encapsulate the visual characteristics of each fashion outfit.



```
# Create a list to hold image embeddings
img_embeddings = []

# Iterate through the image paths and generate the embeddings
for img_path in df['path']:
    img_embedding = generate_image_embedding(img_path)
    img_embeddings.append(img_embedding)
```

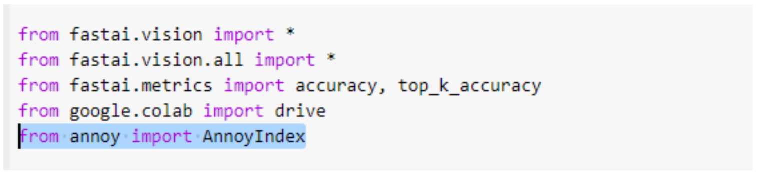
Listing 19: Image Embedding Generation and Collection Code Extract

Beyond simple pixel values, these embeddings condense intricate patterns, textures, and stylistic features into concise numerical representations. Moreover, beyond just recommendations, the indexing of embeddings enables deeper insights into fashion trends and individual preferences. This process transforms raw images into navigable vectors, amplifying the system's capacity to deliver relevant, timely, and visually coherent fashion suggestions to users.

3.5.4. Approximate Nearest Neighbor (ANN) Index

The Annoy library is employed to build an efficient index of the image embeddings. This index expedites the retrieval of similar images based on similarity measurements. Each embedding is integrated into the index, enabling quick searches for similar outfits.

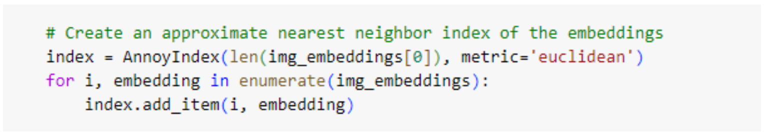
1. **Initialisation:** The AnnoyIndex is instantiated, specifying the dimensionality of the embeddings. This dimensionality corresponds to the length of each embedding vector.



```
from fastai.vision import *
from fastai.vision.all import *
from fastai.metrics import accuracy, top_k_accuracy
from google.colab import drive
from annoy import AnnoyIndex
```

Listing 20: Importing AnnoyIndex Code Extract

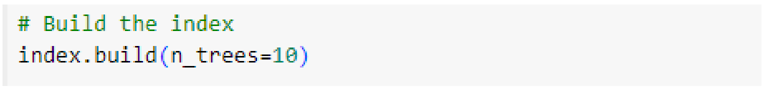
2. **Adding Items:** The index is populated with items, where each item is associated with its respective embedding. In the FRS, this involves iterating through the list of image embeddings and adding them to the index.



```
# Create an approximate nearest neighbor index of the embeddings
index = AnnoyIndex(len(img_embeddings[0]), metric='euclidean')
for i, embedding in enumerate(img_embeddings):
    index.add_item(i, embedding)
```

Listing 21: Annoy Index Building and Population Code Extract

3. **Building the Index:** Once all items are added, the index is built. This process involves constructing tree-based structures that enable efficient search and retrieval operations.



```
# Build the index
index.build(n_trees=10)
```

Listing 22: Annoy Index Building Process Code Extract

The Annoy index expedites finding visually similar fashion outfits. Employing tree-based structures reduces the search area, making it feasible to perform quick similarity searches even in large datasets. The index enhances the FRS's responsiveness, enabling users to receive instant recommendations based on their input images.

The line **index.build(n_trees=10)** constructs the index with 10 trees. Using this value found a good balance between achieving fairly accurate search results and keeping the search process quick enough for immediate interactions.

3.6 Recommending Similar Outfits

The process of recommending similar outfits based on user input images leverages the power of the Annoy index and image embeddings to provide personalised fashion recommendations.

3.6.1. Utilising Image Embeddings and Annoy Index

1. **Generating Input Embedding:** When a user inputs an image, its embedding is generated using the same transformation and ResNet model applied during training. This approach ensures consistency and enables effective comparison.

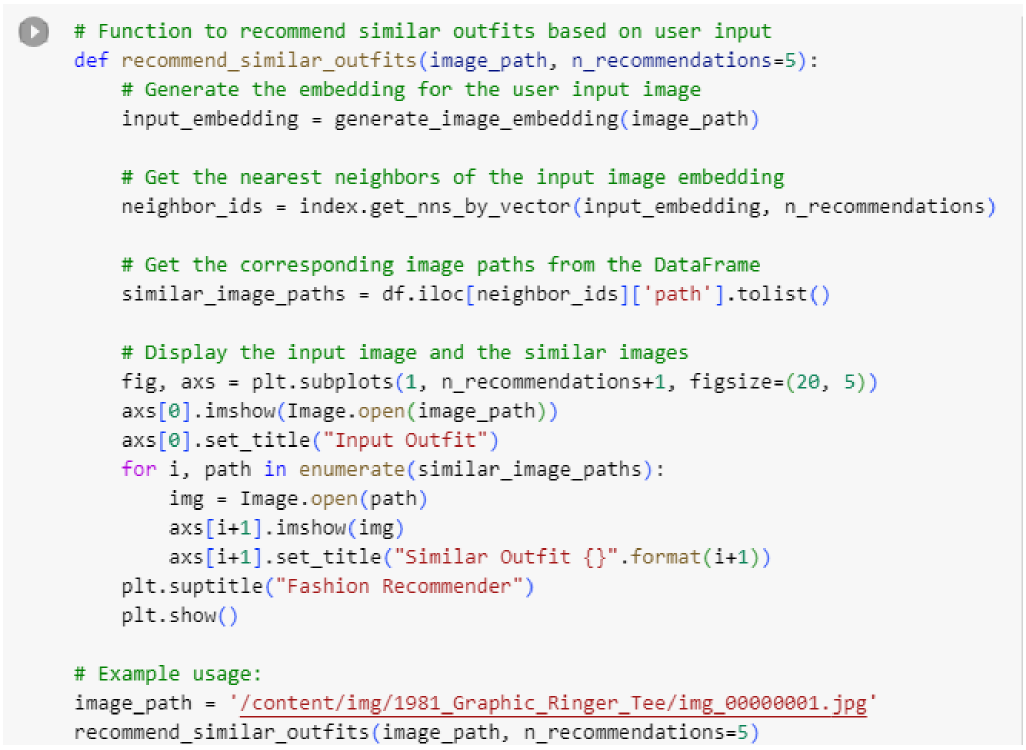
2. **Finding Nearest Neighbors:** The Annoy index has a crucial function in identifying the nearest neighbors of the input embedding. Through ANN searches, the index retrieves embeddings that closely resemble the input.
3. **Retrieving Paths:** The retrieved nearest neighbors' indices are employed to fetch the corresponding image paths from the dataset, which represent outfits that are visually similar.

3.6.2. Displaying Recommendations

The retrieved paths facilitate the display of similar outfit recommendations to users:

1. **Input and Similar Outfits:** The user is presented with both the input image and a set of recommended similar outfits. These outfit suggestions are visually coherent and align with the user's preferences.
2. **User Experience:** The recommendations offer users a diverse array of fashion choices aligned with their input image. This enhances the user experience by providing inspiration and suggestions that resonate with their personal style.
3. **Visual Exploration:** By presenting a comparison between the input outfit and recommended alternatives, users can make informed decisions and effortlessly explore various fashion possibilities effortlessly.

The **recommend_similar_outfits** function demonstrates this process. The input image's embedding is generated, and the Annoy index is queried to retrieve similar outfit indices. The corresponding paths are then fetched, and the input outfit and similar outfits are displayed using **matplotlib.**



```python
# Function to recommend similar outfits based on user input
def recommend_similar_outfits(image_path, n_recommendations=5):
    # Generate the embedding for the user input image
    input_embedding = generate_image_embedding(image_path)

    # Get the nearest neighbors of the input image embedding
    neighbor_ids = index.get_nns_by_vector(input_embedding, n_recommendations)

    # Get the corresponding image paths from the DataFrame
    similar_image_paths = df.iloc[neighbor_ids]['path'].tolist()

    # Display the input image and the similar images
    fig, axs = plt.subplots(1, n_recommendations+1, figsize=(20, 5))
    axs[0].imshow(Image.open(image_path))
    axs[0].set_title("Input Outfit")
    for i, path in enumerate(similar_image_paths):
        img = Image.open(path)
        axs[i+1].imshow(img)
        axs[i+1].set_title("Similar Outfit {}".format(i+1))
    plt.suptitle("Fashion Recommender")
    plt.show()

# Example usage:
image_path = '/content/img/1981_Graphic_Ringer_Tee/img_00000001.jpg'
recommend_similar_outfits(image_path, n_recommendations=5)
```

Listing 23: Fashion Outfit Recommender Function Code Extract

In essence, the recommendation process transforms user input into practical fashion recommendations, enriching their fashion experience and enabling exploration based on visually coherent outfit choices. The integration of image embeddings and the Annoy index empowers the recommendation system to offer quick and personalised fashion insights.

3.6.3. Testing Outfit Recommendations

The recommendation process can be thoroughly tested to ensure its accuracy and responsiveness:

1. **Input Validation:** Tests the function with a variety of input images to verify that the embeddings are accurately generated, and the Annoy index retrieves relevant neighbors.

2. **Diverse User Preferences:** Evaluates the recommendations using input images representing distinct fashion styles. Confirm that the displayed similar outfits are coherent with each style.

3. **Number of Recommendations:** Tests the function with different values for **n_recommendations** to ensure the desired number of similar outfits is displayed.

4. **Performance:** Measure the execution time of the recommendation process to ensure it meets real-time expectations.

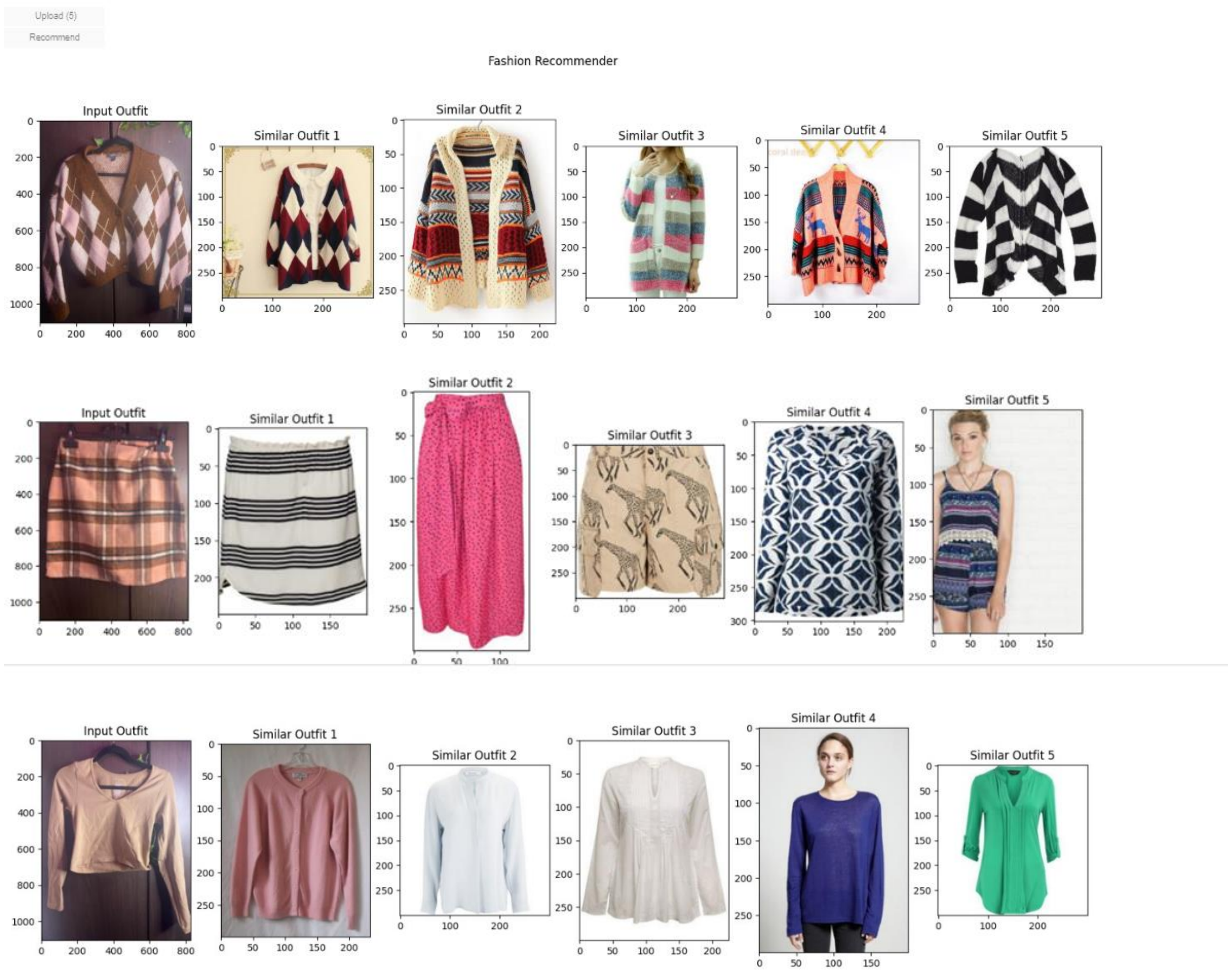


Figure 9: Fashion Recommender System

Testing this recommendation process guarantees its reliability and confirms that the integration of image embeddings and the Annoy index effectively enhances the FRS's ability to provide relevant and engaging outfit suggestions to users.

4. Project Management

The University of Essex's CSEE department equipped Individual Capstone project students with the two widely recognised project management tools – Jira and GitLab.

4.1 Gitlab

Throughout the development of the FRS, I utilised the capabilities of GitLab to establish version control and streamline issue tracking. GitLab emerged as an essential tool, enabling me to manage my project's source code effectively, track development iterations, and address challenges encountered during the implementation process.

4.1.1. Version Control and Code Management

By employing GitLab as a robust version control system, I efficiently maintained various iterations of my codebase. By harnessing Git's branching and merging capabilities, I could work on different features and enhancements separately, ensuring that modifications did not disrupt the stability of the existing codebase.

For example, I uploaded many variations of my iPython FRS system.

4.1.2. Issue Tracking and Problem Solving

GitLab's issue tracking system plays a pivotal role in managing challenges and roadblocks encountered during a development journey. By logging specific issues developers encounter, they can systematically document and address them, ensuring a structured approach to problem-solving [23].

For instance, if an anomaly in the accuracy was encountered of the FRS predictions, I would be able to initiate an issue in GitLab. This would document the problem, provide relevant code snippets, and would allow me to delve into debugging independently. The issue would serve as a reference point for troubleshooting and enable me to iteratively refine the model until the accuracy improved.

4.1.3. Continuous Improvement and Documentation

GitLab facilitated continuous improvement by maintaining a history of changes made to the codebase. Through commit messages, I could comprehensively document modifications and enhancements, ensuring transparency and accountability in the development process [24].

For example, for every significant update, I documented commit messages that outlined the changes made, the rationale behind them, and their impact on the system. This documentation not only aided my own understanding but also provided a reference for future modifications.

4.2 Jira

Within project management, Jira is referred to as a powerful and flexible tool that significantly contributed to overseeing different aspects of developing the Fashion Recommendation System (FRS). The strong capabilities of Jira aided in efficiently managing tasks, tracking issues, and visualising workflows, ultimately boosting the project's effectiveness and organisation [25].

4.2.1. Kanban Board

A significant feature of Jira was the Kanban board, which greatly contributed to organising my project. The Kanban board visually represented the project's workflow, allowing me to comprehensively track tasks from start to finish. By using columns to represent different stages like "To Do," "In Progress," "Review," and "Done," the Kanban board let me monitor task statuses in real time.

The flexibility of the Kanban board enabled me to customise columns, create sections for specific task types, and set up automatic rules for task progress. For example, while enhancing the model's accuracy, I could move data preprocessing tasks from "To Do" to "In Progress" as I worked on them. Once completed, I shifted tasks to the "Review" column for quality assessment before marking them as "Done."

By using Jira's Kanban board, I streamlined task management, enhanced transparency, and optimised project workflows. The board's visual nature and adaptability were crucial in keeping the FRS development organised, efficient, and responsive to changing needs.

4.2.2. Jira Report Analysis

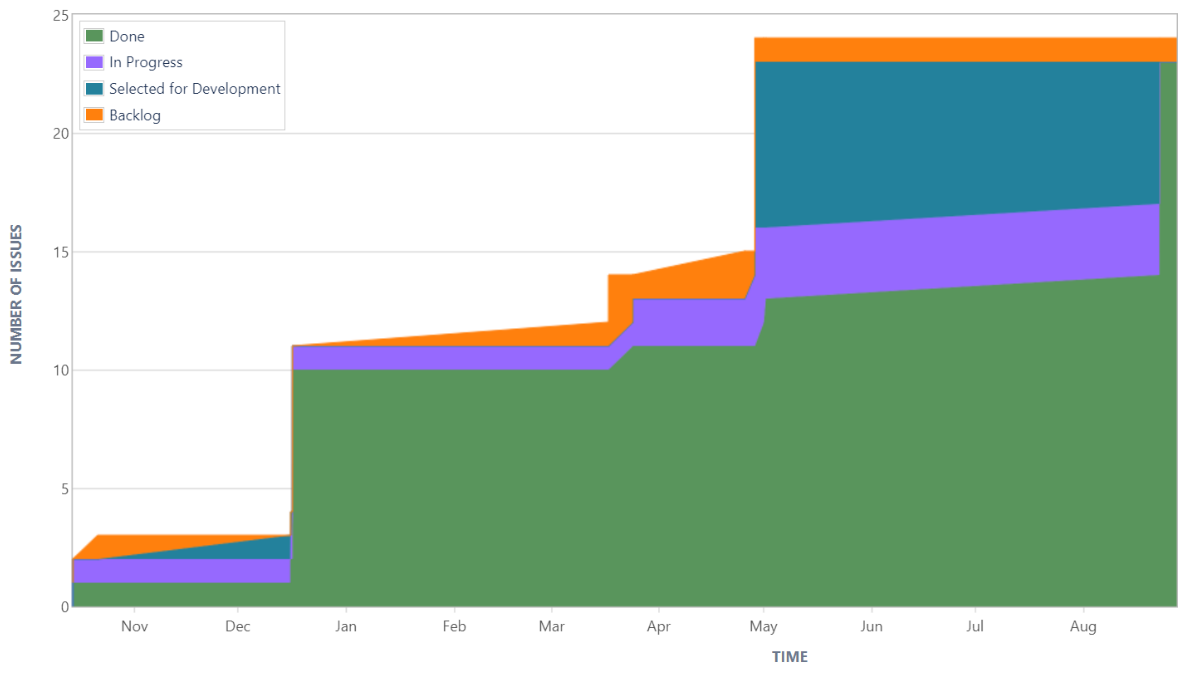


Figure 10: Jira Cumulative Flow Diagram (CFD)

Within Jira, the CFD proved to be a valuable tool for tracking the progress of the FRS development. This diagram visually displayed how tasks moved through different stages over time, helping me keep an eye on their flow. By using the CFD, I could detect any slowdowns, ensure tasks were evenly distributed, and make well-informed decisions to maintain a smooth workflow. This feature in Jira assisted me in balancing workloads, managing resources, and ensuring consistent progress in the FRS project.

My CFD demonstrates that in middle of December there was a massive jump in the number of tickets that were completed, and between May and August many tickets were Selected for Development. And between mid-December and May there was a jump in number of issues in the Backlog.

The analysis of my CFD provides meaningful insights into my project management approach. A notable surge in completed tickets during mid-December indicates a focused effort, leading to a significant milestone. Additionally, the increased number of issues "Selected for Development" between May and August highlights effective task allocation during the peak development phase. Meanwhile, the observed rise in the backlog between mid-December and May suggests I had a proactive strategy in addressing identified issues. These trends in the CFD reflect my systematic approach to resource allocation, task prioritisation, and continuous monitoring, contributing to a coordinated and adaptable project management strategy.

4.3 Risks and Solutions

1. **Data Availability:** A major risk was the unavailability of the "DeepFashion" dataset for direct import through TensorFlow. To address this, I proactively employed the '**gdown**' library to manually download the dataset, ensuring access to the required data for training.

2. **Limited GPU Resources:** I recognised the importance of a powerful GPU for efficient model training, I acknowledged the risk associated with my existing GPU's lower capacity. As a solution, I upgraded to a premium GPU, which seamlessly expedites the training of the model.

3. **Resource Constraints:** The project's complexity and computational demands could potentially lead to technical issues such as the iPython file crashing during extensive computations or large file downloads. To mitigate this risk, I strategically structured the code to manage memory efficiently using batch processing.

4. **Time-Consuming Training:** Training DL models, especially using methods like **'fit_one_cycle,**' can result in lengthy training times. To address this, I employed techniques like model checkpointing, allowing me to resume training from the last saved point, minimising the impact of unexpected interruptions.

4.4 Project Management Reflection

While reflecting on the project management aspect of the FRS development, I acknowledge that my intense focus on code implementation occasionally resulted in unintentionally neglecting the utilisation of the comprehensive project management tools available to me. Notably, there were instances where I didn't fully leverage the functionalities offered by GitLab and Jira, particularly in terms of effectively logging my planning and management activities. A notable example is when I didn't use separate branches in GitLab for different features, which could have streamlined code development and maintenance. Similarly, I realised that I did not utilise GitLab's issue tracking system, I overlooked opportunities to systematically document and address challenges that I encountered during the development process.

Additionally, I acknowledge that I didn't fully explore the advanced features of GitLab beyond basic version control. Specifically, I could have leveraged its capabilities for code reviews and documentation, which could have facilitated better communication and knowledge sharing with my supervisor within the development process. Furthermore, my commit messages often lacked specificity, which limited my ability to effectively track code changes over time.

Despite these areas for improvement, there were positive aspects to highlight. My commitment to version control remained steadfast, ensuring that my codebase was systematically organised and maintained. The use of GitLab's commit history allowed me to revisit previous iterations, which was instrumental in troubleshooting and decision-making. Moreover, my consistent use of the CFD in Jira demonstrated a clear visualisation of my project’s progress, enabling me to manage and allocate resources effectively.

In the future, I recognise the importance of creating a balance between code implementation and utilising project management tools to their full potential. By leveraging features such as branching, issue tracking, and comprehensive commit messages, I can enhance organisation and transparency in the development process. This experience stresses the value of a comprehensive project management approach, ensuring that both coding excellence and effective management converge to produce successful outcomes.

5. Conclusion

5.1 Reflection

In reflection of this project's objectives and accomplishments, it is evident that significant progress has been accomplished in achieving the intended aims. The project's goal of developing an effective Image Classification Model has been successfully met using the ResNet-50 architecture. Through transfer learning, the model was initialised with pre-trained weights to effectively categorise fashion-related images into various clothing categories.

The evaluation of the model's performance, a core objective, demonstrated promising results. Metrics such as top-1 accuracy and top-5 accuracy were employed to thoroughly assess the model's efficiency and reliability in predicting clothing categories. This evaluation assessed the model's ability to accurately identify categories and its capability to suggest alternative predictions.

Investigating image embeddings further contributed to the project's success. The concept of converting images into low-dimensional embeddings demonstrated its efficacy in capturing the semantic essence of outfits. These embeddings facilitated efficient similarity searches, resulted in the creation of an FRS. In addition to this, the utilisation of Annoy Index, a critical objective of this project, was accomplished. This success offered personalised outfit suggestions based on shared visual characteristics.

The project's endeavour to visualise and interpret results was met with valuable insights. Visualisations like top loss images, confusion matrices, and outfit recommendations proved invaluable in understanding the model's behaviour and enhancing transparency in its functioning.

As the project management aspect was also a key objective, a critical self-assessment was conducted within this report. While progress was achieved in the management of code through GitLab, I acknowledged areas for improvement. In future projects, a stronger integration of these tools for planning and monitoring has the potential to further streamline outcome.

5.2 Future Improvements

In the future, the FRS’s capabilities could be significantly improved through targeted refinements.

One notable direction is to refine the FRS's capability to discern intricate clothing textures and patterns. By incorporating advanced algorithms for texture and pattern recognition, the system could offer more precise and personalised recommendations, catering to users' preferences for specific fabric textures and detailed designs.

In addition to this, the augmentation of the FRS to include the categorisation of accessories could be considered. By extending its classification capabilities to encompass accessories such as bags, shoes, and jewellery, the system can offer comprehensive outfit recommendations that consider every fashion detail.

Furthermore, considering the technical nature of this project, developing an intuitive Graphical User Interface (GUI) could significantly enhance the user experience. This GUI would provide a user-friendly platform for individuals to upload outfit images, receive tailored suggestions, and explore a variety of ensemble options, highlighting the system's prowess in understanding visual details.

It is crucial to promote sustainability. The FRS could extend its recommendations to include items from second-hand and sustainable fashion websites, aligning with the growing demand for eco-friendly choices. This strategic expansion would enhance user options but while simultaneously promoting responsible consumer behaviour within the fashion industry.

A crucial step in the iterative development cycle of the FRS is collecting and incorporating user feedback. By actively engaging users and incorporating their insights, the FRS can continuously refine its algorithms, improving its accuracy in recognising textures, patterns, and sustainable fashion elements.

In summary, the FRS has a promising future, with avenues for enhancing its technical capabilities and aligning with sustainability goals. Once these refinements are implemented, the FRS can solidify its position as an intelligent solution within fashion technology.

5.3 Summary

This project was successfully completed, demonstrating the effective combination of academic learning and technical expertise. Drawing from the knowledge gained during my university education, I undertook a transformative journey that resulted in the development of a robust FRS.

The creation of image classification using the ResNet-50 architecture showcased the practical application of DL techniques. The model's accuracy and its ability to offer alternative predictions highlighted its potential to enhance user interactions within the fashion domain.

Exploring image embeddings and subsequently developing an FRS revealed the power of translating theoretical understanding into real-world solutions. Through the utilisation of the Annoy Index, the system perfectly aligns with the modern preference for personalised recommendations, reinforcing its practicality and significance.

My main take away from this experience is the value of translating theoretical knowledge into tangible outcomes. The journey from conceptualising the project to its successful execution stresses the importance of bridging academic concepts with hands-on implementation. It reinforces the idea of leveraging the skills and insights gained from university education to create practical solutions that resonate with current technological trends.

To conclude, the achievements of this project stress the seamless integration of academic learning and technical proficiency. As well as showcasing the potential of CV and DL, the FRS also highlights the advantages of applying university-acquired knowledge to bring innovative ideas to life.

6. References

[1] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[2] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

[3] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[4] Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., ... & Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. Medical image analysis, 42, 60-88.

[5] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... & Zhang, X. (2016). End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.

[6] Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? In Advances in neural information processing systems.

[7] Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering, 22(10), 1345-1359.

[8] Oquab, M., Bottou, L., Laptev, I., & Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

[9] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., & Darrell, T. (2014). DeCAF: A deep convolutional activation feature for generic visual recognition. In International conference on machine learning.

[10] Huh, M., Agrawal, P., & Efros, A. A. (2016). What makes ImageNet good for transfer learning? arXiv preprint arXiv:1608.08614.

[11] Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., ... & Lungren, M. P. (2017). CheXNet: Radiologist-level pneumonia detection on chest X-rays with deep learning. arXiv preprint arXiv:1711.05225.

[12] Kiros, R., Salakhutdinov, R., & Zemel, R. S. (2014). Unifying visual-semantic embeddings with multimodal neural language models. arXiv preprint arXiv:1411.2539.

[13] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In Advances in neural information processing systems.

[14] Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. Journal of Big Data, 6(1), 1-48.

[15] Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., & Le, Q. V. (2019). AutoAugment: Learning augmentation policies from data. arXiv preprint arXiv:1805.09501.

[16] Perez, L., & Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. arXiv preprint arXiv:1712.04621.

[17] Touvron, H., Vedaldi, A., Douze, M., & Jegou, H. (2019). Fixing the train-test resolution discrepancy. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

[18] Powers, D. M. (2011). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. Journal of Machine Learning Technologies, 2(1), 37-63.

[19] Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. Information Processing & Management, 45(4), 427-437.

[20] Davis, J., & Goadrich, M. (2006). The relationship between Precision-Recall and ROC curves. In Proceedings of the 23rd international conference on Machine learning (ICML) (pp. 233-240).

[21] Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., ... & Ng, A. Y. (2017). Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. arXiv preprint arXiv:1711.05225.

[22] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... & Zhang, X. (2016). End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.

[23] GitLab. "Issue Tracking." [Online]. Available: https://docs.gitlab.com/ee/user/project/issues/. Accessed on: Accessed on: April 28, 2023.

[24] GitLab. "Git Basics - Recording Changes to the Repository." [Online]. Available: https://docs.gitlab.com/ee/git/. Accessed on: April 28, 2023.

[25] Atlassian. "Jira Software: Agile project management and tracking." [Online]. Available: https://www.atlassian.com/software/jira. Accessed on: April 28, 2023.

7. List of Figures

8. Listings