ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΠΙΣΤΗΜΗ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΕΣ ΠΛΗΡΟΦΟΡΙΑΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Zero-Shot Domain Adaptation for Sketch Recognition

Νικολαος Χ. Ευθυμιάδης

Επιβλέπων: Αβρίθης Γιαννης, Research Scientist

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2020

NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

INTERDEPARTMENTAL PROGRAM OF POSTGRADUATE STUDIES IN DATA SCIENCE AND INFORMATION TECHNOLOGIES

MSc THESIS

Zero-Shot Domain Adaptation for Sketch Recognition

Nikolaos C. Efthymiadis

Supervisor: Avrithis Yannis, Research Scientist

ATHENS

JULY 2020

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Zero-Shot Domain Adaptation for Sketch Recognition

Νικολαος Χ. Ευθυμιάδης
Α.Μ.: DS1180006

ΕΠΙΒΛΕΠΩΝ: **Αβρίθης Γιαννης**, Research Scientist

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Τόλιας Γιώργος**, Επίκουρος Καθηγητης
Εμίρης Ιωάννης, Καθηγητής

Ιούλιος 2020

MSc THESIS

Zero-Shot Domain Adaptation for Sketch Recognition

Nikolaos C. Efthymiadis
S.N.: DS1180006

SUPERVISOR: **Avrithis Yannis**, Research Scientist

EXAMINATION COMMITTEE: **Tolias Giorgos**, Assistant Professor
Emiris Ioannis, Professor

July 2020

ΠΕΡΙΛΗΨΗ

Στην παρούσα εργασία αντιμετωπίζουμε το πρόβλημα του “zero-shot domain adaptation” μεταξύ των πεδίων των εικόνων και σκίτσων. Τον όρο ”zero-shot” τον εννοούμε ως προς τα σκίτσα. Ο όρος ”domain adaptation” υπονοεί ότι πρόκειται να μάθουμε χρησιμοποιώντας εικόνες ως παραδείγματα και θα χρησιμοποιήσουμε αυτή τη γνώση για να προβλέψουμε σκίτσα.

Για να το κάνουμε αυτό, θα κάνουμε έναν μετασχηματισμό στις παρατηρούμενες εικόνας και έναν μετασχηματισμό στα παρατηρούμενα σκίτσα έτσι ώστε τα μετασχηματισμένα δεδομένα να μπορεί να θεωρηθεί ότι παράχθηκαν από την ίδια τυχαία μεταβλητή. Μετά από αυτό, η στρατηγική μας είναι να προσαρμόσουμε μια συνάρτηση στις εικονες που έχουν μετασχηματιστεί και να προβλέψουμε με αυτή τα σκίτσα κανονικά.

Οι μετασχηματισμοί που πρόκειται να χρησιμοποιήσουμε έχουν ως βασικότερο στόχο να αναπαραστήσουν και τους δύο τομείς ως δυαδικές ακμές πάχους ενός εικονοστοιχείου, κωδικοποιώντας με 1 την ύπαρξη ακμής και με 0 την απουσία της. Για τον τομέα της εικόνας θα χρησιμοποιηθεί ένας συνδυασμός διαδικασιών ανίχνευσης ακμών, αυτόματης κατωφλίωσης και λέπτυνσης σε μια φιλοσοφία επαύξησης δεδομένων. Για τον τομέα του σκίτσου θα χρησιμοποιήσουμε έναν ντετερμινιστικό κανόνα κατωφλίωσης σε συνδυασμό με μια διαδικασία λέπτυνσης.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Όραση Υπολογιστών

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Zero-Shot Learning, Domain Adaptation, Sketch Recognition

ABSTRACT

In this thesis we are tackling the zero-shot domain adaptation problem between the image and the sketch domains. By ”zero-shot” we mean in term of sketches. The term ”domain adaptation” suggests that we are going to learn with images as examples and we will use that knowledge to predict sketches.

In order to do this we are going to make one transformation to the image realizations and one transformation to the sketch realizations such that the transformed realizations can be seen as they were sampled from the same random variable. After that our strategy is to learn a function on the transformed realizations of the image domain and predict our sketches normally.

The transformations we are going to use have as an objective to represent both domains as one-pixel thick binary edge maps, encoding with 1 the existence of an edge and with 0 the absence of it. For the image domain a combination of edge detection, thresholding and thinning procedures is going to be used in a data augmentation philosophy. For the sketch domain we will use a deterministic thresholding rule followed by a thinning procedure.

SUBJECT AREA: Computer Vision

KEYWORDS: Zero-Shot Learning, Domain Adaptation, Sketch Recognition

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Problem Discription and Motivation

In this thesis we are tackling the sketch recognition, or sketch classification problem, but in the case that there are no actual sketches available. In that sense our problem is a zero-shot learning problem in terms of sketches. Furthermore, we will use images of the objects we want to classify, which are the corresponding elements of another domain. In that sense our problem is also a domain adaptation problem.

The motivation behind the use of images instead of sketches is firstly practical. Sketches are more expensive and harder to obtain than images. Although the situation would be reversed for most of the previous century, the fact that there is a camera in every smartphone -and every smartphone is able to connect to the internet- made images trivially obtainable, while sketches continue to need effort to be created.

A second motivation is the self value that the domain adaptation task has in a philosophical manner. Specifically the elements of the real-object subset of the sketch domain have a causal relationship with the elements of the image domain because they are created as abstract representations of them.

Although the sketches are created in order to be understood by humans usually in an obvious, non-cryptic way, the chosen level of abstraction, the personal style and the skill gap between the artists are factors that add variety to the sketch domain even on representations of the same ontological entities. For example, a stick-figure, a sketch of an amateur cartoonist, a draft sketch of a cubist painter and a draft sketch of a statue from a Renaissance sculptor can all represent ontologically a person. In Figure 1.1 we can see an example of two non-professional sketches from the TU-Berlin dataset [17] with different level of abstraction, a draft sketch from an expressionist painter and a drawing from a neoclassical painter. This variety is making the sketch recognition problem challenging even in the classical supervised setting.

In the rest of the chapter 1 we will relate our setting with other settings and frameworks. We will also define our problem formally. In chapter 2 we will discuss the approaches we found the closest to ours in the literature. In chapter 3 we will present some theoretical background for the learning procedure. In chapter 4 we will analyze our whole approach for the problem. In chapter 5 we will set a baseline in order to have a reference point to evaluate our approach. Finally in chapter 6 we will present our results and final conclusions.

1.2 Domain Adaptation

In the supervised setting we assume that the examples of our train and validation sets are drawn from the same distribution with the examples of the test set. This is not true in our

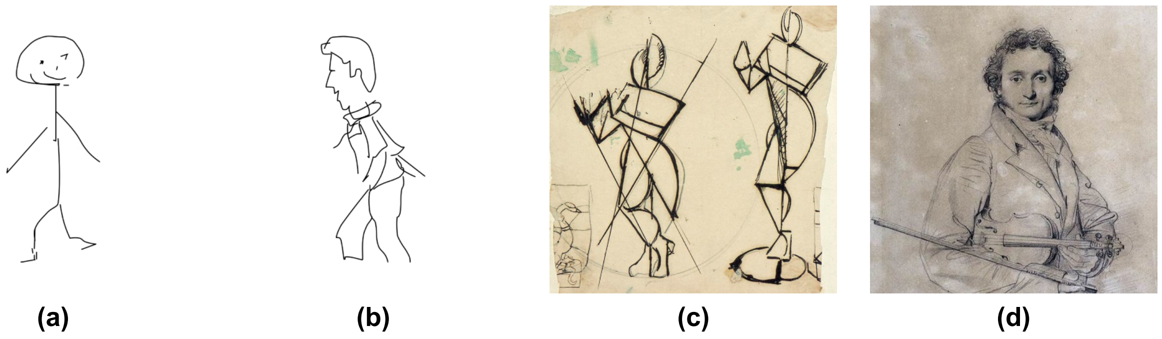


Figure 1.1: In (a) we can see a stick-figure style sketch from the class ”person walking” from TU-Berlin [17]. In (b) we can see a less minimal sketch from the class ”person walking” from TU-Berlin. In (c) we can see ”Krishna playing the flute” by Theo van Doesburg. And in (d) we can see a portret of Niccolo Paganini by Jean Auguste Dominique Ingres. In this work we will focus mainly to the first two cases of sketches

case because our available data are supposed to be only images, while the actual testing must be done on sketches. So the goal is to take any useful information from one domain and use it to do classification on the other domain. That practice falls under the definition of domain adaptation, but in order to define it formally, we need to define transfer learning first.

We will follow [12, 47, 70]. A **domain** $\mathcal{D}$ is defined as a feature space $\mathcal{X} \subset R^d$ with a marginal probability distribution $P(\boldsymbol{X})$. So we can write $\mathcal{D} = \{\mathcal{X},\ P(\boldsymbol{X})\}$. We also define a **task** as a label space $\mathcal{Y}$ with the conditional probability distribution $P(\boldsymbol{Y}|\boldsymbol{X})$, so we can write $\mathcal{T} = \{\mathcal{Y},\ P(\boldsymbol{Y}|\boldsymbol{X})\}$. $\boldsymbol{X}$ and $\boldsymbol{Y}$ are random variables. If we take N realizations of them in a dataset D, such as $\boldsymbol{x_D} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\}$ with $\boldsymbol{x}_i \in \mathcal{X}$ and $\boldsymbol{y_D} = \{y_1, y_2, \dots, y_N\}$ with $y_i \in \mathcal{Y}$ respectively, the $P(\boldsymbol{Y}|\boldsymbol{X})$ can be estimated from the pairs $\{\boldsymbol{x}_i, y_i\}$.

Now, let's say we have two domains. A **source** domain $\mathcal{D}_S = \{\mathcal{X}_S,\ P(\boldsymbol{X}_S)\}$ with it's corresponding task $\mathcal{T}_S = \{\mathcal{Y}_S,\ P(\boldsymbol{Y}_S|\boldsymbol{X}_S)\}$ and a **target** domain $\mathcal{D}_T = \{\mathcal{X}_T,\ P(\boldsymbol{X}_T)\}$ with it's corresponding task $\mathcal{T}_T = \{\mathcal{Y}_T,\ P(\boldsymbol{Y}_T|\boldsymbol{X}_T)\}$. The setting in which we want to use knowledge from $\mathcal{D}_S$ and $\mathcal{T}_S$ in order to improve the $\mathcal{T}_T$ of $\mathcal{D}_T$ is called **transfer learning**. As we can easily see, transfer learning is a generalization of the supervised learning, because in the trivial case of $\mathcal{D}_S = \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$ the transfer learning setting collapses into the standard supervised.

The special case of transfer learning in which the label space is common, that is $\mathcal{Y}_S = \mathcal{Y}_T = \mathcal{Y}$ is called **domain adaptation**. This is a relaxed definition. According to [12], ideally, the definition of domain adaptation should also have the $P(\boldsymbol{Y}|\boldsymbol{X}_S) = P(\boldsymbol{Y}|\boldsymbol{X}_T) = P(\boldsymbol{Y}|\boldsymbol{X})$ condition. So the stricter $\mathcal{T}_S = \mathcal{T}_T = \mathcal{T} = \{\mathcal{Y},\ P(\boldsymbol{Y}|\boldsymbol{X})\}$ condition should hold, but it was rejected as unrealistic in practice. If the source feature space is the same with the target feature space, $\mathcal{X}_S = \mathcal{X}_T$ while $P(\boldsymbol{X}_S) \neq P(\boldsymbol{X}_T)$, the problem is called **homogeneous transfer learning**. Finally, if the labeled data are only available in the source domain then we are in the **unsupervised domain adaptation** setting.

Our problem is of course homogeneous but although we technically don't have labeled data from the target domain, it would be triviality to classify it as unsupervised domain adaptation because it's not only the labels that we are missing, but the whole target dataset. This forces us to make decisions mainly qualitatively because we are not allowed to study examples from both domains to make any estimation. So we must decide using reason and our so far experience with the domains gained from our every-day life. That reasoning is closely related with zero-shot learning practices.

1.3 Zero-Shot Learning

According to [53], while today it is not hard to find images for thousands of basic categories and build models from them, it is quite hard to find enough images of other, infrequent or very specific categories. For example, while it is easy to find plenty of data for the object "car", it is quite hard to find enough data in order to recognize specific car models. As long as we are creating models from the categories we have plenty of data, new needs will be created for more specialized computer vision. Zero-shot learning is the setting in which there are no examples of the categories we want to predict. The definition of zero-shot learning according to [34] is the following:

> Let $(\boldsymbol{x}_1,\ y_1),\ \dots,(\boldsymbol{x}_m,\ y_m) \subset \mathcal{X} \times \mathcal{Y}$ be training samples where $\mathcal{X}$ is an arbitrary feature space ant $\mathcal{Y}$ consists of z discrete classes. The task is to learn a classifier $f : \mathcal{X} \longrightarrow \mathcal{Y}'$ for a label set $\mathcal{Y}'$ of z' classes, that is disjoint from $\mathcal{Y}$.

In order to solve this problem, a very popular strategy was proposed by [54]. We assume that we have as extra information the attributes of all classes, such that every class has a signature combination of a attributes, coded in a matrix $S \in [0,\ 1]^{a \times z}$ and the values can be Boolean or in the range of $[0,\ 1]$. Now, instead of learning the parameters of a matrix W in order to do the class estimation $\hat{\boldsymbol{y}} = X^\top W$, we are learning the parameters of a matrix V such that $\hat{\boldsymbol{y}} = X^\top V S$, where $V \in \mathbb{R}^{d \times a}$. That way the parameters learned a relationship between images and attributes. In order to do the prediction, we assume that we are provided with the attributes of the new z' classes $S' \in [0,\ 1]^{a \times z'}$ and our estimator would be $\hat{\boldsymbol{y}} = x^\top V S'$.

Observing the setting of zero-shot learning we just described, we can see it as a special case of transfer learning. The feature spaces are the same, but not the distributions, so $\mathcal{D}_S = \{\mathcal{X},\ P(\boldsymbol{X}_S)\}$ and $\mathcal{D}_T = \{\mathcal{X},\ P(\boldsymbol{X}_T)\}$. Also the label spaces are different $\mathcal{Y}_S \neq \mathcal{Y}_T$ and consequently are the tasks $\mathcal{T}_S \neq \mathcal{T}_T$. Finally, we have data only from the source domain and task.

The comparison of this setting with our problem is done because both lack of target domain data. While both our distributions over our feature spaces are different, our source-target distributions are much further away and methodologies like the one described above would be difficult to be implemented directly. In the attribute based zero-shot learning problem, some basic features that are indicated from the attributes are following the same distribution in both domains. For example let's say that the source domain has images of horses and the target domain has the images of zebras. The attribute ”eye” is pointing at cues from almost the same distribution, the one of the images of the eyes of mammals. On the other hand, a sketch of an eye is following a very different distribution from an image of an eye, even if they both represent eyes of subjects of the same species.

1.4 Our Problem

As we mentioned in a previous section, our problem is considered a homogeneous transfer learning problem. That is because we are working on the feature space of images and after some minor adjustments a sketch and an image can be represented with the same mathematical objects: Either a 3d tensor, by upgrading the sketches to RGB, or a matrix, by downgrading the images to grayscale. Also, the dimensions of them can be trivially modified to be the same with rescaling, padding or cropping. So we can treat this problem like the condition $\mathcal{X}_S = \mathcal{X}_T$ holds. But of course the marginal probability distributions of the elements of those spaces are not the same. A stick-person-like cue is not equally frequent in an image and in a sketch, so $P(\boldsymbol{X}_S) \neq P(\boldsymbol{X}_T)$.

Furthermore, we are in the setting of the relaxed domain adaptation because the $\mathcal{Y}_S = \mathcal{Y}_T$ condition holds. The labels of our source domain are the same with the labels of our target domain because we want to estimate the classes of the same ontological entities from different representations of them. But we cannot assume that the conditional probabilities of the label random variables given the feature random variables are equal for both domains, so $P(Y|\boldsymbol{X}_S) \neq P(Y|\boldsymbol{X}_T)$. That is because the distributions of the random variables $\boldsymbol{X}_S, \boldsymbol{X}_T$ could be chaotically different and this assumption seems unreasonable to be made at this step.

In terms of transfer learning our problem can be summarized as follows:

Our source domain is the image domain, and our target domain is the sketch domain. We have training and validation data only from the source domain and our target domain data are only for testing. We work under the conditions $\mathcal{X}_S = \mathcal{X}_T = \mathcal{X}$ and $\mathcal{Y}_S = \mathcal{Y}_T = \mathcal{Y}$. So formally we have the domains $\mathcal{D}_S = \{\mathcal{X},\ P(\boldsymbol{X}_S)\}$ and $\mathcal{D}_T = \{\mathcal{X},\ P(\boldsymbol{X}_T)\}$. We also have the tasks $\mathcal{T}_S = \{\mathcal{Y},\ P(\boldsymbol{Y}|\boldsymbol{X}_S)\}$ and $\mathcal{T}_T = \{\mathcal{Y},\ P(\boldsymbol{Y}|\boldsymbol{X}_T)\}$. The goal is to estimate the $P(Y|\boldsymbol{X}_T)$.

In order to tackle this problem we must transform it into a problem that the equality of the conditional probabilities can hold. This leaves us only with the choice of searching for transformations for our realizations $f_1(\boldsymbol{x}_{D_S})$ and $f_2(\boldsymbol{x}_{D_T})$ that define new random variables $\boldsymbol{X}'_S$ and $\boldsymbol{X}'_T$ respectively that follow closer marginal distributions $P(\boldsymbol{X}'_S)$, $P(\boldsymbol{X}'_T)$ and also conditional distributions $P(Y|\boldsymbol{X}'_S)$, $P(Y|\boldsymbol{X}'_T)$. Then we will make the nessecery assumption $P(Y|\boldsymbol{X}'_S) \approx P(Y|\boldsymbol{X'}_T)$ and learn the $P(Y|\boldsymbol{X}'_S)$ and use it in the place of $P(Y|\boldsymbol{X}'_T)$. As mentioned in a previous section, we will choose those transformations qualitatively and by using reasonable arguments and empirical knowledge.

2. RELATED WORK

2.1 Sketch Recognition

A quite recent dataset that comes with an application for sketch recognition is google's Quick, Draw![1][2] which marks the upcoming popularity of the field. This poppularity though could not be achieved without the publication of [17], in which the TU-Berlin -a large scale dataset of free-hand sketches- was released to the community and gave to the researchers the means to change the direction of the sketch recognition task from a mainly hand-crafted practice to a more data driven one. The dataset consisted of 20,000 free-hand sketches from non-experts in 250 categories of real objects. Two benchmarks were set in the same publication: The first one is the human accuracy for the dataset, which is 73%. The second one is a multi-class support vector machine using representations from a bag-of-features approach which achieves 56% accuracy.

Of course in the post AlexNet [32] era, the first instinct was to try some known convolutional neural network (CNN) architectures designed for the image domain, but sketches are textureless and really sparser objects than the images, and their success is not obvious at all. The first model that surpassed the human benchmark was the sketch-a-net [51] which used an AlexNet-like architecture but with bigger filter size in the first layer in order to solve the sparsity issue of the domain. Furthermore, sketch-a-net exploited the sequential nature of the sketch domain by training with multi-channeled sketches with every channel being a snapshot of the sketching procedure. It also used different scaling of the sketches combined with Bayesian fusion [8] in order to deal with the different levels of abstraction. This network achieved 74.9% accuracy on the TU-Berlin dataset. They also achieved an even better 77.95% accuracy with the sketch-a-net, two years later, in [50] mostly by designing augmentations such as stroke removal strategies and local and global deformations.

The idea of exploiting the sequential nature of the sketch domain is very popular in the sketch recognition field. In [58] a recurrent neural network (RNN) architecture with gated recurrent units (GRU) [9] is used. The argument is that although the sketch-a-net can handle the sequence of the strokes up to some degree, the convolutional neural networks are not created to handle sequential data by design equally well with an architecture that deals with sequential data inherently. The results of this work are not comparable with the sketch-a-net because they used a 160-category subset of the TU-Berlin.

Another method that is focusing on the sequence of the strokes is the sketch RNN Rasterization CNN (Sketch-R2CNN) [39]. This architecture takes as input the sketch as a vector with grouped sequences of points. Then it uses a bidirectional long short-term memory (LSTM) [60] unit with two layers to calculate the per-point stroke attention. After that, they introduced a novel neural line rasterization module (NLR) that takes the initial vector

[1]`https://quickdraw.withgoogle.com/`

[2]`https://github.com/googlecreativelab/quickdraw-dataset`

sketch and the calculated attention and it converts them to a rasterized pixel sketch in a differentiable way. After that the procedure continues with a CNN and a classifier. This combination is motivated from the extra information that the temporal ordering has in the vector representation of a sketch and the effectiveness of the current CNN architectures on the rasterized representation of an image. So they unified the two representations in that manner. The Sketch-R2CNN achieves 83.25% accuracy on the TU-Berlin dataset by using the ResNet50 [25] as the last CNN.

A different approach can be seen in [76], which is focused on two kinds of sketch-specific augmentations. The first one is called Bezer pivot-based deformation (BPD), which aims to generate more sketches in order to add variation to the dataset. This is done by partitioning the sketches in small patches and fitting cubic Bezier curves using the largest connected component of each patch. The second augmentation is called mean stroke reconstruction (MSR) and it aims to improve the quality of the sketches by reducing the intra-class variance. Both augmentations can be used on sketches without temporal cues. Those two augmentations used with a fused DenseNet-161 [29] and ResNet-152 architecture, achieved 84.27% accuracy on the TU-Berlin dataset.

2.2 Domain Bridging

The previous section was only about the standard supervised setting for the sketch domain. Bridging the domains of sketches and images is a highly motivated objective because of the instant applications in the sketch-based image retrieval (SBIR) task. If we can represent the images and the sketches in the same space with a consistent and class-aware way, then we could use sketches to search for images. Model-wise, the most common architecture for this is either the Siamese network [10], trained with couples of sketches and photos or the Triplet network [69], trained with triplets of sketches, positive photos and negative photos [73, 57, 74].

According to [41] we are going through the forth era of SBIR since 2014 and its most important characteristics are the use of deep learning models and the extension of the task to the fine-grained SBIR (FG-SBIR), which aims to also distinguish intra-class variations. According to [41], this concept was raised by [40]. Examples of FG-SBIR are the aforementioned [73, 57].

Of course the above approaches are not zero-shot in terms of sketches. A concept closer to ours can be found in [3] where the GoogLeNet [62] and the AlexNet architectures trained on ImageNet [13] are used directly to recognize sketches from the same class. Although the networks performed better than random guessing, the authors concluded that:

> ”The test networks are not useful to perform sketch classification and their classifications are different from the classifications offered by humans”

While the above setting is the same with ours, there was no intention to close the domain gap but rather to verify it. In [52] we can see an approach even closer to ours in the domain of landmark retrieval. The task is a sketch-based image retrieval and the training sketches were substituted with edgemaps of images extracted from a structure-from-motion pipeline.

3. BACKGROUND

3.1 Background Motivation

As we stated in a previous chapter, our strategy is to transform our problem in another one that $P(Y|\boldsymbol{X}'_S) \approx P(Y|\boldsymbol{X'}_T) \approx P(Y|\boldsymbol{X})$ can be a reasonable assumption and we will learn $P(Y|\boldsymbol{X}'_S)$ and treat it like it was $P(Y|\boldsymbol{X'}_T)$. After that we will use only learning theory and from now on we will use $P(Y|\boldsymbol{X})$ to describe our learning problem. The necessary steps we are making in the image processing part of this work are based on logical assumptions and techniques without the use of any strict or specialized theory. The most suitable topic for in-depth presentation is the learning part, which will be presented in a continuous way through the perspective of the bias-variance trade-off.

3.2 Forming a Loss Function for Classification

The first thing that we can say with certainty is that Y follows a categorical distribution with $P(Y = y_k|\boldsymbol{X} = x)$ as parameters, for $k = 1, 2, ..., K$ while K is the number of classes. We could say that the parameters are the $P(Y = y_k)$ but having the insight from our dataset, $P(Y = y_k|\boldsymbol{X} = x)$ can be seen as an improvement. The categorical distribution is a special case of the multinomial distribution with the number of sampled items fixed to 1. Its probability mass function is:

$$F(\boldsymbol{\alpha}|\boldsymbol{p}) = \prod_{i=1}^{C} p_i^{\alpha_i},\ with\ \sum_{i=1}^{C} p_i = 1 \tag{3.1}$$

A reasonable approach to this problem is to assume a function form $\boldsymbol{f} : \mathcal{X} \to \mathcal{Y}$ specified by some weights $\boldsymbol{W}$ that would approximate $P(Y = y_k|\boldsymbol{X} = x)$. After that, the $\boldsymbol{W}$ can be estimated by maximizing the likelihood function of Y. This is generally a good approach because the maximum likelihood estimators (MLE) are guaranteed to be asymptotically unbiased and they asymptotically reach the Crammer-Rao bound, which means that out of all the unbiased estimators, they have asymptotically the minimum possible variance [63]. The function $\boldsymbol{f}$ is also an estimator of the parameters $P(Y = y_k|\boldsymbol{X} = x_i)$, so let's assume it's form and say $f_{ik} = f_k(x_i) = \hat{P}(Y = y_k|\boldsymbol{X} = x_i)$. Its specific form will be discussed in a later section but for now it will be left abstract. Starting from maximizing the likelihood of the categorical distribution:

$$\underset{\boldsymbol{W}}{\text{argmax}}\, \mathcal{L}(\boldsymbol{W}) = \underset{\boldsymbol{W}}{\text{argmax}} \prod_{i=1}^{N} \prod_{k=1}^{K} {f_{ik}}^{y_{ik}} \tag{3.2}$$

$$= \underset{\boldsymbol{W}}{\text{argmax}} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} log f_{ik} \tag{3.3}$$

$$= \underset{\boldsymbol{W}}{\text{argmin}} -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} log f_{ik} \tag{3.4}$$

$$= \underset{\boldsymbol{W}}{\text{argmin}} \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{k=1}^{K} -y_{ik} log f_{ik} + \sum_{k=1}^{K} y_{ik} log y_{ik}\right] \tag{3.5}$$

$$= \underset{\boldsymbol{W}}{\text{argmin}} \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} log \frac{y_{ik}}{f_{ik}} \tag{3.6}$$

For the above equalities [4] and then [64] were followed. In order for some of the mathematical representations to make sense, the following conventions found in [64] were accepted: $0 \log \frac{0}{0} = 0$, $0 \log \frac{0}{q} = 0$ and $p \log \frac{p}{0} = \infty$.

The equation 3.2 is the definition of the likelihood of the categorical distribution inside the likelihood maximization procedure and the 3.3 is the maximization of the log-likelihood function.

The form 3.4 is the minimization procedure of the negative log-likelihood divided by the sample size. In an information theory perspective this form it the minimization of the average cross-entropy of the empirical distribution $\boldsymbol{y}$ and the model distribution $\boldsymbol{f}$ over the sample. In 3.5 the term $\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log y_{ik}$ was added, which is the average Shannon entropy of the empirical distribution over the sample. Which is of course independent from the optimization variables $\boldsymbol{W}$.

All the above were done in order to construct the 3.6 form which is called the Kullback–Leibler divergence (KL-divergence) between the empirical distribution $\boldsymbol{y}$ and the model distribution $\boldsymbol{f}$ [7]. The KL-divergence quantify the difference between the two distributions [33] and therefore it is a reasonable strategy to minimize the empirical and the model distribution difference.

From the above equations it is clear that no matter which optimization is chosen, the result will be theoretically the same. For our problem, we will choose the 3.4 over the 3.6 though because of its simpler form and the place of f_{ik} on the nominator.

3.3 Discussing the Model Form

By using the above strategy it is guaranteed to get an asymptotically unbiased estimation with the lowest possible variance over all the unbiased estimations for the $\boldsymbol{W}$ of a given function form. That also means that if the chosen function form is the ”right” one for the problem, the estimation method will have those good asymptotic properties for $P(Y|\boldsymbol{X})$ too. According to Leo Breiman [5], the search for the actual distribution that nature crafted for each problem and then assume it to learn its parameters is a data modeling culture approach. That culture dominated statistical theory and practices up to the 80s and quoting from Breiman:

”Led to irrelevant theory and questionable scientific conclusions, kept statisticians from using more suitable algorithmic models and prevented statisticians from working on exciting new problems”.

Examples of the data modeling culture are the generalized linear models. In contrast with this culture is the algorithmic modeling culture which believes that:

”Nature produces data in a black box whose insides are complex, mysterious, and, at least, partly unknowable.”

And its goal is to find a function or an algorithm that predicts nature well without following exactly nature’s procedure. Examples of this culture are the decision tree methods and neural networks.

The above rationality explains why the image recognition tasks started to achieve such good of a performance. The acceptance that those problems are way too complicated to look for the ”true natural” procedure that generated their data led the scientific community to a new way to produce useful results. So the approach became data driven but not in the data modeling culture way. In simpler problems, like -for example- an orange-watermelon classification from their weight in kg, someone could work in a data modeling culture approach. They would reasonably assume the distribution of Y as Bernoulli and X as Gaussian and the only things they would need to estimate for $\mathbb{E}[Y \mid x]$ are two means and two variances. In the image classification problem though, the distributions are not that obvious. Someone can’t know for example the distribution of dogs represented on a $\mathcal{X} = [0, 1]^{100\times100}$ feature space, so they need to approach it in an algorithmic modeling culture way. However, the ideal best model $\mathbb{E}[Y \mid x]$ can be used as a means to study the problem theoretically.

By following the approach of [20], and inspired by the use of the mean squared error (MSE) of the Euclidean norm in [72], the effectiveness of a model $\boldsymbol{f}$ is measured in terms of how close its performance is compared to the theoretical $\mathbb{E}[Y \mid x]$. A proof of why the $\mathbb{E}[Y \mid x]$ is the best possible theoretical model for the MSE loss can be found in [20]. The result of any statistical model creation procedure is going to vary in respect to the dataset D, so starting from the expected difference between a function and $\mathbb{E}[Y \mid x]$ over all possible datasets D:

$$\begin{aligned}
\mathbb{E}_D\left[\|\boldsymbol{f}(x;D) - \mathbb{E}[Y_k \mid x]\|_2^2\right] &= \mathbb{E}_D\left[\sum_{k=1}^{K}(f_k(x;D) - \mathbb{E}[Y \mid x])^2\right] \\
&= \sum_{k=1}^{K}\mathbb{E}_D\left[(f_k(x;D) - \mathbb{E}[Y_k \mid x])^2\right] \\
&= \sum_{k=1}^{K}\mathbb{E}_D\left[((f_k(x;D) - \mathbb{E}_D[f_k(x;D)]) + (\mathbb{E}_D[f_k(x;D)] - \mathbb{E}[Y_k \mid x]))^2\right] \\
&= \sum_{k=1}^{K}\mathbb{E}_D[(f_k(x;D) - \mathbb{E}_D[f_k(x;D)])^2] + (\mathbb{E}_D[f_k(x;D)] - \mathbb{E}[Y_k \mid x])^2
\end{aligned}$$

$$\begin{aligned}
&+ 2\mathbb{E}_D[f_k(x;D) - \mathbb{E}_D[f_k(x;D)]] \cdot (\mathbb{E}_D[f_k(x;D)] - \mathbb{E}[Y_k \mid x]) \\
&= \sum_{k=1}^{K} \mathbb{E}_D[(f_k(x;D) - \mathbb{E}_D[f_k(x;D)])^2] + (\mathbb{E}_D[f_k(x;D)] - \mathbb{E}[Y_k \mid x])^2 \\
&= \mathbb{E}_D[\|(\boldsymbol{f}(x;D) - \mathbb{E}_D[\boldsymbol{f}(x;D)])\|_2^2] \qquad 'Variance' \\
&+ \|\mathbb{E}_D[\boldsymbol{f}(x;D)] - \mathbb{E}[Y \mid x]\|_2^2 \qquad 'Bias' \qquad (3.7)
\end{aligned}$$

This decomposition is called Bias-variance decomposition and it describes a principal statement in statistics known as the bias-variance trade-off. As a principle, of course, it exists no matter which cost function we choose to illustrate it. A similar form for the $0-1$ cost function for classification can be found in [15], also for the cross-entropy loss there is a result in [72]. The MSE of the Euclidean norm was chosen because in the probabilistic classification setting it makes sense as a distance in terms of probabilities. Also this result shows the trade-off way more intuitively than the ones with the aforementioned loss functions. The bias part of the decomposition:

$$\|\mathbb{E}_D[\boldsymbol{f}(x;D)] - \mathbb{E}[Y \mid x]\|_2^2$$

It is the error part of the model that is caused due to the structural difference between the theoretical expectation of $Y|x$ and the function of choice. Specifically it is the difference between the theoretical ”true distribution” of the problem and the average function of choice over all possible datasets. This type of error is related to the systematic error of the physical sciences. The nearer the chosen function is to the theoretical one, the better performance will have bias-wise. The problem is that, as we discussed earlier in this section, the true function in the domain of image classification is too complicated and unknowable. So what somebody can do about this is to assume a more general form that includes the true function as a special case. The variance part of the decomposition:

$$\mathbb{E}_D[\|(\boldsymbol{f}(x;D) - \mathbb{E}_D[\boldsymbol{f}(x;D)])\|_2^2]$$

It is the average error that the model has because of the use of the specific dataset and it is related to the function’s complexity. Specifically it is the average distance of the model from the average model trained over every possible dataset. This kind of error is related to the random error of physical sciences. Of course, the larger the dataset, the smaller the error due to variance because the model’s parameters $\boldsymbol{W}$ will be in a better state in terms of convergence. In addition to the data quantity, a simpler structure of the function helps reducing the error of this kind. That’s because in a lot of simple structural choices, the less variables there are in a function, the less data are needed for them to start to converge. That is not a general rule but it applies to the structures we will be interested in.

So in order to reduce the variance error someone needs to assume simpler functions but in order to reduce the bias error the ”true function” must be included in the model as a special case. In other words, for a given dataset, the whole procedure is a trade-off of making the model simple but not too simple to include the ”true function”. This is going to be achieved by starting from a very complex and flexible family of functions and then by making reasonable assumptions for the nature of the problem to discard functions from the possible solutions.

3.4 Starting from a Low Bias Architecture

In order to start from a model with very low bias error, the fully connected multilayer perceptron (MLP) can be chosen as a baseline. This model, according to [23], is the quintessence of the deep learning models. The idea in the neural networks framework is to create a very complex and flexible function as a composition of a lot of simple functions. The fully connected MLP function for example will have the form:

$$\boldsymbol{f}(\boldsymbol{x}) = \sigma\left(f_L^{(1)}(\boldsymbol{x}), f_L^{(2)}(\boldsymbol{x}), ..., f_L^{(K)}(\boldsymbol{x})\right) \quad with \quad \sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad and$$
$$f_L^{(k)} = \sum_{m''=1}^{M''} b_k + w_{km''} f_{(L-1)}^{(m'')} \left(... \sum_{m=1}^{M} b'_m + w_{m'm} f_1^{(m)} \left(\sum_{i-1}^{d} b_m + w_{mi} x_i \right) \right) \tag{3.8}$$

The $\boldsymbol{\sigma}(\cdot)$ function is called softmax and it is transforming the estimations to probabilities. The vectorized image $\boldsymbol{x}$ is called input layer and $\boldsymbol{f}(\boldsymbol{x})$ is called output layer. Every $\boldsymbol{f}_l$ is called the $l-$layer while every layer except for the input and the output are the hidden layers. The number of the layers are defining the depth of the network. It can be seen from the formalism that every $\boldsymbol{f}_l$ is a vector function and their dimensions can be different for each layer. So for example the layer 1 can have dimension M such that: $\boldsymbol{f}_1(\boldsymbol{x}) = (f_1^{(1)}(\boldsymbol{x}), f_1^{(2)}(\boldsymbol{x}), ..., f_1^{(M)}(\boldsymbol{x}))$ and the layer 2 can have dimension M' such that: $\boldsymbol{f}_2(\boldsymbol{x}') = (f_2^{(1)}(\boldsymbol{x}'), f_2^{(2)}(\boldsymbol{x}'), ..., f_2^{(M')}(\boldsymbol{x}'))$. Every basic function $f_l^{(m)}$ is called an activation function and is taking the weighted sum of the outputs of the previous layer plus a weight b (which is referred in the literature as ”bias”) and it returns a number in $\mathbb{R}$.

The activation functions are offering nonlinearity to the model and therefore the ability to describe more complex problems. Their role is crucial because a composition of linear functions is trivialized to a single linear function. For our problem, we will choose as our nonlinearity the rectified linear unit (ReLU), which is one of the most common activation functions, with formula:

$$ReLU(x) = \begin{cases} x & if \;\; x \geq 0 \\ 0 & if \;\; x < 0 \end{cases}$$

This architecture is called fully connected because every activation function $f_l^{(m)}$ is taking as input a weighted sum of the output of every activation function of the previous layer

$\boldsymbol{f}_{l-1}$. It also has learnable weights all the elements of $\boldsymbol{W}$ and the $\boldsymbol{b}$. So $|\boldsymbol{W}| + |\boldsymbol{b}| = (d \times M + M) + (M \times M' + M') + ... + (M'' \times K + K)$ weights have to be estimated. The weights can become too many quite fast in terms of depth, and the functions can be very flexible. Of course the bias error of those architectures is going to be really small, but the variance error is going to be really high. A strategy to overcome this issue is to imbue domain specific knowledge in order to discard some possible solutions that it is known they will be wrong. The first approach is to find a reasonable way to remove some weights.

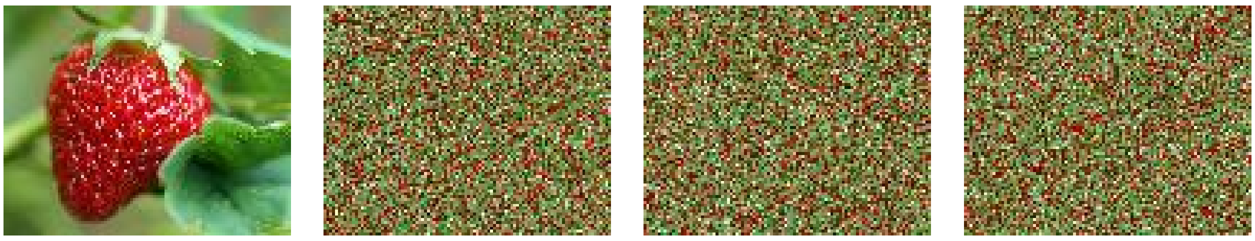

(a) A strawberry **(b)** Shuffle 1 **(c)** Shuffle 2 **(d)** Shuffle 3

Figure 3.1: In (a) we can see a strawberry from ImageNet [13]. In (b), (c) and (d) we can see three different shuffles of its pixels. The fully connected MLP can learn each representation of the image equally well if it was trained on a dataset with the same shuffling rule. This kind of visualization for the MNIST handwritten digit database [36] can be found in [2]

3.5 Adding Reasonable Bias to Fight Variance Error

3.5.1 Adding Architectural Bias

The rationality of [2] will be followed. The fully connected MLP is not taking into account any sense of neighboring between pixels. Let's discuss an image recognition problem in the feature space $\mathcal{X} = [0,1]^{100\times100}$. Then, for a given image there are going to be $10,000!$ ways to shuffle its pixels. Let's define the operator $Shuffle_i : \mathcal{X} \to \mathcal{X}$ with the identity $Shuffle_I(X) = X$ and for i from 1 to $10,000!$ that takes as input an image and applies the i^{th} shuffling.

Let's also define the datasets $D_i = Shuffle_i(D)$ as the datasets that their every element was applied the same i^{th} shuffling rule. Also, $D_I = Shuffle_I(D) = D$. Then, the fully connected multilayer perceptron can learn equally well all of those datasets. That is, if an MLP$_1$ was trained on D_1 and an MLP$_2$ was trained on D_2 with the same conditions, the only difference between them would be a variance declination due to the stochastic nature of non-convex optimization. Also, with MLP$_1$, somebody can achieve the exact same performance on the D_2 without train on it, but only by rearrange the weights of MLP$_1$. Of course the only useful problem is the one with D_I as dataset and the rest capabilities of the MLP are useless.

The fact that the fully connected MLP is capable of also solving that enormous number of useless problems is making it having high variance error in practice. The slower convergence of MLP in terms of data is its side effect for its capability of solving all those extra problems. By observing that, somebody is able to deny it that capability by fixating some of the weights to 0. More specifically, in each layer l, lets have one activation function for every single output of the layer $l - 1$. Now every activation function has an anchor pixel. For every activation function, lets fix to 0 every weight that corresponds to a pixel that is further from the anchor pixel more than an number of n pixels straight or diagonally. That would create $2n + 1$ by $2n + 1$ blocks of weights corresponding to neighboring pixel representations.

Up to here the number of weights were reduced dramatically by fixating to 0 every weight that was going to create a relationship for its corresponding pixel with a pixel that is not near. That is an assumption for the image domain which implies that structure is created strictly locally, which is a reasonable statement. That's why it is hoped that no potential was lost in terms of solving the problem using D_I. An even stricter assumption can be made now. Instead of leaving those $2n + 1$ by $2n + 1$ blocks of weights free for every activation function, it can be demanded that it is going to be the shame block of weights for the whole layer. That means that the weight w_1 corresponding to the top right block of weight of the the activation function $f_l^{(m)}$ is going to be the same variable w_1 with the one corresponding to the top right block of weights of the activation function $f_l^{(m+1)}$ and so on. That new jointly defined block of weights is called a convolution.

This assumption is useful not only because it minimizes the number of weights even more, but also because by creating this structure, it is the equivalent of shifting the weights over the previous layer, letting them search for features. Practically, the convolutions are filters like the hand-crafted ones used in classical image processing methods, but instead of searching manually for a filter that works for the specific task, here the weights are going to be learned to do the classification job optimally.

In the majority of the classical CNN architectures there is also a down-sampling procedure after most convolutions and activation functions that takes the representation and lowers its dimension by either returning the maximum or the average of a grid of its pixels. This procedure is called max pooling or average pooling respectively and its main goal is to make the representation on each layer more abstract and therefore to lower even more the variance related error. That's because it is making each representation robust to small translations and deformations.

3.5.2 Adding Dataset Bias

Even after making the architecture that stricter from the starting MLP, it is still way more flexible than the most of the traditional machine learning models partially because vision problems are way more complicated than most of the traditional problems in classical statistics. As mentioned in a previous section, one way to battle variance error is by using more data. So, the appearance of large scale datasets like the [13] played a very important role for deep neural networks architectures to be feasible. In addition to that,

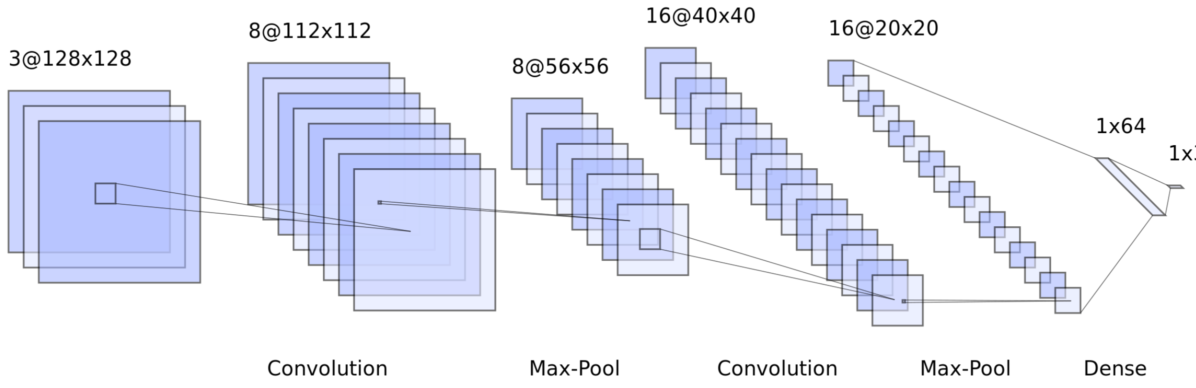


Figure 3.2: This is the architecture of a convolutional neural network. As we can see we do not have only one convolution per layer, but every convolution applied to the l layer is creating an additional channel to the $l+1$ layer. Also, for the creation of the dense layer we can either use one weight for each input pixel, or to use a global pooling method to minimize the weights. This diagram was created with the tool of [37] with the visualization style of [35].

some techniques were developed in order to make the datasets effectively even larger and those techniques are called data augmentations.

Data augmentations are procedures that use specific domain knowledge in order to enlarge the dataset, so in essence they add bias to the procedure. Which means that it is the same idea applied to the architecture but now it is applied to the dataset. It is the same principle: By adding bias, the variance error is shortened, but with the risk of raising the bias error. If the added bias is the ”right bias” then the bias error is not raising. Some of the most common augmentations are the inclusion of rotations of the images, translations of the images by random cropping and horizontal flips of the images. The last two augmentations were used in [32] and they achieved to increase the training set by a factor of 2048.

Of course those augmentations are the right ones because they were training on the ImageNet and a horizontal flip of a ”dog” is still a ”dog”. This couldn’t be done for the MNIST handwritten digit database because a horizontal flip of the digit ”3” is not a ”3” anymore. So in the first example it is expected that the bias inclusion is not going to raise the bias error, while in the second example it is expected that it will. In both cases though, it is expected for the variance error to shorten.

3.5.3 Adding Bias to the Weights

Another way to add bias to the procedure, for a given architecture and a given augmented dataset, is to add prior knowledge directly to the weights. There are a lot of techniques in the literature that display random initializations in order for the model to be optimized easily and without technical issues. Some of them, indicatively are in [22, 26]. Those techniques are adding some bias in order to decrease the variance due to initialization, which is one of the sources of variance error in neural networks according to [44].

Another way to do this is by using the weights of a pre-trained network as initialization. Of course this pre-trained network must have been trained on a domain that is relevant. This is adding bias because it is assumed that the solution is closer between models of close domains than a model of one domain and a random model. So it is expected that a model initialized closer to its solution is going to converge faster.

Another approach for adding bias to the weights is to tackle the problem in the Bayesian framework. That is done by treating the weights $\boldsymbol{W}$ as random variables and assume prior distributions for them. Then instead of learning them point-wise, their distribution is estimated. In the Bayesian framework somebody want to calculate the densities:

$$g(\boldsymbol{W}|D) = \frac{g(D|\boldsymbol{W})g(\boldsymbol{W})}{\int\limits_{\boldsymbol{W}} g(D|\boldsymbol{W})g(\boldsymbol{W})d\boldsymbol{W}} = \frac{\mathcal{L}(\boldsymbol{W})g(\boldsymbol{W})}{\int\limits_{\boldsymbol{W}} \mathcal{L}(\boldsymbol{W})g(\boldsymbol{W})d\boldsymbol{W}} \tag{3.9}$$

The density $g(\boldsymbol{W}|D)$ is called the posterior or a posteriori distribution because, as the conditional symbol implies, it is the idea someone has for the distribution of $\boldsymbol{W}$ given that the dataset D is observed. The density $g(\boldsymbol{W})$ is called prior or a priori distribution because it is the idea of the distribution of $\boldsymbol{W}$ before the dataset is observed. Finally, $g(D|\boldsymbol{W})$ is the likelihood function. Estimating this density can be really hard and usually the assumption of independence of the w_i is made. But even then, in a lot of situations the likelihood function multiplied with the prior is not returning a known distribution. In those cases, the integral of the denominator, which is the normalization constant of the density of the nominator, cannot be calculated analytically. If the posterior follows the same distribution with the prior, it is said that the likelihood is conjugate with the prior.

A way to bypass those problems is to follow a semi-Bayesian procedure by maximizing the posterior. The estimators resulting from this procedure are called maximum a posteriori (MAP) estimators. So instead of the MLE somebody can optimize:

$$\underset{\boldsymbol{W}}{\text{argmax}}\, g(\boldsymbol{W}|D) = \underset{\boldsymbol{W}}{\text{argmax}} \frac{\mathcal{L}(\boldsymbol{W})g(\boldsymbol{W})}{\int\limits_{\boldsymbol{W}} \mathcal{L}(\boldsymbol{W})g(\boldsymbol{W})d\boldsymbol{W}} = \underset{\boldsymbol{W}}{\text{argmax}}\, \mathcal{L}(\boldsymbol{W})g(\boldsymbol{W}) \tag{3.10}$$

$$= \underset{\boldsymbol{W}}{\text{argmax}} \prod_{i=1}^{N}\prod_{k=1}^{K} {f_{ik}}^{y_{ik}} g(\boldsymbol{W}) \tag{3.11}$$

$$= \underset{\boldsymbol{W}}{\text{argmax}} \sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} log f_{ik} + log(g(\boldsymbol{W})) \tag{3.12}$$

$$= \underset{\boldsymbol{W}}{\text{argmin}} -\frac{1}{N}\left[\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} log f_{ik} + log(g(\boldsymbol{W}))\right] \tag{3.13}$$

$$= \underset{\boldsymbol{W}}{\text{argmin}} \frac{1}{N} \sum_{i=1}^{N} \left[\sum_{k=1}^{K} -y_{ik} log f_{ik} + \sum_{k=1}^{K} y_{ik} log y_{ik} - \sum_{k=1}^{K} log(g(\boldsymbol{W})) \right] \quad (3.14)$$

$$= \underset{\boldsymbol{W}}{\text{argmin}} \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} log \frac{y_{ik}}{f_{ik}} - log(g(\boldsymbol{W})) \quad (3.15)$$

The above procedure is equivalent with adding a regularization term in MLE. Regularization can be seen as adding the concept of Occam's razor to the problem, which, according to [23], as a principle states that:

"Among competing hypotheses that explain known observations equally well,
we should choose the simplest one"

So, regularization's essence is adding to the model -in the form of bias- the belief that a simpler solution in terms of the values $\boldsymbol{W}$ is going to be a more general one, and therefore it will generalize better on unseen data.

The most common regularization technique is the weight decay, which is done by adding the term $\lambda \|\boldsymbol{W}\|_2^2$ to the minimization procedure. This way the model is constrained not only to perform well but to do it closer to the origin. This regularization is also known as L^2 regularization or ridge regression [28]. According to [45], this regularization can also be constructed by making the Gaussian assumption to the MAP estimator, because:

$$-log(g(\boldsymbol{W})) = \frac{1}{2b^2} \|\boldsymbol{W}\|_2^2 + constant$$

For $\lambda = \frac{1}{2b^2}$ and the constant is irrelevant to the optimization procedure.

A second way to penaltize the weights is to add the sum of the absolute values of the weights $\lambda \|\boldsymbol{W}\|_1$ to the minimization procedure. Which causes some of the weights to become 0 and therefore it is playing a role of feature selection in the procedure. This regularizer is also known as L^1 regularization or least absolute shrinkage and selection operator (LASSO) [65]. This can be seen as a special case of MAP too, by using independent mean zero Laplace priors [48].

A combination of the L^1 and L^2 regularizations is called elastic net [77] and it is done by adding $\lambda_2 \|\boldsymbol{W}\|_2^2 + \lambda_1 \|\boldsymbol{W}\|_1$ to the minimization procedure.

In [27] dropout was introduced. The goal of this regularization is to prevent complex co-adaptations between activation functions. That means that it tackles the problem of some activations being useful only because of the existence of other activations. This is done by setting to zero the output of each activation for a single optimization step with probability 0.5. That way the network can be seen as a set of networks that all of them can do the classification job separately, while in the end the whole architecture normalized with 0.5 will be used. Doing this also forces the network to use all neurons to do the classification task instead of being based to a subset of them.

3.6 Adam Optimizer

The use of very flexible functions in order to solve more and more complicated problems led to non convex loss functions, and therefore the convex optimization guarantees for the global optimum solutions are abandoned. Of course, those problems can be tackled only with iterative optimization techniques. Additionally, the dimensionality of those problems is so high that the approaches are limited to the first-order methods. Even for those though, the calculation of the first derivative is not trivial and a dynamic procedure based on the chain rule had to be used in order for the neural networks to be commonly accepted as feasible methods. This procedure is called backpropagation, invented or improved independently by [42, 31, 6, 16] and popularized for the specific task by [56]. Another limitation is caused by the huge volume of the data needed in the deep learning practice, which makes it impossible for the iterative methods to take the whole dataset into account in every iteration. Following [55], the simpler optimization method that is feasible is the mini-batch gradient descent with batch size n:

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} J(\theta_t; x^{(i:i+n)}; y^{(i:i+n)}) = \theta_t - \eta \nabla_{\theta_t} J(\theta_t) = \theta_t - \eta g_t \tag{3.16}$$

There are cases where the gradient descent is affected by dimensions that are contributing way more to the gradient than others, making the procedure to follow a reciprocating trajectory. Also, in the case of the mini-batch version there is an additional difficulty: Each batch is creating an approximation of the loss function in every step of the procedure, so there is a need for robustness across the iterations. Finally a strategy to surpass saddle points is important. For those reasons momentum was introduced in the learning problems [56]. The updating formula for the stochastic gradient descent with momentum is:

$$\begin{aligned} u_{t+1} &= \gamma u_t + \eta g_t \\ \theta_t &= \theta_{t-1} - u_t \end{aligned} \tag{3.17}$$

It is also useful to adapt the learning rate to each dimension separately, giving bigger learning rates to dimensions that are contributing less to the gradient. That is because it is a good idea to learn faster a dimension that is not going to converge soon. The methods using this idea are called adaptive methods and one of the most famous is the root mean square propagation (RMSprop) [66], with the updating formula:

$$\begin{aligned} \mathbb{E}[g^2]_t &= \gamma \mathbb{E}[g^2]_{t-1} + (1-\gamma) g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\mathbb{E}[g^2]_t + \epsilon}} g_t \end{aligned} \tag{3.18}$$

A method that combines the adaptive and the momentum idea is Adam, which derives from ”adaptive moments”. According to [23], in Adam, the momentum is resulting directly from the first-order method of moments of the expectation of the gradient. It also keeps the second-order method of moments by averaging the squared gradient like RMSProp:

$$\begin{aligned} m_t &= \beta_1 m_t + (1-\beta_1) g_t \\ u_t &= \beta_2 u_{t-1} + (1-\beta_2) g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{u}_t} + \epsilon} \hat{m}_t \end{aligned} \tag{3.19}$$

With $\hat{m}_t,\ \hat{u}_t$ the corrected unbiased estimators corresponding to $m_t,\ u_t$:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t},\ \ \hat{u}_t = \frac{u_t}{1-\beta_2^t}$$

3.7 The ResNet Architectures

According to [25], the deep neural networks, when they are upgraded to a deeper architecture, have the ability to collapse to their shallower versions if needed. That is, if the shallow architecture was the ”right” one for the problem, the newly added layers can theoretically converge to be the identities and therefore the architecture can be transformed to the shallow one. The problem in practice is that it is hard for a model to converge to such an ideal solution. This problem is tackled by the residual block [25]. A residual block consists of a number of sequential convolutional layers and activation functions, but the input is added to the output in the end. So if those sequential layers are a mapping $\mathcal{F}(x)$, now the output of this block is going to be $\mathcal{F}(x) + x$. That way, if the identity function is the optimum for those layers, it is going to be easier for the procedure to fit the $\mathcal{F}(x) = 0$ instead if fitting an identity function using a stack of nonlinear layers.

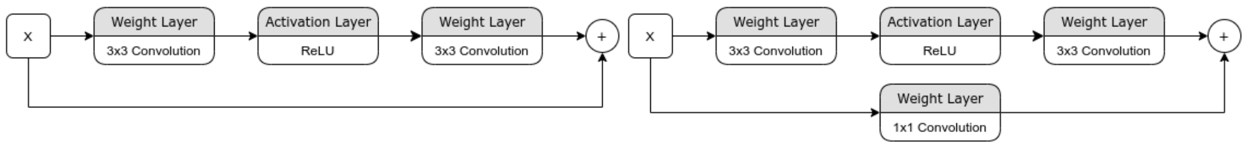


Figure 3.3: On this figures we can see two kinds of residual blocks. The first one (left) is used when $\mathcal{F}(x),\ x$ are having the same dimensions and the second one (right) is using a 1x1 convolution as a dimension corrector.

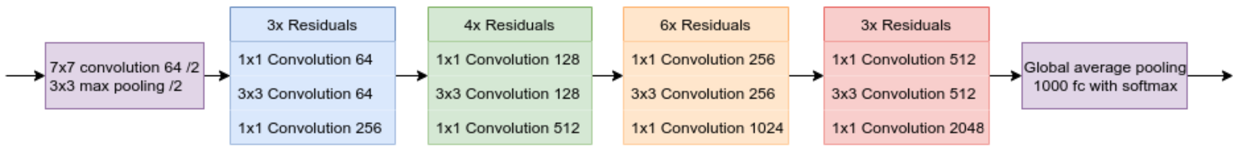


Figure 3.4: The ResNet-50 architecture introduced in [25]. It is composed of 50 layers: 16 residual blocks with three layers each, a 7x7 convolution and a fully connected layer.

For our problem we used the ResNet-50 architecture which consists of 16 residual blocks as we can see in the Figure 3.4. It starts with a 7x7 convolution and after the residual blocks it is flattening the results for the classifier with global average pooling (GAP). GAP is giving us the convenience not to be very strict with the input size and also it gives our model translation invariance properties.

4. METHOD

4.1 Overview

As we described in the introduction, our approach is to apply a function $f_1(\boldsymbol{x}_{D_S}) \sim \boldsymbol{X}'_S$ to our realizations from the image domain and a function $f_2(\boldsymbol{x}_{D_T}) \sim \boldsymbol{X}'_T$ to our realizations from the sketch domain such that the assumption $P(Y|\boldsymbol{X}'_S) \approx P(Y|\boldsymbol{X'}_T)$ is reasonable. We chose for the f_1, f_2 to be a composition of the following functions:

$$\begin{aligned} \boldsymbol{f}_1(\boldsymbol{x}_{D_S}) &= F_{Thin}\left(\tilde{\boldsymbol{F}}_{Thresh}\left(\boldsymbol{F}_{Edge}\left(\boldsymbol{x}_{D_S}\right)\right)\right) \\ f_2(\boldsymbol{x}_{D_T}) &= F_{Thin}\left(F_{Thresh}\left(\mathbf{1} - \boldsymbol{x}_{D_T}\right)\right) \end{aligned} \tag{4.1}$$

In contrast to f_2, $\boldsymbol{f}_1$ and some of its components are in bold. That's because, even for a given constant realization $\boldsymbol{x}_{D_S}$, $\boldsymbol{f}_1$ is a random variable. So, in addition to the domain bridging, $\boldsymbol{f}_1$ is also a data augmentation in nature. The corresponding components of f_2 on the other hand are all normal functions, because we want our testing to be deterministic.

The function $\boldsymbol{F}_{Edge}$ is a random variable and it is calculating a randomized edge map for a given image. More formally, it is defined as:

$$\begin{aligned} &\boldsymbol{F}_{Edge} : [0,1]^{w\times h\times 3} \rightarrow [0,1]^{w\times h} \\ &\boldsymbol{F}_{Edge}(\boldsymbol{x}) = \mathbb{1}_{[\alpha=i]} F_{Edge_{(i)}}(\boldsymbol{x}),\ i = 1,2,...,A \\ &a \sim Cat(A, \boldsymbol{p}),\ with\ p_i = \frac{1}{A} \end{aligned}$$

The function $\tilde{\boldsymbol{F}}_{Thresh}$ is also a random variable. It is taking an edge map and it calculates its binarized equivalent by applying a random thresholding method. For data augmentation reasons we used a Gaussian perturbation to the values of each thresholder. Formally, $\tilde{\boldsymbol{F}}_{Thresh}$ is defined as:

$$\begin{aligned} &\tilde{\boldsymbol{F}}_{Thresh} : [0,1]^{w\times h} \rightarrow \{0,1\}^{w\times h} \\ &\tilde{\boldsymbol{F}}_{Thresh}(\boldsymbol{x}) = \mathbb{1}_{[\alpha=i]}\left[F_{Thresh(i)}(\boldsymbol{x}, b)\right],\ i = 1,2,...,A' \\ &a \sim Cat(A', \boldsymbol{p}),\ with\ p_i = \frac{1}{A'}\ and\ b \sim N(0, \sigma^2) \end{aligned}$$

F_{Thresh} and F_{Thin} on the other hand are normal functions:

$$\begin{aligned} F_{Thresh} &: [0,1]^{w\times h} \rightarrow \{0,1\}^{w\times h} \\ F_{Thin} &: \{0,1\}^{w\times h} \rightarrow \{0,1\}^{w\times h} \end{aligned}$$

4.1.1 Edge Detection

As we can understand from the components of 4.1, the first level of functions applied to our data is about edge detection. That is because in most of the sketch domain every line drawn, and therefore every information of the sketch, is an effort to describe an edge of the object. This step is essential because it brings our two domains closer by representing their elements using the edges as the main visual cue. In this step we are abandoning the color, the shape and the texture of the image domain as unreliable sources of information regarding the sketch domain.

Edge Detection for Training

We are applying the edge detection function in the f_1. We symbolized the $\boldsymbol{F}_{Edge}(\boldsymbol{x}_{D_S})$ with bold because, -as we already mentioned- even for a given realization $\boldsymbol{x}_{D_S}$, it is still a random variable with five possible outcomes: It applies to the input image with equal probabilities (a) the structured forests for fast edge detection or Dollar edge detection [14], (b) the holistically-nested edge detection (HED) [71], (c) the bi-directional cascade network edge detection (BDCN) [24], (d) the pixel-wise minimum operation of the results of the three edge detectors or (e) the pixel-wise multiplication of the results of the three edge detectors.

We chose to use three different edge detectors because each detector is possible to detect different features and therefore to provide to the problem with new information. It is worth mentioning that all three edge detectors were trained in the Berkeley segmentation data set and benchmarks 500 (BSDS500) [1], which is a segmentation dataset with only 300 images for training and 200 for testing. The fact that all three edge detectors were trained in the same dataset and also the extremely low volume of the specific dataset is keeping the involved non image-domain data of our procedure to the minimum. Both HED and BDCN edge detectors are using a pre-trained visual geometry group network (VGG16) [61] on ImageNet but that is not increasing the data usage because we are anyway using a pre-trained ResNet-50 on ImageNet as an initialization for our learning procedure. The indirect use of the BSDS500 dataset can cause ambiguity in terms of the zero-shot nature of our work, because it is a segmentation dataset and therefore not plain images were used. Our argument is that the segmentation labels are not sketches and also the BSDS500 dataset is class agnostic, so its use is not affecting the generality of use of our approach. Specifically, for the three edge detectors:

(a) The structured forests edge detector was one of the best in terms of performance in the pre deep learning era. It is using the random forests ensemble and achieved an optimal dataset score (ODS) F-measure of 0.75 on BSDS500. We used the official GitHub implementation[1].

[1]`https://github.com/pdollar/edges`

(b) The HED is using a deep architecture with multiple stages using different strides in order to take into account the scaling of the edge map. The chosen architecture was the VGG16 without its last stage, including the fifth pooling and also they connected a side output layer to the last convolutional layer of each stage so the network will have supervision on every stage. HED achieved an ODS F-measure of 0.782 on BSDS500. We used a PyTorch re-implementation[2].

(c) The BDCN is introducing the idea to use supervision to the representations in each layer at the scale of the representation in order to learn the data in multiple scales. Also they introduce a scale enhancement module (SEM) with dilated convolutions to improve the multi-scale learning. The most general blocks in this architecture are called incremental detection blocks and every one of them is trained with supervision of different scale by a cascade structure. The 3x3 convolutions in the incremental detection blocks are the pre-trained convolutions on ImageNet of the VGG16. This architecture achieved an ODS F-measure of 0.828 on BSDS500, which surpasses the human benchmark with ODS F-measure of 0.803. We used the official GitHub implementation[3].

So for the training, for every image, we randomly apply the Dollar, the HED, the BDCN, the element-wise multiplication of the three or the element-wise minimum of the three, multiplying our data volume by a factor of 5.

Edge Detection for Testing

The reason the edge detection function is missing from f_2 is that we assumed that each sketch is an edge map already. The only thing we need to say about f_2 in this step, is that we inverted the meaning of the pixels of the sketches, because we wanted to encode with 0 -and therefore black- the absence of information and with 1 -and therefore white- the presence of information in terms of edges drawn. This conversion is already effective in almost every edge detector so the function $\boldsymbol{F}_{Edge}(\boldsymbol{x}_{D_S})$ implies it. This semantic inversion of the pixels of the sketch domain can be seen as a trivial edge map conversion.

4.1.2 Thresholding

With the thresholding functions we want to find the optimum way for the binarization of the image. So, the threshold is the procedure with which we are going to replace every ”small” value with 0 and every ”big” with 1. We want to do this because the majority of sketches have no tones. Even if some of them had though, by downgrade them we are sacrificing some information in order to build a more general procedure. In this step we are restricting our two domains to have the same stricter (binarized) feature space but without changing

[2]`https://github.com/sniklaus/pytorch-hed`
[3]`https://github.com/pkuCactus/BDCN`

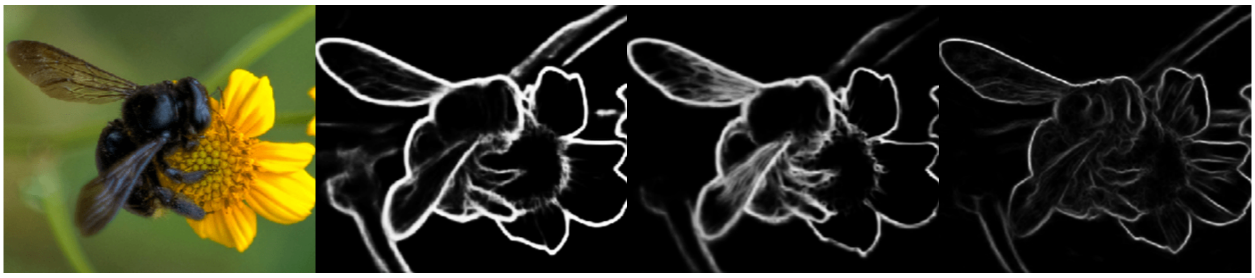


Figure 4.1: In this figure in (a) we can see an image of a bee from ImageNet. In (b), (c) and (d) we can see the three edge maps as calculated from BDCN, HED and Dollar edge detectors respectively.

the sketch domain that much. In this step we are abandoning the tone cues of the edge map domain as unreliable source of information regarding the sketch domain, making the edge our sole visual cue.

Thresholding for Training

For the $\boldsymbol{f}_1(\boldsymbol{x})$, like with the case of $\boldsymbol{F}_{Edge}$, even for a fixed realization $\boldsymbol{x}$ the $\tilde{\boldsymbol{F}}_{Thresh}$ is also a random variable that takes an edge map and applies randomly with equal probabilities one of the seven following thresholding techniques implemented in [68]: (a) The Otsu thresholder [46], (b) the Yen thresholder [30], (c) the Mean thresholder [21], (d) the Li thresholder [38], (e) the IsoData thresholder [67], (f) the Sauvola thresholder [59] or (g) the Local Mean thresholder [11].

The $(a) - (e)$ are global thresholders, which means that they calculate one threshold for each image and they replace every element that is smaller than this threshold with 0 and all others with 1. More specifically:

(a) The Otsu thresholder is trying exhaustively all possible threshold values and then it calculates for all of them their intra-class variance (or equivalently their inter-class variance). Then it choose the threshold that minimizes the intra-class variance (or equivalently the one that maximizes the inter-class variance). This method is the one dimensional discrete -in terms of feature space- and binary -in terms of classes- analog of the Fisher's linear discriminant analysis (LDA) [18].

(b) The Yen thresholder is defining a criterion that takes into account two things. The first one is the number of bits needed to describe the resulted image. The second one is the discrepancy between the thresholded and original image, which is defined based on a maximum correlation criterion from [19]. Then it combines those two factors in a single loss function which is minimized.

(c) The Mean thresholder is setting as threshold the mean value of the elements of the image.

(d) The Li thresholder is calculating the threshold as the number that minimizes the cross-entropy of the input image from the thresholded image. Also in [38], where the Li thresholder is proposed, it is also shown that the Otsu thresholder is minimizing the mean square error of the two.

(e) The IsoData method, also known as Ridler-Calvard or inter-means method, is exhaustively looking for the threshold that divides the foreground (values that are going to be replaced with 1) and the background (values that are going to be replaced with zero) such that the threshold value is equidistant from their means.

The $(f) - (g)$ are called local or adaptive thresholders. Those methods are deciding about the value of each pixel by using information from its neighborhood. The neighborhood is defined by using a filter of predetermined size. Those methods are preferred in cases of uneven lighting in an image.

(f) The Sauvola thresholder is calculating the threshold in each filter with the formula $T = m(x, y)\left[1 + k\left(\frac{s(x, y)}{R} - 1\right)\right]$ where the $m(x, y)$ is the mean and the $s(x, y)$ is the standard deviation of the pixels in the filter. The k is a hyperparameter and the R is the maximum standard deviation of the image.

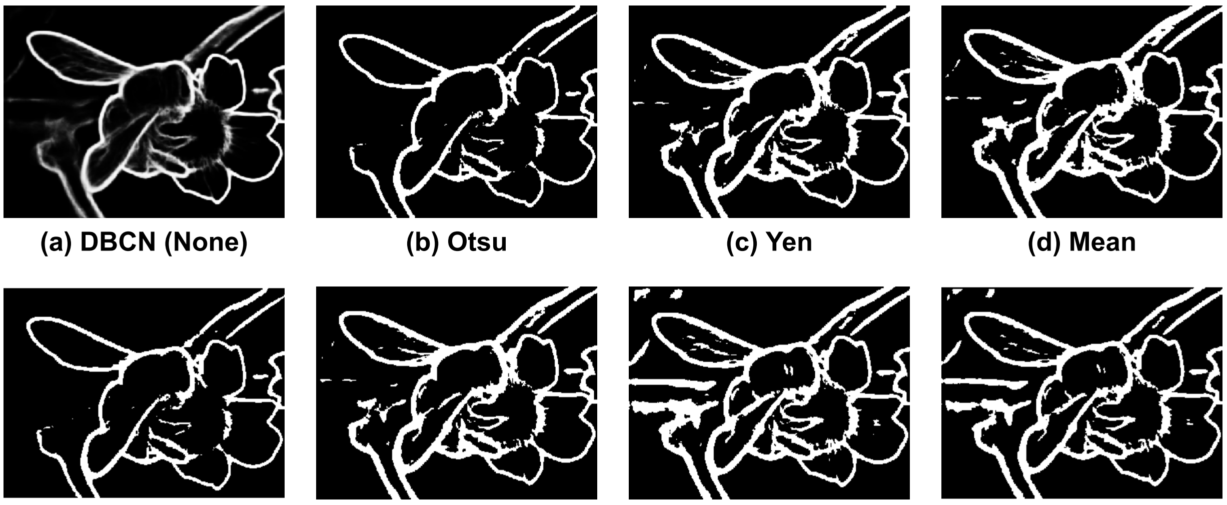
(g) The Local Mean thresholder is the local version of the mean thresholder. It sets the threshold in each filter to be the mean value of the pixels of the filter.

We used all those different thresholders in order to augment our data believing, in the same time, that every thresholder can provide to our problem with extra information. Up to now, by combining the different thresholders with the different edge detectors we enlarged our dataset effectively 35 times.

In order to augment our data more we chose to add a Gaussian perturbation on the values of the thresholders. The choice of the variance of the Gaussian was made qualitatively by studying the distributions of the values of each thresholder when applied to our whole dataset. As we can see in Figure 4.3 we visualized the total distribution by adding the distributions of every global thresholder and then we applied a perturbation that is powerful enough to augment our data but not too powerful to change completely the form of the distribution. The preservation of the distribution's shape has value because in the threshold of 75 for example we have a very small likelihood. That means that almost all thresholders agree that there is no additional useful information between the thresholds 55-85. The information peaks are around 40 and then around 100. If we destroyed the form of the distribution and we randomly used thresholds around the value of 75 we would just have added noise without gaining additional information.

Thresholding for Testing

The F_{Thresh} function in f_2 is written for completeness because, specifically the TU-Berlin dataset is already binary. If it was not we would choose strictly one of the aforementioned



(a) DBCN (None) (b) Otsu (c) Yen (d) Mean

(e) IsoData (f) Li (g) Sauvola (h) Local Mean

Figure 4.2: In this figure we can see the seven thresholders applied to the same DBCN edge map.

thresholding procedures we described for the f_1 function but as we can see from the 4.1 we would not apply the corresponding perturbation.

4.1.3 Thinning

With the F_{thin} function we want to eliminate the differences in terms of thickness between our binarized edge maps and our sketches. To do this we will set the thickness of each edge to one pixel with a morphological procedure called thinning or skeletonization. By doing this we did not only set up the edge as the sole cue, but also we ensured that every edge is represented in equal terms.

We are applying the same algorithm of [75] both in training and testing. This method is iteratively removing pixels from the borders of an object with the rule that their removal cannot break the connectivity of the object. This is possible by using a 3×3 filter and on each step the pixel in the middle is studied in terms of its role according to its neighbors. Every iteration in this method is divided into two subiterations. In the first subiteration considers to remove only the pixels located on the bottom or right of the filter, while in the second subiteration only the pixels located on the top or left of the filter are considered for removal. That way, if the first subiteration leaves only the skeleton of the image, the second subiteration will not delete it. This is repeated until there are no more pixels to be removed. We used the skeletonization function implemented in [68].

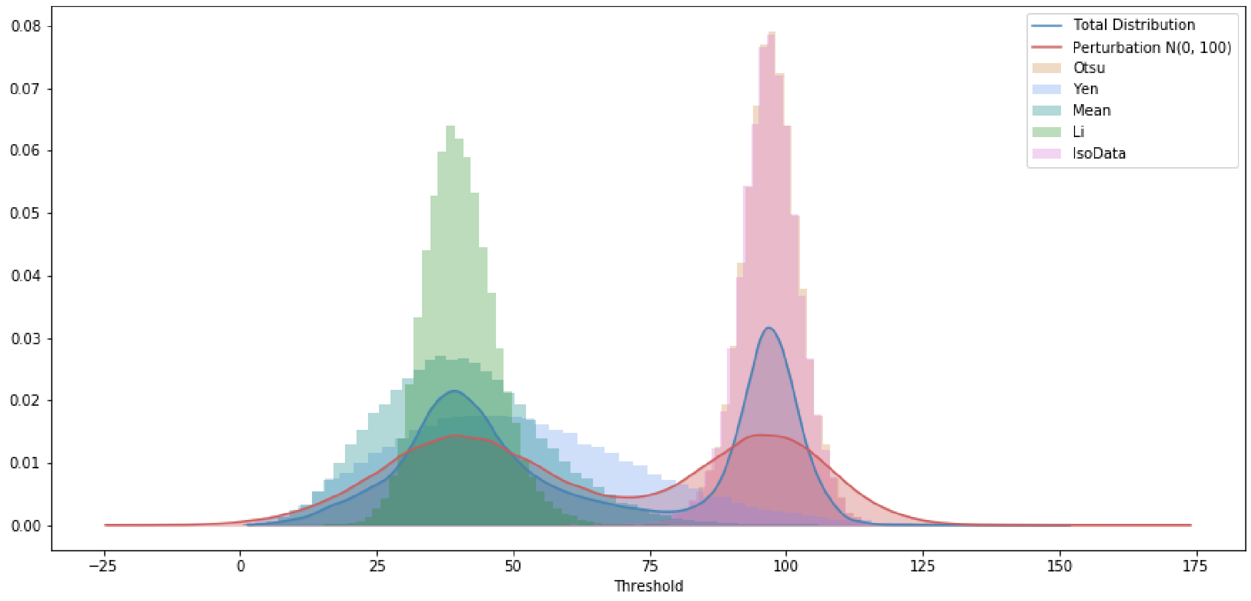


Figure 4.3: In this figure we can see the distributions of the values of all the global thresholders when they were applied to the entire BDCN edge maps of our training data. We can also see the cumulative distribution and the perturbation of our choice that is powerful enough to make an impact but keeps the structure of the total distribution.

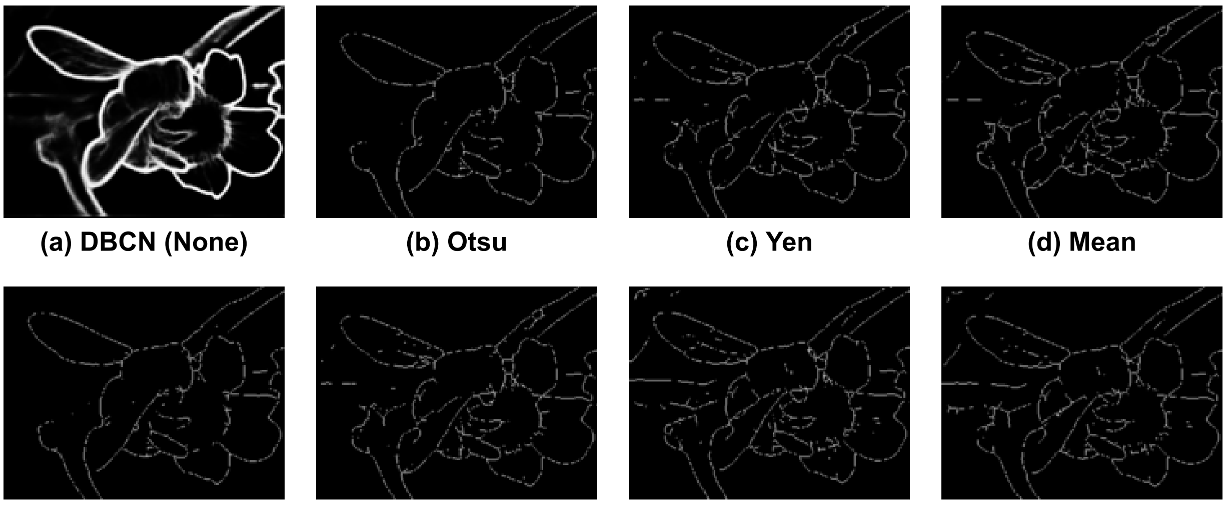
4.2 Datasets

In our work we used two datasets: The TU-Berlin [17] sketch dataset with over $20,000$ non-professional sketches partitioned in 250 classes and the ILSVRC2012 version of ImageNet [13] with 1.2 million training images and $150,000$ validation images partitioned in $1,000$ classes. In order to make our datasets comparable and therefore to relate them, we kept only the classes that belong simultaneously to the lexical and the ontological intersection of the two datasets. That way we concluded to the 47 classes shown in the Figure A.1. From now on we will call those subsets of the datasets as $ImageNet^{47}$ and $TU-Berlin^{47}$.

We used the train set of the $ImageNet^{47}$ for training, the validation set of the $ImageNet^{47}$ for validation and the whole $TU-Berlin^{47}$ for testing.

In contrast with the thinning, thresholding and perturbation procedures of 4.1 that we implemented them as augmentations, the edge detection is too computationally expensive to be included this way, so we run all three edge detectors for the whole $ImageNet^{47}$. We saved for every image each edge map in one of the RGB channels and this way we created the $ImageNet^{47}_{Edge}$ dataset. That way we achieved two things: First, each image from $ImageNet^{47}_{Edge}$ is a visualization of the differences of the three edge detectors and second, we have a very convenient way to read the data. To save some extra computational time we saved and we will call $TU-Berlin^{47}_{Edge}$ the thinned version of the inverted $TU-Berlin^{47}$ as described in the edge detection section.

Finally, in order to see the potential of the domain bridging, we manually chose six subsets

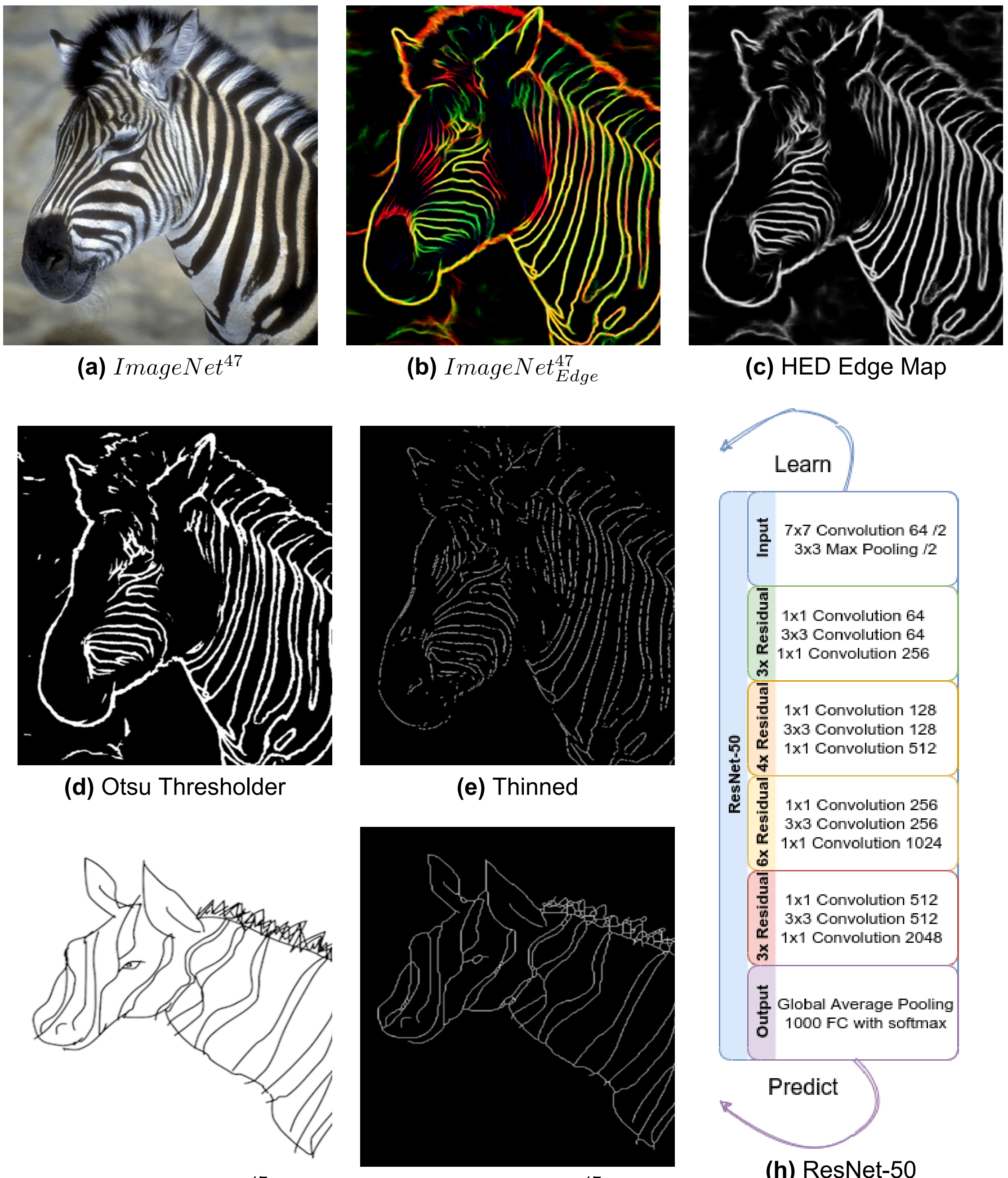


(a) DBCN (None) **(b) Otsu** **(c) Yen** **(d) Mean**

(e) IsoData **(f) Li** **(g) Sauvola** **(h) Local Mean**

Figure 4.4: In this figure we can see the thinned version of the seven thresholders applied to the same DBCN edge map.

of the $ImageNet^{47}_{Edge}$ dataset in terms of cleanliness. We selected the 4, 8, 12, 16, 20 and 24 subjectively best images per class and we created the $Cleaned^{04}ImageNet^{47}_{Edge}$ - $Cleaned^{24}ImageNet^{47}_{Edge}$ datasets respectively.



(a) $ImageNet^{47}$ **(b)** $ImageNet^{47}_{Edge}$ **(c)** HED Edge Map

(d) Otsu Thresholder **(e)** Thinned **(h)** ResNet-50

(f) $TU - Berlin^{47}$ **(g)** $TU - Berlin^{47}_{Edge}$

Figure 4.5: In this figure we can see our whole approach. In (a) we can see an image from $ImageNet^{47}$. In (b) we can see the corresponding image of $ImageNet^{47}_{Edge}$. In (c) we randomly selected the second channel of (b), which is the HED edge map. In (d) we randomly selected the Otsu thresholder of (c). In (e) we skeletonized. In (f) we can see a sketch from the $TU - Berlin^{47}$ dataset. In (g) we can see a sketch from the $TU - Berlin^{47}_{Edge}$ dataset. As we can see it is quite close to the (e) representation. In (e) and (g) we can see elements from $f_1(x_{D_S})$ and $f_2(x_{D_T})$ respectively. Finally in (h) the ResNet-50 will learn the (e) and predict the (g).

4.3 Learning

In terms of the learning procedure, we used the PyTorch framework [49] in which we trained the ImageNet pre-trained ResNet-50 architecture from the Torchvision library [43] with a modified output layer to fit our 47 classes.

The images before reaching the network were transformed into a square aspect ratio by padding the smallest dimension. That's because some classes in $ImageNet^{47}_{Edge}$ had some extreme aspect ratios and the edges of the objects were too close to the boundaries of the image, making it hard for the convolution filters to extract useful information. After that the images were resized to 256×256. Then we added the literature standard augmentations: Random rotation of $[-5, 5]$ degrees, random resized crop to 224×224 with random scale in $[0.8, 1]$ and a random horizontal flip with probability of 0.5. Then the augmentations described in the overview section were applied.

We used the Adam optimizer with the default hyperparameters $(0.9, 0.999)$ for the β_1, β_2 of 3.19. We optimized the cross entropy loss function described in 3.4. Our realistic batch size range was 32-128. Less than 32 was too time consuming while more than 128 was impossible due to memory limitations. After some experimentation we concluded that there was no significant difference due to batch size and we settled for a batch size of 64.

The learning rate was tested for fixed other hyperparameters. We run some epochs starting from a way too small learning rate and used a scheduler to multiply the learning rate by 10 every 100 iterations. After that we studied the behavior of the loss, accuracy and L_2 norm of the gradient of the first convolution for each iteration and learning rate. We wanted to see the learning rates that caused ascending accuracy, descending loss and a reliable L_2 norm of the gradient. That is an L_2 norm of the gradient that shows no signs of future exploding gradient or vanishing gradient. Then we decided to set the learning rate with the higher acceptable value in terms of those conditions. For the training, we set the learning rate to 10^{-4} to have the maximum possible learning rate for the beginning. We set the scheduler to divide it by 10 every 140 epochs. We saw in the diagram that for the first two learning rates we would have a learning behavior. We also run it for 20 extra epochs with a learning rate of 10^{-6}, for a total of 300 epochs, in order to learn the details of the hyperspace.

Every experiment we made with the $ImageNet^{47}_{Edge}$ was trained for a fixed 300 epochs, that's because we couldn't trust our validation set in terms of a possible earlier stopping, because of the domain difference from our test set. We used our validation set though to monitor the possibility of overfitting. We kept the epoch number fixed in order to have comparable results between different experiments with the $ImageNet^{47}_{Edge}$ dataset.

In order to be consistent with the experiments between the $Cleaned^{04}ImageNet^{47}_{Edge}$ - $Cleaned^{24}ImageNet^{47}_{Edge}$ datasets instead of keeping the the epoch fixed, we kept the iteration number fixed. That's because there would be a huge difference between the iteration number of $Cleaned^{04}ImageNet^{47}_{Edge}$ and $Cleaned^{24}ImageNet^{47}_{Edge}$ for a given epoch number due to the data size difference. We wanted to be sure that the procedures would converge both statistically -in terms of augmented data- and optimization-wise -in terms of iterations- so we chose a fixed 5.400 iterations. That corresponds to 300 epochs for the $Cleaned^{24}ImageNet^{47}_{Edge}$ dataset, 1.805 epochs for the $Cleaned^{04}ImageNet^{47}_{Edge}$ dataset and for the rest between this range. In all cases we set the scheduler to divide the learning rate by 10 every $\frac{MaxEpoch}{3}$ epochs.

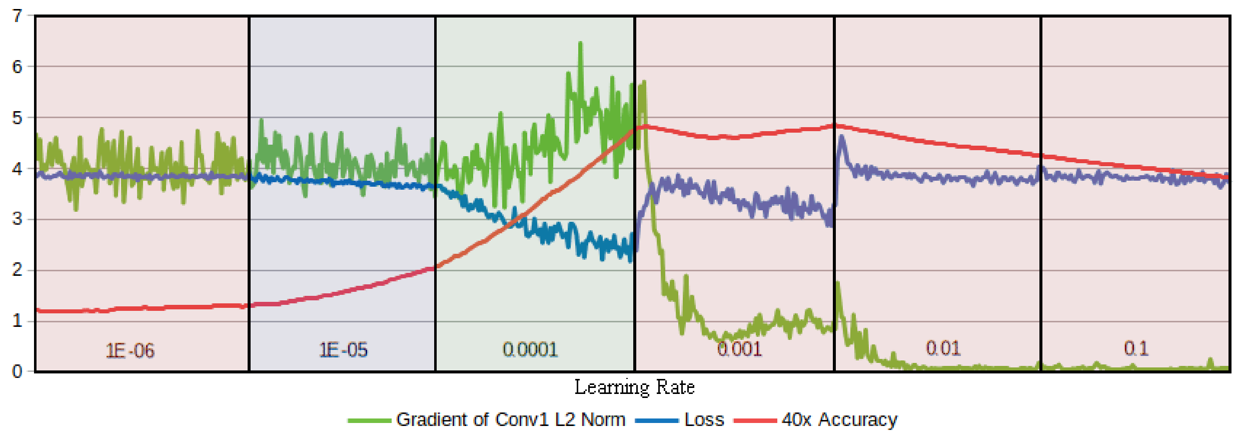


Figure 4.6: In this figure we can see the gradient of the L_2 norm of our model, the loss and a $40\times$ enhanced accuracy for ascending learning rates in order to find the learning rate that shows a better learning behavior. We chose to train the model with a starting learning rate of 10^{-4} for 140 epochs, then with a learning rate of 10^{-5} for another 140 epochs and finally with a learning rate of 10^{-6} for 20 epochs.

5. BASELINE

In order to be able to compare our results with a reference point, we decided to create a baseline. We trained using the same architecture, loss function and optimizer directly to the $TU-Berlin^{47}_{edge}$ for different sketches per class. That way a statistical convergence plot was created and every approach of cross-domain learning that learned without observing a sketch can be translated to an equivalent number of sketches.

We trained on 14 different versions of subsets of the $TU - Berlin^{47}_{edge}$, each one of them had different sketches per class. We started by training on 2, 3, 4, 5, 6, 7 and 8 sketches per class and then we progressed in eights: 16, 24, 32, 40, 48, 56, 64 sketches per class. In order to be precise, on every one of them we used a 5-fold cross-validation by partitioning the whole $TU - Berlin^{47}_{Edge}$ with the rule that the union of all the five test sets of the five folds is going to be the whole $TU - Berlin^{47}_{Edge}$. That's because we wanted for the results of the baseline to be directly comparable with the results our methods. The training sets of each fold were selected randomly.

For fairness, we run each 5-fold two times. In the first one, we divided the training sets into training and validation subsets with a 80-20 split and with a minimum of one image per class in the validation set. In this run we determined the epoch and scheduler state in which the validation accuracy was maximized. Then we run the whole 5-fold from the beginning using both the training and validation subsets and we accepted the model of the indicated best epoch.

That way we trained for 14 different subsets a 5-fold and for two times, which results to 140 training procedures. Like in the case of the $Cleaned\ ImageNet^{47}_{Edge}$ trainings, here we have also differences on the data volume, so we preferred a fixed iteration number instead of a fixed epoch. We chose to run every training for 25.000 iterations which corresponds to 25.000 epochs for the case of the 2 sketches per class and 1.336 epochs to for the case of 64 sketches per class. The idea was to run the training for as many epochs as possible but use the early stopping rule determined to the first run in order to fight overfitting.

For our hyperparameter tuning we experimented on the 8 sketches per class and then we trained for the same hyperparameters to all other cases. We also followed the same logic for the scheduler, by setting it always to divide the learning rate by 10 every $\frac{MaxEpoch}{3}$ epochs. The starting learning rate was calculated with the same approach as in the $ImageNet^{47}_{Edge}$ training case and we set it to 10^{-5}. In contrast with our method, here we trained with the maximum batch size possible given our resources which is 128. That's because this procedure was too time consuming, taking a magnitude of weeks to finish. The results of this baseline can be seen in the table B.1

6. RESULTS

6.1 Experiments

Our basic experiments are the ones that used $ImageNet^{47}_{Edge}$ as the dataset. We tried 8 combinations in terms for their augmentations by experimenting with different edge detection method, thresholding method and perturbation. We trained every model five times in order to report the expected performance of each approach and also to have an illustration about their variation. We did that to decrease the stochastic factor of the augmentations and non-convex optimization up to a certain degree. We started by trying for a fixed edge detector, a fixed random threshold and without perturbation. When we chose a fixed threshold that was derived from observing our data we had a minor improvement (less than 1%). Also by adding the Otsu thresholder for each individual image we had another minor improvement. By adding a Gaussian perturbation we had a third minor improvement. The first major improvement was by either adding all the thresholders ($+2.3$%) or all the edge detectors ($+3.5$%). By adding them both we achieved our top performance.

Table 6.1: The results for our methods trained on $ImageNet^{47}_{Edge}$. The threshold 127 was chosen at random and the threshold 97 is the average Otsu threshold for the whole BDCN channel of $ImageNet^{47}_{Edge}$. All the Gaussian perturbations are $\sim N(0, 100)$.

Test Accuracies for Epoch 300		
Rand - Edge - Thresh	Mean	SD
None - BDCN - 127	**0.539**	0.007
None - BDCN - 97[1]	**0.547**	0.012
None - BDCN - Otsu	**0.554**	0.007
Gaussian - BDCN - Otsu	**0.561**	0.007
Gaussian - BDCN - All	**0.584**	0.008
Gaussian - All - Otsu	**0.596**	0.006
None - All - All	**0.605**	0.015
Gaussian - All - All	**0.604**	0.006

As we can see from the Table 6.1 the model with the Gaussian perturbation with all edge detectors and all thresholders has the most consistent results, being only 0.1% below the equivalent model without the perturbation. We prefer the former though because the latter has more than double standard deviation and therefore is a more unstable approach. An hypothesis for this behavior is that by adding the perturbation as augmentation we added bias that equally increased the bias error and decreased the variance error. The most interesting part is that just by adding the same perturbation increased the performance of the fixed edge map - fixed thresholder approach. The detailed results for all 5 models of each experiment can be seen in the Table C.1

[1]97 is the average Otsu threshold for the whole BDCN channel of $ImageNet^{47}_{Edge}$.

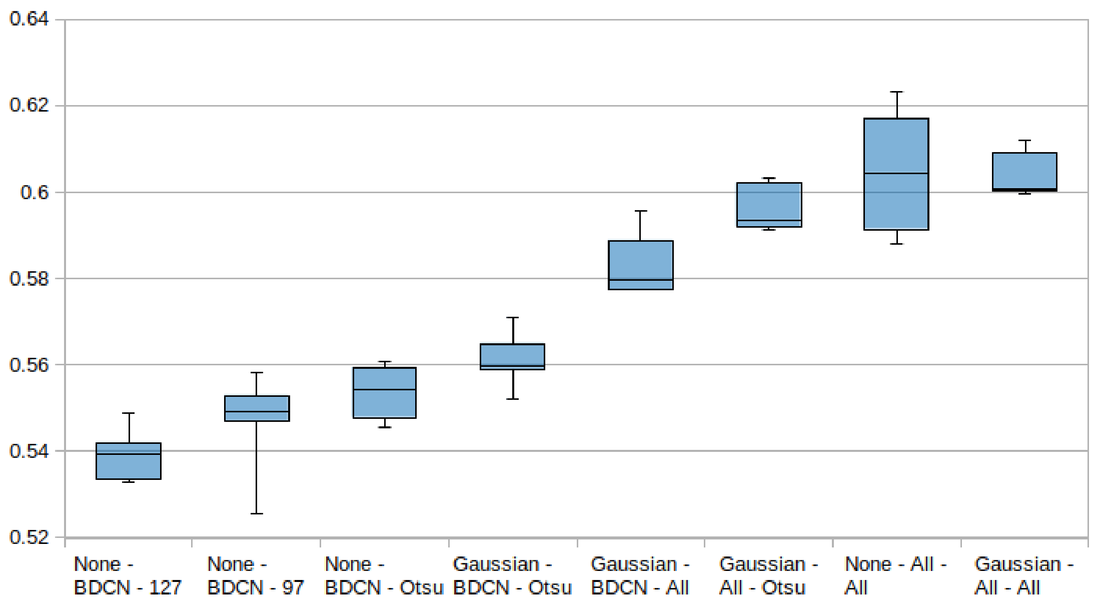


Figure 6.1: Here we can see the model performance of our basic experiments with $ImageNet^{47}_{Edge}$ in terms of accuracy. Because we happened to run exactly 5 models for each experiment, the minimum, 1^{st} quantile, median, 3^{rd} quantile and maximum values of the box-plots are the actual 5 performances of our models in ascending order.

As an additional set of experiments we trained using our chosen "Gaussian-All-All" approach in the $Cleaned\ ImageNet^{47}_{Edge}$ datasets in order to see the potential of domain adaptation in a more ideal setting. We hypothesized that if we applied some human effort to the problem that would translate to better results.

Table 6.2: The results for our methods trained on the $Cleaned\ ImageNet^{47}_{Edge}$ datasets.

Cleaned Dataset 5400 Iterations		
Images per class	Mean	SD
4	**0.489**	0.011
8	**0.592**	0.008
12	**0.630**	0.013
16	**0.663**	0.007
20	**0.676**	0.006
24	**0.682**	0.003

The results are collected in 6.2 and their detailed version with tha whole 5 trained models for each experiment can be found in C.2. As we can see, the performance is ascending with the number of images and the cleaned version models are surpassing our standard method after the 8 images per class. We know that our standard dataset is itself trivially the $Cleaned^{1300}\ ImageNet^{47}_{Edge}$ and therefore its performance would be the last element of this sequence. That means that the sequence cannot be infinitely increasing and it must have a maximum point.

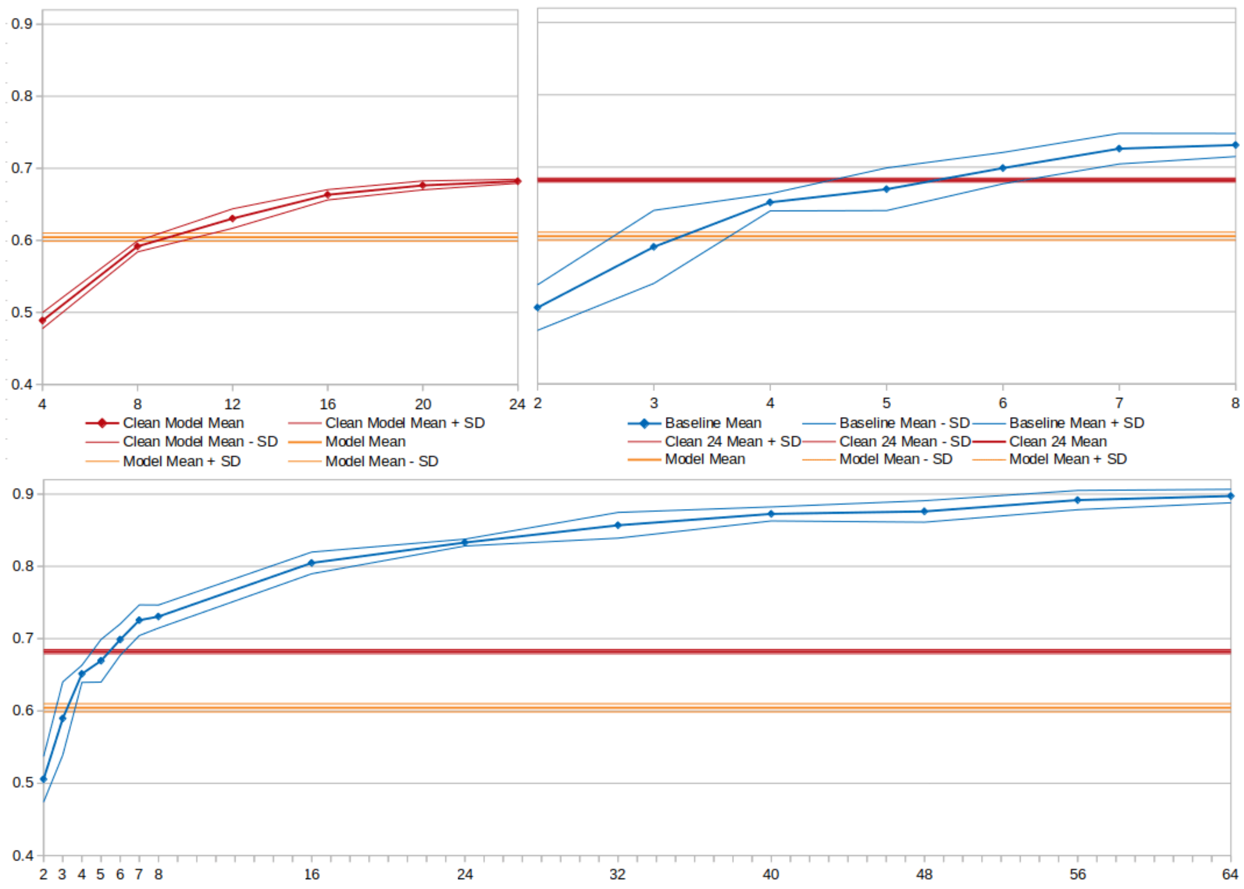


Figure 6.2: In the upper left figure we can see the accuracy of each model trained on $ImageNet^{47}_{Edge}$ compared with the accuracy of each model trained on different sizes of the cleaned version. On the upper right we can see the same model (yellow) and the $Cleaned^{24}ImageNet^{47}_{Edge}$ which is the best cleaned model (red), compared with the baseline (blue). As we can see, our basic model surpasses the baseline for 3 sketches per class while the cleaned model achieves an accuracy within one standard deviation from the baseline of 6 sketches per class. The figure at the bottom is the complete illustration of the baseline situation, showing the baseline trained up for the maximum 64 sketches per class.

In the Figure 6.2 we can see out results collectively. Our model of choice of the standard procedure achieved an accuracy of 60.4% on $TU-Berlin^{47}_{Edge}$, which -as we can see from the baseline- surpasses the performance of the same architecture trained on 3 sketches per class. The equivalent sketch domain data volume of our model is more than 141 sketches. Also The cleaned version of our method achieved an accuracy of 68.2% on $TU-Berlin^{47}_{Edge}$ which is within one standard deviation from the performance of the same model trained on 6 sketches per class. The corresponding sketch data volume for this performance is 282.

Finally, for supervisory reasons we are calculating in every model evaluation the representations of each image of $ImageNet^{47}_{Edge}$ as well as for the $TU-Berlin^{47}_{Edge}$ by removing the classifier of the evaluating model and saving the output. After that we can see the nearest neighbors of each sketch in the image domain according to the model. We show some of the best classified sketches in the Figure D.1, some of the most critical decisions in the Figure E.1 and some of the most confidently wrong classified sketches in Figure F.1 as well as their nearest neighbors of the image domain according to our models.

6.2 Conclusion

In this thesis we showed that it is possible to learn from the image domain and use that information to predict -at some degree- the sketch domain. Also we presented a way to quantify the degree of the extracted information in the form of sketch volume equivalence by creating a baseline from the TU-Berlin subset.

Even in the case of a dataset that is hard and noisy in terms of edge maps like ImageNet, we showed that we can learn information worth as much as three actual sketches per class. We showed this in a 47 class setting but we can assume a level of generalization in an abstract number of classes. Of course we expect that, as the number of classes increase, the volume of sketch-equivalence will increase too so the 141 substituted sketches could increase just by adding more classes to the problem.

We also showed that by manually cleaning the dataset we can at least double our performance in relation to the volume of substituted sketches. This shows that even untrained -in terms of sketch related skills- extra human work can improve our approach in an crucial level. Also, this opens a new expectation for a future approach in terms of an automated cleaning procedure of the images.

We achieved this performance by extracting information only from the one-pixel thick binarized edges of the images, bringing information from the image domain nearer to the sketch domain. This is also making us expect that sketch domain specific augmentations like the ones in [76] could help our procedure in the future.

Finally, like most zero-shot approaches, our results can be useful for the corresponding few-shot setting.

ABBREVIATIONS - ACRONYMS

CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
GRU	Gated Recurrent Unit
Sketch-R2CNN	Sketch RNN Rasterization CNN
LSTM	Long Short-Term Memory
NLR	Neural Line Rasterization
BPD	Bezer Pivot-based Deformation
MSR	Mean Stroke Reconstruction
SBIR	Sketch-Based Image Retrieval
FG-SBIR	Fine-Grained SBIR
MLE	Maximum Likelihood Estimators
KL-Divergence	Kullback–Leibler Divergence
MSE	Mean Square Error
MLP	Multilayer Perceptron
ReLU	Rectified Linear Unit
LASSO	Least Absolute Shrinkage and Selection Operator
MAP	Maximum a Posteriori
RMSprop	Root Mean Square Propagation
GAP	Global Average Pooling
HED	Holistically-nested Edge Detection
BDCN	Bi-Directional Cascade Network
BSDS500	Berkeley Segmentation Data Set and benchmarks 500
VGG	Visual Geometry Group
SEM	Scale Enchancement Module
ODS	Optimal Dataset Score
LDA	Linear Discriminant Analysis

APPENDIX A. THE $TU - Berlin^{47}$ AND $ImageNet^{47}$ CLASSES

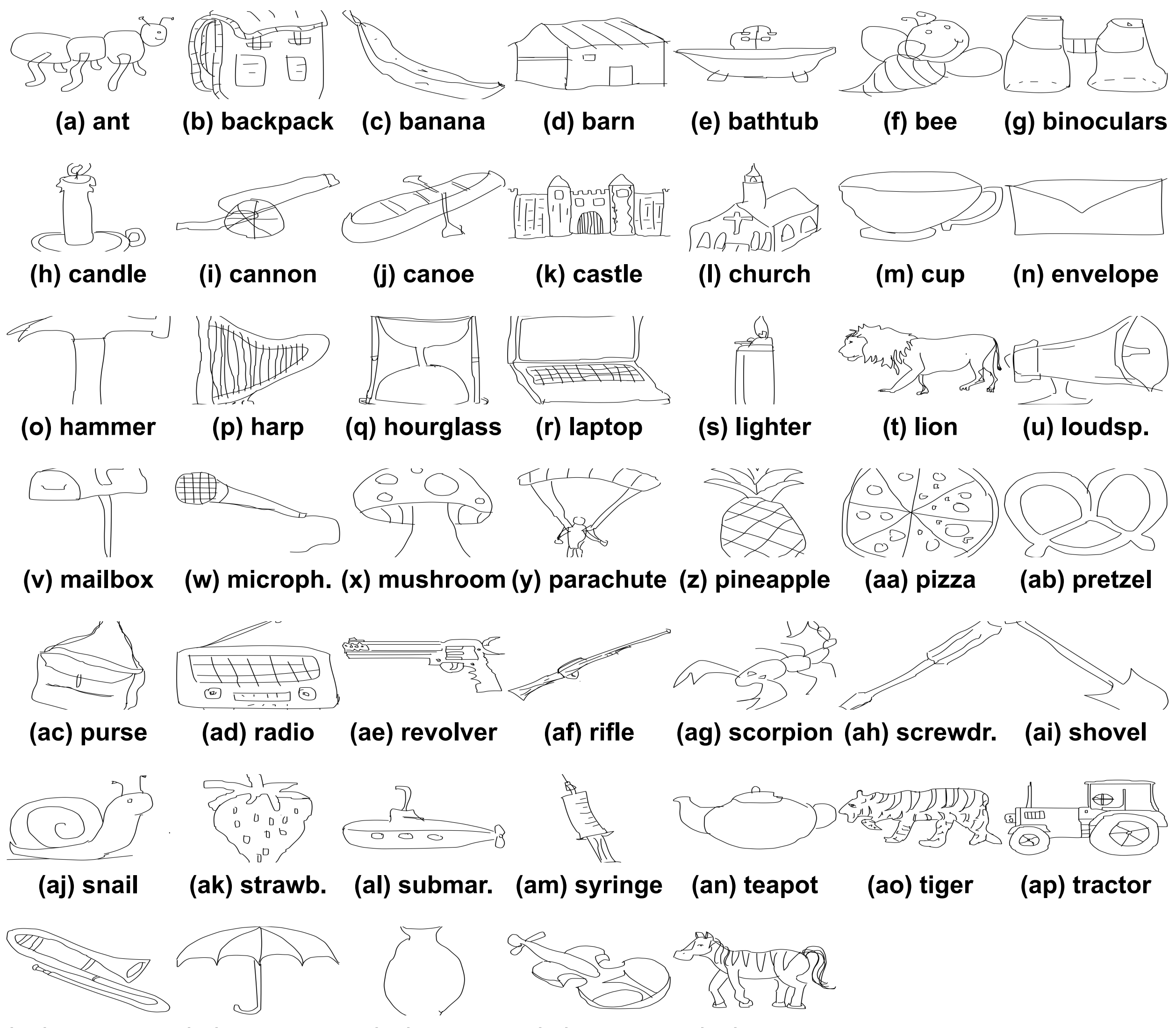


Figure A.1: The classes of $TU - Berlin^{47}$ and $ImageNet^{47}$, with an indicative sketch from $TU - Berlin^{47}$.

APPENDIX B. BASELINE TABLES

Table B.1: The 5-fold cross-validation of the $TU - Berlin^{47}_{Edge}$ dataset baseline.

Test accuracy TU-Berlin Baseline 5-fold ResNet-50 k=2-8							
k	2	3	4	5	6	7	8
Fold 1	0.501	0.584	0.657	0.703	0.719	0.713	0.745
Fold 2	0.493	0.582	0.669	0.672	0.693	0.706	0.707
Fold 3	0.535	0.633	0.649	0.641	0.719	0.738	0.745
Fold 4	0.537	0.637	0.642	0.638	0.668	0.714	0.722
Fold 5	0.461	0.512	0.640	0.693	0.694	0.757	0.734
SD	0.031	0.051	0.012	0.030	0.022	0.021	0.016
Mean	**0.506**	**0.590**	**0.651**	**0.669**	**0.699**	**0.726**	**0.731**
Mean - SD	0.474	0.539	0.639	0.640	0.677	0.704	0.715
Mean + SD	0.537	0.640	0.663	0.699	0.720	0.747	0.747

Test accuracy TU-Berlin Baseline 5-fold ResNet-50 k=16-64							
k	16	24	32	40	48	56	64
Fold 1	0.798	0.832	0.872	0.886	0.880	0.904	0.906
Fold 2	0.782	0.828	0.828	0.862	0.867	0.879	0.883
Fold 3	0.814	0.828	0.851	0.866	0.855	0.878	0.895
Fold 4	0.811	0.836	0.868	0.879	0.891	0.892	0.906
Fold 5	0.819	0.839	0.864	0.871	0.887	0.906	0.898
SD	0.015	0.005	0.018	0.010	0.015	0.013	0.009
Mean	**0.805**	**0.833**	**0.857**	**0.873**	**0.876**	**0.892**	**0.897**
Mean - SD	0.790	0.828	0.839	0.863	0.861	0.878	0.888
Mean + SD	0.820	0.838	0.875	0.882	0.891	0.905	0.907

APPENDIX C. RESULTS

Table C.1: The detailed results for our methods trained on $ImageNet^{47}_{Edge}$.

Test Accuracies for Epoch 300							
Rand-Edge-Thresh	Run 1	Run 2	Run 3	Run 4	Run 5	Mean	SD
None-BDCN-127	0.534	0.533	0.542	0.539	0.549	**0.539**	0.007
None-BDCN-97	0.547	0.549	0.558	0.526	0.553	**0.547**	0.012
None-BDCN-Otsu	0.546	0.561	0.559	0.554	0.548	**0.554**	0.007
Gauss-BDCN-Otsu	0.560	0.565	0.571	0.552	0.559	**0.561**	0.007
Gauss-BDCN-All	0.577	0.596	0.589	0.580	0.577	**0.584**	0.008
Gauss-All-Otsu	0.603	0.602	0.591	0.594	0.592	**0.596**	0.006
None-All-All	0.617	0.591	0.605	0.623	0.588	**0.605**	0.015
Gauss-All-All	0.600	0.601	0.612	0.601	0.609	**0.604**	0.006

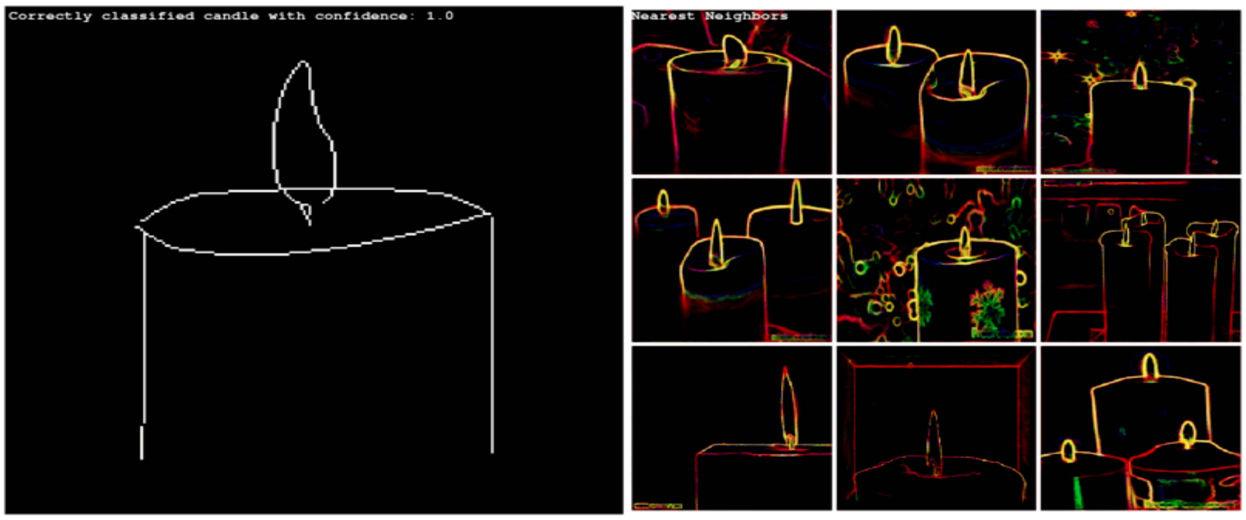
Test Accuracies for Epoch 300					
Rand-Edge-Thresh	Min	LQ	Median	UQ	Max
None-BDCN-127	0.533	0.534	0.539	0.542	0.549
None-BDCN-97	0.526	0.547	0.549	0.553	0.558
None-BDCN-Otsu	0.546	0.548	0.554	0.559	0.561
Gauss-BDCN-Otsu	0.552	0.559	0.560	0.565	0.571
Gauss-BDCN-All	0.577	0.577	0.580	0.589	0.596
Gauss-All-Otsu	0.591	0.592	0.594	0.602	0.603
None-All-All	0.588	0.591	0.605	0.617	0.623
Gauss-All-All	0.600	0.601	0.601	0.609	0.612

Table C.2: The detailed results for our methods trained on the $Cleaned\ ImageNet^{47}_{Edge}$ datasets.

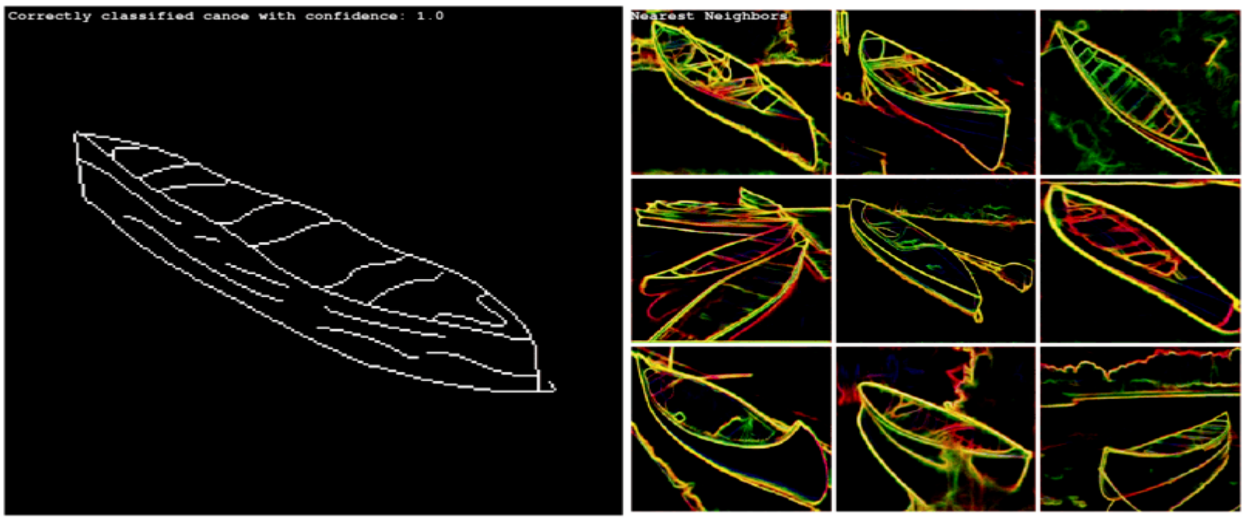
Cleaned Dataset 5400 Iterations							
Images per class	Run 1	Run 2	Run 3	Run 4	Run 5	Mean	SD
4	0.483	0.486	0.479	0.508	0.488	**0.489**	0.011
8	0.599	0.595	0.580	0.589	0.595	**0.592**	0.008
12	0.626	0.633	0.612	0.649	0.631	**0.630**	0.013
16	0.660	0.672	0.667	0.653	0.664	**0.663**	0.007
20	0.677	0.670	0.676	0.673	0.686	**0.676**	0.006
24	0.685	0.683	0.683	0.677	0.682	**0.682**	0.003

Cleaned Dataset 5400 Iterations					
Images per class	Min	LQ	Median	UQ	Max
4	0.479	0.483	0.486	0.488	0.508
8	0.580	0.589	0.595	0.595	0.599
12	0.612	0.626	0.631	0.633	0.649
16	0.653	0.660	0.664	0.667	0.672
20	0.670	0.673	0.676	0.677	0.686
24	0.677	0.682	0.683	0.683	0.685

APPENDIX D. GOOD EXAMPLES



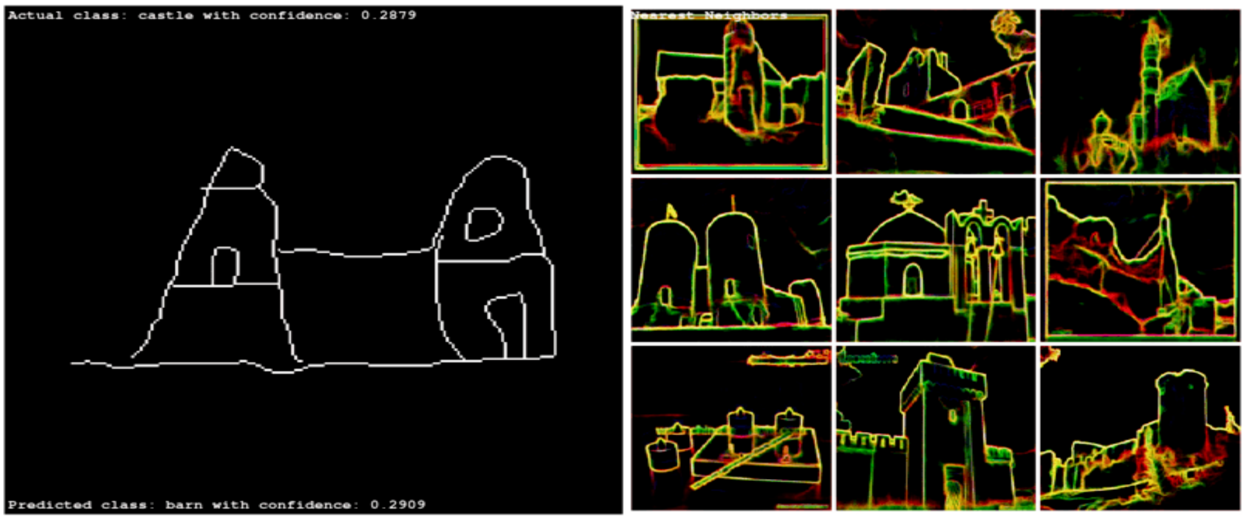
(a) A correctly classified candle



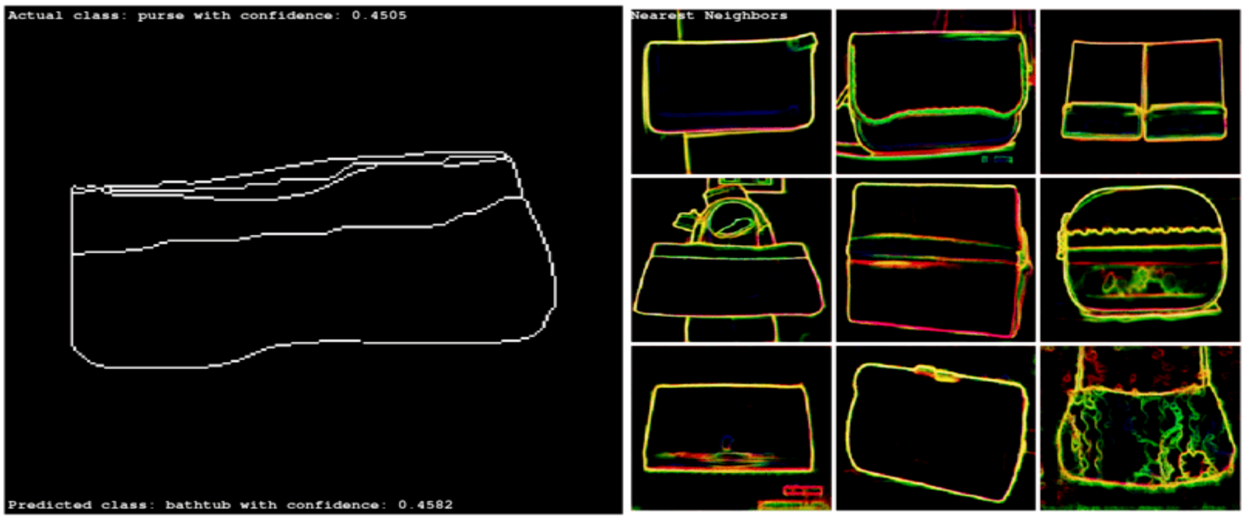
(b) A correctly classified canoe

Figure D.1: Correctly classified sketches from our model with the nearest neighbors of the photograph domain according to our model.

APPENDIX E. CRITICAL EXAMPLES


Actual class: castle with confidence: 0.2879
Predicted class: barn with confidence: 0.2909
Nearest Neighbors

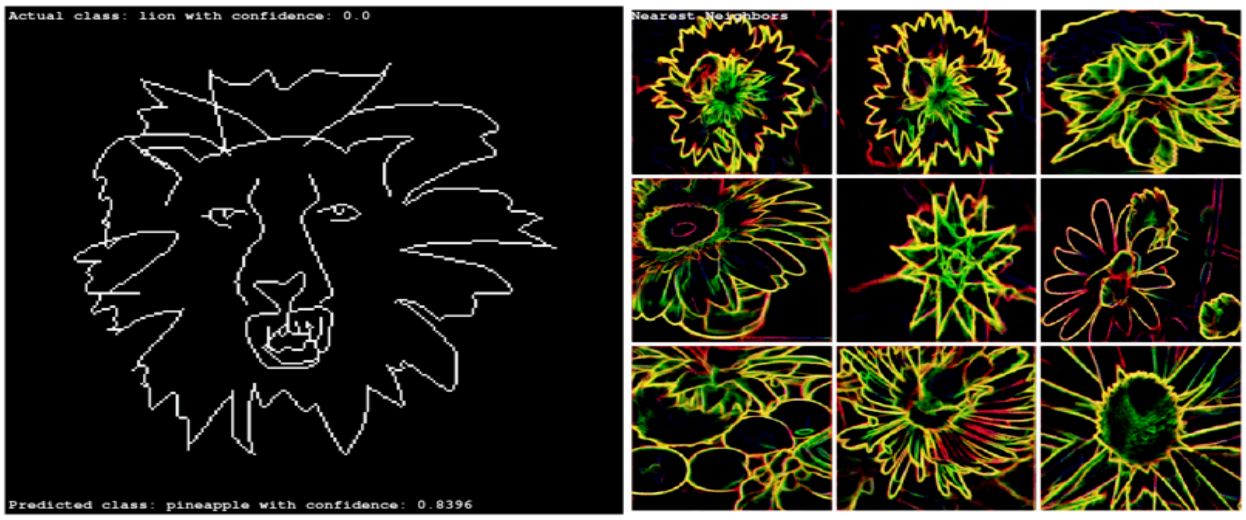
(a) A castle classified as a barn


Actual class: purse with confidence: 0.4505
Predicted class: bathtub with confidence: 0.4582
Nearest Neighbors

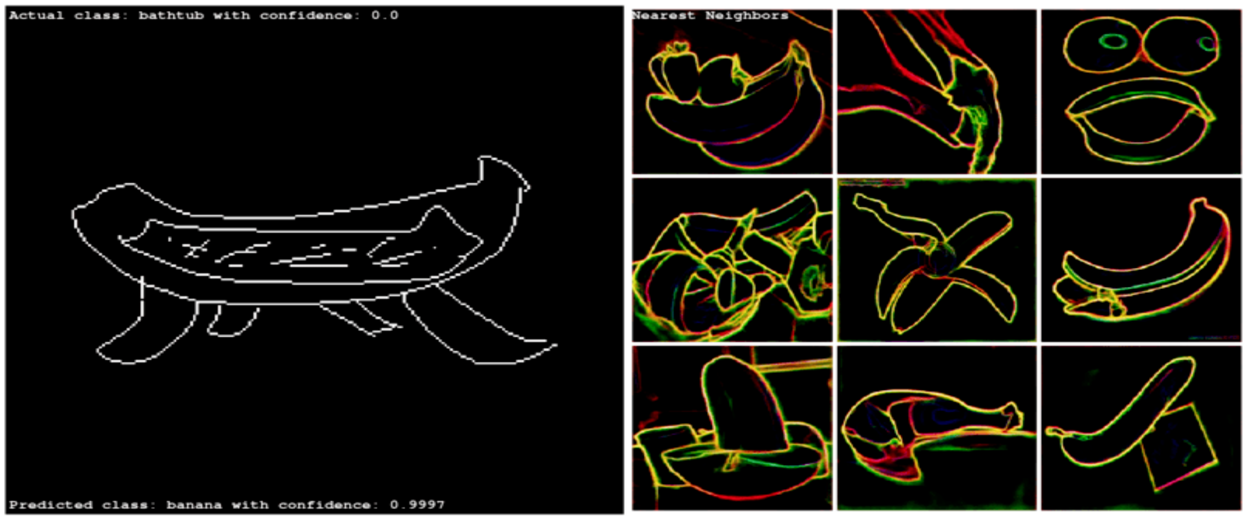
(b) A purse classified as a bathtub

Figure E.1: Wrongly classified sketches from our model by a close call, next to their the nearest neighbors of the photograph domain according to our model.

APPENDIX F. WORST EXAMPLES


Actual class: lion with confidence: 0.0
Nearest Neighbors
Predicted class: pineapple with confidence: 0.8396

(a) A lion classified as a pineapple


Actual class: bathtub with confidence: 0.0
Nearest Neighbors
Predicted class: banana with confidence: 0.9997

(b) A canoe classified as a banana

Figure F.1: Wrongly classified sketches from our model with very high confidence, next to their the nearest neighbors of the photograph domain according to our model.

REFERENCES

[1] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.

[2] Yannis Avrithis. Lecture notes in deep learning for vision: Convolution and network architectures. `https://sif-dlv.github.io`, December 2019. Notes from the master program research in computer science of the university of Rennes 1. Online; accessed 29 June 2020.

[3] Pedro Ballester and Ricardo Matsumura Araujo. On the performance of googlenet and alexnet applied to sketches. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 1124–1128. AAAI Press, 2016.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[5] Leo Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statist. Sci.*, 16(3):199–231, 08 2001.

[6] A. E. Bryson. A gradient method for optimizing multi-stage allocation processes. In *Proc. Harvard Univ. Symposium on digital computers and their applications*, 1961.

[7] Held M Buhmann J.M. *Classification, Automation, and New Media: Proceedings of the 24th Annual Conference of the Gesellschaft für Klassifikation e.V., University of Passau, March 15—17, 2000*. Studies in Classification, Data Analysis, and Knowledge Organization. Springer-Verlag Berlin Heidelberg, 1 edition, 2002.

[8] Dong Chen, Xudong Cao, Liwei Wang, Fang Wen, and Jian Sun. Bayesian face revisited: A joint formulation. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, pages 566–579, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[9] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[10] Sumit Chopra, Raia Hadsell, and Yann Lecun. Learning a similarity metric discriminatively, with application to face verification. volume 1, pages 539– 546 vol. 1, 07 2005.

[11] C.K. Chow and T. Kaneko. Automatic boundary detection of the left ventricle from cineangiograms. *Computers and Biomedical Research*, 5(4):388 – 410, 1972.

[12] Gabriela Csurka. *Domain Adaptation in Computer Vision Applications*. Springer Publishing Company, Incorporated, 1st edition, 2017.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[14] Piotr Dollár and C. Lawrence Zitnick. Structured forests for fast edge detection. In *Proceedings of the 2013 IEEE International Conference on Computer Vision*, ICCV '13, page 1841–1848, USA, 2013. IEEE Computer Society.

[15] Pedro M. Domingos. A unified bias-variance decomposition for zero-one and squared loss. In Henry A. Kautz and Bruce W. Porter, editors, *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA*, pages 564–569. AAAI Press / The MIT Press, 2000.

[16] S. E. Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45, 1962.

[17] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Trans. Graph. (Proc. SIGGRAPH)*, 31(4):44:1–44:10, 2012.

[18] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(7):179–188, 1936.

[19] Fu-Juay Chang, Jui-Cheng Yen, and Shyang Chang. Gray-level thresholding via maximum correlation criterion. In *Proceedings of the 3rd International Conference on Advances in Communication and Control Systems*, Victoria, Canada, October 1991.

[20] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Comput.*, 4(1):1–58, January 1992.

[21] C.A. Glasbey. An analysis of histogram-based thresholding algorithms. *CVGIP: Graphical Models and Image Processing*, 55(6):532 – 537, 1993.

[22] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[23] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[24] Jianzhong He, Shiliang Zhang, Ming Yang, Yanhu Shan, and Tiejun Huang. Bi-directional cascade network for perceptual edge detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[25] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *IEEE International Conference on Computer Vision (ICCV 2015)*, 1502, 02 2015.

[27] Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint*, arXiv, 07 2012.

[28] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[29] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[30] Jui-Cheng Yen, Fu-Juay Chang, and Shyang Chang. A new criterion for automatic multilevel thresholding. *IEEE Transactions on Image Processing*, 4(3):370–378, 1995.

[31] Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[33] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[34] Christoph Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. 06 2009.

[35] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[36] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[37] Alexander LeNail. Nn-svg: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, 4(33):747, 2019.

[38] C.H. Li and C.K. Lee. Minimum cross entropy thresholding. *Pattern Recognition*, 26(4):617 – 625, 1993.

[39] Lei Li, Changqing Zou, Youyi Zheng, Su Qingkun, Hongbo Fu, and Chiew-Lan Tai. Sketch-r2cnn: An attentive network for vector sketch recognition, 11 2018.

[40] Yi Li, Timothy Hospedales, Yi-Zhe Song, and Shaogang Gong. Fine-grained sketch-based image retrieval by matching deformable part models. *BMVC 2014 - Proceedings of the British Machine Vision Conference 2014*, 01 2014.

[41] Yi Li and Wenzhao Li. A survey of sketch-based image retrieval. *Machine Vision and Applications*, 2018.

[42] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160, 1976.

[43] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM ’10, page 1485–1488, New York, NY, USA, 2010. Association for Computing Machinery.

[44] Brady Neal, Sarthak Mittal, Aristide Baratin, Vinayak Tantia, Matthew Scicluna, Simon Lacoste-Julien, and Ioannis Mitliagkas. A modern take on the bias-variance tradeoff in neural networks. *ArXiv*, abs/1810.08591, 2018.

[45] Marc Peter Deisenroth; A Aldo Faisal; Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[46] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[47] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[48] Trevor Park and George Casella. The bayesian lasso. *Journal of the American Statistical Association*, 103(482):681–686, 2008.

[49] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[50] Yu Qian, Yang Yongxin, Liu Feng, Song Yi-Zhe, Xiang Tao, and Hospedales Timothy M. Sketch-a-net: A deep neural network that beats humans. *International Journal of Computer Vision*, 2017.

[51] Yu Qian, Yang Yongxin, Song Yi-Zhe, Xiang Tao, and Hospedales Timothy M. Sketch-a-net that beats humans. *British Machine Vision Conference*, pages 7.1–7.2, 2015.

[52] Filip Radenovic, Giorgos Tolias, and Ondrej Chum. Deep shape matching. *ECCV*, pages 774–791, 2018.

[53] Feris Rogerio Schmidt, Lampert Christoph, and Parikh Devi. *Visual Attributes*. Springer, 1st edition, 2017.

[54] Bernardino Romera-Paredes and Philip H. S. Torr. An embarrassingly simple approach to zero-shot learning. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML’15, page 2152–2161. JMLR.org, 2015.

[55] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[56] David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[57] Patsorn Sangkloy, Nathan Burnell, Cusuh Ham, and James Hays. The sketchy database: Learning to retrieve badly drawn bunnies. *ACM Transactions on Graphics (proceedings of SIGGRAPH)*, 2016.

[58] Santosh Ravi Kiran Sarvadevabhatla, Jogendra Kundu, and R. Babu. Enabling my robot to play pictionary : Recurrent neural networks for sketch recognition. 10 2016.

[59] J. Sauvola and M. Pietikäinen. Adaptive document image binarization. *Pattern Recognition*, 33(2):225 – 236, 2000.

[60] Mike Schuster and Kuldip Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45:2673 – 2681, 12 1997.

[61] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[62] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[63] S. Theodoridis. *Machine Learning: A Bayesian and Optimization Perspective*. 05 2015.

[64] Joy A. Thomas Thomas M. Cover. *Elements of Information Theory*. Wiley Series in Telecommunications and Signal Processing. Wiley-Interscience, 2nd ed edition, 2006.

[65] Robert Tibshirani. Regression shrinkage and selection via the lasso. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 58:267–288, 1994.

[66] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[67] Ridler TW and Calvard S. Picture thresholding using an iterative selection method. *IEEE Transactions on Systems, Man, and Cybernetics*, 8(8):630–632, 1978.

[68] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014.

[69] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu. Learning fine-grained image similarity with deep ranking. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1386–1393, 2014.

[70] Karl R. Weiss, Taghi M. Khoshgoftaar, and Dingding Wang. A survey of transfer learning. *Journal of Big Data*, 3:1–40, 2016.

[71] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. *Int. J. Comput. Vision*, 125(1–3):3–18, December 2017.

[72] Zitong Yang, Yaodong Yu, Chong You, Jacob Steinhardt, and Yuliang Ma. Rethinking bias-variance trade-off for generalization of neural networks. *ArXiv*, abs/2002.11328, 2020.

[73] Qian Yu, Feng Liu, Yi-Zhe SonG, Tao Xiang, Timothy Hospedales, and Chen Change Loy. Sketch me that shoe. In *Computer Vision and Pattern Recognition*, 2016.

[74] H. Zhang, S. Liu, C. Zhang, W. Ren, R. Wang, and X. Cao. Sketchnet: Sketch classification with web images. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1105–1113, 2016.

[75] T. Y. Zhang and C. Y. Suen. A fast parallel algorithm for thinning digital patterns. *Commun. ACM*, 27(3):236–239, March 1984.

[76] Ying Zheng, Hongxun Yao, Xiaoshuai Sun, Shengping Zhang, Sicheng Zhao, and Fatih Porikli. Sketch-specific data augmentation for freehand sketch recognition, 10 2019.

[77] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67:301–320, 2005.