BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION [Secție]

DIPLOMA THESIS

Image classification of tourist attractions

Supervisor
Lect. dr. Borza Diana-Laura

Author
Dorobăț Maria-Diana

2025

ABSTRACT

The inspiration for this thesis came from the unique Space Invaders game in Paris, where participants engage in a treasure hunt-like activity by searching for and documenting mosaic tiles featuring characters from the iconic arcade game scattered across the city. This activity combines urban exploration with the thrill of discovery, encouraging participants to explore the city interactively. It sparked the idea of a mobile application designed to enhance the tourism experience by offering users an engaging and interactive way to discover and share tourist attractions. The concept merges elements of gaming and social media, allowing users to track and validate their visits to landmarks in a playful manner, similar to a scavenger hunt game.

To bring this idea to life, deep learning techniques, specifically CNNs, were employed for the real-time recognition and classification of tourist attractions. The thesis explores the use of lightweight CNN architectures, such as MobileNetV2[SHZ+18] and EfficientNet[TL19], which strike a balance between computational efficiency and performance, particularly on mobile devices. In addition, ResNet50[HZRS15] was considered as an alternative to ensure maximum accuracy when necessary. These models were trained using transfer learning techniques, enabling them to perform well on mobile devices while minimizing computational load. Alongside this primary approach, the study also investigated the use of a Siamese network architecture, leveraging one-shot learning techniques as an alternative method for landmark recognition.

The final application, developed for the Android platform, allows users to capture images of tourist attractions and recognize them in real-time using the API of the deployed best-performing CNN model. The app includes a two-way validation system, combining GPS verification with image analysis to authenticate the landmarks being visited. This interactive system ensures the accuracy and reliability of the visited landmarks, providing users with a gamified, educational, and engaging experience. Additionally, users can share their visits on social media, adding a social element to the experience.

For training the CNN models, the dataset used is Pictures of Famous Places[Rya22]. The models' performance was evaluated based on their ability to accurately recognize landmarks while maintaining efficient operation on mobile devices.

The final results of the model after training demonstrate its strong performance in recognizing tourist attractions. The model achieved a training accuracy of 96.46%, which indicates that it effectively learned the patterns in the data. The training loss was 0.3120, reflecting the model's efficient learning process. Achieving 84.92% accuracy on the validation set, the model demonstrated strong capabilities of generalization on unincountered data. The corresponding validation loss was 0.8377, which

suggests that while the model performs well, there may be some room for improvement.

The results demonstrate the model's capacity to balance high accuracy with computational efficiency, confirming its applicability for mobile applications. Although the gap between the training and validation accuracy suggests there may be slight overfitting, the overall performance shows the model's capabilities of recognizing tourist attractions effectively. Further optimization and fine-tuning could potentially reduce this gap and improve the generalizability of the model.

In conclusion, the model achieves a good harmony between accuracy and efficiency, establishing it as a practical solution for the intended mobile application. The reports show that the model is ready for deployment in real-world settings, with potential for further improvement to enhance its generalization and robustness.

Contents

1 Introduction **1**

1.1 Objectives 2

1.2 Thesis Structure 3

2 Existing methods for landmark recognition **4**

2.1 Classical Approaches to Landmark Recognition 4

2.2 Learning-Based Approaches 6

2.3 Commercial and Hybrid Solutions 7

2.4 Summary and Motivation for My Approach 7

3 Theoretical foundations **9**

3.1 Introduction to Supervised Learning 9

3.2 Artificial Neural Networks 10

3.3 Convolutional Neural Networks (CNNs) 11

3.4 Key Layers in CNN Architectures 12

3.5 Training CNNs 14

3.6 Popular CNN Architectures 19

3.7 One-Shot Learning 23

3.8 Siamese Networks 23

4 Landmark detection and recognition **25**

4.1 Dataset 25

4.2 Model 27

4.3 Exploration of a Siamese Network Approach 31

4.4 Deployment 33

4.5 Implementation 34

5 Mobile Application **35**

5.1 Development Framework 35

5.2 Application Architecture 36

5.3 Key Functionalities and Use Cases 37

5.4 User Interface Design 40

6 Experimental Results **43**

6.1 Performance evaluation 43

6.2 Comparison with Other Methods 48

7 Conclusions and future work **50**

Bibliography **51**

Chapter 1

Introduction

In recent times, the application of deep learning, particularly CNNs, has greatly improved the area of image classification. CNNs have been particularly effective in recognizing objects, scenes, and landmarks in images, making them ideal for use in tourism and location-based applications. This thesis investigates the use of fine-tuned CNN models for the recognition and classification of tourist attractions, with the ultimate goal of improving the tourist experience through technology.

The primary dataset used in this study is the Pictures of Famous Places dataset [Rya22], which includes images of the most iconic and globally recognized landmarks. These images were selected from the List Challenges website [Cha25], a popular platform where users curate and engage with lists of notable locations. The diverse nature of this dataset, covering various geographic regions and landmark types, presents both a challenging and exciting opportunity for applying deep learning techniques in a real-world context.

A key objective of this study is to explore the performance of lightweight CNN architectures, such as MobileNetV2 [SHZ+18] and EfficientNet [TL19], which are designed to achieve an equilibrium between high accuracy and computational efficiency. These models are especially suitable for mobile and embedded systems, where performance and limited resources are crucial factors. While these architectures serve as the primary candidates, the study also evaluates ResNet50 [HZRS15], a more robust and computationally intensive architecture, to assess whether it can provide higher recognition accuracy in specific scenarios.

The scope of this thesis is not only to assess the effectiveness of different CNN models but also to integrate the most effective model into a practical mobile application. The proposed application is a tourist attraction tracker embedded within an Android-based platform, designed to help users track and share their visits to landmarks. In order to validate the authenticity of these visits, the application employs a dual-validation system that combines GPS data with real-time image recognition. The CNN model plays a crucial role in this system by analyzing captured images of

landmarks and cross-referencing them with a database of known tourist attractions.

One of the main additions of this study is demonstrating how lightweight CNN architectures can be applied to a real-time, mobile-based application while sustaining strong accuracy and efficiency. This study also explores the challenges involved in developing and deploying such systems, including the limitations of mobile devices and the need for computationally efficient models. The results of this work could have a great impact on the way travelers interact with landmarks and the tourism industry as a whole, offering potential for future innovations in smart tourism applications.

1.1 Objectives

To further these aims, this study focuses on creating and assessing a deep learning-focused application for the recognition and classification of tourist attractions using CNNs. The goal is to leverage cutting-edge techniques in computer vision to provide a more interactive and efficient way for tourists to engage with landmarks across the globe.

Given the growing reliance on mobile applications for tourism, the application designed in this study aims to empower users to easily track and share their visits to landmarks, while ensuring the accuracy and authenticity of these visits. The application will integrate a dual-validation system, which combines GPS verification with real-time image analysis to authenticate the user's location and the landmark being visited.

The first specific goal of this thesis is to discover and evaluate different CNN architectures for landmark recognition. This study will focus on comparing lightweight CNN architectures, such as MobileNetV2 [SHZ+18] and EfficientNet [TL19], as well as the more robust ResNet50 [HZRS15]. Additionally, an alternative approach using a Siamese network architecture with one-shot learning techniques was explored to assess its feasibility for this task. The goal is to assess the accuracy and computational efficiency of these models to ensure a practical, real-time experience for users.

The second objective is to develop a mobile application that integrates CNN-based landmark recognition with a dual-validation system. The objective is to create a functional Android-based mobile application that leverages the trained CNN model to identify and track visited landmarks. The dual-validation system will combine GPS data and image analysis to verify the authenticity of the user's visit, allowing users to "check-in" to tourist destinations in a gamified manner.

Finally, the study seeks to assess the practical challenges and solutions in deploying deep learning models on mobile devices. This includes addressing issues such as computational efficiency, memory usage, and battery life, which are critical for

real-time applications. The research will explore strategies to overcome these challenges and ensure that the application can run smoothly on resource-constrained mobile devices.

1.2 Thesis Structure

The thesis is structured into several chapters, each presenting a key piece of the study. Chapter 1, Introduction, provides the background and motivation for the study, detailing the use of CNNs for landmark recognition and the development of a mobile application to track tourist visits. It establishes the context and relevance of the research in the growing field of smart tourism.

Following the introduction, Chapter 2, Existing Methods for Landmark Recognition, reviews the current techniques and approaches in the field of landmark recognition. This chapter provides the necessary foundation for understanding the advancements and challenges in the domain.

Chapter 3, Theoretical Foundations, dives into the deep learning concepts, particularly CNNs, that are central to this thesis. It outlines the underlying theory of image recognition and its application to the classification of landmarks, laying the groundwork for the implementation in later chapters.

Chapter 4, Landmark Detection and Recognition, the focus shifts to the practical aspects of the research. This chapter details the entire landmark detection and recognition pipeline, from the initial dataset selection and preparation through the critical choices regarding CNN architectures and the details of the model fine-tuning process. Furthermore, it discusses the deployment of the best-performing model, outlining how it's integrated for effective use.

Chapter 5 explains the development of the mobile application, focusing on its architecture, selected development tools, and its interaction with the CNN model for landmark recognition via API calls to the deployed service.

Chapter 6, Experimental Results, presents the reports from the assessment of the CNN models and the mobile application. It compares the performance of the proposed methods with existing techniques and provides an analysis of the outcomes.

Finally, Chapter 7, Conclusions and Future Work, wraps up the thesis by summarizing the key discoveries and suggesting future work for bettering the model's accuracy and reliability.

Chapter 2

Existing methods for landmark recognition

This chapter provides an overview of existing approaches to landmark recognition, tracing the evolution of the field from traditional computer vision techniques based on handcrafted features to modern deep learning-based models. Early methods, while effective under controlled conditions, struggled with challenges such as lighting changes, occlusions, and variations in viewpoint. These limitations have been addressed by deep learning approaches, particularly CNNs, which have revolutionized the domain by learning multilayered characteristics directly from raw images.

I will begin by reviewing classical methods, including the use of feature detectors and descriptors like SIFT[Low04], SURF[BTVG06], and ORB[RRKB11], and discuss how these are combined with matching algorithms and machine learning classifiers. Next, I explore deep learning models like NetVLAD[AGT+16], DELG[CAS20], and PlaNet[WKP16], which have demonstrated superior performance on large-scale datasets and in real-world scenarios. The chapter also includes a discussion of hybrid and commercial solutions, such as the Google Vision API[Goo24], and highlights the strengths and weaknesses of each approach.

By examining the advantages and disadvantages of existing methods, this chapter lays the groundwork for the design and implementation of a mobile-friendly landmark recognition system, which is the primary focus of this thesis.

2.1 Classical Approaches to Landmark Recognition

Before the emergence of deep learning, landmark recognition was primarily tackled using traditional computer vision techniques. These approaches relied heavily on manually designed feature detectors and descriptors, combined with matching algorithms and conventional machine learning classifiers. Although modern deep

learning methods have largely superseded them, classical approaches laid the foundation for understanding visual similarities between images and remain relevant for comparison and benchmarking.

Feature detection is a method of finding key points in an image that are likely to be distinctive and stable under various alterations such as viewpoint and exposure changes. Once detected, descriptors are computed to describe the local neighborhood of each key point in a way that enables comparison across images.

The most commonly used algorithms for feature detection and description include:

1. **SIFT (Scale-Invariant Feature Transform) [Low04],** introduced by David Lowe, detects and describes local image features that are constant to scale and rotation and resilient to changes in viewpoint, noise, and illumination. It enables reliable matching across different views using distinctive keypoints and geometric verification techniques, making it effective even under occlusion and clutter.

2. **SURF (Speeded-Up Robust Features)[BTVG06]** is a quicker option to SIFT that uses integral images and approximated Hessian matrix-based detectors. It provides rotation and scale-invariant features, though its performance under extreme image distortions may be lower than SIFT.

3. **ORB (Oriented FAST and Rotated BRIEF)[RRKB11]** combines the FAST keypoint detector with the BRIEF descriptor, modified for rotation invariance. It is much faster and suitable for real-time applications, although it is less robust than SIFT or SURF in complex scenes.

Once features are extracted, the next step involves matching these descriptors between the input image and a database of associated images. The goal is to identify correspondences between similar keypoints, which can be used to infer the identity of the landmark.

A common technique for matching is the **brute-force matcher**, which compares each descriptor in the query image with all descriptors in the database using a distance metric such as Euclidean distance for SIFT and SURF, or Hamming distance for ORB. To improve efficiency and scalability, especially in large datasets, approximate nearest-neighbor (ANN) methods are often used.

To ensure that the matched features are actually correct and follow a consistent geometric pattern, algorithms such as **RANSAC** (Random Sample Consensus)[connd] are used. RANSAC helps remove incorrect matches by keeping only those that agree with a possible transformation, such as a shift, rotation, or scaling between the im-

ages. This step makes the system more reliable, in particular when the image has random variations or repeating patterns.

After the matching process, the system needs to decide which landmark the image belongs to. This is often done using classification methods such as **k-Nearest Neighbors (k-NN)**[IBM20b] or **Support Vector Machines (SVM)**[IBM20a]. These methods look at how many features match between the input image and all known landmarks and select the one with the most or best matches.

Despite their historical importance, these classical approaches often struggle in unconstrained settings due to their intolerance to image clutter, occlusions, and large variations in viewpoint. This has motivated the development of learning-based methods that are more robust and scalable.

2.2 Learning-Based Approaches

Recent advances in deep learning have significantly improved landmark recognition by enabling models to learn discriminative and invariant features directly from the data. CNNs have become the backbone of modern approaches, offering robustness to variations in scale, lighting, and perspective without requiring manual feature engineering.

In the next section, I explore several state-of-the-art models, including NetVLAD, DELG, and PlaNet, that demonstrate how deep learning has transformed landmark recognition into a highly accurate and scalable task suitable for real-world applications.

One of the most significant advances in landmark recognition through deep learning is the use of **NetVLAD** (CNN with Vector of Locally Aggregated Descriptors) [AGT+16]. NetVLAD addresses large-scale visual place recognition by developing a CNN architecture designed for the task. The main innovation is a new generalized VLAD layer. VLAD layers work by aggregating local image descriptors into a fixed-length global vector, capturing both the presence and distribution of features across the image. This allows the model to retain fine-grained spatial information while producing compact, discriminative image representations. The network is trained using a weakly supervised ranking loss on images from Google Street View that depict the same locations over time. The proposed architecture outperforms traditional hand-crafted image representations and standard CNN descriptors on place recognition benchmarks while also enhancing the performance of compact image representations in typical image retrieval tasks.

Another notable advancement in the field of place recognition is **PlaNet** (Photo Geolocation with CNNs) [WKP16]. PlaNet approaches the problem of photo geolocation by formulating it as a classification task. The method divides the Earth's

surface into multiple geographic cells, and the network is trained using millions of geotagged images. This deep network model outperforms previous image retrieval-based methods and even achieves superhuman accuracy in certain cases. Additionally, PlaNet is extended to photo albums by combining it with a long-short-term memory (LSTM) architecture, allowing it to leverage temporal coherence between images. This integration results in a 50% improvement in performance compared to single-image models, further enhancing the accuracy of photo geolocation.

DELG (DEep Local and Global Features) [CAS20] presents a unified deep learning model that integrates both global and local image features for effective image retrieval. Unlike traditional approaches that treat global and local descriptors separately, DELG combines them within a single network architecture. Generalized mean pooling is employed by the model to derive robust global descriptors. To selectively gather discriminative local features, an attention mechanism is also integrated. Network training occurs end-to-end, relying exclusively on image-level supervision, with a careful balance of learning signals between both types of features. An integrated auto-encoder-based dimensionality reduction method further enhances the efficiency of local feature extraction. DELG sets new expectations for the results on key benchmarks such as Revisited Oxford, Revisited Paris, and Google Landmarks v2, demonstrating its effectiveness in instance-level recognition and retrieval tasks.

2.3 Commercial and Hybrid Solutions

In addition to academic research, several commercial and hybrid systems have been developed to provide landmark recognition capabilities in real-world applications. These services often leverage large proprietary datasets and scalable cloud infrastructure, offering robust performance with minimal setup.

One prominent example is the **Google Cloud Vision API**[Goo24], which provides a high-level interface for landmark detection. It uses a combination of deep learning and large-scale image databases to identify landmarks around the world. The API is capable of recognizing thousands of famous landmarks and provides metadata such as name and geographic coordinates.

2.4 Summary and Motivation for My Approach

While several existing methods for landmark recognition, such as NetVLAD[AGT$^+$16], DELG[CAS20], and commercial APIs like Google Vision[Goo24], demonstrate high accuracy and robustness, they often rely on large-scale datasets, powerful compute

resources, or closed-source infrastructures. In contrast, this work does not aim to outperform these state-of-the-art solutions, but instead focuses on adapting accessible, well-documented techniques to build a lightweight, practical prototype suitable for mobile or resource-constrained environments. The implementation highlights how to achieve a balance between performance, efficiency, and feasibility, offering insights into how existing methods can be combined and applied in other use cases with limited resources.

Chapter 3

Theoretical foundations

In this chapter, I will introduce the notion of supervised learning, focusing on the use of CNNs to address our problem. I will explore the key principles behind training CNNs and discuss how different architectures contribute to improving model performance. Furthermore, I will delve into alternative approaches such as Siamese networks and the one-shot learning paradigm, examining their potential for specific recognition tasks. Additionally, I will highlight the importance of various model designs and their impact on solving the task effectively.

3.1 Introduction to Supervised Learning

Supervised Learning is an essential notion in machine learning where a model is trained on a dataset that contains input data and their valid outputs. Each training example in the dataset is paired with a label, allowing the model to grasp the mapping from inputs to the correct outputs based on these provided labels. The objective is for the model to generate a function that can generalize well enough to form accurate predictions on new data.

Supervised learning is primarily divided into two categories:

1. **Classification:** Classification tasks involve discrete output labels y, where the model's scope is to categorize the input x into one of various predetermined classes. An example is image classification, where the model could be tasked with identifying whether an image contains a cat, dog, or bird.

2. **Regression:** In regression problems, the output labels are continuous, and the model's task is to predict a numerical value. For example, a regression problem involving pizza could be predicting the final baked diameter of a pizza crust. This prediction would be based on features such as the initial weight of the dough ball and the oven temperature used for baking.

In this thesis, I focus specifically on **classification** problems, as the primary task is classifying images of landmarks into various categories.

Supervised learning involves several essential stages, each adding up to the creation of a robust model able to make accurate predictions. The steps involved in the supervised learning process are outlined below:

1. **Data Collection and Preprocessing:** The initial step involves gathering a labeled dataset, which consists of input-output sets where each input corresponds to a correct label. The quality of this data is crucial for model performance. In image classification tasks, for example, it may involve collecting images of landmarks with their corresponding labels. Data preprocessing follows, which includes cleaning the data (removing errors or irrelevant information), handling missing values, normalizing, and augmenting the data (e.g., resizing or rotating images, normalizing pixel values).

2. **Splitting the Dataset:** For effective model development, data is typically divided into three distinct sets: a training set to build the model, a validation set to optimize its parameters, and a test set to assess its final performance. A common distribution for these sets is 70% for training, with the remaining 30% equally divided between validation and testing (15% each).

3. **Model Selection and Training:** At this step, a suitable model is chosen, taking into account the task and data. The next phase is training the model by processing the training set and adapting parameters to minimize prediction errors using optimization methods, for instance, stochastic gradient descent (SGD) or Adam. The training process enables the model to learn the relationship between inputs and outputs.

4. **Hyperparameter Tuning and Model Evaluation:** After initial training, hyperparameters are fine-tuned using the validation set to optimize the model's ability to correctly classify data. Once the model is tuned, its performance is verified against the test set. Common performance metrics for classification include accuracy, precision, recall, and F1 score, which help to determine the precision of the model's generalization abilities to unseen data.

[IBM21]

3.2 Artificial Neural Networks

Having outlined the general supervised learning pipeline, we now turn our attention to a powerful class of models frequently employed for complex tasks like image

recognition: artificial neural networks (ANNs). ANNs are models adapted to mimic the functioning of biological nervous systems such as the human brain. At their core, ANNs consist of multiple interconnected units, often called nodes. These units collaborate in a distributed manner, allowing the model to learn from the input dataset and refine the output.

The fundamental architecture of an ANN typically involves layers of these interconnected nodes, as illustrated in Figure 3.1 . Input data, usually structured as a multi-dimensional vector, is fed into the input layer. This layer then transmits the information to one or more hidden layers.

Within the hidden layers, each neuron processes the signals it receives from the preceding layer. A crucial aspect of this processing is the network's ability to learn. This learning occurs as the network adjusts the connections between neurons and the internal parameters of these neurons. The network essentially evaluates how small changes within its hidden layers affect the final output. By iteratively adjusting these parameters to minimize errors or maximize desired outcomes, the network "learns" the patterns in the data. The stacking of multiple hidden layers on top of each other is a technique commonly referred to as deep learning, enabling the network to acquire progressively more complicated representations of the raw data. Finally, the processed information is passed to the output layer, which produces the result of the model.

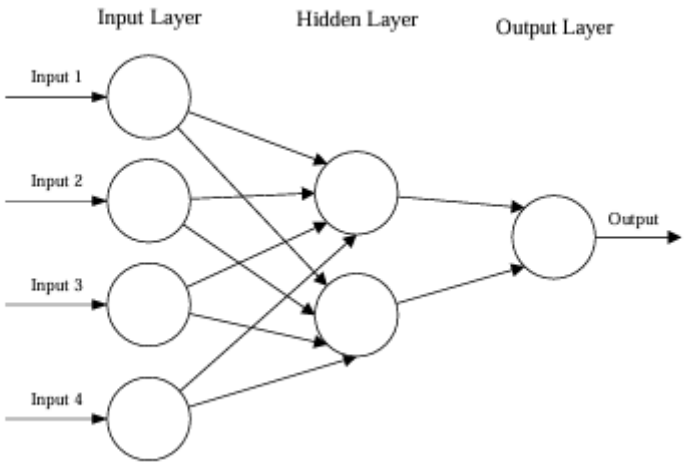

Figure 3.1: A simple three layered feedforward neural network (FNN)[ON15]

3.3 Convolutional Neural Networks (CNNs)

CNNs share fundamental similarities with traditional ANNs. Both are composed of interconnected processing units, or neurons, that learn and improve their performance through self-optimization. Each neuron, as in many ANNs, takes an input,

performs a calculation (often a dot product followed by a non-linear activation), and produces an output. The entire network, from the initial raw image data to the final classification score, can be viewed as implementing a single complex function that assigns a perceptive score (determined by the CNN's weights). Furthermore, the final layer of a CNN typically contains loss functions corresponding to the different classes, and many of the traditional techniques and best practices developed for training ANNs remain consistent.

The main difference between CNNs and ANNs is their primary domain: CNNs are predominantly employed for pattern recognition in images. This focus makes CNN architecture suitable to integrate image-specific characteristics. This specialized approach leads to a substantial reduction in the total number of parameters compared to a standard fully connected ANN attempting to analyze identical image inputs.

3.4 Key Layers in CNN Architectures

A CNN is constructed by stacking three main kinds of layers: convolutional, pooling, and fully connected layers. The specific arrangement and amount of these layers define a particular CNN architecture. A basic CNN structure designed *(see Figure 3.2)* illustrates this concept.

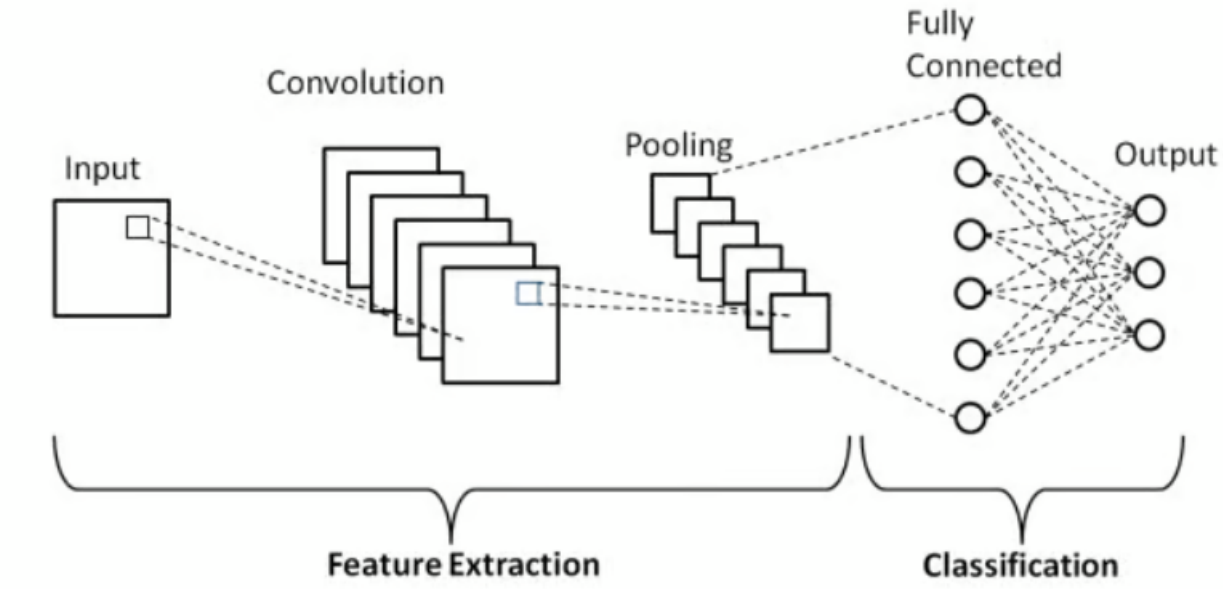

Figure 3.2: A standard CNN design[Don24]

The flow of information within a CNN can be conceptualized as a sequence of four primary stages:

1. **Input Layer:** Analogous to other ANNs, the initial layer—referred to as the input layer—receives the raw pixel values of the input image. In practice, these images are represented as tensors, which are multidimensional arrays that encode the spatial and channel-wise structure of the data. For instance, each MNIST digit image is a 28×28 grayscale image and is thus represented as a 2D tensor (or a $28 \times 28 \times 1$ tensor if the channel dimension is explicitly included). This tensor serves as the input to the convolutional layers that follow.

2. **Convolutional Layer:** The convolutional layer extracts meaningful features by applying learnable kernels (filters) over small, localized regions of the input, computing a scalar product between kernel weights and pixel values. To capture complicated, non-proportional patterns, the convolution outputs then undergo a transformation by a non-linear activation function, with the rectified linear unit (ReLU) being the most commonly employed. In modern CNN architectures, convolutional layers and their activations are organized into blocks, each block including several convolutional layers succeeded by activations before any pooling is performed. This grouping allows the network to develop features in a layered manner, starting from elementary components such as lines and other shapes and building up to more sophisticated and complex concepts. Additionally, batch normalization is commonly inserted either immediately after the convolution or between the convolution and activation, helping to stabilize and accelerate training while improving model convergence and generalization.

3. **Pooling Layer:** Following the convolutional layer and its activation, a pooling layer is often applied to downsample the spatial dimensions (width and height) of the feature maps. A key consequence of this decrease is a diminished parameter load for the layers that follow, which is crucial for preventing overfitting and for making the model more robust to subtle changes in the input's position or appearance. Max pooling and average pooling represent the standard techniques employed for pooling operations. Some modern architectures also explore alternatives to traditional pooling, such as strided convolutions and adaptive pooling, which can achieve downsampling while preserving more spatial information or allowing flexible output sizes.

4. **Fully-Connected Layers:** Commonly positioned at the end of CNN architectures are fully-connected (dense) layers, integrating high-level features extracted by earlier layers to perform classification or regression. A hallmark of a fully connected layer is that each neuron receives input from every output of the prior layer, thereby facilitating the learning of complex feature interactions. While earlier designs often include one or more dense layers with

ReLU activations before the output, more parameter-efficient approaches reduce this reliance to limit overfitting. Instead, global average pooling is often used to condense each spatial feature map into a single value, producing a compact vector representation. To categorize the input, this vector is then channeled into a single fully connected component. The network's terminal layer, typically designed with one neuron for every potential class, computes class probabilities via a Softmax operation.

3.5 Training CNNs

The training of CNNs hinges on a few interconnected key concepts that enable these networks to learn from data. These include understanding the role of activation functions, defining a loss function to quantify errors, employing an optimization algorithm guided by backpropagation to adjust network parameters, and continuously evaluating the network's performance on unseen data to ensure proper learning and prevent overfitting. Data preparation, while critical, will be discussed in a subsequent section.

Following this overview, we'll now delve into each of these key concepts in detail, explaining their individual roles and how they collectively contribute to the successful training of CNN models.

To facilitate the learning of complex data patterns, CNNs rely on activation functions that introduce nonlinearity into the model. Widely adopted examples of such functions are the sigmoid, hyperbolic tangent (tanh), and the Rectified Linear Unit (ReLU).

The **sigmoid function**, defined in Equation 3.1, is an S-shaped function that scales the input to an output between (0,1):

$$\mathrm{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

[JXZ22]

Despite its widespread use, the sigmoid function can cause the *vanishing gradient problem*, especially when input values are at either extreme of the spectrum. In these regions, the function saturates and its gradient becomes nearly zero, which hinders weight updates during backpropagation and slows down training.

The **tanh function**, shown in Equation 3.2, is a scaled and shifted version of the sigmoid:

$$\mathrm{Tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.2}$$

[JXZ22]

The tanh function transforms inputs into the range of $(-1, 1)$ and is centered around zero, which often leads to quicker model convergence compared to the sigmoid. Despite this, it shares the sigmoid's vulnerability to vanishing gradients, especially with extreme input values.

ReLU, defined in Equation 3.3, is frequently the go-to choice in a multitude of CNN designs:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{3.3}$$

[JXZ22]

ReLU is a non-saturating function that retains positive values and zeroes out negative ones, improving training efficiency and reducing the risk of vanishing gradients. It also introduces sparsity in the network, which can lead to better generalization. ReLU isn't without its limitations; neurons can turn off permanently when exposed to a continuous stream of negative inputs. This specific issue is termed the dying ReLU problem.

By allowing a minimal, non-zero gradient when inputs are negative, **Leaky ReLU** provides a solution to the 'dying ReLU' phenomenon. Unlike ReLU, which sets all negative values to zero, Leaky ReLU assigns them a small slope, preventing neurons from becoming inactive. Its function is defined in Equation 3.4.

$$\mathrm{LeakyReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{if } x < 0 \end{cases} \tag{3.4}$$

[JXZ22]

Here α is a small, fixed coefficient (e.g., 0.01) that determines the slope for negative input values.

The loss function plays a crucial role in training CNNs, serving as the metric that measures the disparity between the model's predicted outputs and the desired target values. It guides the optimization process throughout training by quantifying this difference.

A well-chosen loss function ensures that the network learns meaningful features and converges toward a solution that generalizes well to unseen data.

For tasks involving classification, key loss functions typically utilized consist of:

Binary cross-entropy (BCE) is widely utilized in binary classification problems where each input is assigned to one of two classes. The network outputs a probability score, which is then compared with the actual label to compute the loss.

For a given instance i, where $y_i \in \{0, 1\}$ is the ground truth label and $\hat{y}_i$ is the estimated probability, the BCE loss function is defined as:

$$L_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i + \epsilon) + (1 - y_i)\log(1 - \hat{y}_i + \epsilon)\right] \tag{6}$$

Here, N signifies the total count of individual data points being evaluated in the current batch or dataset. The variable y_i denotes the ground truth category for the i-th sample. Conversely, $\hat{y}_i$ represents the model's confidence that the i-th sample belongs to class 1. The term ϵ is a small constant (commonly set to 10^{-7}) included to prevent numerical instability during logarithmic computation. [Yat22]

Categorical cross-entropy combines the softmax function with cross-entropy loss. It is favored for its quick learning speed and robust performance. This loss function is commonly used in multiclass classification tasks, where each input is assigned to one of several mutually exclusive classes. The formula for the Softmax loss function is

$$L_{\text{Softmax}} = \frac{1}{N}\sum_{i} -\log\left(\frac{e^{z_{y_i}}}{\sum_j e^{z_j}}\right) \tag{1}$$

where N is the number of samples, z_{y_i} represents the output from the last fully connected layer for the correct class y_i, and z_j is the output for the j-th class from the final fully connected layer. [ZZ22]

Optimization is essential in training CNNs, aiming to discover the best parameters that maximize model performance. The process often involves backpropagation, which figures out the impact of each weight and bias on the overall loss and then adjusts the parameters accordingly.

Gradient descent is the most common optimization method used in neural networks. It iteratively adjusts the model's weights to reduce the loss function. It is simple to implement and works well for many problems. However, it requires numerous iterations to converge and can be less efficient with very large or complex datasets since it computes gradients using the entire dataset. The fundamental weight update rule in basic gradient descent is given by:

$$W \leftarrow W - \eta \nabla Q(\omega) \tag{3.5}$$

where W represents the updated weight, ω is the current weight, η is the learning rate that controls the size of each adjustment, and $\nabla Q(\omega)$ serves as the gradient of the objective function Q relative to the weights, indicating where the function rises most rapidly, it's substracted (multiplied by the learning rate) to move towards a minimum. [MAW23]

An alternative is **Stochastic Gradient Descent** , which computes the gradient using a small batch of samples, reducing computational cost. In the context of SGD,

the gradient is typically computed on an individual sample or a mini-batch of the dataset (Q_i) rather than the entire dataset. The weight update in SGD is:

$$W = \omega - \eta \nabla Q_i(\omega) \tag{3.6}$$

[MAW23]

Here, $\nabla Q_i(\omega)$ is the gradient of the error relative to a single data instance or a mini-batch. SGD iteratively adjusts the weights based on these noisy but computationally cheaper gradient estimates.

SGD is particularly useful for large datasets, as it offers a more efficient way to optimize without requiring the full dataset at each step.

Root Mean Square Propagation (RMSProp) modifies the learning rate on a per-parameter basis. This adaptation is driven by the moving average of the squared gradients. This helps to address the diminishing learning rates of AdaGrad and performs well in non-convex optimization problems. The RMSProp update can be expressed as:

$$v_t = \delta \cdot v_{t-1} + (1 - \delta) \cdot g_t^2 \tag{3.7}$$

$$\Delta\omega_t = -\frac{\eta}{\sqrt{v_t + \epsilon}} \cdot g_t \tag{3.8}$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t \tag{3.9}$$

where η is the learning rate, g_t is the gradient at time t, v_t is the moving average of the squared gradients, δ is a decay rate, and ϵ is present to prevent numerical inconsistencies. [MAW23]

The **Adam optimizer**[KAT21] is an optimization algorithm that adaptively sets learning rates based on gradient information. It adaptively sets step size and learning rate (using RMSprop), mimicking SGD dynamics to guide optimization. Although slower and more resource-intensive than SGD due to its stochastic nature, its practical benefits often outweigh the computational cost.

Adam, an efficient gradient-based momentum method, estimates adaptive learning rates for all parameters. Its update rules are:

$$\begin{aligned} y_t &= \delta_2 \cdot y_{t-1} - (1 - \delta_2) \cdot g_t^2 \\ x_t &= \delta_1 \cdot x_{t-1} - (1 - \delta_1) \cdot g_t \\ \Delta\omega_t &= -\eta \frac{x_t}{\sqrt{y_t + \epsilon}} \cdot g_t \\ \omega_{t+1} &= \omega_t + \Delta\omega_t \end{aligned} \tag{3.10}$$

where η is the learning rate, g_t is the gradient at time t, x_t is the exponential average of gradients, y_t is the exponential average of squared gradients, and δ_1 and δ_2 are hyperparameters. [MAW23]

After training a CNN, a critical step is to evaluate its generalization capability on new data. This assessment determines the CNNs ability to generalize beyond the training set. For classification tasks, several metrics provide valuable insights into different aspects of the model's effectiveness.

A fundamental tool for understanding a classifier's performance is the **confusion matrix**. This table visualizes the predictions against the actual ground truth, categorizing outcomes into four types: **True Positives (TP)** (correctly identified positive instances), **False Positives (FP)** (negative instances incorrectly identified as positive), **False Negatives (FN)** (positive instances incorrectly identified as negative), and **True Negatives (TN)** (negative instances correctly identified as negative).

	Actual Positive	**Actual Negative**
Predicted Positive	TP	FP
Predicted Negative	FN	TN

From the confusion matrix, several key evaluation metrics can be derived:

Accuracy represents the proportion of correct classifications made by the model, considering all instances evaluated [Deva]:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

While intuitive, accuracy can be deceptive when it comes to disproportionate datasets, where there are classes with different numbers of samples. In this case, a model that predicts the class with the largest number of samples can have a high accuracy without the ability to accurately predict the classes with smaller numbers of samples.

Recall focuses on the model's capability to capture all the actual positive cases [Deva]:

$$\text{Recall (TPR)} = \frac{TP}{TP + FN}$$

A high recall is important in programs where not succeeding to capture positive cases (false negatives) has significant consequences.

The **False Positive Rate (FPR)** complements recall by indicating the percentage of negative instances that were misclassified as positive [Deva]:

$$\text{FPR} = \frac{FP}{FP + TN}$$

A low FPR is desirable to minimize false alarms.

Precision quantifies the accuracy of the model's positive predictions. It indicates, among all instances classified as positive, what proportion were truly positive. [Deva]:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Often, an inverse relationship exists between precision and recall. Modifying the classification threshold can increase one metric while decreasing the other. Therefore, it is often useful to consider metrics that balance both, such as the F_1**-score**, which is the harmonic average of precision and recall:

$$F_1\text{-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F_1-score provides a value that summarizes the model's performance in terms of both false positives and false negatives.

The **Receiver Operating Characteristic (ROC) curve** and the **Area Under the ROC Curve (AUC)** are powerful tools for evaluating binary classifiers. The ROC curve plots the TP Rate (Recall) against the FP Rate at different classification cutoff points. The **AUC** gives a single score for how well the model distinguishes between classes at all possible cutoff points. A higher AUC generally signifies a better model. [Devb]

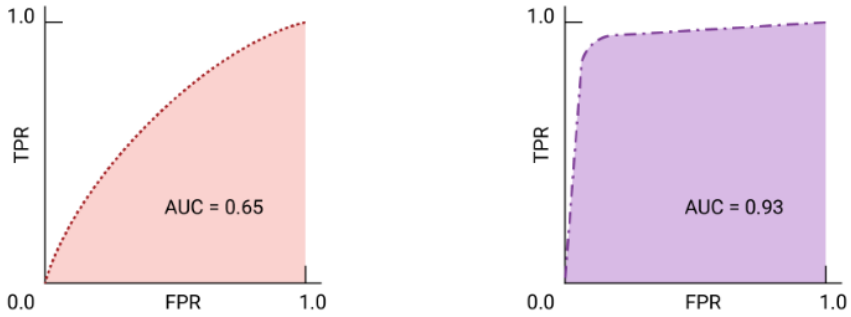

Figure 3.3: An illustration of ROC curve. AUC represents the overall performance of the classifier.

Finally, a classification report serves to summarize the model's per-class effectiveness, detailing measures such as precision, recall, F_1-score, and the count of examples for each class.

3.6 Popular CNN Architectures

This section looks at two important CNN architectures, MobileNetV2[SHZ+18] and EfficientNet[TL19], which are the focus of this thesis. MobileNetV2[SHZ+18] is designed to run well on mobile and low-power devices. It does this by using special types of layers, like depthwise separable convolutions and inverted residuals, to diminish the load of computation. EfficientNet-B0[TL19] takes a smart approach to

scaling CNNs by adjusting their depth, width, and input size all at once, helping it achieve high accuracy with fewer parameters.

Both architectures reflect a shift toward making deep learning more accessible, efficient, and scalable—qualities that are increasingly important as CNNs move from research labs to real-world applications.

MobileNetV2[SHZ+18] introduces a lightweight and efficient architecture optimized for mobile and embedded devices.

Depthwise separable convolutions are a key component of MobileNetV2[SHZ+18] that help reduce the computation needed for processing images. Normally, a convolutional layer applies a filter to each pixel in the input image, and this filter is the same for all channels. However, depthwise separable convolutions split this process into two parts:

Depthwise Convolution: Instead of using a single filter for all channels, each channel in the input is processed by its own filter. This reduces the number of computations because fewer filters are used.

Pointwise Convolution (1x1): After the depthwise convolution, a pointwise convolution (which uses a 1x1 filter) is applied to combine the outputs from the depthwise convolution. This step allows the model to mix the information from different channels and generate more complex features.

This technique is important because it substantially diminishes the parameter count and the amount of computation needed, which is crucial for mobile and embedded devices with limited resources.

Linear bottlenecks are another crucial feature in MobileNetV2's[SHZ+18] design. A bottleneck in a network typically refers to a layer that reduces the number of channels, and in MobileNetV2[SHZ+18], this is done using linear activations instead of the usual non-linear ones like ReLU.

In most architectures, non-linear activations like ReLU are applied after each convolution, but in MobileNetV2[SHZ+18], they are applied only at the very end of the block. This helps preserve information as it passes through the bottleneck and avoids the loss of valuable features in the intermediate layers. By avoiding non-linearity in the bottleneck layers, the architecture can lower the amount of computations and parameters while still preserving the model's ability to learn sophisticated patterns.

MobileNetV2[SHZ+18] begins with a standard 2D convolution layer that expands the input image (with 3 channels) to 32 channels. The core of the architecture is a sequence of **bottleneck blocks**, which are repeated with different configurations throughout the network. Each bottleneck block contains the next three steps:

1. **Expansion Layer**: A 1x1 convolution extends the number of input channels by a factor t.

2. **Depthwise Convolution**: A depthwise separable convolution filters spatial features independently across channels.

3. **Linear Bottleneck**: A 1x1 convolution reduces the number of channels back to c, using a *linear* activation (no non-linearity).

These blocks are designed to reduce computational cost while retaining representational power. As the model progresses, the spatial resolution is gradually reduced and the number of channels is expanded. At the end of the network, a final 1x1 convolution layer projects the feature maps to 1280 channels, succeeded by global average pooling and a final fully connected layer for classification.

Table 3.1 summarizes the full MobileNetV2[SHZ+18] architecture, including input sizes, block configurations, and strides.

Table 3.1: MobileNetV2[SHZ+18] Architecture Summary [SHZ+18]

Input	**Operator**	**t**	**c**	**n**	**s**
$224^2 \times 3$	Conv2D	-	32	1	2
$112^2 \times 32$	Bottleneck	1	16	1	1
$112^2 \times 16$	Bottleneck	6	24	2	2
$56^2 \times 24$	Bottleneck	6	32	3	2
$28^2 \times 32$	Bottleneck	6	64	4	2
$14^2 \times 64$	Bottleneck	6	96	3	1
$14^2 \times 96$	Bottleneck	6	160	3	2
$7^2 \times 160$	Bottleneck	6	320	1	1
$7^2 \times 320$	Conv2D 1x1	-	1280	1	1
$7^2 \times 1280$	AvgPool 7x7	-	-	1	-
$1^2 \times 1280$	Conv2D 1x1	-	k	1	-

Continuing the pursuit of highly efficient yet powerful neural networks, as seen with architectures like MobileNetV2[SHZ+18], EfficientNet-B0 [TL19] introduces a novel approach to model design. EfficientNet-B0[TL19] is the baseline model in the EfficientNet family and introduces a principled method of model scaling, designed for optimal performance with fewer parameters and FLOPs.

At the core of EfficientNet-B0 are **MBConv blocks**, which are inspired by the inverted residual blocks used in MobileNetV2. Each MBConv block consists of several key components:

1. **Expansion Layer**: A 1x1 convolution extends the number of input channels by a factor t.

2. **Depthwise Convolution**: Similar to MobileNetV2, a depthwise convolution is used to process each input channel independently, reducing execution cost.

3. **Squeeze-and-Excitation (SE) Block**: A lightweight attention mechanism that dynamically adjusts the importance of feature channels, allowing the model to prioritize more relevant information.

4. **Projection Layer**: A 1x1 convolution that lowers the number of channels back to the original input size, forming the bottleneck.

5. **Skip Connection**: If the input and output shapes match, a residual connection is added to help preserve information across layers and improve gradient flow.

These blocks allow EfficientNet to achieve high accuracy with fewer resources by leveraging both efficient design and advanced regularization techniques.

Unlike many earlier CNN architectures that use ReLU, EfficientNet employs the **Swish** activation function, defined as:

$$\text{Swish}(x) = x \cdot \text{sigmoid}(x)$$

Swish provides smoother gradients and allows for better information flow, especially in deeper networks. It has been empirically shown to outperform ReLU in many computer vision tasks.

EfficientNet introduces a novel **compound scaling** method to grow CNNs. Instead of arbitrarily increasing just the depth or width of a model, compound scaling jointly scales three dimensions:

- **Depth**: Number of layers in the network.
- **Width**: Number of channels in each layer.
- **Resolution**: Input image size.

This scaling is performed using a compound coefficient ϕ and predefined constants α, β, and γ, such that:

$$\text{depth} = \alpha^\phi, \quad \text{width} = \beta^\phi, \quad \text{resolution} = \gamma^\phi$$

subject to the constraint: $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$. [TL19]

This balanced approach ensures that the network grows in all dimensions in a resource-efficient manner, leading to improved accuracy and efficiency across the EfficientNet family.

EfficientNet-B0 begins with a standard 3x3 convolution layer followed by a series of MBConv blocks. Each block is configured with specific expansion factors, output

channels, number of repetitions, and strides. The architecture concludes with a 1x1 convolution, a global average pooling layer, and a final fully connected layer for classification.

Table 3.2 summarizes the EfficientNet-B0 architecture, adapted from [TL19].

Table 3.2: EfficientNet-B0 Architecture Summary

Input	Operator	exp	out	n	s
$224^2 \times 3$	Conv3x3	-	32	1	2
$112^2 \times 32$	MBConv1, 3x3	1	16	1	1
$112^2 \times 16$	MBConv6, 3x3	6	24	2	2
$56^2 \times 24$	MBConv6, 5x5	6	40	2	2
$28^2 \times 40$	MBConv6, 3x3	6	80	3	2
$14^2 \times 80$	MBConv6, 5x5	6	112	3	1
$14^2 \times 112$	MBConv6, 5x5	6	192	4	2
$7^2 \times 192$	MBConv6, 3x3	6	320	1	1
$7^2 \times 320$	Conv1x1	-	1280	1	1
$7^2 \times 1280$	AvgPool	-	-	1	-
$1^2 \times 1280$	FC	-	1000	1	-

3.7 One-Shot Learning

One-shot learning is a machine learning approach designed to enable models to recognize new object categories using only a single or very limited number of examples. Unlike conventional machine learning techniques that rely on large, labeled datasets for each category, one-shot learning focuses on achieving generalization from minimal data. This makes it especially valuable in situations where gathering extensive labeled examples is difficult or expensive. The key to this method is the model's ability to draw on its understanding of similarities and distinctions between classes to make accurate predictions from very little data.

3.8 Siamese Networks

Siamese networks are a specialized neural network architecture commonly applied in one-shot learning and tasks involving similarity comparison. Their central concept is to map inputs into a shared embedding space in such a way that inputs deemed similar are positioned close together, while those that are different are placed further apart.

This setup usually involves two identical subnetworks, hence the term "Siamese", that process a pair of inputs. These subnetworks share the same parameters and are trained to convert the inputs into fixed-size vector representations. The similarity or distance between these vectors is then computed to assess whether the input pair belongs to the same category or not.

Siamese networks typically incorporate the following elements:

- **Weight-Sharing Convolutional Layers**: These layers are responsible for extracting features from the input data. Because the layers share weights across both input branches, the network is encouraged to learn consistent features for inputs that are alike.
- **Similarity Measure**: A mathematical function, such as Euclidean distance or cosine similarity, is applied to the feature vectors generated by the network. This function evaluates how closely the two inputs resemble each other in the learned embedding space.
- **Training Objective**: The network is trained using an appropriate loss function that shapes the embedding space. Examples include contrastive loss, triplet loss, or binary cross-entropy. These loss functions aim to reduce the distance between embeddings of similar items and increase it for dissimilar ones. The selection of the loss function typically depends on the nature of the problem and the structure of the dataset.

Chapter 4

Landmark detection and recognition

This chapter covers the practical steps involved in building an efficient and deployable landmark classification system. We start by creating a balanced dataset from a publicly available collection, ensuring equal class distribution to prevent training bias.

For classification, we leverage pre-trained convolutional models known for their efficiency, applying fine-tuning to adapt them to our specific dataset. This approach speeds up training and enhances generalization with limited data.

Finally, we discuss deployment considerations, focusing on lightweight models suitable for real-time use on mobile or edge devices. These combined efforts establish a practical pipeline for scalable landmark recognition in real-world applications.

This chapter presents the process of preparing the dataset, fine-tuning pre-trained image classification architectures, and setting up the final model for deployment. The focus is on adapting existing models to our specific task and ensuring the system is both accurate and efficient in real-world use.

4.1 Dataset

For training the object detection model, we utilized the Pictures of Famous Places Dataset [Rya22], which consists of a comprehensive collection of images representing well-known landmarks and famous places from around the world. The dataset features an extensive collection of iconic landmarks, each associated with relevant categories or classes such as ”Eiffel Tower,” ”Tower Bridge,” ”Big Ben,” and many more iconic landmarks.

During the initial analysis, the dataset revealed an imbalance in the number of images per class, where some classes were significantly overrepresented compared to others. To address this issue, the dataset was curated to ensure that each class had the same number of images allocated for both training and testing purposes. This

balancing process helps mitigate class bias, ensuring that the object detection model does not become overly focused on overrepresented classes.

The final dataset consists of 6,100 images, with 50 distinct classes. Each class contains 61 images for both training and testing, resulting in an equal distribution of data across all classes. This approach ensures a reliable and fair assessment of the model's accuracy.

Total Images	Train Images per Class	Test Images per Class	Number of Classes
6,100	61	61	50

Table 4.1: Dataset Overview

To provide a clearer understanding of the types of images used for training and testing, below are a few sample images from the dataset. These images showcase some of the landmarks included in the dataset and demonstrate the variety of classes used for model training.

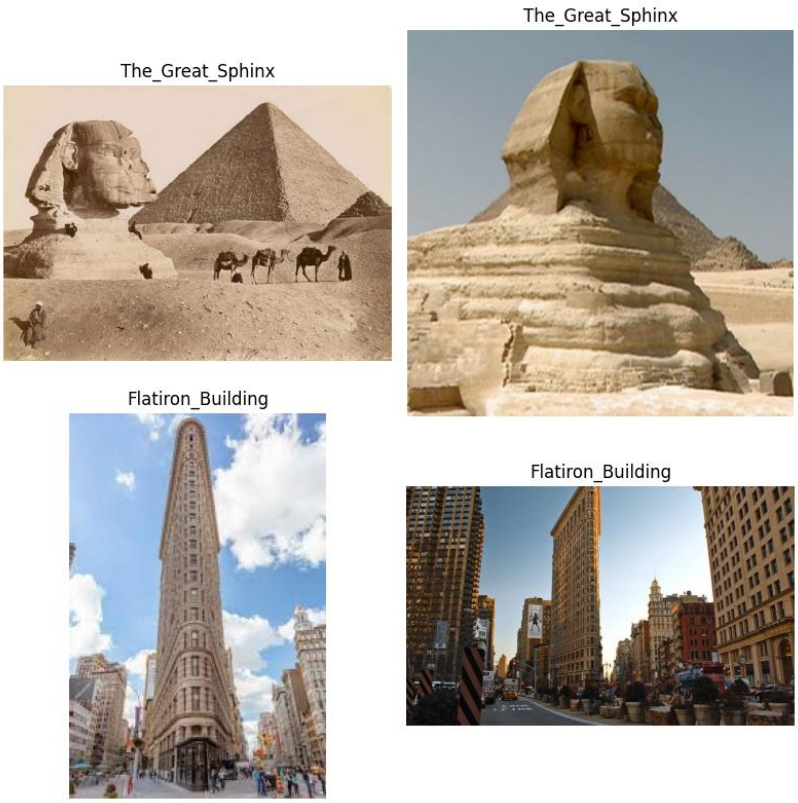

Figure 4.1: Sample Images from the Pictures of Famous Places Dataset.

[Rya22]

4.2 Model

The key to the landmark recognition system is represented by a deep learning-based image classifier built using CNNs. In particular, we explored two state-of-the-art architectures: **EfficientNetB0**[TL19] and **MobileNetV2**[SHZ+18], both of which are known for their strong performance on image classification tasks while maintaining computational efficiency. These models were employed using a transfer learning approach, leveraging pre-trained weights from ImageNet to accelerate training and improve generalization.

EfficientNetB0[TL19] is an initial model within the EfficientNet series, notable for pioneering a compound scaling approach. This technique simultaneously adjusts the network's depth, width, and input resolution based on a predetermined set of scaling coefficients. EfficientNet models are particularly well-regarded for their ability to reach high precision with reduced parameter counts when contrasted with traditional CNNs.

MobileNetV2[SHZ+18] is designed for lightweight applications, particularly on mobile devices. By utilizing depthwise separable convolutions and inverted residuals, it drastically reduces computational cost while preserving an impressive accuracy.

To expedite training and leverage the benefits of large-scale pre-training, we adopted a transfer learning approach using both EfficientNetB0[TL19] and MobileNetV2[SHZ+18] as the backbone models. These models were initialized with weights pre-trained on the ImageNet dataset [RDS+15], a collection of more than 1 million images. The pre-trained weights allow the models to extract generic image features effectively, even when the target dataset is relatively small.

For adaptation to our specific task, we removed the original classification head of each pre-trained CNN and substituted it with a new fully connected layer stack configured for 50 output classes. The new classification head includes a global average pooling layer, a dropout layer to mitigate overfitting, and a final dense layer that uses a Softmax activation to generate class probabilities.

The modified architecture is summarized as follows:

Table 4.2: EfficientNetB0 Classification Head

Layer	**Details**
include_top	False (removed original head)
GlobalAveragePooling2D	Output: 1D vector
Dense	1024 units, ReLU activation
Dropout	Rate = 0.2
Dense (output)	50 units, Softmax activation

Table 4.3: MobileNetV2 Classification Head

Layer	**Details**
include_top	False (removed original head)
GlobalAveragePooling2D	Output: 1D vector
Dense	1024 units, ReLU activation
Dropout	Rate = 0.5
Dense (output)	50 units, Softmax activation

After the initial training phase, the models were evaluated on the test set with the following outcomes:

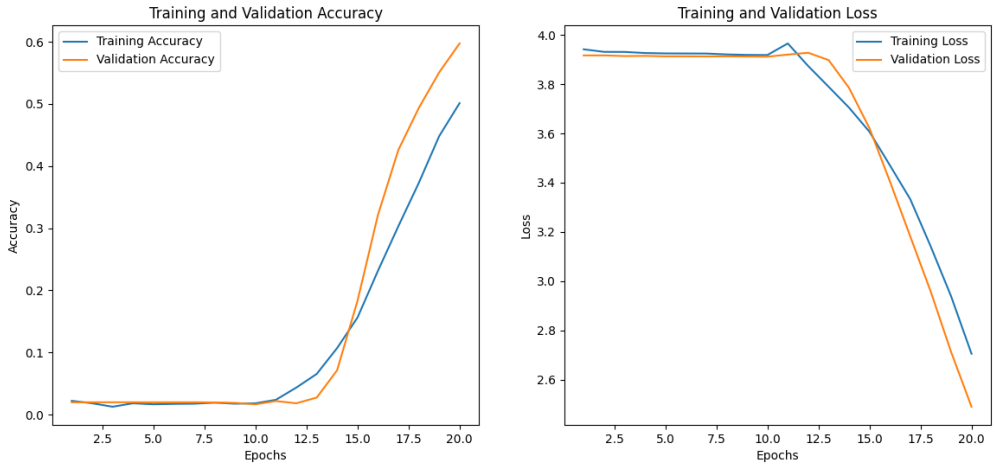

Figure 4.2: Accuracy and Loss curves for EfficientNetB0

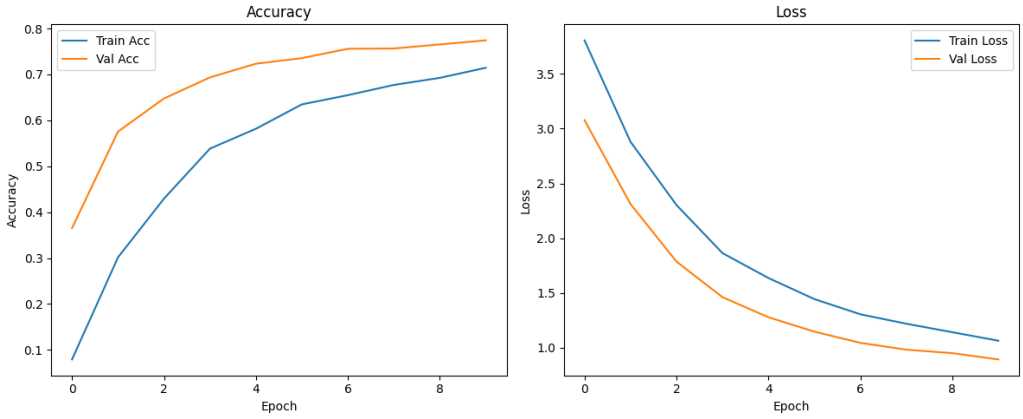

Figure 4.3: Accuracy and Loss curves for MobileNetV2

- **EfficientNetB0** achieved a test loss of approximately 2.53 and a test accuracy of 57.8%.

- **MobileNetV2**[SHZ+18] outperformed EfficientNetB0[TL19] with a test loss of about 0.99 and a test accuracy of 75.1%.

These results indicate that MobileNetV2[SHZ+18] not only converges more quickly but also generalizes better to unseen data in this task. The improved performance, combined with MobileNetV2's lighter architecture tailored for mobile and edge applications, makes it the preferred model for further refinement.

Given these findings, the remainder of this work will focus on MobileNetV2[SHZ+18] as the backbone for landmark classification.

Following the initial evaluation of MobileNetV2[SHZ+18] with the appended classification head, we proceed to the fine-tuning phase — a crucial stage in transfer learning that helps the model align its learning more precisely with the target dataset's particular characteristics.

Fine-tuning involves selectively unfreezing layers of the pre-trained convolutional base and retraining them on the new dataset. This step Boosts the model's capability to extract higher-level, domain-specific features, improving both accuracy and generalization capability.

In our approach, we unfreeze the top five convolutional layers of the MobileNetV2[SHZ+18] architecture, while keeping the earlier layers frozen. The rationale behind this is to allow fine adjustments to deeper layers (which capture more abstract patterns), while preserving the general, low-level features learned from the original ImageNet dataset.

To ensure stable learning during fine-tuning, we compile the model using a low learning rate of 1×10^{-5} with the Adam optimizer. This conservative learning rate minimizes the possibility of forgetting and ensures that the original weights are only slightly adjusted.

Regularization Techniques To prevent overfitting and to maintain training efficiency, we incorporate two key callbacks:

- **EarlyStopping**: This callback observes the validation loss and automatically stops the training process if there's no noticeable improvement over five consecutive training cycles (epochs).

- **ModelCheckpoint**: It saves the model that achieves the lowest validation loss throughout training, guaranteeing that the most effective version is available for subsequent assessment or use.

The training is conducted over a maximum of 50 epochs using the augmented training and validation datasets. The training process is summarized in Table 4.4.

Table 4.4: Steps in Fine-Tuning the MobileNetV2 Model

Step	Description
1	Unfreeze the top 5 convolutional layers for fine-tuning.
2	Compile the model using Adam optimizer with a learning rate of 1×10^{-5}.
3	Configure callbacks: `EarlyStopping` and `ModelCheckpoint`.
4	Train the model using augmented data for up to 50 epochs.

After initial fine-tuning, I further enhance the model's performance by incorporating regularization strategies and modifying the classification head. In this phase, I re-instantiate the MobileNetV2[SHZ+18] model with a new classification block that includes Dropout and L2 weight regularization — both of which are effective in reducing overfitting and improving model generalization.

I first extract the base MobileNetV2[SHZ+18] model from the composite model and unfreeze the top 10 layers, allowing these to be trainable. This strategy aims to update the deeper layers of the network while keeping the earlier, low-level feature extractors intact. The updated classification head consists of:

- **GlobalAveragePooling2D** layer to reduce the spatial dimensions.
- **Dropout layers** (rate = 0.3) applied before and after a fully connected layer to reduce excessive reliance between neurons.
- A dense layer with 256 units and ReLU activation, equipped with **L2 regularization** (factor = 0.001).
- A final softmax output layer matching the amount of target classes.

The data augmentation strategy is retained from earlier phases, and the model is set up with the Adam optimizer and employs categorical cross-entropy as its loss function. To ensure robust training and avoid overfitting, we reuse the previously defined callbacks:

- **EarlyStopping** with a patience of 5 epochs.
- **ModelCheckpoint** to store the best model based on validation loss.
- **ReduceLROnPlateau** to decrease the learning rate by a factor of 0.5 when validation loss plateaus, with a minimum learning rate threshold of 1×10^{-7}.

Training is conducted for a maximum of 30 epochs. The validation accuracy achieved at the end of this phase indicates a notable improvement, confirming the effectiveness of deeper fine-tuning with regularization. The training process is summarized in Table4.5

Table 4.5: Steps in Extended Fine-Tuning with Dropout and L2 Regularization

Step	**Description**
1	Extract base MobileNetV2 model from the composite model.
2	Unfreeze the top 10 layers of the MobileNetV2 base model.
3	Add a new classification head with Dropout and L2 regularization.
4	Recompile the model using Adam optimizer and categorical crossentropy.
5	Use data augmentation for training and validation sets.
6	Apply callbacks: EarlyStopping, ModelCheckpoint, and ReduceLROnPlateau.
7	Train the model for up to 30 epochs on the augmented dataset.
8	Evaluate performance on the validation set.

4.3 Exploration of a Siamese Network Approach

Beyond the standard classification setup, I also explored a Siamese network architecture for landmark recognition. This approach was particularly relevant for one-shot learning scenarios where data for new classes might be limited. This approach was investigated with the aim of facilitating future dataset expansion by efficiently incorporating novel landmark categories with minimal examples. The goal was to train a system capable of discerning similarities between images rather than directly classifying them into predefined categories.

Training a Siamese network requires a specific dataset structure of image pairs, labeled as either similar (positive pair) or dissimilar (negative pair). For this experiment, I generated pairs from our curated landmark dataset. Positive pairs were created by randomly selecting two distinct images belonging to the same landmark class. Conversely, negative pairs involved randomly selecting two images belonging to different landmark classes. For robust training, I aimed to balance the number

of positive and negative pairs generated. Each image served as an "anchor," from which both positive and negative comparison images were sampled. All incoming images were preprocessed by resizing to 224×224 pixels and normalizing pixel values prior to their input to the model.

The Siamese network design leverages the pre-trained MobileNetV2[SHZ+18] model as its core feature extractor. This choice was motivated by MobileNetV2's balance of efficiency and its power to capture complex features, as established in the initial evaluation. The architecture consists of two identical branches, both employing the MobileNetV2[SHZ+18] backbone to process two input images simultaneously.

In the first phase of training, the outputs from these twin MobileNetV2[SHZ+18] feature extractors are fed into a custom head designed to compute their similarity. Specifically, I calculated the L1 distance between the feature embeddings generated by each branch. This distance serves as the input to a small, fully connected neural network head tasked with predicting the similarity score. This head contains:

1. A dense layer with 128 units and ReLU activation.
2. A dropout layer (rate = 0.3) for regularization.
3. Another dense layer of 64 units with ReLU activation.
4. A second dropout layer (rate = 0.3).
5. A final dense layer with a single unit and a sigmoid activation function to output a similarity score between 0 (dissimilar) and 1 (highly similar).

The MobileNetV2[SHZ+18] feature extractor was kept frozen during this initial training phase, allowing the custom head to focus on learning how to interpret the distances between the high-level features extracted by the pre-trained backbone. The Siamese network was compiled using the Adam optimizer and binary cross-entropy loss, reflecting its binary classification task (similar/dissimilar).

To further enhance the model's ability to generalize to unseen data, a second fine-tuning phase was initiated. In this phase, the top 10 layers of the MobileNetV2[SHZ+18] feature extractor within the Siamese network were unfrozen, allowing their weights to be adjusted. This strategy facilitates the adaptation of the backbone's higher-level feature representations to the nuances of landmark image similarities. A lower learning rate (2×10^{-5}) was used for this fine-tuning phase to prevent drastic changes to the pre-trained weights. To ensure stable learning and preserve the best-performing model, several regularization techniques were employed via Keras callbacks:

- **EarlyStopping**: This callback monitored the validation loss, stopping training if no improvement occurred for 10 consecutive epochs. It also restored the best weights upon stopping.
- **ModelCheckpoint**: This callback saved the model with the lowest validation loss during training, ensuring the optimal version was preserved.
- **ReduceLROnPlateau**: This callback adjusted the learning rate. If validation loss didn't improve for 5 epochs , it reduced the learning rate by a factor of 0.2 , down to a minimum of 1×10^{-7}, helping convergence.

After training and fine-tuning our Siamese network, the next step is to see how well it could handle one-shot learning. This means checking its ability to identify new landmark images when it only has one ,or very few, example pictures of that landmark. Unlike typical image classification where a model directly names a picture's category, one-shot evaluation tests how well the network can tell an unknown image apart from a small group of reference images by comparing their similarities.

I set up the evaluation as an N-way one-shot task, specifically a 5-way one-shot paradigm, and ran 500 trials. This means that in each trial, the model had to choose the correct landmark from 5 different, previously unseen, landmark categories.

For each trial, an unseen query image was selected. Subsequently, five distinct landmark categories were randomly chosen. A single support image (reference example) was provided for each of these five categories. The trained Siamese network then computed a similarity score between the query image and each of the five support images. The network's prediction for the query image's class was the category associated with the support image that yielded the highest similarity score.

The model's performance in this context was quantified by its one-shot accuracy. This metric represents the proportion of trials in which the model correctly identified the query image's class.

4.4 Deployment

To enable easy access and real-time inference, the best fine-tuned MobileNetV2 model was deployed as a RESTful API using the Flask web framework. The deployment is hosted on Render, a cloud platform for hosting web services, accessible at `https://licentaserver.onrender.com`.

The Flask application loads the trained model and its corresponding class mapping upon the first prediction request. It exposes a `/predict` endpoint that accepts POST requests containing an image file. Images undergo preprocessing, including resizing to 224×224 pixels and normalizing pixel values, prior to feeding them into

the model. The API returns the predicted class label in JSON format, providing a user-friendly interface for classification of famous places.

This deployment approach facilitates seamless integration with front-end applications or other services, enabling scalable and efficient usage of the deep learning model in production environments. By leveraging Flask's lightweight server capabilities and Render's cloud infrastructure, the system ensures reliable availability and low latency for inference requests.

4.5 Implementation

The entire project was developed using Python, mainly because of its straightforward syntax and the wide range of libraries it offers for both machine learning and web development tasks. The solution consists of two major components: model training and deployment.

For the image classification task, I used TensorFlow along with its high-level API, Keras. The model architecture is developed on MobileNetV2[SHZ+18], a lightweight and efficient CNN that performs well on image classification tasks, especially in resource-constrained environments.

To enhance the model's capacity to perform well on novel data, I applied real-time data augmentation using the `ImageDataGenerator` class from Keras. Augmentation techniques such as rotation, zooming, and flipping were employed. Additionally, training was optimized using callback functions like `EarlyStopping`, `ModelCheckpoint`, and `ReduceLROnPlateau` to prevent overfitting and dynamically adjust learning rates.

After training and evaluation, the model was stored and integrated into a web application using Flask, a lightweight Python web framework designed for simplicity and flexibility. The application exposes a single endpoint, `/predict`, which accepts image uploads via HTTP POST requests and returns a prediction in JSON format.

The image preprocessing for inference ,such as resizing and normalization, was handled using the Pillow (PIL) library and NumPy. The class predictions were mapped using a JSON file that stores the index-to-class mappings, ensuring the output is human-readable.

To make the model accessible to users, the Flask application was deployed on Render, a cloud platform that provides a straightforward interface for hosting web services.

By integrating these technologies, the project delivers a complete pipeline—from training a deep learning model to serving predictions through a user-accessible web API.

Chapter 5

Mobile Application

This chapter presents the development of the mobile application, which acts as the user interface for the image classification system. The app allows users to take photos of places or landmarks and receive real-time predictions that identify them.

The main purpose of the app is to make traveling more engaging and fun, introducing a competitive element to exploring new cities. Inspired by interactive experiences like the Space Invaders game in Paris, which turns city discovery into a game, this app aims to encourage users to explore without rigid plans.

Users can follow challenges tailored to guide them through cities , adding a sense of adventure and direction. The app lets users add friends, track all their visited locations on a map over time, complete challenges, and earn experience points. Visit validation is done through a combination of GPS location and instant photo verification powered by AI, ensuring an interactive and authentic travel experience.

The following sections will detail the development framework, architecture, user interface and backend integration.

5.1 Development Framework

The mobile application was developed using the Android platform, leveraging Kotlin as the primary programming language. Kotlin was chosen for its modern syntax, safety features, and seamless integration with Android development tools.

For the user interface, Jetpack Compose was used. Compose simplifies UI development by allowing declarative UI design, resulting in more readable and maintainable code, as well as faster iteration cycles.

Firebase provided the backend infrastructure, addressing needs like user authentication, real-time database synchronization, and cloud storage. Its scalable and easy-to-use infrastructure made it an ideal choice for managing user data and app backend without the need for complex server setup.

Additional libraries and tools used in the development include:

- **Retrofit** – for networking and API calls between the app and the Flask server hosting the image classification model.
- **Coroutines** – to handle asynchronous operations and improve app responsiveness.
- **Google Maps SDK** – for displaying visited locations and challenges on an interactive map.
- **Hilt** – for dependency injection to simplify code modularity.
- **CameraX** – to manage camera functions for capturing photos within the app.

These technologies combined provide a robust foundation for a responsive, scalable, and user-friendly mobile experience.

5.2 Application Architecture

The mobile application follows the MVVM (Model–View–ViewModel) architecture, a modern and widely adopted pattern in Android development. MVVM provides a clear separation of concerns and helps keep the codebase organized and maintainable. It is particularly effective for handling asynchronous data flows and dynamic UI updates, making it a strong fit for this application's real-time interactions and state-driven interface.

The MVVM components are divided as follows:

View (UI Layer) The View is implemented using Jetpack Compose, which enables declarative UI design. Each screen observes state from the ViewModel and reacts automatically to changes. The View is stateless in itself and only responsible for rendering UI components and forwarding user events.

ViewModel The ViewModel acts as an intermediary between the UI and the business logic. It holds UI-related data using StateFlow or LiveData, processes user interactions, and triggers updates in the Model. ViewModels manage coroutine scopes to handle asynchronous operations like API calls or Firebase interactions.

Model The Model layer is responsible for managing application data and core business logic. It includes:

- Network and database access (e.g., Retrofit, Firebase).
- Domain logic such as location verification, challenge completion, and prediction handling.

- Repositories that abstract data sources and provide clean APIs to the ViewModel.

MVVM is particularly well suited for this application due to several key factors. First, it supports real-time updates, allowing users to receive immediate feedback from image classification, location verification, and progress tracking. Additionally, the architecture handles multiple data sources efficiently, combining input from the device camera, GPS, an external API (Flask server) and Firebase services. The use of lifecycle-aware ViewModels ensures that data survive configuration changes, providing a consistent and smooth user experience. Finally, MVVM promotes a cleaner structure by assigning a single responsibility to each layer, which helps to keep the codebase organized and easier to maintain.

5.3 Key Functionalities and Use Cases

This section outlines the main use cases of the mobile application from a user's perspective. Each use case represents a distinct user interaction or feature provided by the app. These use cases serve as the foundation for the application's architecture, backend integration, and user interface design.

The use case diagram in Figure 5.1 illustrates the key interactions.

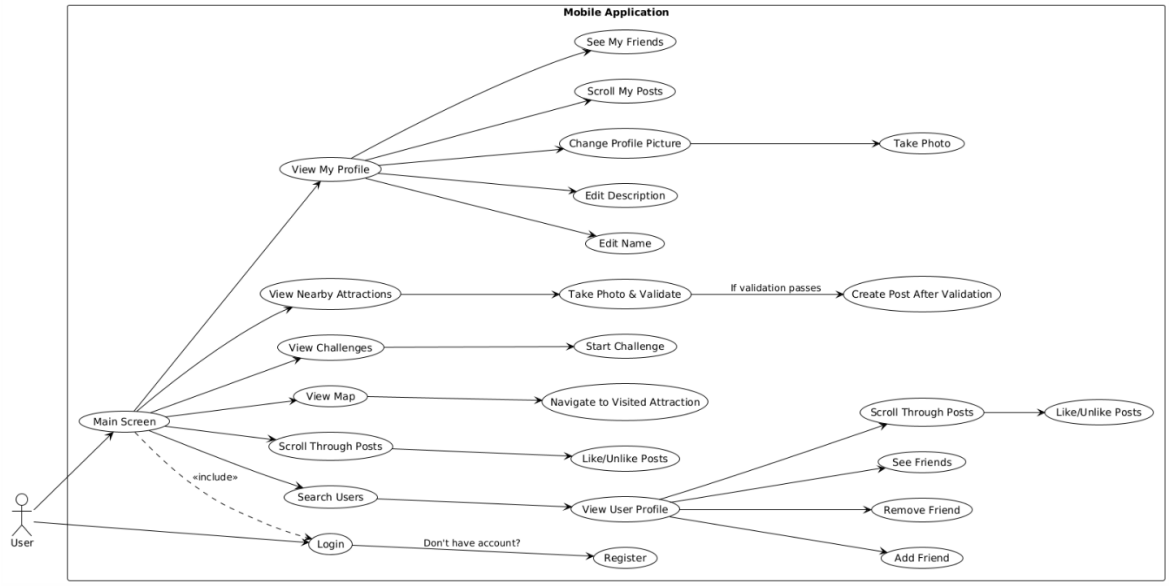

Figure 5.1: Use Case Diagram for the Mobile Application

The login screen is the first thing users see when they launch the app. They can sign in using their email and password. In cases where an account has not yet been created, there's a clear option to go to the registration screen. During registration, users enter their email, choose a password, and confirm it. The app checks that the

email is correctly formatted and that the password meets the requirements before creating a new profile with a default username and profile picture. Both login and registration use Firebase Authentication to securely handle user credentials. Once the process is successful, the app gets a unique user ID token from Firebase, which helps identify the user and personalize their experience.

Figures 5.2 and 5.3 illustrates the login and register flow in detail.

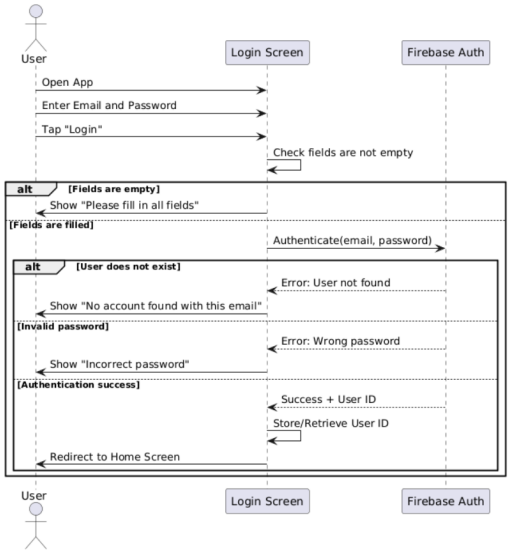

Figure 5.2: Sequence Diagram for User Login

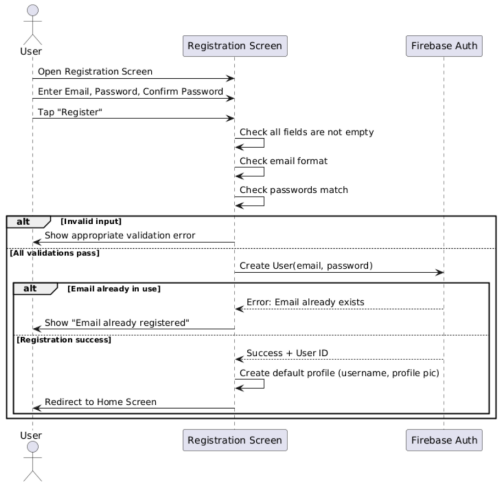

Figure 5.3: Sequence Diagram for User Register

After logging in, users are taken to the home screen, where they can view posts from their friends made within the last 48 hours. They can scroll through these posts and interact by liking or unliking them. At the top of the screen, a search bar allows users to find other people. When viewing another user's profile, they can add or remove them as a friend, see their friends list, browse their posts, and view their current level in the app. The level system adds an element of gamification, encouraging users to remain engaged and connected.

The app also includes a map feature that shows pins marking all the locations a user has visited. These pins provide a visual way to explore past places directly on the map. In one corner of the screen, there's a list displaying all these locations by name. When a user taps on a location from the list, the map automatically moves to that specific pin, making it easy to navigate between visited places. This feature is built using Firebase to store location data, and it leverages the native Android Maps API for smooth, interactive map functionality.

The "Create Post" feature allows users to share geo-verified photos of real-world attractions. When the user opens the post creation screen, the application first retrieves the user's current GPS location. Using this data, it queries Firebase Firestore

for nearby attractions within a 500-meter radius. These attractions are dynamically displayed in a list that updates in real-time as the user's location changes.

Once the user selects an attraction from the list, the camera interface opens, allowing them to take or retake a photo. After capturing the image, the user can submit it for validation. The photo is sent to an AI model hosted at

`https://licentaserver.onrender.com`, which returns a predicted class label based on the image content. If the predicted label matches the name of the selected attraction, a new post is created in Firebase Firestore containing the photo and location metadata. If there is a mismatch, an error is shown, prompting the user to try again.

Figure 5.4 illustrates this process through a sequence diagram.

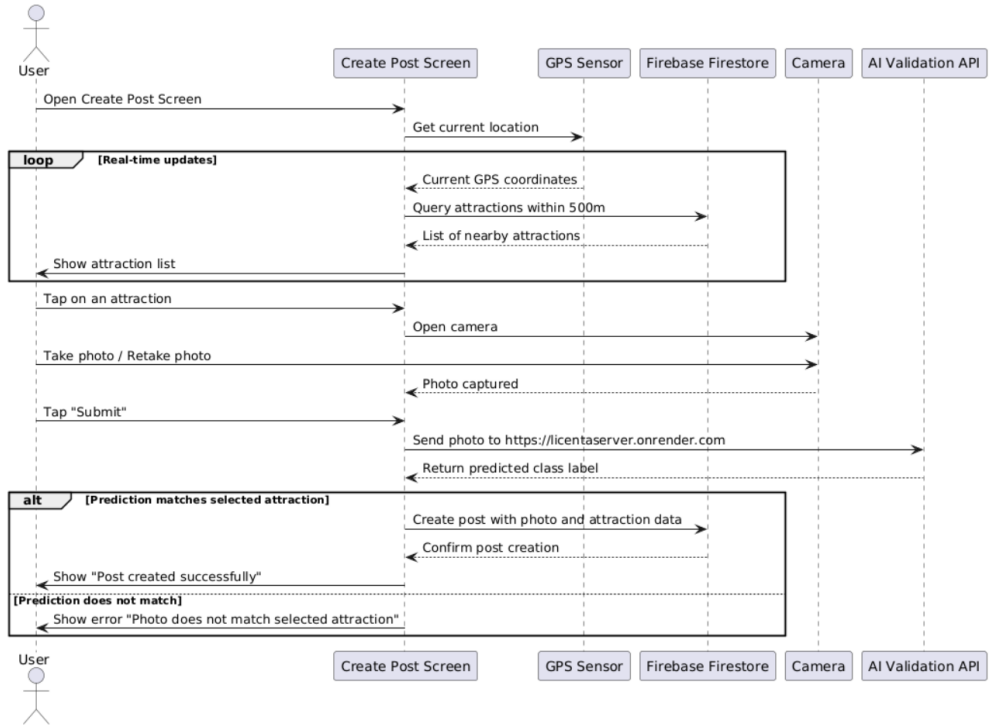

Figure 5.4: Sequence Diagram for Create Post Feature

The Challenges screen is a key part of how the app encourages exploration through gamification. It gives users an easy way to discover new places by offering pre-made itineraries tailored to the city they're in—without needing to spend time planning. Challenges are organized into three groups: those in progress, those not yet started, and completed ones.

Each challenge comes with a specific set of locations and a time limit, guiding the user through a curated experience. The focus is on making exploration accessible and enjoyable. By starting a challenge, users commit to visiting a series of places within a certain timeframe, turning travel into a fun and goal-driven activity. This

feature helps users stay engaged with the app and makes sightseeing more interactive and rewarding.

The user profile page provides a personalized space where users can manage their account and track their progress. From this screen, users can update their username and description, as well as take a photo to set as their profile picture. The interface also includes a section listing all the user's friends, allowing for quick access to their profiles and interactions.

An important feature of the profile page is the level tracking system, which reflects the user's engagement within the app. Each time a user creates a post, they earn 1 experience point (XP). After accumulating 5 XP, the user advances to the next level. Additionally, completing a challenge results in an immediate level-up, on top of the XP gained from any posts made during that challenge. This system encourages users to explore and engage more actively with the app.

The profile screen also functions as a personal archive, allowing users to scroll through all their previous posts. This makes it easy to revisit past experiences and see how much they've explored and accomplished over time.

5.4 User Interface Design

The mobile application's user interface was build to be clean, intuitive, and easy to navigate. Built using Jetpack Compose, the app uses a modern design approach that helps keep the layout consistent and responsive across different screens. Each part of the UI is focused on making the user experience smooth and straightforward, with clear visual cues, fast feedback, and simple navigation.

The layout adapts well to different screen sizes, making the app just as functional and visually balanced on tablets as it is on smartphones.

The screenshots below show the main screens and features described earlier, giving a visual overview of how users interact with the app.

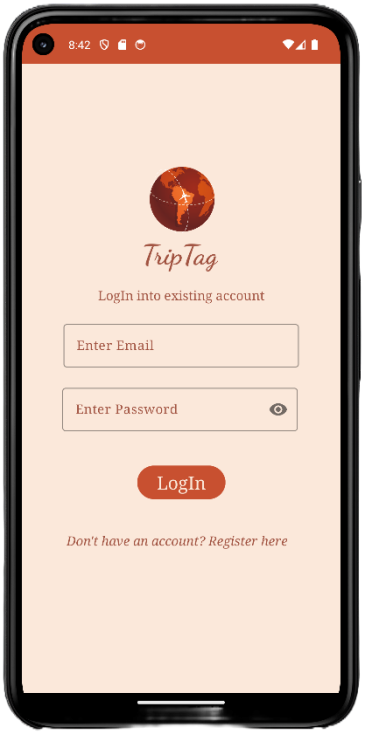

Figure 5.5: Screen for User Login

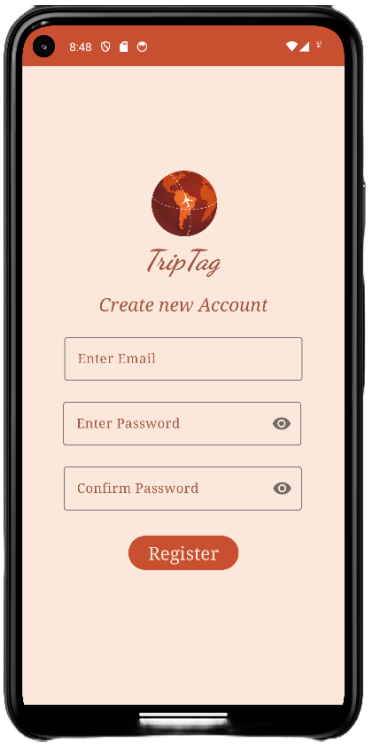

Figure 5.6: Screen for User Register

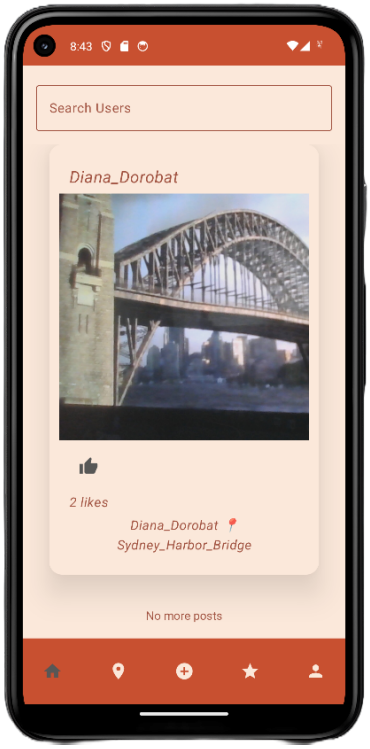

Figure 5.7: Screen for Home

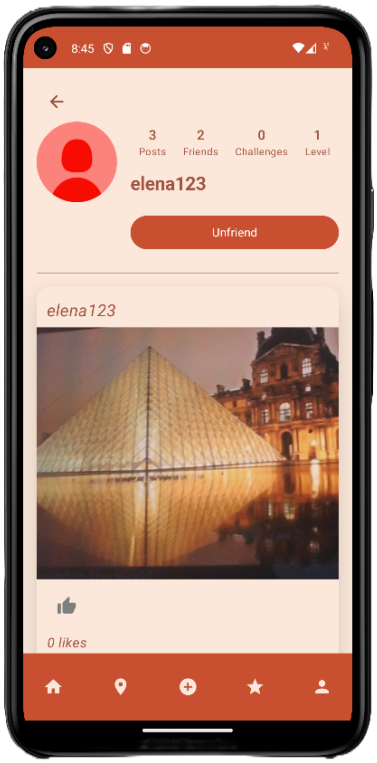

Figure 5.8: Screen for Profile

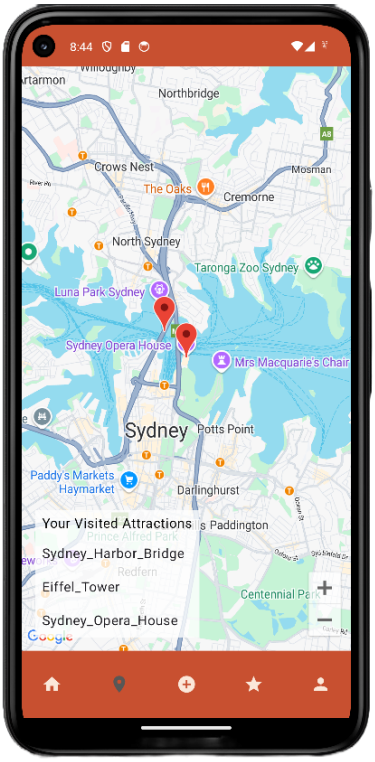

Figure 5.9: Screen for Map

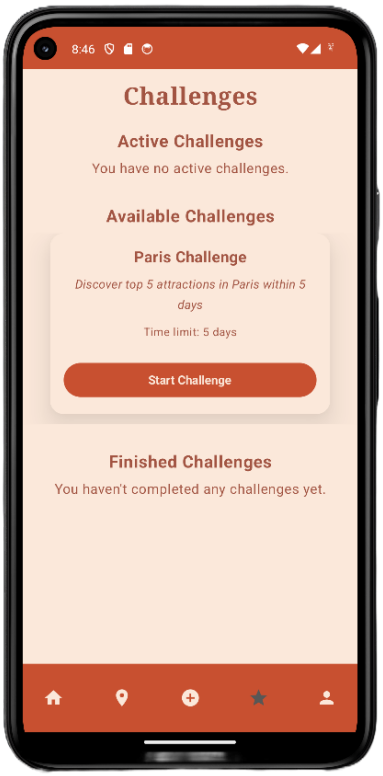

Figure 5.10: Screen for Challenges

Figure 5.11: Screen for Nearby Attractions

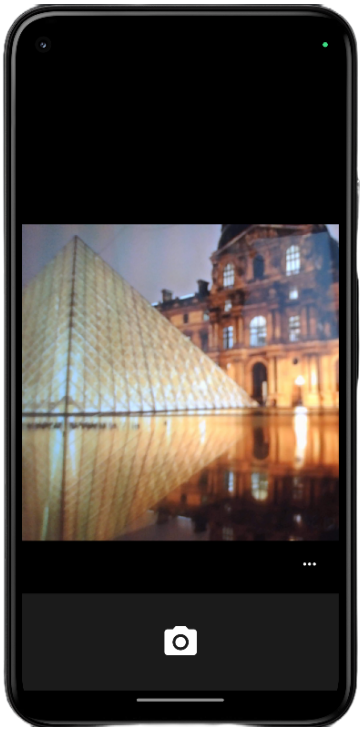

Figure 5.12: Screen for Taking photo of attractions

Chapter 6

Experimental Results

This chapter presents the outcomes of the experiments conducted to assess the performance of the fine-tuned MobileNetV2[SHZ+18] model and its integration into the mobile application. Includes model performance metrics and comparisons with baseline or other models.

6.1 Performance evaluation

Figures 6.1 and 6.2 illustrate the model's learning progression throughout the training process by graphing the accuracy and loss values across the training epochs, respectively.

Figures 6.1 and 6.2 show how my model's accuracy and loss shaped up during its final fine-tuning stages, right before I settled on the version for deployment.

It can be seen in Figure 6.1 that the train accuracy climbed impressively past 0.95 and then leveled out. This tells that the model really got a sound grasp of the training data. Meanwhile, the validation accuracy hit a solid peak around 0.85.

I decided to stop training around the 30-epoch mark because, at this point, the validation accuracy had started to level off. Also, as seen in Figure 6.2, the validation loss, despite its general downward trend, began to flatten out. This was a deliberate choice to pick the model version that would perform best on new, unseen data, rather than letting it get too specialized on just the training set.

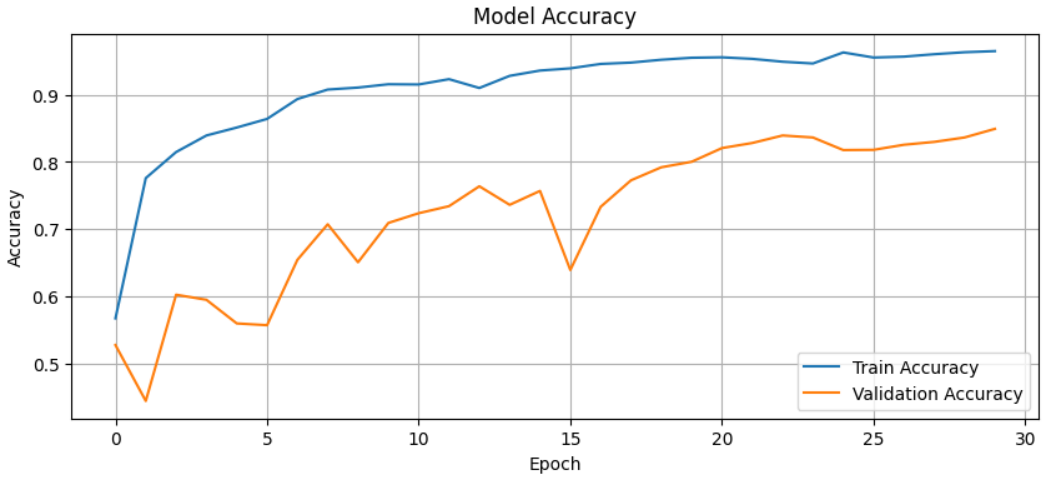

Figure 6.1: Accuracy curve

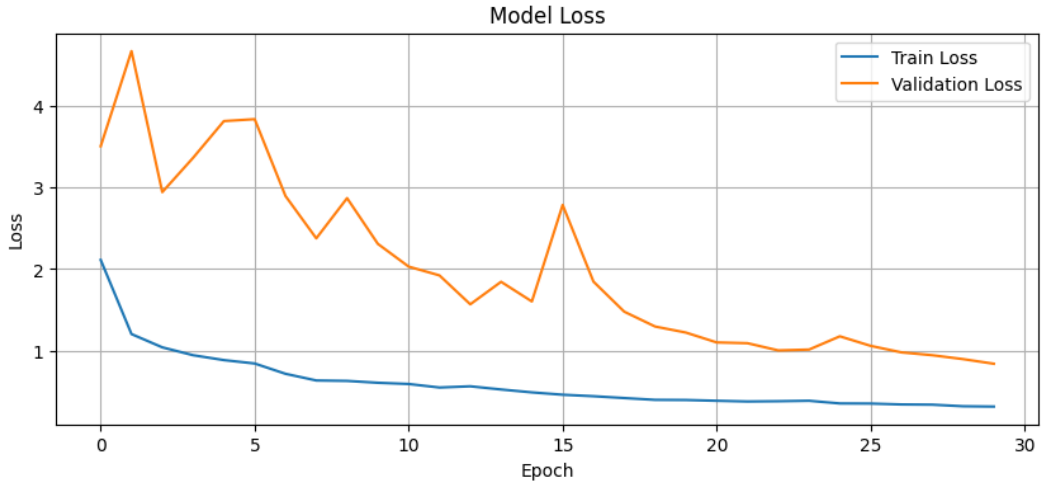

Figure 6.2: Loss curve

To get a clearer picture of how effectively the model discriminates between the different classes, we can look at the confusion matrix in Figure 6.3. This matrix, from "Phase 3" of my work, essentially maps out the model's predictions against the actual true classes for all 50 categories.

The first thing that stands out is the strong, dark diagonal line running from the top-left to the bottom-right. This is excellent news, as it means the model correctly identified the class for the vast majority of instances. Most of the other squares off the diagonal are very light, indicating that misclassifications were generally infrequent.

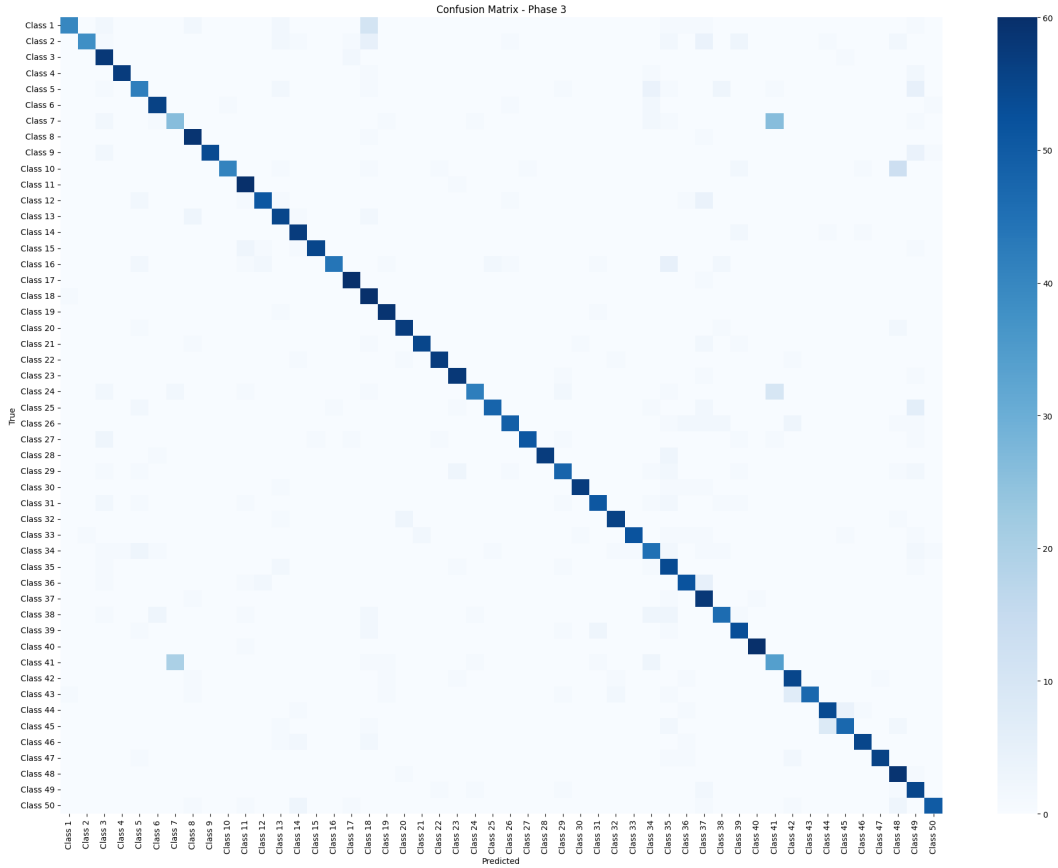

Figure 6.3: Confusion Matrix

Complementing the visual insights from the confusion matrix, the Classification Report, shown in Figure 6.4, gives us a numerical breakdown of the model's performance for each of the 50 classes. This report details the precision, recall, and f1-score for every class, along with the support, which is consistently 61 samples per class, indicating a well-balanced test set.

Overall, the model achieves an accuracy of approximately 84.9%. The macro average f1-score stands at about 0.850, and the weighted average f1-score is similar, reflecting solid performance across the board. Looking at individual classes, many achieve excellent precision and recall, with f1-scores often well above 0.90 – for instance, "Class 9," "Class 18," and "Class 28" show particularly strong results.

The classification report offers a valuable deeper dive into the performance nuances for each class, further clarifying the patterns observed in the confusion matrix. For instance, while the vast majority of classes are classified with high accuracy, the report details the specific performance for categories such as 'Class 7', labeled 'Blue Mosque' (with an f1-score around 0.47), and 'Class 41', labeled 'Sultan Ahmed Mosque' (f1-score around 0.51).

It's critical to note that these two classes actually represent the very same landmark – the Sultan Ahmed Mosque is the official name for the structure more pop-

ularly known as the Blue Mosque. Their appearance as distinct categories in the dataset is an artifact of the data sourcing process: the website from which the images were scraped listed both ”Blue Mosque” and ”Sultan Ahmed Mosque” as separate entries in its top attractions.

Similarly, variations in recall values for other iconic, yet visually distinct, landmarks like ’Class 1’ (Angkor Wat), ’Class 24’ (Hagia Sophia), and ’Class 30’ (Machu Picchu) indicate that some of their instances were occasionally missed. This points to the unique and sometimes intricate visual characteristics of certain sites that can present a challenge.

In conclusion, the performance evaluation of the fine-tuned MobileNetV2 model demonstrates its strong capabilities and suitability for the intended task. The accuracy and loss curves indicated effective learning, with training strategically halted at an optimal point, achieving a solid validation accuracy of approximately 85

Column1	precision	recall	f1-score	support
Class 1	0.952381	0.655738	0.776699	61
Class 2	0.974359	0.622951	0.76	61
Class 3	0.7435897	0.95082	0.834532	61
Class 4	0.9827586	0.934426	0.957983	61
Class 5	0.75	0.688525	0.717949	61
Class 6	0.9032258	0.918033	0.910569	61
Class 7	0.5416667	0.42623	0.477064	61
Class 8	0.8550725	0.967213	0.907692	61
Class 9	1	0.885246	0.93913	61
Class 10	0.9761905	0.672131	0.796117	61
Class 11	0.8450704	0.983607	0.909091	61
Class 12	0.9107143	0.836066	0.871795	61
Class 13	0.7857143	0.901639	0.839695	61
Class 14	0.8507463	0.934426	0.890625	61
Class 15	0.9821429	0.901639	0.940171	61
Class 16	0.9777778	0.721311	0.830189	61
Class 17	0.9230769	0.983607	0.952381	61
Class 18	0.6521739	0.983607	0.784314	61
Class 19	0.921875	0.967213	0.944	61
Class 20	0.9193548	0.934426	0.926829	61
Class 21	0.9649123	0.901639	0.932203	61
Class 22	0.95	0.934426	0.942149	61
Class 23	0.8923077	0.95082	0.920635	61
Class 24	0.9130435	0.688525	0.785047	61
Class 25	0.9411765	0.786885	0.857143	61
Class 26	0.9074074	0.803279	0.852174	61

(a) Top half of report

Class 27	0.9807692	0.836066	0.902655	61
Class 28	1	0.934426	0.966102	61
Class 29	0.8275862	0.786885	0.806723	61
Class 30	0.9661017	0.934426	0.95	61
Class 31	0.8947368	0.836066	0.864407	61
Class 32	0.9180328	0.918033	0.918033	61
Class 33	0.9811321	0.852459	0.912281	61
Class 34	0.703125	0.737705	0.72	61
Class 35	0.627907	0.885246	0.734694	61
Class 36	0.8387097	0.852459	0.845528	61
Class 37	0.6666667	0.95082	0.783784	61
Class 38	0.8214286	0.754098	0.786325	61
Class 39	0.8153846	0.868852	0.84127	61
Class 40	0.9836066	0.983607	0.983607	61
Class 41	0.4722222	0.557377	0.511278	61
Class 42	0.7971014	0.901639	0.846154	61
Class 43	1	0.770492	0.87037	61
Class 44	0.84375	0.885246	0.864	61
Class 45	0.8867925	0.770492	0.824561	61
Class 46	0.9482759	0.901639	0.92437	61
Class 47	0.9824561	0.918033	0.949153	61
Class 48	0.6941176	0.967213	0.808219	61
Class 49	0.6470588	0.901639	0.753425	61
Class 50	0.9433962	0.819672	0.877193	61
accuracy	0.8491803	0.84918	0.84918	0.84918
macro avg	0.8651419	0.84918	0.850006	3050
weighted av	0.8651419	0.84918	0.850006	3050

(b) Bottom half of report

Figure 6.4: Classification Report (presented in two parts for better layout)

Beyond the standard classification evaluation, I also assessed the trained Siamese network’s performance in an 5-way one-shot learning setup.

The initial training phase of the Siamese network’s custom head, while the MobileNetV2 feature extractor was frozen, is illustrated by its loss and accuracy curves in figure6.5. As seen, the loss decreased rapidly, and accuracy improved, indicating effective learning by the newly added layers. However, the validation loss began to plateau after a certain point, suggesting that the custom head had learned as much

as it could with a frozen backbone, and further fine-tuning of the feature extractor was necessary to achieve optimal performance.

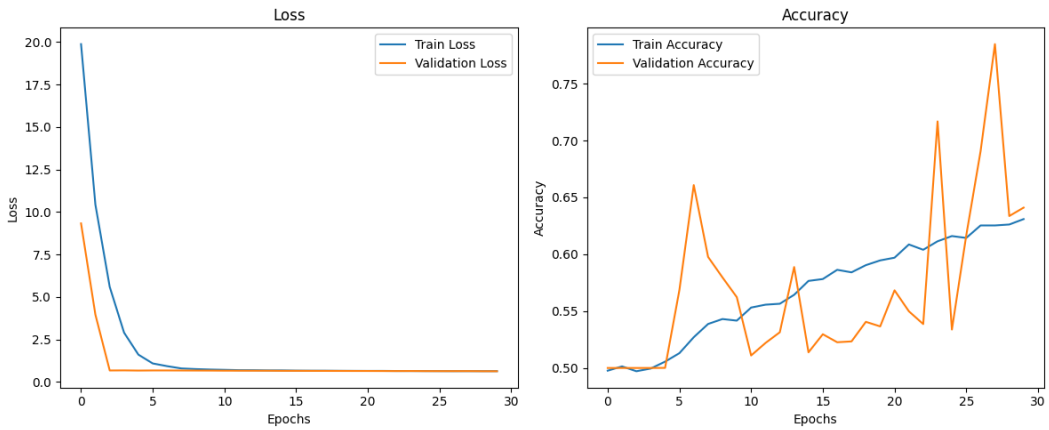

Figure 6.5: Siamese network's initial training phase

Following this, the subsequent fine-tuning phase, where the top layers of the MobileNetV2[SHZ+18] backbone were unfrozen, showed continued improvement, as depicted by the training curves in figure6.6. While the training loss continued to decrease and training accuracy steadily climbed, the increasing divergence between training and validation metrics indicated signs of overfitting. This observation prompted the decision to halt training and utilize the model from the epoch that yielded the best validation performance.

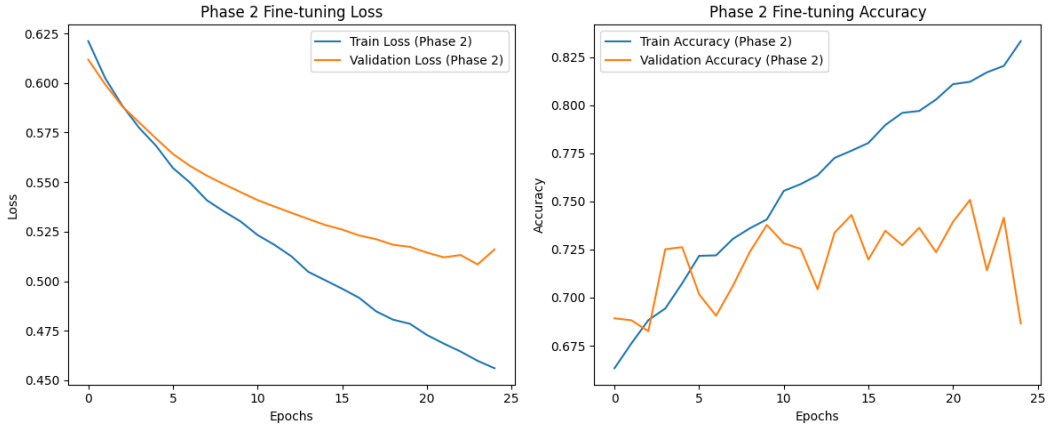

Figure 6.6: Siamese network's initial training phase

Across 500 trials of a 5-way one-shot task, the Siamese model achieved an accuracy of 81.20% (406 correct predictions out of 500 trials). An example of such a trial is presented in Figure 6.7, demonstrating how the model compares a query image to a set of support images and selects the most similar one.

Figure 6.7: Siamese network's second training phase

While respectable for a scenario with extremely limited examples per class, this performance was comparatively lower than the 84.9% accuracy attained by the MobileNetV2[SHZ model configured for direct multi-class classification. This outcome indicates that while the Siamese approach holds significant promise for tasks involving novel categories with scarce data, its current iteration requires further refinement and optimization to match the direct classification model's overall accuracy for this specific dataset. Consequently, the standard fine-tuned MobileNetV2[SHZ+18] classifier was selected as the primary solution for deployment due to its superior current performance.

6.2 Comparison with Other Methods

To contextualize the performance of the fine-tuned MobileNetV2[SHZ+18] model, this section compares it to both its baseline counterpart and other established methods.

The baseline model serves as the main point of comparison for evaluating the impact of fine-tuning. It uses theMobileNetV2[SHZ+18] architecture pre-trained on the ImageNet[RDS+15] dataset, with only the final classification layers trained on the Pictures of Famous Places Dataset[Rya22]. The convolutional base of the network was kept frozen to rely solely on the features learned during pretraining. This setup helps measure how well the pre-trained MobileNetV2[SHZ+18] can generalize to the specific task before any further adaptation.

The baseline MobileNetV2[SHZ+18] model, with a frozen convolutional base and only the classifier head trained, achieved a validation accuracy of 77% as shown in Figure 4.3. In contrast, the fine-tuned model, where selected base layers were unfrozen and trained, showed notable improvements, as summarized in Table 6.1. Detailed results and training curves are presented in Section 6.1.

Table 6.1: Performance Comparison: Baseline vs. Fine-tuned MobileNetV2

Metric	**Baseline Model**	**Fine-tuned Model**	**Improvement (pp)**
Accuracy	77.0%	84.9%	+7.9
F1-score (macro average)	77.0%	85.0%	+8.0

pp: percentage points

Table 6.1 shows that fine-tuning improved accuracy by 7.9 percentage points and macro F1-score by 8.0, highlighting the effectiveness of adapting pre-trained features to the target dataset.

To put my fine-tuned MobileNetV2[SHZ+18] results in context, it helps to look at two well-known approaches: the top models from the Google Landmark Recognition 2021 competition[ACA+21] and NU-LiteNet[TKM18].

The Google Landmark Recognition 2021 competition[ACA+21] featured some really impressive models built on the huge GLDv2 dataset[RTC21], which has over 4 million images across 200,000 classes. The winners, like Christof Henkel et al.'s first-place solution[Hen21] and J. Radenović et al.'s third-place model[XWL+21], used complex architectures combining CNNs and transformers or attention mechanisms to get GAP scores close to 0.49. These models are very accurate but need a lot of computing power, so they usually run on servers or cloud setups.

NU-LiteNet[TKM18], on the other hand, was developed to be easily deployable and efficient enough to run on mobile devices. It reached top-1 accuracies of 81.15% on the Singapore dataset (50 classes, 4,060 images) and 69.58% on the Paris dataset (12 classes, 1,200 images), with really small model sizes of 0.94 MB (B variant) and 0.27 MB (A variant). My MobileNetV2[SHZ+18] model, fine-tuned and running on Render using a less expensive variant, got an accuracy of 84.9% on Pictures of Famous Places Dataset[Rya22] (about 6,100 images over 50 classes), with a bigger model size of 14 MB. So, while it's not as compact as NU-LiteNet[TKM18], it still holds its own in accuracy. Comparison between the two proposed solutions is hard, since the datasets and evaluation methods aren't exactly the same.

To wrap up, the Google Landmark winners clearly push the accuracy limits but require heavy-duty hardware. NU-LiteNet[TKM18] and my MobileNetV2[SHZ+18] approach lean more towards efficiency and lower resource use. My model fits well within the range of these established methods, offering a solid balance between good accuracy and practical deployment costs.

Chapter 7

Conclusions and future work

In this project, we successfully fine-tuned the MobileNetV2[SHZ+18] model and integrated it into a lightweight deployment setup on Render, prioritizing cost efficiency without compromising performance. The model demonstrated strong classification capabilities on the Pictures of Famous Places dataset[Rya22], achieving a solid validation accuracy of approximately 85%, a significant improvement over the baseline and competitive with other mobile-friendly approaches.

Evaluation results showed that the model learns effectively and generalizes well to unencountered data, as reflected by the accuracy and loss curves, confusion matrix, and detailed classification report in Section 6.1. Although some landmark classes exhibited lower performance, these cases often stemmed from dataset inconsistencies or visually similar landmarks labeled separately.

Looking ahead, there is potential to extend the system by increasing the number of landmark classes to cover a wider range of locations, enhancing the application's practical usefulness. However, this would require gathering and curating additional high-quality, balanced data to maintain or improve classification accuracy. Additionally, to improve accuracy for visually similar buildings, such as landmarks with subtle architectural differences, further dataset refinement and incorporation of more advanced feature extraction techniques could be beneficial.

Finally, while the present findings show promise, there is room for future improvement by exploring more sophisticated model architectures. Expanding the dataset and adopting techniques to better distinguish visually similar landmarks will strengthen the system's stability and accuracy. Overall, this project establishes a strong groundwork for a practical and scalable landmark recognition application, with clear directions for continued development and refinement.

Bibliography

[ACA+21] Andre Araujo, Bingyi Cao, Cam Askew, Jack Sim, Maggie, Tobias Weyand, and Will Cukierski. Google landmark recognition 2021. https://kaggle.com/competitions/landmark-recognition-2021, 2021. Kaggle Competition.

[AGT+16] Relja Arandjelović, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5297–5307, 2016.

[BTVG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*, pages 404–417. Springer, 2006.

[CAS20] Bingyi Cao, André Araujo, and Jack Sim. Unifying deep local and global features for image search. In *European Conference on Computer Vision (ECCV)*. Springer, 2020.

[Cha25] List Challenges. Top 250 famous attractions in the world, 2025. Accessed: 2025-04-05.

[connd] Wikipedia contributors. Random sample consensus. https://en.wikipedia.org/wiki/Random_sample_consensus, n.d. Accessed: 2025-05-14.

[Deva] Google Developers. Accuracy, precision, and recall. *Google Machine Learning Crash Course*.

[Devb] Google Developers. Classification: Roc and auc. *Google Machine Learning Crash Course*.

[Don24] Edwin Dong. Modern convolutional neural network architectures. https://www.aibutsimple.com/p/modern-convolutional-neural-network-architectures, 2024. Accessed: 2025-05-25.

[Goo24] Google Cloud. Cloud Vision API Documentation, 2024. Accessed: 2025-05-15.

[Hen21] Christof Henkel. Efficient large-scale image retrieval with deep feature orthogonality and hybrid-swin-transformers, 2021.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint*, 2015.

[IBM20a] IBM. What is a support vector machine (svm)?, 2020. Accessed: 2025-05-14.

[IBM20b] IBM. What is the k-nearest neighbors (knn) algorithm?, 2020. Accessed: 2025-05-14.

[IBM21] IBM. Supervised vs unsupervised learning. *IBM Article*, 2021. Accessed: 2025-04-06.

[JXZ22] Yuanyuan Jiang, Jinyang Xie, and Dong Zhang. An adaptive offset activation function for cnn image classification tasks. *Electronics*, 11(22), 2022.

[KAT21] Ibrahim Karabayir, Oguz Akbilgic, and Nihat Tas. A novel learning algorithm to optimize deep neural networks: Evolved gradient direction optimizer (evgo). *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):685–694, 2021.

[Low04] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[MAW23] Faisal Mehmood, Shabir Ahmad, and Taeg Keun Whangbo. An efficient optimization technique for training deep neural networks. *Mathematics*, 11(6), 2023.

[ON15] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2564–2571. IEEE, 2011.

[RTC21] Filip Radenović, Giorgos Tolias, and Ondřej Chum. Google landmarks dataset v2 - a large-scale benchmark for instance-level recognition and retrieval. *arXiv preprint arXiv:2004.01804*, 2021.

[Rya22] Ilya Ryabov. Pictures of famous places. `https://www.kaggle.com/datasets/ilyaryabov/pictures-of-famous-places/data`, 2022. Accessed: 2025-04-05.

[SHZ+18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018.

[TKM18] Chakkrit Termritthikun, Surachet Kanprachar, and Paisarn Muneesawang. Nu-litenet: Mobile landmark recognition using convolutional neural networks, 2018.

[TL19] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 6105–6114, 2019.

[WKP16] Tobias Weyand, Ilya Kostrikov, and James Philbin. Planet: Photo geolocation with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1971–1979, 2016.

[XWL+21] Cheng Xu, Weimin Wang, Shuai Liu, Yong Wang, Yuxiang Tang, Tianling Bian, Yanyu Yan, Qi She, and Cheng Yang. 3rd place solution to google landmark recognition competition 2021. *CoRR*, abs/2110.02794, 2021.

[Yat22] Vishal Yathish. Loss functions and their use in neural networks, 2022. Accessed: 2025-04-07.

[ZZ22] Qiuyu Zhu and Xuewen Zu. Fully convolutional neural network structure and its loss function for image classification. *IEEE Access*, 10:35541–35549, 2022.