Deep learning methods for style extraction and transfer

Omar Mohammed
Thesis director: Gérard Bailly
Thesis co-director: Damien Pellier

Contents

List of Figures

List of Tables

Part I

Introduction and Problem Description

Chapter 1

Introduction

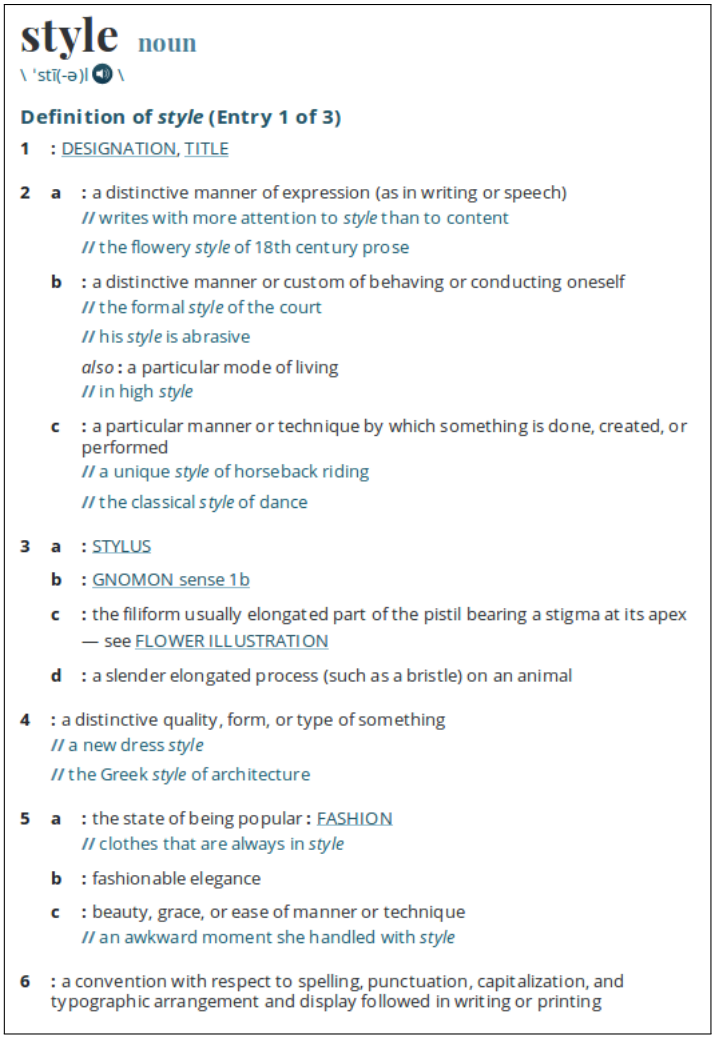
Contents

In one sentence, my PhD focus on the extraction, characterization and *transfer* of *styles* or *personas*, isolated from a task, using deep neural networks.

1.1 What is a style?

Style is generically defined in Merriam-Webster dictionary (figure 1.1). In general, it is defined as *the manner of doing things* (Gallaher, 1992). To get more sense of what style is, it is better to give some examples first, to get an idea about what we are dealing with.

- When we say the word "seriously?". Depending on the manner we say it, it will carry different meaning (sarcastic or surprise for example). One word, two different manners to say it.
- Handwriting: You can write down the same the letter (the task), but with multiple typefaces (the style) (figure 1.2).



style noun

\ ˈstī(-ə)l \

Definition of *style* (Entry 1 of 3)

1 : DESIGNATION, TITLE

2 **a** : a distinctive manner of expression (as in writing or speech)
// writes with more attention to *style* than to content
// the flowery *style* of 18th century prose

b : a distinctive manner or custom of behaving or conducting oneself
// the formal *style* of the court
// his *style* is abrasive
also : a particular mode of living
// in high *style*

c : a particular manner or technique by which something is done, created, or performed
// a unique *style* of horseback riding
// the classical *style* of dance

3 **a** : STYLUS

b : GNOMON sense 1b

c : the filiform usually elongated part of the pistil bearing a stigma at its apex
— see FLOWER ILLUSTRATION

d : a slender elongated process (such as a bristle) on an animal

4 : a distinctive quality, form, or type of something
// a new dress *style*
// the Greek *style* of architecture

5 **a** : the state of being popular : FASHION
// clothes that are always in *style*

b : fashionable elegance

c : beauty, grace, or ease of manner or technique
// an awkward moment she handled with *style*

6 : a convention with respect to spelling, punctuation, capitalization, and typographic arrangement and display followed in writing or printing

Figure 1.1: Definition of style in Merriam-Webster dictionary

- A movie setting: the script is provided to the actor. There are, however, many ways for the actor to perform what is written in the script, in order to convey different messages/experiences to the audience.
- Clothing and fashion: different groups of people have different general style lines – depending on the region, ethnicity ... etc –. Within each group, we can see people having diverse styles within this general defining style.

Figure 1.2: Multiple styles (different typefaces) for the same letters.

In these two examples, we see a basic structure:

- A fixed part: the word to be said, or the letter identity. We will call this the *content*.
- A variable part: the manner we say the word, or the manner we write the letter. We will call this the *style*.
- Together, a style and a content forms a *task*.

The mention of styles is quite a lot in the literature in multiple domains, for example:

- **Speech synthesis**: (Tachibana et al., 2004) defines speaking style in a high-level manner, in terms of emotions expressed by the speaker, like 'joy', 'sad', or the interpolation between them, when reading a text. (Wang et al., 2018) looks at the speech style in a more detailed manner, considering different aspects of speech prosody, like the paralinguistic information, intonation and stress.
- **Car driving**: there are multiple ways to categorize the different driving styles. It can be based on the safety aspect (Johnson and Trivedi, 2011), the aggressiveness

of the maneuver (Dörr et al., 2014; Xu et al., 2015b), the impact on fuel consumption (Manzoni et al., 2010; Neubauer and Wood, 2013). Many other identification basis for driving styles are summarized nicely in (Martinez et al., 2017).

- **Handwriting**: handwriting can be offline (the final image of the letter) or online (recording the movement of the pen/drawing tool). Depending on which one considered, the style profile can change. Figure 1.2 is an example of different offline styles (the typefaces). But when we consider the online aspect of the drawing, we can see different aspects, like in figure 1.3, where we see that the same drawing can be in clockwise or counterclockwise direction (we will expand more on this point in chapter 4.2.4).

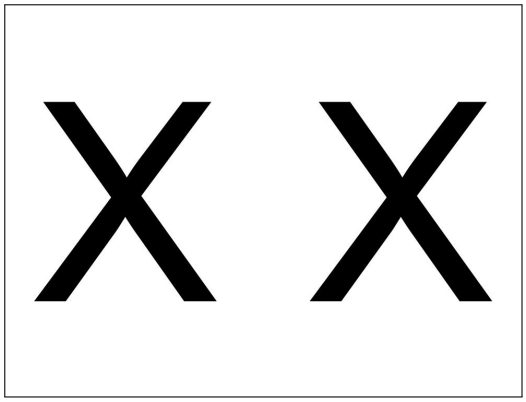
One thing we would like to highlight here: that styles are not thing we all agree on. It can be hierarchical as well. In the clothing example earlier, people are affected by the style group they belong to, but within this cluster, people have diverse individual styles. The same happens in handwriting: education and culture affects the style cluster people belong to, but we can observe a wide range of individual styles in each cluster.

Another thing to highlight is that there is no one definition for styles. It depends on the aspect of interest that we want to observe and study – in car driving, safety and fuel consumption are two areas of interest, leading to different type of styles –. This leads to an important characteristic of *styles*, that it is an ill-defined concept. We know that *styles* are rich in information and important in communication between humans. They are needed in order to convey meaning. As noted in (Taylor, 2009) – in the context of speech synthesis –, a proper rendering of styles affects the overall perception. However, we can not completely remove the ambiguity in this definition.

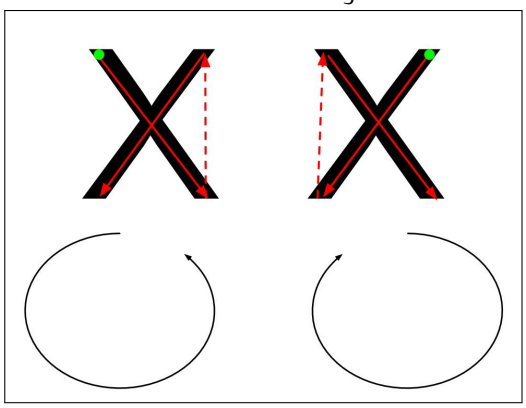
1.2 What is the objective of this project?

Our long-term objective is to enable our humanoid iCub robot Nina 1.4, to have a exhibit personalized behavior suitable for the person interacting with it. This will enhance the user experience, and will allow for a more natural interaction with the robot. It is shown that a robot which exhibit a personalized behavior is more likable acceptable by humans, and perceived as an intelligent entity (Churamani et al., 2017). Humans have different preferences when interacting with each other, or interacting with the robot, and taking them into account does improve the quality of interaction, and the potential of success for the task (Kashi and Levy-Tzedek, 2018).

At the moment, we successfully used machine learning approaches in order to build models of human-robot interaction (Bailly et al., 2018; Mihoub et al., 2016; Nguyen et al., 2017). However, when using these models to generate behaviors, this



1 Offline drawing



2 Online drawing

Figure 1.3: Example for style in case of online handwriting. Although both examples looks the same when we look at it in the offline mode (the final drawing), they are quite different when we consider the online aspect (the dynamics of the pen when drawing them). Although not illustrated here, it is important to note that some aspects of the online drawing dynamics can be deduced from the offline drawing – in other words, the dynamics can affect the end result – (Diaz et al., 2017). In this case, the starting point and the direction of drawing (clockwise or counterclockwise) are different. The solid line indicate a stroke, and the dotted line indicate an air stroke (the transition of the pen in the air between two strokes). The green dot is the starting point.

behavior usually represents an average over the learned behaviors (which is expected). The goal is to learn models of styles, and use it to bias the models of interaction that we have, in order to generate more personalized behaviors. We want the robot to adapt to the human partner on different levels, and not just act in a reactive manner to the human actions. It has been shown that a suitable cognitive model for human-robot interaction takes into account long term styles – for example, the person age, gender and if he/she is shy or not –, and short term decision making – in talking with the human for example – (Bailly et al., 2010; Thórisson, 2002). The ability to identify, extract and use the human style traits will enable biasing the robot model to adapt for the human partner.

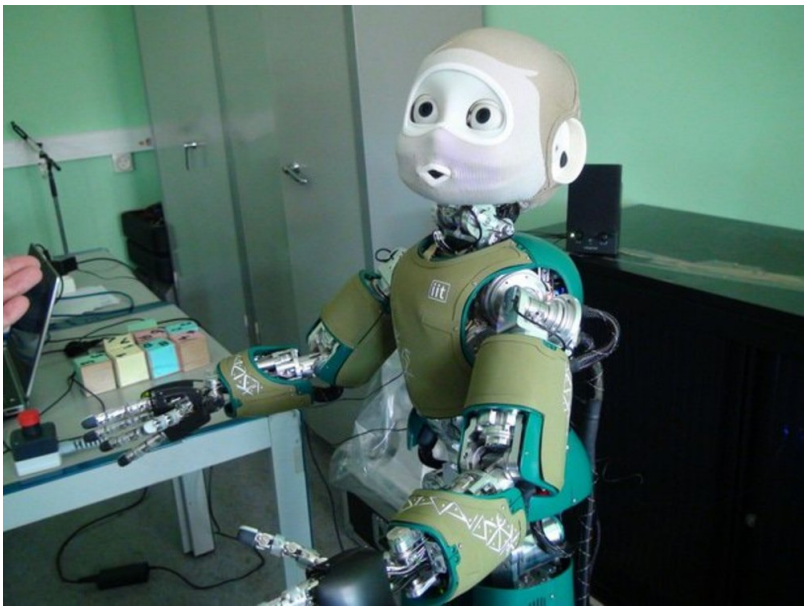


Figure 1.4: The iCub humanoid robot (Nina).

1.3 Why Handwriting?

As mentioned earlier, the final objective of this project is to extract and transfer styles in the context of human-robot interaction. What this has to do with handwriting?

The usage of deep learning in HRI is still in its infancy, mainly because of the lack of datasets. The issue is not collecting the data (there are many small open-source datasets available online), but having a unified set of objective and platforms for HRI, which is not an easy challenge. We envision that this problem will be resolved, as there is a growing interest in the community to address it.

Thus come handwriting. We use it as a proxy platform to understand, build and test different approaches to use deep learning. It has many advantages to make it a good proxy, including:

- Availability of datasets: several datasets already exist, with large quantities of data.
- Diverse of tasks and styles: there are many tasks (letters) in handwriting, ranging from simple (like letter 'C') to complex (like letter 'E'), and the writers exhibit a diverse set of styles on the different tasks, making handwriting a good candidate to explore the problem of styles.
- Several style aspects are accessible to investigate visually, making it more accessible for in-depth analysis, and getting insight on how the model behaves.
- Several datasets provide information about the writers, like the age, handiness, gender and the origin. This data is interesting in some aspects of the style problem.
- We have a clear idea about the content of each task (the identify of the letter or the shape). Usually, the task is presented to us with the content and the style mixed together. How to disentangle the content from the style is an open question, and the fact the styles is an ill-defined problem makes it more ambiguous. With the assumption that we know the content, we can focus our effort on the styles problem[1].

While the subjects are interacting with the environment (the pen, tablet ... etc) – which is observed and recorded –, there are no human interaction aspects in this domain of data, which is a disadvantage

1.4 What is transfer learning? and why do we need it?

We will discuss transfer learning in more detail in chapter 5, but for now, we want to motivate having this as one of the PhD objectives.

Transfer of knowledge deals with the problem of leveraging the knowledge learned from one task, to accelerate/improve the learning of a new task. This is a skill human do naturally, for example, if you learn Mathematics, and you want to learn physics, you can easily leverage the knowledge of Mathematics to bootstrap your learning of physics. This, intuitive as it seems, is not straightforward for machine learning

[1] We will argue later that the task identity is not necessarily a good representation for the task content.

models. A change in the distribution of the input to the model leads to significant degradation in the performance of the model (Shimodaira, 2000).

Transfer learning is thus a field of machine learning, concerned with developing algorithms and procedures, to enable the transfer of knowledge between different tasks. So many techniques are available for transfer learning, but there is always a common assumption, that the transfer has the potential of success if the tasks are related (i.e., if there is common knowledge between the tasks), otherwise, a transfer learning can at best lead to no improvement, or even reduce the performance of the new model (Weiss et al., 2016).

Why do we need transfer learning? We do not always have the availability of large datasets on the tasks that we want. In many cases, the acquisition and/or the annotation of large dataset can be prohibitively expensive. For example:

- In robotics (Konidaris et al., 2012a,b), collecting data can be quite expensive process, due to hardware limitations from one side, and human limitation as well (in case of human-robot interaction scenarios). In addition, with techniques like reinforcement learning (Sutton and Barto, 2018), where the robot learns by trial and error, the process can be prohibitively slow, with safety concerns sometimes. Also, no data augmentation techniques does exist in the literature for HRI, thus synthesizing extra data is not possible. Thus, we need to be able to transfer the knowledge from the task where we have large amount of data, to a relevant task where we do not have this advantage (figure 1.5).
- In underwater acoustics (Malfante, 2018), an essential task is collecting and cleaning the data about the different fish sounds. This is a tediously manual job, and any change (type of fish, time of the day or place in the ocean) degrades the quality of prediction a lot. Transfer learning can be very useful in this case, to reduce the effort needed to collect, clean and annotate the data.

What we want to achieve in this thesis is to transfer the styles between different tasks. We hypothesize that, when the tasks are relevant (but different tasks, with different contents), that humans share leverage styles between the different tasks. In case of handwriting for example, we can reuse the strokes that we learn in the uppercase letters in order to learn the lowercase letters and digits. We will test this hypothesis in details in chapter 5.

1.5 If we want transfer learning, why extracting styles?

It is important not to forget our mission while being consumed by the shiny light of machine learning. It is easy to fall into the trap of "getting the extra 0.1% accuracy", figure

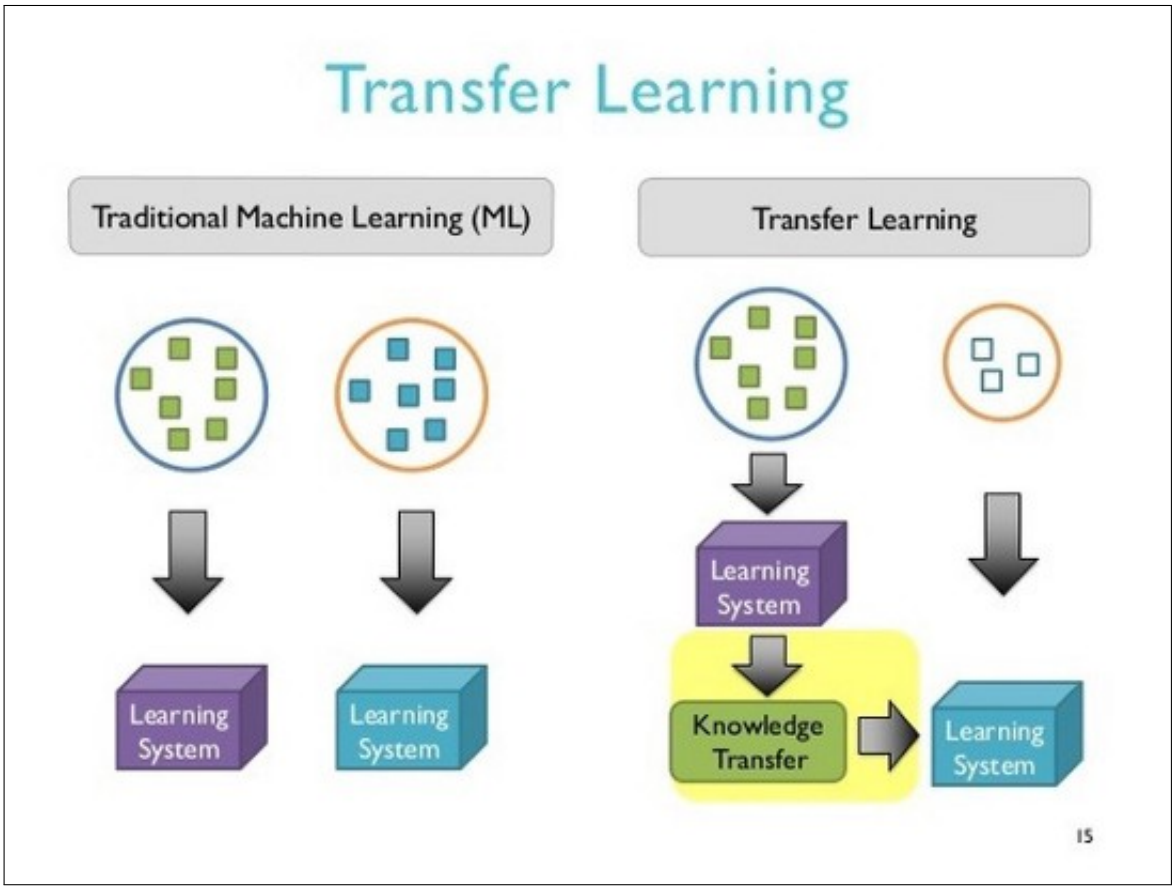


Figure 1.5: Illustration for a practical case of using transfer learning, where we have insufficient data on the new task, and we want to leverage the knowledge learned from another relevant task in order to learn the new task. Source of this image is (Jain).

1.6. Extracting styles enables us to have an idea of what the model actually learned (i.e., it makes the model more interpretable). But why do we need interpretability? Many reasons:

- Debugging: Neural networks are notoriously known for being black box models. Extracting styles gives us some indication on what the system learned, and whether it learned relevant information.
- Discovery of new things: in the era of big data, trying to reason on the data directly is no longer feasible. Instead, a good approach is to reason on a model that fits these data (i.e., the model compresses the data into fewer dimensions). We would like to use this model to reason and discover new things about the data. After all, **the goal of science** is to gain knowledge about the world, and an understanding of how it works. One thing we are interested in is to illuminate the workspace of interaction: what are the limits of possible actions, expected reactions and end results. These kind of questions will benefit a lot understanding what the model actually learned.
- Understanding why: in some cases (e.g., in case of unexpected events), it is human curiosity to understand the reasoning behind the different decisions. We would like to get insight on why the model led to that particular decision.
- Developing safety measures: in many applications (like when dealing with the robots), it is important to be sure that the robot is a 100% safe for the human. Understanding what the model learned can give us insight on the shortcomings of the model, allowing us to fix or improve.
- Social acceptance: humans always tries to attribute beliefs, intentions, personality traits and desires to different objects (Heider and Simmel, 1944). An interpretable machine will reinforce these sensations in humans.
- Improve social interactions: when the robot can explain itself and its perception of the world, it creates a common understanding with the human. This allows the humans to build a mental model for what the robot is actually trying to do, thus, building trust between humans and the robot.

For further details, we strongly recommend the book (Molnar, 2019) on the topic of machine learning interpretability.

1.6 Contributions of this PhD

In this manuscript, we discuss the different contribution of this PhD, addressing different aspects of the styles:

Figure 1.6: Source: `https://xkcd.com/`

1. Propose a manner of thinking about styles: traditionally, a lot of work has been done in order to manually extract and annotate styles, and deal with styles in terms of predictability (using the extracting styles in a regression/classification problem). In this PhD, we propose to implicitly evaluate styles by observing the generative aspects of the model (letting the model generates behaviors, and trying to evaluate the distance between these behaviors and the target/ground truth ones). This is discussed in chapter 3.

2. Propose and build benchmarks and evaluation metrics, and ground those metrics, in order to compare and evaluate future style extraction methods. This is discussed in chapter 3.

3. Propose a generic framework to study styles, using a conditioned-autoencoder. We evaluate this framework in its basic form against the benchmarks. We further validate this framework by extracting verbose styles from it, including ones that are not known from the literature. This is discussed in chapter 4.

4. Last, we address the problem of style transfer. We show how to use our proposed framework in order to transfer styles. We perform extensive experiments in a lot of combinations of tasks, on two different datasets. We also enhance and expand on the evaluation metrics we use in order to quantify the quality of transfer. This is discussed in chapter 5.

1.7 Thesis outlines

We start in chapter 2 by explaining the datasets used in this PhD, and explore their different characteristics, and the preprocessing performed on them. In chapter 3, we discuss first the different aspect of deep learning that we are using in this PhD. We then discuss the different benchmarks and evaluation metrics we propose, and how can we ground them.

Once this step is done, it paves the way to discuss our proposed framework to study styles in chapter 4. We start first by presenting the relevant literature, then move on to the experimental part, where we show the performance of the proposed framework relative to the benchmarks. We conclude this chapter by a section on style extraction, where we shade some light on what the model actually learned.

With the elements in place, we can explore the topic of style transfer, in chapter 5. As usual, we study with literature over transfer learning, followed by the proposed experiments in order address our hypotheses. We perform a wide range of experiments, to solidify our conclusions about style transfer. We also discuss another possibility to interpret what the model actually learned in section 5.3.

We conclude the manuscript by a general discussion over this thesis (chapter 6), the difficulties it faced, and the possible future research directions based on the results. We believe that this thesis answered multiple questions, but it also created more questions and research interests for further investigation. We then make a short summary and conclusions about the work done in this thesis in chapter 7.

Chapter 2

Datasets

Contents

In order to test the different hypotheses and ideas presented in the introduction chapter, we need to select suitable domain to carry out the experiments, which determined by the data. The choice of the data is a selection between different trade-offs: relevance (i.e., it contains the relevant information to perform the study) VS readiness of the data (i.e., is it already available? and how expensive it is to acquire the data?), simplicity VS realism (simple data has less noise and less irrelevant patterns, but the ability to handle realistic data provides stronger support for the hypothesis), and the amount of data available.

It is traditional in the machine learning community to use two or more datasets to address the questions. This way, it is possible to detect issues like having a very specific method that work only in a limited context[1], and to avoid the effect of unknown contributing variables.

[1] This is not wrong in itself, it depends on the objective of the work. In our case, we want to show an indication that our methods can generalize.

We settled on the domain of handwriting and drawing. In this chapter we present two datasets: online English letters handwriting dataset, *IRONOFF*, and online sketch drawing dataset, *Quick Draw!*. We present general information and exploratory statistics about both datasets, and discuss the categories/tasks in each dataset, and argue why both datasets are suitable for this study.

Points addressed in this chapter

- Present *IRONOFF* handwriting dataset.
- Present *QuickDraw!* sketch drawing dataset.
- Motivate the suitability of these datasets for this study.

2.1 Online Handwriting – *IRONOFF*

IRONOFF (Viard-Gaudin et al., 1999) is a cursive handwriting dataset provides us with isolated letters, thus allowing us to focus on the problem of styles with a reasonable complexity, and gives us the advantage that the content of the task is well known beforehand (i.e, the identity of the letter). Other cursive handwriting datasets do exist, like *IAM Handwriting Database* (Marti and Bunke, 1999). However, they use whole sentences/paragraphs, instead of individual letter, thus making the problem more complicated.

Basic information about *IRONOFF* dataset as a whole:

- Around 700 writers in total. We use the 412 writers who have written isolated letters.
- 10,685 isolated lower case letters, 10,679 isolated upper case letters, 4,086 isolated digits and 410 euro signs.
- The gender, handiness, age and nationality of the writers.
- For each writer/task (letter or digit) example, we have that example's image - with size of 167x214 pixels, and a resolution of 300 dpi -, pen movement timed sequence comprising continuous X, Y and pen pressure, and also discrete pen state. This data is sampled every 10ms at maximum, on a Wacom UltraPad A4, but the actual sampling results is not uniform[2]. Figure 2.1 shows an example for format of the the provided data.

We explored the information available about the writers in the dataset (see figure 2.2), we can see that almost all the participants are of French nationality, and majority of them are less than 30 years old. The data is almost balanced between males and females, but largely unbalanced between left and right handed people.

We then explored the handwriting examples from multiple points of view. In figure 2.3, we can see the frequency of strokes for different tasks. We can see that letters like **C** and **L** needs one stroke only, while **I** and **E** requires the most number of strokes in order to draw properly. By combining this with observation about the drawing time for different tasks (see figure 2.4) and the pausing time (see figure 2.5), we can have a good indication about the complexity of each task relative to the other tasks. The more strokes the task has, the more drawing and pausing time is needed, and the more complex the task is.

One challenging issue with this dataset however is that we have only one example for each writer-letter combination. This makes the task more difficult, because

[2]To be treated in the preprocessing step.

it is hard to extract a writer style using very few items (the 26 letters/writer in this case).

```
.DATE           08 07 1998

.COMMENT        Declarations
.X_DIM             3600
.Y_DIM             3600
.X_POINTS_PER_INCH 300
.Y_POINTS_PER_INCH 300
.POINTS_PER_SECOND 100
.COORD          X Y P T

.COMMENT        Data
.WRITER_ID      sk18091962
.COUNTRY        Germany
.HAND           R
.AGE            35
.SEX            M
.SEGMENT CHARACTER ? ? "a"
.PEN_DOWN
88 104 2 0
89 103 12 10
90 101 24 21
90 100 30 31
90 99 39 40
91 97 46 50
92 95 53 62
92 89 57 74
91 86 61 85
90 83 64 97
88 80 67 109
85 78 69 121
80 80 71 132
74 88 70 144
69 101 69 156
64 114 68 167
62 120 68 179
61 127 68 191
62 131 68 203
68 132 67 214
72 130 67 224
81 123 68 234
89 116 68 244
93 113 70 256
94 111 73 272
94 112 75 285
93 119 76 297
92 123 76 308
92 130 76 320
93 136 74 331
96 138 66 340
99 142 15 350
.PEN_UP
```

Figure 2.1: An example from *IRONOFF* (letter 'a'). To the left, we have the image of the letter (i.e., offline-handwriting). To the right is an example of the format for the online-handwriting. We have the writer's ID, origin, handiness, age and gender. The sequence of pen movement to draw the letter are then given: pen state (PEN_DOWN, PEN_UP), X, Y coordinates, pen pressure, and time.

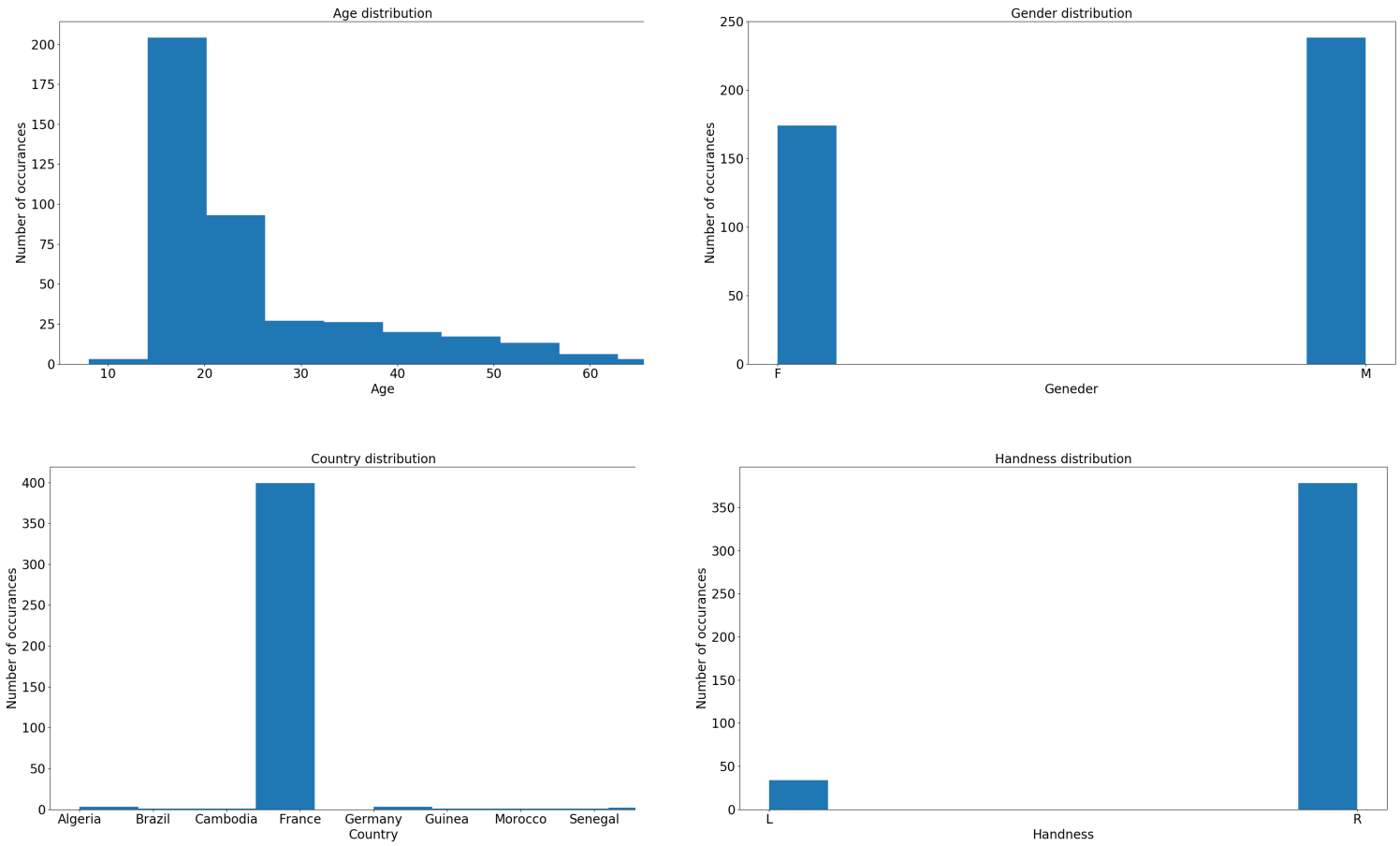


Figure 2.2: Summary statistics about the writers in *IRONOFF*: the age, gender, country and handiness. Most of participants are around 20 years old, and the majority are French. Also males are the dominant participants, the data can be concerned balanced in that regard, while on the aspect of handiness, the vast majority of the participants are right handed.

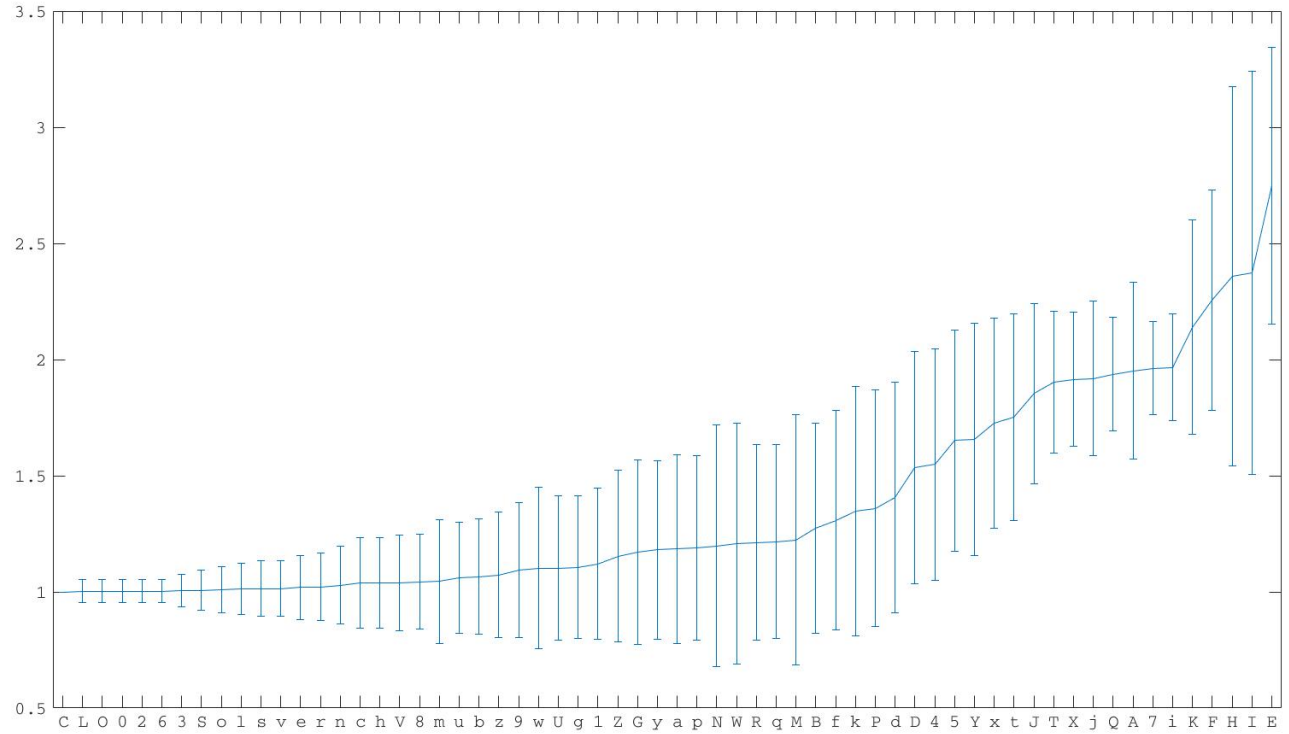


Figure 2.3: Summary statistics strokes for all categories in *IRONOFF* dataset (uppercase and lowercase letters, and digits), starting from the simplest to the more complex. We can see that letters/digits like *C, L, O, 0* are quite simple, just one mostly one stroke and a small variances, whole letters like *H, I, E* have more strokes, and more variance in the way they are drawn.

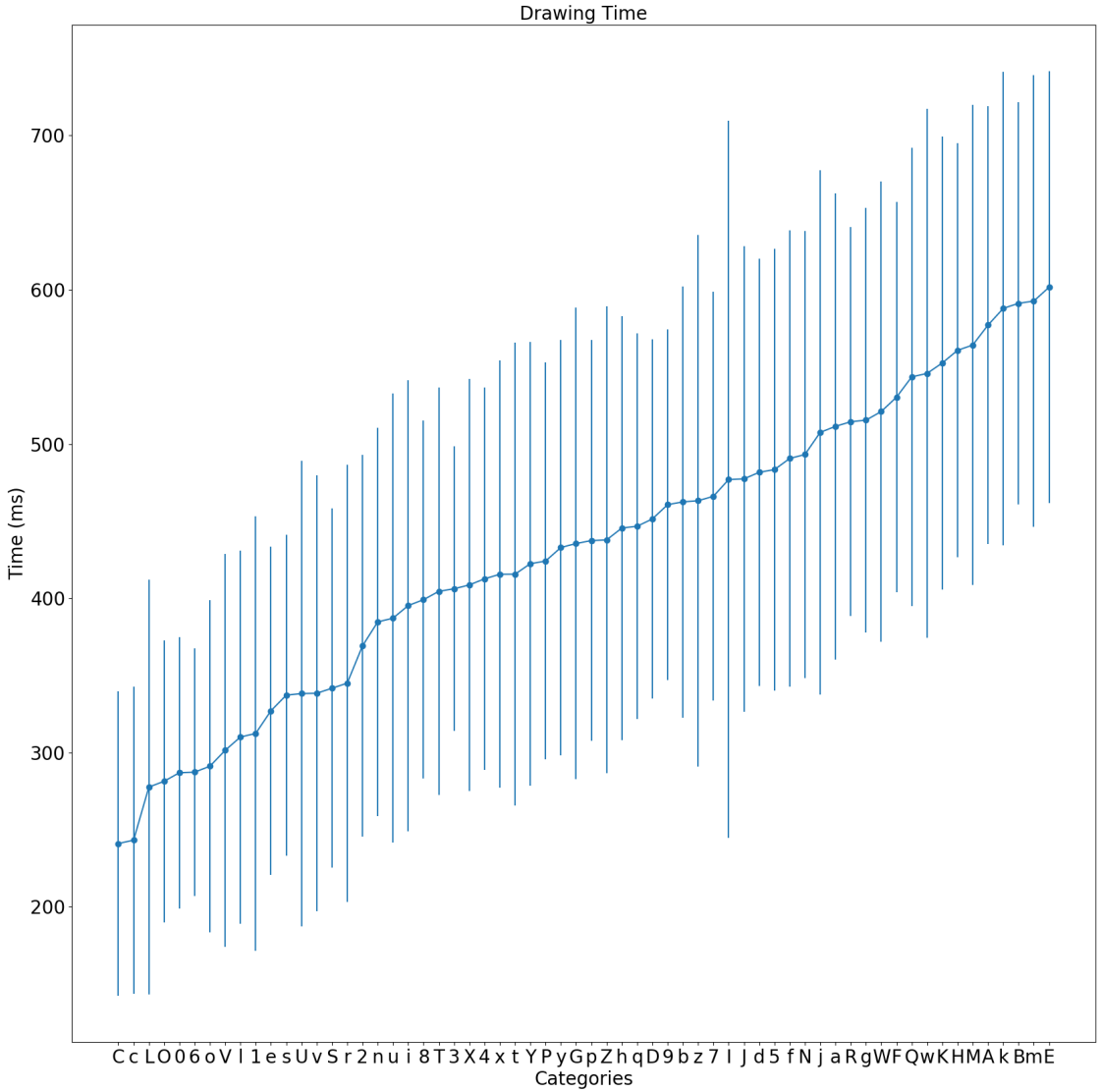


Figure 2.4: Drawing time for all categories in *IRONOFF* dataset, arranged from the smallest to the largest. Drawing time is an indication on the complexity of the task. Categories like *C,c,L,O,0* have small pausing time – since they are mostly one stroke –, while categories like *k,B,m,E* have a longer pausing time, since they do have multiple different strokes, or complex stroke – like in case of *m*–.

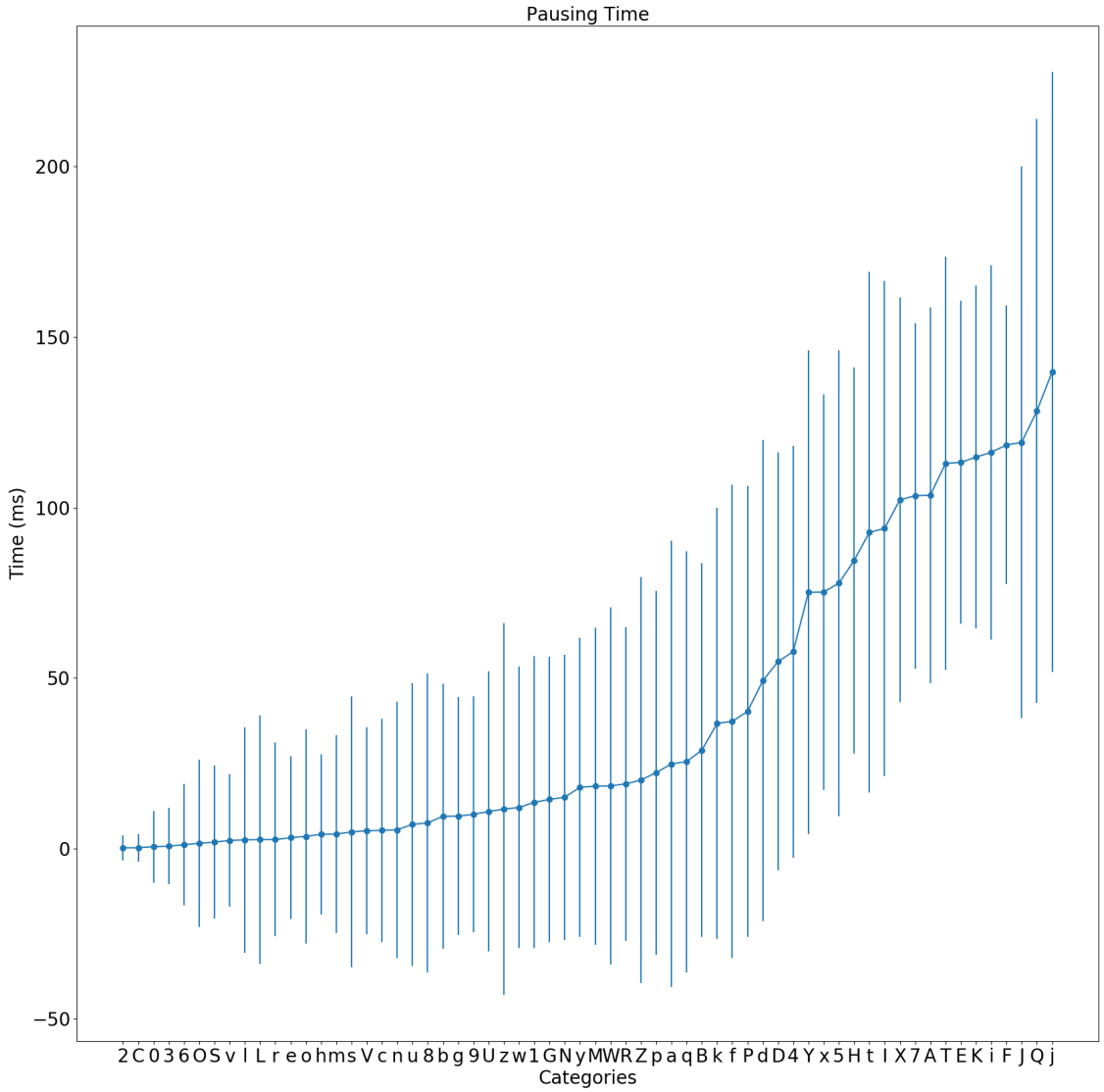


Figure 2.5: Pausing time for all categories in *IRONOFF* dataset. Pausing time is one indication on the complexity of the task, and in case of multiple strokes, they can indicate of the strokes are repetitive or different, or that that strokes are simple or complex (Séraphin-Thibon et al., 2019). Categories like *2,C,0,3,6* have small pausing time – since they are mostly one stroke –, while categories like *F,J,Q,j* have a longer pausing time, since they do have multiple different strokes, or complex stroke – like in case of *m*–.

2.2 Sketch Drawing – *QuickDraw!*

Around 50 million drawings have been collected by players of the game *Quick, Draw!* (Jonas Jongejan and Team, 2017), where players are asked to draw one of 345 categories, and a neural network tries to classify the drawings into the right categories. With more data collected and labeled, the network gets better (it learns from the flagged errors). The collected dataset is available online for free (Google, 2017). Example of the shapes collected are in figure 2.6.

Each sample in the dataset contains the following data:

- key_id: a unique identifier for this sample.
- word: the category the player was asked to draw.
- recognized: if the neural network in the game did recognize the drawing as part of this category.
- time-stamp: to mark the creation time of the drawing.
- country code: the location of the player when the drawing was made.
- drawing: an array containing the X, Y trajectories, and the time T for each point in the trajectory. Points belonging to each stroke are grouped together.

In order to focus on the styles aspect, we decided to focus on 5 categories: circle, triangle, square, hexagon and octagon. Our reasoning is that the more complex the task gets (cats for example), it is harder to have a subjective opinion about the styles, and hard to give insights about the results. This is not a limitation on the methods we are proposing though.

We sampled 20K samples from each task to perform exploratory analysis on them. In terms of strokes (see figure 2.9), we can see that there is a large variance surrounding the mean of each category. This trend continues when we look at the drawing time (see figure 2.11) and the pausing time (see figure 2.10). In the sample we analyzed, there are players from around 160 countries, figure 2.12 – mostly from US and Great Britain –. This is one indication to the increasing complexity *QuickDraw!* dataset presents in compare of *IRONOFF* dataset.

Not all examples are recognizable during the game though. In many cases, the players do not draw the required shape, or draw something quite complicated. The results of the recognition can be seen in figure 2.7, with the relation between number of strokes and length of the drawing. We can conclude that, for the selected categories,

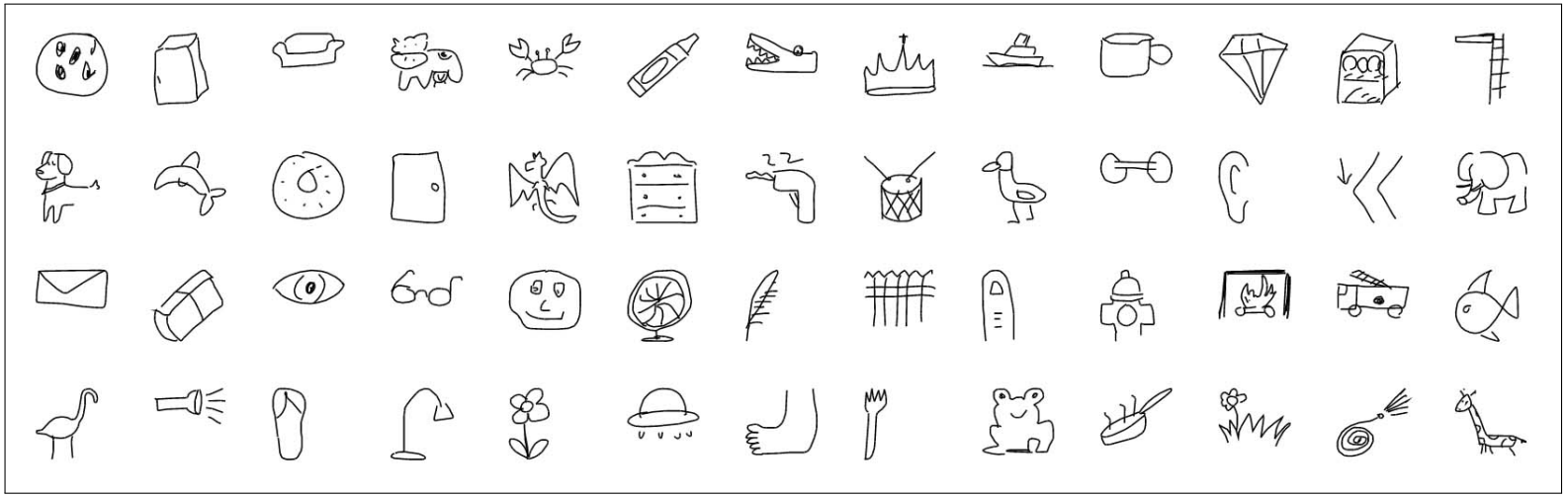


Figure 2.6: Examples from different categories in *QuickDraw!* dataset. Source of the images are (Google, 2017). All-in-all, the data has 345 categories.

the more complex the drawing is (more length or more strokes), the less likely it is to become a recognizable drawing.

This dataset is considerably more challenging than *IRONOFF*, for several reasons:

- Even though the players are asked to perform a particular task (draw a circle for example), in several cases, there is no clear resemblance between the drawing and task (e.g., when drawing an octagon, a lot of the recognized drawings do not really resemble an octagon).
- The players used a mouse in order to perform the drawing[3]. This sometimes lead to weird behaviors in terms of speed of movement (too slow, too fast), and the number of strokes (players sometimes tend to simplify complex shape, by drawing the whole shape in one stroke, and sometimes they spend too much time to draw it well, with too many strokes).

  This is unlike handwriting, where the writers usually tend to follow some rules (Séraphin-Thibon et al., 2019), which is not mostly the case in this dataset.
- Thus, some extra parts of pre-processing will need to be added in order to reduce these effects of the mouse, and make the data more closer to handwriting behavior.
- As mentioned earlier, the variance for each of the selected categories in this dataset is considerable higher than *IRONOFF*.

Since these drawings are done using the mouse, an interesting aspect for the recognizable images is the simplicity of strokes used – easier for player – (see figure 2.9). If a pen is used in the drawings, this particular behavior would not observed. For example, in case of hexagon and octagon, one can expect a higher density on the 6 and 8 strokes respectively, and much less on 1 stroke. Our observation is that it is easier with the mouse to draw the whole shape with one (or few) strokes only. This has two consequences:

- It is difficult to generate the strokes: One/few strokes means that a direct stroke representation is quite sparse (if the length of the shape sequence is 200 time steps, then among all the zeros (199 zeros, representing the current stroke), there is a single 1 value (representing the end of the stroke). This problem has also been noted in the work done in (Ha and Eck, 2017), and it is a challenging task to tackle.

[3]Although there is no reporting about the tools used in the drawing, it is a valid assumption to assume that the mouse is the main tool.

- Unlike in *IRONOFF* dataset, where the strokes is a contributing feature in identifying the letter and the rules of drawing it, the strokes are not expected to play the same rule in *QuickDraw* – unless further processing is done –.

For each class, we randomly – without replacement – sampled only from the recognized drawing, traces with less than 200 time step long. 2K samples (total is 10K samples). 1K is used for test, 900 for validation, and the rest is the training data.

2.2.1 Comparison between *QuickDraw!* and *IRONOFF*

Since participants in both data sets use different methods in order to perform the drawing/writing, it is interesting to investigate this aspect. Figure 2.13 shows a comparison between the drawing time for the different round shapes in both datasets – letters *O*, *o* and digit *0* in *IRONOFF*, and the circle in *QuickDraw!* –. We can see that the drawing speed *IRONOFF* is much faster than *QuickDraw!*. The average for *IRONOFF* is 270ms, compared to 900ms for *QuickDraw!*. Also, the variance in *QuickDraw!* is much larger than *IRONOFF*. We believe this is a result from the drawing tools used – a mouse, or a finger on a tablet in case *QuickDraw!*, and a pen in case of *IRONOFF* –. Also, a mastery of the drawing tool has an important effect on the speed; it reduces the anticipation time, allowing faster performance.

2.3 Data representation

The choices of data representation is a key factor in the success or failure of the machine learning based approaches. This choice, however, is also entangled with the task to be done (in this case, the study of styles).

A good representation tries to:

- Maximize the density of $data/patterns$ ratio: machine learning algorithms are statistical algorithms. It performs better when we have more examples for the patterns we want to learn (e.g., in case of cat/dog image recognition, having more example images for these two categories will always lead to a better performance). Another way is to reduce the number of irrelevant/unnecessary patterns to be learned from the data. This is the task of *feature selection*, which is a fundamental step in machine learning. The target is to increase the amount of signal-to-noise ratio in the data. All-in-all, the objective is to increase the ratio of $data/patterns$, either by adding more data (if it is possible, or by using synthesis data using methods like *Generative Adversarial Networks* (Goodfellow et al., 2014)), or removing irrelevant patterns.

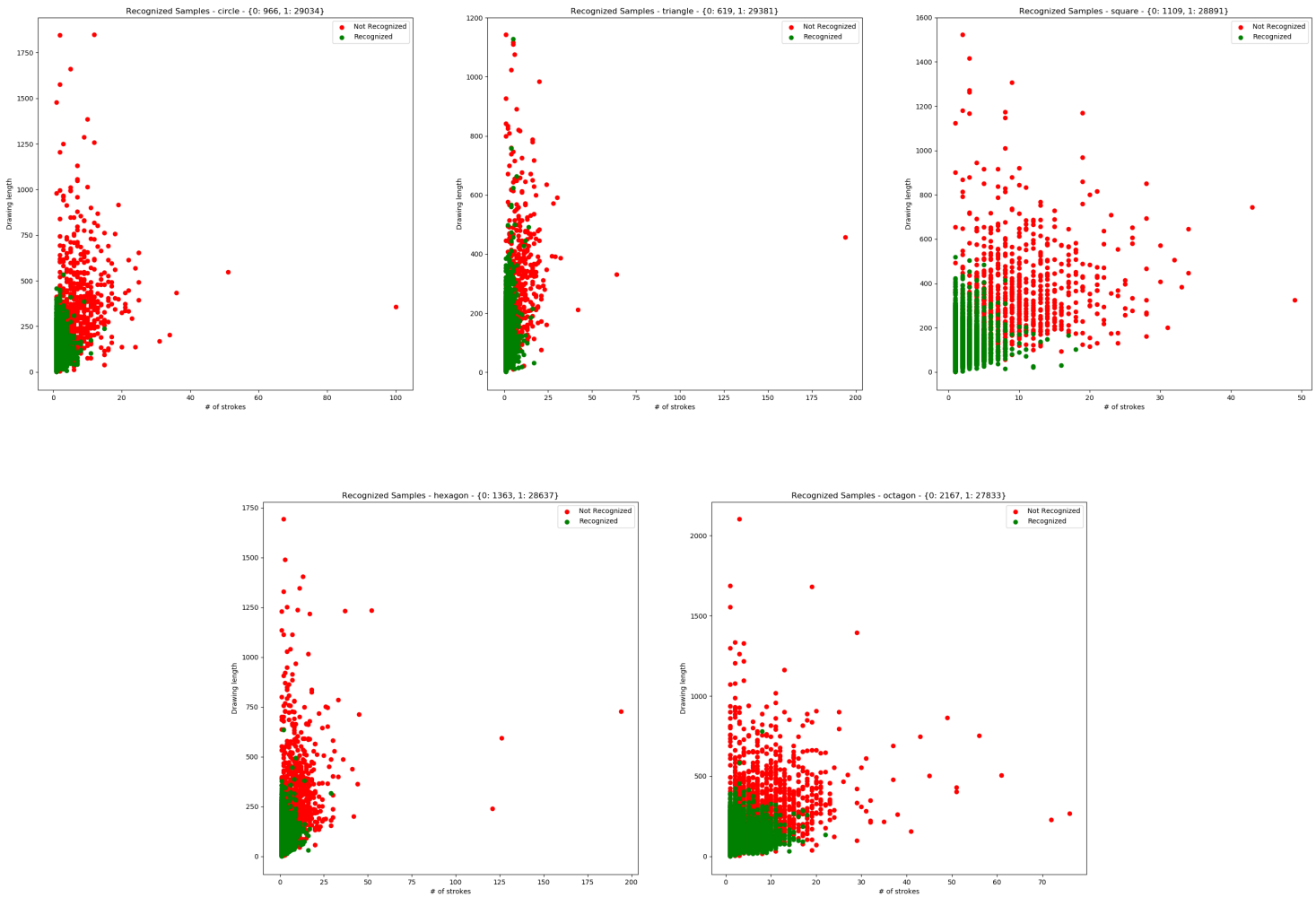


Figure 2.7: The recognized VS non-recognized drawings in *QuickDraw!* in each of the selected categories. The majority of the shapes are recognized. Unrecognized drawings usually have more strokes or longer than the rest of the drawings.

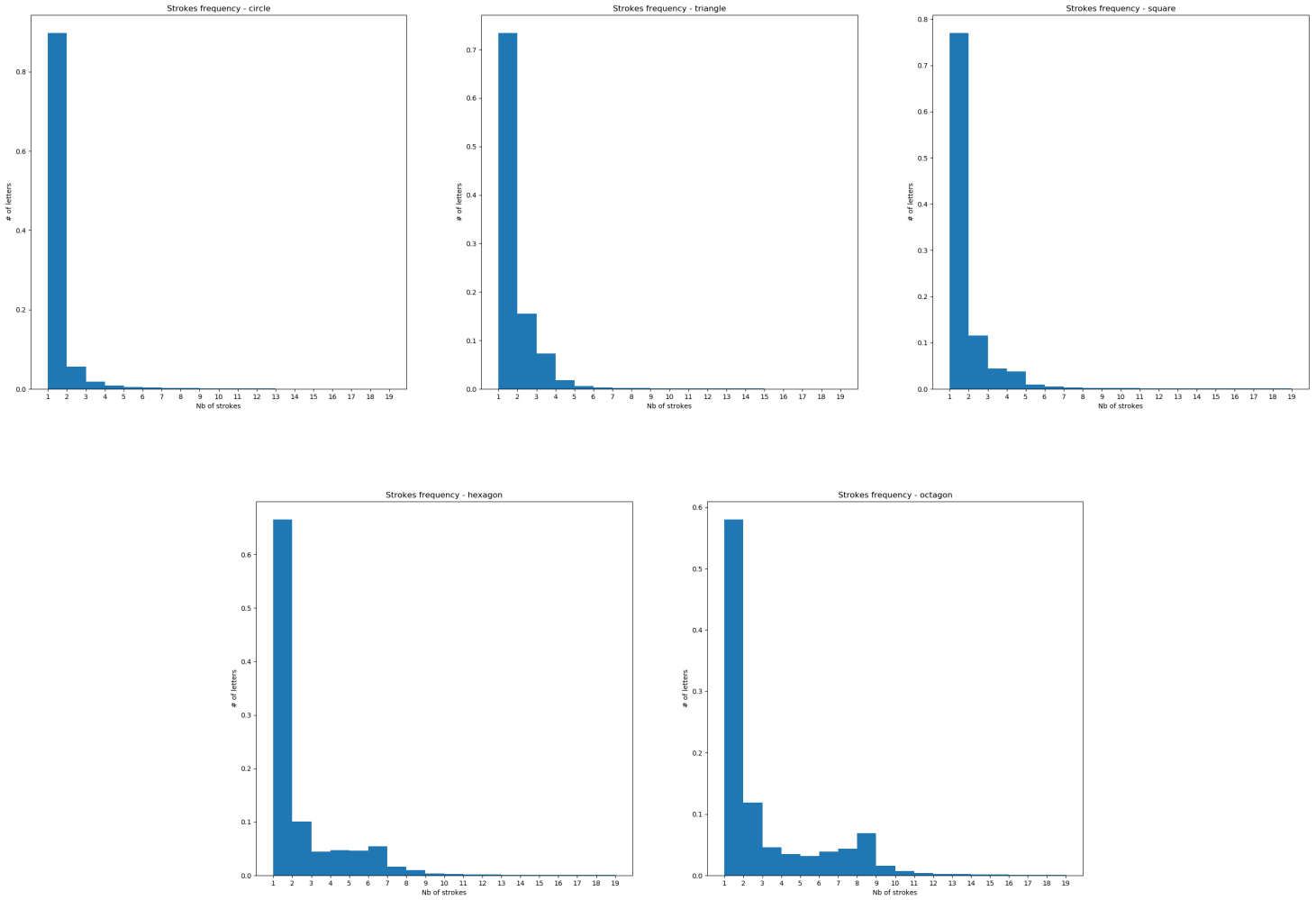


Figure 2.8: The distribution of the strokes in *QuickDraw!* for the recognized shapes, for each category. We can see that most of the drawings, for the different categories, have only one stroke, and the distribution spread slightly as the shape gets more complicated. It was expected that the strokes will have a different distribution – for the octagon, we were expecting to see majority of the drawing around 8 strokes –. We believe this is due to the use of the mouse during the drawing, and the non-mastery level of participants in using the mouse for drawing.

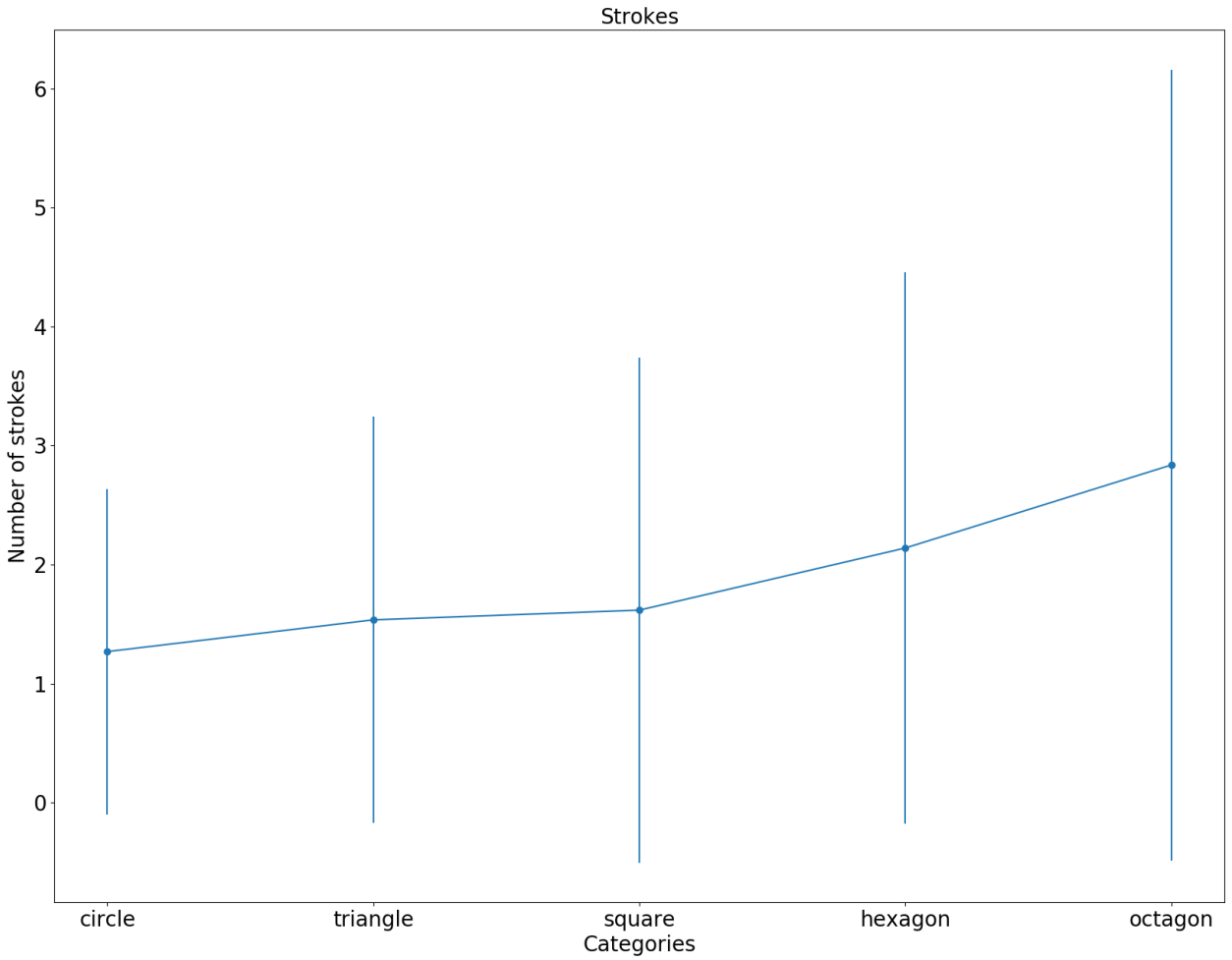


Figure 2.9: *QuickDraw!* strokes statistics for each of the selected categories. Strokes is an indication of the complexity of the drawing. Circle have the lowest number of strokes, thus the simplest, while octagon has the largest number – and largest variance –, thus it is the most complex shape.

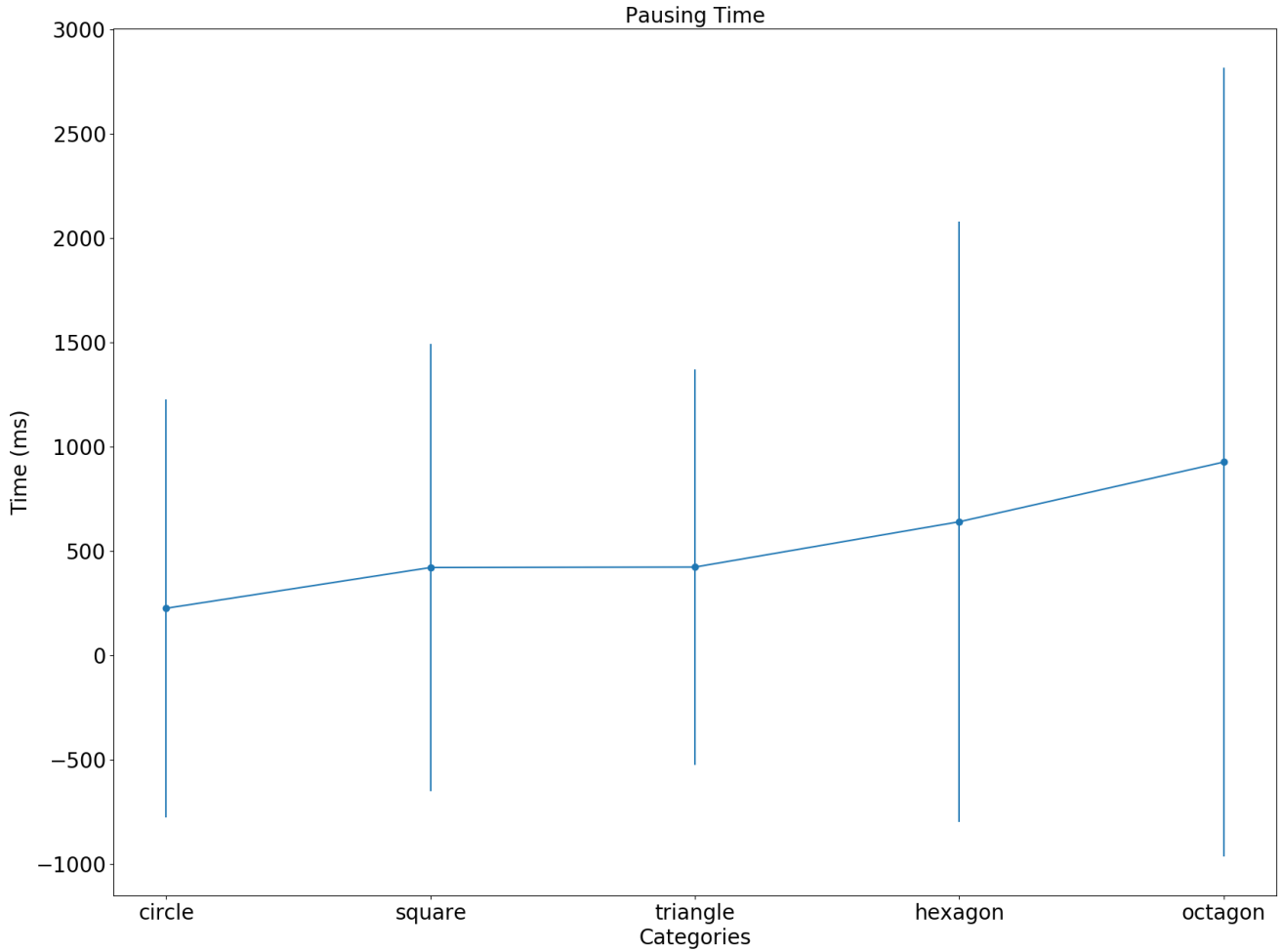


Figure 2.10: QuickDraw! pausing time statistics for each of the selected categories. It is another indication on the complexity of the shape and its strokes. Circle category has the least pausing time, while the octagon has the largest one.

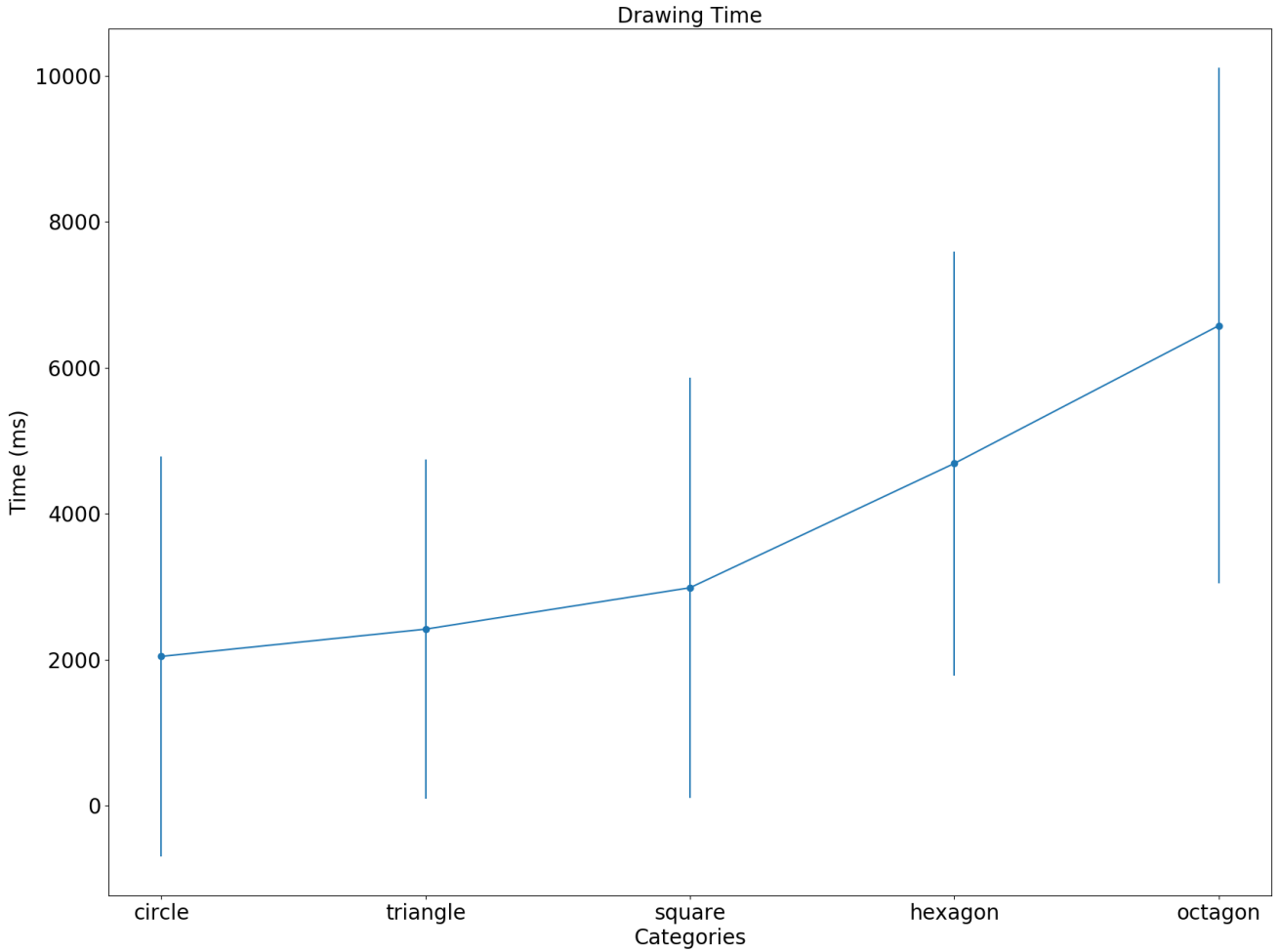


Figure 2.11: QuickDraw! drawing time statistics for each of the selected categories. It is another indication on the complexity of the shape and its strokes. Circle category has the least drawing time, while the octagon has the largest one.

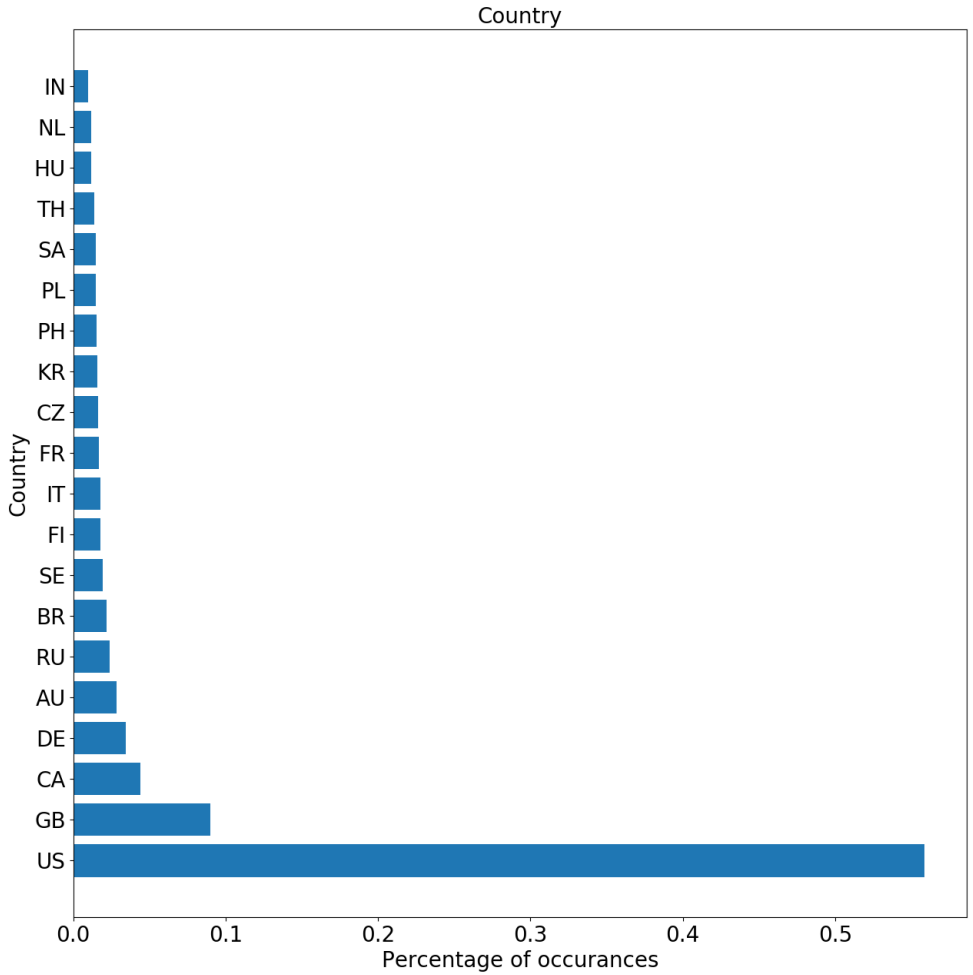


Figure 2.12: The most dominant countries for the players in QuickDraw! dataset. In the sample we analyzed, there is players from around 160 countries, but the majorty are from United States, followed by Great Britain.

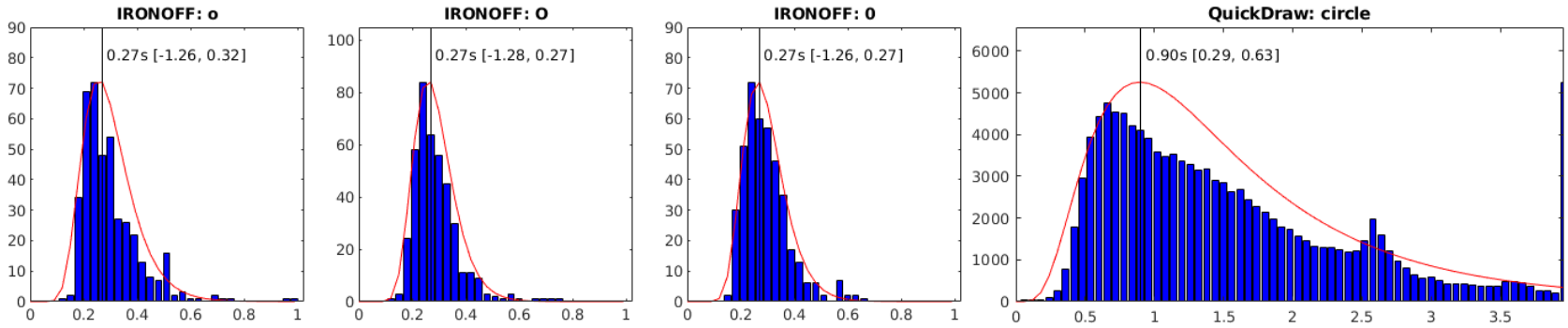


Figure 2.13: A comparison between the drawing time of different round shapes in both *QuickDraw!* and *IRONOFF*. We can see that, the average drawing time in case of *IRONOFF* (270ms) is much lower than for *QuickDraw!* (900ms). Also, the variance in the distribution is much higher in case of *QuickDraw!*. This probably a result from the different drawing tools used in both datasets – a mouse, or a finger on a tablet in case *QuickDraw!*, and a pen in case of *IRONOFF* –, and also from the mastery level of the different tools – usually people are more comfortable using the pen than the mouse in drawing –.

- Simplify the learning procedure (differentiability/richness of distributions): while working with deep neural networks provide us with a lot of power in modeling a large variety of task, it also imposes some constraints. For example, the whole learning pipeline must be differentiable (this is a limitation imposed by the optimization methods, which will be discussed later). There are a lot of already available *off-the-shelf* functions that can be used, but care must be in the design of the experiment, to make it fit with these functions. Sometimes however, these functions are not enough to model the task properly. Thus, there is a need to adapt new tools to fit within the neural networks paradigm. This including finding a proper way to differentiate them, or, if not possible, to move around the non-differentiability problem (usually by finding a surrogate function to optimize), which is not always a straightforward task for deep learning practitioners.
- Keep enough patterns to perform the intended study: it is quite tempting to focus on the scores of the machine learning algorithm, while forgetting about the original task. In our case, machine learning is a tool to help us perform our analysis, but not the final objective. For example, a low-level quantization of the X and Y traces of the pen will remove a lot of patterns, and will make it easier for the algorithm to learn the data distribution. But it will also remove essential information about the styles in this case.

In the following subsections, we will discuss two different data representation choices (continuous versus discrete representation, and the features used), and the implications of each of them on the machine learning, and the study of the styles.

2.3.1 Continuous or Discrete representation?

The X, Y and pressure of handwriting tracings are always recorded as continuous distribution, while the pen state is discrete (categorical). Thus, it probably makes sense to model the data in their native form. In the neural network design, a typical design choice will be to use a linear activation function as output function, and the *Minimum Square Error* (MSE) as loss function. Unfortunately, it is not that straightforward.

Continuous Data Representation To understand the problem, we first need to consider how a simple feed-forward neural network works, figure 2.14. The input is x, the output of the network is a, and we want to predict y. The neural network tries to project x to a new space z_{21} through a series of continuous folding of space. Then, the rule of the last activation layer is to model $P(y|z_{21})$.

Although a linear activation is very simple, $y = z_{21}$, it is also shown to have limitations. In (Bishop, 1994), a simple example is shown (see figure 2.15). If each

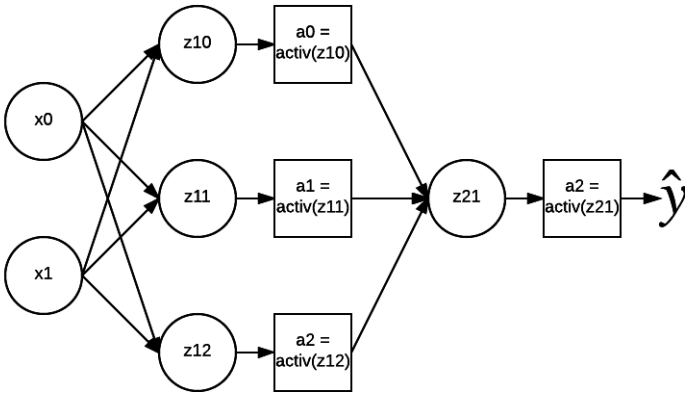


Figure 2.14: Example of a single-hidden layer neural network. The circle units is the sum of the weighted neurons connected to this unit, and the square units is the application of the activation function on that sum.

input x gets a unique output y (one-to-one mapping), then linear activation performs well (figure 2.15, left). But if the input can have multiple possible outputs (one-to-many), the model learns to average over these outputs (figure 2.15, right). The author concludes that a simple linear activation function is not powerful enough to represent a complex/rich distributions. He then proposed the use of a *Gaussian Mixture Model* (GMM) (Murphy, 2012) as the final activation of the neural network, which is powerful enough to enable modeling complex continuous distributions, and avoids the problems of a simple linear activation function. This combination of neural network and GMM is called *Mixture Density Network* (MDN). The loss function in this case is changed from the MSE to the a posterior log-likelihood of the GMM. The neural network output in this case is not the required prediction directly, but the parameters of the GMM (means, variances, weights, correlations). The required prediction is then sampled from this parameterized GMM.

The work done in Graves (2013) demonstrated a system which generates impressive results on handwriting demonstration. He used an adaptation of the MDN for temporal data. Other applications for MDN can also be found in speech synthesis (Wang et al., 2016, 2017a; Zen and Senior, 2014). While there is no question about the power the MDN approach provides, it was reported in (Graves, 2013) that training model kept collapsing (the explosion of gradients, the loss going to inf or NaN), thus requiring tweaks and tricks in order to train properly. This is in-line with the experiments I performed with MDN, leading to similar conclusions.

Discrete Data Representation Discrete representation of originally continuous data requires transforming the data via feature engineering and/or quantization step on the raw data. While it is guaranteed there will be some information loss in the discrete data, discretization provide a lot of robustness to noise (this is why it is used in digital

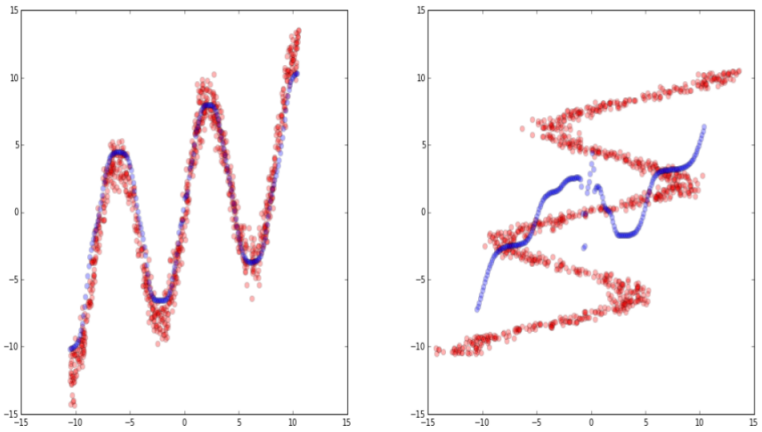


Figure 2.15: The performance of the simple linear activation function on two setups. Left: in case of one-to-one mapping between the input and the output, it performs well. Right: In case of one-to-many mapping, the function starts to average over the seen observations, leading to undesirable behavior. Source of this image is (Ha, 2015).

communication for example), and flexibility (many tools based on information theory do exist to handle digital data). Plus, a categorical distribution does not make assumption of the shape of the data distribution, unlike continuous distributions. The use of discrete distribution and how to infer from a discrete distribution will be covered later in section 3.1.

The challenge in this case is to choose a good quantization technique, that preserve the relevant information in the original signal one one side, and while keeping the dimensionality of the problem to a tractable level. For example, in case of speech synthesis, the authors in (Oord et al., 2016a,b) showed that applying the μ-law to the raw speech signal, and then quantize it (thus, the quantization is not linear), revealed a superb sound quality. This is better than performing naive linear quantization on the data, and saves a lot of memory as well: with non-linear quantization, they only need 256 levels. To get a similar behavior with linear quantization, around 65K quantization levels or required.

2.3.2 Feature engineering: Direction and Speed

As argued in the previous section, we choose discrete representation for our data. A requirement for a good quantization scheme should keep the important information in the original signal[4]

[4]In some applications, like in digital communications for example, the quantization (or digitization process) has other rules, like compressing the signal, increasing the signal-to-noise ratio, and increase the robustness of the signal. This is outside the scope of our work however.

In case of *IRONOFF*, the letters tracings has been cleaned by removing points related to false starts or corrections as well extra strokes. Tracings were re-sampled with $10ms$ time interval, and the ones with length exceeding 1 second has been removed, as well as tracings more than 100 time steps. This is because they are quite rare, thus, their existence would significantly degrade the performance of our model.

For *QuickDraw!*, the re-sampling interval is $20ms$, and consider tracings that are less than 200 time steps. This is because the tracings tend to be longer than in *IRONOFF*. This start to became clearer when the task gets more complex (the octagon being the most complex one).

We represent each letter tracing by two features: directions and speed. Each feature is quantized into 16 levels and represented as a one-hot encoded vector.

Freeman codes (Freeman, 1961) is used in order to encode the direction feature. It belongs to a family of compression algorithms called *Chain Codes*. This set of algorithms proved to be useful to encode an image with connected components. They can transform a sparse matrix to just a small fraction of the size of the image, in the form of a sequence of codes. Thus, they are being used as compression algorithms as well.

Freeman codes can N-directional codes (where N are the directions), depending on the needed resolution. It is quite simple as it encodes each direction with a unique number from 0 to N-1. A direction is defined as the directed vector connecting two neighboring pixels on the contour of a connected component in the image.

We compute the change of directions between three consecutive points. Then, we map this change to its corresponding freeman code number, as shown in figure 2.16. Last, we transform the direction number into one-hot encoding scheme, and use this as input to our network. We also quantize the speed of each displacement.

2.3.3 *QuickDraw!* strokes preprocessing

As mentioned earlier, *QuickDraw!* drawings was done mainly using the mouse, which leads to different characteristics than *IRONOFF*, mainly in terms of strokes. As shown in the exploratory analysis of the strokes distribution, the participants usually perform the drawings in fewer strokes than expects. This creates a problem for us in the machine learning part, as the strokes are very sparse, and hard to learn. The work done in (Ha and Eck, 2017) report the same issue. Early experiments on our side on the machine learning part showed great difficulty in learning the proper strokes.

In order to get around this issue, we processed the data in order to generate more balanced distribution of strokes. After some exploration, we first identify that the

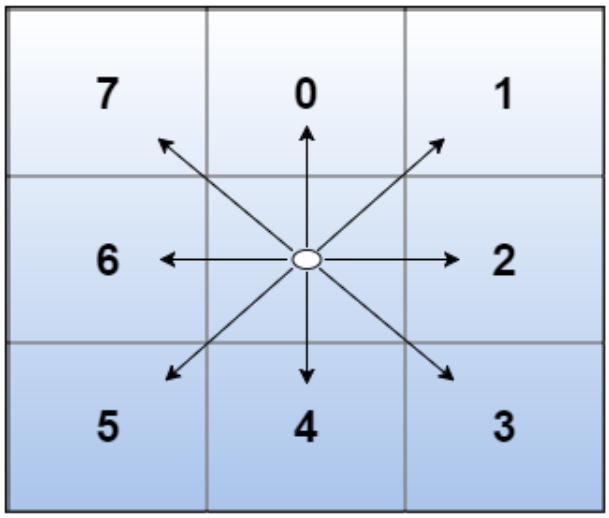


Figure 2.16: Example for freeman code representation for 8 directions. Each direction is given a unique number.

problem is that the participants do not release the stroke, instead, they wait or slow instead of performing a stroke. The period we find is 90 ms. We monitor the time used during the drawing, and search for this 'waiting' period during the drawing. We identify this waiting period as a stroke. This process helped us having a better distribution, thus, enhancing the performance of our machine learning models.

2.4 Summary

In this chapter, we explored two datasets: Cursive Handwriting dataset, *IRONOFF* and sketch drawing dataset *QuickDraw!*. Each of these datasets contains several tasks/categories (the letters/digits in case of *IRONOFF*, and the shapes in case of *QuickDraw!*), and we can explore the styles around each of those tasks. We explored basic information about each task (number of strokes, drawing time, and the pausing time).

We motivate the use of these datasets because we can see that there is a variance in these information of each task, suggesting different styles for each task. This variance differs a lot depending on the complexity of tasks: in *IRONOFF* for example, letter *c* seems to be the simplest task, thus, the variance in it is relatively not that large. This variance increases as the task get more complex (letter *E* for example). In *QuickDraw!* dataset, a similar trend can be observed (the *circle* being the simplest task, and the *octagon* being the most complex one).

Last, we argue that although that *QuickDraw!* has simpler tasks than *IRONOFF*, it is actually more complicated. We hypothesize that this is due to the fact that there is a huge variety in the players (many different countries, compared to mostly *French* people in *IRONOFF*). The players use mostly the mouse in order to draw[5], which leads to behaviors usually unobserved with hand drawing/writing. Also, in such environment, it is expected that the human curiosity will prevail, and people will try to draw complex shapes, outside the limit of the required task, in order to see if the neural network classifier will recognize it correctly or not.

We will consider that there are two hierarchies of tasks in *IRONOFF*: the first is uppercase, lowercase letters, and digits. The second is the individual letters and digits. When we perform transfer learning, we will do it on the tasks of the first hierarchy only (for computational reasons).

We will use *QuickDraw!* side-by-side to *IRONOFF* in our last part of our work in order to validate our approach and conclusions, but the first two parts will be done on *IRONOFF* only.

[5]The players did not receive any special training on drawing with the mouse beforehand.

Part II

Experiments

Chapter 3

Generation, benchmarks and evaluation

Contents

Since styles are ill-defined, they can not be evaluated explicitly (we can not quantify them in advance). Thus, we need a proxy method in order to implicitly evaluate them. We can do this by using them in order to generate behaviors (i.e., synthesis handwriting traces), and evaluate the quality of those behaviors relative to the target behaviors (i.e., the original letters traces). In other words, to study styles, we would like to reconstruct the target behaviors using generative models, and the task and the styles to perform the behavior. In this chapter, we discuss the recent advances in neural

generative models: how are they trained? how do we infer from them? Then we look at paradigm for reconstruct target behaviors, with focus on the usage of neural networks.

After we generate the behaviors, we need to evaluate them in a manner that evaluates the styles. This is still an open research question, and a complicated one as well. It also goes hand-in-hand with the choices made in generative models. We discuss the used evaluation metrics across different domains (e.g., text, speech and handwriting evaluation). We also discuss the difficulties associated with finding the suitable evaluation metric.

I then present the experiments done in order generate drawings and ground our proposed evaluation metrics. I will explain our choices for the model design, the inference methods, and our approach to ground the metrics.

Questions addressed in this chapter

- How to generate handwritten letters using deep learning framework?
- How to evaluate the generated traces?
- What benchmarks are suitable to compare to?

3.1 Background

We start by introducing the different basis in the literature for our work. Most of this work uses deep learning and generative models. How and why deep learning emerged in the last few years is quite important to keep in mind, since it is the starting point for any successful usage of deep learning. We then explore a particular aspect of deep learning: generative models. In particular, we focus on the domain of sequential data, where the problem gets more challenging. A direct consequence for using generative models is the challenge of evaluating what is generated. Generation, after all, has an artistic aspect, unlike well-defined machines learning tasks like regression and classification. We try to make sense of what exists in the literature, and try to deduce what would be a good criteria for new evaluation metrics for a generative task.

Deep learning is a subset of machine learning (Goodfellow et al., 2016; LeCun et al., 2015), mainly applicable on neural networks. These set of techniques perform remarkably well nowadays on a wide range of tasks and benchmarks (for example, in image recognition (He et al., 2016; Krizhevsky et al., 2012; Simonyan and Zisserman, 2014), speech synthesis (Oord et al., 2016a), image segmentation, handwriting recognition, image captioning (Karpathy and Fei-Fei, 2015; Vinyals et al., 2014), language translation (Sutskever et al., 2014b)), even outperforming humans in some of them (GO game (Silver et al., 2016)).

Before deep learning, in order to use classical machine learning algorithms, an important step was *feature engineering*: extracting the relevant features from the data, in order to get the relevant information needed to perform the task. In images, techniques like *Scale-invariant feature transform* (SIFT) (Lowe et al., 1999), and in speech, feature like *Mel-frequency cepstrum* (MFCC) and *Probabilistic Linear Discriminate Analysis* (PLDA) (Narang and Gupta, 2015), were quite dominant at that time. There is no one solution that fits all here; it depends on the task in hand, and that required experience. Thus, feature engineering was quite challenging.

The advantage of deep learning techniques is that it overcome (to a big extent) the need for the daunting task of feature engineering. Instead, it tries to learn, from the raw data, hierarchy of features, that are optimal in order to solve the task in hand (i.e., it performs *automated feature engineering*). In our work for example, we did not need to do any kind of temporal feature extraction[1].

A detailed discussion into the basics of deep learning is out of the scope of this document (and has been done properly in many books and tutorial available online). We refer you to the book (Goodfellow et al., 2016) for more theoretical treatment of deep learning, and to (Chollet, 2017; Géron, 2017) for a more practical aspect.

[1] When it comes to spatial features, we performed feature-extraction using Freeman codes and Speed modalities, as discussed in the previous chapter.

3.1.1 Sequential data

Sequential data appears in a lot of our daily life, for example: text, speech, the weather status, etc. In a more formal manner, a sequential data example is formed of tokens, and can be represented as $x_1, x_2, ..., x_N$, where x_n is one token, and N is the total length of this sequential data point.

Let's take a more concrete example: text. We have multiple sentences in a given text. We can consider each sentence as a data point/example. If we consider an example sentence: *the cat is eating the food*, and we define our tokens to be the words[2]. In this case, $x_1 = the$, $x_2 = cat \ldots etc$.

What is common between all the sequential data (and what also distinguish them from non-sequential data points) is that each token is dependent on the previous token/s. In general, when we want to model sequential data, we in essence want to learn the following probability distribution:

$$p(x_1, x_2, ..., x_N) = p(x_1) \times p(x_2|x_1) \times p(x_3|x_2, x_1) \times .... \times p(x_N|x_1 ... x_{N-1}) \tag{3.1}$$

In order to model such a distribution using neural networks, we can either:

1. Adding a *state variable* in order to factorize the problem. In this case, we introduce an intermediate state variable, h, which model the dependency on the previous tokens. Our objective in this case will be

   $$p(h_n) = p(h_{n-1}, x_n) \tag{3.2}$$

   This can be done using *Recurrent Neural Networks* (RNN), which will be explained later in detail (section 3.1.2). In this case, the network is looping over the tokens one-by-one, updating the state variable each time.

2. Use the whole sequential data (all its tokens) as input to the network in the same time. In this case, the network is trying to model equation 3.1 heads-on. This approach has gained popularity recently with the work done in (Gehring et al., 2017; Oord et al., 2016a), using convolution networks in order to model sequential data (speech for the first, language text for the second). The advantages of this approach is removing the sequential aspect of the network, and leveraging parallelism when treating the whole sequence. This results in faster training, while still achieving state-of-the art results.

[2] There is no rule here for how we define the tokens. We can, for example, define the letters to be our tokens. In the case of text, it is a trade-off: considering words as tokens allow the model to be more fluent, but it also means that the learning space is very large (sometimes the number of unique words to predict at each time step is in the order of tens of thousands). If we consider letters as tokens however, it makes the model job more tractable (all the letters, symbols, digits, can be in the order of tens), but it leads to less quality for the model.

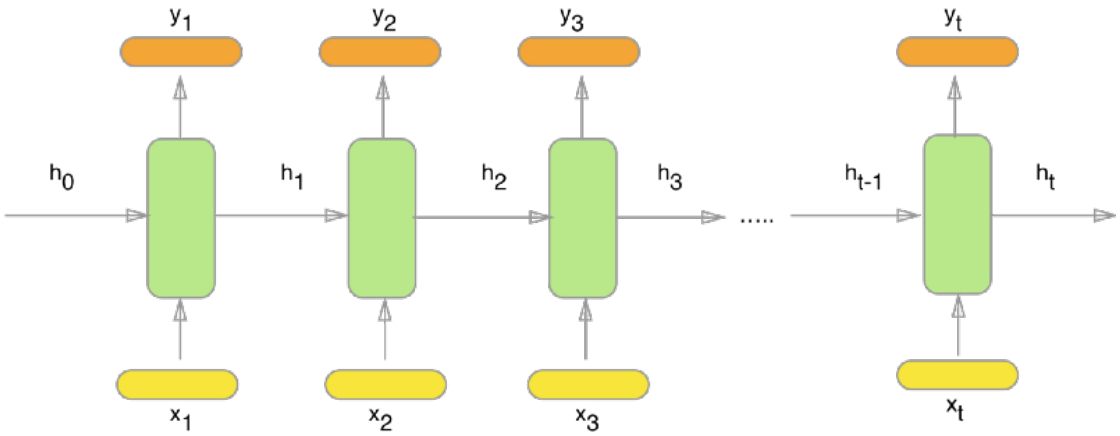


Figure 3.1: A demonstration of how RNN works: the network is applied on each token in the input $(x_1, x_2, ..., x_t)$, while update the hidden state variable every time $(h_0, h_1, ..., h_t)$. The output at each step is a function of the hidden state variable (not demonstrated here). Source of the image is from (Kostadinov, 2017).

There is no one solution that fits all here, it simply depends on the problem and the constraints on the solution. In case of variable-length sequential data, the first approach is the one to go for. In the case of fixed-length sequential data, the second approach should be considered (easier to train the model, faster to optimize, faster to infer from).

3.1.2 Recurrent Neural Networks and Sequence Modeling

As mentioned briefly in the previous section (3.1.1), *Recurrent Neural Networks* (RNN) is a type of neural networks, that can handle sequential aspect in data. In its simplest format, it is a simple feed-forward network, applied on each token in the sequential data point, while carrying the information about the previous tokens using a latent variable h, commonly referred to as *the hidden state variable*. This is demonstrated in figure 3.1.

To describe it in a more formal manner, let's first assume the following:

- The input x is first processed by a set of weights, W_{ih}, and bias b_{ih}.
- From each step to the other, the hidden state h is processed via a set weights, W_{hh}, and bias b_{hh}.
- Last, the output is given by processing the hidden state h via another set of weights, W_{ho}, and bias b_{ho}.

If we assume that output activation is a simple linear layer (thus, it is a regression task), then the equations of this simple RNN are the following:

$$\begin{aligned} z_n &= W_{ih} \cdot x_n + b_{ih} \\ h_n &= W_{hh} \cdot [h_{n-1}, z_n] + b_{hh} \\ y_n &= W_{ho} \cdot h_n + b_{ho} \end{aligned} \tag{3.3}$$

Where the brackets [ ] indicate a concatenation process. In case of a simple regression task, a typical loss function is the *Minimum Square Error*, which can be formulated as:

$$Loss = \frac{1}{T \times N} \sum_{n}^{N} \sum_{t}^{T} (y_{n,t} - \hat{y_{n,t}})^2 \tag{3.4}$$

where T is the length of the sequence, and the N is the number of sequential data points available, and $\hat{y_{n,t}}$ is predicted output by the model, while $\hat{y_{n,t}}$ is the ground truth output.

Given this formalization, the objective is to find the set of weights and biases of the network, that will minimize the chosen loss function. We will explore the optimization algorithms in section 3.1.3.

3.1.3 Optimization Algorithms

One of the important factors in the recent success of deep learning is the advances in optimization algorithms. These algorithmic advances make the optimization easier, converge to a better solution, and require less tweaking in order to achieve a good performance. All these methods are derivatives of gradient descent optimization (Boyd and Vandenberghe, 2004).

If the function to be optimized is convex (Ben-Tal and Nemirovski, 2001) – differentiable, and have a global optima –, then gradient descent can find this global optima. An example for the different optimization iteration can be seen in figure 3.2.

Gradient descent can be formalize by the following equation:

$$\theta_{t+1} = \theta_t - \alpha \times \frac{dLoss(\theta_t)}{d\theta_t} \tag{3.5}$$

where θ refers to the parameters we want to optimize, $t, t+1$ refers to the current and the next iterations respectively, α is the learning rate (how large the step to take at each iteration), and *Loss* is the loss/objective function we want to optimize our parameter for. Optimization by gradient descent is a *batch optimization*: the loss function and its

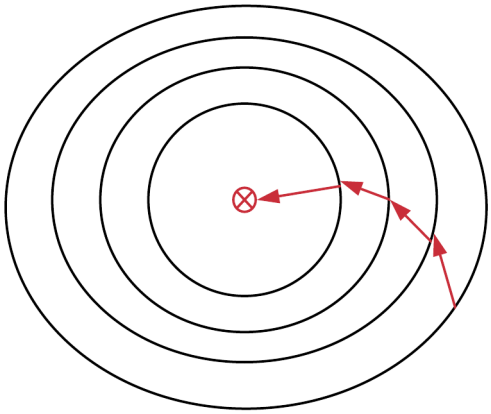


Figure 3.2: A demonstration for how a gradient descent algorithm work. The optimization process progress each step towards the global optima (the indicated point in the center).

gradient is calculated all given data examples. This iterative process keeps going till we wither do not observe a tangible change in the performance of the parameters, or we exhaust our computational budget. An illustration for this process can be seen in figure 3.3.

In case of recurrent neural networks, an adaptation from of *Back-Propagation* step is used, called *Back-Propagation Through Time* (BPTT) (Mozer, 1995; Robinson and Fallside, 1987; Werbos, 1988). It simply works by unfolding the recurrent neural network to the length of the input sequence – with each copy of the network having the same parameters –, and then apply the normal *Back-Propagation* step consequently on each step, in order to find the proper network parameters.

Before we dive into the different optimization methods currently used, it is important first to stress on an important characteristic of all modern deep neural networks, which are (Goodfellow et al., 2016):

- **Differentiability**: All the components of the neural networks (activation function, loss objective, regularization, ...etc) are differentiable components. This issue is not about mandatory by theory, but by convenience: good optimizers exist that can use this feature in order to optimize the network faster, while being able to scale with appropriately with the given number of data points and the number of parameters to be optimized.
- **Non-convexity**: While the neural network is built of convex parts, the composition of those parts together is not convex. The reasoning for this particular point is out

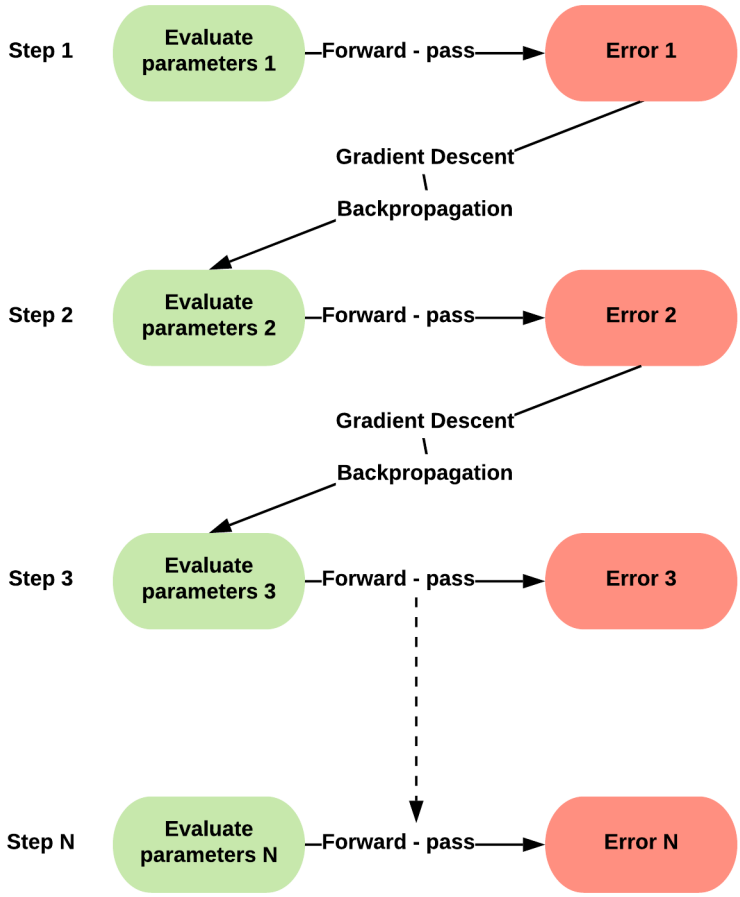


Figure 3.3: The process of gradient descent is iterative, and include 3 steps: evaluate the quality of the current parameters (forward-pass step), get the error value, use the error-value in order to update the parameters/getting new set of parameters (*Back-Propagation* step). This process is repeated N times, which is decided by either not observing any update in the error, or we ran out of computational resources.

of the scope of this document. We refer you to the book (Goodfellow et al., 2016) for more details –. While this seems to be a disadvantage, it is actually a powerful advantage of the neural network. Neural networks are *universal approximator*, meaning that they can approximate/learn any function. A convex function can not approximate non-convex function, but the other way around works (De Sa, 2017).

Using gradient descent optimization methods in this case leads to convergence to local optima points. The initial conditions of the network (the initial random parameters, and the strategy of their selection) and the optimization strategy and parameters (learning rate, decay factor...,etc) will play an important rule to determine the convergence characteristics (speed of convergence, and local optima) of the neural network.

Another important aspect of the success of deep learning is the availability of large amount of data. To optimize a deep neural network (can be the order of millions of parameters) using large amount of data (can be in the order of millions of data examples) using gradient descent, is not physically feasible. We do not have the hardware capability to process all of these data in the same time.

To get around this issue, *mini-batch* optimization is used: instead of performing gradient descent based on the information from the whole data, we divide the data into chunks (mini-batches). For each mini-batch, we calculate the loss and the gradient, and update the parameters, and then move on the next batch. The gradient descent in this case is called *Stochastic Gradient Descent*, SGD, (Robbins and Monro, 1951), following the same equation 3.5, but applied to only a mini-batch at a time, instead of the whole data (batch).

Many advances built on top of SGD helped in advancing deep learning, like combining SGD with momentum (Rumelhart et al., 1988), RMSProp algorithm (Hinton et al., 2012) and Adam algorithm (Kingma and Ba, 2014).

3.1.4 Inference: How to generate sequences from the network?

There exists several approaches in order to generate information from the networks. In the case of recurrent neural networks, this is a sequential process (one step at a time, till we generate the whole sequence), as illustrated in figure 3.4.

During the inference mode, the objective is to generate the most likely sequence. We want to solve the following problem

$$\begin{gathered}\text{Find } x_1, x_2, ..., x_N \text{ that} \\ argmax_{x_1,x_2,...,x_N} p(x_1, x_2, ..., x_N)\end{gathered} \tag{3.6}$$

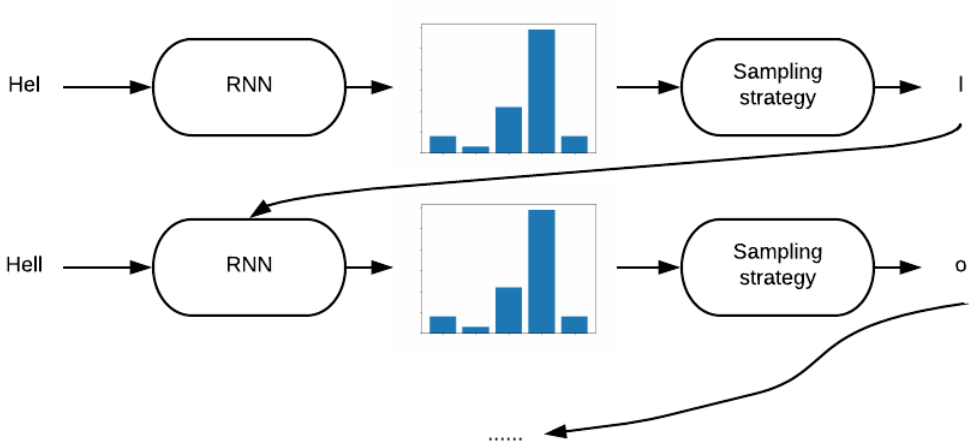


Figure 3.4: An example of using RNN in order to infer a sentence. The token to be generated here is a character. At each time step, the model is given the part of the sentence that has been generated so far, and asked to give the probability distribution over the next character. This distribution is then given to the selected sampling distribution, which sample the next character, and so on.

This problem, however, is not tractable, as it scales badly with the number of options (i.e, dimensions) per time step, and the number of time steps required. One simple way is to perform *greedy sampling*, where, at each time step, we select the most likely token. This, however, leads to repetitive and predictable patterns (Chollet, 2017).

A better way will be to use *stochastic sampling*, by leveraging the fact that at each step, we have a probability distribution over all possible tokens. This allows more diversity in the generated tokens, and also allow unlikely tokens to be sampled some of the time.

But what if we want to control the level of randomness in the sampling process? Having such control will allow us to explore different ways to infer from the model, in order to determine the most satisfying way. This control can be done using *temperature sampling*. The idea is to reshape the probability distribution over the different token. On one extreme, very high temperature (going to infinity) will flatten the distribution, making the distribution equivalent to *uniform distribution*. On the other extreme, a temperature of zero will be mount to greedy sampling. This is illustrated in figure 3.5.

A clear advantage for temperature sampling is its simplicity, ease of implementation, and ability to generate diverse outputs. However, this also create a challenge when we want to focus on issues like repeatability and reproducibility.

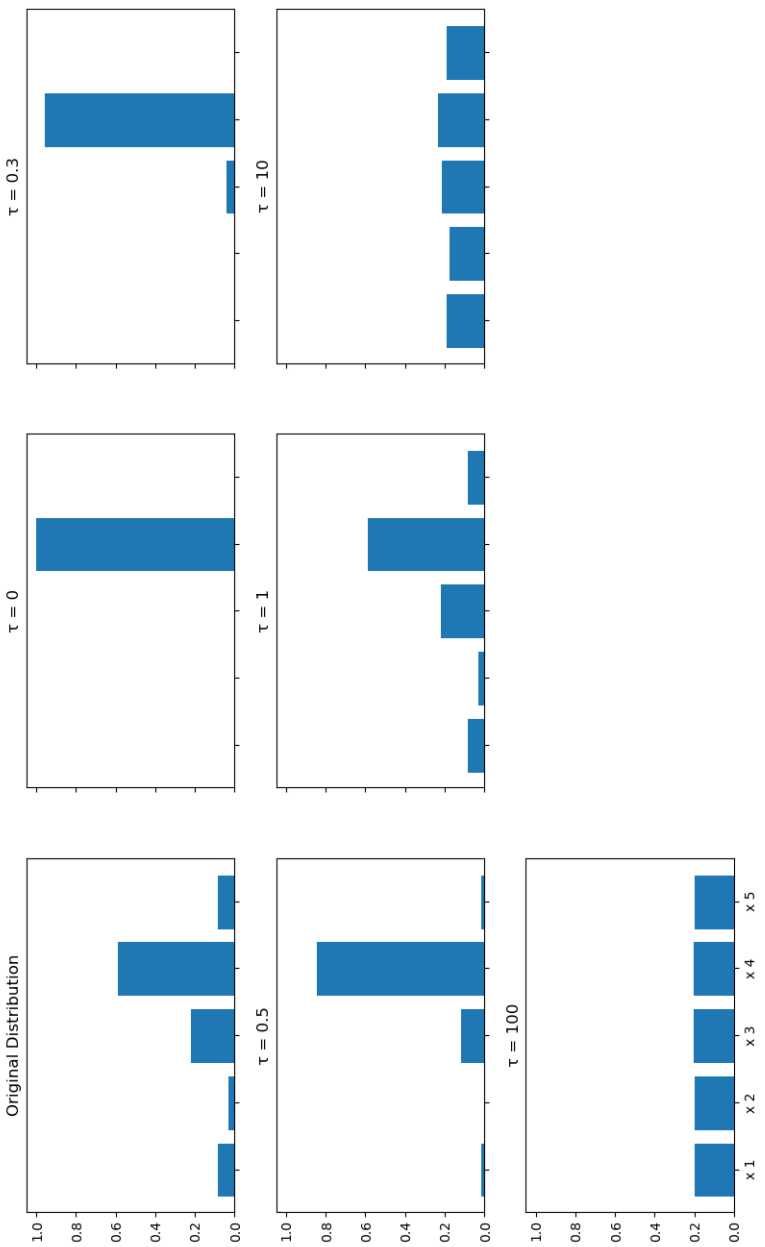


Figure 3.5: Illustration of temperature sampling. When the temperature τ is very low, it becomes greedy sampling. With the increase of temperature, we can see more higher possibility of sampling the lower-probability tokens. When the temperature is too high, it becomes uniform sampling.

Another important thing to notice is that the training/optimization part is not the same as the generation part. Usually in machine learning, there is a symmetry between the training and testing procedures. However, in generation, we are suddenly letting the model generate long sequences, and use it output from the current step as the input to the next step. The model is not trained to see its own output in the input. It is trained to see always the ground truth in the input. Overtime, there is an accumulation of errors that build up, leading to the degradation in the quality of generation (Ranzato et al., 2015). This discrepancy between the training and generation will have consequences in the evaluation process, as we will see shortly.

3.1.5 How to introduce prior to the model? (conditioning the model)

When training a RNN network, we initialize the first hidden state with zeros. This way, we are informing the model that we are not making any prior assumptions about that particular sequence, and that all sequences in the data have the same 'no prior assumption' condition.

However, what if you want to add some prior knowledge about the sequence to the model? There are two reasons why we may want to do that:

- Increase accuracy: In case we are doing some classic pattern recognition task (classification, regression), having extra useful information will definitely help increasing the model final performance.
- Act as a command: In case of generative model – during the generative mode – we need a way to *trigger* the model in order to start generating a sequence in a particular context. For example, we want to tell the model to generate letter 'A'. Initializing the model with this value allow it to start generating letter A.

There are multiple ways to bias the model, all targeting the same thing: initializing the hidden state of the model. Some work in the literature combines multiple of these approaches in the same setup. To the best of my knowledge, there is no one place which all these methods are discussed together.

Initialize the first hidden state directly This is a common approach, used image captioning (Karpathy and Fei-Fei, 2015), machine translation, and sequence-to-sequence autoencoders. This is illustrated in figure 3.61.

Using the first time-step This method was used in the work done in (Vinyals et al., 2015), in the area of image captioning. The prior in this case is the image, and the

objective is to generate the text caption for it. In order to condition the model, the authors projected the image information into the same size as the word embedding used, thus creating a *fake* word, and concatenated this new word with the rest of the words. The hidden state after this fake word is now condition on the information from the image. This is illustrated in figure 3.62.

Using context sequence – multiple time-steps – In this approach, the model is provided with some time-steps from the ground truth, in give it a context. At the end of the these given time-steps, the hidden state is initialized with information about what to be done. A use case for this scenario – for example – is when training a language model on multiple authors. When asking the model to generate, you can provide some sentences from the author you want, so the model can follow on this.

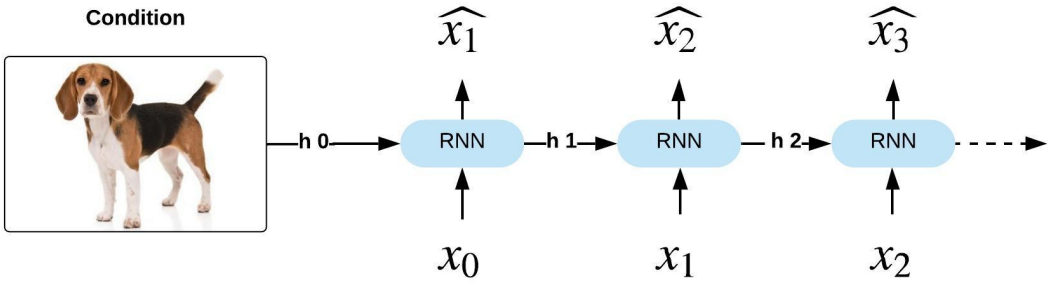
Concatenating input time-steps with the condition In the work done by (Ha and Eck, 2017), although not mentioned explicitly, the dataset used – the *QuickDraw* dataset, discussed earlier – is quite complicated – they choose different tasks than us –. It is hard to make the model remember the information about such a complex task over a long time span. Thus, the authors use a mix of *initializing the first hidden state* and *concatenating with the first time-step* in order to make it easier for the model to remember the task. This is illustrated in figure 3.63.

Concatenating the hidden state with the condition This approach is more popular now, since it allows the use of *attention mechanisms* (Denil et al., 2012; Larochelle and Hinton, 2010). Examples for this in image captioning (Xu et al., 2015a), speech synthesis (Wang et al., 2017b).

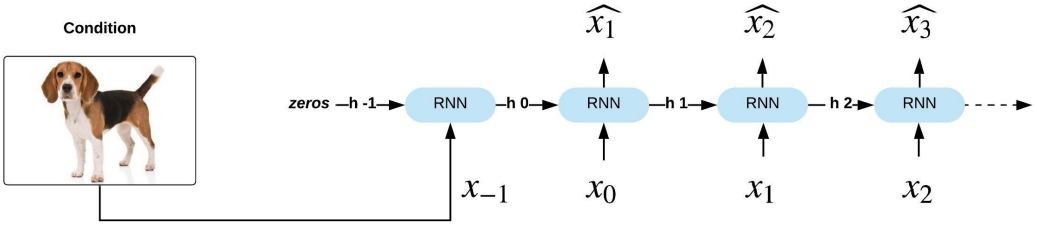
3.1.6 How to evaluate the quality of generation?

The objective evaluation of a generative model is a challenging task, since there is no consensus for objective evaluation metrics. We can not simply make strict comparisons between the generated data and the ground truth, since, as we saw earlier in the inference section, there is asymmetry between the training and the generation task. While a subjective evaluation can be a workaround to this problem, it is quite slow, expensive and hinders the development of new models and approaches. The need for objective evaluation that are consistent with the relevant subjective evaluation is thus a necessity.

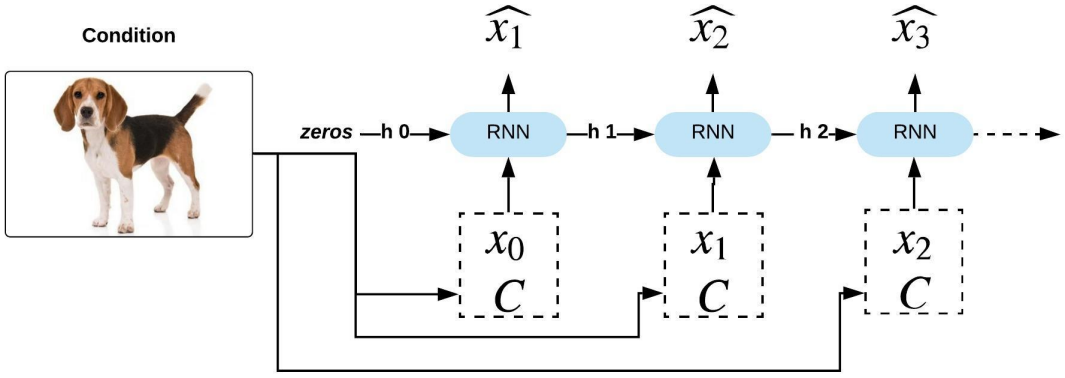
Let's see how this problem is approached in different domains:



1 Initializing the hidden state, similar to the one used in (Karpathy and Fei-Fei, 2015).



2 Initializing using the first time-step, similar to the one used in (Vinyals et al., 2015).



3 Concatenating the condition with time-step, similar to the one used in (Ha and Eck, 2017).

Figure 3.6: Different conditioning method for RNN

Text Evaluating text is an essential task to assess the quality of generation in many applications, like image captioning, language translation and language modeling. An example for an objective text evaluation metric is *BLEU score* (Papineni et al., 2002), which stands for *bilingual evaluation understudy*, it is a well known metric to evaluate text generation applications, like image captioning (Karpathy and Fei-Fei, 2015; Vinyals et al., 2015) and machine translation (Sutskever et al., 2014a). It tries to capture the human evaluation criteria for the quality of the generated text. It compares individual generated segments – the size of the segments is a parameter to set – to see if they exist in the ground truth. It does not focus on the location of that segment, just the fact if it exists or not. The small segments – letters or words – are used in order to measure the *adequacy* of the generation, while the longer segments are used to measure the *fluency* of the generation. Usually, the metrics is reported on segments from one to four words, to give and overall idea about the quality of the system.

Other objective metrics were developed and commonly used, such as *METEOR* (Denkowski and Lavie, 2014) and *Word Error Rate* (Klakow and Peters, 2002).

Speech synthesis We are not aware of commonly used objective metrics in this area. A commonly used subjective criteria is Mean opinion score (MOS), which is the average of individuals' opinions on the quality of the system. As an example, the *Blizzard Challenge* (of ISCA , the International Speech Communication Association) is an annual competition in speech synthesis, to encourage the development of better speech synthesizers. The evaluation is much more detailed that MOS, by trying to assess multiple levels for speech, like intelligibility, naturalness, utterance and the efficiency of the speech.

Handwriting of offline Chinese letters Chang et al. (2018) proposed two metrics to evaluate the quality of their generated letters: content accuracy and style discrepancy. For the first metrics, they train an *evaluator* model on the ground truth data, and use it to recognize the letters produced by their generator. For the second metric, they follow an approach developed in Gatys et al. (2015), where the authors studied the problem of image style transfer, by measuring the correlation between different filter activations (in convolution neural network) at one layer, which represents the style representation.

I would like to phrase the objective from these metrics here as *the desire to capture the distance between the generated and the ground truth distribution*. It is not important if some individual mistakes happens in the generated sequence, what matters is to capture the essence of the ground truth distribution. That being said, for complex distribution, capturing the distribution, or even measuring the distance between two distributions, is not always a tractable task. As noted in the speech synthesis point, sometimes it is better and more practical to consider multiple criteria in the same time in order to better understand and evaluate the behavior of the system. This point will later reflect our decisions concerning the evaluation criteria for our work.

Another way to perform the evaluation seen in the literature – like in (Wang et al., 2018) – is to use a machine learning model (sometimes called the *Oracle*), trained on the task needed (recognizing the speaker in the synthesized sound for example). In that case, the model is trained on the ground truth, and used in order to evaluate the samples generated by the model. To put simply, I strictly disagree with such approach, for two reasons:

Improper data distribution It violates the rules of machine learning: if the model is trained on some distribution, then we expect it to perform well on that distribution. What another model generates is simply a different distribution from the ground truth (even though we want this distance to be as close as possible). Thus, the model behavior in this case is not covered in the statistical learning theory. If the oracle – on the ground truth – has an accuracy of 90%, and then we change the data distribution, we can not really trust the prediction of the system, it is no longer the 90% percent, and we do not know what it is.

The problem continues when we have two generator, and we want to determine which is best, using an oracle, then we can not really compare them. If one generator achieves 80% and the other is 85% accuracy, we can not make a conclusion about which is better. The process itself is flawed. Besides, we do not have access to confidence level on the quality of the oracle, thus it is not possible to perform such comparison using the oracle.

Adversarial problem The problem gets worse when we use the oracle in order to evaluate the generator, then use this value as feedback to the generator in order to "improve" it. We experimented on this part to better understand it. Check appendix C for more details.

The takeaway messages is: it is not about numbers and face value. It is about what these numbers actually means and implies. A model is an approximation of ground truth distribution, and not the ground truth in itself. Thus, dealing with it as the ground truth will easily lead to a loss

3.2 Putting it all together

In the previous section, we explored multiple building blocks for the work to come, like recurrent neural networks (architectures, training, inference and conditioning). We also discussed the issue of evaluation the output of generative models, discussing three aspects: evaluation in case of text (image captioning, translation, and text generation in general), speech synthesis, and the dangers of using another model (oracle model) in order to evaluate the generation quality.

In this section, I discuss how to put all these elements together, the decisions made, and the experimental setup used, in order to address the following three questions:

- How to generate handwritten letters using deep learning framework?
- How to evaluate the generated traces?
- What benchmarks are suitable to compare to?

Our contributions will be on how to evaluate the generated traces, proposing multiple benchmarks, and using the prior knowledge about the power of those benchmarks in order to ground the the proposed evaluation metrics. The work done in this chapter was published in (Mohammed et al., 2018).

3.2.1 Our proposed evaluation metrics

Evaluation is a challenging problem when using generative models. We want metrics to capture the distance between the generated and the ground truth distributions. One of our contributions is to propose using the following metrics:

BLEU score As mentioned earlier, BLEU score is used in the evaluation of text generation. Since we discretized the letter drawings, this fits nicely within our work. The general intuition is the following: if we take a segment from the generated letter, did this segment happen in the ground truth letter? We keep doing this for segments of increasing length (the length of the segment here is the number of grams used in the BLEU score). For our work, we report the results on segments from 1 to 3 time steps.

Each part of the letter has two parallel segments: freeman codes and speed, thus, we report the BLEU score for both of them. The equation to compute the BLEU score is the following:

$$BLEU_N = \frac{\sum_{C \in G} \sum_{N \in C} Count_{Clipped}(N)}{\sum_{C \in G} \sum_{N \in C} Count(N)} \tag{3.7}$$

$$Score_N = \min(0, 1 - \frac{L_R}{L_G}) \prod_{n=1}^{N} BLEU_n \tag{3.8}$$

where: G is all the generated sequences, N is the total number of N-grams we want to consider. $Count_{Clipped}$ is clipped N-grams count (if the number of N-grams

in the generate sequence is larger than the reference sequence, the count is limited to the number in the reference sequence only), L_R is the length of the reference sequence, L_G is the length of the generated sequence. The term $\min(0, 1 - \frac{L_R}{L_G})$ is added in order to penalize short generated sequences (shorter than the reference sequence), which will deceptively achieve high scores.

In order to get into the intuition of using such a metric in evaluating the quality of handwriting generation, we can imagine that we are comparing segments of a generated trace to segments in the ground truth letter, by asking the following question: does the segment in the generated trace exist (anywhere) in the original trace? The shorter segments represent *adequacy* (i.e., does the generated trace use the same elementary moves like the ground truth trace). The longer segments represents *fluency* of drawing (i.e., how good the drawing is)[3]. See figure 3.7.

Figure 3.7: Using BLEU score of different sizes, we compare segments of variable length in the generated trace to the target trace. The smaller BLEU scores evaluate the adequacy of the drawing, while the bigger BLEU scores evalute the *fluency* (Papineni et al., 2002).

End of Sequence The length of the letter is another aspect of the style. The distribution of length in the generated examples should follow the ground truth examples. In order to perform this analysis, we compute *Pearson correlation coefficient* between the generated examples and the ground truth data.

3.2.2 How to ground the metrics?

We assess multiple methods to condition our handwritten letter generator, and evaluate their ability to capture of writing styles. We know their cardinal order of the power of these methods (depends on the kind of information available to each method). Knowing this information beforehand, we can use it to ground our performance metrics. The methods are:

[3]This last interpretation of short and long segments is adapted from the (Papineni et al., 2002), which uses them for text translation evaluation.

Letter identity : the letter id only is used as bias. No style information is thus included. The model will try to average over the different example for the same letter. We consider this as a lower baseline.

Letter + Writer identities : the letter id and writer id are used as a bias. Thus, the model has an explicit access information about the writer. This method is expected to perform the best. This model will also serve as a upper baseline.

Image classifier embedding : we train a convolution neural network (CNN) to classify the letters images[4], as shown in figure 3.8. We use an intermediate layer as to extract embeddings, that will encode information about the letter images. This model should perform the same or a more performance than using the letter identity only, since it learns to clusters the letters, and there are classification errors. But we expect it to perform less than the letter + writer identities.

Image auto-encoder latent space : we train a letter image autoencoder, using reconstruction error, and use the latent space as a representation of the letter + style. The architecture we use can be seen in figure 3.8. The latent space encodes the similarity between the letters. This model should perform worse than using the letter identity only, since, while it capture the similarity between the letter images, it does not capture discriminative features about each letter itself.

From this discussion, we can say that cardinal power of the different conditions is:

$$autoencoder < letter \leq classifier < letter + writer \tag{3.9}$$

3.2.3 Proposed model

We use a type of RNN called Gated-Recurrent Network (GRU) (Chung et al., 2014), which is known for having a better memory ability than basic RNN, thus, making it suitable choice for long sequence. Our model is a conditioned-GRU model, demonstrated in figure 3.9. Using this model, we compare different style approached discussed in section 3.2.2, and use the generation results from that model in order to ground the proposed evaluation metrics, discussed in section 3.2.1.

3.2.4 Results

We train our different models and generate the traces from them as explained earlier. In this section, we compare the different models using the evaluation metrics discussed

[4]This letter classification task achieves 95.1% classification accuracy, which we consider very good.

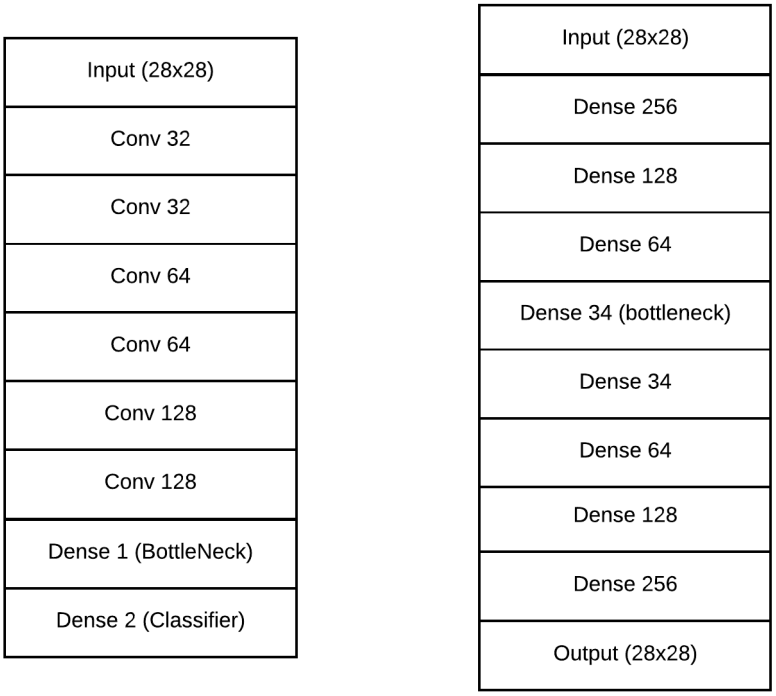
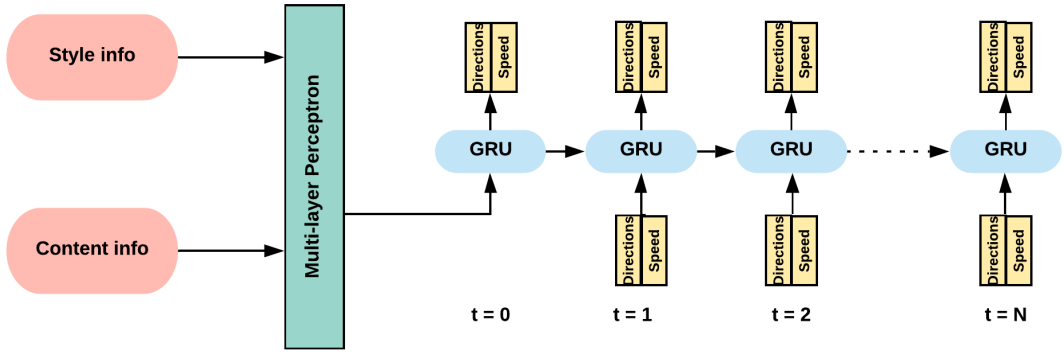
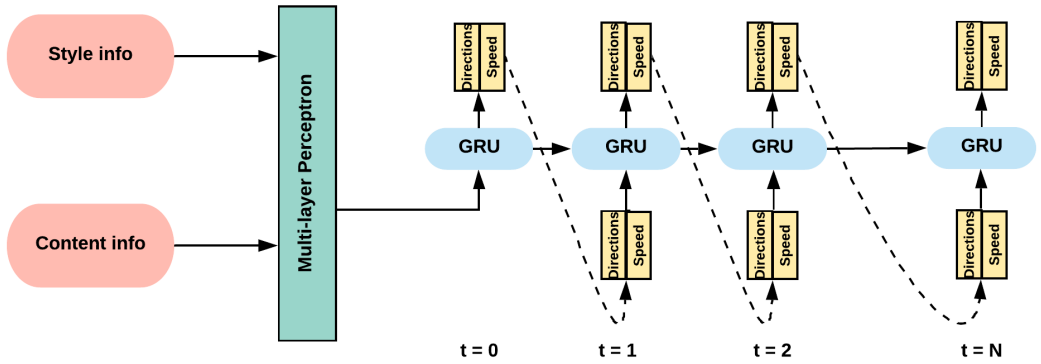


Figure 3.8: Left: architecture of the CNN letter classifier we used. Batch normalization is used after each convolution layer. The *Dense 1* layer – with a size of 34 – is the embedding that is used to condition our generator. Right: the autoencoder architecture we used. The output of the first *Dense 34* layer provides the latent space used to condition the generator.



1 Training mode, using *teacher forcing* methodology (Goodfellow et al., 2016; Williams and Zipser, 1989), where the model gets its input from the ground truth all the time.



2 Generation/Inference mode, where the model gets its input at each time step from its own output at the previous time step.

Figure 3.9: The conditioned-GRU model used in this work. During the training mode 1, the input of the model is always the ground truth, and the predicted value is compared to the ground truth. During the generation mode 2, the input to the model at each step is the model prediction from the previous step. The 'style info' and the 'task info' inputs are separated here for demonstration (they could be mixed).

before. We observe the consistency of the reported metrics with the prior information about the cardinal power of the different methods, equation 3.9. This is how we ground our metrics.

BLEU score

The final results using the BLEU score can be seen in table 1. The results vary when measuring BLEU-1. But, as we increase the number of grams, BLEU-2 and BLEU-3, to measure the similarity between larger segments of the traces, we can observe:

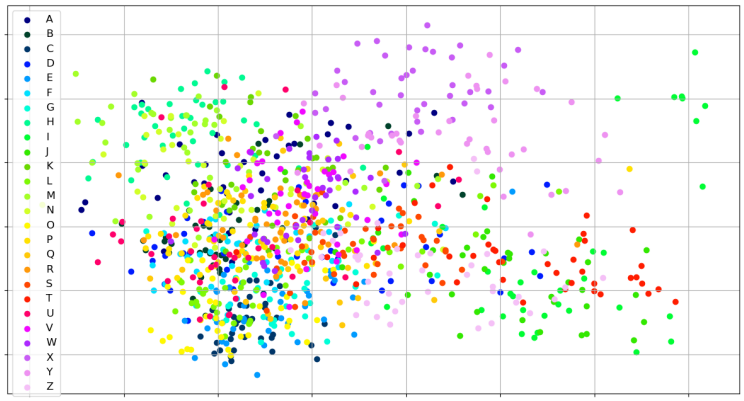
- The letter + writer condition performed better than all other conditions, thus showing that having access to information about the writer, like the writer id, improve the quality of the handwriting synthesis.

- The image classifier condition performs better than the letter identity only, but less than the letter + writer bias. Since the classifier is trained on a single objective only (to classify the letters), and the classifier performs well, we expect the embedding to cluster the letters well, as seen in figure 3.102. We can expect the model to capture some of the writer style, possibly in the inter-cluster variance. This is an interesting result, suggesting that some fine tuning for the image classifier while in the generation task could be beneficial to capture more details about the styles.

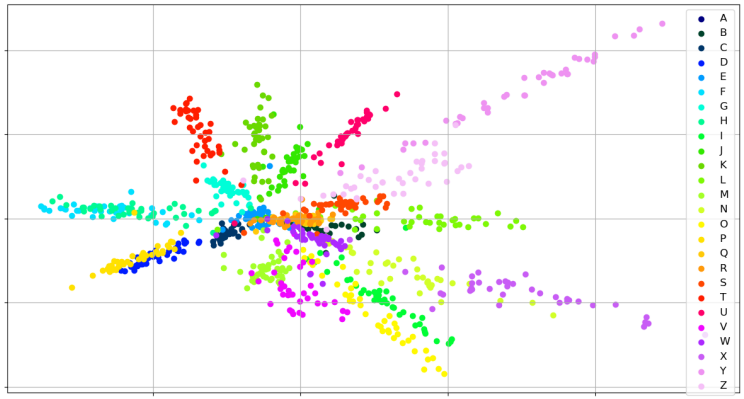
- The image autoencoder bias performed the worst. To understand why, we plot a 2-D projection of its latent space using t-SNE (van der Maaten and Hinton, 2008), figure 3.101. Since the autoencoder is trained to minimize the reconstruction error, the distance in the latent space encode the proximity between the images. It can be observed also that this latent space does not encode discriminative features for the letters. Using this latent space for our generator, we find the model gets confused between nearby letters, resulting sometimes in generating different letters than requested.

Aspect/Feature	Speed			Freeman		
Model / B-score	B-1	B-2	B-3	B-1	B-2	B-3
Letter identity	49.7	37.3	24.2	47.4	36.6	26.8
Image classifier	50.9	38.2	24.6	48.5	37.9	28.1
Image autoencoder	**51.9**	37.9	23.1	46.4	35.0	24.5
Letter + Writer identities	51.5	**41.4**	**25.1**	**56.7**	**39.4**	**28.3**

Table 1: Comparing different approaches for style extraction using clipped n-grams



1 Bottleneck of the auto-encoder



2 Bottleneck of the classifier

Figure 3.10: Figure 1 shows the autoencoder latent space, there is no clear separation between letters; the encoding is based on the similarity of the images only, while figure 2 shows the classifier embedding, there is a clear separation between the letters - with few exceptions -.

Sequence length

As mentioned earlier, we performed a statistical test between the paired distributions of lengths of the generated and the reference tracings. The results are shown in table 2. We can see the following:

- The results from the statistical test shows that the letter + writer bias outperform the rest of the biases, achieving p-value < 0.05. This is quite reassuring, since it is also in line with the results from the BLEU score.
- The results from the Pearson correlation coefficients are also consistent with the rest of the results. High coefficients are given to the letter + writer bias, compared to the other methods. The image classifier and autoencoder gives the lowest results. This could be due to insufficient information about the letter length that can be inferred from the image. For the image classifier, as noted earlier, a fine-tuning during the generation task is worth exploring.

Models	Pearson coefficient	p-value
Letter bias	0.38	0.84
Image classifier	0.32	0.62
Image autoencoder	0.25	0.29
Letter + Writer bias	**0.55**	**0.04**

Table 2: Pearson correlation coefficients and associated p-values for the EOS distributions of the different style biases. A *letter + writer* bias performs much better than others biases, while the *image autoencoder* bias performs the worse. This confirms with our expectations about the relative power of the different biases.

3.2.5 Examples of the generated letters

The design choices of our experiments (discretization, and ignoring the pen state) affects the final shape of the letters, yet, the letters and their style are quite recognizable. See examples for the original letters in figure 3.11. Examples for the generation with our methods are in figure 3.12. This is a subjective indication that our model is working properly, producing real-like comprehensible letters.

3.3 Summary

In this chapter, we sit the foundations for the work done in my thesis. I first discussed the relevant areas in the state-of-the-art concerning recurrent neural networks, opti-

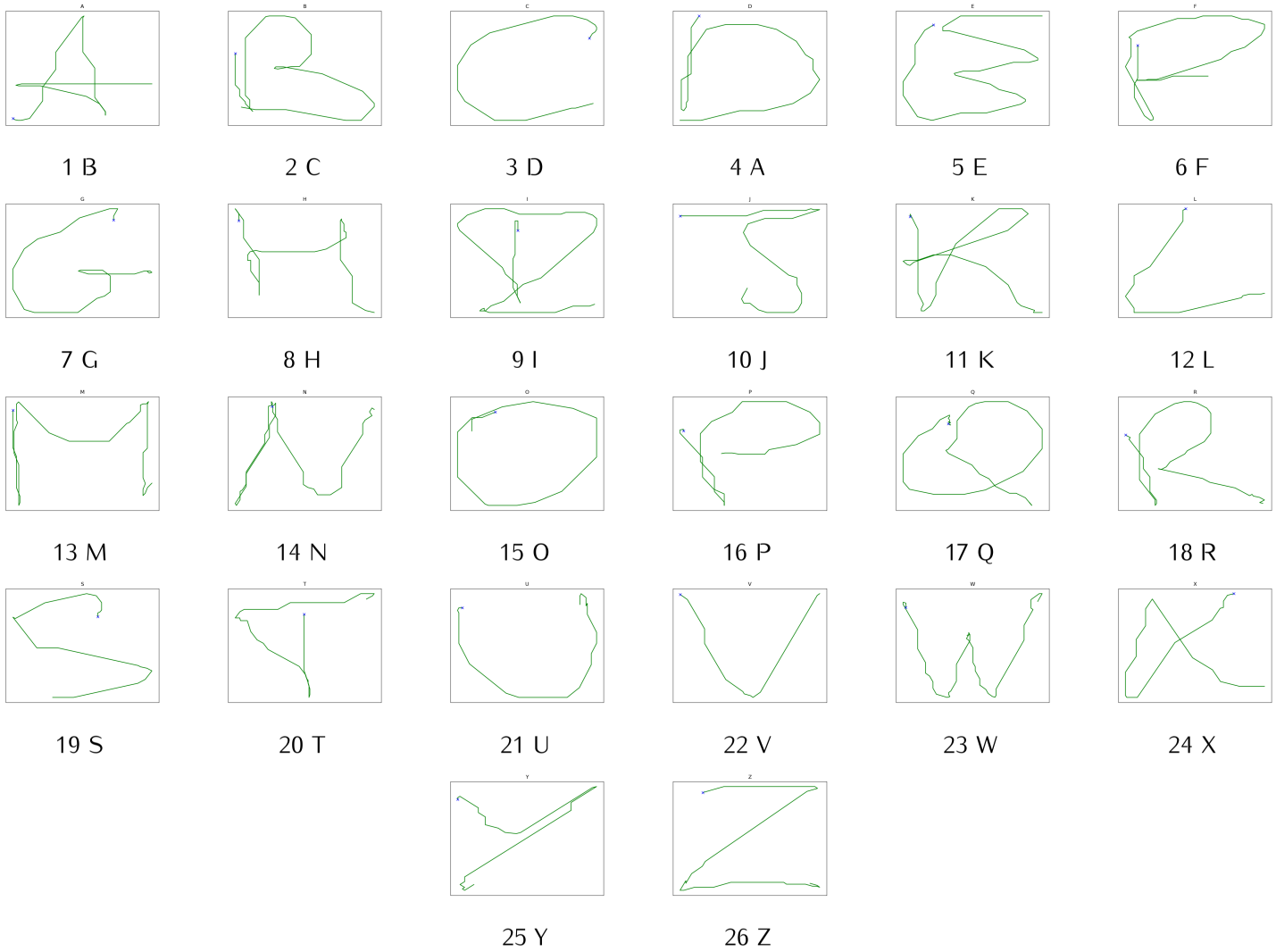


Figure 3.11: Examples of original letters. The blue *x* mark is the starting point. These ones are generated using the letter + Writer bias. E and F are visually harder to recognize, since we do not model the pen pressure, otherwise, the rest of the letters are well recognizable.

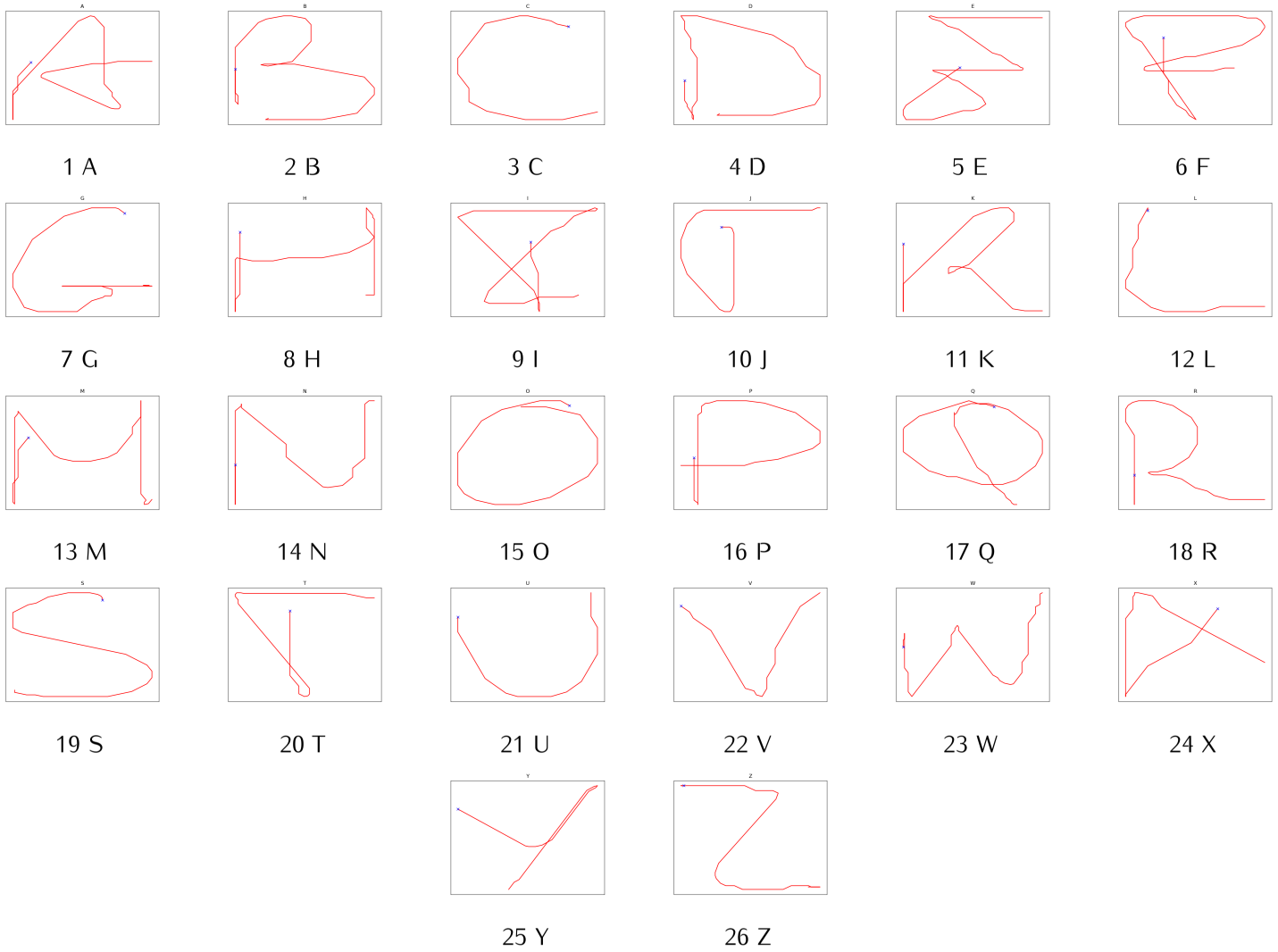


Figure 3.12: Examples of generated letters. The blue *x* mark is the starting point. These ones are generated using the letter + Writer bias. The general quality of this quite acceptable.

mization, inference/generation and evaluation of generation. I then detailed how we combine these elements in order to address the questions in this thesis. We addressed three points:

- How to generate traces using deep generative models?: we proposed to use a conditioned-GRU model. I explained the process behind training, optimizing and choosing hyper-parameters for this model, and showed some of the generated letters.
- How to evaluate the quality of the generated letters?: we proposed two evaluation criteria, BLEU score metric – inspired by its usage in evaluating text quality – and End-of-Sequence quality, as a way to capture some aspects of the distribution, and a feasible way to measure the distance between two distributions.
- We proposed multiple benchmarks for future evaluation of style extraction models, and to ground the proposed evaluation metrics as well.

Now that we have our benchmarks and evaluation metrics, it is time to go for the next chapter...

Chapter 4

Framework

Contents

In the previous chapter, we set the foundations of studying styles, which is the work done in the PhD. We explored our choices for generative models and the evaluation metrics, and how to ground them. We also had a discussion about lower- and upper-bound benchmarks. These foundations are necessary in order to have baselines to compare to, and performance aspects (i.e., metrics) use for this comparison.

In this chapter, we build on these foundations, by proposing the use of conditional temporal auto-encoder framework in order to study and extract styles, in the context of sequences (i.e., a time aspect exists in the data). We take advantage from the fact that the content is well known in our dataset (i.e., the identity of the letters in *IRONOFF*).

Questions addressed in this chapter

- What possible framework to study styles? and why?
- How does this framework performance compare to the benchmarks?
- What kind of styles we can extract from this framework? and how do we extract them?

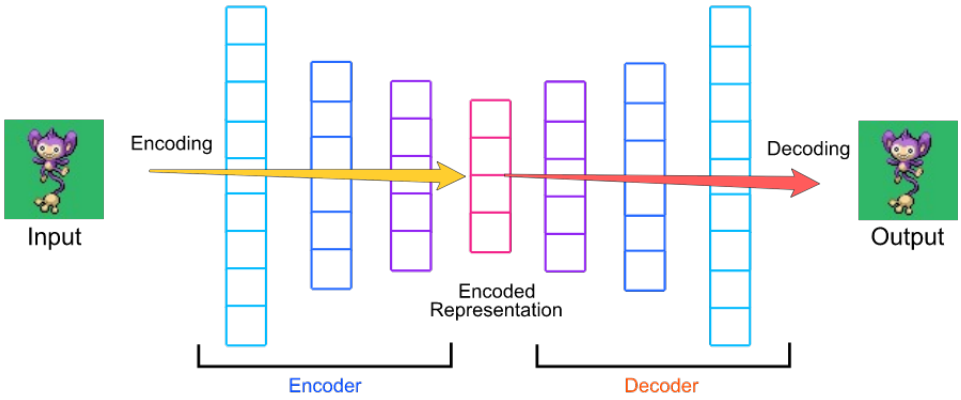


Figure 4.1: An example for the main components of an auto-encoder, used on image compression: an encoder takes the image, transfer it via a set of transformations into a bottleneck code, which is a compressed representation for that image. The decoder then takes this bottleneck code, and apply a series of transformations on it, in order to reconstruct the original input image. Source of this image is (Mohammed).

4.1 Background

4.1.1 What is an auto-encoder?

Auto-encoders (Hinton and Salakhutdinov, 2006) is identity-capturing framework, that allow the emergence of interesting behaviors. To expand on this, the objective of the auto-encoder is to capture/learn the distribution of the input data (identity-capturing). An auto-encoder consists mainly of three parts (see figure 4.1):

1. Encoder: it takes the input data, and project it into a manifold (the bottleneck).
2. Bottleneck code/learned representation: this is the representation learned by the encoder. In the basic form, this just the output of some linear/non-linear activations. However, a lot of the literature exists on how to organize and shape the bottleneck, in order to allow the emergence of interesting information about the data.
3. Decoder: it takes the learned representation, and reconstruct the original input from it.

There are many reasons for using auto-encoders, to name a few:

Dimensionality Reduction It is one of the main motivations to study auto-encoders. By mapping a high-dimensional data into a smaller low-dimension space, we can better explore the data, or use it as a step in a pipeline of machine learning/data analysis operations, where it provides a more manageable format of the original data, as in (Ha and Schmidhuber, 2018). It can also be used in classification system (Goodfellow et al., 2016).

Information Retrieval The kind of search used in information retrieval is an efficient in low-dimensional space. In (Salakhutdinov and Hinton, 2009), auto-encoder is trained to produce low-dimensional binary code, which can be then used in queries (by returning the entities that have the same binary code).

Anomaly Detection Multiple works have explored the use of auto-encoders in order to perform anomaly detection (An and Cho, 2015; Ribeiro et al., 2018; Sakurada and Yairi, 2014). The main hypothesis is that the auto-encoder will learn the most salient features in the data. Thus, when faced with an anomaly, a significant degradation in the quality of reconstruction will be noticed. The reconstruction error can be used in this case as an indicator if the input data point is an anomaly or not.

Image De-noising Multiple works have explored the use of auto-encoder in order to de-noise images (Cho, 2013a,b; Gondara, 2016). The applications of image de-noising are diverse, from post-processing of digital images, to more sensitive areas like medical images.

The idea of compressing data is not new. Techniques like *Principal Components Analysis* (PCA) (Jolliffe, 2011) or *Independent component analysis* (ICA) (Hyvärinen and Oja, 2000) do exist in order to project the data into smaller dimensions. But they are usually restricted by several assumptions. In case of PCA, it assumes linearity and orthogonality in the dimensions of variation in the data. Neural networks enables us to get around this issue, by leveraging nonlinearity and multiple layers, this giving us a more flexible approach to find dimensions of variation in the data.

4.1.2 Sequence auto-encoder

It a special case of auto-encoder, where RNN is used in order to compress a sequence into a fixed size bottleneck code. The sequence itself could have a varied size. The first architecture proposed for sequence auto-encoder was proposed in (Cho et al., 2014; Sutskever et al., 2014b), with statistical machine translation as the main application. They use RNN encoder and decoder parts, and consider that the last hidden state of the RNN encoder to be the summary/compression of the sequence. The RNN decoder uses this bottleneck in order to reconstruct the whole original sequence (see figure 4.2).

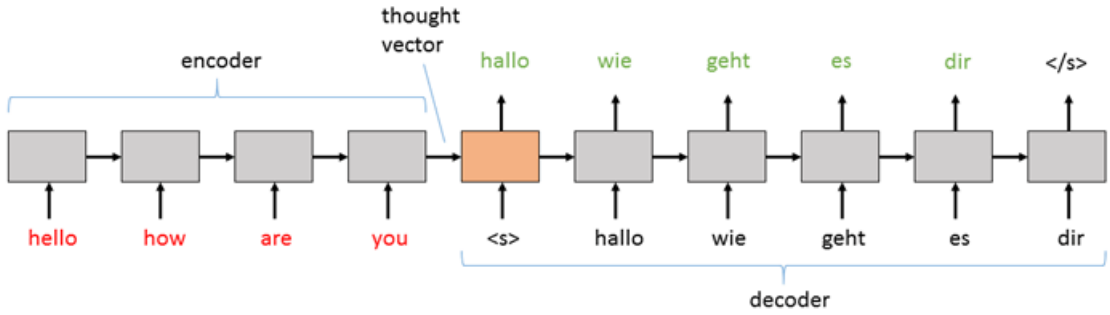


Figure 4.2: An illustration for a sequence-to-sequence architecture, used for language translation between English and German. The encoder summarize the English sentence, and the decoder use it as to bias its own output, to generate the equivalent German sentence.

There many applications where a sequence-to-sequence model is used, for example:

Machine translation (Sutskever et al., 2014b) used sequence-to-sequence architecture in order to develop a machine translation system that surpassed the published results to that moment. The first sequence is the source language, and the second sequence is the target language (that can also differ in the word count from the source language).

Speech synthesis Speech synthesis also benefited from a sequence-to-sequence architecture, like the work done in (Oord et al., 2016a; Wang et al., 2017b), achieving currently the state-of-the-art results. Another common approach for sequence-to-sequence learning is to use convolution network instead of RNN for the encoder and/or the decoder, like the work done in (Ping et al., 2017). A convolution network is generally faster than RNN and parallelize better, thus making it a lucrative approach. The underlying assumptions are the same however.

Video captioning Another application that benefited from sequence-to-sequence architecture is generating text (sequence of words/letters) that describe a video (sequence of frames), as nicely summarized in (Aafaq et al., 2018). In this case, the encoder – dealing with the video part – is usual a convolution neural network (because of its excellent ability to capture spatial features), while the encoder – dealing with the text part – is usually a RNN.

4.1.3 Conditioned auto-encoder

In the examples mentioned before, we focused on unconditional auto-encoder, where in the decoder only have the information given to it from the encoder part. A conditioned

auto-encoder is when we concatenate extra information to the output of the encoder, and feed it to the decoder[1]. Why conditioning an auto-encoder? It frees the encoder from learning the condition information – since this information is given for free – , allowing it to focus on other parts.

An example of this is the work done in (van den Oord et al., 2017), where they used an auto-encoder to compress audio, and condition the decoder on the speaker-id. This led the encoder to factor out speaker-specific information in the learned bottleneck, thus, learning speaker-independent information[2].

4.2 Putting it all together

In order to address the research questions stated at the beginning of the chapter, we chose to adopt the concept of conditioned auto-encoder as our framework to explore styles in handwriting, for the potential following benefits discussed in the previous section:

- Conditioning the decoder on the content identity of the task (i.e., the letter identity) will free the encoder from learning this information, thus allowing it to focus on learning the letter-independent style relevant information.
- The encoder will learn a bottleneck, that is a compressed information about the sequence style. This can allow us to explore this bottleneck via traditional techniques (PCA, tSNE, clustering, classification of the bottleneck...etc), thus, getting more insight about what the model actually learned

In this following subsections, I will present our contributions: the model architecture we used, and quantifying the quality of the generated letter using the evaluation metrics and benchmarks we discussed in the previous chapter. Then, we will briefly take a look at our first attempt to tackle transfer learning[3]. I then end with the style extraction part, where we explore what knowledge/information about the styles our model has extracted. The work done in this chapter was published in (Mohammed. et al., 2019).

[1]The word 'conditioned' is used when a neural network has information from about the task. So a decoder is a conditioned network on the encoder information. We distinguish here in the terminology between 'unconditioned auto-encoder', where the decoder is not conditioned on anything else except the encoder, and 'conditioned auto-encoder' where the decoder has access to extra information other than the encoder.

[2]Simple explanation and demonstration for that paper can be found in `https://avdnoord.github.io/homepage/vqvae/`

[3]More details on transfer learning in the next chapter

4.2.1 Model architecture

The model architecture is illustrated in figure 4.3. The input/output frames of the model are detailed in figure 4.4. The tracing of the letter is first fed to the encoder module. The final hidden state of that module summarizes the letter. In order to allow this module to focus on learning the style embedding, we complement this last hidden state with the one-hot encoding of the letter identity, and use an embedding of them as the bias input to the generator. The encoder thus is free from the need to learn the letter identity, and can focus learning the style information that enables the generator to better approximate the ground truth tracings.

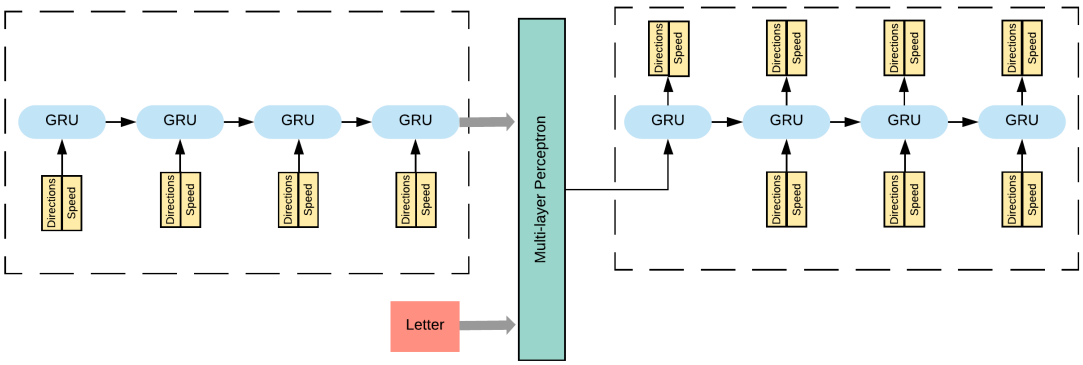
In the decoder, we follow the framework proposed by Vinyals et al. (2015) in order to bias the model – as in the previous chapter – : we create an extra time step at the beginning, which has the information we want to bias the model with. In this case, this time step (34-D) is the projection of the encoder last hidden state (128-D) and the letter encode (26-D). This has a much lower dimension than encoder hidden state. This further encourage the model to learn only the necessary style information, as suggested in Skerry-Ryan et al. (2018).

In order to allow for faster exploration of different hyper-parameters, we use an early stopping of 20 epochs (when no improvement in the validation score happens during these epochs). To summarize, the current model specifications:

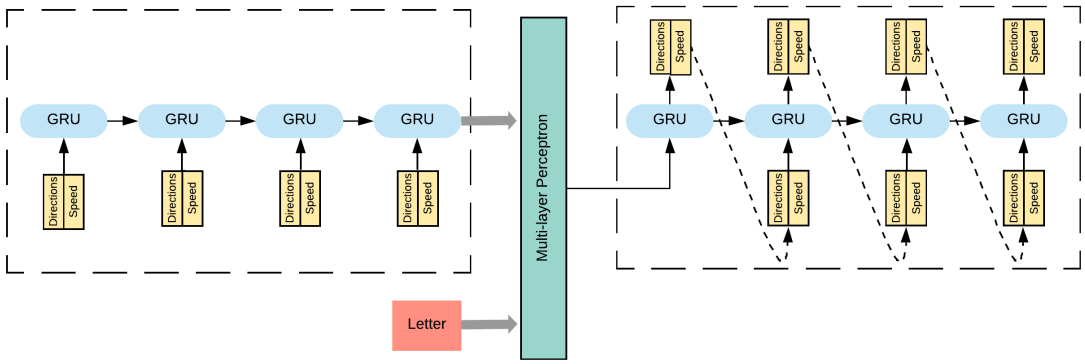
- Encoder hidden size: 128
- Decoder hidden size: 128
- Encoder layers: 2
- Decoder layers: 2
- Encoder dropout: 0.0
- Decoder dropout: 0.2
- Learning rate: 0.001

4.2.2 Letter generation with style preservation

The objective here to compare the quality of the generated letters to the state-of-the-art benchmarks. As mentioned earlier, we compare using the BLEU score metric and the EoS analysis. The BLEU score results can be seen in table 1, and the results for EoS analysis results are in table 3. We can see that the BLEU-3 score results of our model achieves 32.3% accuracy in Speed feature and 38.7% accuracy in Freeman feature,



1 Training mode, using the *teacher forcing* methodology (Goodfellow et al., 2016; Williams and Zipser, 1989).



2 Inference mode, using the *softmax temperature sampling* method.

Figure 4.3: Schematic diagram of the model we used. During the training time 4.31, the input to the model is always the ground truth. During the inference time 4.32 however, the input to the decoder (generator) part at each time step is its own predication in the previous time step.

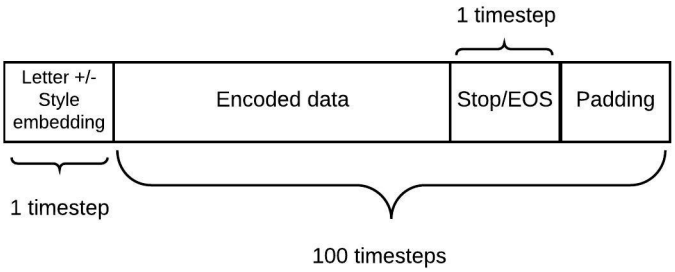


Figure 4.4: Input sequence to our model. The first time step contains the information necessary to condition/bias our model. In case of the encoder, this first time step (the bias) is not included.

compared to 25.1% and 28.3% accuracy using the benchmark model on both features respectively.

The same goes for the EoS analysis. In comparing the Person Coefficient, our model achieves 0.99 score compared to 0.55 for the benchmark model (the highest score is 1.0). This is a support that our model capture the style of handwriting better than the benchmark.

Examples for generated letters can be found in figure 4.14.

Aspect/Feature	Speed			Freeman		
Model / B-score	B-1	B-2	B-3	B-1	B-2	B-3
Letter + Writer bias	51.5	41.4	25.1	56.7	39.4	28.3
Style Extractor	**71**	**51.7**	**32.3**	**65.6**	**51.5**	**38.7**

Table 1: BLEU scores for different models for known writers.

Aspect/Feature	Speed			Freeman		
Model / B-score	B-1	B-2	B-3	B-1	B-2	B-3
Letter + Writer bias	55.4	39.6	25.3	50.2	38.6	27.7
Style Extractor	**72.4**	**52.4**	**32.2**	**70.4**	**55.6**	**42.1**

Table 2: BLEU scores for different models for style extraction for 30 new writers (style transfer).

Models	Pearson coefficient
Letter + Writer bias	0.55
Style Extractor	**0.99**

Table 3: Pearson correlation coefficients for the End-of-Sequence (EoS) distributions for the conditioned-autoencoder framework (style extractor) compared to the baseline (with letter and writer information only), on the generated letters compared to the ground truth. We can see that the (style extractor) outperforms the baseline.

Models	Pearson coefficient
Letter + Writer bias	0.5
Style Extractor	**0.99**

Table 4: Pearson correlation coefficients for the End-Of-Sequence (EoS) distributions for the different models on 30 new writers (style transfer). Even though the baseline model is given explicit information about the writer, the style extractor still outperforms the baseline. This could be an indication that the there is a limited number of styles after-all.

4.2.3 Style transfer

One of the hypotheses we want to test is whether there is a limited number of styles needed, to generalize over new writers. To achieve this, the learned representation for styles should extract generic information about the styles.

In order to test this hypothesis, we expose our model to 30 writers that have not been seen before. We compare our model performance on these writers with a model biased by the writer and letter identities (the benchmark model). The latter model was not constrained from seeing those writers (thus, the reported results of the comparison overestimates the actual performance of that model).

The BLEU scores can be seen in table 2. Our model achieves on BLEU-3 score 32.2% and 42.1% accuracy on the Speed and Freeman code features, compared to 25.3% and 27.7% on the benchmark model for the same features respectively.

The EoS analysis can be seen in table 4. Our model achieves a Pearson coefficient value of 0.99, compared to 0.5 for the benchmark. Thus, the new model clearly outperform the current benchmarks on the transfer task, on both BLEU score and EoS analysis.

4.2.4 Styles per letter

One of the nice consequences of using our model is that we can have a better look at the styles. We explore the latent space for multiple letters, and see that we can uncover interesting writing styles. A full scale analysis is beyond the scope of this thesis. We project the latent space using *Principal Components Analysis* (PCA) (Jolliffe, 2011) and t-SNE (van der Maaten and Hinton, 2008).

As a start, we take a look at letter X. Beforehand, we identified a style feature in letter X: some writer draw X clockwise, and some draw it anti-clockwise[4]. We manually annotated the whole dataset for this feature; the result can be seen in figure 4.5. Almost half of the writers draw the letter X clockwise, and the other half draw it anti-clockwise. If our assumption is correct, our model should be able to capture this feature. We project the latent of the model using PCA on all the letter X, which can be seen in figure 4.6. The model latent space clusters almost perfectly match those based on rotation. Examples for letters from both clusters are in figure 4.7.

Encouraged by the results on letter X, we explored more letters. For letter C, we can see the latent space project in figure 4.8. It can be seen that there are at least two main clusters. Examples from this cluster in the red ellipse are in figure 4.10. This clearly represents the Edwardian handwriting style. The rest of the writers (in the big cluster) have a very similar style (this is expected, since the drawing of the letter C is quite simple).

For letter A, our model latent space create two main clusters, figure 4.9. We give examples from those two in figure 4.11, where we can see clear difference in the style. Some people start drawing the letter from down-left, other writers start from the top of letter A, move down, then continue drawing of the letter.

Another example is for letter S bottleneck, figure 4.12. There are three resulting clusters which we investigated. The indicated cluster (in red) is clearly different from the other two clusters (not indicated). Examples can be seen in figure 4.13. The indicated cluster is again for people with Edwardian handwriting style. We did not find a clear difference between the other two clusters though, but this is an expected outcome of using t-SNE (since it does not have the clear objective of clustering styles).

These examples show is that we can use our model to extract verbose style information.

[4]We did not find a connection between this point and the handiness of the participants.

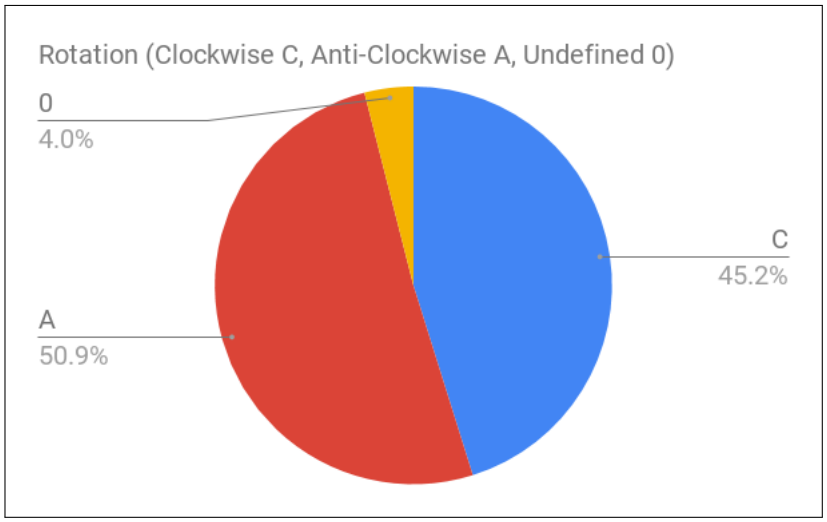


Figure 4.5: Results of the manual annotation for the rotation of letter X drawings over the whole dataset. Almost half the writers drew X clockwise, the other half anti-clockwise. The undefined styles were unclear to determine.

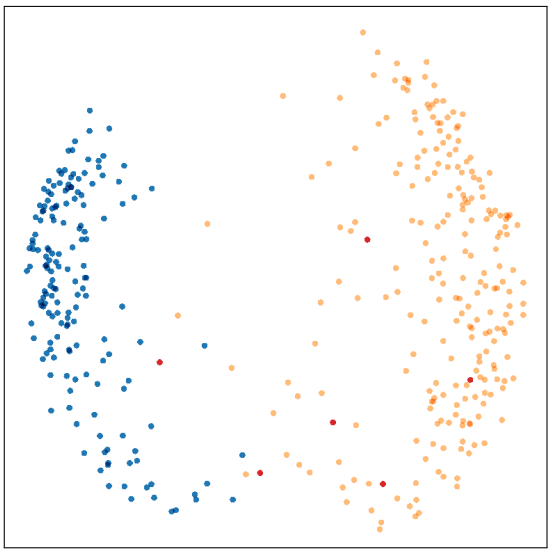


Figure 4.6: Projection for latent space for letter X using PCA. The colors show the ground truth of the X rotation: blue is counter clockwise, orange is clockwise, and the few red points are undefined.

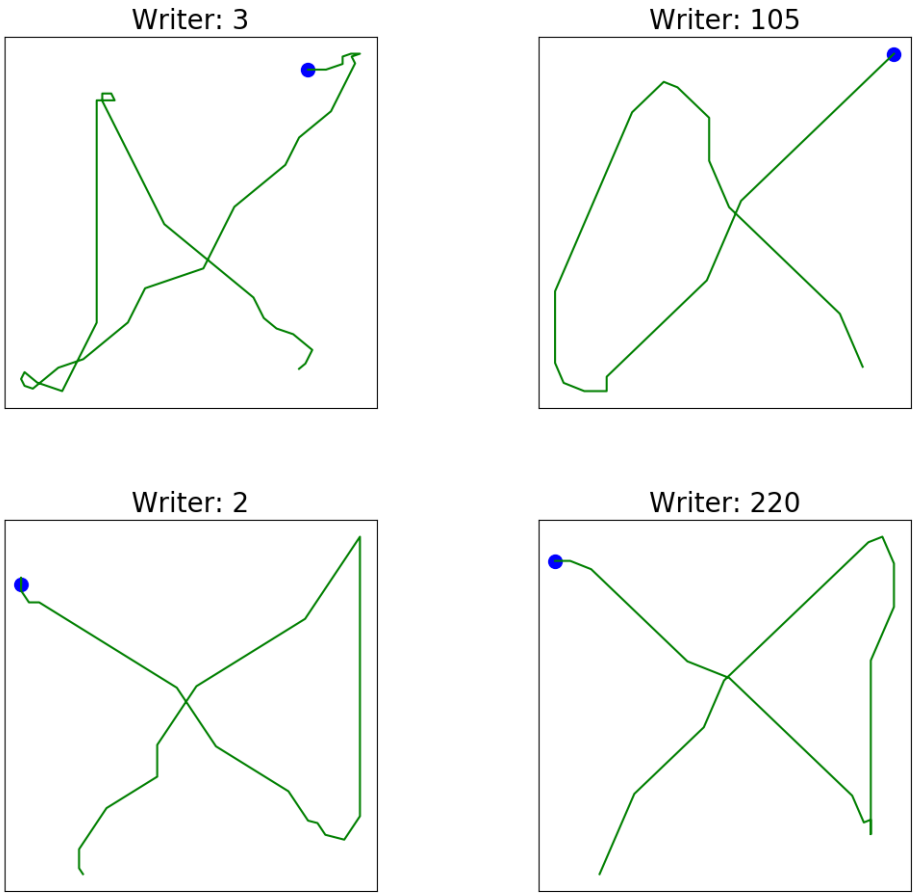


Figure 4.7: Examples for writing of letter X. Starting point is marked with the blue mark. Each raw is randomly sampled from each cluster in the bottleneck. The clusters shows that almost half the writers draw the letter clockwise (first row, first cluster), and the other half draw it anti-clockwise (second row, second cluster).

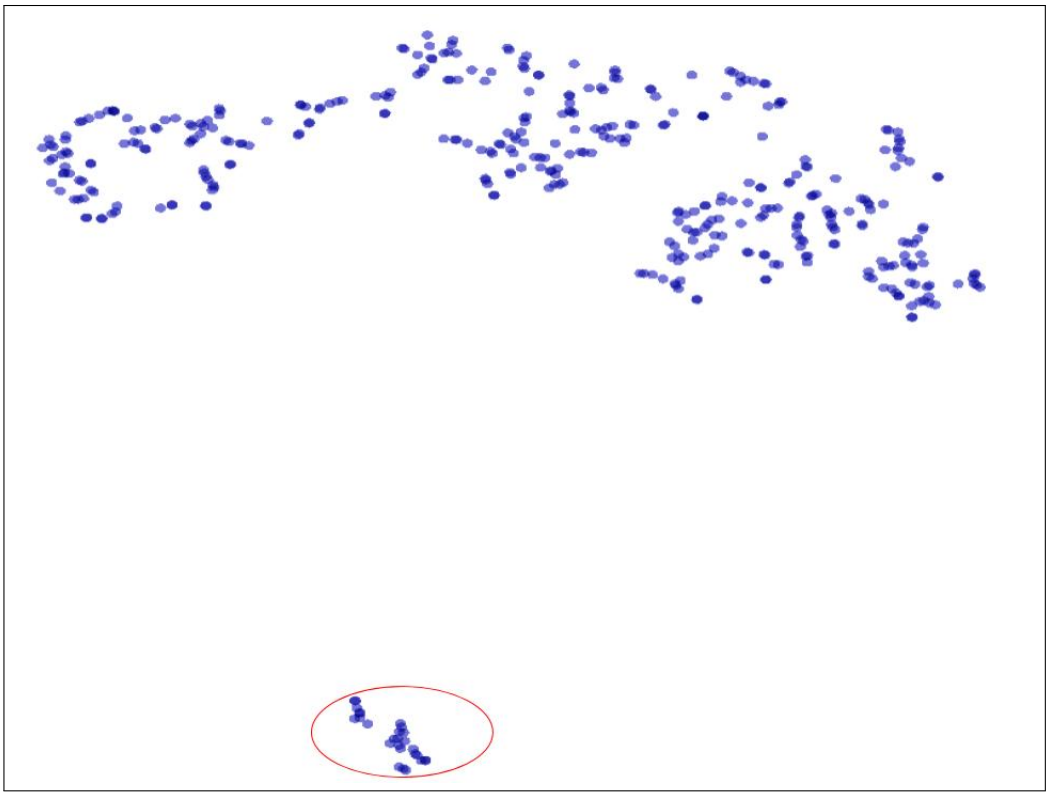


Figure 4.8: Projection for latent space for letter C using t-SNE. The cluster surrounded by the red circle has a clear interpretation, where writers have a cursive style.

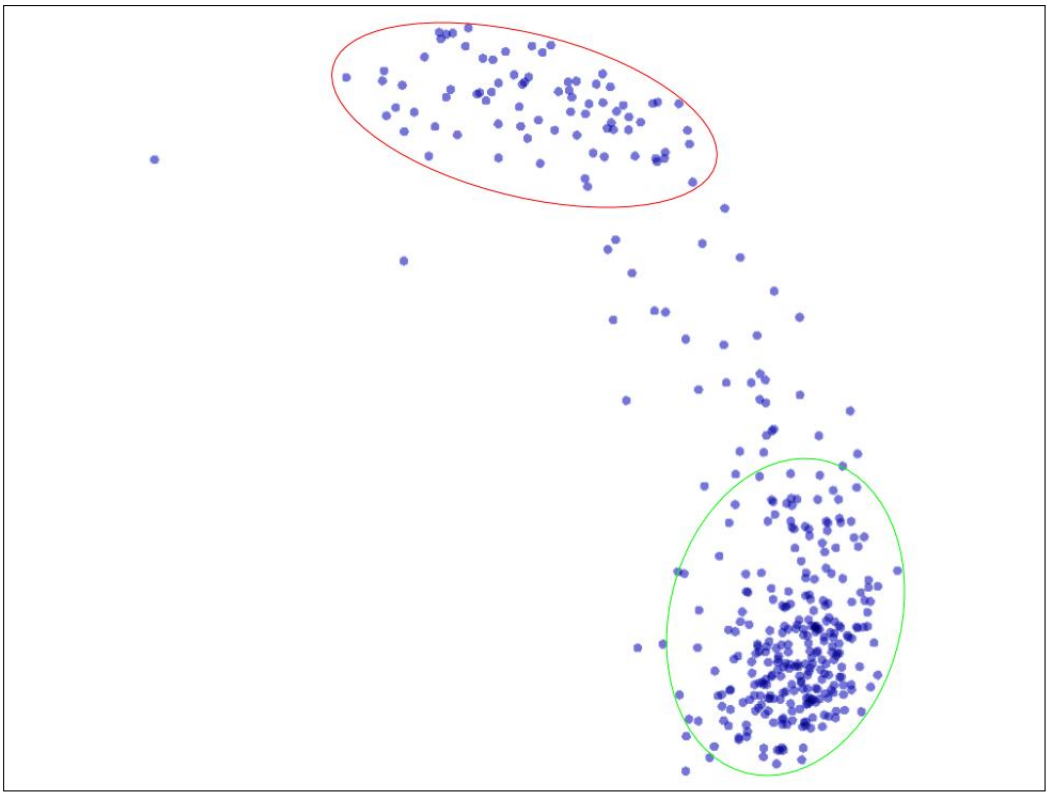


Figure 4.9: Projection for latent space for letter A using PCA.

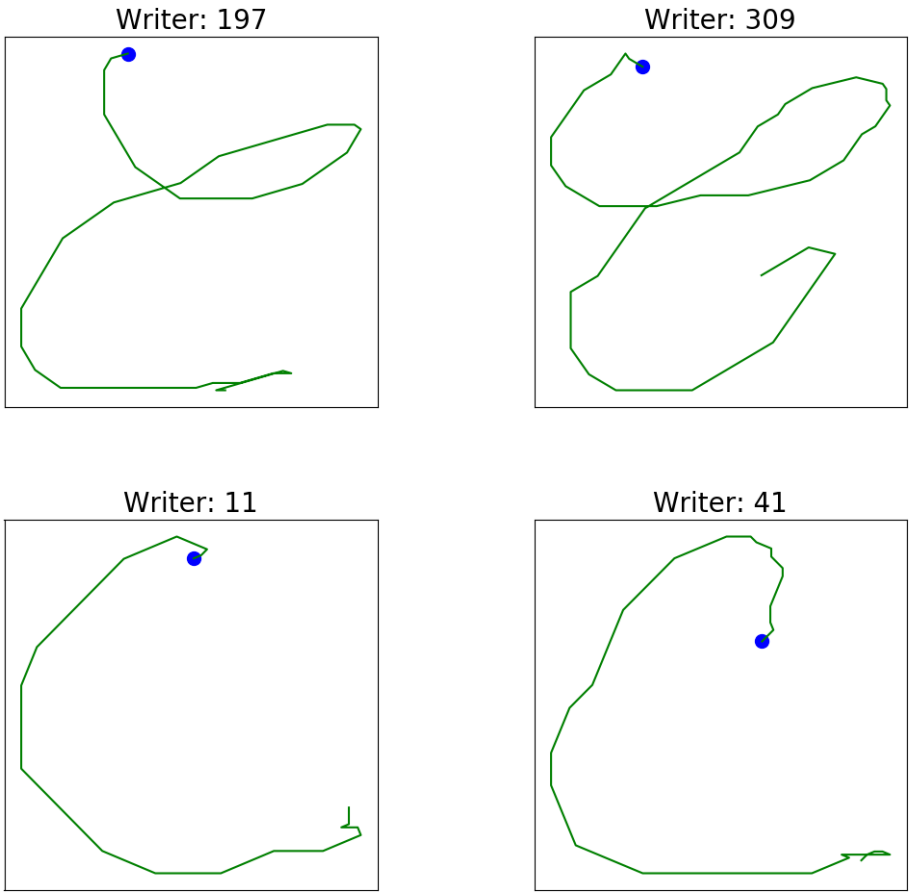


Figure 4.10: Examples for writing of letter C from the selected cluster (first row) versus the rest of the letter drawings (second row). Starting point is marked with the blue mark. The drawings from the selected cluster show people with Edwardian style of handwriting.

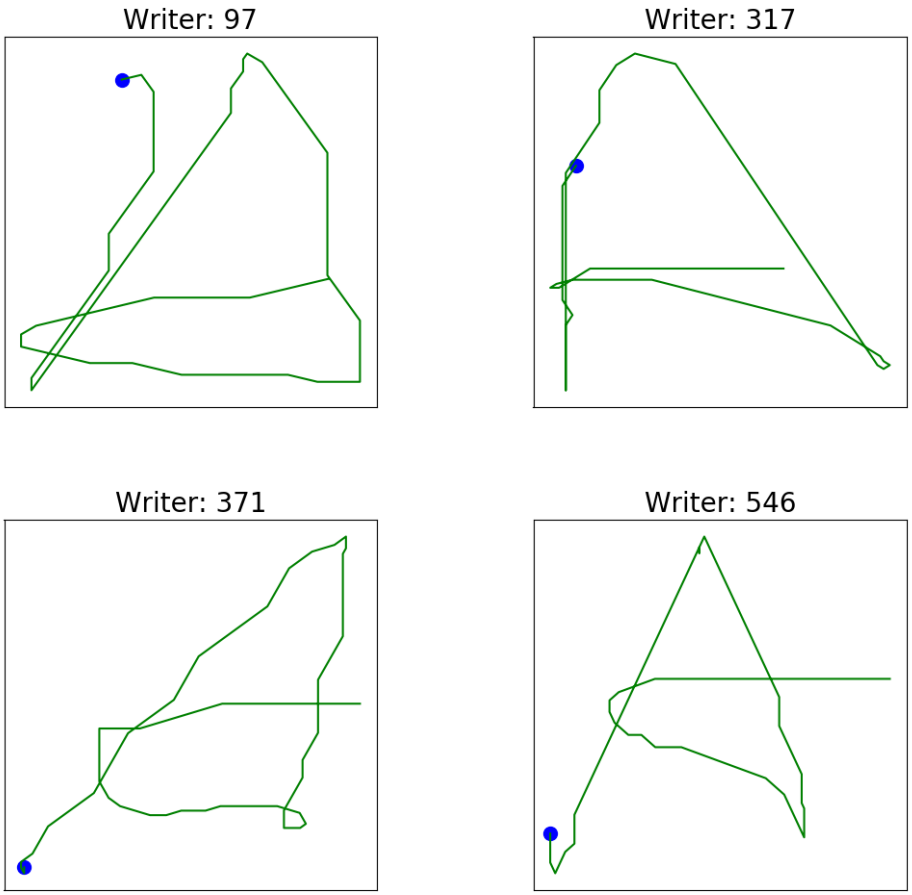


Figure 4.11: Examples for writing of letter A from the selected clusters. Starting point is marked with the blue mark. Each row is from one cluster. The first row show people who start drawing the letter from the top, going down, and then continue the drawing of the letter. The second row show people who start drawing from down directly.

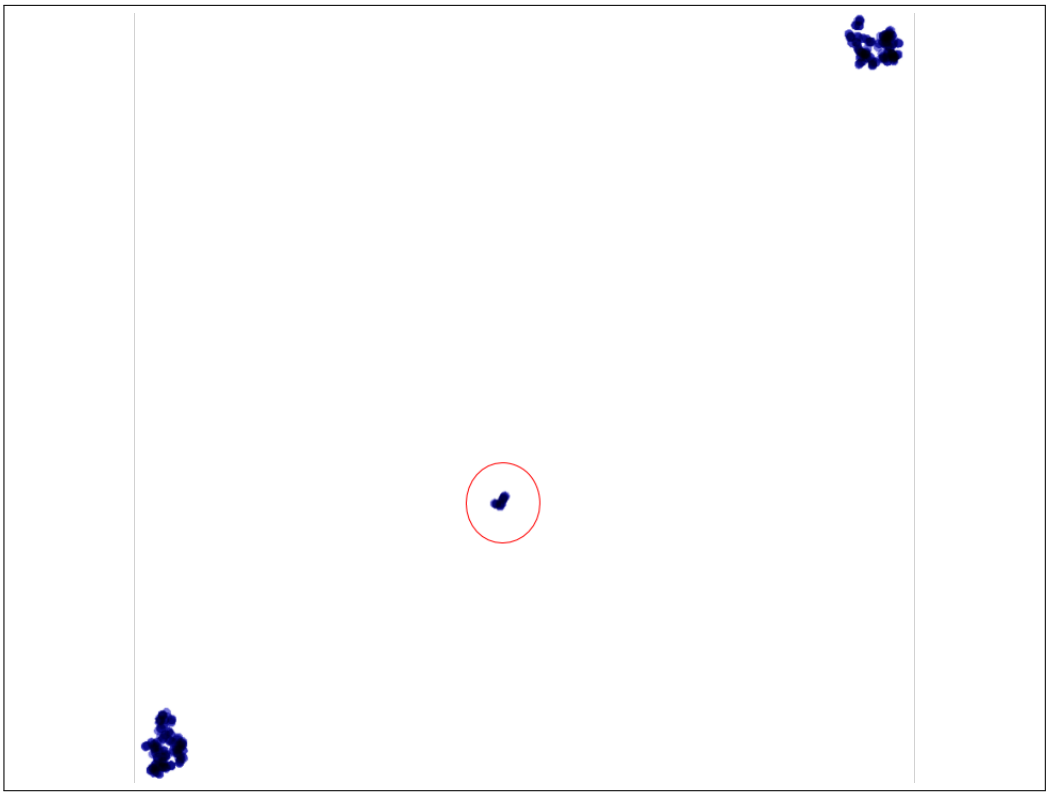


Figure 4.12: Projection for latent space for letter S using t-SNE. We manage to interpret the indicated cluster as the Edwardian style in drawing. The other two clusters (not indicated) did not show clear difference in the style, but this is an expected behavior from using the t-SNE algorithm, since it does not try to cluster styles as an objective.

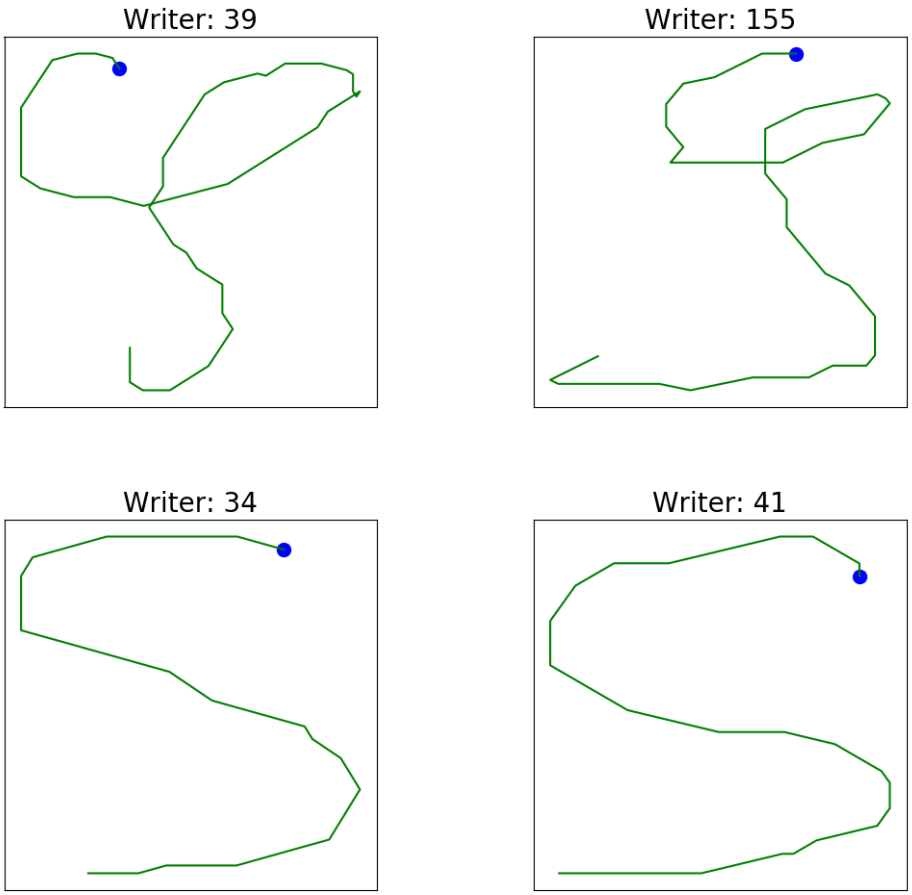


Figure 4.13: Examples for writing of letter S from the selected cluster (first row) versus the other two clusters (second row). Starting point is marked with the blue mark. The drawings from the selected cluster is always Edwardian style.

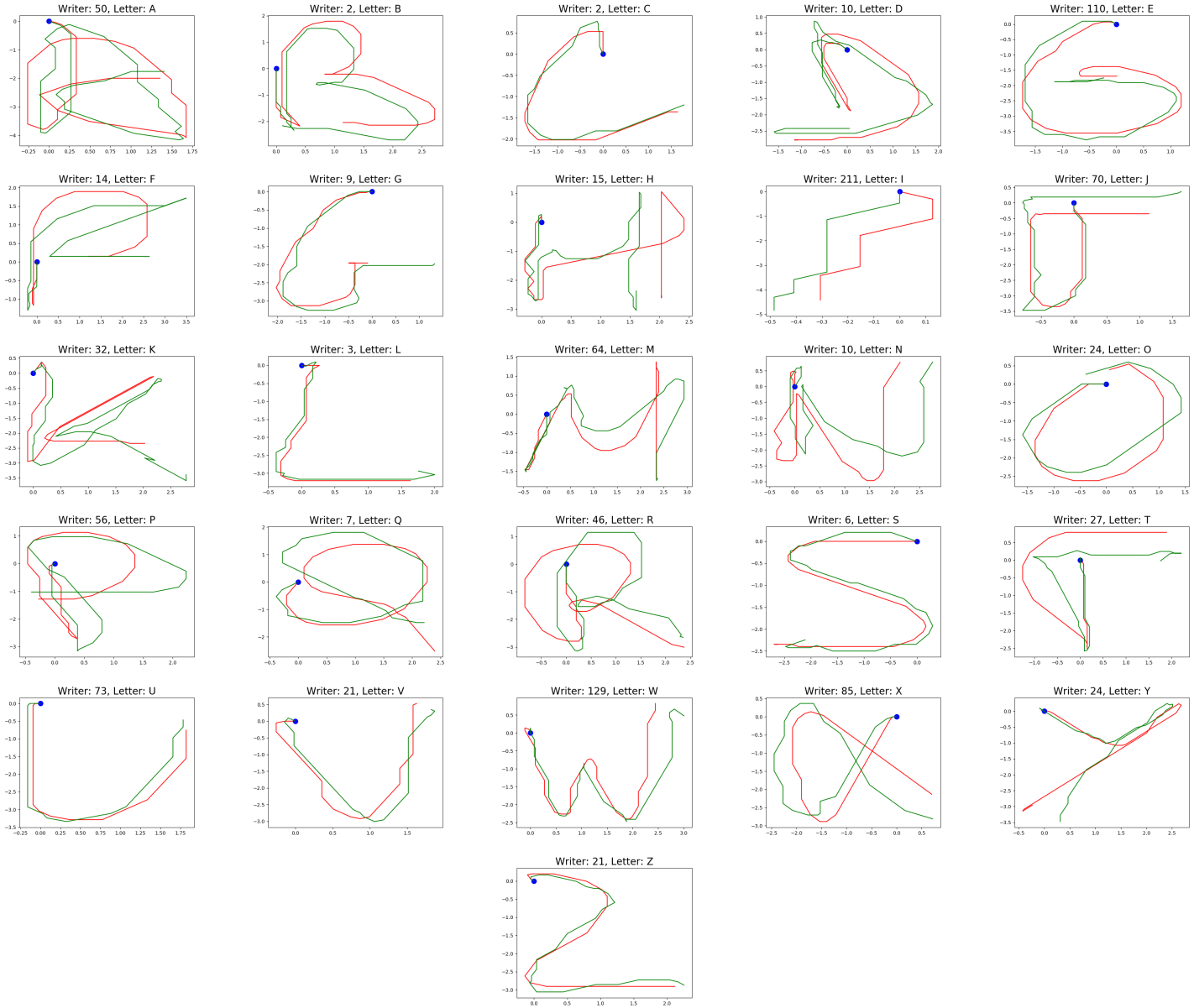


Figure 4.14: Examples of generated letters. The blue mark is the starting point. The traces in green is the ground truth, and the red is the generated ones by our model.

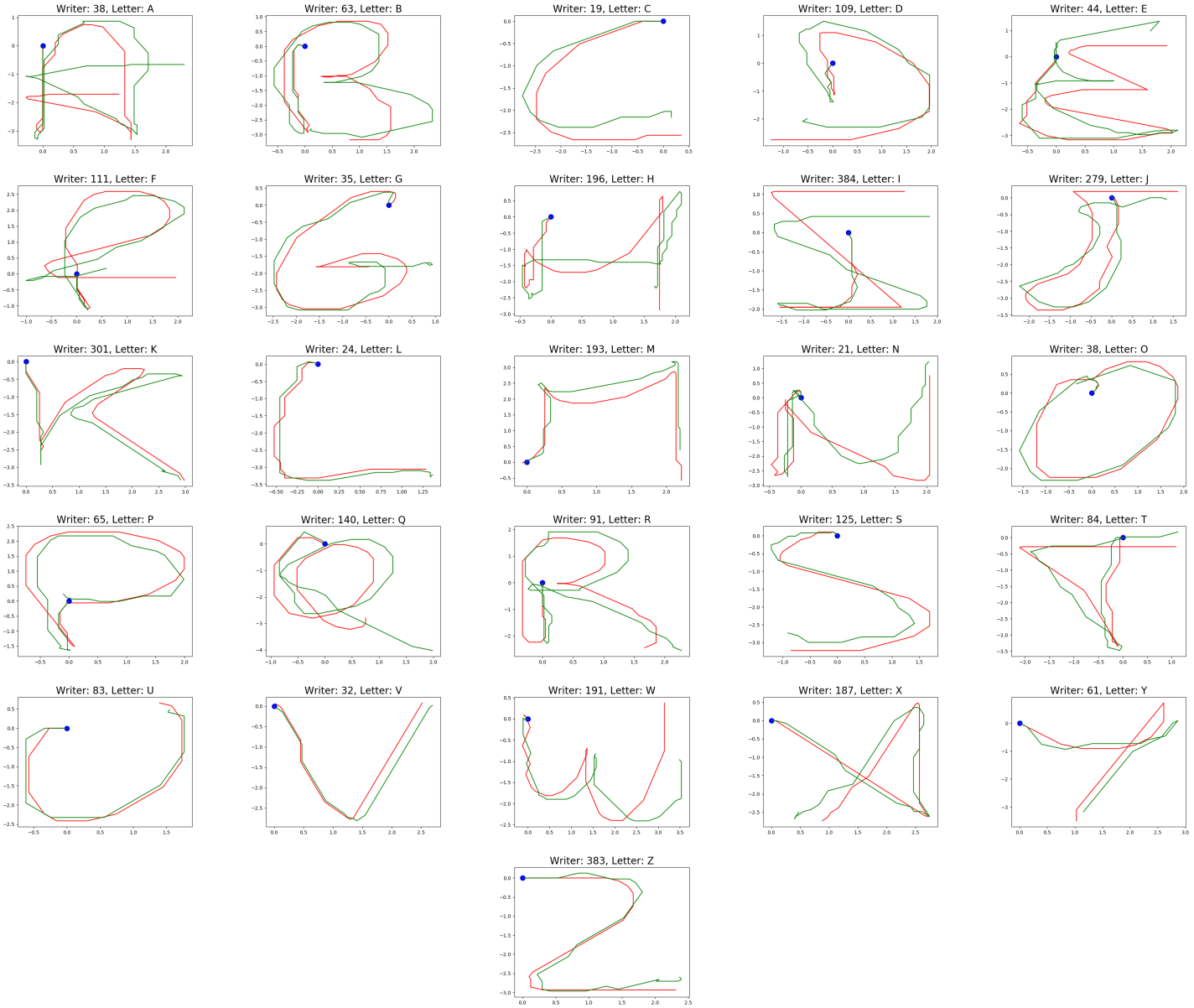


Figure 4.15: Examples of generated letters. The blue mark is the starting point. The traces in green is the ground truth, and the red is the generated ones by our model.

4.3 Summary

In this chapter, we discussed the way we chose in order to study styles, by using the auto-encoder framework. We hypothesized that we can study styles implicitly by looking at how they contribute to the reconstruction/generation of the letters. We looked at the state-of-the-art concerning auto-encoders, the sequence-to-sequence case, and why we may choose to condition an auto-encoder.

I then presented our work, addressing the following points:

- What possible framework to study styles? And how to validate it? We test the auto-encoder framework relative the benchmarks we proposed in the previous chapter, in order to determine if it is actually a valid framework to study styles. We find that an auto-encoder outperforms the benchmarks in all evaluation metrics, suggesting that this is a good approach.
- What kind of styles we can extract from this framework? And how do we extract them? Given the good performance results of the auto-encoder, we wanted to have further confirmation that our model is actual learning relevant style information from one side, and if we can learn something new about styles. We explored the model bottleneck for different letters, showing a strong evidence that the model is quite useful in the study of styles.
- Can we transfer the styles over different writers? We hypothesis that there is a limited number of styles in general – if we have enough writers in our training data, we will generalize well to new writers –. We test this hypothesis by hiding a number of writers from the training data, and compare a model trained on the other writers (transfer model) versus a model trained only on those writers (baseline model). We see clearly the transfer model outperforms the baseline, thus giving good evidence for our hypothesis.

We thus are ready for the next chapter, where we dive more into the world of transfer learning...

Chapter 5

Style Extraction and Transfer

Contents

Missing points

- Add a diagram explaining transfer Learning
- Add a diagram explaining which part of the model we are transferring, and which one we are retraining (maybe a series of diagrams explaining visually all the steps)
- Add examples of the generated letters/shapes

We finally arrive to the core objective of my thesis: how to leverage information of style from one (or more) task/s, in order to bootstrap the learning of a new task?

Questions addressed in this chapter

- What is transfer learning? and what are the different approaches to perform it? I will explain the different paradigms and metrics used to characterise

transfer learning.

- How do we approach the problem of style transfer, for both handwriting and sketch drawing?
- The experiments performed, the results, and our conclusions.

5.1 Transfer learning

An important research direction in machine learning nowadays is transfer learning. If humans and machines are able to learn how to perform a task, one of the thing that separates humans from machines is the ability to leverage this knowledge in order to acquire new skills and perform new tasks, without the need for additional trials and errors from tabula rasa. This however, is not a straightforward thing for machine learning to do. The algorithms are fitted to data responding directly to the task required (i.e., has the same input feature space and same distribution). Thus, a change in the task can lead to degradation in the algorithm performance (Pan and Yang, 2009; Shimodaira, 2000; Tan et al., 2018; Weiss et al., 2016).

Let's first introduce some notations that will help in formulating the problem:

- We first introduce the concept of *Domain*. A domain defines a feature space (e.g., images of animals), and the probability distribution of this space (i.e., the distribution of pixels in the images of animals). We can consider the domain as the available *knowledge* to us. Thus, a domain $\mathcal{D}$ is defined as $\mathcal{D} = \{\mathcal{X}, P(X)\}$, where:
  $\mathcal{X}$ is the feature space, X is the data samples available to us from the feature samples, $X = \{x_1, x_2, \cdots, x_n\} \in \mathcal{X}$, where n is the size of the learning sample. $P(X)$ is the marginal distribution probability of this data sample.

- For a given domain (aka, the knowledge available to us), we define the concept of a *task*. A task is something we would like to achieve using the knowledge we have. For example, a task can be classifying animals given animal pictures, or perform robot navigation given a map of the building. Thus, a task $\mathcal{T}$ is defined as $\mathcal{T} = \{\mathcal{Y}, f(.)\}$, where:
  $\mathcal{Y}$ is the label space (the task objectives), $f(.)$ is the mapping function (mapping the domain knowledge to the task objectives). It can also be rewritten as a conditional probability over the domain knowledge, $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$.

- Based on this notation, we define two more concepts: *Source* and *Target*. A source defines a domain and a task/s that are available to us already (where we have plenty of domain knowledge, and examples on the task/s). A target defines a domain and a task/s as well, where we usually do not have enough domain knowledge and/or examples on the task/s.

Now that we clarified some basic terminology, we can move on to define transfer learning: given source domain data D_S, source task $\mathcal{T}_S$, target domain D_T and target task $\mathcal{T}_T$, we wish to improve the performance of the target task $f_T(.)$ by using D_S and $\mathcal{T}_S$.

Given this definition, we can categorize different types of problems that transfer learning covers:

- The source and target domains are different, $D_S \neq D_T$, which means that the feature space is different, $\mathcal{X}_S \neq \mathcal{X}_T$, and/or the probability distribution of the feature space are not the same, $P(X_S) \neq P(X_T)$. If $\mathcal{X}_S \neq \mathcal{X}_T$, the transfer learning problem is *Heterogeneous*. Otherwise, it is *Homogeneous*.
- The source and target tasks are different, $\mathcal{T}_S \neq \mathcal{T}_T$, which means that the objectives are different, $\mathcal{Y}_S \neq \mathcal{Y}_T$, and/or the mapping function (from the feature space to the objectives) are different, $P(Y_S|X_S) \neq P(Y_T|X_T)$.

Many approaches in order to achieve transfer learning has been proposed in the literature. We will discuss the different approaches, and give examples from the literature on each one. We follow the categorization of transfer learning approaches done in (Tan et al., 2018)[1], by first identifying four main categories of transfer learning:

- *Instances-based*: in this case, we utilize examples from the source domain into the training of the new target domain, by defining weights on them.
- *Mapping-based*: the objective in this case is to project the instances from the two domains into a new manifold, that increases the similarity between the two domains.
- *Network-based*: the more common type of deep transfer learning. It is based on the idea that the layers of the deep neural network extracts basic and general information, that shared a lot with other domains. In this case, the network or some of its layers are re-used on the target task.
- *Adversarial-based*: similar objective to *Mapping-based*, by using generative adversarial networks (Goodfellow et al., 2014) in order to find a manifold that are fit for both source and the target domains.

In the context of *deep transfer leaning* – the main domain in our work –, *Network-based* and *Adversarial-based* transfer are the relevant categories in this case. *Adversarial-based* transfer is quite recent, and there is not much to talk about at the moment, so we will focus *Network-based* transfer, as it is the most common type of deep transfer learning.

When it comes to evaluating the success of the transfer, there is no one way to evaluate transfer learning in general. This depends a lot on the objectives of transfer

[1] Other categorization exists, like the one used in (Weiss et al., 2016). When it comes to deep transfer learning, we believe the categorization in (Tan et al., 2018) to be the most relevant.

learning, and the criteria of success. In the case of machine learning, the improve in end quality of the model is the primary performance aspect being measured and reported – like the classification accuracy (Chattopadhyay et al., 2012; Glorot et al., 2011; Long et al., 2013; Pan et al., 2010a), the reduction in the average error (Pan et al., 2010b)...,etc –. The transfer is considered successful if it achieves better performance than the baseline method.

We expect that, with the introduction of transfer learning via deep learning, that the *time to train* the model could be another aspect to consider – similar to what is being used in reinforcement learning (Taylor and Stone, 2007) –, although – to the best of our knowledge – we do not find studies mentioning this at the moment.

The idea of using deep learning (LeCun et al., 2015) in order to achieve transfer learning has gain popularity during the last years, following the achievements in having better computational resources (Raina et al., 2009), and the availability of large benchmark datasets - most notably: ImageNet (Deng et al., 2009) for object detection, MS-COCO (Lin et al., 2014) for image captioning ....

The first notable success of deep learning happened in the area of computer vision, with the AlexNet architecture (Krizhevsky et al., 2012). It was found out that such a deep network manages to extract generic features about the images: it learns simple, hierarchical filters, that are generic enough to be applicable for different datasets (see figure 5.1). The filters can be also seen as a representation of knowledge learned on the given tasks, with the first layers learning primitive filters (like edge detection for example), and the subsequent layers learning more complex filters. This observation led to another surge in the usage of pre-trained AlexNet – and later newer architectures, like VGG16 (Simonyan and Zisserman, 2014), Inception (Szegedy et al., 2015)...,etc – as feature extractors for new, unseen datasets (i.e, transfer learning using these deep learning architectures).

Examples on transfer learning in case of network-based transfer are:

- *Sentiment Classification*: (Glorot et al., 2011) discusses a deep learning approach for transfer learning for sentiment classification, by using stacked de-noising auto-encoders (Vincent et al., 2008) to correct the marginal distribution between the source and the target domain, by learning latent variables/features common between the two data sources in two steps: first, train an auto-encoder on the unlabeled data from the source and the target. This will produce latent variables that will make $P(X_S)$ closer to $P(X_T)$. Then, use those latent features to train a classifier on the labeled source data.

  Experiments are done on 12 different sources and target domain pairs. The data used reviews for different products (4 different products). The performance metric used in this case is the classification error rate when using a classifier trained on the source task only, minus the classification error rate of a classifier trained

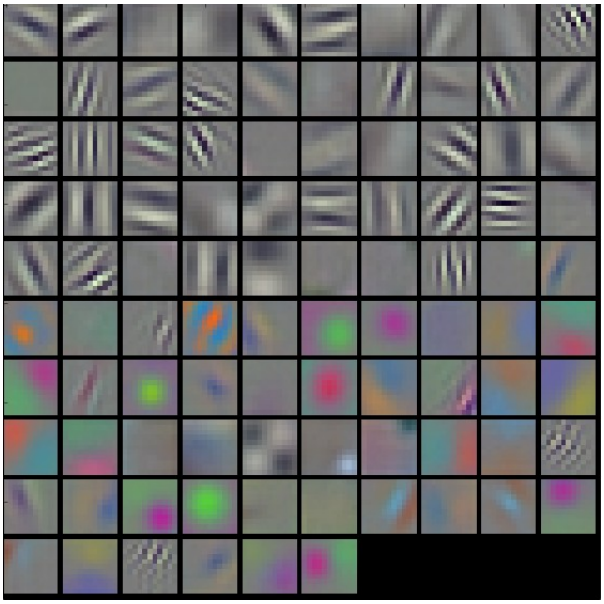


Figure 5.1: Visualization of the first convolution layer of a trained AlexNet. Note the basic shape of filters that resemble to Gabor filters widely used in image processing for decades (Fogel and Sagi, 1989; Jain and Farrokhnia, 1991). It is possible to see that those same filters will be useful in other computer vision or image-related tasks.

on the target task only. A SVM classifier trained on the source domain is used as a baseline, and a comparison with other transfer methods (Blitzer et al., 2006; Li and Zong, 2008; Pan et al., 2010a). All these methods performs better than the baseline, and (Glorot et al., 2011) peforms better than all of them.

- *Underwater Acoustics*: (Malfante et al., 2018b) compares the use of deep transfer learning to manual features (Malfante et al., 2016, 2018a), in order to recognize the sounds of fish underwater. Interestingly, the deep learning models are trained on ImageNet (Deng et al., 2009) dataset (which is completely unrelated to acoustics), then used in order to extract features from the spectrogram of the underwater recordings. The assumption here is the the filters learned with deep learning are basic and generic enough, to be used in other domains, thus, deep learning can provide a natural correction for $P(X_S)$, to make it close to $P(X_T)$. Without any fine-tuning, the deep learning achieves quite a high performance, although less than the state of the art, suggesting that a further investment in this direction is worth the time.

- *Speech to speech translation*: (Xu et al., 2014) studies the idea of transferring the speech knowledge learned from one language to another using deep neural networks. In particular, they study the transfer between Mandarin and English (in both ways). Their hypothesis is that when modeling the speech in a language,

there is a part in the model that is language independent. Their work was to find this part, and use it to bootstrap the learning of a new language (in case of neural networks, this can be rephrased as trying to find the good part in the neural network that is language independent).

They compared their proposed transfer approach to a baseline model (a deep learning model trained on the target task only), and compare the different the usage of different layers of a deep neural network trained on the source task (in order to determine the best part, or the language independent part). They report multiple performance metrics: segmental SNR (SSNR), log-spectral distortion (LSD), and perceptual evaluation of speech quality (PESQ). They find that, given insufficient data on the target task (which is one of the motivations to perform transfer learning), this transfer scheme works better than the baseline.

- *Image classification*: (Oquab et al., 2014) investigated the usage of a convolution neural network (CNN) trained on ImageNet (Deng et al., 2009) (where the object of interest is centered in the image), to extract low-level and mid-level features, that can transfer well to more complex dataset, PASCAL VOC (Everingham et al., 2010) (where there are several objects of interest in the image), and is smaller than ImageNet. They use the *average precision* as their performance metric, and they show that the transfer is better than training a model from scratch on this target dataset.

- *Styles of speech synthesis*: (Wang et al., 2018) discusses the problem of transferring the styles between different speakers, in the task of speech synthesis. Traditionally, the oucome of speech synthesis systems is the same style. One of the challenges is to capture the richness of style of different speakers on some training data, and transfer this style to new text. In their work, they propose an embedding approach called *Global Style Tokens* (GST), in order to extract the styles from the different speakers during the training phase. They then show that they can use these tokens as extra information to the speech synthesis system, to bias/affect the style of the outcome. They compare this to a baseline model, called *TACOTRON* (Wang et al., 2017b), without these GST addition. They use *Mean Opinion Score* (MOS) performance metric – a subjective metric to assess the quality of experience experienced –. They conclude that transfer learning using GST outperforms the baseline model.

5.2 Putting it all together

In the previous section, we explored the concept of transfer learning, with focus on network-based transfer, with multiple examples from the literature on this type of transfer. In this section, I will explain the work done during my thesis on transfer learning.

The main challenge we had is how to capture and transfer the styles between different tasks. This proposes multiple questions:

- Which framework/methodology of transfer we should use?
- How to assess the quality of transfer?

Our contributions in this part is that we study these questions on both *IRONOFF* and *QuickDraw!*. We proposed an experimental protocol in order to perform a rigorous evaluation for the usage of transfer learning – partially on the basis of the work done in (Lathuiliére et al., 2019) –. We use a slightly different experimental protocol between the two datasets however, depending on the computational power available. I will discuss the reasoning behind this, and will argue that both setups are good enough to address our questions concerning the transfer of styles. The work done in this chapter is in the publication phase.

What are we transferring? A typical approach to perform transfer learning in deep learning models is to:

1. Train the model on the source task,
2. freeze some parts of that model,
3. when going to the target task, use the frozen parts in a new model, and train the rest of that new model on the target task

The part that we freeze/transfer in our case is illustrated in figure 5.2.

How to evaluate the quality of transfer? Concerning the performance metrics, we decided to use multiple metrics to determine the effect of transfer learning, and compare it to the baseline (no transfer). While the usage of a single metric offers more convenience for decision making (i.e., which model to choose), our goal is have a better understanding for the transfer of style. To achieve this, we would like to shade light on the outcome from different angles, by using multiple metrics. We first evaluate the quality of the model in prediction by looking at the log-likelihood on the test data[2]. We then evaluate the quality of generation using our previously proposed metrics (the BLEU score and EOS analysis). We compare perform the generated strokes count, and include it in the comparison.

[2]Based on experience, the cross-entropy matters when it comes to generation. A difference of around 0.1 in cross-entropy between two models gives different generation quality.

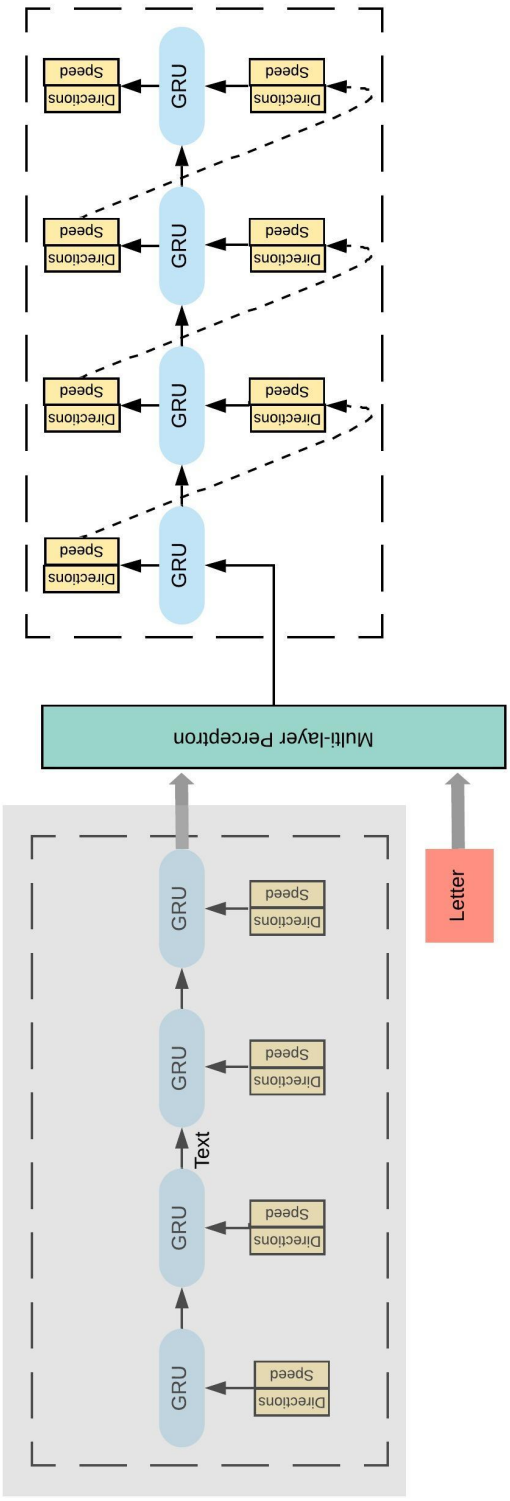


Figure 5.2: An illustration for model we use. The part identified by the gray area – the style extraction module – is what we transfer. Since we give the model the task content/identity, the gray part is expected to focus more on the style extraction. During the exposure to the source task, all the different components of the model are being trained. During the exposure to the target task, the gray area is the trained one on the source task, with frozen parameters. The other parts will be trained.

One of the things we changed from the previous work is that, when analyzing *end of sequence*, instead of using Pearson Coefficient, we use Krippendorff-Alpha (Krippendorff, 2011). The reason for this is that it offers a smoother transition between what is good and bad. In Pearson coefficient, if there is a mismatch between the generation and ground truth, it will not matter how big is the mistake (the distance between the generated and the ground truth). With Krippendorff-alpha, we can take into account the distance between the generated and the target symbols, thus providing a softer and more realistic estimation for the correlation.

Another aspect we consider in our analysis is the number of *strokes* the model is generating. In the previous work, we did not model the strokes, in order to simplify the system and the analysis. Now that we are confident about our approach, we added the strokes to our model. Strokes are more complex to model, because the stroke signals are more sparse. A good stroke generation is an indication that the model can perform hard discrete decisions in order to generate the whole shape. We use Krippendorff coefficients as well to report and analyze the strokes. We also consider the confusion matrices for the generated strokes versus the ground truth ones, in order to shade more light on the behavior of the model.

One important choice to make is to decide the amount of data to be used in the target task. Normally, the whole point from transfer learning is to address the insufficient data available for the target task. However, since we do not have a particular scenario to solve (we are mainly interested in that 'what if this happens' question), there is no clear criteria to decide the amount of data. If we select very few points, we may be biased towards giving transfer learning an advantage, while also sacrificing having generally bad results (even if transfer learning proved to be better than baseline on our performance metrics). It is important that the overall behavior of the system is acceptable (generating shapes with acceptable quality). To get around this design problem, we decided to use the entire data available to the target task, thus, testing the 'worst case scenario' for transfer learning (where the baseline has plenty of data, and it is used to train the whole model, compared to training only half the model in case of transfer learning). This will be the case for both *IRONOFF* and *QuickDraw!* datasets.

To summarize, our contributions in this chapter are:

- We show how to use the conditioned-autoencoder framework in order to perform style transfer.
- We propose two protocols – different in power – in order to perform the experiments of transfer learning.
- We intensively evaluate the usage of transfer learning relative the baselines, across a wide range of metrics.

- We perform the tests on two different datasets, for extra confirmation.

5.2.1 IRONOFF

In case of handwriting letters, we distinguish between three tasks: uppercase, lowercase and digits[3].

We explore the idea of transfer learning on all possible combinations of tasks:

- From uppercase and lowercase to digits,
- from uppercase and digits to lowercase,
- and from lowercase and digits to uppercase.

Experimental setup

Figure 5.3 details the protocol we used for this experiment. For each source/target task combination, we first perform hyper-parameters search for the network for both the source and the target tasks. The best performing hyper-parameters gives use the source and the target models. Using these models:

- We use the source model to extract the encoder module (aka, the style extraction module). We then add it to a new model, freeze it (it will not be part of training), and train the new model on the target task. We also search for the best hyper-parameters for this new model. We call this model the 'transfer model'. We retrain this transfer model 5 times with different random weights each time, and report the stats on the cross-entropy of the test data for these repetitions.
- We retrain the target model 5 time as well.
- Then, we use the best performing transfer and targets models for generating the target tasks, and report the stats of the different generation metrics.

We use 10% of the data for testing, another 10% for validation, and the rest is our training set.

[3] I do not see a problem of working on the lower categories (A, a, B, b, ..., 7, 8, 9). However, it would have consumed a lot of computational time test on all possible combinations of source/target tasks.

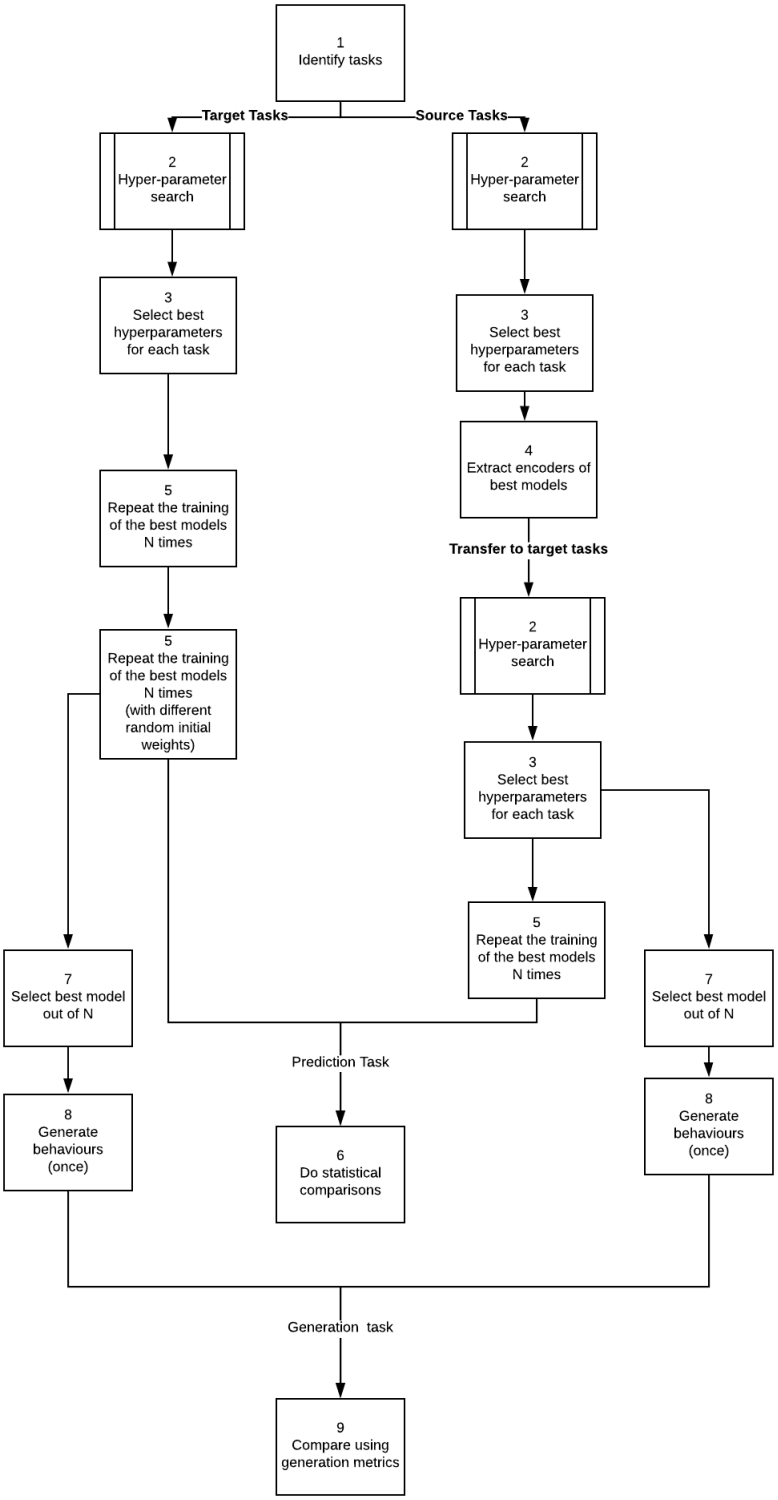


Figure 5.3: Flow chart explaining the experiment protocol used in *IRONOFF* dataset.

Results

Loglikelihood of prediction In the prediction mode, the model is tested in a similar manner to the way it was trained: it is given the input from the ground truth, representing the current time step, and asked to predict the next time step. This tells us about the quality of the training procedure from one side, and shade light on the confidence of the model in predicting the next time step.

The result of 5 times repetitions, for all the different combinations of source/target tasks, are mentioned in figure 5.4. We can see that the transfer learning always gives a significant advantage over baseline models.

BLEU score Now we go to the generation part, our main concern in my thesis. As discussed earlier, we use BLEU score to assess the quality of matching segments between the generated and the ground truth letters, in a gradual manner (i.e., we increase the size of the segments to match gradually). Table 1 summarizes the numbers. We see that transfer learning always performs better than the baseline.

End-of-Sequence analysis One aspect to measure the quality of the generation, that we identified previously, is to analyze how the model predicts the end of the sequence generation. Table 2 shows the results of the Krippendorff coefficients for the different modes. In general, the different modes are performing quite well. It can be seen that transfer learning is actually performing better than the baseline models, adding another indication to the benefit of using transfer learning.

Strokes analysis As mentioned earlier in this section, we started to consider the the strokes in this part of our work. Table 3 shows the strokes results of the Krippendorff coefficients for the different modes. With the exception for the uppercase letters, transfer learning performs better than baseline models. For the uppercase, this could be due to the extra complexity of these letters (see figure 2.3, where it can be found that uppercase letters are usually the ones with higher number of strokes). A fine tuning for the style extraction module is the next logical step to perform here, to make it adapt to complexity of the this task. We also consider the confusion matrix for the generated strokes in comparison with the ground truth, figure 5.5. We can see that it is consistent with the Krippendorff results, where transfer learning outperforms the baselines. In the uppercase letters however, it is notable that the transfer learning performs better on the single stroke, while the baseline performs better on the 2 and 3 strokes.

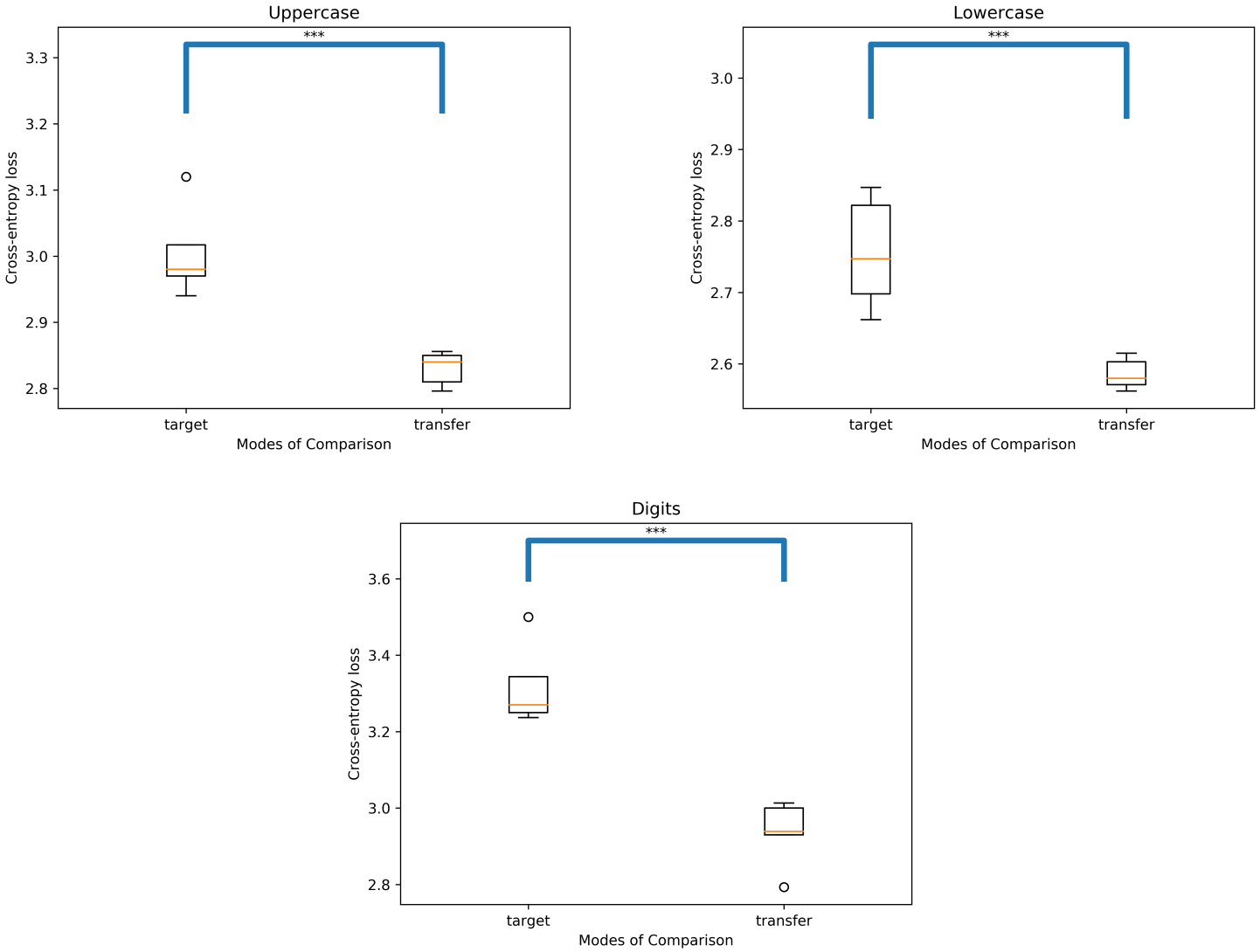


Figure 5.4: *IRONOFF*: log cross-entropy of prediction of test dataset for different combinations of source/target tasks, with 5 repetitions. We can see that, in all possible source/target task combinations, the transfer learning gives better advantage than just learning from scratch on the target task. The stars indicate the statistical significance level (1 for < 0.05, 2 for < 0.01 and 3 for < 0.001).

Aspect/Feature	Speed			Freeman		
Model / B-score	B-1	B-2	B-3	B-1	B-2	B-3
Uppercase-baseline	66.1	46.3	27.2	62.8	49.4	37.1
Uppercase-transfer	**68.3**	**47.8**	**28.3**	**65.47**	**51.8**	**39.0**
Lowercase-baseline	73.1	69.7	55.9	54.8	37.2	40.9
Lowercase-transfer	**75.5**	**71.2**	**58.0**	**56.0**	**39.4**	**41.9**
Digits-baseline	68.7	65.2	49.1	49.6	29.3	34.6
Digits-transfer	**71.5**	**70.7**	**51.2**	**55.9**	**31.4**	**41.7**

Table 1: *IRONOFF*: BLEU score results on the generated letters, for the baseline models (trained on the target task only), and the transfer models (the encoder – style extractor – is trained on the source task, while the decoder is trained on the target task). The results show the advantage of using transfer learning.

Task/Model	Baseline	Transfer
Uppercase	0.95	**0.97**
Lowercase	0.96	**1.0**
Digits	0.92	**0.99**

Table 2: *IRONOFF*: Krippendorff correlation coefficients for the End-Of-Sequence (EoS) distributions between the transfer and baseline, for all tasks.

Task/Model	Baseline	Transfer
Uppercase	**0.38**	0.25
Lowercase	0.56	**0.65**
Digits	0.4	**0.71**

Table 3: *IRONOFF*: Krippendorff correlation coefficients for the strokes distributions between the transfer and baseline, for all tasks. Except for the uppercase case, transfer learning seems to perform well in the lowercase and the digits tasks.

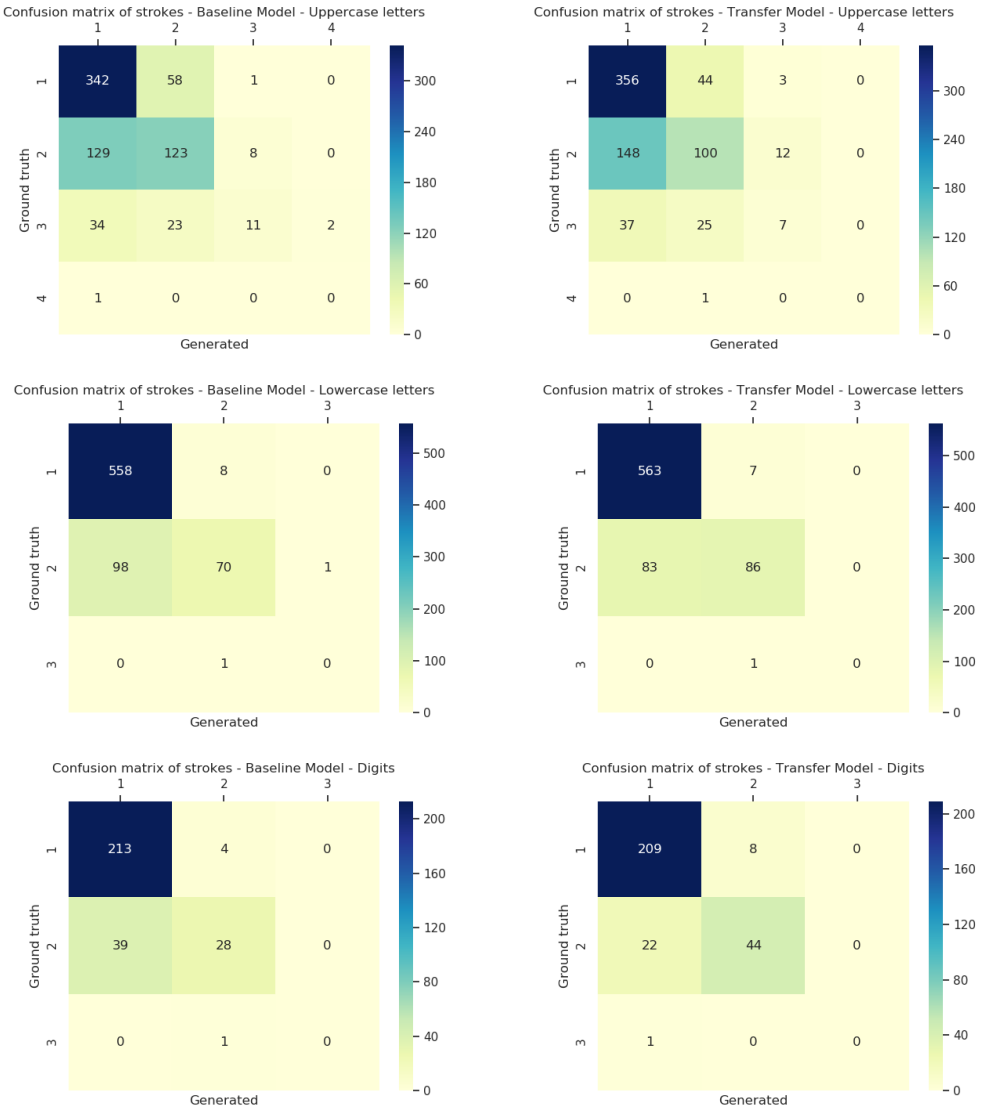


Figure 5.5: *IRONOFF!*: Confusion matrix for strokes for both baseline (left) and transfer (right) models, on the different tasks.

5.2.2 QuickDraw!

In case of sktech drawings, we define the task by the class it belongs to. So we have 5 tasks: circle, triangle, square, hexagon and octagon. We explore the idea of transfer learning on all possible combinations of tasks: in each combination, one task is removed (the target task), and the other task are considered source tasks.

Experimental setup

Due to the increase of the number of tasks – compared to *IRONOFF* –, and using the same experiment setup proved to be too much compared to the computational resources available to us. We identify that one of the expensive bottlenecks in our work is the number of times we need to perform hyperparameter search. In order to get around this problem, we are using a simpler (and less powerful protocol), see figure 5.6. We perform the hyper-parameters search once in the beginning of the experiment, on the all the tasks combined, to find good hyper-parameters that is suitable to the domain of 'sketch drawing'. From there, we fix the hyper-parameters for all the different steps in the experiment. The analysis steps are the same as in *IRONOFF*.

Another thing we noticed is that retraining the model 5 times only (like in *IRONOFF*) only leads to unstable conclusions, thus make it hard to quantify the difference between the baseline and the transfer modes. We increased the number of models to 30 in this case, in order to have a more consistent trend.

Results

Loglikelihood of prediction The results can be seen in figure 5.7. For all the combinations, there is a clear advantage for using transfer learning over the baseline model. The difference in all cases is statistically significant.

BLEU score The BLEU score results are summarized in table 4. Transfer learning outperforms the baseline models in this case.

End of Sequence analysis Table 5 summarizes the results on end-of-sequence analysis in case of quickdraw. The benefits of transfer learning are clear.

Strokes analysis Table 6 summarize the results on strokes analysis in case of *QuickDraw!*. The benefits of transfer learning are clear. We also report the confusion matrix

Figure 5.6: Flow chart explaining the experiment protocol used in *QuickDraw!* dataset.

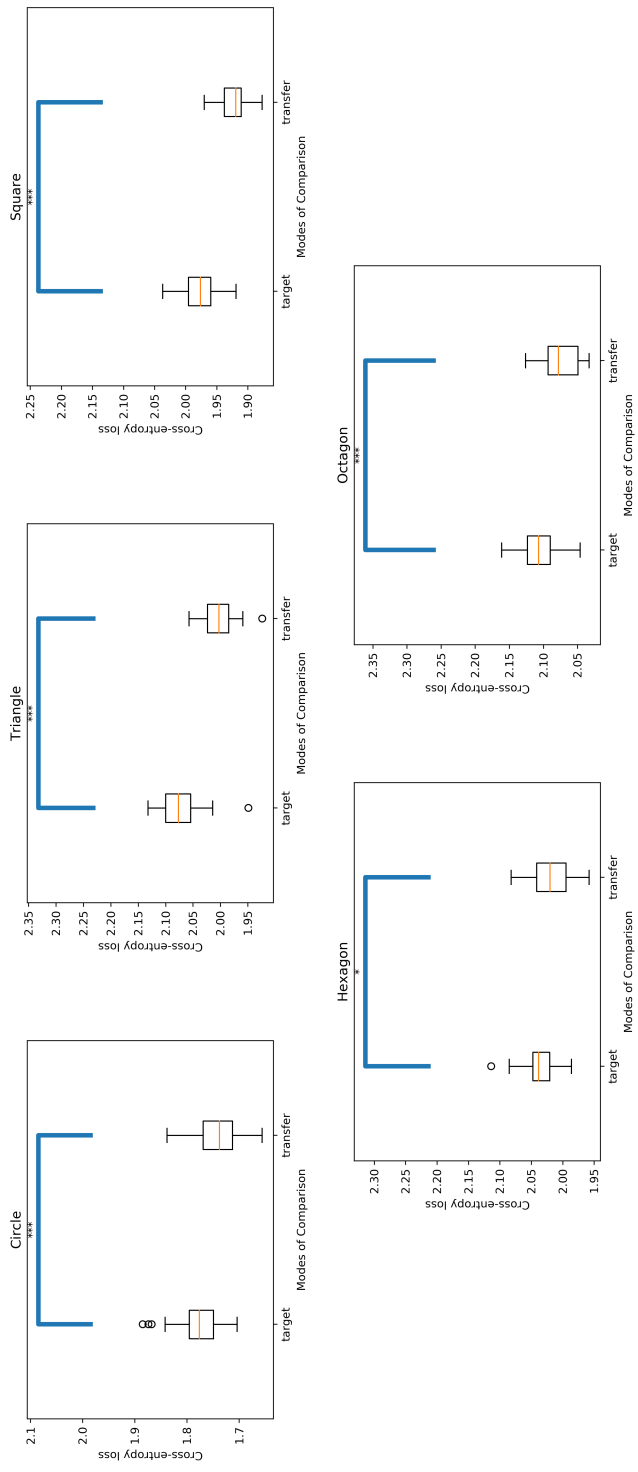


Figure 5.7: *QuickDraw!*: cross-entropy of prediction of test dataset for different combinations of source/target tasks, with 30 repetitions. We can see that, in all possible source/target task combinations, the transfer learning gives better advantage than just learning from scratch on the target task. The stars indicate the statistical significance level (1 for < 0.05, 2 for < 0.01 and 3 for < 0.001).

Aspect/Feature	Speed			Freeman		
Model / B-score	B-1	B-2	B-3	B-1	B-2	B-3
Circle-baseline	59.0	54.5	49.6	60.0	54.7	48.9
Circle-transfer	**70.4**	**65.5**	**60.3**	**65.0**	**58.1**	**50.6**
Triangle-baseline	47.3	40.0	32.6	33.2	28.2	24.0
Triangle-transfer	**61.3**	**52.4**	**44.1**	**50.6**	**44.8**	**39.8**
Square-baseline	46.8	40.1	32.7	44.0	39.1	34.9
Square-transfer	**57.9**	**50.8**	**42.9**	**53.0**	**47.4**	**42.3**
Hexagon-baseline	58.1	50.4	41.4	45.4	40.3	35.9
Hexagon-transfer	**62.0**	**54.0**	**44.8**	**47.6**	**42.3**	**37.8**
Octagon-baseline	55.2	47.1	38.3	43.7	38.7	34.6
Octagon-transfer	**57.3**	**49.3**	**40.5**	**46.1**	**41.1**	**36.7**

Table 4: *QuickDraw!*: BLEU score results on the generated letters, for the baseline models (trained on the target task only), and the transfer models (the encoder – style extractor – is trained on the source task, while the decoder is trained on the target task). The results show an advantage in using transfer learning.

Task/Model	Baseline	Transfer
Circle	0.6	**0.84**
Triangle	-0.05	**0.61**
Square	0.04	**0.35**
Hexagon	0.07	**0.2**
Octagon	0.07	**0.16**

Table 5: *QuickDraw!*: Krippendorff correlation coefficients for the end-of-sequence distributions between the generated letters and the ground truth letters.

for the strokes, figure 5.8. It is notable that the performance of the system in general decreases, as the diversity in the number of strokes increases. But still, transfer learning provides the better results.

Task/Model	Baseline	Transfer
Circle	-0.04	**0.1**
Triangle	-0.04	**0.42**
Square	0.03	**0.25**
Hexagon	-0.08	**0.23**
Octagon	0.06	**0.18**

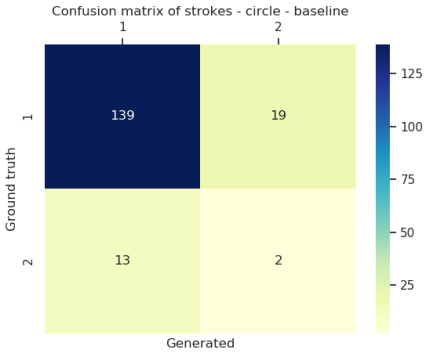
Table 6: *QuickDraw!*: Krippendorff correlation coefficients for the strokes distributions between the generated letters and the ground truth letters. Transfer learning achieves better results than the baseline on all the different tasks.

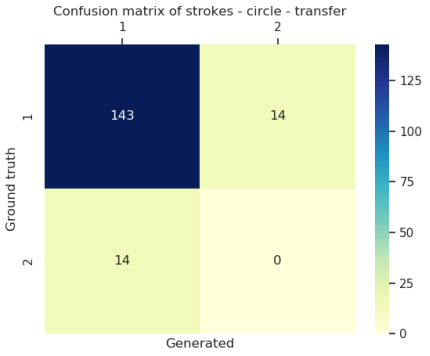
5.2.3 A word of caution about confusion matrix

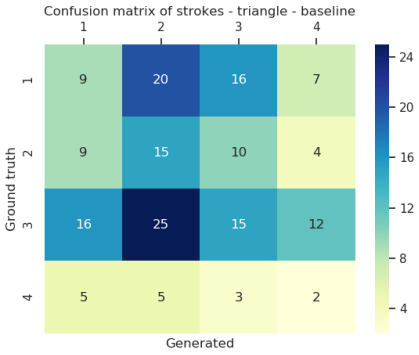
In the previous two experiment, we presented the confusion matrix for the strokes. However, it is important to understand what these numbers actually mean. In a normal classification problem, the confusion matrix gives important information about the precision and recall of the classification. When comparing two algorithms – for example –, the usage of the values of confusion matrix is an acceptable approach to consider. However, this is not the case when dealing with the generative aspects of machine learning. Bear in mind the asymmetry between the training of the model (using the log-likelihood of prediction) and the usage of the model (by sampling from the model), thus, a one-to-one comparison between what is generated and the ground truth is not possible or meaningful. A logical consequence for this is that we can not use such a method to compare two models together (i.e., compare transfer learning and the baseline). What we would like to achieve with generative model is to *capture* the distribution of the ground truth, thus, for a meaningful comparison, we want to compare the distribution of the generation relative to the distribution of the ground truth. That is a core challenge in my thesis: to provide tools to capture the distribution, to facilitate the comparison and evaluation of different models/methods.

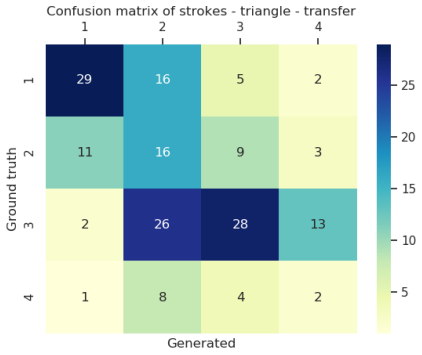
One way to look at the confusion matrix is the general trends in the matrix, like:

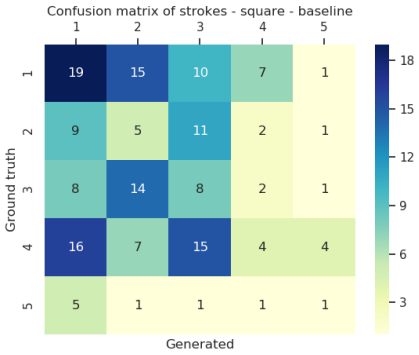
- The deviation around the diagonal: as the shapes gets more complicated, and as the number strokes increases, we expect that the deviation will increase.
- Looking for systematic errors (e.g., the number of strokes for a square is always one instead of being three or four): this could help diagnosing problems with the models design, data processing...,etc.

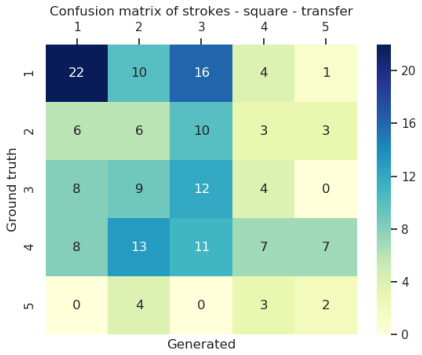

Confusion matrix of strokes - circle - baseline
Ground truth
Generated
139
19
13
2


Confusion matrix of strokes - circle - transfer
Ground truth
Generated
143
14
14
0


Confusion matrix of strokes - triangle - baseline
Ground truth
Generated
9 20 16 7
9 15 10 4
16 25 15 12
5 5 3 2


Confusion matrix of strokes - triangle - transfer
Ground truth
Generated
29 16 5 2
11 16 9 3
2 26 28 13
1 8 4 2


Confusion matrix of strokes - square - baseline
Ground truth
Generated
19 15 10 7 1
9 5 11 2 1
8 14 8 2 1
16 7 15 4 4
5 1 1 1 1


Confusion matrix of strokes - square - transfer
Ground truth
Generated
22 10 16 4 1
6 6 10 3 3
8 9 12 4 0
8 13 11 7 7
0 4 0 3 2

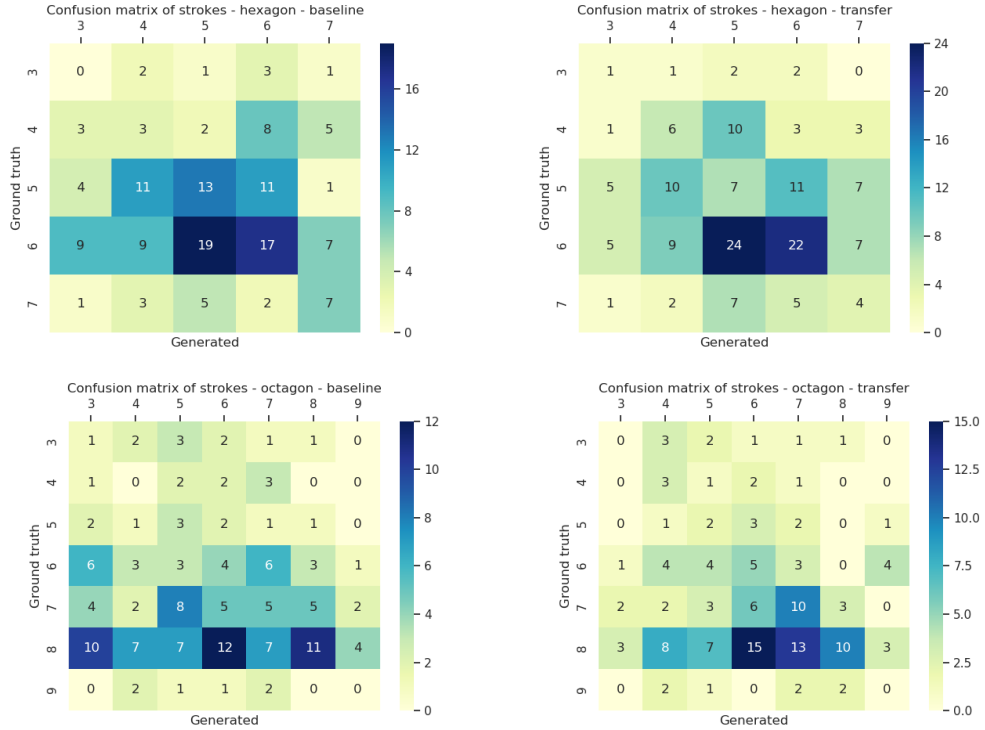


Figure 5.8: *QuickDraw!* Confusion matrix for strokes for both baseline and transfer modes, on the different tasks.

5.3 Are we actually capturing styles?

In the previous section, we showed that all the metrics indicate the benefits of transfer learning over using the target data only. We argued that this approach captures the styles, and that we are transferring the styles to a new task. But, are really capturing the styles this way?. In section 4.2.4, we tried to motivate this point, by some exploring the bottleneck of the styles. We showed that looks consistent with what we know beforehand (in case of letter 'X' for example), or that it uncovers things we either did not notice beforehand (like in letters 'C' and 'S'), or uncover new things that we did not anticipate (in case of letter 'A').

Testing for something that is ill-defined, and not always known beforehand, like styles, is quite challenging. To add to the challenge, the methods we used to explore the bottleneck space (PCA and tSNE) are not designed to uncover a 'styles' manifold, so sometimes, basic styles went missing (for example, with this exploration methods, we could not uncover the clockwise/counterclockwise direction in letter 'O' for example, which we know a priori that it exists!).

In this section, we want to shade more light on this problem, from a different and quantifiable angle. We will choose one aspect using *QuickDraw!* dataset this time, we manually annotated circles and octagons into clockwise/counterclockwise categories. Then, we build a classifier on top of the styles bottleneck, to measure the quality of the capturing of this style aspect. If we can not see this style visually with PCA and tSNE, then we should be able to measure its impact.

We annotated 716 drawings (see table 7). We trained a classifier (*Random forests* classifier) on this data directly, randomly separating training and test data, and repeated this 5 times. We did not configure the hyper-parameters of that classifier (we want a general figure, not interested in an exact one). We obtained 92.9 6.0 % on the weighted F1 score. This suggests that this style extraction method does indeed capture this style aspect.

Task/Direction	Clockwise (CW)	Counter-clockwise (CCW)
Circle	339	57
Octagon	115	205

Table 7: Results of manual annotation for CW-CCW on 716 drawings (octagon/circles) in *QuickDraw!* dataset. Sometimes the drawing is not clear, so we did not include. The selected examples are the clear ones only. It can be seen that the data is not balanced.[4]

5.4 Summary and take-away message

In this chapter, we presented our hypotheses for style transfer learning. We presented our proposed approach, and the proposed experimental protocol in order to investigate these hypotheses. To have better support for our conclusions, we carried out the experiments on two different datasets, *IRONOFF* and *QuickDraw!*, both of them presenting different challenges and behaviors: *IRONOFF* has more clear semantics, made by a pen, while in *QuickDraw!* the contributors exerted more freedom on their work, and most of the work is believed to be done by the mouse or maybe a tablet (using a finger or a pen). Also, we chose to perform the study on the worst case scenario, where there is an abundance of data available to the target task. To better understand and compare transfer learning versus the baseline models (models trained only on the target task), we considered multiple performance metrics: the log-likelihood of prediction, the BLEU score, end-of-sequence and strokes analyses for the generated shapes.

The results overwhelmingly point to the benefit of transfer learning compare to the baselines on the different proposed metrics, thus providing a strong evidence to our hypotheses.

Part III

Discussion and Closing Remarks

Chapter 6

Prespective and future work

Contents

This chapter will be a free discussion about what I had done, lessons learned, shortcomings of this work, difficulties in the PhD, and potential areas of development.

Science should always be about honesty, humility and respect, and not just flashy results and wide conclusions. Science can always make use of learning from setbacks – something that is almost missing in the scientific literature –. I will do my best in this chapter to highlight the other side of good results.

Maybe one day someone – maybe a PhD student – will decide to follow on this work. It is important for me that they do not repeat the same mistakes. Instead,

everything should be ready for them in order to make new mistakes. That is how we move forward. After all, we our objective in this PhD is transfer learning between different tasks. It is now time to transfer learning between two different colleagues.

6.1 Challenges

In this section, I will discuss the challenges faced during the PhD – from my personal perspective –, in the hope that I will learn from it for the future, and to illuminate the path for others as well.

6.1.1 Choice of how to tackle the topic?

The usage of deep learning in this topic has not been the first or clear option from the beginning. As I mentioned earlier, the objective of the project was to develop components for human-robot interaction domain. In a first glance, deep learning and human-robot interaction do not really mix well. The problem is simply the availability of data.

There is a stretch of imagination in this thesis, that we assume – and I believe rightly so – that the data problem in HRI will be resolved in the future. Better and more reliable hardware is becoming available, and there is a general awareness now in the community about the need to do something concerning the data: a lot of data is being recorded by the research group, but there is no standardization or culture of open sourcing the data, even within the same team, leading to a big waste of efforts and time.

At first, it seems that more data-efficient methods – that depends on well designed priors from humans – are the way to go in such project. However, the advances in deep learning application in areas like speech synthesis, image captioning, text and music generation, and the lucrative possibilities that data-driven approaches provide were hard to ignore. Besides, the current advances in machine learning indicate that computational approaches, even with simple algorithms, are outperforming methods that depends on human knowledge and prior[1].

6.1.2 Determine the scope of interest in the state-of-the art

A major challenge during the PhD was to determine the relevant state of the art. For generative models, the usage of deep learning methods was not the clear choice of the beginning, and once chosen, it took considerable effort to determine the scope of the relevant literature.

The same goes for the state of the art on styles. The word itself, and the

[1] As nicely noted by Richard Sutton, one of the god fathers of reinforcement learning, in his article *The bitter lesson*, `http://incompleteideas.net/IncIdeas/BitterLesson.html`

range of study, is very wide – as noted early in the introduction. By far, the work done in handwriting styles was the least relevant to our work; most of the work is done on offline handwriting (thus not dealing with the dynamics of writing itself) – and this is most of the work done currently –[2]. The problem does not always manifest itself in a technical shape, sometimes – and most annoying – it is mostly that we do not know what we are looking for exactly, and even if we do know, we do not know the exact terminology other people are using to describe it.

I am thankful for the great deal of openness that researchers in machine learning are embracing. The discussions through online forums, blogs, tutorials, online courses, and recent books, had definitely made this massive search space more tractable.

6.1.3 Lack of Benchmarks, evaluation metrics

Getting around these issues was quite a dilemma, for many reasons:

- I do not believe it is a healthy practice to pick up the benchmarks to use. This choice can be easily biased, and it could be argued that the benchmarks are chosen to be weak enough in order to show progress. In our case, it was mandatory to do so nevertheless, and we tried as much as possible to be fair in making these choices.
- In engineering, it is the right practice to have different teams for design and test of the product. If one team do both, the testing process tend to be biased (i.e., even with the best intention, the team is looking for confirmation of their design, not the problems in it). By analogy, I think this a pitfall of us developing the metrics, the benchmarks, and using them. What if we are mainly looking at the metrics that confirms our hypothesis? It is hard to rule out this possibility.

We tried our best to avoid this when selecting the metrics, and by using multiple metrics to evaluate our hypothesis. However, an independent investigation in this issue is favorable.

6.1.4 Deep learning: theory, hardware and software frameworks

Several excellent frameworks – like *Keras* (Chollet et al., 2015), *PyTorch* (Paszke et al., 2017) and *TensorFlow* (Abadi et al., 2015) – do exist at the moment in order to provide

[2]Most of the great advances that happened recently in neural networks – especially in generative models – is related to computer vision. Thus, it is more convenient to deal with with handwriting as images than as a dynamic process.

friendly APIs for deep learning, with many online tutorials. This gives the impression that you can just jump in the topic, train a neural network, and now you can harness the power of deep learning. However, in my experience, the quick gains of this approach will be lost soon in the face of the first problem. Even for someone experienced with traditional machine learning, deep learning poses an extra challenge: it is usually computationally expensive. More thinking is needed about what to do and what not to do in this case.

It is important to understand the fundamental of statistical learning theory (Hastie et al., 2001), machine learning and deep learning before engaging in an endeavor that uses deep learning. This particular strategy was a key factor in any progress done during my PhD. There are plenty of excellent online courses, free books and many resources, that provide a gradual and methodological approach, which a learner can use in order to achieve this.

Another thing to highlight here is the hardware needs. Having access a GPU is a necessity in order to learn, experiment and develop using deep learning. Recently, interesting cloud-based solutions – like *Colab*[3], which is free – provide quick access to hardware suitable for deep learning. Some other solutions that I used – like *Amazon AWS*, which is not free – do exist, with the advantage of being easily configured and scalable, and with a support from the research institution, could provide a good replacement for buying and maintaining expensive hardware in-house. The bottom line is: it is important to keep in mind the hardware available, otherwise, the whole process will be hindered.

Another aspect to consider is the framework to use. *Keras* for example provide interesting high-level APIs, while *PyTorch* and *TensorFlow* provide low-level APIs. *PyTorch* focuses more on being close to the Python language way of thinking, while *TensorFlow* provide a wide variety of interesting functions, including deployment capabilities[4]. It is lucrative to go for *Keras*, but once a low-level development is needed, I find that it adds an unpleasant overhead, requiring a mastery level of the underlying framework. Besides, in my opinion, starting by working on high-level directly encourages bad practices – since everything is done in background, it is easy to bypass important details in the way deep learning works, thus, developing poor debugging and problems diagnostic skills –. Discovering *PyTorch* was, by far, the unspoken hero in this PhD, and one of the best engineering decisions I have made. My point from discussion is illuminate the different trade-offs between the different platform. No one is better than the other. It is important to understand the task in hand, and choose the suitable tool for it.

[3] `https://colab.research.google.com/`

[4] This gap between *PyTorch* and *TensorFlow* is closing, with every new version of both.

6.2 Limitations of the current work

In this section, I discuss what I consider shortcomings for some of the methods used in this work.

6.2.1 Style extraction and exploration using PCA and tSNE methods

In this work, when exploring the latent space of our model, I used either PCA or tSNE projection methods (to project the latent space from the high dimensional space into a smaller one)[5], and tried to use the assumptions behind both methods to extract meaningful information from the latent space.

While this is an acceptable approach, it really stretches these methods to a breaking point, plus, it may hinder further investigation:

PCA It assumes orthogonality and linearity in the space to be projected. There is no reason however to assume that these assumptions hold for different styles. In the non-linearity aspect, the latent space does not have the clear objective of transferring non-linear style relationship into linear ones (simply, because no such objective can be formulated directly, since the problem of styles is ill-defined), unlike what can be noticed for the last layers of neural network classifiers (where an embedded objective of the network is project the data from their non-linear manifold into a linear one). Finding orthogonality in the style space is an interesting aspect to explore, but this is a strong assumption, and there is no reason to believe that it holds for all aspects of styles.

tSNE It provides a way to deal with non-linearity, thus allowing another further exploring the latent space, but it is hard to repeat the results (the method is stochastic) and the projection does not necessarily yield information about the styles. Changing the *perplexity* parameter leads to different results as well (I didn't explore the relation of that parameter to find a more suitable style manifold, and I am not sure if it is worth the effort).

But what is a good projection criteria in this case? should we let the organization of styles emerge on its own, by constraining the latent space and add regularization to the loss function (i.e., during an end-to-end training of the network)? should a second optimization step be performed on the latent space, in order to disentangle it? I discuss some of these ideas briefly in section 6.3.

[5] Or I used the latent space in a classification task to identify if particular information exists in the latent space.

6.2.2 Leak in the style module

The idea of conditioning is to provide the task content/identity as an input to the decoder (the condition), thus, relieving the encoder from learning it, and focus only on learning the styles, thus enhancing the style transfer capability. Ideally in this case, we expect that the output of the encoder has little to none information about the task identity. However, careful testing shows that this is not the case. There is a considerable leak of information about the task content/identity into the encoder.

I do not have an explanation at the moment for the reason behind this phenomena. My intuition[6] is that the problem lies in the way we describe the task content. The assumption in my thesis is that a harsh one-hot encoding of the task is sufficient to describe the task correctly is flawed in my opinion. An analogy for this can been drawn from clustering (hard clustering versus fuzzy clustering). Hard clustering, similar to one-hot encoding, does provide us with which this task is, but nothing about how this task relates to other tasks (i.e., proximity/similarity to other task), which is what fuzzy clustering do.

The influential work done by Geoffrey Hinton in (Hinton et al., 2015) – performed on the MNIST dataset (LeCun and Cortes, 2010) – is a contributing factor in my intuition. I will not dive into details about this article here, since it is outside the scope of this work, I will just mention two interesting results from this study:

- In a classification task, the traditional description of the labels is one-hot encoding. However, using a soft/fuzzy description of the labels reveals much better results (makes sense, since it is more rich in information).
- If you train a classifier on the soft labels of digits 7 and 8 only, the classifier will perform almost 90% accuracy on the other labels [7]!. It means that a better task description may increase the data efficiency of the model.

A similar concept should definitely be explored in the context of this work.

6.3 Future directions

During my thesis, with each step, with each question answered, the door was unlocked to many new questions. This is the beautiful part about science. The not-so beautiful part is that time is limited, and choices have to be made, I can not pursue them all. I try to document what I believe is the possible directions to go from here in this section.

[6] I did not have the time to perform rigorous testing for this idea unfortunately.
[7] I personally find this particular result fascinating.

6.3.1 Disentanglement of latent space to uncover styles

So far, we did not try to impose any constraints or structure on the latent space. We used general post-processing methods (PCA, tSNE or classification) in order to shade some light on the content of the latent space. Adding structure to the latent space is very interesting, it may allow the emergence of these styles on their own. Some work has been done in that direction in case of images (Higgins et al., 2017). An example of adding structure can be simply by forming the latent space as a number of Gaussian distributions (Kingma and Welling, 2013), enforcing discretization in the latent space (Jang et al., 2016; Maddison et al., 2016; van den Oord et al., 2017), or many other ideas. Another interesting dimension to explore is to have a hierarchical latent space structure (Hsu et al., 2018), which can add more interpretability to the latent space, enabling a better comprehension of the extracted styles. In all these examples, the model is optimized end-to-end.

Finding a good structure and constraints is probably a challenging task though, but the potential rewards are huge, for two reasons at least:

- It can be an efficient way to discover and understand styles. After all, an important aspect of my PhD is to use machine learning as a tool to perform science, as discussed in the introduction. Facilitating this objective is a good news.
- In a generative framework, this structure can be seen as control knobs for the generator. A good disentanglement will give us meaningful control knobs. Once we have them, we can start use them to synthesize new data with the characteristics that we want.

Another possible direction to look at a better post-processing methodology. For example, given some criteria, we can optimized the learned latent space in order to structure the latent space. I implicitly did this by using PCA and tSNE, but we can envision developing our own methods and cost functions that will help with the problem in hand.

6.3.2 Data efficiency

By data efficiency I mean that amount of data needed in order to achieve the desired performance. This depends on many things, including the data itself, the complexity of the distribution, the machine learning algorithm used...,etc. This point is important to consider when data is expensive, like in robotics for example. In my thesis, due to the availability of the data in the chosen domains, the problem of data efficiency did not surface and was not of concern. It can be considered that transfer learning is one way to address data efficiency (by requiring less data samples in order to learn the

target task). However, some aspects of data efficiency needs to be addressed heads on: for example, during HRI, the robot may need to adjust its behavior to suit to the human. In such case, some trial-and-error is needed in order to get proper data that will allow the robot to modify its approach. This data must be limited – the human will not withstand weird behavior from the robot for so long –. Some research work is in progress to investigate learning policies from a handful of trials (Chatzilygeroudis et al., 2018; Cully et al., 2015). This point is a necessity for any successful deployment on the robot.

6.3.3 Perceptual evaluation and system specification

In this thesis, we showed the validity of transfer learning in case of styles in an objective manner, using many performance metrics, which we believe that they matter and are relevant. But there is an important point we did not address: how much difference in each performance metric do we actually care about? If I say that, concerning the EOS metric, that one system has a Krippendorff correlation of 0.9 and another one has 0.95, the question here is simply: should we be concerned about this 0.05 difference? What is th minimal acceptable value? and how much difference should concern us? And if we do care about this 0.05 difference, then is it worth the effort spent in order to get it (time wise, model complexity...,etc)? It is very important to go from the numerical universe to the physical universe, and get a sense of what those numbers actually mean, and determine what to care for.

This leads to another important point, which is to determine what do we actually want (i.e., develop the necessary specifications and criteria for our final objective). In case of human-robot interaction for example – I will use arbitrary numbers here –, we can consider that the robot can try a particular action with the human three times only in order to get it right (maybe the human will get bored after). Thus we need to consider algorithms that can update their decisions efficiently. Another question could be about the acceptable level of performance that should be achieved in order to have a successful interaction. This will give good guidance for the development of new algorithms.

Bottom line is, it is important to know what we want first, and what is the limitation that we have, otherwise, the development can be easily misguided. It is important that when we do machine learning, we do machine learning that actually matters (Wagstaff, 2012).

6.3.4 Experimental protocol

The experiment of transfer learning in chapter 5, in my opinion, is quite complex, with too moving parts. A good portion of the time I spent on this experiment was on developing the protocol of the experiment, and I am still not feel fully satisfied about it. For example:

- The choice of the number of repetitions in case of *IRONOFF* (5 repetitions) was based on the numbers in (Lathuiliére et al., 2019), but there is not good motivation for this number. Performing an in-depth analysis into this matter is possible – I believe –, but would have consumed so much time and resources beyond our capacity. The choice of 30 repetitions in case of *QuickDraw!* is also arbitrary; it is high enough to avoid any statistical problem, but maybe a lower number is possible.

- I did not study the role of the random seed in the performance of the networks. This is an issue in case of reinforcement learning – it is important to test for different random seeds –. I am not aware of something similar for the kind of generative models used in my thesis, but that does not mean such an effect should be ruled out.

- I used a weighted random generation policy during the PhD. It is simple and quite effective. However, during such an experiment, there is a question on how to take it into account: should I consider performing a statistical measure on different generation runs? should I take the best of a number of generation?

  A similar problem goes for the weights of the neural network. I repeated the training N times. Should I report the generation over those N times, or only for the best set of weights of those N times? In my thesis, I chose the best generation output on the best set of weights, under the assumption that performing such a statistical analysis will just focus on the generation policy and the weights initialization policy, which are things out of the scope of my concern. I can assume that over time, better generation and initialization policies will exist. This is argument though could be flawed.

- There is an argument about the necessity of performing hyper-parameters search for each step of the experiment – the source, target and transfer parts –. In *IRONOFF*, I performed this in all the parts of the experiments. In *QuickDraw!* however, due to the need for more repetitions, a choice had to be made – we can not computationally afford all of this –. I fixed the hyper-parameters in all parts of the experiment, and focused on the repetitions only.

  There are two arguments here for which is the correct protocol: my argument is that machine learning is a search problem, thus, in order to claim that one approach is better than the other – transfer learning is better than the baseline

models in that case –, a search process is needed for both of them. The data quantity is also not the same for both approaches, thus, a particular choice of hyper-parameters could be large for one approach, and small for another approach. The other side of the argument is that, to assess something like transfer learning, we need to test them both on the same hyper-parameters, thus, we can know what is the actual advantage of one approach over the other. I am more inclined towards my arguments, as I believe it fits within the core of machine learning paradigm. However, probably there is a point in between where the both arguments meet. Worst case scenario in my opinion is that the second argument leads to a limited test about transfer learning. Revisiting and rethinking this process is essential.

Testing for all these issues is simply not feasible in my case, but I do genuinely believe that building a solid protocol is essential in order to have solid conclusions.

6.4 Summary

In this chapter, I explained some of the major challenges of my thesis. Finding the right balance between the angle of attack, the methodology to be used – and the constraints that come with it – and what needs to be done, is the key to have a feasible thesis. After all, resources – mainly time – are limited, thus the need to visit this balance. It will never works well from the first time; evaluation and adaptation are needed every now and then. I discussed also my personal intake on how to think when using deep learning, in terms of hardware, software framework and different trade-offs.

I then moved on to discuss two limitations of our work. The way that we explore styles in the latent space at the moment has shown interesting information, but it is simply inadequate on the long run. We also identified that, despite using a conditioned-autoencoder, that there encoder is still trying to learn information about the condition. We expect that a better separation between the style and the task content/identity will lead to better transfer results, thus, more work is needed to investigate the reasons behind this leak.

Last, I proposed three directions to go from here, which I believe are important. Structuring the latent space in order to allow information about styles to emerge is a very interesting direction: it will make it more easier to interpret the styles in the data, and can have the potential to be used as control knobs in order to synthesize new data with the desired characteristics. Data efficiency is another direction, where algorithms are developed in order to take into account the data that we can actually acquire. It is also time to move from the numerical world (where my work resides) into the physical world, to better understand what these numbers actually mean in reality, what is the differences that matter in the different metrics, and what is the minimum thresholds

in each of those metrics needed in order to achieve a satisfying performance. Finally, I discussed the importance of the experimental protocol, especially with a complex experiment like the transfer learning one, and the need have clear objectives, proper statistical tests and alignment with the and paradigm of machine learning.

Chapter 7

Closing Remarks

Contents

Here we arrive to the end of the journey. I would like to take the space here to summarize what were the objective of my thesis and the motivation behind those objectives, what was achieved during this thesis.

7.1 At the beginning...

We started with a hypothesis that, any task human do consists of two parts: a content/identity (the core of the task) and the style (the manner the task is performed). My thesis represents our interest in studying styles, in the framework of machine learning. The reason for using machine learning is that data are becoming more complex, and applying traditional tools on it directly is no longer effective. Machine learning provide a tool for scientific research in order to explore large amounts of data; as an inquiry to see if particular information exists (as in classification), or in summarizing and compressing the data into suitable manifold, that we can perform analysis on.

Dealing with styles is a problem that emerges in many areas, most relevant to us, in case of human-robot interaction. Applying machine learning algorithms in a naive way directly on the data in order to learn models of human behavior leads to an averaging phenomena, where the specific style of the humans are averaged and removed. Thus, we need to find a way to extract those styles, and enable the machine learning algorithm to take into account while building models of interaction. Taking

the style of the human into account during the interaction is important in order for the human to have a better experience, thus allowing some level of trust and confidence to emerge during the interaction.

The problem of styles is ill-defined, and poses a lot of challenges in the way we can approach it. It is not clear how to think about the problem, what framework to use, what suitable metrics to evaluate the styles, and to what benchmarks we should compare different methods. Some think about styles in an explicit way – some particular aspects exist – and some think about it in an implicit way – generating behaviors and comparing them to the desired behaviors –. Besides, we do not have suitable data in the area of human-robot interaction in order to performs such study. Building this data set from scratch would have been very expensive.

In order to have a more controlled environment, where we can study styles, we focused our attention on the problem on another problem, that we believe has relevant characteristics to our original objective: online handwriting and sketch drawing. There are several advantages of working on such domain, like the availability of large quantities of annotated data, and the task content/identity is well defined. This allowed us to focus on the problem of developing proper methods to study styles.

We were curious about the idea of styles in different tasks: how can we study them? is it possible to extract them? and if so, are they transferable between different tasks? Transferability of styles between task is an immensely useful idea; it can save us a lot of work in terms of collecting and annotating new data. Plus, it can allow us to better understand the common styles and aspects between different tasks.

7.2 What did we do?

The first step is to break these general objectives into smaller ones, address these small objectives, and then combine them to realize the original general objectives. For the case of studying styles, we did not have any evaluation metrics, benchmark to compare to, framework to reason about styles. All these points are entangled together; tackling one of them will affect the other.

We first decided to use an implicit way to look at styles, by generating behaviors and evaluating them. Thus, given the plenty of data available, we went for deep generative models framework. We proposed evaluation metrics, but they need to be grounded – even if they look logical –, so we proposed multiple benchmarks, that we know beforehand their relative power, and used this knowledge in order to ground the proposed metrics. See chapter 3 for more details.

Once the basis were laid, it was time to study styles. We used a conditioned-

autoencoder to study styles. The condition was on the task identity/content, thus letting the autoencoder focus on learning relevant style information. We analyzed the latent space, and showed some examples of the extracted styles – some of them we were not aware of their existence before –. See chapter 4 for more details.

Once we successfully extracted styles, we had a strong interest in transferring those styles between different tasks. This is especially important when collecting and annotating data is expensive (like in case of human-robot interaction). We capitalized on the success of the conditioned-autoencoder framework we used, by reusing the relevant part (the encoder/style extractor) in new task. We expanded and refined the the evaluation metrics we use, and we added another data set, and performed extensive statistical tests to investigate our methods. We showed the validity and the potential of our approach. See chapter 5 for more details.

Finally, I presented what I believe as the take-away message from my personal experience during this PhD. I discussed the challenges faced (from choosing the choice of the research point. diversity of the literature, the lack of benchmark and performance metrics and the proper usage of deep learning), and the shortcomings of my work (the need for better/more specialized methods to explore the latent space and extract styles, and the leak of information about the task identity/content in the style extraction module), and what I think is the potential directions to investigate (the proper structuring of the latent space, the data efficiency aspect, and better realization and understanding for the physical implication of the evaluation metrics we used). See chapter 6 for more details.

Part IV

Appendices and Resources

Appendix A

Hyper-parameter tuning

Machine learning in its core is a search in the hypothesis space of a particular algorithm, in order to find the suitable parameters/hyper-parameters that best fit the data, while adhering the rules of statistical learning theory (Hastie et al., 2001). For parameters tuning in neural networks (i.e., learning the weights of the networks), back-propagation (Rumelhart et al., 1988) is one of the well established algorithms most commonly used[1].

Yet, when it comes to hyper-parameters, it is not that obvious[2]. There is two main strategies for finding hyper-parameters[3]:

- Manual search: for people with small computational budget. The idea is to get some sense of how the model behave, have educated guess overtime on the behavior of the different hyper-parameters, change them in a local manner.
- Using a search strategy: in order to better cover the possible hyper-parameters by using a heuristic to perform the search, and one of the dominant search strategies in deep learning is a random search Bergstra and Bengio (2012). It is important though to have a suitable range for the hyper-parameters, otherwise, a lot of computational resources can be wasted. During my thesis, I used the random search strategy.

  In a more aggressive kind of search, the search for hyper-parameters start from coarse and converge to fine type of search, in order to find the best possible set of hyper-parameters. I did not adapt such a strategy, just the coarse search.

[1] Several other algorithms do exist, like evolutionary algorithms (Eiben et al., 2003) – which recently are achieving remarkable results –.

[2] By not obvious, I am politely meaning it was an utter suffering!

[3] As wonderfully explained by 'Andrew Ng' in his *Deep Learning Specialization* in *Coursera* platform, `https://www.coursera.org/specializations/deep-learning`

Intensive methods to search for hyper-parameters, like grid-search, are not feasible with computationally demanding methods like deep learning.

The range of the hyper-parameters differs, for example, the learning rate of the optimizer is recommended to be on log-scale. For width of layer, I performed quick exploration on what are the upper and lower limits that makes sense (in my case, 64 and 256 served as the limits). For the number of layers, it was between 1 and 3. I search for the learning rate was done alone first, and then fixed for the rest of the experiments, to reduce the search time from one side.

Another point is when to stop the training of a particular set of hyper-parameters, and move on to the other sets. To completely converge, the network can take a lot of time (and many epochs). I used a very conservative approach, of using the validation set to determine the stopping point (early stopping), and putting a threshold on the max number of epochs. Probably a better (and faster) solution exist, but to the best of my knowledge, I could not find such a solution.

Appendix B

List of publications

This is a list of publications done during this PhD

B.1 International Conferences

- O.Mohammed, G.Bailly, D.Pellier. *Style transfer and extraction for the handwritten letters using deep learning*. 2019 ICAART (International Conference on Agents and Artificial Intelligence), Prague, Czech Republic.

- O.Mohammed, G.Bailly, D.Pellier. *Handwriting styles: benchmarks and evaluation metrics*. Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS), 2018. Valencia, Spain.

- Gerazov, B., G. Bailly, O. Mohammed and Y. Xu (2018), *Embedding Context-Dependent Variations of Prosodic Contours using Variational Encoding for Decomposing the Structure of Speech Prosody*, 2018 Workshop on Prosody and Meaning: Information Structure and Beyond, Aix-en-Provence, France

- Marielle MALFANTE, Omar MOHAMMED, Cédric GERVAISE, Mauro DALLA MURA, Jérôme I. MARS, *Use of deep features for the automatic classification of fish sounds*, 2018 OCEANS - MTS/IEEE Kobe Techno-Ocean

- Omar Mohammed, Gerard Bailly, Damien Pellier, *Acquiring Human-Robot Interaction skills with Transfer Learning Techniques*, HRI '17 Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction

B.2 Journal articles

B. Gerazov, G. Bailly, O. Mohammed, Y. Xu and P. Garner, *A Variational Prosody Model for the decomposition and synthesis of speech prosody*, to be submitted.

B.3 Infomal communication

- RHUM Workshop, presentation, 2017
- RHUM Workshop, poster, 2018
- FADEX sur l'AI, presentation, 2018
- RHUM Workshop, presentation, 2019
- JDD of DPC, presentation, 2019

Appendix C

Adversarial evaluation

In this part, I mention an experiment I performed, in order to investigate the problem of using an oracle to evaluate the generator, and use the quality assessment of the oracle as a feedback to the generator, in order to improve it. I will first describe what is the problem, then explain the hypothesis, and then show the experiments performed, and the lessons learned.

A similar work was done in (Nguyen et al., 2015) [1] and heavily inspired by the work done in (Papernot, 2017). I would like to thank *Ludovic Darmet* for the valuable discussions and resources about this topic.

C.1 What is the problem?[2]

Given a data distribution that separates points into two classes (as in figure C.1), and given some examples from that distribution (the sampled data points), the objective is to estimate this decision boundary given the sampled data points, figure C.2. Using test data in order to estimate the errors of the model estimation does not always reveal the problem or the boundaries that the model actually learned, figure C.3.

This is where adversarial examples comes in. Adversarial examples exploit the fact that that the there is a difference between the ground truth decision boundary and the model approximated decision boundary. The model is blind about what the reality looks like, thus, it can be exploited by some fake examples, in order to generate improper classification, figure C.4. The problem gets worse when we increase dimensionality of the problem – *curse of dimensionality* –.

[1] I was not aware of that work beforehand. The paper is neatly explained though.
[2] The images in this section are taken from (Papernot, 2017).

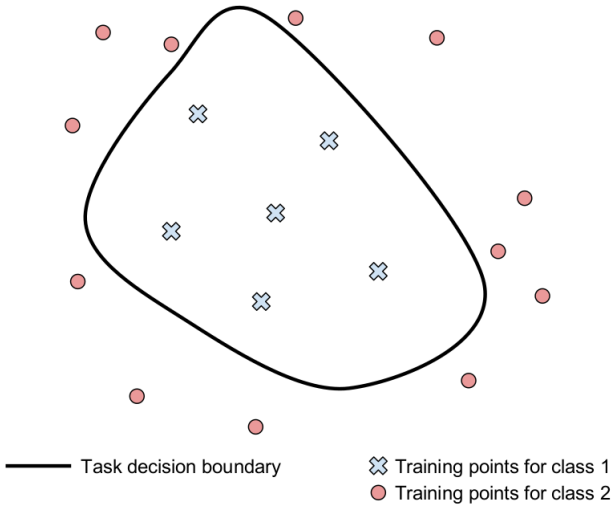


Figure C.1: The ground truth decision boundary that separates data into two classes. This decision boundary is unknown in advance. The objective of machine learning is to estimate/approximate/learn this decision boundary.

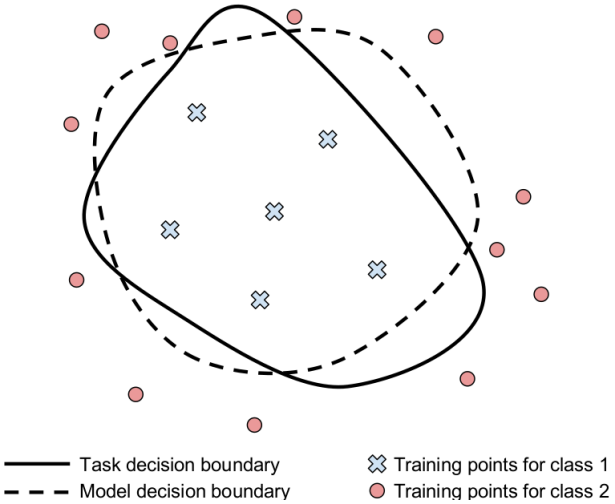


Figure C.2: The machine learning model estimate the decisions boundary based on the given examples. The estimation is not perfect, and not necessarily matches the ground truth boundary.

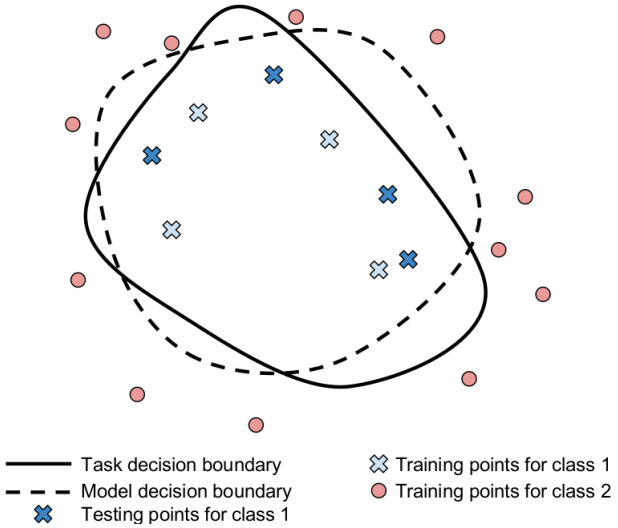


Figure C.3: Using the test data, sometimes we do not get a sense for the limitations of the model approximation. Even when the model shows errors, it is not possible to estimate the boundaries of the model approximation from this information.

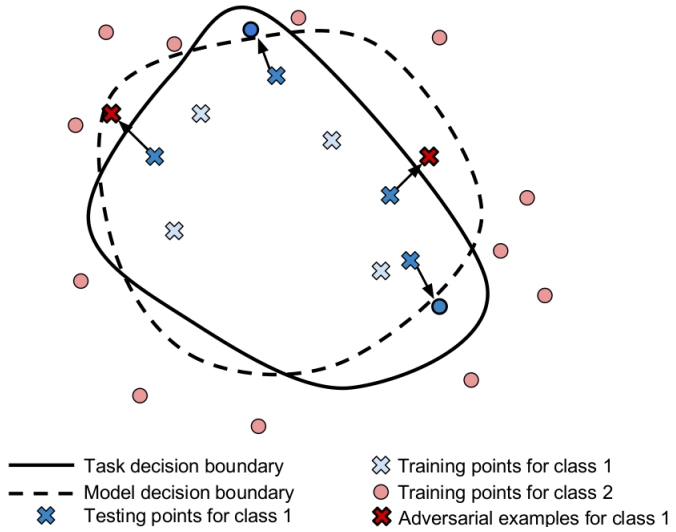


Figure C.4: Adversarial examples exploit the fact that the model only approximates the ground truth decision boundary, thus, there is a gap between them. Using this gap can easily misguide the model and lead to misclassification.

C.2 How to test this?

C.2.1 Experimental setup

I designed an experiment in order to see how this actually works. I build a variety of classifier models (oracles) – in order study the phenomena across multiple algorithms – on the MNIST dataset (LeCun and Cortes, 2010), figure C.5, and then build an image generator that is optimized on the outcome of each classifier (on the softmax outcome of the classifier[3]). The optimizer is an genetic algorithm (Eiben et al., 2003). The experiment setup is shown in figure C.6.

Different algorithms had been tested, starting from logistic regression, kNN[4](5 neighbors), neural networks (1 hidden layer, 5 neurons).

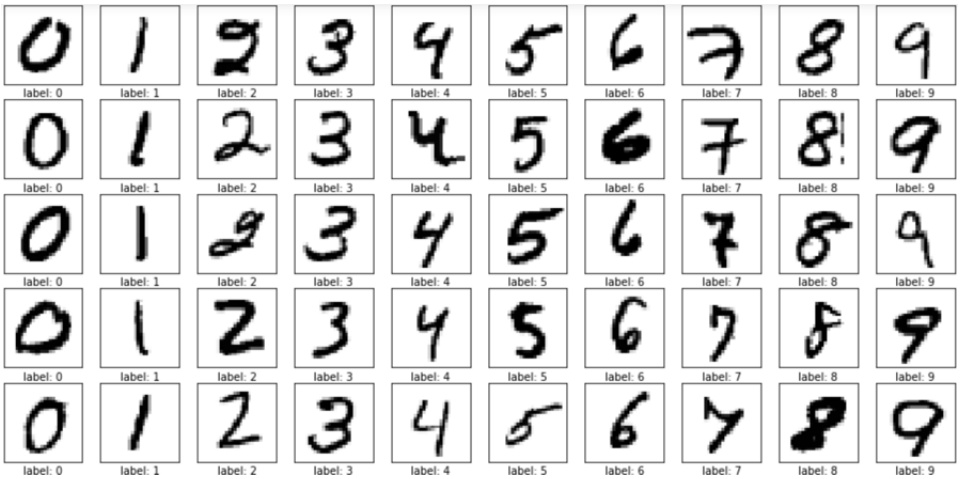


Figure C.5: MNIST is a popular offline handwriting dataset for digits from 0-9. 70K examples are available, 60K for training/validation and 10K for testing. The images size is 64x64.

C.2.2 Models performance on the classification task

I report the accuracy performance of these classifiers on the training and test data in table 1. It can be seen that the models seems to perform quite well on the task, thus, these models seem as good candidates to be oracles in order to train a generator.

[3]The classifier outcome is the highest value of the softmax for one of the outcomes. Thus, the objective of the generator is to generate an image that maximize the softmax value for the desired decision.

[4]In this case, I reduced the images size from 64x64 to 8x8, in order to reduce the computational time and memory needed.

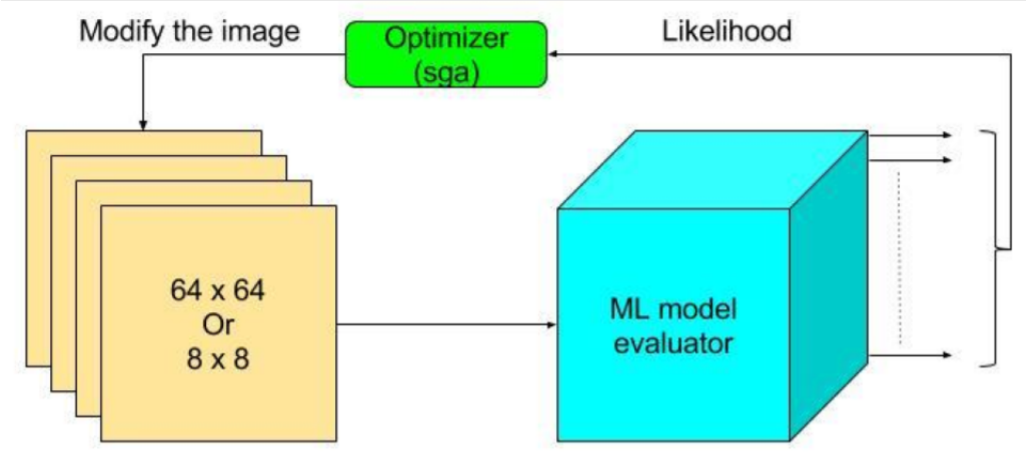


Figure C.6: The setup of the experiment to understand the effect of using an oracle as the guidance for training a generator. The oracles were trained on MNIST dataset in order to classify the digits, and the generator is optimized based on what the oracle output. The generator objective is to generate images for different digits.

Classifier	Train	Test
Logistic Regression	92.7	92
kNN	99	96.1
Neural networks	98.57	97.1

Table 1: The accuracy of the different classifiers used. The models perform well on the MNIST data. All the models have a satisfying performance.

C.2.3 Using the oracles to train the generator

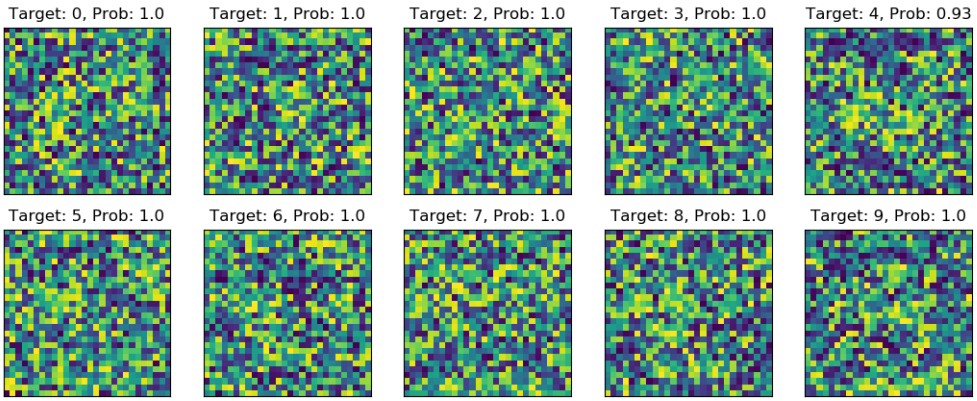
Now that the potential oracles are trained, I started using them to train generators. The objective is to generate an image that maximizes the relevant likelihood of the required digit in the oracle model (i.e., maximize the probability of this digits in the oracle). The results are in figure ??. We can see that all the oracles had been fooled quite easily by the generator. The oracles are show absolute confidence about the content of the images from the generator, while those images have no meaningful content or any resemblance to the ground truth digits. This here shows my original point, that the numbers from the oracle do not represent a trustworthy feedback in order to train a generator.

C.2.4 Diving into KNN

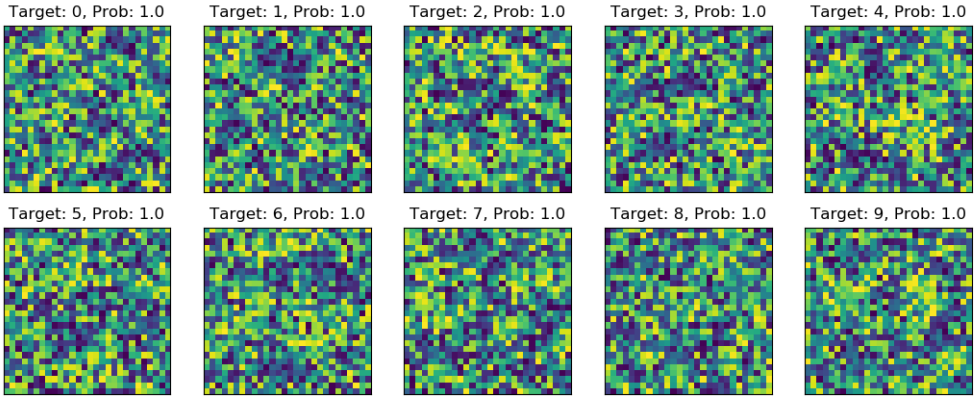
Given the previous results, I was curious if something can be done to avoid these results. I started to look at the KNN algorithm, since it is more easier to interpret. I analyzed the average distance of the test set and the adversarial examples to the training set. The results are shown in figure C.8. The adversarial examples have a much higher distance to the training data than the test data.

To understand what is happening, let's discuss it on case of only 2 categories, see figure C.9. What the generator is doing is that it tries to place the adversarial examples far from the relative examples. KNN classifies examples based on the nearest neighbors, aside from the distance. Thus, it is easily fooled.

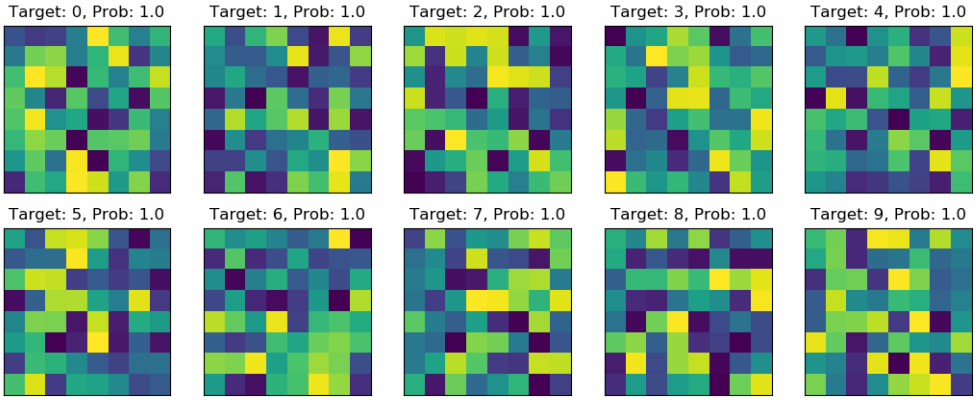
I tried to see if we can mitigate this point, by using this information to develop a new generator, with an optimization objective of both maximizing the confidence of the KNN likelihood, while in the same time adding a penalty on the distance to the relevant training data. The results are shown in figure C.10. We can see that we start to have more clear numbers. Except for digit 1, the other images are comprehensible and correct. However, repeating this experiment multiple times did not reveal changes in the generated images (see figure C.11). What I conclude is that the resulting images are reflecting the average/mean of the different images in for the required digits, thus, no diverse set of images for each digit are generated. What we would like to have in a generator training is to capture the data distribution, and not just zoom in on the mean value.



1 Logistic Regression



2 Multi-layer Perceptron



3 KNN

Figure C.7: Results of using different oracles in order to train a generator. Each generated image has the target digit and the final oracle confidence about it. Logistic regression, MLP and KNN were fooled very easily, with absolute confidence about the meaning of the different images.

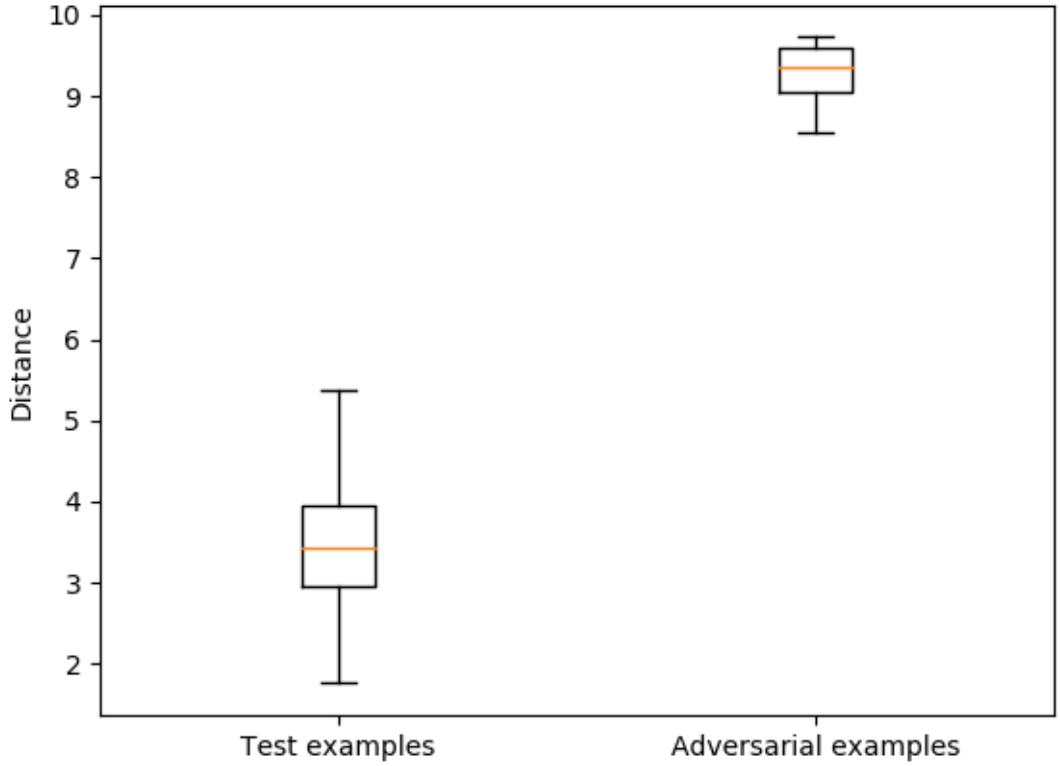


Figure C.8: KNN analysis: comparing the difference in the distance between the test data and the adversarial examples relative to the training data. We can see that the adversarial examples have a much higher distance than the test data.

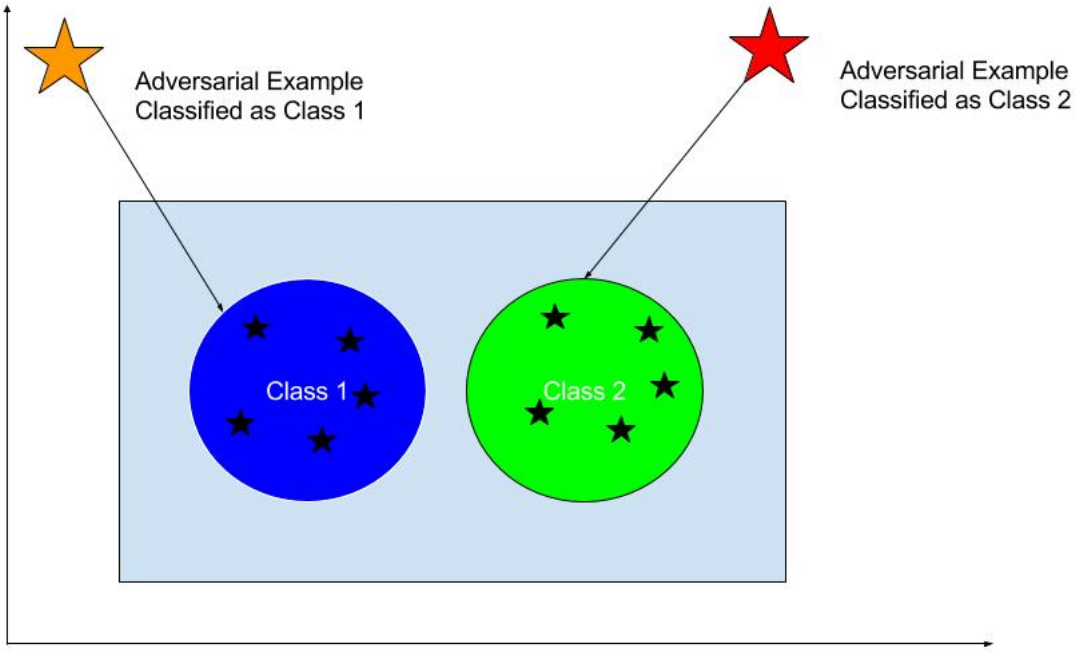


Figure C.9: How adversarial examples are developed to fool KNN: the generator simply tries to put the malicious examples as far as possible from the clusters. In the same time, the KNN algorithm just classifies examples based on the nearest neighbors, aside from the distance.

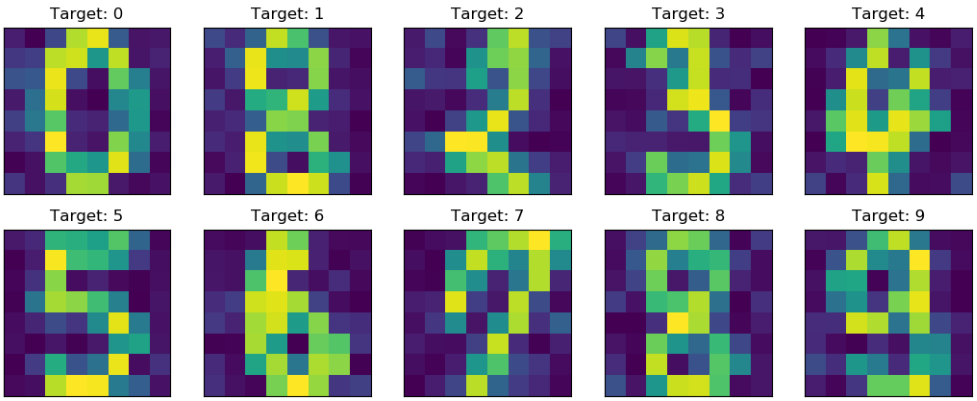


Figure C.10: KNN results after modifying the optimization objective: instead of only maximize the likelihood of the intended digit, I also added a penalty on the distance to the training data. The images are more relevant and comprehensible in this case.

C.3 What is the lesson learned?

Even if the model (oracle) is doing extremely well on the ground-truth distribution, it is should not be used naively as a replacement for the ground truth, in order to evaluate other models (like a generator). The generator can easily learn how to deceives the oracle in order to maximize the final performance numbers, without any consideration for what these numbers implies. I believe that this is an interesting direction in research though, with a lot of potential, by developing some safeguards to protect the model from being deceived, reporting the model confidence about the current input, or having some estimation for the model error once we change the distribution.

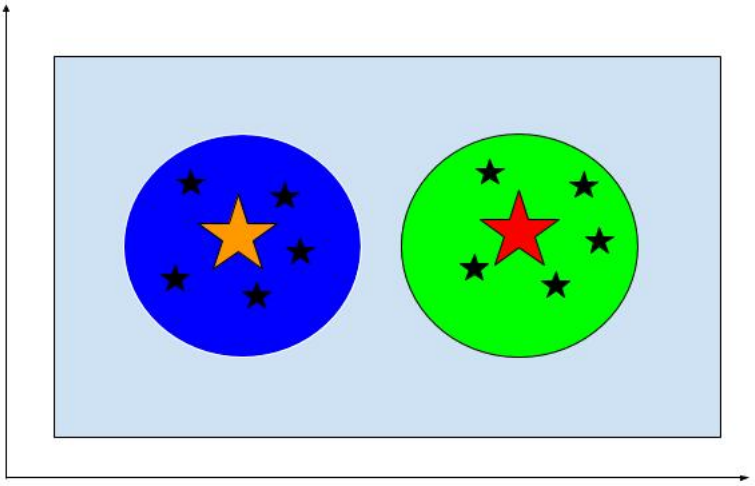


Figure C.11: Illustration for the performance of KNN after having the modified objective function (the likelihood of the oracle and the distance to the training data). While the results are quite good, the resulting images are reflecting the average/mean of the different images in for the required digits, thus, no diverse set of images for each digit are generated.

Bibliography

Nayyer Aafaq, Ajmal Mian, Wei Liu, Syed Zulqarnain Gilani, and Mubarak Shah. Video description: a survey of methods, datasets and evaluation metrics. *arXiv preprint arXiv:1806.00186*, 2018.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. 2015.

Gérard Bailly, Stephan Raidt, and Frédéric Elisei. Gaze, conversational agents and face-to-face communication. *Speech Communication*, 52(6):598–612, 2010.

Gérard Bailly, Alaeddine Mihoub, Christian Wolf, and Frédéric Elisei. Gaze and face-to-face interaction. In Geert Brône & Bert Oben, editor, *Eye-tracking in Interaction. Studies on the role of eye gaze indialogue*, pages 139 – 168. Benjamins, 2018. doi: 10.1075/ais.10.07bai. URL `https://hal.archives-ouvertes.fr/hal-01939223`.

Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*, volume 2. Siam, 2001.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

Christopher M Bishop. Mixture density networks. *Technical Report. Aston University, Birmingham*, 1994.

John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 conference on empirical methods in natural language processing*, pages 120–128. Association for Computational Linguistics, 2006.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Bo Chang, Qiong Zhang, Shenyi Pan, and Lili Meng. Generating handwritten chinese characters using cyclegan. *CoRR*, abs/1801.08624, 2018. URL `http://arxiv.org/abs/1801.08624`.

Rita Chattopadhyay, Qian Sun, Wei Fan, Ian Davidson, Sethuraman Panchanathan, and Jieping Ye. Multisource domain adaptation and its application to early detection of fatigue. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(4):18, 2012.

K. Chatzilygeroudis, A. Vassiliades, F. Stulp, S. Calinon, and J.-B. Mouret. A survey on policy search algorithms for learning robot controllers in a handful of trials. *arXiv:1807.02303*, pages 1–19, 2018.

Kyunghyun Cho. Boltzmann machines and denoising autoencoders for image denoising. *arXiv preprint arXiv:1301.3468*, 2013a.

Kyunghyun Cho. Simple sparsification improves sparse denoising autoencoders in denoising highly corrupted images. In *International Conference on Machine Learning*, pages 432–440, 2013b.

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. *arXiv preprint arXiv:1406.1078*, 2014.

François Chollet et al. Keras. `https://keras.io`, 2015.

Francois Chollet. *Deep Learning with Python*. Manning Publications, 2017. `https://www.manning.com/books/deep-learning-with-python`.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Nikhil Churamani, Paul Anton, Marc Brügger, Erik Fließwasser, Thomas Hummel, Julius Mayer, Waleed Mustafa, Hwei Geok Ng, Thi Linh Chi Nguyen, Quan Nguyen, et al. The impact of personalisation on human-robot interaction in learning scenarios. In *Proceedings of the 5th International Conference on Human Agent Interaction*, pages 171–180. ACM, 2017.

Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503, 2015.

Christopher De Sa. Non-convex optimization. 2017.

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.

Misha Denil, Loris Bazzani, Hugo Larochelle, and Nando de Freitas. Learning where to attend with deep architectures for image tracking. *Neural computation*, 24(8): 2151–2184, 2012.

Michael Denkowski and Alon Lavie. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the EACL 2014 Workshop on Statistical Machine Translation*, 2014.

Moises Diaz, Miguel A Ferrer, Antonio Parziale, and Angelo Marcelli. Recovering western on-line signatures from image-based specimens. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 1204–1209. IEEE, 2017.

Dominik Dörr, David Grabengiesser, and Frank Gauterin. Online driving style recognition using fuzzy logic. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1021–1026. IEEE, 2014.

Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

Itzhak Fogel and Dov Sagi. Gabor filters as texture discriminator. *Biological cybernetics*, 61(2):103–113, 1989.

Herbert Freeman. On the encoding of arbitrary geometric configurations. *IRE Transactions on Electronic Computers*, 2:260–268, 1961.

Peggy E Gallaher. Individual differences in nonverbal behavior: Dimensions of style. *Journal of personality and social psychology*, 63(1):133, 1992.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks. *CoRR*, abs/1505.07376, 2015. URL `http://arxiv.org/abs/1505.07376`.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org, 2017.

Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2017.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 513–520, 2011.

Lovedeep Gondara. Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 241–246. IEEE, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

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

Google. The quick, draw! dataset, 2017. URL `https://github.com/googlecreativelab/quickdraw-dataset#the-raw-moderated-dataset`.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

David Ha. Mixture density networks with tensorflow. *blog.otoro.net*, 2015. URL `http://blog.otoro.net/2015/11/24/mixture-density-networks-with-tensorflow/`.

David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Fritz Heider and Marianne Simmel. An experimental study of apparent behavior. *The American journal of psychology*, 57(2):243–259, 1944.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR*, 2(5):6, 2017.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. 2012.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Wei-Ning Hsu, Yu Zhang, Ron J Weiss, Heiga Zen, Yonghui Wu, Yuxuan Wang, Yuan Cao, Ye Jia, Zhifeng Chen, Jonathan Shen, et al. Hierarchical generative modeling for controllable speech synthesis. *arXiv preprint arXiv:1810.07217*, 2018.

Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4-5):411–430, 2000.

Anchit Jain. Improve your model accuracy by transfer learning. URL `https://medium.com/data-science-101/transfer-learning-57ce3b98650`. 2018-07-09.

Anil K Jain and Farshid Farrokhnia. Unsupervised texture segmentation using gabor filters. *Pattern recognition*, 24(12):1167–1186, 1991.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Derick A Johnson and Mohan M Trivedi. Driving style recognition using a smartphone as a sensor platform. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1609–1615. IEEE, 2011.

Ian Jolliffe. Principal component analysis. In *International encyclopedia of statistical science*, pages 1094–1096. Springer, 2011.

Takashi Kawashima Jongmin Kim Nick Fox-Gieg with friends at Google Creative Lab Jonas Jongejan, Henry Rowley and Data Arts Team. The quick, draw! game, 2017. URL `https://quickdraw.withgoogle.com/`.

Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

Shir Kashi and Shelly Levy-Tzedek. Smooth leader or sharp follower? playing the mirror game with a robot. *Restorative neurology and neuroscience*, 36(2):147–159, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Dietrich Klakow and Jochen Peters. Testing the correlation of word error rate and perplexity. *Speech Communication*, 38(1-2):19–28, 2002.

George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3):360–375, 2012a.

George Konidaris, Ilya Scheidwasser, and Andrew G. Barto. Transfer in reinforcement learning via shared features. *J. Mach. Learn. Res.*, 13:1333–1371, May 2012b. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2188385.2343689.

Simeon Kostadinov. How recurrent neural networks work, 2017. URL https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7.

Klaus Krippendorff. Computing krippendorff's alpha-reliability. 2011.

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

Hugo Larochelle and Geoffrey E Hinton. Learning to combine foveal glimpses with a third-order boltzmann machine. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1243–1251. Curran Associates, Inc., 2010. URL http://papers.nips.cc/paper/4089-learning-to-combine-foveal-glimpses-with-a-third-order-boltzmann-machine.pdf.

S. Lathuiliére, P. Mesejo, X. Alameda-Pineda, and R. Horaud. A comprehensive analysis of deep regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2019. ISSN 0162-8828. doi: 10.1109/TPAMI.2019.2910523.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Shoushan Li and Chengqing Zong. Multi-domain adaptation for sentiment classification: Using multiple classifier combining methods. In *2008 International Conference on Natural Language Processing and Knowledge Engineering*, pages 1–8. IEEE, 2008.

T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft COCO: Common Objects in Context. *ArXiv e-prints*, May 2014.

Mingsheng Long, Jianmin Wang, Guiguang Ding, Jiaguang Sun, and Philip S Yu. Transfer feature learning with joint distribution adaptation. In *Proceedings of the IEEE international conference on computer vision*, pages 2200–2207, 2013.

David G Lowe et al. Object recognition from local scale-invariant features. 1999.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Marielle Malfante. *Automatic classification of natural signals for environmental monitoring*. 2018.

Marielle Malfante, Mauro Dalla Mura, Jerome I Mars, and Cedric Gervaise. Automatic fish sounds classification. *The Journal of the Acoustical Society of America*, 139(4): 2115–2116, 2016.

Marielle Malfante, Mauro Dalla Mura, Jean-Philippe Métaxian, Jerome I Mars, Orlando Macedo, and Adolfo Inza. Machine learning for volcano-seismic signals: Challenges and perspectives. *IEEE Signal Processing Magazine*, 35(2):20–30, 2018a.

Marielle Malfante, Omar Mohammed, Cedric Gervaise, Mauro Dalla Mura, and Jérôme I Mars. Use of deep features for the automatic classification of fish sounds. In *2018 OCEANS-MTS/IEEE Kobe Techno-Oceans (OTO)*, pages 1–5. IEEE, 2018b.

Vincenzo Manzoni, Andrea Corti, Pietro De Luca, and Sergio M Savaresi. Driving style estimation via inertial measurements. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 777–782. IEEE, 2010.

U-V Marti and Horst Bunke. A full english sentence database for off-line handwriting recognition. In *Document Analysis and Recognition, 1999. ICDAR'99. Proceedings of the Fifth International Conference on*, pages 705–708. IEEE, 1999.

Clara Marina Martinez, Mira Heucke, Fei-Yue Wang, Bo Gao, and Dongpu Cao. Driving style recognition for intelligent vehicle control and advanced driver assistance: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 19(3):666–676, 2017.

Alaeddine Mihoub, Gérard Bailly, Christian Wolf, and Frédéric Elisei. Graphical models for social behavior modeling in face-to face interaction. *Pattern Recognition Letters*, 74:82–89, 2016.

Niyas Mohammed. How to autoencode your pokémon. URL `http://hackernoon.storage.googleapis.com/how-to-autoencode-your-pok%C3%A9mon-6b0f5c7b7d97`. 15-04-2017.

Omar Mohammed, Gerard Bailly, and Damien Pellier. Handwriting styles: benchmarks and evaluation metrics. In *2018 Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 159–166. IEEE, 2018.

Omar Mohammed., Gérard Bailly., and Damien Pellier. Transfer and extraction of the style of handwritten letters using deep learning. In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,*, pages 677–684. INSTICC, SciTePress, 2019. ISBN 978-989-758-350-6. doi: 10.5220/0007388606770684.

Christoph Molnar. *Interpretable Machine Learning*. 2019. https://christophm.github.io/interpretable-ml-book/.

Michael C Mozer. A focused backpropagation algorithm for temporal. *Backpropagation: Theory, architectures, and applications*, 137, 1995.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029.

Shreya Narang and Ms Divya Gupta. Speech feature extraction techniques: a review. 2015.

Jeremy Neubauer and Eric Wood. Accounting for the variation of driver aggression in the simulation of conventional and advanced vehicles. Technical report, National Renewable Energy Lab.(NREL), Golden, CO (United States), 2013.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436, 2015.

Duc Canh Nguyen, Gérard Bailly, and Frédéric Elisei. Pattern Recognition Letters Learning Off-line vs. On-line Models of Interactive Multimodal Behaviors with Recurrent Neural Networks. *Pattern Recognition Letters*, 100:29–36, December 2017. doi: 10.1016/j.patrec.2017.09.033. URL https://hal.archives-ouvertes.fr/hal-01609535.

SynSIG Special Interest Group of ISCA (the International Speech Communication Association). Blizzard challenge 2019, 2019. URL https://www.synsig.org/index.php/Blizzard_Challenge_2019.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016a.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016b.

Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

Sinno Jialin Pan, Xiaochuan Ni, Jian-Tao Sun, Qiang Yang, and Zheng Chen. Cross-domain sentiment classification via spectral feature alignment. In *Proceedings of the 19th international conference on World wide web*, pages 751–760. ACM, 2010a.

Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2): 199–210, 2010b.

Nicolas Papernot. Security and privacy in machine learning. Tutorial at IEEE WIFS, Rennes, France, 2017. URL `https://project.inria.fr/wifs2017/files/2017/12/WIFS_T2_Papernot.pdf`.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

Wei Ping, Kainan Peng, Andrew Gibiansky, Sercan O Arik, Ajay Kannan, Sharan Narang, Jonathan Raiman, and John Miller. Deep voice 3: Scaling text-to-speech with convolutional sequence learning. *arXiv preprint arXiv:1710.07654*, 2017.

Rajat Raina, Anand Madhavan, and Andrew Y Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*, pages 873–880. ACM, 2009.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *CoRR*, abs/1511.06732, 2015. URL `http://arxiv.org/abs/1511.06732`.

Manassés Ribeiro, André Eugênio Lazzaretti, and Heitor Silvério Lopes. A study of deep convolutional auto-encoders for anomaly detection in videos. *Pattern Recognition Letters*, 105:13–22, 2018.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

A. J. Robinson and Frank Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Engineering Department, Cambridge University, Cambridge, UK, 1987.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, page 4. ACM, 2014.

Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

Laurence Séraphin-Thibon, Silvain Gerber, and Sonia Kandel. Analyzing variability in upper-case letter production in adults. In *Spelling and Writing Words*, pages 163–178. BRILL, 2019.

Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

R. J. Skerry-Ryan, Eric Battenberg, Ying Xiao, Yuxuan Wang, Daisy Stanton, Joel Shor, Ron J. Weiss, Rob Clark, and Rif A. Saurous. Towards end-to-end prosody transfer for expressive speech synthesis with tacotron. *CoRR*, abs/1803.09047, 2018. URL http://arxiv.org/abs/1803.09047.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, Cambridge, MA, USA, 2014a. MIT Press. URL http://dl.acm.org/citation.cfm?id=2969033.2969173.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014b.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Makoto Tachibana, Junichi Yamagishi, Koji Onishi, Takashi Masuko, and Takao Kobayashi. Hmm-based speech synthesis with various speaking styles using model interpolation. In *Speech Prosody 2004, International Conference*, 2004.

Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. *ArXiv*, abs/1808.01974, 2018.

Matthew E Taylor and Peter Stone. Cross-domain transfer for reinforcement learning. In *Proceedings of the 24th international conference on Machine learning*, pages 879–886. ACM, 2007.

Paul Taylor. *Text-to-speech synthesis*. Cambridge university press, 2009.

Kristinn R Thórisson. Natural turn-taking needs no manual: Computational theory and model, from perception to action. In *Multimodality in language and speech systems*, pages 173–207. Springer, 2002.

Aaron van den Oord, Oriol Vinyals, et al. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pages 6306–6315, 2017.

L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-sne. 2008.

C. Viard-Gaudin, P. M. Lallican, S. Knerr, and P. Binter. The ireste on/off (ironoff) dual handwriting database. In *Document Analysis and Recognition, 1999. ICDAR '99. Proceedings of the Fifth International Conference on*, pages 455–458, Sep 1999. doi: 10.1109/ICDAR.1999.791823.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. *CoRR*, abs/1411.4555, 2014. URL `http://arxiv.org/abs/1411.4555`.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 3156–3164. IEEE, 2015.

Kiri Wagstaff. Machine learning that matters. *arXiv preprint arXiv:1206.4656*, 2012.

Wenfu Wang, Shuang Xu, and Bo Xu. Gating recurrent mixture density networks for acoustic modeling in statistical parametric speech synthesis. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5520–5524. IEEE, 2016.

Xin Wang, Shinji Takaki, and Junichi Yamagishi. An autoregressive recurrent mixture density network for parametric speech synthesis. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4895–4899, 2017a.

Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, et al. Tacotron: Towards end-to-end speech synthesis. *arXiv preprint arXiv:1703.10135*, 2017b.

Yuxuan Wang, Daisy Stanton, Yu Zhang, RJ Skerry-Ryan, Eric Battenberg, Joel Shor, Ying Xiao, Fei Ren, Ye Jia, and Rif A Saurous. Style tokens: Unsupervised style modeling, control and transfer in end-to-end speech synthesis. *arXiv preprint arXiv:1803.09017*, 2018.

Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, 2016.

Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356, 1988.

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.*, 1(2):270–280, June 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.2.270. URL http://dx.doi.org/10.1162/neco.1989.1.2.270.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015a.

Li Xu, Jie Hu, Hong Jiang, and Wuqiang Meng. Establishing style-oriented driver models by imitating human driving behaviors. *IEEE Transactions on Intelligent Transportation Systems*, 16(5):2522–2530, 2015b.

Yong Xu, Jun Du, Li-Rong Dai, and Chin-Hui Lee. Cross-language transfer learning for deep neural network based speech enhancement. In *The 9th International Symposium on Chinese Spoken Language Processing*, pages 336–340. IEEE, 2014.

Heiga Zen and Andrew Senior. Deep mixture density networks for acoustic modeling in statistical parametric speech synthesis. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 3844–3848. IEEE, 2014.