Urban Buildings Classification Using CNN-based Hierarchical Models

A thesis proposal presented for the degree of
Master of Science in Mathematics and Its Applications

Salma Taoufiq

Under the supervision of:

Csaba Benedek
&
Balázs Nagy

Mathematics and Its Applications
Central European University
Budapest, Hungary
February 2020

Declaration of Authorship

This master thesis is an account of research undertaken between December 2019 and May 2020 for the partial fulfillment of the degree of Master of Science in Mathematics and its Applications at Central European University in Budapest, Hungary. I hereby certify that except where otherwise indicated and acknowledged, this thesis is my own original work. Moreover, I declare that this work has not been submitted for a degree in any university.

Salma Taoufiq
May, 2020

Abstract

Urban building characterization is a complex problem with many involved parts whose solution can benefit the development of smart autonomous driving systems, the digital archiving of cultural artifacts, as well as the automation of real estate valuation. To contribute to this research area, this work focuses on a specific part of the problem: the classification of urban buildings from photographs of their facades into 10 categories: Church, Mosque, Synagogue, Buddhist Temple, House, Apartment Building, Mall, Store, Restaurant, and Office Building. For this purpose, a novel hierarchical multi-label CNN-based model is proposed. Based on the coarse-to-fine paradigm, a label tree is set up, and the model provides outputs corresponding to each level of the resulting hierarchy. Feedback from the coarser level to the finer one runs through the model using simple probabilistic notions encoded through a multiplicative layer connecting the parent coarse branch of the model to the child fine branch. The resulting model solves the urban building classification task while performing better than both a classical convolutional neural network, as well as an existing hierarchical model known as Branch Convolutional Neural Network (B-CNN), while using less parameters than its B-CNN counterpart.

Acknowledgments

Special thanks to my supervisors Dr. Csaba Benedek and Balázs Nagy from the Institute for Computer Science and Control (SZTAKI). It has been delightful working with them. I am grateful for their guidance, valuable insights, and encouragement all throughout the research process. They have helped shape this thesis in immeasurable ways.

I would also like to express my sincere gratitude to my family for their unwavering care and support throughout my life. Their love and confidence in me has pushed me farther than I ever thought I could go.

Finally, my heartfelt thanks to the Department of Mathematics at Central European University (CEU) for their continual support, and to my friends for their encouragements and the joy they have been bringing me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

With the surge in automated driving systems, the interest in the preservation of national cultural heritage, and the automation of the valuation of real estate objects, there is a concurrent growing demand for powerful computer vision algorithms to be applied to urban environment understanding. A natural first step in this endeavor is the classification of urban buildings in terms of their architectural style, purpose, state, etc. This can signal the main function of a certain building to a self-driving car, enrich digital visual cultural archives, as well as provide more information about the buildings to help in their automated price valuation along with the assessment of their states. Consequently, the vision behind this research is to provide one such algorithm that can readily determine the functional purpose of a building given a photograph of its facade.

Convolutional Neural Networks (CNNs) have fueled major leaps in computer vision applications in general, and in image classification in particular. Traditional CNN image classifiers are built with sequential convolutional layers and provide a single output. This design decision is rooted in the assumption that given a classification task, the categories to be predicted should be analyzed and treated equally. However, this assumption is challenged by many real-world classification tasks since some classes can be more difficult to differentiate than others. In the context of urban building classification for instance, a mall is more easily discerned from a house, than from a big department store. Such tasks can benefit from using a coarse-to-fine ordering of classes, thus turning the flat classification problem into a hierarchical one. Department stores and malls are *commercial* buildings, while a house is a *residential* one, and it is arguably easier to distinguish between commercial buildings and residential ones. In fact, the finer class predictions may benefit from this prior separation. A traditional flat CNN is unequipped to deal with such problems. In this research work, a novel CNN-based hierarchical model is proposed and applied to the new urban building functional classification problem. This method is then compared to a well-established model: the Branch Convolutional Neural Network proposed by Zhu et al. [31].

Before attempting to build the hierarchical models to solve this task, an extensive and varied labeled dataset is required. Thus, part of this work revolves around building a thorough and large dataset. Overall, the aim of this project is to achieve a multi-label hierarchical classification to establish the function of the building more accurately.

1.2 Contributions

To summarize the contributions of this research, it:

1. provides a large-scale extensive dataset of a myriad of buildings in terms of their main purpose;
2. offers two approaches to the multi-label hierarchical classification of buildings' functional purposes;
3. evaluates these techniques both quantitatively and qualitatively on the above-mentioned unique dataset;
4. provides a reliable tool for the digital archiving of urban and cultural artifacts in terms of their functions.

1.3 Thesis Outline

The rest of this thesis is organized as follows. Related Work where an appreciation of related and relevant literature regarding the classification of urban buildings and hierarchical deep networks is given in Chapter 2. Chapter 3 presents background theory and explanations to set the stage for the upcoming analysis and research work. Chapter 4 analyzes the methodology starting with the extensive dataset built for this project, as well as the challenges faced while building it, followed by a detailed explanation of the implemented methods for the classification task. Experimental results and their discussions are presented in Chapter 5. Finally, the work culminates in a conclusion and future work for further explorations of this area of research.

Chapter 2

Literature Review

The analysis of buildings has been an active area of research in computer vision in recent years. The majority of these projects have, however, focused on satellite and aerial imagery [1]. Indeed, this is useful when it comes to the detection of building footprints [2], and the detection and segmentation of land covers [18]. These projects are not concerned with the investigation of buildings' facades.

There are also some methods that are mainly concerned with the retrieval of photographs of buildings. Indeed, many popular applications focus on finding building-related images that show the same building from different distances and perspectives. For instance, Y.-I. Pyo et al. present a building retrieval method based on the combination of color and line features to find images regardless of rotation and illumination change [19]. As for X. Yuan and C.-T. Li, they parameterize buildings using an edge-based approach based on the Hough transform. Their image retrieval method proved to be effective in the detection of buildings with strong linear edges [29]. These methods focus on retrieving building images from a database given a query image, rather than on the analysis of the buildings themselves.

In spite of image classification being a popular and active research area in computer vision, not much work has been reported in relation to the classification of functional purposes of buildings from digital images. Research in building classification has mostly been concerned with their architectural styles. Architecture combines artistic sensibility with practical scientific factors to design and create buildings. As such, architectural styles evolve with time, capturing and reflecting the progress of aesthetic trends, and the social, technological, cultural, historical, as well as socioeconomic circumstances of those who made them. Thus, the analysis of the architectural styles of buildings can be informative, and some research has been conducted in this direction. Shalunts et al. propose an approach that relies on clustering and learning local features along with integrating the knowledge used by architects to classify windows of different architecture styles, at the level of the training stage [22]. The same authors also classified building domes into 3 architecture types: Romanesque, Gothic, and Baroque in [23]. They have promising results, but the dataset used was quite small -only a few hundred cropped images which focused on specific elements (facade windows or domes). In contrast, the present thesis investigates building's functions classification rather than their styles, and the dataset used here includes several thousands images.

A more recent paper proposes a model of classifying the architectural styles of historical Mexican buildings across three categories: Prehispanic, Colonial, and Modern [17]. Their

method uses a deep convolutional neural network whose input is composed of sparse features in conjunction with primary color pixel values to increase its accuracy. Their results are good, and have shown that the inclusion of sparse features has definitely improved the accuracy of the network. For my project, a more challenging task is faced as this work's functional purpose classification of buildings is not restricted to a specific geographical area, but rather uses buildings in the dataset from all around the world, calling for more diversity to be accounted for. Only about 284 images from the Mexican buildings dataset are made available through the MexCulture142 set. A few of these images have also been included in this work's dataset.

Regarding the classification of buildings based on their functional purpose, Li et al. proposed a method to predict the type of building (residential vs. non-residential building) from Google Street View images based on Histograms of Oriented Gradients, Scale Invariant Feature Transform (SIFT) and Fisher Vectors [13]. In the framework of this research, the objective is to determine the function of a building using a hierarchical CNN-based model. Instead of focusing on a binary classification (residential vs. non-residential), our aim is to make a more detailed classification across 10 different classes.

In their paper [28], Yan et al. introduce Hierarchical Deep CNNs (HD-CNNs) by incorporating deep convolutional neural networks into a hierarchy of categories. Given a classification task, it is decomposed into two stages following the coarse-to-fine classification paradigm. First, a coarse category classifier is used to separate easy classes from each other, then more difficult classes are distinguished using a fine category classifier. HD-CNNs present a great contribution in that they represent the first method embedding a hierarchical class structure with CNNs to obtain better results. This method can achieve lower error compared to its building block CNN alone at the cost of an affordable increase in complexity. However, the training strategy on which it relies can be time-consuming since two steps are required: first, the coarse and fine classifiers need to be pre-trained, and second, they need to be fine-tuned. Additionally, the HD-CNN has a limited scalability as it cannot be used to classify across more than two levels of hierarchy. Although only two hierarchical levels are used at the current stage of this research project, for future work, it would be interesting to add more hierarchical levels to account for buildings' architectural styles for a more comprehensive classification.

A more recent hierarchical CNN model called Branch CNN (B-CNN) was introduced in 2017 by Zhu and Bain [31]. The authors presented a scalable method in which multiple branch classifiers share the same main convolution workflow. Each branch makes a coarse prediction following a novel and effective training strategy. As part of the present work, the B-CNN model is implemented to solve the task at-hand. Since the B-CNN's branches learn autonomously, this research work goes on to design a new model that explicitly accounts for the coarse predictions in the subsequent finer branch in order to use this prior knowledge as guidance. To build this new model, attempts to implement custom losses were made, finally culminating in a novel method with a custom layer that relies on notions of conditional probability to improve upon the classification performance.

Chapter 3

Convolutional Neural Networks

Convolutional neural networks (CNNs) constitute an important breakthrough in the field of computer vision. They are the fruit of numerous advancements starting with the perceptron algorithm in 1958 [20]. Since they are the building blocks of this project, it is worth diving into the functionality of neural networks and how CNNs work. In the following, an overview of computer vision and deep learning, the perceptron algorithm, artificial neural networks, and the basics of CNNs are explained.

3.1 Computer Vision: A Brief Overview

Computer Vision (CV) sits at the intersection of a variety of disciplines including machine learning, probability, and image processing. It is concerned with automatically extracting, analyzing, and understanding useful information from images. The goal is to achieve automatic visual comprehension through the development of a theoretical and algorithmic basis. To a computer, an image is essentially a numerical matrix of pixel values and hence the objective of CV algorithms is to bridge the gap between the pixel values and the meaning they convey.

For the scope of this project, focus is on image classification which is the task of assigning one or several labels to an input image from a set of fixed categories [9]. For a computer algorithm the recognition of a visual concept (e.g. house, mall, church...) from an image is not as straightforward as it might be for a human. In fact, there are numerous challenges that the algorithm needs to overcome [9]. A few examples of these challenges:

- the object of interest may be occluded by other objects of the image (see (a) in Fig. 3.1 where the house is occluded by trees),
- the background may be cluttered (see (c) and (f) in Fig. 3.1),
- the object may take varying shapes and positions from one image to another (see (b) in Fig. 3.1),
- the object's orientation and size can greatly vary depending on camera angles and distance (see (a) and (e) in Fig. 3.1),
- the lighting and illumination conditions can have a significant impact on pixel values (see (d) in Fig. 3.1).

A good and robust classification model must hence be invariant to the cross product of these possible challenges, while simultaneously retaining sensitivity to the inter-class variations [9]. For instance, the algorithm should be able to recognize that the images of Fig. 3.1 all represent houses.

(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure 3.1: Examples of variations in images representing the same class: House

A specific caste of models referred to as Artificial Neural Networks (ANNs) has fueled spectacular leaps in several areas including computer vision. In an endeavor to understand how the brain is able to produce extremely complex patterns through interconnected cells known as neurons, McCulloch and Pitts developed a model back in 1943 [16]. This model contributed to the development of ANNs beginning with the perceptron algortihm, thus igniting a series of advancements in machine learning. This progress has specifically benefited from the increasing availability of large-scale, high-quality labeled datasets, as well as from the transition from CPUs to GPUs which significantly accelerate deep models' training.

3.2 Artificial Neural Networks

3.2.1 The Perceptron

Observations from biological research of the brain and nervous systems provide important insights to the field of artificial intelligence. Neurons constitute the fundamental computational units of the brain and are connected together to form an immense network which we have yet to fully explore and comprehend. Consequently, when developing ANNs, the objective is not to simulate the inner structure of the brain, but rather to find a good trade-off between the size of the ANN and its performance. To begin understanding

ANN models, it is useful to draw a simple parallelism between a biological neuron and an artificial one.

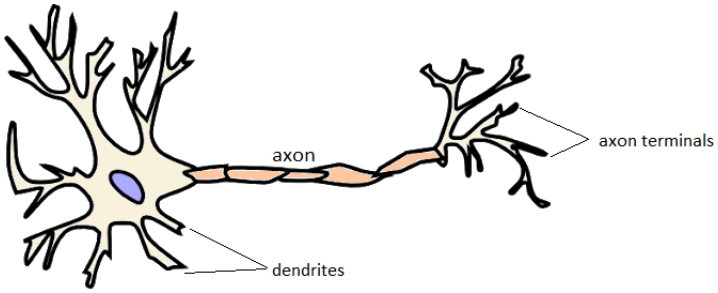


Figure 3.2: Simplified drawing of a biological neuron

An artificial neuron gets n inputs which may be represented as a vector $\mathbf{x} \in \mathbb{R}^n$. The inputs of an artificial neuron can be considered as corresponding to the biological neuron's dendrites, and its output to a biological axon as depicted in Fig. 3.2. The axon eventually branches out and connects to the dendrites of other neurons via synapses [10]. Fig. 3.3 shows a computational model of an artificial neuron. Every input x_i, with $1 \leq i \leq n$, interacts multiplicatively with the neuron by getting assigned a weight $w_1, \ldots, w_n$ respectively. The now weighted inputs all get summed, and a bias b is added. The sum is then run through an activation function f to finally produce an output.

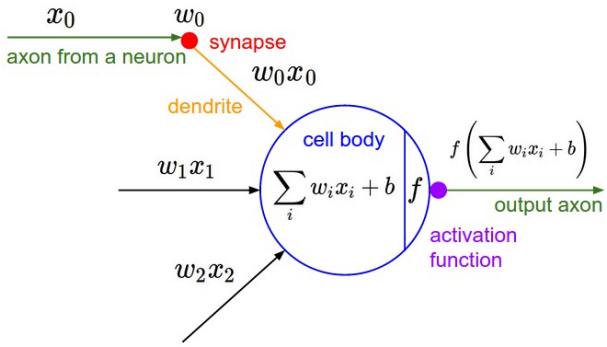


Figure 3.3: Computational model of a neuron: the perceptron, from [10]

This artificial neuron is actually known as the perceptron which was designed in 1958 by Rosenblatt [20]. In essence, the perceptron is a linear classifier defined by weights w_i, a bias b, along with an activation function f according to (3.1).

$$f(\sum_i w_i x_i + b) \tag{3.1}$$

The original perceptron uses the Heaviside step function as its activation function f thus giving a binary output and making the perceptron a linear classifier. Its weights define a hyperplane representing a linear decision boundary between classes. Consequently, an obvious limitation of the perceptron is its inability to learn non-linearly separable patterns.

3.2.2 Neural Networks

In an endeavor to overcome the limitations of a single perceptron, ANNs were developed to provide higher representational capabilities. A neural network is a collection of artificial neurons or units (perceptrons) connected to form an acyclic graph. One of the most common architectures of ANNs is the multi-layer perceptron (MLP). In this model, the neurons are arranged into $l \geq 2$ layers. The set of neurons making up the model is split into mutually disjoint subsets called layers $L_1, \ldots, L_l$:

$$\forall i, j \; 1 \leq i, j \leq l \; (L_i \neq \emptyset \wedge L_i \cap L_j \neq \emptyset) \Longrightarrow i = j$$

The layers are stacked onto one another, L_1 being the input layer, L_l the output layer, and the layers $L_2, \ldots, L_{l-1}$ in between are referred to as hidden. Hidden layers can be added to increase a network's size and complexity. The number of hidden layers that a network has is referred to as its *depth*. A simple example of a multi-layer perceptron topology is given in Fig. 3.4. Typically, the layers are fully connected, meaning that every neuron is connected to each neuron in the previous and next layers while working independently from the other neurons within the same layer it belongs to.

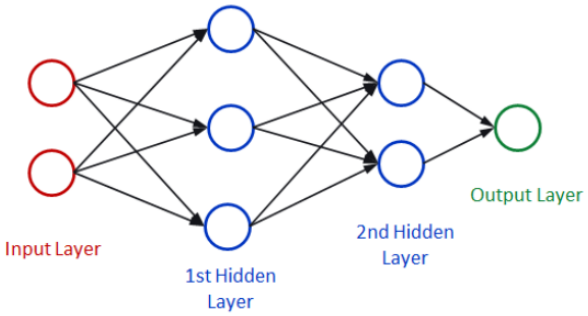


Figure 3.4: A neural network with 2 hidden layers

Building upon (3.1), it is possible to formally define a neural network. Consider the following notation:

- d: dimension of the input layer
- l: the number of hidden layers,
- N_k: the number of neurons in the layer k^{th} layer, $k = 1, ..., l$
- $f : \mathbb{R} \to \mathbb{R}$ the activation function
- $(W_k), (b_k)$ are the network parameters: the weights and biases respectively
- $w^k_{i,j} \in \mathbb{R}$ is the weight between the i^{th} node in the $(k-1)^{th}$ layer to the j^{th} node in the k^{th} layer, so

- $W_k := \begin{bmatrix} w^k_{1,1} & \dots & w^k_{1,N_{k-1}} \\ \vdots & \ddots & \vdots \\ w^k_{N_k,1} & \dots & w^k_{N_k,N_{k-1}} \end{bmatrix}$ represents the matrix of weights connecting the k^{th} and $(k-1)^{th}$ layers
- $S_k : \mathbb{R}^{N_{k-1}} \to \mathbb{R}^{N_k}$, with $1 \leq k \leq l$, $x \longrightarrow W_k x + b_k$

Definition 3.2.1. A map: $\gamma : \mathbb{R}^d \to \mathbb{R}^l$ given by

$$\gamma(x) = S_l f(S_{l-1} f(...f(S_1(x)))),\ x \in \mathbb{R}^d$$

is called a neural network.

Neural Networks as Universal Approximators

The power of ANNs lies in their universal approximation ability. The universal approximation theorem states that a standard feed-forward network with a single hidden layer containing a finite number of hidden neurons, under certain assumptions regarding the activation function f, can approximate continuous functions on compact subsets of $\mathbb{R}^n$ [6]. A first version of this theorem was proven back in 1989 by George Cybenko for sigmoid activation functions [3]. In 1991, Kurt Hornik proved that it is actually the architecture of the feed-forward multi-layer network rather than the specific choice of activation function that determines its expressive power and universal approximation potential [7].

The initial study of feed-forward networks with only one hidden layer as universal approximators entails that the width of these networks needs to be exponentially large to enhance their representation capability. A new advancement with regards to the universal approximation theorem has shown that it is also valid for deep neural networks with a limited width in 2017 [15]. The width a network is defined to be the maximal number of nodes in a layer. In their work, Lu et al. proved that width-(n + 4) networks with ReLu activations are able to approximate any Lebesgue integrable function $f : \mathbb{R}^n \to \mathbb{R}$ with respect to the L^1 distance if network depth is allowed to grow [15].

Hence, it is possible to reformulate the problem that this work tries to solve as an endeavor to approximate the classification function mapping the buildings' photographs to the 10 categories of functional purposes using a neural network with ReLu activations. In order to get a good approximate, the neural network has to find the optimal parameters (and structure)[1] to solve the given classification task. This is achieved through a training process which is a key concept when discussing ANNs.

The Training Process

Prior to starting the training process, network parameters must be initialized. Usually, the initial values are chosen randomly, but it is also possible to use some heuristics for

[1]The number of neurons in each layer and the number of layers in the network are to be chosen. They are hyperparameters that lead to approximating functions of varying complexity. Consequently, choosing a good architecture can be challenging since having too many neurons and layers can produce an excessively complex function that overfits the training data, while using too few neurons may result in a biased overly simplified function that is unable to represent the data distribution well enough.

the initialization as that may speed up the adjustment of parameters towards the desired optimal values. The training process is then carried out by feeding the training data through the network. This procedure is an iterative process. A forward pass through the network results in the production of predictions $\hat{y}_i$ for each input x_i from the training set. The predicted labels are then compared to their corresponding ground truth labels y_i to assess how the model is performing using a loss function $loss(\hat{y}_i, y_i)$. The network can then be adujsted to provide better results through a backward pass. The ANN is finally considered to be trained once a target performance is reached on the training data. Various metrics can be used to assess model performance. In this work, cross entropy loss (also referred to as logarithmic loss) is used. Cross entropy is defined later in Eq. (3.2).

In this thesis, focus is on learning from a labeled dataset, described in Chapter 4. Such dataset consists of input images (photographs of building facades) for the ANN along with their corresponding labels, i.e. expected network outputs. This is known as supervised learning: the process of learning from a labeled dataset $\{(x_i, y_i) : 1 \leq i \leq N\}$ with N the number of individual training samples. One common problem that may be encountered in this context is overfitting: when a neural network model learns the peculiarities of the training data, rather than information that can be generalized to the task being solved. Overfitting is observed when a network's predictive performance is improving on the training set, but worsening on previously unseen test data. One of the techniques used to combat overfitting is to split the original dataset into a training and a validation or testing set. This way, the error rates can be monitored. Usually 20% of the dataset is held out for testing.

As explained earlier, training an ANN relies on repeatedly estimating the loss which represents the error for the current state of the ANN model. It captures how "bad" the network performs in order to subsequently update its parameters to reduce the loss on the next evaluation. For a multi-class classification task, such as the one to be solved in this project, the process may be formulated as follows:

Given training data: $\{(x_i, y_i) : 1 \leq i \leq N\}$ where x_i are the inputs which correspond to the images of dimension d in our case, and y_i are the true labels we are trying to predict and they correspond to one of n_c classes. We want to find $\hat{y}_i$ — the network's prediction for x_i — that minimizes a cost function L:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} loss(\hat{y}_i, y_i)$$

representing the averaged loss over the complete training set. We use θ to refer the network's parameters indiscriminately (both the weights W and the biases b at once). Essentially, the goal is to find the optimal parameters by solving:

$$\hat{\theta} = \arg\min_{\theta} L(\theta) \tag{3.2}$$

The elements $\hat{y}_i$ are obtained from the network's output layer. They represent the normalized probability assigned to the correct label y_i given the image x_i with the network being parameterized by θ:

$$P(y_i \mid x_i; \theta) = \frac{e^{s_i}}{\sum_c^{n_c} e^{s_c}} \tag{3.3}$$

with s the weighted sum of the output neuron's external inputs plus the bias.

In fact, this is called the *softmax* function which gives the model's predicted class scores, and is used as the activation of the network's last layer. As given in [4], we define:

Definition 3.2.2. With $\mathbf{z} = (z_1, \ldots, z_n) \in \mathbb{R}^n$, the standard softmax function $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ is defined as $\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$ for $i = 1, \ldots, n$.

It applies the standard exponential function to each element of a vector of arbitrary real values and ensures that the resulting vector's component are in the range $(0, 1)$ and sum to unity through division by the sum of these exponentials. This way, the softmax function provides a way of predicting a discrete probability distribution over the task's classes. It turns logits (the numeric output of the previous layer of the classification neural network) into probabilities. The softmax function's use of exponentials ensures positivity, since the logits may take negative values and adding their raw values may lead to negative or zero results. Softmax is actually the standard choice of activation function for the output layer of multiple-class classification tasks.

Since for each training example (x_i, y_i), the objective is the maximization of the probability of the correct class y_i, then the negative log probability of that class needs to be minimized:

$$-\log(P(y_i \mid x_i); \theta) = -\log(\frac{\exp(s_i)}{\sum_{c=1}^{n_c} \exp(s_c)})$$

This leads us to *cross-entropy* which is the standard loss function used for a multi-class classification problem. For two given probability distributions, cross-entropy measures the difference between them [25]. Assume a ground truth probability distribution that is 1 at the correct class and 0 otherwise: $p = [0, \ldots, 0, 1, 0, \ldots, 0]$, and q is the computed probability, then the cross entropy is:

$$H(p, q) = -\sum_{c=1}^{n_c} p(c) \log q(c) \tag{3.4}$$

Cross-entropy loss (or log loss) heavily penalizes predictions that are wrong and confident: the cost incurred gets higher and higher the more the predicted probability diverges (wrong prediction) from the ground truth label [25].

Hence, within our context and over the entire dataset $\{x_i, y_i\}_{i=1}^{N}$, the cross entropy loss is used as follows:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} -y_i \cdot \log(\sigma(s_i))$$

Considering that y_i is one-hot encoded, we are left with only one term: the negative log probability of the correct class.

Gradient Descent

In order to find the optimal model parameters to minimize the cross-entropy loss according to Eq. (3.2), we rely on the *gradient descent* algorithm.

Definition 3.2.3. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function. The *g*radient of f at a point $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ is a vector in $\mathbb{R}^n$ of the form: $\nabla f(x) := (\frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_n}(x))$

A well-established and useful result regarding gradients is that given a point $x \in \mathbb{R}^n$ at which f is differentiable, the gradient of the function at that point indicates the direction of steepest ascent. Therefore, finding the direction of steepest descent is as simple as moving in the opposite direction, i.e. following the vector $-\nabla f(x)$. Since the objective of the gradient descent algorithm is to find the minimum value of a function f, the algorithm starts at a given $x_0 \in \mathbb{R}^n$, calculates the value of the gradient at that point $\nabla f(x_0)$, then simply gets to a new point $x_1 := x_0 - \eta \nabla f(x_0)$, with $\eta > 0$ a constant called the *learning rate* representing the step size of the gradient descent search. This process is repeated creating a sequence $\{x_i\}$ defined by the initial choice of the starting point x_0, the learning rate η, and the update rule $x_{i+1} := x_i - \eta \nabla f(x_i)$ for any $i \in \mathbb{N}$. Gradient descent continues building upon this sequence until it approaches a region close to the desired minimum.

If the loss function is convex, gradient descent is guaranteed to find the minimum. In the case of NNs, the loss function is non-convex making it more challenging to reach the global minimum as the algorithm may get stuck in one of the local minima. Additionally, the performance of this algorithm depends on the location of the initial x_0, as well as the choice of the learning rate η. For instance, when the learning rate is too large, it can result in drastic updates that lead to divergent behaviors. In contrast, a learning rate that is too small may lead to an extremely slow convergence. In practice, this problem is solved by starting at a specific value of η and setting up a schedule for shrinking its value gradually. In spite of all these possible challenges, the gradient descent strategy has proven to be successful in a number of real life applications and is the standard algorithm used for training ANNs.

In the context of a feed-forward neural network's learning process, to minimize the loss function, its gradient is calculated with respect to the network's weights and biases. Then, these parameters are adjusted according to:

$$\theta := \theta - \eta \nabla L(\theta) \tag{3.5}$$

There are three variants of gradient descent which differ in the amount of data used for the computation of the gradient of the loss function. Depending on the amount of data, a trade-off is made between the accuracy of the parameter update and the time it takes to perform the update [21].

- Stochastic Gradient Descent: uses only a single training example to calculate the gradient and update the parameters.
- Batch Gradient Descent: calculates the gradients for the whole dataset and performs just one update at each iteration.
- Mini-batch Gradient Descent: works like stochastic gradient descent but uses a mini-batch of samples instead of just one. This variation is one of the most popular optimization algorithms.

Backpropagation

In ANNs, applying gradient descent is quite challenging as it requires the calculations of the partial derivatives of the loss function with respect to every single weight and bias: namely every $\frac{\partial L}{\partial w_{i,j}^k}$ and $\frac{\partial L}{\partial b_j^k}$. It needs to deal with high-dimensional compound functions. Fortunately, the backpropagation algorithm (also referred to as "backprop" for short) provides a method of computing these partial derivatives. Backpropagation makes use of repeated applications of the multivariable chain rule. The algorithm calculates the loss function's partial derivatives starting from the last layer. Using these results, it goes "backwards" inductively through the neural network, computing the partial derivatives at each layer until it reaches the model's first layer. The name of the algorithm refers exactly to this backward progression through the network.

To illustrate the calculations involved in this algorithm, consider a multi-class classification problem and the simple ANN in Fig. 3.5 with only one hidden layer and a softmax output layer:

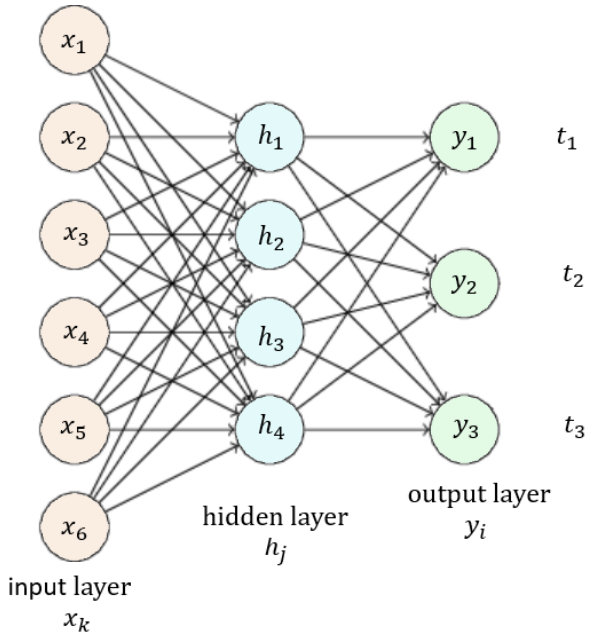


Figure 3.5: Example of a neural network with 1 hidden layer

The following notation is adopted for this example:

- $\mathbf{t}$: target vector of true labels,
- n_c: number of classes,
- x_k: input unit indexed by k,
- h_j: activation of the hidden unit indexed by j,
- y_i: softmax activation of the output unit indexed by i

- $L = -\sum_{i=1}^{n_c} t_i \log(y_i)$ the cross-entropy error function for a single example.

Since the output units use softmax activation, we have:

$$y_i = \frac{e^{s_i^2}}{\sum_c^{n_c} e^{s_c}}$$

with s_i^2 the weighted sum of the hidden layer's activations: $s_i^2 = \sum_{j=1} h_j w_{i,j}^2$. Proceeding with the backprop algorithm, its secret ingredient, the chain rule, is first applied to obtain the derivative of the loss w.r.t each weight connecting the hidden neurons to the output ones:

$$\frac{\partial L}{\partial w_{i,j}^2} = \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial s_i^2} \cdot \frac{\partial s_i^2}{\partial w_{i,j}^2}$$

Computing the gradients yields:

$$\frac{\partial L}{\partial y_i} = \frac{-t_i}{y_i}$$
$$\frac{\partial s_i^2}{\partial w_{i,j}^2} = h_j$$

and

$$\frac{\partial y_i}{\partial s_m^2} = \begin{cases} \frac{e^{s_i}}{\sum_c^{n_c} e^{s_c^2}} - \left(\frac{e^{s_i^2}}{\sum_c^{n_c} e^{s_c^2}}\right)^2, i = m \\ -\frac{e^{s_i^2} e^{s_m^2}}{\left(\sum_c^{n_c} e^{s_c^2}\right)^2}, i \neq m \end{cases}$$
$$= \begin{cases} y_i(1 - y_i), i = m \\ -y_i y_m, i \neq m \end{cases}$$

Therefore, we obtain:

$$\begin{aligned} \frac{\partial L}{\partial s_i^2} &= \sum_m^{n_C} \frac{\partial L}{\partial y_m} \cdot \frac{\partial y_m}{\partial s_i^2} \\ &= \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial s_i^2} - \sum_{i \neq k} \frac{\partial L}{\partial y_m} \cdot \frac{\partial y_m}{\partial s_i^2} \\ &= -t_i(1 - y_i) + \sum_{i \neq m} t_m y_i \\ &= -t_i + y_i \sum_{i \neq m} t_m \\ &= y_i - t_i \end{aligned}$$

To get the gradient of the loss with respect to the weights in the last layer of the network, we compute:

$$\frac{\partial L}{\partial w_{i,j}^2} = (y_i - t_i) h_j$$

In order to trace back to the gradients of the loss with respect to the weights in the network's preceding layer, another application of the chain rule is needed. In the case of

our simple illustrative model, we compute the gradients of the loss with respect to the weights connecting the inputs to the hidden layer units in the following way:

At this stage, it is useful to compute:

$$\frac{\partial L}{\partial s_j^1} = \sum_i^{n_c} \frac{\partial L}{\partial s_i^2} \cdot \frac{\partial s_i^2}{\partial h_j} \cdot \frac{\partial h_j}{\partial s_j^1}$$

where j indexes the hidden neurons, s_j^1 is the weighted input sum at hidden unit j, and h_j is the activation at hidden neuron j.

$$\frac{\partial L}{\partial s_j^1} = \sum_i^{n_c} (y_i - t_i)(w_{i,j}^2)(h_j(1 - h_j))$$

Therefore, the gradient of the loss function with respect to a weight $w_{j,k}^1$ connecting an input neuron k to a hidden unit j is:

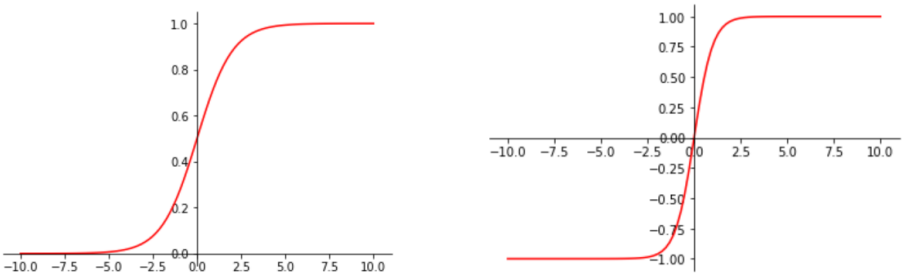
$$\frac{\partial L}{\partial w_{j,k}^1} = \frac{\partial L}{\partial s_j^1} \cdot \frac{\partial s_j^1}{\partial w_{j,k}^1}$$

$$\frac{\partial L}{\partial w_{j,k}^1} = \sum_i^{n_c} (y_i - t_i)(w_{i,j}2)(h_j(1 - h_j))(x_k)$$

All in all, with backpropagation, it is possible to compute the gradient of the error function with respect to all weights in the neural network by recursively computing its gradients with respect to the activity of each neuron.

Activation Functions

In a nutshell, the training process of an ANN is all about learning the optimal weights and biases that lead to a good performance. Along with the loss function, the activation functions are crucial to the backpropagation method. The three most common ones are: sigmoid, hyperbolic tangent, and ReLu. The sigmoid function $\sigma : \mathbb{R} \to]0, 1[$ with $\sigma(x) = \frac{1}{1+e^{-}x}$ has historically been commonly chosen due to its continuity and greater flexibility compared to the unit step function. The hyperbolic tangent function tanh is also used as an activation function as it provides a similar form but has the advantage of being zero-centered. However, both of these functions present the drawback of saturation: when the neuron's activation saturates at either tail of their respective ranges, the gradient at these regions is almost zero [10] as can be inferred from their graphs in Fig. 3.6. The backpropagation process suffers when the neuron's local gradient is very small since it may "kill" the gradient preventing the signal from flowing through the neuron and updating its weights.



(a) Sigmoid **(b)** Tanh

Figure 3.6: Two of the commonly used activation functions

Since 2010, the Rectified Linear Unit (ReLu) has become the standard go-to activation function. ReLu computes the function $f(x) = max(0, x)$, i.e. the activation is simply thresholded at zero as can be seen on Fig. 3.7. One notable advantage of the ReLu activation is that it was found to greatly accelerate the convergence of stochastic gradient descent in comparison to sigmoid and tanh [11] due to its non-saturating form. Unlike tanh and sigmoid functions, ReLu is simpler to implement since it does not use computationally demanding operations such as exponentials [10]. However, with ReLu, there is a potential risk of "dead" neurons. Indeed, ReLu units may become inactive and only output 0 for any given input. By definition, when a ReLu unit gets a negative input x, then $f(x) = 0$ and consequently it will always output 0 and is unlikely to recover. These "dead" neurons will cease to play any role in the learning process. Over time, this may worsen to the point where a large portion of the neural network just dies and does nothing. This problem is referred to as dying ReLu and can be due to using a learning rate η that is too high. As can be seen from Eq. 3.2 if η is too high, the updated weight could turn out to be negative. A large gradient flowing through a ReLu neuron could also cause the weights to update negatively and cause the neuron to never activate on any datapoint again [10]. This results in the gradient flowing through the unit to be zero from then on. ReLu neurons run the risk of irreversibly dying during training as they can get knocked off the data manifold. With a proper setting of the learning rate this is less frequently an issue.

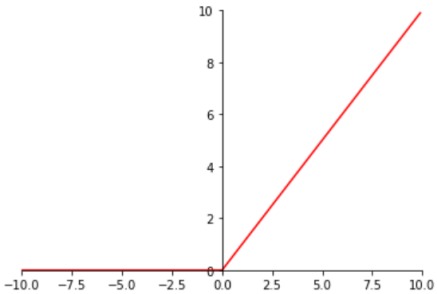


Figure 3.7: ReLu function

3.3 Convolutional Neural Networks

Regular neural networks come with full connectivity of the hidden layers which makes them scale poorly to full images since every single neuron in the first hidden layer would have to connect to every pixel value of the image. For instance, the images used for this project are of size $224 \times 224 \times 3$. The original images gathered in the dataset are much larger and had to be downscaled as the neural network used (VGG-16) accepts inputs of the specific shape $224 \times 224 \times 3$. Even for this downscaled size, the numbers are huge since each neuron in the first hidden layer would have $224 \times 224 \times 3 = 150,528$ parameters. These numbers would add up quickly making the computational and memory requirements wasteful and restrictive. To surpass this limitation, a particular type of NNs can be used: Convolutional Neural Networks (CNNs) which are state-of-the-art in a variety of computer vision applications. These models are constructed with the explicit assumption that the inputs are images. CNNs use layers in which neurons are arranged across three dimensions, namely: height, width, and depth, with each neuron only connecting to a small region of the previous layer. As we will see next, the architecture of a CNN is set up in such a way that the full image gets progressively reduced into a single vector of class scores, arranged along the depth dimension.

To build a CNN, three main types of layers are used: Convolutional, Pooling, and Fully Connected.

3.3.1 Convolutional layers

A linear image filter is an element $F \in \mathbb{R}^{w_f \times h_f \times d}$ where w_f represents the width of the filter, h_f its height, and d is the number of input channels to which the filter is applied. Let $I \in \mathbb{R}^{w_i \times h_i \times d}$ be an image whose width is w_i, its heigth is h_i, and d being the number of channels it has (for instance, if I is an RGB image, then $d = 3$). Convolving the filter F with image I produces an output image I' which only has one depth channel. Every entry $I'(x, y)$ can be obtained through the point-wise multiplication of the elements of F and those of the image I according to:

$$\sum_{i_x=1-\lceil \frac{w_f}{2} \rceil}^{\lfloor \frac{w_f}{2} \rfloor} \sum_{i_y=1-\lceil \frac{h_f}{2} \rceil}^{\lfloor \frac{h_f}{2} \rfloor} \sum_{i_c=1}^{d} I(x + i_x, y + i_y, i_c) \cdot F(i_x, i_y, i_c)$$

This operation is essentially a discrete *convolution*. Fig. 3.8 provides a visualization of this operation:

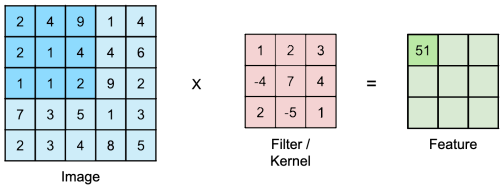


Figure 3.8: Example of a convolution using a filter ($d = 1$)

Filters are used to detect specific patterns, called *features* from the input image. Convolutional neural networks are able to take into consideration the spatial structure of pixels by learning image filters through *convolutional layers* (conv. layers). A layer in a CNN takes *feature maps* (or activation maps) as input. A feature map is a matrix $M \in \mathbb{R}^{w \times h}$ (typically, $w = h$). For instance, if the input is an RGB image, then the number of feature maps is $d = 3$, as each color channel is considered a feature map.

Given an input volume with width w, height h, and depth d, a convolutional layer computes the output of neurons which are connected to small $n \times n$ ($n \in \mathbb{N}$) local regions of the input. Each neuron casts a square matrix F (of size $n \times n$), whose entries are the neuron's weights, to that region of the input and computes the dot product. The matrix F represents the filter, and the region of the input it connects to is the local receptive field. Each filter when applied to the input produces an activation map. These maps are stacked along the depth dimension, hence producing the output volume of the layer which is then passed to the next layer of the network. In this manner, conv. layers are able to apply convolution operations across the input, making the network able to learn to recognize meaningful local features, such as edges or corners at first, and then higher-level features that can help it differentiate between the different classes given a classification task. In practice, a CNN does not just learn a single filter per layer, it actually learns multiple ones in parallel for a given input. This enables specialization within the network, making it able to learn properties that are specific to the dataset on-hand.

Convolutional layers present two game-changing advantages: local connectivity and parameter sharing. As abovementioned, unlike fully connected layers, the units in a conv. layer are organized within a volume. Neurons within the same depth slice have the same parameters (weights and biases) thus enabling parameter sharing. This way, they cast the same filter to different local regions of the image (local connectivity). Hence, a forward pass of a conv. layer across each depth slice is a convolution: each filter slides across the width w and height h of the input while computing dot products between its weights, and the entries of the input at any position. Even though a filter has a spatially small size across the width and height ($n \times n$), it extends through the full depth d. As the filter slides across the input volume according to a certain stride s (the number of pixels by which the filter moves, it usually takes a value of 1 or 2), the resulting responses at every position are stored in a two-dimensional activation map. Each filter also adds an offset bias. During backward passes, each neuron in the volume computes the gradient with respect to its weights. Then, the resulting gradients are summed up across each depth slice of the volume, and only update a single set of weights per slice [8]. This parameter sharing scheme enables conv. layers to control the number of parameters.

The number of filters f in a given layer, the receptive field size n, and the stride s are all hyperparameters that can be fine-tuned to improve a model's performance. As for a conv. layer's parameters, they are precisely the set of learnable filters and biases.

3.3.2 Pooling layers

Another important type of layers in CNNs is *pooling layers*. A pooling layer is usually inserted between successive convolutional layers. Such a layer reduces the number of parameters and the amount of computations in the network.

Pooling summarizes the presence of features in certain regions of the feature map by spatially sub-sampling the input volume independently within each of its depth slices. Similarly to convolutions, pooling is applied in a sliding window fashion, so these layers require two hyperparameters: the length of the side of the square receptive field $k \in \mathbb{N}$ (typically $k \in \{2, 3, 4, 5\}$), and the stride length $s' \in \mathbb{N}_{>1}$ (typically $s' = 2$, as is the case in the VGG-16 model used later). Pooling layers do not introduce any parameters since they only compute a fixed function of the input. Although max pooling is the most common type of pooling, there also exists other types such as average pooling and L^2-norm pooling. Max pooling simply uses the function $max(a \in A)$ for a set A of activations $a \in \mathbb{R}$, thereby selecting the maximum value from a local receptive field. Fig. 3.9 below provides a visual representation of max pooling:

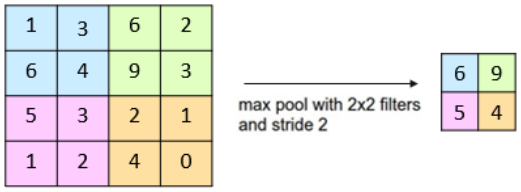


Figure 3.9: 2×2 max pooling applied to a 4×4 feature map with stride $s' = 2$

Pooling is essential to the reduction of data to $\frac{1}{s'^2}$th of the input data which consequently aids in memory cost reduction. Apart from saving memory space, pooling enforces local translational invariance as well as invariance against minor changes. Indeed, since this operation gets rid of exact feature locations and only keeps relative and approximate location information, it makes the overall network less sensitive to shifts and distortions.

3.3.3 Fully Connected layers

As mentioned previously, in fully connected (or dense) layers, every neuron is connected to the neurons of the next layer. Therefore, neurons in a FC layer establish full connections to all of the activations in the preceding layer. FC layers in a CNN work as in a regular artificial neural network that is fed the features extracted by the previous part of the CNN and outputs the label(s) for solving the classification task. These layers represent the classifier part of the CNN while the convolutional and pooling layers below them represent the model's feature extractor.

All in all, like regular neural networks, convolutional neural networks use gradient-based learning to learn the set of parameters that leads it to minimize the loss function and solve the given task. With convolutional and pooling layers, these networks provide a particularly robust model for handling images. In the present thesis, the objective is to improve the performance of the overall model in the urban buildings categorization problem by going beyond the traditional flat CNN classifier. In particular, we investigate how a predefined class hierarchy can be incorporated within the network's structure to build a hierarchical multi-output model.

Chapter 4

Methodology and Experimental Setup

4.1 Methodology

This work is divided into three main steps. First and foremost, an appropriate dataset for the urban buildings categorization task is needed. To my knowledge, the majority of existing datasets which are related to the classification of buildings based on photographs and that are readily available are based on satellite imagery. Nonetheless, two datasets with architectural-style classes and images of building facades were found online: one called MexCulture142, which contains 284 images of Mexican buildings divided into 142 subclasses of: prehispanic, colonial, and modern styles [17], and the other is a 25-class dataset of different architectural styles which contains 4,794 images in total, with 60 to 300 per class [27]. Unfortunately, only about 20% of all these images across the two datasets combined were relevant for use in this project as most images do not fit the scope of this work which focuses on colored photographs of facades of buildings that fulfill specific functions.

Due to the lack of publicly available extensive datasets, building one from scratch is a necessity. It is important to keep in mind that the quality of the training data determines the performance of the machine learning system being developed. The dataset should be representative of the objects belonging to each class and include several images taken from different angles, distances, perspectives, and lighting conditions. The goal is to have a robust model that overcomes the challenges posed earlier in Section 1.1. Additionally, with a varied dataset, the model is better trained for real-life applications where it is expected that the end user will input images of varying quality and points of view. Consequently, building this dataset requires careful examination of the diversity of the images. Web scraping techniques were used to automate the process of image collection from multiple sources, such as the Wikimedia collection and the real-estate website Zillow. Further details about the dataset are given in Section 4.2.

Second, to classify building facades from photographs, this work uses CNNs through transfer learning. Transfer learning is the usage of all or just parts of a model that was already trained on a related task since it is quite uncommon in practice to have a sufficiently large dataset to train an entire network from scratch. By using a network that was already pre-trained on a huge dataset, the learning process is able to start with a better weight initialization and learn robust filters that are generalizable to new application-specific data in a time-saving manner by benefiting from features identical

or similar to the pre-trained ones. Hence, the knowledge obtained from a source task is *transferred* to the task at-hand. Keras, Python's open-source neural-network library, provides a range of such pre-trained models which can be loaded and implemented wholly or partially to ease transfer learning.

In the present work, we rely on the VGG-16 (also dubbed OxfordNet) which was pre-trained on the ImageNet dataset. This dataset is used for the ImageNet Large Scale Visual Recognition Challenge ILSVRC — a benchmark in object category classification and detection — and has over 14 million images across 1,000 categories. The VGG-16 model was developed by and named after the Visual Geometry Group from Oxford. As its name suggests, this network is made up of 16 convolutional layers. This model was proposed in 2014 by Karen Simonyan and Andrew Zisserman from the University of Oxford in their article "Very Deep Convolutional Networks for Large-Scale Image Recognition" [24], and was initially trained for about 2-3 weeks on multiple GPUs. It has achieved a test accuracy of approximately 92.7% in ILSVRC landing it among the best-achieving models in 2014. VGG-16 has a uniform architecture, mostly consisting of small 3x3 convolutions, but with a large number of filters. It is currently one of the computer vision community's preferred choices for feature extraction tasks from images.

Initial experiments have shown that VGG-16 is well-adjusted to this project's preliminary dataset. It is just deep enough to provide satisfactory results without falling in overfitting. Fig. 4.1 below shows the overall architecture of VGG-16:

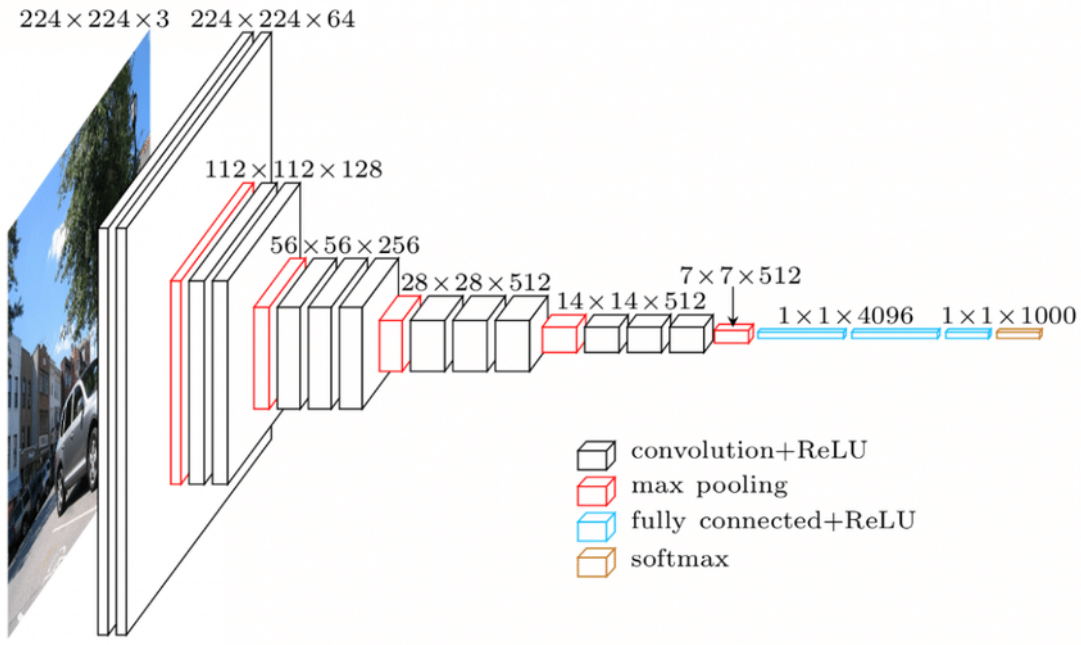


Figure 4.1: VGG-16 architecture. Image taken from [14]

.

The VGG-16 model consists of two main components: the feature extractor that is comprised of the VGG blocks (made of comvolutional and pooling layers), and then the classifier which has the fully connected (FC) layers along with the output layer. The first two FC layers of the classifier have 4096 units each, while the third performs 1000-way

ImageNet classification with 1000 units, one for each class in the ImageNet task. The final layer is the Softmax layer. Consequently, the FC layers need to be replaced from the output-end of the VGG-16 to fit the task of interest. The existing FC layers are removed and new ones are added to interpret the model output and be able to make a prediction consistent with the urban buildings classification problem. As shown in Fig. 4.1, this model takes input of shape (224, 224, 3). The urban buildings dataset images are thus resized accordingly prior to feeding them to the network.

Each image goes through a stack of conv. layers with ReLu activations for feature extraction. The filters are implemented with a small receptive field: 3×3. As for the convolution stride, it is fixed to 1 pixel. The spatial padding of the conv. layers is set in such a way that the spatial resolution is preserved after convolution, i.e. *same* padding is used. Considering pooling, it is carried out by five max-pooling layers, which follow some of the conv. layers. Max-pooling is performed over a 2×2 pixel window, with stride $s = 2$.

The VGG-16 model was chosen for this work as it is smaller, well-established, and well-understood. Additionally, the purpose of this research work is not the find the ultimate best model for the task on-hand, but rather to improve upon a regular flat CNN classifier by introducing a hierarchical structure. Numerous experiments have been conducted in this research work, progressively building up their complexity in the hopes of fully exploring and establishing a better understanding of the urban building hierarchical classification task. Some of the most relevant preliminary findings will be presented later in Section 4.3 in order to set the stage for the problem to be solved and the methods used.

Finally, the bulk of this work is to set up a robust hierarchical model for classifying buildings. Accordingly, this research proposes a new network which is then compared to an existing state-of-the-art solution called Branch Convolutional Neural Network (B-CNN). Both models leverage the inherent hierarchical structure of CNNs' architecture. The B-CNN is a hierarchical model proposed by Zhu and Bain in [31]. It branches out from a building block CNN at different levels corresponding to a predefined class tree, and introduces a loss function that preserves the flow of the network's backpropagation. Our proposed method builds upon observations from the B-CNN model and achieves a better performance on the urban building classification task, benefiting from the coarse-to-fine paradigm to make more informed fine predictions. Each one of these methods uses VGG-16 as its baseline model. They are both analyzed by implementing working prototypes and comparing their performances with regards to the task at-hand.

4.2 Dataset

The dataset, in and of itself, presented the first challenge. It was built throughout different stages as it evolved hand-in-hand with the progress of this research work in order to meet its advancements and answer new questions as they arise.

Consequently, the description of the dataset building process will be presented through different steps, as follows:

1. Through web scraping[1], a preliminary dataset was built with 4 main classes:

   (a) **Residential**: consists of buildings where people live and reside: mainly houses and apartment buildings

   (b) **Commercial**: consists of those buildings with a commercial purpose: grocery stores, retail and department stores, restaurants, cafes, and malls

   (c) **Business**: mainly includes office buildings, and corporate headquarters.

   (d) **Religious**: it consists of buildings with a religious spiritual purpose with a focus on mosques, churches, synagogues, and buddhist temples.

   At this stage, the dataset held about 2,000 images of varying sizes and aspect ratios. Additionally, the pictures gathered have been taken under a multitude of lighting conditions. A few preliminary experiments have made use of this initial dataset in order to better grasp the problem on-hand.

2. From these initial experiments, the classification of buildings' general functional purpose right off the bat lead to good results, albeit improvable ones. The idea to add an extra layer of complexity to the task in hopes of adding information that could lead to better classification results was studied; including the architectural-style component can achieve that. Indeed, certain types of buildings that fulfill a similar purpose share certain architectural features: for instance, many places of worship include elements such as towers, cupolas, and colorful elaborate decorations which are, in contrast, rarely used in office buildings. The 4-class initial dataset was to be further labelled based on additional architectural-style-based categories in order to check whether a multi-label classification based on both function and architecture would lead to better classification results.

   At this stage, the total images annotated with an architectural-style label is a mere 1,018 divided across 6 architectural classes: Gothic, Byzantine, Baroque, Modern, Islamic, and Colonial, which are, in turn, further grouped based on their functional purpose. It is a tedious task to label all images gathered for the 4-class dataset based on their architectural style since a building may showcase multiple architectural styles due to practical reasons, aesthetic preferences, or as a result of renovations throughout the years. It is also not possible to find out the style of unknown buildings such as the residential ones for instance. Additionally, the majority of business and commercial buildings fall within the modern architectural style, leading to potentially extremely unbalanced architecture-based classes.

   Furthermore, our task is unlike that of other projects which focus on a specific geographical location: such as Mexico in [17], and Paris in [12] where the authors find visual patterns corresponding to semantic-level architectural elements distinctive to particular time periods. Such a geographical restriction leads to a clear cut division of classes based on the history of the location studied. In the case of this work, no similar restriction was enforced since the aim is to solve the classification task for

[1] A sample of the code written for web-scraping is provided in: https://github.com/salmatfq/Scraping-Website-For-Images.

buildings from all around the globe. Consequently, it is more challenging to have a succinct all-encompassing division of architecture-based classes.

3. Another approach to get better and finer results would be to attempt a more detailed classification task by actually predicting a more specific function of the building. For this purpose, the initial 4-class dataset was further divided to make a 10-class dataset where the labels are: Mosque - Church - Synagogue - Buddhist Temple - House - Apartment Building - Office Building - Mall - Store [2] - Restaurant [3].

   More images have been collected culminating in a final extensive dataset of approximately 6,300 images. This project's task is thus to classify images of urban buildings across these 10 classes.

4.3 Setting the stage: Preliminary Results

As mentioned previously, this project makes use of transfer learning by using the VGG-16 model as the building block model based on which different methods are implemented. VGG-16 is thereby used with a parameter initialization from the ImageNet task. Preliminary experiments were conducted in order to progressively focus the scope of this research and solve the task of buildings classification based on their photographs. Table 4.1 presents results of two urban buildings classification tasks based on photographs of their facades, solved using a regular VGG-16 network with modified FC layers to fit the task:

- Task A: a 4-class classification across the classes: Business, Residential, Religious, Commercial
- Task B: a more detailed classification with the 10 abovementioned classes: Mosque, Church, House, Office Building, ...

	Task A	Task B
Accuracy	87.34%	78.22%
Loss	0.3774	0.6669

Table 4.1: VGG-16's average performance on tasks A and B

As shown in Table 4.1, it is easier for a regular CNN classifier (VGG-16 in this case) to classify images across broader classes (Task A), compared to more specific ones (Task B). Intuitively, this makes sense as it is easier to discern a church from an office building than from a building fulfilling a similar function like a synagogue for instance. In the context of the urban building classification problem, the objective of is to improve upon the results for Task B. Accordingly, our findings from this initial experiment can be leveraged by adopting the 2-level hierarchy presented in Fig. 4.2:

[2] includes retail stores such as grocery supermarkets, department stores, and clothing brands stores
[3] also includes cafes

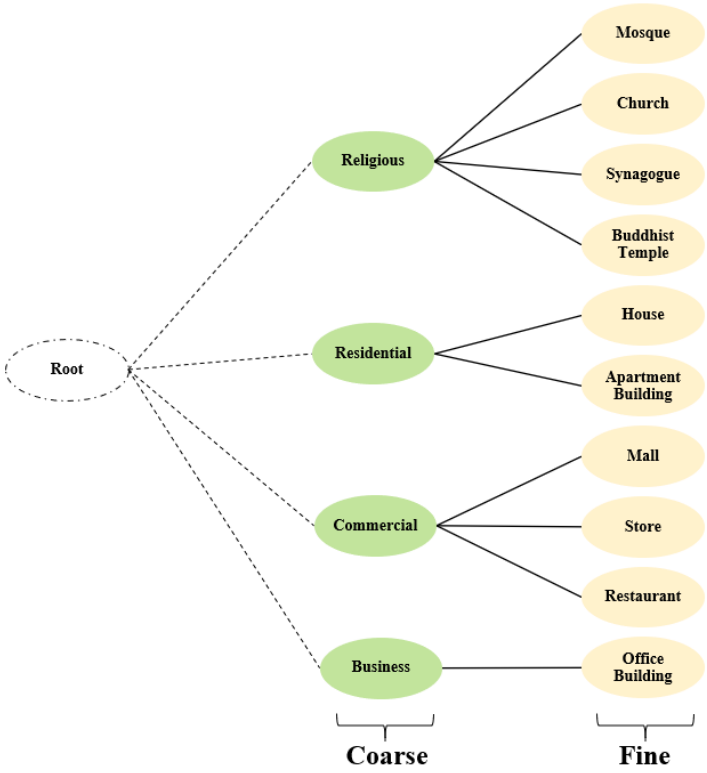


Figure 4.2: Hierarchical class tree for buildings' functions

This way, using the underlying semantic taxonomy (e.g. a house is a residential building), it is possible to combine both tasks in order to benefit the desired final 10-classification results by partitioning the original 10 classes into corresponding superclasses. For future use, we denote by N_f the number of fine classes (subclasses), and N_c the number of coarse classes (superclasses). In the current context: $N_f = 10$ and $N_c = 4$. The hierarchy given in Fig. 4.2 can be conceptualized as a mapping $H : [1, N_f] \to [1, N_c]$. Observe from the class tree structure that every fine class is the child of only one coarse class. The coarse classes do not overlap.

4.4 Branch Convolutional Neural Network

CNNs come with an inherent hierarchical approach to feature extraction; lower layers capture low-level features, while higher layers extract high-level features as explicitly shown by Zeiler and Fergus through visualizations of a CNN's layers during training [30]. This hierarchical nature of features within a conv. network can be leveraged in order to improve the performance of a CNN model. This is precisely the premise of the first method applied in this project, and referred to as Branch Convolutional Neural Network (B-CNN). The B-CNN model is a CNN structure which integrates the prior knowledge of hierarchical relations among classes within the architecture of the model. It was proposed by Xinqi Zhu and Michael Bain in 2017 [31]. The authors have confirmed that this method presents notable benefits over a traditional CNN.

The B-CNN structure relies on the usage of multiple branch classifiers along the main convolution workflow hence allowing it to make predictions hierarchically. This model's backbone is a baseline CNN to which the B-CNN adds internal output branches. Multiple predictions are made, ordered from coarse to fine in a hierarchy corresponding to that of the target classes.

Based on the defined class tree for our task in Fig. 4.2, the model will output two predictions:

1. a coarse one defining the general functional purpose, and
2. a fine one, corresponding to the tree's leaves and representing the task's target classes.

An overview architecture of the B-CNN used for a 2-level hierarchy such as the one in this research work is given below in Fig. 4.3:

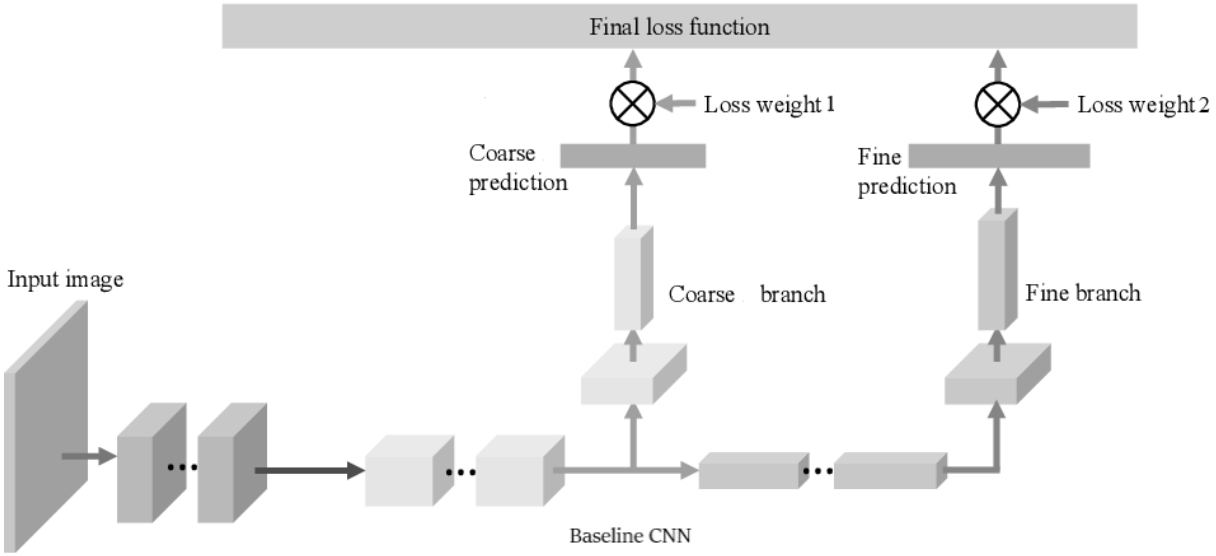


Figure 4.3: BCNN architecture modified and adapted from [31]

As mentioned above, the B-CNN structure relies on existing CNN components as its building blocks. They are represented by the CNN at the bottom in Fig. 4.3. This CNN base can be any arbitrary CNN model; in this project, VGG-16 is used. The middle part shown in the figure represents the output branch networks of the B-CNN. Each branch network produces a prediction on the corresponding level in the label tree. At the top of each branch, fully connected layers and a softmax layer are used to produce the output in one-hot representation. Branch networks can consist of convolutional networks and fully connected neural networks [31]. In our experiments, for the sake of simplicity, only fully connected NNs are used as branch networks.

Zhu and Bain also proposed a training strategy specifically designed for B-CNN models: the *Branch Training strategy* (BT-strategy). This approach aims at activating and

training lower level parameters before the higher level ones, thereby adjusting the parameters on the output layers to minimize the loss. The BT-strategy's objective is to relieve the vanishing gradient problem discussed early on in Chapter 3. By assigning each branch a loss weight, this training technique then forces the model to successively learn coarse to fine concepts by shifting the loss weights of different level outputs as the learning process progresses, ultimately making the B-CNN converge to a traditional CNN classifier [31].

Considering the multi-output structure of the B-CNN, the final loss function is a weighted summation of all losses. It is formulated as [31]:

$$L_i = \sum_{k=1}^{K} -A_k log(\frac{e^{f^k_{y_i}}}{\sum_j e^{f_k}_j}) \tag{4.1}$$

where:

- i denotes the i^{th} sample in the mini-batch (mini-batch gradient descent is the optimization technique used)
- K is the number of levels in the label tree
- A_k is the loss weight corresponding to the k^th level contributing to the loss function
- The term $-log(\frac{e^{f^k_{y_i}}}{\sum_j e^{f_k}_j})$ is the cross entropy loss of the i^{th} sample on the k^{th} label tree level
- f_j denotes the j^{th} element in the vector f of class scores, outputted by the last layer of the model

As can be observed from (4.1), for each sample i the loss is basically a sum of the cross entropy loss of that sample at a given level multiplied by its respective loss weight A_k, for each level k. The loss function takes the losses from all levels into consideration in order to ensure that the structure prior can play a role of internal guide to the whole model and make it easier to flow the gradients back to the shallow layers.

In (4.1), the loss weight A_k of each branch network in the B-CNN structure defines the amount of the contribution a level k makes to the final loss function. Note that:

$$\forall k \; A_k \in [0, 1]$$

$$\sum_k A_k = 1$$

The usage of loss weights makes B-CNNs super models of traditional CNNs. For instance, consider a B-CNN with three branches such as their respective loss weights are fixed to $[0, 0, 1]$, this B-CNN thus converges to a regular CNN model with only the last output branch being trainable. In contrast, with choosing the loss weights to be $[1, 0, 0]$, the B-CNN only activates the shallower part of the entire network up to the first coarse branch, without training the two finer levels. The distribution of the loss weights is also relevant as it indicates the importance of one level relative to the others. Consider a

two-branch B-CNN with loss weights [0.9, 0.1]: this leads the model to value the low level feature extraction of the shallower layers while also training to a much lesser degree deeper layers. In fact, in our experiments, this is the assignment used to initialize the model's loss weights.

The potential of the loss weights distribution is exploited through the BT-strategy by modifying it during the training process of the B-CNN model. For instance, in our experiment with 50 epochs, the initial loss weights are initialized to be $[0.9, 0.1]$, which is changed to $[0.3, 0.7]$ after 15 epochs, and finally to $[0, 1]$ after 25 epochs. The greatest loss weight in each assignment can be viewed as the *focus* [31]: e.g. 0.7 is the focus in $[0.3, 0.7]$. Although the specification of a focus is not necessary as all levels can be equally important, setting a focus explicitly signals to the classifier which level it should learn with more efforts. The BT-strategy relies on shifting the focus of the loss weights distribution from lower to higher levels, i.e. from coarse to fine levels so the classifier extracts lower features first with coarse instructions and fine tune parameters with fine instructions later. Following this procedure helps prevent the vanishing gradient problem which would make the updates to parameters on lower layers more challenging when the network is very deep [31].

The B-CNN model is more informative than a flat CNN, and the error can be restricted to a particular subcategory either the coarse class or the fine class in our 2-level-only case. This will be a good guide for the fine classifier which benefits from the prior knowledge obtained from the coarse branch. In addition, predictions of the B-CNN model can reflect the complexity of semantic labels in the real world much better than a flat CNN.

4.5 Proposed Method

As can be observed from the B-CNN model presentation given previously, the network does not explicitly take the coarse prediction as input for the subsequent layers in order to better guide the fine classifier towards the right direction. For instance, if the coarse branch predicts that the building is residential, then this piece of information can be used as an explicit input for the following layers in order for the fine branch to give higher consideration to the consequent children classes of the predicted coarse label, i.e. "House" and "Apartment Building" in this case. It follows that the coarse prediction should be quite reliable if it is to be used in guiding the fine classifier into making a more informed decision to eventually provide a better solution to the urban building categorization task. For this reason, the coarse branch could benefit from higher-level features as well in order to achieve a better coarse classification performance. Consequently, unlike in the structure proposed for the B-CNN model, in developing our proposed method, the coarse branch is positioned at the bottom of the baseline network, alongside the fine branch. Experimenting with this multi-output structure leads to better results for the coarse component compared to those obtained with the B-CNN, as can be seen from Table 4.2. The multi-output model is obtained by simply shifting the coarse branch to the end of the VGG-16 model, alongside the fine branch. For each image x_i in the dataset, there is a coarse label y_{c_i} and a fine label y_{f_i}.

	Coarse Branch Accuracy	Coarse Branch Loss	Fine Branch Accuracy	Fine Branch Loss
B-CNN	82.35%	0.5071	79.85%	0.6023
Multi-output Model	**90.65%**	**0.2782**	79.87%	0.6060

Table 4.2: Average performance of the B-CNN model vs. the flat multi-output version

Results show that the accuracy of the coarse classifier are indeed higher, and its loss lower. As can also be observed from Table 4.2, the fine classifier's performance is not meaningfully affected by the improvement in the coarse classifier's accuracy. In other words, the fine branch does not benefit from the improved performance of the coarse branch as both its accuracy and loss remain roughly the same.

As could be seen from Fig. 4.3, in the B-CNN model, the coarse and fine branches only share the shallower layers of the network. Then, after branching out to the coarse classifier, the remaining convolutional layers tend to specialize in learning parameters that would help the model discriminate between fine classes. By putting the coarse and fine branches at the same level, at the end of the network, one can assume that all convolutional layers of the baseline network tend to learn parameters that are useful for both tasks (the coarse and fine classifications).

Essentially, recall from Chapter 3 that a flat CNN classifier with its softmax layer gives a value that can be viewed as a probability. In fact, it is a conditional probability of belonging to a certain class given an image parameterized by the model's weights and biases $P(y_i \mid x_i; \theta)$ as previously shown in Eq. 3.3. The multi-output model, hence, computes both probabilities: $P(y_{c_i} \mid x_i; \theta)$ and $P(y_{f_i} \mid x_i; \theta)$ using its softmax layers.

In order to explicitly account for the coarse prediction in the decision of the fine classifier using the hierarchical label tree in Fig. 4.2, consider the probability of outputting a fine class f_j with $j \in \{1, \ldots, N_f\}$ given an image x fed to the network parameterized by θ as the probability of outputting its corresponding parent coarse class $c_i = H(f_j)$ with $i \in \{1, \ldots, N_c\}$ and that out of that class's subclasses, the subclass $sub_{c_i k}$ coinciding with f_j is indeed chosen, with k indexing the subclasses of c_i. Formally, it can be formulated as:

$$P(y_f \mid x; \theta) = P(y_c, y_{sub_c} \mid x; \theta) \tag{4.2}$$

These probabilities coincide since the prediction of the fine class is precisely reliant on the coarse class predicted and which one of its children in the label tree is most likely to correspond to the image x. By reformulating the fine prediction in terms of the coarse label, it is clearer that the network can benefit from the prior knowledge gained from the use of the coarse-to-fine hierarchy. Given (4.2), with simple probability notions, it is possible to express $P(y_f \mid x; \theta)$ in terms of $P(y_c \mid x; \theta)$. Assume A and B are two dependent events. The intersection of A and B is defined by:

$$P(A \cap B) = P(A)P(B \mid A) \quad (*)$$

We can thus obtain the conditional probability:

$$P(B \mid A) = \frac{P(A \cap B)}{P(A)} \text{ with } P(A) > 0 \quad (**)$$

Using $(*)$, $(**)$, and (4.2), we get the following:

$$\begin{aligned} P(y_f \mid x) &\overset{(**)}{=} \frac{P(y_{sub_c}, y_c, x)}{P(x)} \\ &\overset{(*)}{=} \frac{P(y_c, x)P(y_{sub_c} \mid y_c, x)}{P(x)} \\ &\overset{(*)}{=} \frac{P(x)P(y_c \mid x)P(y_{sub_c} \mid y_c, x)}{P(x)} \\ &= P(y_c \mid x)P(y_{sub_c} \mid y_c, x) \end{aligned} \tag{4.3}$$

Accordingly, the coarse prediction can be explicitly used to inform the fine classifier following the coarse-to-fine paradigm through the label structure defined for this urban buildings classification. Two main approaches were taken to integrate this prior knowledge within the learning process: developing a custom loss function or modifying the structure of the model itself. Several attempts of customizing the fine classifier's loss function to give higher consideration to the subclasses of the previously predicted coarse class have failed. On the other hand, experiments involving the customization of the neural network itself while using regular cross entropy as the loss function for both branches were more promising.

Incorporating (4.3) in the model has the potential of leading to a better solution to the urban buildings categorization problem. Experiments culminated in the encoding of the product in (4.3) into the network through the development of a new additional layer that performs the multiplicative operation as pictured in Fig. 4.5. The color code is used to express which elements are multiplied together.

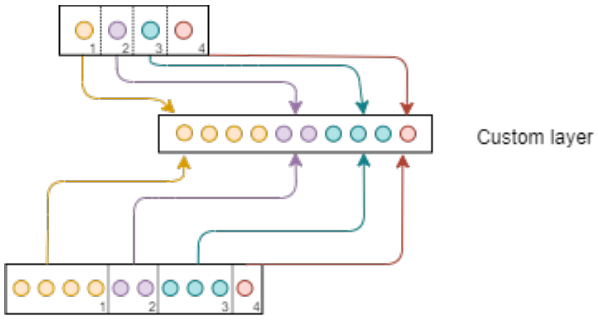


Figure 4.4: Simplified representation of the multiplicative custom layer

Let $\boldsymbol{v_c}$ and $\boldsymbol{v_{sub}}$ be two resulting tensors containing the coarse classes logits and the sub-classes logits respectively. First, $\boldsymbol{v_{sub}}$ needs to be divided into N_c sub-vectors $\boldsymbol{v_{sub_i}}$ with $i \in 1, \ldots, N_c$, such that each resulting $\boldsymbol{v_{sub_i}}$ has $|H(i)|$ elements, i.e. the number of fine classes the corresponding coarse class i is mapped to according to the defined label tree. Then, for each $i \in 1, \ldots, N_c$, the scalar multiplication

$$v_{c_i} \boldsymbol{v_{sub_i}}$$

is performed, with the scalar v_{c_i} being the ith element of $\boldsymbol{v_c}$.

In the context of the urban buildings hierarchical classification task on-hand, this multiplicative layer needs as input the tensor of coarse classes logits $\boldsymbol{v_c}$ which is obtained from a fully connected layer with N_c neurons, along with the tensor of sub-classes logits $\boldsymbol{v_{sub}}$ which in turn is obtained from a fully connected layer with N_f neurons. By putting all the pieces together, we can now build a custom hierarchical model for solving the given task:

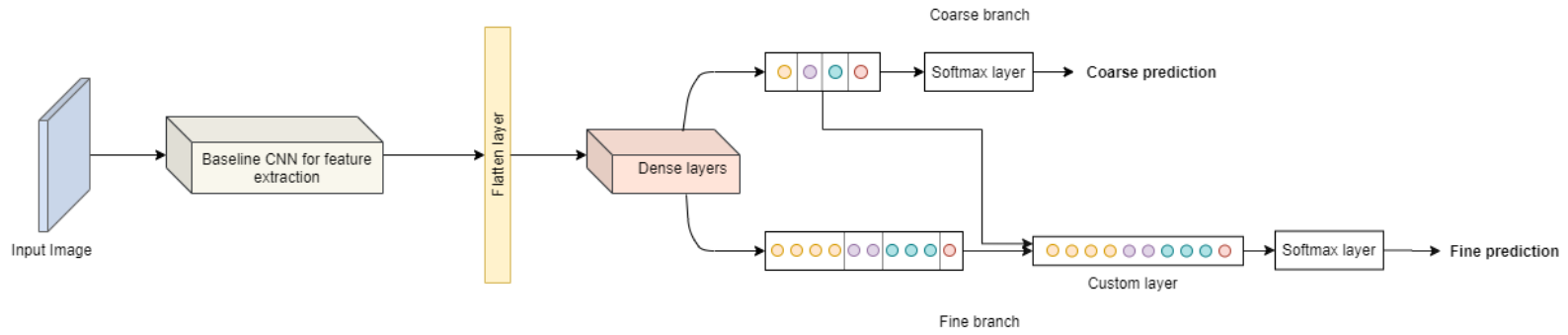


Figure 4.5: Proposed Model

Using the findings listed in Table 4.2, the proposed model branches out into coarse and fine branches at the same level as can be observed from Fig. 4.5. The model extracts features from a given image I by passing it through the baseline CNN model, VGG-16, then after routing the features through fully connected layers, the model splits into two different branches corresponding to the coarse and fine levels of the pre-defined class tree structure. For the coarse branch, the model outputs the coarse logits which are then normalized by a softmax layer to get the model's interpretable coarse classes predictions. Similarly, for the fine branch, it gets the sub-classes logits. At this point, the coarse branch gives an explicit feedback to the fine classifier by routing its coarse logits as input to the multiplicative layer alongside the sub-classes logits to compute products (4.3) which are then routed to a softmax layer to obtain the fine classes predictions.

Regarding the coarse and fine classifiers' loss functions, L_{coarse} and L_{fine} respectively, cross entropy is used for both. The entire model's loss is simply the weighted sum of both losses. Inspired by Zhu et al., we also follow the BT-strategy explained earlier [31], as it helps guide the model, shifting its focus from one branch to the other throughout training. Hence the model's overall loss is the same as in (4.1), and it can be simply rewritten as:

$$L_i = A_1 L_{coarse_i} + A_2 L_{fine_i} \tag{4.4}$$

4.6 Learned Hierarchy with Spectral Clustering

So far, the hierarchical classification task was only explored based on a pre-defined, "natural" division of fine classes into coarse ones as previously shown in Fig. 4.2. How would the models perform given a learned hierarchy? In other words, instead of dictating a given category class tree, it can be interesting to use an algorithmic approach, such as clusterning, to group confusing fine sub-classes together resulting in a learned hierarchy

of categories. Clustering is an unsupervised learning technique. In contrast to supervised learning introduced early on, unsupervised learning aims at finding previously undetected patterns in a given dataset that has not been labeled thus far.

In order to group similar and confusing fine classes under the same parent coarse class, we rely on one of the most popular clustering techniques: spectral clustering. This technique considers the clustering problem as a graph partitioning task. For the implementation of spectral clustering, the first step is to obtain a similarity matrix $S \in \mathbb{R}^{n \times n}$, with $n = N_f = 10$ in the context of the urban buildings problem on-hand.

The similarity matrix can be obtained as follows:

1. Train the VGG-16 flat CNN and evaluate it on a balanced held-out test set.
2. Compute the confusion matrix M from the network's predictions on the test set.
3. Set the diagonal entries of M to 0.
4. Symmetrize M to obtain the similarity matrix by performing: $S = \frac{1}{2}(M + M^T)$

As one of the most commonly used performance measures used for the evaluation of classification models, the confusion matrix is particularly useful in this use-case. In general, for an n-class classification task, a confusion matrix is a matrix M with entries $M_{ij} \in \mathbb{N}_{\geq 0}^{n \times n}$. This matrix contains all correct and wrong predictions made by the evaluated model such that M_{ij} represents the number of instances from class j which were classified as belonging to class i. Consequently, the diagonal of the confusion matrix holds all the correct predictions (M_{ii}), while all the remaining entries represent wrong predictions. Since the objective of building a learned hierarchy is to identify similar classes which tend to be "*confused*" (misclassified) with each other, the confusion matrix is a great starting point for setting up the similarity matrix to be used for spectral clustering. Entries S_{ij} of the resulting matrix express how easy it is to differentiate between classes i and j. Using this information, the goal is to build a different mapping $H' : [1, N_f] \rightarrow [1, N_c]$

In order to proceed with spectral clustering, we programmatically implement an algorithm adapted from Ulrike von Luxburg's work "A Tutorial on Spectral Clustering" [26] as shown in Algorithm 1 below. Given the similarity matrix $S \in \mathbb{R}^{n \times n}$ obtained earlier, and k the number of clusters to construct:

Algorithm 1 Spectral Clustering

Construct a similarity graph based on the k-nearest neighbor graph approach along with its weighted adjacency matrix A.
Compute the degree matrix[4] D.
Compute the graph's Laplacian $L = D - A$.
Compute the first k eigenvectors $v_1, \ldots, v_k$ of the Laplacian L and set them as the columns of a matrix $V \in \mathbb{R}^{n \times k}$.
For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the ith row of V.
Cluster the points $(y_i)_{i=1,\ldots,n}$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, \ldots, C_k$.

As can be observed from Algorithm 1, spectral clustering results in a mapping of the classes into clusters by relying on the spectrum of the similarity graph's Laplacian. This procedure outputs a set of fine class indices with their corresponding clusters which represent the parent coarse classes they are grouped under. In the next chapter, results of the implementation of spectral clustering are presented. The resulting hierarchy is also fed to both the B-CNN and our proposed model in order to investigate how these networks perform with a learned label tree in comparison to the previously given "natural" one.

[4]It is a diagonal matrix where each entry D_{ii} represents the number of edges connected to vertex i's (i.e. its degree)

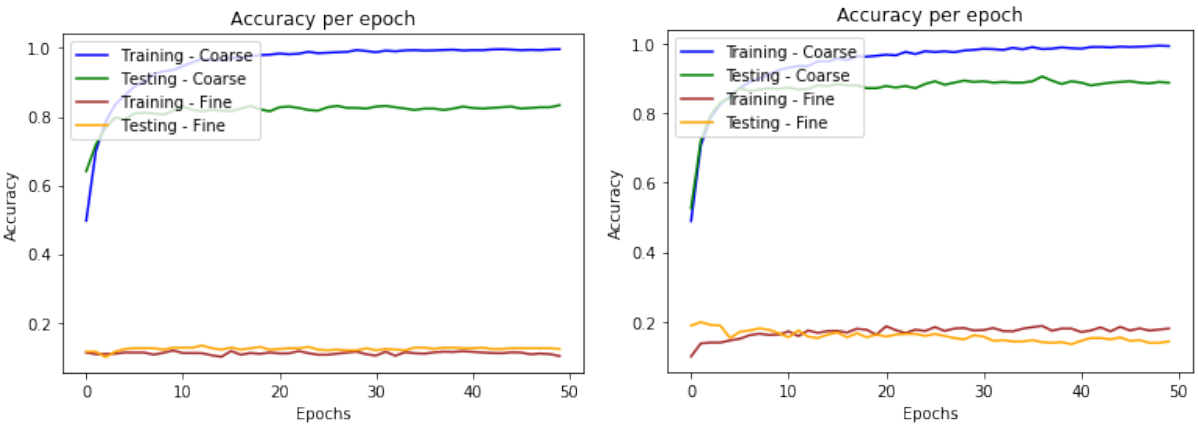
Chapter 5

Results and Discussion

In these experiments, stochastic gradient descent (SGD) is used as the optimizer with momentum set to 0.9. As the objective of this research work is mainly to solve the urban buildings classification task and compare different hierarchical model configurations, the number of training epochs is limited to 50. Since the dataset only contains about 6,300 images, this number of epochs is reasonable especially that we do not seek to achieve the best possible performance, but rather to set a proof of concept and check how the proposed method compares to well-established techniques.

First, the manual pre-defined label tree (presented in Fig. 4.2) is used for training and evaluating the models. Then, the experiments are reproduced for a learned hierarchy of classes obtained using spectral clustering.

In order to illustrate the role of the loss weights in putting the focus on a certain branch in the BT-strategy of the B-CNN model and that of the hierarchical proposed method, suppose we set the loss weights to $A_1 = 1$ and $A_2 = 0$ in (4.1), then the models ignore the fine branch classification, and only focus on learning the coarse classification as can be seen from Fig. 5.1 below:

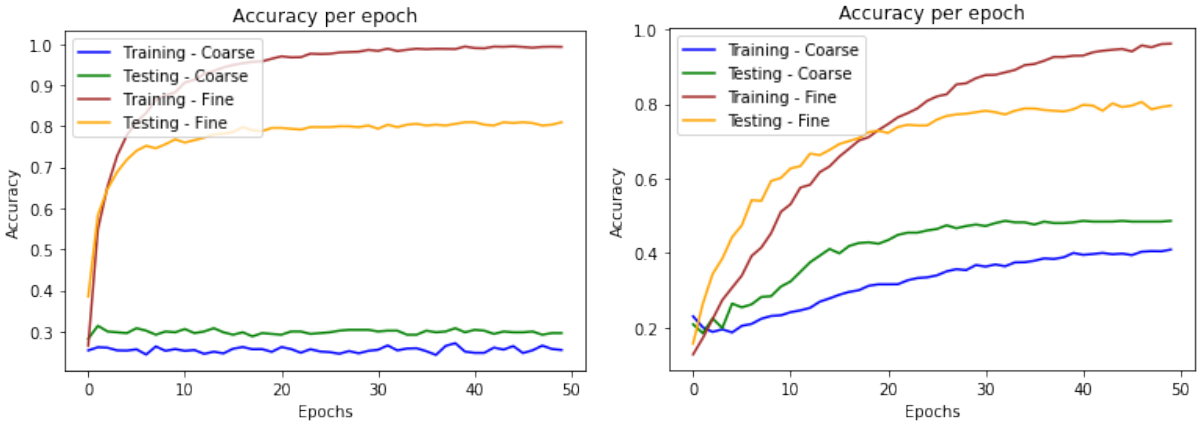


(a) B-CNN **(b)** Proposed model

Figure 5.1: Accuracy per epoch when the loss weights are 1 and 0 respectively for the coarse and fine branches

Similarly, if we set $A_1 = 0$ and $A_2 = 1$, the model nullifies the learning process of the

coarse branch:



(a) B-CNN **(b)** Proposed model

Figure 5.2: Accuracy per epoch when the loss weights are 0 and 1 respectively for the coarse and fine branches

As can be observed from Fig. 5.2, in the proposed method compared to the B-CNN's behavior, the fine classifier is slower to learn when the coarse branch's loss weight is zero. This is because the fine branch in the proposed model takes a direct input from the coarse branch. When the latter gives unreliable input due to the loss weights setting, it impedes on the fine classifier's learning process. In contrast, in the B-CNN model, with $A_1 = 0$ and $A_2 = 1$, the B-CNN merely converges to a flat CNN.

In the B-CNN model, the information gained from the coarse prediction is implicit. It benefits from the fact that lower-level layers of CNNs learn generic patterns, and as they get deeper, higher-level features are learned by branching out at different points of the baseline VGG-16 neural network. Therefore, the deeper levels of the network can focus on solely solving the fine classification task and only use the features that were useful to the coarse branch as an overall hint to target certain fine classes (usually the children of the coarse class) more than others.

On the other hand, the proposed model makes coarse and fine predictions at the same level, so the features learned by the model should help it solve both tasks at once. With the BT-strategy, the network tries to find a trade-off between learning features that are relevant for the coarse classification along with those relevant for the fine classification while decreasing the overall loss. It also explicitly takes the coarse logits as input in order to include this information in making the fine classification decision. This explains why the coarse branch is so crucial to the fine branch training process which leads to the behavior observed in Fig. 5.2.

In order to compare the performance of the proposed method with the B-CNN (as set up previously in Chapter 4), and with the VGG-16 as a flat CNN, it was trained using the BT-strategy with starting loss weights: $A_1 = 0.3$ and $A_2 = 0.7$ in order to put emphasis on learning features relevant to the fine classification, then after 15 epochs, they are set to $A_1 = A_2 = 0.5$. The results obtained for the proposed hierarchical model compared

to the abovementioned models are given below in Table 5.1. The proposed multi-label hierarchical model is more accurate than the B-CNN in classifying urban buildings across both levels of the hierarchy.

	Coarse Branch Accuracy	Coarse Branch Loss	Fine Branch Accuracy	Fine Branch Loss
VGG-16	N/A	N/A	78.22%	0.67
B-CNN	82.35%	0.51	79.85%	0.60
Proposed Method	**92.65%**	**0.23**	**82.10%**	0.89

Table 5.1: Average performance of the models

The hierarchical categorization problem posed in this thesis can be viewed as an instance of multitask learning. Indeed, most of the proposed model — its CNN feature extractor and dense layers — are all fully shared across two tasks (coarse classification and fine classification). In our context, the sharing of parameters is justified by the existing relationship between the two tasks which can be inferred from the label tree previously shown in Fig. 4.2. The parameter sharing scheme in multitask learning often yields better generalization and generalization error bounds compared to using different layers for each task [5]. Table 5.2 provides a comparison of the number of parameters used by the B-CNN model adapted to the task on-hand, and the number of parameters used by the proposed model. For reference, the VGG-16's total number of parameters is given as well.

	Total number of parameters
VGG-16 conv. blocks	14,714,688
B-CNN	79,163,470
Proposed Method	21,190,606

Table 5.2: Total number of parameters per model

Compared to the B-CNN structure, the proposed model configuration only added about 6,475,918 parameters (vs. the 64,448782 parameters added in the B-CNN) to the baseline conv. network. The B-CNN needed about 10 times more parameters than our proposed model. The stronger parameter sharing scheme used in the proposed method presents a significant advantage in the reduction of the model's memory requirements and footprint. As can be observed from Fig. 4.4, not only do the coarse and fine branches rely on the same baseline CNN feature extractor, the VGG-16; but they also share fully connected layers. This parameter sharing scheme gives the proposed method a stark advantage over the B-CNN in that it uses significantly less parameters without it taking a toll on the model's performance.

To better comprehend the proposed model's capabilities, an instance was trained then evaluated on a held-out set of 1,260 images. Of these images, 84 were assigned a wrong coarse label, and 232 a wrong fine label. Out of the 84 images for which the model gave a wrong coarse prediction, 94% were assigned an incorrect fine label that is a subclass of

the wrong coarse class predicted by the model, thereby enforcing the importance of the coarse classifier in guiding the fine branch since the former constrains the latter. This is precisely why the coarse branch should perform as well as possible, which justifies the design decision of placing it at the end of the feature extraction baseline conv. net so it benefits from higher-level learned features as well.

Out of the images that were assigned an incorrect fine label: 32% resulted from a wrong coarse prediction, and 64% resulted from the prediction of the incorrect subclass among the right coarse class's children. The remaining 4% of these 232 images had a wrong fine label that is not any of the children of its correct predicted coarse label. These results indicate that even though the proposed method is able to classify images better by using coarse predictions to guide the fine branch, the latter presents a limitation in its ability to discriminate between the subclasses of the predicted coarse class.

Comparison with Human Vision

To establish a comparison between the performance of a human compared to that of the proposed method, a total of 464 images were collected from the held-out set on which the model was evaluated earlier. 232 of these images correspond to all the images that the model misclassified and are gathered into Set 1. For the sake of comparison, 232 of the correctly classified images are also collected into Set 2. The two resulting sets of images are presented to five human participants instructed to categorize them into one of the N_f fine classes[1] (Mosque, House, etc.). All images were resized[2] to 224×224 prior to being presented to the participants. Table 5.3 gives an overview of the results of this experiment.

	Set 1	Set 2
Participants' average accuracy	66.4.%	86.6%

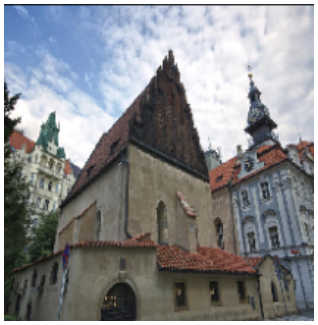
Table 5.3: Participants' average accuracy in the classification of two sets of images

The participants' worse performance on the classification of Set 1 images indicates that those images are particularly hard to classify, even for human viewers who may recognize brand logos and read name boards which can hint at the building's function even if it is unrecognizable from the building itself. Human observers were able to correctly classify 66.4% of the buildings that the hierarchical model misclassified. In contrast, out of Set 2 images which were all correctly labeled by the model, the participants only got 86.6% right. This comparison against human observers provides a useful benchmark for the evaluation of the proposed method. Examples of images from both sets are provided in Fig. 5.3 and Fig 5.4 respectively, along with their ground truth labels.

[1]There is no need to ask the participants to provide a coarse label since it is straightforwardly inferred from the fine class.

[2]This is done to have a "fair" comparison since the proposed method relies on the VGG-16 which accepts images of this specific size

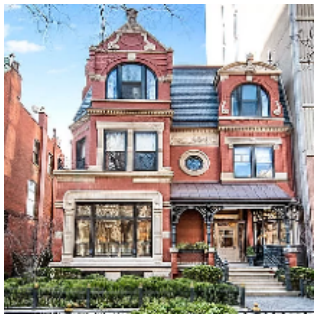
(a) Apartment Building



(b) Synagogue

(c) Church

(d) Mall



(e) House

(f) Restaurant

Figure 5.3: Examples of Set 1 images

(a) House

(b) Office Building

(c) Church

(d) Store

(e) Mosque

(f) Synagogue

Figure 5.4: Examples of Set 2 images

Learned Hierarchy with Spectral Clustering

After performing spectral clustering as explained earlier to group confusing classes together, the label tree presented in Fig. 5.5 was obtained.

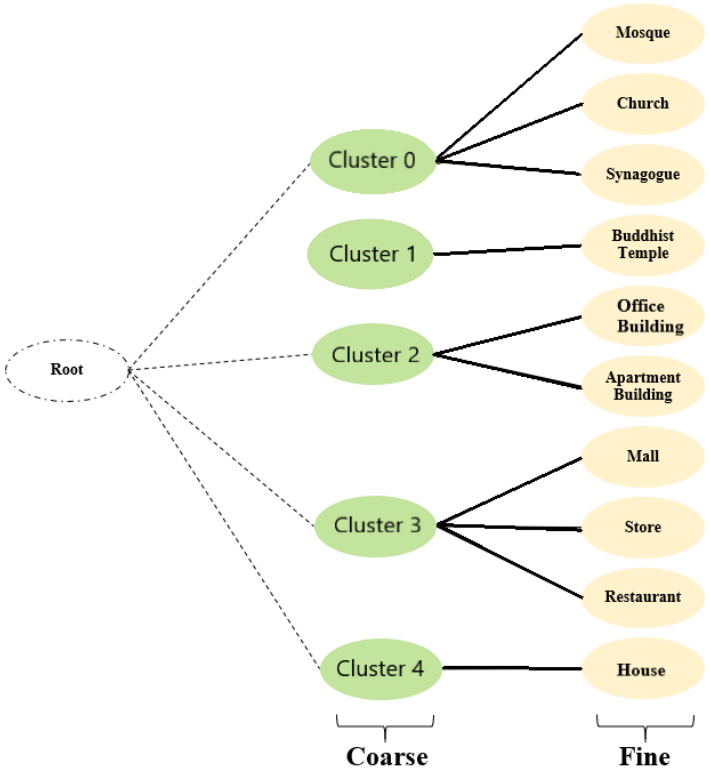


Figure 5.5: Resulting learned hierarchy

The resulting hierarchy is quite similar to the pre-defined one: urban buildings fulfilling the same general functional purposes tend to share certain architectural elements that are useful to the role they play and/or are indicative of it. Consequently, generally speaking, a church is more likely to be confused with a synagogue than with a mall. However, given our urban buildings dataset, apartment buildings and office buildings are clustered together in spite of them not having the same functional purpose as can be seen from Fig. 5.5. This may be because most apartment buildings tend to be tall buildings with numerous windows per floor on their facades, and these are similar characteristics that can usually be found on office buildings. For the sake of investigating how the networks perform in light of this learned hierarchy, in Table 5.4, the average performance of both the B-CNN and the proposed model are given.

	Coarse Branch Accuracy	Coarse Branch Loss	Fine Branch Accuracy	Fine Branch Loss
B-CNN	79.87%	0.58	76.28%	0.72
Proposed Method	**91.83**%	**0.26**	**79.40%**	0.86

Table 5.4: Average performance of the models under the learned hierarchy of labels

Training the models following the class hierarchy obtained from spectral clustering, and evaluating them on held-out sets led to the results given in Table 5.4. They both have a decreased performance compared to when the pre-defined hierarchy is used (see Table 5.1), yet the proposed model still performs better than the B-CNN even under this different hierarchy.

In this label tree, there are 5 coarse classes. The lower performance of the B-CNN can be explained by the fact that it branches out to the coarse classifier at an intermediary level within the baseline model, hence it does not benefit from the full extent of the model's feature extraction. Therefore, with now more coarse classes, it is even harder for the B-CNN's coarse branch to perform as well. The proposed model surpasses this since its coarse classifier does not suffer as much from the higher number of coarse classes thanks to its position at the end of the baseline model.

Regarding the proposed model's fine classifier's accuracy. It is also lower compared to the one obtained with the pre-defined hierarchy. This is expected since the fine branch is constrained with the coarse prediction. Hence, as the model makes more incorrect coarse predictions, the fine branch is more likely to be induced into error. As for the fine branch's loss, it is slightly lower on the learned hierarchy (0.86) than it is for the pre-defined one (0.89). Since the hierarchy has 5 superclasses with a couple having only one subclass, the model tends to be more confident in its predictions of these subclasses, i.e. the resulting probabilities for the correct classes tend to be higher.

Chapter 6

Conclusion and Future Work

In hereby solving the urban building classification task, experimentation with different model configurations has led to the development of a novel hierarchical model that performs better than its well-established hierarchical counterpart, the Branch Convolutional Neural Network. Following a defined label structure that relies on the coarse-to-fine paradigm, the proposed model explicitly incorporates prior knowledge obtained from a coarser level as input to the finer level thereby improving the accuracy of the overall model in solving the urban building categorization task while using significantly less parameters. Future research should consider the extension of the proposed model to adapt to potentially deeper label trees (i.e. more than just two levels of hierarchy) by for instance adding an additional layer of complexity to the urban buildings classification task by including the architectural styles in the class hierarchy. Consequently, the multi-label multi-class model will be able to give more enriching information about a given urban building from its photograph. The model can be further improved by investigating ways to boost the discrimination abilities of the model among the fine classes, subclasses of a given coarse class. It would also be interesting to expand the label tree's width by adding more classes such as: school, hospital, museum, etc. and enlarging the dataset accordingly. Looking forward, other avenues of research could be pursued to further benefit the literature by investigating the applicability of this model to other problems beyond the posed urban buildings one, and checking how it performs with existing well-known datasets.

Bibliography

[1] Gong Cheng and Junwei Han. "A survey on object detection in optical remote sensing images". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 117 (2016), pp. 11 –28. ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2016.03.014. URL: http://www.sciencedirect.com/science/article/pii/S0924271616300144.

[2] Joseph Paul Cohen et al. "Rapid building detection using machine learning". In: *Applied Intelligence* 45 (2016), pp. 443–457.

[3] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[4] Bolin Gao and Lacra Pavel. "On the properties of the softmax function with application in game theory and reinforcement learning". In: *arXiv preprint arXiv:1704.00805* (2017).

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[6] Namig J Guliyev and Vugar E Ismailov. "A single hidden layer feedforward network with only one neuron in the hidden layer can approximate any univariate function". In: *Neural computation* 28.7 (2016), pp. 1289–1304.

[7] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural networks* 4.2 (1991), pp. 251–257.

[8] Andrej Karpathy. *Convolutional Neural Networks*. https://cs231n.github.io/convolutional-networks/. Accessed: 2020-01-06.

[9] Andrej Karpathy. *Image Classification*. https://cs231n.github.io/classification/. Accessed: 2020-01-05.

[10] Andrej Karpathy. *Neural Networks Part 1: Setting up the Architecture*. https://cs231n.github.io/neural-networks-1/. Accessed: 2020-01-06.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[12] S. Lee et al. "Linking Past to Present: Discovering Style in Two Centuries of Architecture". In: *2015 IEEE International Conference on Computational Photography (ICCP)*. 2015, pp. 1–10. DOI: 10.1109/ICCPHOT.2015.7168368.

[13] Xiaojiang Li, Chuanrong Zhang, and Weidong Li. "Building block level urban land-use information retrieval based on Google Street View images". In: *GIScience & Remote Sensing* 54.6 (2017), pp. 819–835. DOI: 10.1080/15481603.2017.1338389. eprint: https://doi.org/10.1080/15481603.2017.1338389. URL: https://doi.org/10.1080/15481603.2017.1338389.

[14] Manolis Loukadakis, José Cano, and Michael O'Boyle. "Accelerating Deep Neural Networks on Low Power Heterogeneous Architectures". In: Jan. 2018.

[15] Zhou Lu et al. "The expressive power of neural networks: A view from the width". In: *Advances in neural information processing systems*. 2017, pp. 6231–6239.

[16] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[17] Abraham Montoya Obeso et al. "Architectural style classification of Mexican historical buildings using deep convolutional neural networks and sparse features". In: *Journal of Electronic Imaging* 26 (Dec. 2016), p. 011016. DOI: 10.1117/1.JEI.26.1.011016.

[18] Valentin Muhr et al. "Towards Automated Real Estate Assessment from Satellite Images with CNNs". In: *Forum Media Technology*. 2017.

[19] Young-Il Pyo, Ji Lee, and Rae-Hong Park. "Building image retrieval using color and line features". In: Oct. 2009, pp. 1381 –1386. DOI: 10.1109/ISCIT.2009.5341070.

[20] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[21] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[22] Gayane Shalunts, Yll Haxhimusa, and Robert Sablatnig. "Architectural Style Classification of Building Facade Windows". In: *Advances in Visual Computing*. Ed. by George Bebis et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 280–289. ISBN: 978-3-642-24031-7.

[23] Gayane Shalunts, Yll Haxhimusa, and Robert Sablatnig. "Architectural Style Classification of Domes". In: *Advances in Visual Computing*. Ed. by George Bebis et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 420–429. ISBN: 978-3-642-33191-6.

[24] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: 2014. arXiv: 1409.1556 [cs.CV].

[25] Carlo Tomasi. "Training Convolutional Neural Networks". In: (). URL: https://www2.cs.duke.edu/courses/spring19/compsci527/notes/cnn-training.pdf.

[26] Ulrike Von Luxburg. "A tutorial on spectral clustering". In: *Statistics and computing* 17.4 (2007), pp. 395–416.

[27] Zhe Xu et al. “Architectural Style Classification Using Multinomial Latent Logistic Regression”. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 600–615. ISBN: 978-3-319-10590-1.

[28] Zhicheng Yan et al. “HD-CNN: hierarchical deep convolutional neural networks for large scale visual recognition”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2740–2748.

[29] X. Yuan and C. Li. “CBIR approach to building image retrieval based on invariant characteristics in Hough domain”. In: *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*. 2008, pp. 1209–1212. DOI: 10.1109/ICASSP.2008.4517833.

[30] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[31] Xinqi Zhu and Michael Bain. “B-CNN: Branch Convolutional Neural Network for Hierarchical Classification”. In: 2017. arXiv: 1709.09890 [cs.CV].