

**Learning a metric embedding of hand poses with
Siamese networks for low-shot learning in
fingerspelling recognition**

Kirill Ignatiev, B.Sc.

A Dissertation

Presented to the University of Dublin, Trinity College
in partial fulfilment of the requirements for the degree of

**Master of Science in Computer Science (Augmented and
Virtual Reality)**

Supervisor: Gerard Lacey

August 2019

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Kirill Ignatiev

August 13, 2019

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Kirill Ignatiev

August 13, 2019

Acknowledgments

I would like to thank my parents for their support.

KIRILL IGNATIEV

*University of Dublin, Trinity College
August 2019*

Learning a metric embedding of hand poses with Siamese networks for low-shot learning in fingerspelling recognition

Kirill Ignatiev, Master of Science in Computer Science
University of Dublin, Trinity College, 2019

Supervisor: Gerard Lacey

We investigate a method for signer-independent fingerspelling recognition based on learning a metric embedding of hand poses using siamese networks and using the embedding to construct a low-shot classifier. We present an efficient neural network for 2D hand pose estimation and show how to use transfer learning to learn an embedding of hand poses on a much smaller fingerspelling dataset than would be necessary to train the full model from scratch. We find that our method successfully learns a metric embedding of hand poses, but only reaches comparable levels of accuracy to other methods for cross-signer fingerspelling recognition reported in the literature.

Summary

Fingerspelling is a subset of sign language in which most gestures are single-hand static gestures. It has been shown that existing methods overfit to the particular signers performing the gestures, resulting in same-signer accuracy rates of 90–95%, compared to cross-signer accuracy of 40–45%, a substantial performance gap.

Following an analogy from natural language processing, we propose to model this problem as one of low-shot learning by learning a metric embedding of hand poses. Each new previously-unseen signer is assumed to have a small number of labelled gestures available for each letter, and the model learns to compare this signer’s new gestures to the small number of samples from the same signer, without direct comparison to gestures in the training data.

We show how to train an efficient neural network model for 2D hand pose estimation, and use transfer learning to enable the model to learn from the much smaller fingerspelling datasets. The disparity in dataset size and quality renders training the whole model from scratch infeasible, and this approach solves this problem.

We present the results on one standard dataset, KMNIST, and two fingerspelling datasets. We find that the model can perform low-shot learning on KMNIST at rates comparable to classifiers trained on the whole datasets even on characters not present during training. We find due that the limited size and variability of images of fingerspelling datasets, the model only achieves rates of accuracy comparable with those reported in the literature for other methods, around 45%.

Contents

Acknowledgments	iii
Abstract	iv
Summary	v
List of Tables	viii
List of Figures	ix
Chapter 1 Introduction	1
1.1 Motivation	1
1.2 Method	3
Chapter 2 Background and Related Work	8
2.1 Fingerspelling	8
2.2 Siamese Networks and Low-Shot Learning	13
2.2.1 Unsupervised Learning Approaches	13
2.2.2 Generating New Training Data	15
2.2.3 Siamese Networks and Low-Shot Learning	17
2.3 Hand Pose Estimation with Neural Networks	18
Chapter 3 Implementation and Results	24
3.1 Faster 2D Hand Pose Estimation	25
3.1.1 Network Architecture	25
3.1.2 Efficient Training	28

3.1.3	Comparisons and Results	32
3.2	Learning Embeddings on KMNIST using Siamese Networks	33
3.2.1	Learning a Siamese Embedding for the KMNIST dataset	36
3.2.2	Results	43
3.3	Cross-Signer Fingerspelling Recognition	53
3.3.1	The RWTH Fingerspelling Dataset	55
3.3.2	The ASL Fingerspelling Dataset	57
Chapter 4 Conclusions and Future Work		65
Bibliography		67

List of Tables

List of Figures

2.1	American Sign Language alphabet, reproduced from [1]	10
2.2	Architecture used by [2].	22
3.1	Hand pose heatmap regression network for 21 heatmaps, one per keypoint.	27
3.2	Test error convergence for hand keypoint heatmap regression, l_2 pixel distance. The model was refined further after this run.	31
3.3	GANerated hand pose samples	32
3.4	Comparison of performance for 2D hand pose estimation. Keypoint error is measured in pixels in a 256-pixel image. Data: [3, 2, 4] and our own work. Keypoint error numbers are re-computed using the data in the cited papers, to make them directly comparable as average l_2 errors measured in pixels.	33
3.5	Heatmap regression results, highlighted in red, for the downsampled GANerated dataset (64 pixels), <i>best seen in colour</i> . Highlighted are the base of the wrist the tip of the fingers (across columns).	34
3.6	A random selection of samples from the KMNIST dataset. Note the amount of variability between characters in the same class.	35
3.7	The basic classifier architecture, the convolutional block above the line, the classification block below.	37
3.8	The inverted residual block [5], shown with expand ratio 4, with c input channels and c' output channels.	37
3.9	Siamese embedding network, f is the number of features, typically $f = 10$.	41
3.10	Multi-headed network architecture	41
3.11	Linear dual-path layer, which creates Δc new channels.	41

3.12 Confusion matrix for the learned embedding on the 9 unseen characters. Using 10 prototypes per class.	44
3.13 Confusion matrix for the learned embedding on the 9 unseen characters. Using 25 prototypes per class.	44
3.14 Confusion matrix for the learned embedding on all the seen and unseen characters together. Using 25 prototypes per class.	45
3.15 Confusion matrix for the learned embedding on the 9 unseen characters. Using 25 prototypes per class.	46
3.16 Confusion matrix for the learned embedding on all the seen and unseen characters together. Using 25 prototypes per class.	47
3.17 Kernel PCA applied to the cosine similarity embedding	49
3.18 Pairwise similarity gradients for 10 randomly-selected characters. <i>Best seen in colour pdf.</i>	52
3.19 Siamese embedding network architecture for fingerspelling, using f fea- tures in the embedding.	54
3.20 Samples from the RWTH dataset [6]	55
3.21 Non-convergence of the siamese embedding on the RWTH dataset, test loss shown.	57
3.22 Samples from the ASL dataset [7]	58
3.23 Training loss convergence for the siamese embedding ASL dataset. . . .	59
3.24 Test accuracy convergence for the siamese embedding ASL dataset. . .	59
3.25 Accuracies for cross-signer and non-cross-signer classification	59
3.26 Effect of varying the number of embedding features on the classification accuracy.	60
3.27 Confusion matrix for the same-signer split of [7].	61
3.28 Confusion matrix for the learned embedding on the left-out signer from the ASL dataset.	62
3.29 Top confused pairs (above 10% error rate) by the siamese embedding classifier. Symbols reproduced from [8] (public domain).	63

Chapter 1

Introduction

1.1 Motivation

The primary motivation for this work is recognition of sign language and other complex hand gestures in unlabelled videos online. In this work we focus on a subset of this problem, recognition of static hand gestures within individual frames. Computer sign language recognition more generally is extremely important for developing computer interaction methods for sign language speakers [9, 10], which are a large group of users of up to 600,000 members not counting those who speak sign language as a second language [11].

Both action recognition [12, 13, 14, 15] and gesture recognition [16, 17] are important problems in computer vision. The primary challenge is to take a video and correctly identify the particular type of interaction between the subject and the environment. We are motivated by two primary examples of this, fingerspelling and surgical knot tying.

In fingerspelling, the range of interactions is much more limited. Typically only a single hand performs the gestures, and the subject needs not convey any extra information through facial expression, body pose, the non-signing hand, or arm motion. By contrast in action recognition, a good model is required to learn to recognize all aspects of the action.

This renders fingerspelling a harder, distinct problem. The details in the input image that the model must respond to are necessary small. Indeed, in a typical fin-

gerspelling video sequence the signing hand occupies only a small part of the frame, compared for example to the subject’s face. The gestures, seen in Figure 2.1, are often distinguished not by the general orientation of the arm or hand, as in action or gesture recognition, but the much smaller differences in finger joint angles. Thus, for example, a “B” and an “F” are quite similar, with the hand straight, distinguished by the position of the thumb and whether the index finger is touching the thumb. As another example, the main difference between a “7” and “8”, is whether the tip of the thumb is touching the tip of the fourth or the middle finger. This means a computer vision method for fingerspelling recognition is required to deal with the whole range of hand poses. As we discuss in Section 2.3, the hand poses obtained from fingerspelling are quite distinct from the hand poses obtained, for example, from object manipulation. This is a limitation of some published datasets of hand poses for hand pose estimation, if the protocol they were collected with was restricted to one or the other. Both choices are reasonable, but they result in computer vision methods that do not generalize between different sets of hand poses.

We reiterate that the general problem of sign language recognition is one of recognizing dynamic sequences of frames, in which both hands may be signing hands, and in which facial expressions, body pose, and all finger positions contribute to the particular gesture. In this work we focus only on the problem of recognizing static fingerspelling gestures from static images extracted from such videos. As we discuss later, a key difference is the availability of labelled training data. For fingerspelling there exist relatively small datasets in which each image is labelled with the letter the gesture in the image corresponds to. For sign language more generally, but also for in-the-wild datasets [18], the labels are typically sequences letter applied to the whole video sequence. This requires the method to not only correctly model hand poses, but also include a sequence-to-sequence model for language translation, the two languages being American Sign Language and American English.

A second motivating problem in gesture recognition comes from a separate domain, surgical knot tying. Knot tying is a basic clinical skill taught to surgical students through direct human supervision [19]. Unlike with sign language, where computer recognition of sign language would enable the development rich user interface for sign language speakers as well as sign language to text translation services, computer recognition knot tying would enable the development of a machine supervision system allow-

ing surgical students to learn this particular basic skill more efficiently. The traditional teaching method relies strongly on human supervision primarily because of the potential cost of mistakes [20, 21]. This process enables the teacher to directly intervene in the learning process, correcting mistakes before they become habits. Research into improving the efficiency of this teaching process primarily focuses on ensuring the supervisors’ interventions are effective at preventing future mistakes [22]. Such one-on-one supervision is nevertheless highly time-consuming for the supervisors, and we consider what would be necessary to automate this process.

Similar to sign language and fingerspelling recognition, knot tying results in fine finger motion. Unlike in sign language, knot tying further involves the external environment, specifically the suture and the body tissues. The hand poses involved in knot tying are of a similar complexity to the fingerspelling alphabet [19], but they can only be judged as correct or incorrect with respect to their relative position to the suture and the tissues. Additional tools have been proposed to make this problem more tractable, using multi-view cameras [23], depth sensors [24], and color marker gloves [25]. This tools, however, are not available in the large amounts of unlabelled data online, and require additional setup and equipment. What is needed is a system for analyzing knot tying performance using RGB video only.

Knot tying, unlike fingerspelling, lacks labelled datasets, either synthetic or collected from real-world videos. In this work we restrict our attention to static hand pose recognition from still images. Comparing the results of hand pose estimation on unoccluded hands [3] with the results on hands handling diverse objects [26], we can see that even manipulating large, well-defined objects causes current methods based on deep learning to fail. Prior to the advances in deep learning, work on hand pose estimation was focused on only partial subproblems, such as detecting hand bounding boxes in an image [27], or estimating full-hand gestures in 2D [17]. Whole-hand gesture recognition in particular does not require as much accuracy from the hand pose estimator as what would be required to accurately recover the finger joint positions.

1.2 Method

To motivate our method we begin with an analogy from natural language processing [28]. It is a common approach to view a sentence as a sequence of words $w_1 \cdots w_n$,

and ask for the probability that two words, w_i, w_j will occur together, $p(w_i, w_j)$. This probability can be computed directly for the most common from the large English corpora that are freely available such as the Penn Tree Bank [29]. Thus we can estimate the empirical distribution $p(w_i, w_j)$ as the fraction of times that the pair $w_i w_j$ occurs directly in the corpus. To be precise, if e_w is the one-hot coding vector corresponding to the word w , the matrix $p_{ij} = p(w_i, w_j)$ is computed as

$$\sum_i e_{w_i} e_{w_{i+1}}^\top, \quad (1.1)$$

suitably normalized. This approach is fundamentally unsatisfactory for two reasons. First, such an explicit representation consumes a lot of memory. Second, only *common* words can have their pairwise occurrence probabilities estimated this way accurately. Uncommon *related* words will not have any information about them shared with similar common words, and the semantic meaning of the words is partially lost in this procedure. In the analogy with hand poses, this is equivalent to representing a hand pose through 63 coordinates of the keypoints in 3D.

The *word2vec* model assigns a d -dimensional vector $v : w \mapsto \mathbb{R}^d$ to each word w , and models the probability of two words appearing next to each other as

$$p(w_i, w_j) = \sigma(v(w_i) \cdot v(w_j)), \quad (1.2)$$

where $\sigma(x) = (1 + e^{-x})^{-1}$ is the sigmoid function. Words that are commonly seen together in text receive similar embeddings $v(w)$, with large cosine similarity. The model is trained by iteratively updating the embedding vectors for every word to ensure that words that occur close together in text receive similar embedding vectors. An uncommon word w_i to which the model assigns a similar embedding to the embedding of a common word w_j shares parts of the embedding with w_j , and places in text where pairs with w_j appear frequently tell the model that w_i may appear there also.

This may be thought of as a kind of a compression scheme, in which the full $\{0, 1\}^N$ N -dimensional space of one-hot encodings of the full dictionary of N words is compressed into the unit sphere of \mathbb{R}^d . This has become a standard technique in natural language processing [28].

Hand poses, which play such an important part in sign language and fingerspelling

have no such obvious representation. Hand pose datasets traditionally annotate each hand with 63 coordinates, the 3D coordinates of each joint, relative to an abstract coordinate system attached to the camera. This particular embedding is the equivalent of the one-hot coding above: it is precise, but loses important semantic information. A human looking at a hand would notice whether the thumb is pressed against the base of the fifth finger, for example, but this is a very small portion of the overall configuration space of hand poses. Indeed, even large synthetic datasets such as [3] do not include such *extreme* poses, and focus mainly on sampling the joint angles uniformly at random. Other datasets such as [30] focus on hand poses that occur during object manipulation, and miss hand poses that may only occur elsewhere. By analogy with the word2vec model, what is needed is a hand2vec model, a learned embedding of hand poses into a low-dimensional space that discards the fine details of keypoint locations but instead preserves the important semantic information present in sign language gestures.

This embedding cannot be learned directly from unlabelled video. Recall that the word2vec model only requires unannotated texts, it does not require the words to be labelled. This same approach is infeasible with hand poses in sign language recognition because within a single fingerspelling sequence there may only be a small number of frames with the specific letter gestures, but there will be many frames of transitioning between the different hand poses. Without labelling the frames to establish which hand poses are the salient ones that actually carry information, it is not possible to learn a hand pose embedding using only unlabelled sequential information from a video.

In this work we propose an alternative method to learn such an embedding, and investigate whether it allows us to construct a signer-independent classifier of fingerspelling gestures.

In the context of signer-independent fingerspelling recognition, we propose to model the problem as one of low-shot learning. Suppose that we know such an embedding of hand poses similar in spirit to word2vec, and $v(x_1) \cdot v(x_2)$ is a cosine similarity measure of how close the hand poses are between images x_1 and x_2 . As discussed in Section 2.1, a key challenge of signer-independent fingerspelling recognition is that the models overfit to the particular signers present in the dataset. Expressing this more precisely in terms of the embedding, if $x_{i,c}$ is the i -th example of the letter c in the

training dataset, the classifier computes for an input image x

$$\arg \max_c \frac{1}{N_c} \sum_{i=1}^{N_c} \sigma(v(x) \cdot v(x_{i,c})), \quad (1.3)$$

or a similar softmax function of similarities as in [31].

If the embedding correctly learns to compress and distribute hand poses across the unit sphere in \mathbb{R}^d , the idea proposed in this work is that we can try to learn a better classifier by using

$$\arg \max_c \sum_j \sigma(v(x) \cdot v(\hat{x}_{j,c})), \quad (1.4)$$

where $\hat{x}_{j,c}$ are a small number S of images for letter c collected from the same signer as the input image x . If S were large, this would be little different from constructing a new classifier on the signer of x . Using a low-shot approach, S is kept small, typically 10, and the classifier can reuse the embedding v learned on the much larger training dataset, and only adapt itself to the new signer by selecting a small representative number S of hand poses from the new signer.

The approach belongs to the wider transfer learning framework of trying to take a model that was trained on a large general dataset and reuse it on a smaller less-varied dataset [32]. We may think of the training dataset as the one that should capture enough variability in hand poses to let the model learn an embedding, and we can think of the 10 images per letter per new signer as a very small dataset that should be representative of the new signer and therefore reduce the variance due to inter-signer variability when the model is tested on this single signer. This assumption appears to some extent to be self-contradictory. On the one hand to learn a meaningful embedding the model *must* have access to a lot of the variability to compress the entire configuration space of hand poses into a low-dimensional space. On the other hand, the training dataset is assumed not to have enough variability to capture the idiosyncratic features of the gestures of the previously unseen signer that the model will be tested on.

Existing work in image recognition and natural language processing has generally focused on using larger and larger datasets [33, 34]. We would still like to be able to learn from smaller datasets. Specifically in the field of hand pose estimation, sign

language recognition and its subset of fingerspelling recognition, the large datasets compared to those available in image recognition and machine translation are unavailable, and we would like to make use of the data we have. Assuming that the dataset is both large enough to learn an embedding and small enough that existing models overfit, we are ultimately making the assumption that the embedding-based method makes more efficient use of the limited amount of data available to it, and the success or failure of this method is largely about testing this assumption.

This approach makes the assumption that the number S of previously unseen samples from the new signer is sufficient to capture the idiosyncratic portion of their gestures, and therefore increase the classification accuracy beyond the low level of 40–45% reported in the literature on signer-independent fingerspelling. As we discuss in Chapter 4, we ultimately find that this approach generates appropriate results on a test dataset, specifically the 49-character KMNIST [35], but does not improve on signer-independent fingerspelling accuracy. While using 10 images per letter for the new signer provides enough information about per-signer variability to produce a classifier that matches existing results, our conclusion is that the embedding learned on the limited number of signers in the fingerspelling dataset is at least as specific to that dataset as typical classifier would be.

Chapter 2

Background and Related Work

In this chapter we present the background for the implementation and results in Chapter 3. In Section 2.1 we explain in detail the issues affecting sign language and fingerspelling recognition accuracy on cross-signer datasets. It is already established that an important problem in sign language and fingerspelling recognition is the issue of overfitting. Specifically, models tend to overfit to the particular dataset they are trained on, such as clean in-the-lab datasets versus low-quality in-the-wild datasets. We focus in particular on the divide between multi-signer and cross-signer model performance, and the divide between studio and in-the-wild datasets. In Section 2.2, we survey existing approaches to this general problem of overfitting to insufficiently varied data, covering low-shot learning [36], matching networks [31], and also known techniques for unsupervised deep learning. In Section 2.3, we focus on the recent progress made in the problem of hand pose estimation, assess how it can be used as a building block for a more general fingerspelling recognition system, and note the limitations of the current state of the art.

2.1 Fingerspelling

Sign languages are separate languages in which grammatically structured sentences consist of sequences of hand poses, gestures, and often facial expressions [37]. The number of American Sign Language (ASL) speakers is not generally known [11], partly because the speakers include both deaf people and people who speak ASL as a second

language. Being an important means of communication for many people, automatic recognition of sign language is an important problem in computer vision.

Sign language as a whole includes many elements, hand poses, arm movement, movements and gestures by *both* arms at the same time, both hands at the same time, facial expressions, and some body motion. Each of these elements, while being approximately described by the grammar of a sign language, is also partially unique to the speaker, resulting in a great deal of variability in the actual gestures made while speaking. Fundamentally, this is no different from automated speech recognition, but the primary challenge in sign language recognition is the complexity of the underlying data, with video and multiple hands and fingers, compared to the much simpler format of audio data.

The *American Sign Language* (ASL) fingerspelling alphabet [38], shown in Figure 2.1, is a set of gestures, each representing a letter. Of the gestures, only a few are *dynamic* gestures, and none involve both hands. As explained in Chapter 1, in the scope this work we exclusively focus on static gestures, with the ultimate motivation being to be able to extend this work to dynamic gestures also.

Fingerspelling datasets have generally been collected from two types of sources. Some datasets like [7, 6] are collected in lab conditions, asking volunteers to successively show each letter to the camera. The main reason for this particular approach is that it results in correctly labelled images, and each frame of a video is assigned a single correct letter, which is different from more general video datasets.

Depth information is often collected for sign language and fingerspelling datasets developed in lab conditions [7, 39, 40]. This is was very useful prior to the advances in deep learning that can handle hand pose estimation with only RGB data. Prior to this progress, RGB+D data was generally used even for hand pose estimation, and thus was largely necessary for fingerspelling recognition as well. Most available sign language video do not include depth information at all, and represent the most common source of spoken sign language. It is therefore desirable to focus specifically on RGB-only fingerspelling recognition, ignoring depth information even when it is available. Furthermore, as we discuss in Section 2.3, the primary accuracy issues in hand pose estimation with convolutional neural networks are related to self-occlusion, occlusion by objects, uncertainty about hand shape, and similar issues. Given the amount of complexity that is added by requiring depth data, and given that depth data does not

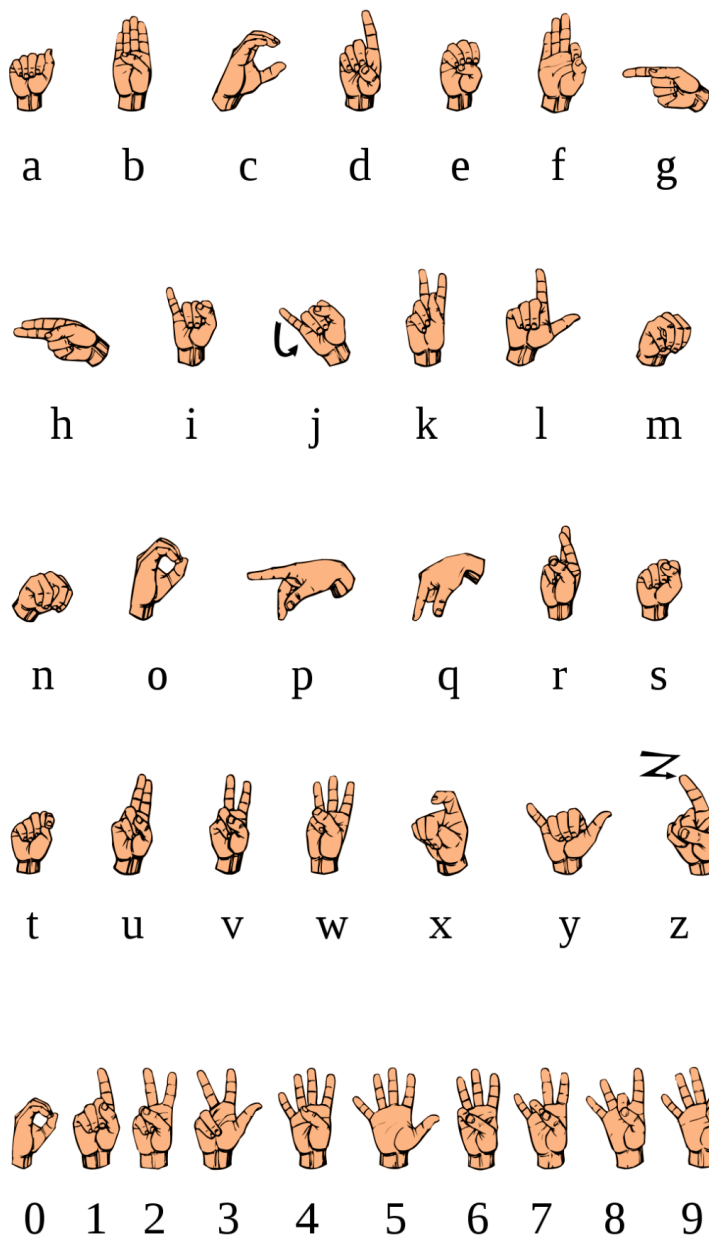


Figure 2.1: American Sign Language alphabet, reproduced from [1]

address the primary issues with modern hand pose estimation methods, focussing on RGB-only images is the correct approach.

Other fingerspelling datasets are reused from more general sign language datasets. For example, the RWTH-PHOENIX dataset [41] is a labelled set of videos, collected

from a news studio stream, with a limited set of 9 signers in total. The authors test their methods’ performance on a cross-signer split of their data. Without a cross-signer split, which means the training set includes the same signers as the signers in the test set, they report 66% accuracy, compared to only 46% accuracy when the model is tested on signers *not* within its training dataset. This behaviour is common to other dataset, as we discuss below, and represents a fundamental challenge for fingerspelling recognition: the models overfit strongly to the particular signers in the training dataset, and most datasets, especially studio datasets, have too few signers to properly guard the model against this. This limitation is common to other studio-based sign language datasets [42, 43, 44], and the reported accuracy is similarly low. For example, the authors of [44] report 67% top-20 accuracy on cross-signer sign language. Given the low signer-independent accuracies reported even on fingerspelling, it is correct to focus on fingerspelling with the current state of the art, rather than on the much more difficult problem of full sign language recognition.

The poor signer-independent results standard in contrast to same-signer results even on the harder problem of sign language recognition. For example, [45] report 99.3% accuracy on single-signer recognition with a restricted vocabulary of 232 signs, which reduces to 87.8% on multi-signer recognition, and further down to 44.1% on signer-independent recognition. Similarly, [46] report classification accuracy of 92% on same-signer recognition with a vocabulary of 132 signs.

The largest known fingerspelling dataset to date is the ChicagoFSWild dataset [18]. It was extracted from 214 videos, collected from YouTube, ASLized, and DeafVideo.tv. The videos cover a large number of different signers, and the fingerspelling sequences were extract and manually annotated. Thus results in 7.3k sequences, with a median number of 20 frames each. Each sequence is annotated with a word, usually, but may also contain spaces and word breaks. Because we focus on static fingerspelling recognition in this, and therefore require each frame to be labelled with a letter, we were unable to use this large in-the-wild dataset.

There are further limitations that we encountered using the ChicagoFSWild. Each frame is a still from the original video, and there are typically two of the signer’s hands visible in the frame. This creates two issues. First, we have to extract the hands’ bounding boxes, and second, we have to decide which hand is the signing hand for fingerspelling. In the original work [18], this was addressed by training the entire model

from the bottom up, based on convolutional neural networks (specifically AlexNet [47]) for hand bounding box detection and hand pose estimation, and long short-term memory recurrent networks for learning to translate between the features learned by the CNN and the actual letters. However, partly due the small size of each hand in the image, and partly due to the performance limitations of the hand pose detector we used [2], it is not currently possible to accurately automatically extract the signing hand from each frame. This would be necessary to apply an approach like ours, where we need the input image to contain the cropped hand, with a letter label attached to the image.

On [18] the authors reach 41.9% accuracy, which compares poorly to human-level accuracy of 82.7%. This figure is in line with previous published research on signer-independent fingerspelling recognition, and also signer-independent sign language recognition. In principle, this could be solved by collecting a large enough dataset, but the costs of this render this idea infeasible. This means that this problem with cross-signer performance is unlikely to be solved by using a larger, more complicated neural network architecture. Indeed, the architecture used in [18] is already quite complex, and it is required to solve multiple problems at once: (a) hand bounding box detection, (b) *signing* hand bounding box detection, (c) gesture recognition, (d) sequence-to-sequence translation.

It is plausible to imagine that a better approach may be to train each individual part separately. Indeed, while we found the performance of off-the-shelf hand bounding box detectors insufficient to convert the ChicagoFSWild dataset for our purposes, a better bounding box detector does not need to be trained specifically on the fingerspelling dataset. It can instead be trained on a more artificially diverse dataset of synthetic images such as [2, 3]. We suspect that the reason the specific methods used in [2, 3] were insufficient on this dataset was the disparity in image quality, which generally used large 256×256 pixel images, compared to a hand size of 32×32 that is more typical of low-quality YouTube video in ChicagoFSWild. In this work we use, mostly for other reasons, an image size of 64×64 and this is much closer to what is needed for accurately recognizing hands in fingerspelling videos in the wild.

Gesture recognition may also conceivably be trained separately from the fingerspelling dataset. Indeed, there are potentially two steps to solving this problem as we do it in this work. We train a hand pose estimation neural network, and then use

transfer learning to reuse the features for fingerspelling-specific gesture recognition. The main reason we do this is the quality and quantity of data. Hand pose estimation is a problem for which large synthetic datasets are available (Section 2.3), while the datasets for fingerspelling recognition are primarily small, use low-quality images, and come with only video labels rather than frame labels. Gesture recognition is therefore a second candidate for transfer learning, which has not yet been reported in the literature.

Sequence-to-sequence translation is the third candidate in this system suitable for transfer learning. In the field of machine translation it is already known that abstract representations of languages can be learned without having every sentence in one language be matched to the corresponding sentence in the other language [48]. This means that it may not be necessary to train a language model from scratch based on the fingerspelling data available, but it may be possible to reuse a model trained on the large available corpora of English.

2.2 Siamese Networks and Low-Shot Learning

In this section we review a number of different approaches to low-shot learning that may be suitable for signer-independent fingerspelling recognition. We may think of the problem as one of weakly-supervised learning. As discussed above, the labelled fingerspelling images are few, compared with the large amounts of unlabelled fingerspelling videos available in general. The labels also do come in a convenient format of one label per frame, restricting the kinds of models that can be applied to them. Some of the issues presented in the settings are similar to issues present in natural language processing more generally, being made only more complicated by the specific domain of sign language.

2.2.1 Unsupervised Learning Approaches

The first important class of approaches we would like to examine is through the use of deep unsupervised learning techniques. Unsupervised learning is an exciting topic in deep learning, primarily because unlike more traditional computer vision techniques, unsupervised deep learning techniques may be able to use the vast amounts of un-

labelled data readily available online. One fundamental limitation of unsupervised techniques when applied to sign language is that compared to image recognition tasks, sign language recognition relies much more on the smaller, more fragile portions of the input image.

In [49] it was shown that it is possible to learn an abstract representation of images in an unlabelled dataset by using a *jigsaw puzzle task*. Similar in spirit to a super-resolution task, a square fragment of the input image is cropped randomly, and is split into several squares like a jigsaw puzzle. The model is fed as input a scrambled configuration of the jigsaw pieces, and asked to predict the correct configuration of these pieces. What the network is required to do, in effect, is to learn which image pieces belong together in what order.

This process forces the model to learn an abstract representation of an input piece, in order to use it as input to the final fully-connected layer that outputs the predicted configuration. The purpose of the jigsaw puzzle task itself is to force this process to happen, rather like the choice of a loss function defines the task the neural network will perform.

It was already shown in [50], that in a typical convolutional neural network, for which the authors used AlexNet [47], only the final layers in the network perform most of the work of classification, whereas the first layers in the network are more general. The jigsaw puzzle task therefore forces the network to learn to perform well on an irrelevant task, but the initial layers of the network are then expected to be useful in general, and represent something meaningful about the images in the unlabelled dataset. The final accuracy using transfer learning for ImageNet classification is 34.6% by retraining only the fully connected layers.

In the context of fingerspelling recognition, it is not clear what this intermediate task should be. It is already a standard transfer learning technique [32] to do what we do in Chapter 3, and reuse the layers learned for hand pose estimation in a network for fingerspelling recognition. We do not want to conclude, however that this approach is entirely unsuitable, because of the breadth of possible hand pose-related intermediate tasks. Furthermore, as we emphasized in Section 1.2, learning an abstract representation of hand poses would be a valuable step forward.

An example of another approach to unsupervised learning is DeepCluster [51]. Since the authors' goal is to learn a clustering of ImageNet without access to labels, the

number of clusters is specified as hyperparameter for the model. By analogy with k -means, the model then learns an embedding of the input images into an abstract f -dimensional space, $f_\theta(x)$, minimizing the mean l_2 error of each cluster and updating cluster assignments as in the standard EM-algorithm [52], thus minimizing

$$\arg \min_C \frac{1}{N} \sum_n \min_{y_n} \|f_\theta(x_n) - Cy_n\|^2, \quad (2.1)$$

where $y_n \in \{0, 1\}^k$, $y_n^\top \mathbf{1} = 1$ are the cluster assignments. This results in a clustering that matches ImageNet labels 43% of the time.

The primary benefit of this approach, despite the complexity, is that it is independent of the labels in the dataset, and can in principle be trained on a separate dataset. In the context of fingerspelling, there is no equivalent concept of a cluster for hand poses.

2.2.2 Generating New Training Data

Another semi-supervised approach that was investigated in earlier work is based on the idea of generating new training data using generative adversarial networks. Let us first define a GAN. Consider a dataset X , which we think of as a probability distribution over all possible images, and a *discriminator* $D : x \mapsto [0, 1]$ that takes an input image and outputs the estimated probability density for x being sampled from X , $p_X(x)$. We have a dataset $\{x\}$ that contains only positive examples x , and if we were to train D with the obvious loss function

$$- \mathbb{E}_{x \sim X} \log D(x), \quad (2.2)$$

the network would learn that every input belongs to X , and learn the trivial probability density function $p_X(x) = 1$.

We now define a *generator* $G : z \mapsto X$, which takes as input random noise, and generates an image. We can now train G to produce images that D will recognize as belonging to X , which means minimizing

$$\mathbb{E}_{z \sim Z} \log(1 - D(G(z))) \quad (2.3)$$

We would like the discriminator in turn to learn to accept true images $x \sim X$, and reject fake images $G(z) \sim G(Z)$, minimizing the loss function

$$- \mathbb{E}_{x \sim X} \log D(x) - \mathbb{E}_{z \sim Z} \log(1 - D(G(z))). \quad (2.4)$$

It can be shown [32] that this procedure results in a discriminator that learns the true p.d.f. $D(x) = p_X(x)$ in equilibrium, and a generator that can generate images following the same distribution as X .

When carried out on ImageNet, or CelebA-HQ [53], this procedure results in a neural network G that can take random noise and sample from a probability distribution approximately equal to the distribution the training dataset was sampled from, generating images from ImageNet, or faces from CelebA-HQ. Further work on *conditional GANs* [54] allows us to train a function G_c for each known class c , and sample images from that particular class.

In the context of low-shot learning, the authors of both [55, 56] consider having a small number of examples of a particular class, and a much larger body of examples. They learn a particular generator $G(x, z)$ that takes as inputs an image and random noise and is able to sample from the distribution of images in the large dataset that are close to the input image x . The authors show evidence that this procedure can generate new training samples starting from a smaller number of available examples x .

An example of using GANs to learn meaningful abstract embeddings of images is the work of [57], where the authors train two generators G_{XY}, G_{YX} for transforming images between two domains X and Y , with a further siamese network S and approximately impose the constraint that mapping images with S to an abstract embedding space preserved magnitudes and orientation in that embedding space,

$$S(x_1) - S(x_2) = S(G_{XY}(x_1)) + S(G_{XY}(x_2)), \quad (2.5)$$

together with a hinge loss on S to ensure it does not learn the trivial function 0. The authors show that this is a powerful tool for learning semantic information about images, as the network S learns a semantic embedding of images and thus, for example, if x_1 and x_2 are two similar images that differ in style, the vector $S(x_2) - S(x_1)$ represents the operation of style transfer, and given a third image x_3 in the style of x_1 ,

an image with the embedding vector $S(x_3) + (S(x_2) - S(x_1))$ is the image x_3 transferred to the style of x_2 .

In the context of fingerspelling and hand pose recognition, the primary challenge to using GANs to represent probability distributions over the space of hand poses is the complexity of the hand pose space together with the fact that it is only indirectly captured by the rendered image of a hand pose. This means there is no known way to train GANs to learn a prior over hand poses, or over fingerspelling gestures, or to generate new hand pose images. Indeed, generating hand poses on its own is a trivial problem, as most hand poses are admissible except those that violate the joint angle constraints of the hand bones. A potential workaround for this challenge would be to use learn a prior over hand poses in unlabelled fingerspelling videos, but this approach is likely to be substantially harder than generating an equivalent labelled synthetic dataset.

2.2.3 Siamese Networks and Low-Shot Learning

Siamese network, introduced in [58], are a general term for using a model in which the same neural network appears twice with shared weights. As an illustrative example, consider the problem of classifying pairs of images of faces according to whether they represent the same person. Let $f_\theta : X \mapsto \mathbb{R}^d$ be a neural network that maps images in X to an abstract d -dimensional representation. Let $(x_1, y_1), (x_2, y_2)$ be two pairs of labelled images, and consider modelling the probability that $y_1 = y_2$ by

$$e^{-\|f(x_1) - f(x_2)\|^2}. \quad (2.6)$$

This results in a model which applies the same neural network to two different images, and outputs two d -dimensional vectors. The similarity $s = \|f(x_1) - f(x_2)\|^2$ between the embeddings $f(x_1), f(x_2)$ then estimates the probability that x_1, x_2 are images that belong to the same class. Note that in this process, we only require the label $y_{1,2}$, whether the images are in the same class, but the training procedure does not need the actual labels y_1, y_2 , and is therefore somewhat independent of the true number of classes.

In one-shot learning on the Omniglot dataset [59], siamese networks were one of the first successful approaches [36, 60]. The Omniglot dataset consists 15 to 40 hand-

written examples per letter collected from 50 alphabets. The one-shot learning task on Omniglot takes one image from each of C different letters, for example $C = 20$, and then asks the model to classify an unseen image as belonging to the same class as one of the C one-shot examples. In [36], this is done by selecting $d = 4096$, using a convolutional neural network for f_θ , and using the prediction

$$\sigma(\alpha^\top |f(x_1) - f(x_2)|), \quad (2.7)$$

where α is a vector of learnable parameters, and $|\cdot|$ represents elementwise absolute value. The model is then trained by using a binary cross entropy loss

$$l_{1,2} = y_{1,2} \log p_{1,2} + (1 - y_{1,2}) \log(1 - p_{1,2}), \quad (2.8)$$

for each presented positive or negative pair of images $(x_1, x_2, y_{1,2})$. The total number of classes, $|C| = 4,800$ is greater than the number of features d in the embedding vector, so the model in effect compresses the information available in the input image, unlike a classifier that would predict a probability distribution over $|C|$ different classes. The training procedure also imposes a metric structure on the learned embedding $f_\theta(x)$, in this case using the l_1 norm.

In the context of fingerspelling, and specifically with the analogy with natural language processing of Section 1.2 in mind, we are interested in learning a metric embedding of hand poses similar to the embedding learned in [36] for the images in the Omniglot dataset.

2.3 Hand Pose Estimation with Neural Networks

We now review the progress made in recent years on the problem of 2D and 3D hand pose estimation. Following the discussion in Chapter 1, we would in particular like to examine this progress on three main questions:

- its ability to successfully generalize to tasks beyond direct estimation of keypoint locations, such as more general gesture recognition,
- its ability to robustly handle a wide range of hands, which is strongly desirable for any application to fingerspelling,

- and its computational complexity, in other words its ability to be used as a building block for larger systems that work with RGB images of hands.

These are the question primarily because we aim to use a convolutional neural network for hand pose estimation as a building block for the problem of fingerspelling, and are therefore somewhat less interested in its performance on specific benchmarks.

The problem of hand pose estimation is similar to the problem to the extent that each image $x \in \mathbb{R}^{3 \times 224 \times 224}$ comes with 21 labels, ground-truth locations of the 21 keypoints $y \in \mathbb{R}^{2 \times 21}$ in the case of hand pose, and 16 keypoints in the case of body pose.

Much of the success in the field of body pose estimation was spurred by the MPII Human Pose benchmark [61], which provided a very large set of 25k diverse images, covering 40k human people, with annotations of 16 keypoint locations in 2D image coordinates (ankles, knees, hips, pelvis, thorax, neck, head, shoulder, elbow, wrist). The particular benefit of having such a large dataset is that it is sufficient to train a large convolutional neural network without overfitting, but such large datasets are also in some sense rare and expensive to collect. Other datasets, such as *HumanEva* [62], and *Human3.6M* [63] are captured in controlled environments. From the point of view of deep learning, this is undesirable because a neural network-based model can easily overfit to the particular style of images and the restricted number of settings. Indeed this particular effect later led to the creation of better synthetic hand pose datasets [3], as we will discuss below.

Due to the cost of correctly annotating a large dataset of body poses, a similar dataset of annotated hand poses, specifically one that relies on in-the-wild captured video with real people, does not yet exist. The challenges of collecting such a dataset are primarily two. First, manually annotating 2D and 3D keypoint locations in an image, although possible, is harder than in the case of body pose datasets. Unlike common body poses, which include relatively little self-occlusion, most interesting hand poses, even simple and common ones such as fists, ones that appear in the alphabet for American sign language (Figure 2.1), involve a great deal of self-occlusion. While a human can easily recognize the specific gesture, and perceive the relative positions of fingers and their joint positions, it is substantially harder, compared to synthetic images, to accurately label all occluded keypoints including completely occluded fingers. This problem is alleviated by constructing synthetic datasets in which all keypoint locations

are known a priori.

Second, extracting hand images from in-the-wild videos results in very low-quality images, similar to what can be seen in Figure 3.22, or in [18]. As reported in [18], the resulting images are often extremely small, and include large amounts of motion blur. In a typical video, the hands occupy only a very small part of the frame, and both the hands and the fingers move very quickly. While it is an important challenge to produce a robust model for estimating 3D hand pose from small motion-blurred images, this is not currently a solved problem yet [4], and most existing work uses large synthetic high-quality images of 224×224 pixels. To note, the image size we will use in this work, 64×64 results in lower accuracy, but is much closer to the typical size of a hand extract from a real video.

There have been published many new datasets for hand pose estimation since the advances in deep learning made 3D hand pose estimation feasible. As discussed in Section 1.1, we specifically look for datasets that use RGB images, as opposed to RGB+D images. The *Stereo Hand Pose Tracking Benchmark* [64] was published with 18k images, collected from a single person with six backgrounds and different lighting conditions. Once again, the limitations of this dataset with respect to its diversity, the diversity of hand shapes, hand poses, backgrounds and lighting, limit its usefulness for training deep neural networks. The *Dexter+Object* dataset [65] contains 3.1k images was also collected in a lab, and focused on images of hands manipulating a cube. It is also limited in the variety of hand poses, hand shapes, and backgrounds and lighting, but importantly it includes object manipulation, which means a successful model is required to deal with the hand being occluded by an object. The images came from a total of 6 videos, and all used the same object.

Object occlusion, in general, is an important theme in hand pose estimation research. As was pointed out in [26], which published a new dataset for hand pose estimation with object occlusion, increasing the number of videos to 6k, and the number of objects to 150, significantly decreases the performance of CNN-based methods that were trained either with no occlusion or with occlusion by the same small number of objects from a restricted set of objects. To be more precise, this increases the typical l_2 error of 2D keypoints from around 20px to above 50px. This means even the best published models suffers from substantial overfitting to the dataset that they were trained on, and we cannot determine whether, for example, a 10px l_2 error on

the test set leads to a similarly low generalization error on a different dataset. This is especially so with respect to hands occluded by objects.

There are yet other datasets, such as the *NYU Hand Pose Dataset* [66], which was published using registered color images, which makes it much harder to use for training CNN-based models. Furthermore, this dataset contains only 3 subjects, which further limits its utility.

One of the first successful works on 3D hand pose estimation was the work of [2], and it used a convolutional neural network. Unlike later work, which generally uses convolutional-deconvolutional architectures, as well as batch normalization as the primary regularization method, [2] used a simpler pure-convolutional neural network architecture shown in Figure 2.2. One implication of this architecture, discussed in Section 3.3, is that we find that using a large number of layers, here 512, results in overfitting when attempting to transfer the network’s learned features to other, much smaller datasets. To train their network, [2] created their own synthetic dataset using 20 freely available 3D models of humans, performing 39 actions, for a total of 41k images.

The authors further demonstrate that the model produces sufficiently accurate key-point location estimates, by training a 3-layer fully-connected neural network on the [6] that takes as inputs a vector of 3D keypoint locations $x \in \mathbb{R}^{3 \times 21}$ and produces per-class log-odds. This procedure reaches a similar level of classification accuracy, 66.8%, as reported in the original dataset [6].

One of the limitations of synthetic datasets such as [2] is the generally artificial style of images. In [3], the authors report their finding that neural networks overfit to the particular style of images. They introduce an alternative method of generating hand pose datasets, based on style transfer with generative adversarial networks. This dataset was published and is freely available, and this is the dataset we will use in Section 3.1. The authors of [3] train a style transfer network for *unpaired* images, transferring style between real hands captured with a camera and synthetic hands rendered with a model. They report better results with models trained on the dataset generated with GAN transfer. A key benefit of this approach is that it frees the authors from having to collect a wide variety of hand models for rendering, as the variety of hand colors and textures should be captured by the GAN style transfer model. The hand pose estimation network, based on ResNet-36, produces scoremaps, and has its output modified by the priors imposed on it by hand, specifically joint angle constraints,

Name	Channels	Size
(input)	3	256×256
$2 \times$ Conv+ReLU	64	256×256
MaxPool		128×128
$2 \times$ Conv+ReLU	128	128×128
MaxPool		64×64
$4 \times$ Conv+ReLU	256	64×64
MaxPool		32×32
$5 \times$ Conv+ReLU	512	32×32
Depth-wise dual-path layer	533	32×32
$5 \times$ Conv+ReLU	128	32×32
Depth-wise dual-path layer	554	32×32
$5 \times$ Conv+ReLU	128	32×32
Conv($k = 1$) (output scoremap y)	21	32×32

Figure 2.2: Architecture used by [2].

frame-to-frame (temporal) smoothness constraints, and known-skeleton fitting to the 3D keypoint estimates. This results in typical 3D keypoint errors of under 20mm, and median 2D keypoint errors of under 7px.

This still, however, results in models that overfit to the hand *shape* [4]. In [4], the authors use primarily the non-synthetic datasets mentioned above, together with the Panoptic dataset [67]. This is important for the question of which dataset features the neural networks will overfit to, as the totality of images is smaller and less varied than in the synthetic datasets such as [3]. The authors use a different dataset for investigating generalization errors, a combination of [61] and a New Zealand sign language dataset. While this renders their reported results not directly comparable to other works, their results point out an important issue. Specifically, their models are explicitly tested on *in-the-wild* images, rather than synthetic or studio images, and they report 2D keypoint errors 20px versus 60px errors for [2]. This in particular implies that while synthetic datasets have been extremely useful for training large CNN-based models, they also limit these models in how well they generalize to in-the-wild data.

A further clue to how we know what the models overfit to is the particular approach chosen in [4]. The *MANO* model [30] is a hand model collected in lab conditions from volumetric scans of volunteers who were going through a protocol of manipulating objects in prescribed ways. The model differs greatly from the presentation of hand

pose models we used above in several respects. First, it contains 15 joints (not 21). Second, it is based on 3D scans of volunteers’ hands, with a 3D mesh reconstructed from those scans. This allows the model to include *blendshapes* [68] of hand shape, which means a 20-dimensional vector of real numbers approximately describes each hand shape. This has a further effect that this model is capable of representing non-rigid hands. Hand pose, on the other hand, is represented using 9 PCA components of the hand poses observed in the lab conditions. This is problematic for the purposes of fingerspelling, as many shapes in the fingerspelling alphabet cannot be represented using the few chosen PCA components, which can be seen directly by visual comparison of Figure 2.1 and [30].

The authors of [4] then train a neural network to output the parameters of a MANO model, minimizing the reprojection error of the keypoints from the MANO model rendered with the parameters computed by the neural network. This rather roundabout method of estimating hand pose is successful in not overfitting to hand shape, but it is unclear whether it can successfully capture all the hand poses not represented explicitly by the MANO model.

Having summarized the key works in the recent progress on hand pose estimation, we would like to point that it is still not clear which features of hands in the available datasets the neural network models overfit to. The above can only be a partial list of mis-features of the synthetic datasets. Furthermore, computational considerations are not generally being considered in the design of these neural network architectures. It would be very useful to the research community to have a high-quality high-speed hand pose estimation model included in a common project like OpenCV [69]. The published models are either too inaccurate on in-the-wild images, or are too computationally intensive to be of use for building other real-time systems on top of them. The work of [4], for example, uses a ResNet-50 network, which cannot generally be made to run in real time except with substantial effort through the use of non-standard deep learning frameworks. An alternative, which has not been pursued in the literature to our knowledge, and which we do not pursue here, would be to create a rich parameterized hand model that includes both blendshapes for hand shape, and more richness of hand pose than is available in the MANO model, and use that to train a more efficient hand pose estimation model that can be more readily used in practice.

Chapter 3

Implementation and Results

We now describe the main contributions of this thesis, their implementation and results. In Section 3.1, we describe the building block, the convolutional neural network for recognizing hand keypoint locations in 2D, and show that it achieves results close to state-of-the-art at substantially higher framerates. We focus in particular on the architecture choices that allow the neural network to be fast at both training and inference, the choices in training the neural network that ultimately allow it to be trained in under 10 minutes on a machine with a single NVIDIA P100 GPU. We train the neural network on a scaled down version of a standard synthetic dataset, and show that the neural network achieves only marginally worse results on the same dataset, despite having access to $\frac{1}{16}$ of the pixels in the downsampled images, and having much better inference times at real-time framerates.

In Section 3.2, we demonstrate the other building block of the fingerspelling recognition system, the method for learning metric embeddings using siamese neural networks. We demonstrate this method as a low-show learning method on KMNIST [35], a dataset of hand-written Japanese characters. Unlike on fingerspelling datasets, which are expensive to collect and generally do not contain enough variability in their data samples to train a robust classifier based on a neural network, KMNIST is sufficiently large and clear that we are able to investigate what embedding the neural network actually learns. We investigate the ability of the network to classify characters from classes that were not present at training, and use adversarial example generation techniques to show that it learns an ensemble of orthogonal feature detectors. Finally, because of

a multi-headed neural network architecture, we are able to partially investigate how much data the siamese neural network needs to learn a metric embedding.

Finally, in Section 3.3, we combine the two building blocks, a fast 2D hand pose estimation neural network, and the method for learning metric embeddings using siamese networks, to demonstrate a system for cross-signer fingerspelling recognition. We apply it to a cross-signer fingerspelling dataset, and investigate its performance.

3.1 Faster 2D Hand Pose Estimation

We begin by describing the implementation of a faster, more efficient convolutional neural network for 2D hand pose estimation, specifically key point location. This accounts for the *hand pose feature extractor* used in Section 3.3. The individual architectures, and training choices, are already widely known in the literature, but have not yet been applied to the problem of 2D hand pose estimation, resulting in unnecessary complexity when using transfer learning to build systems using prior art. One of our main findings, we believe, is that in the existing literature insufficient attention was paid to the issue of performance, and that performance was traded off for quality of results to an unreasonably degree.

3.1.1 Network Architecture

We based our main neural network architecture choices on the work on MobileNetV2 [70, 5], having investigated a number of alternative network architectures, specifically residual networks [71], shuffle networks [72], and dual-path networks [73].

There are two primary approaches to neural network design for hand pose estimation. The primary existing one, as in the works [3, 2] is a convolution-deconvolution neural network architecture 3.1. For this approach the primary determinant of inference speed and accuracy is (a) the depth of the network, (b) the number of internal features in the final layers preceding the deconvolutional block, and (c) the number of channels in the final convolutional layers before the pooling step.

The other existing approach, as used in [4], is to use an existing neural network architecture, as trained on ImageNet, as reuse it with only the final classification layer replaced by a domain-specific layer.

We used a number of comparable designs, generated by simply varying the parameters ($a-c$), and investigated their effects on performance. We find that the primary causes of poor inference speed on input images are (a) the size of the image (we saved a factor of $\frac{1}{16}$ by scaling images by $\frac{1}{4}$ in both dimensions), and (b) the number of channels in *both* the convolutional and the deconvolutional block. We did not find that the depth of the network, keeping the number of channels approximately constant had as much of an effect.

In this architecture, the primary purpose of the bottleneck between the convolutional and deconvolutional blocks is to “compress” the hand pose features extracted from the image. Each channel i in the final convolutional layer after the pooling step is essentially a learned map $\phi_i(x)$ mapping an input image $x \in \mathbb{R}^{3 \times 256 \times 256}$ to $\phi_i(x) \in \mathbb{R}^{256}$, using the typical 256-channel network.

If $C = 256$ is the number of channels in the convolutional layers, the number of learnable parameters in a convolutional layer is $O(C^2)$. Thus halving the number of channels can be profitably performed on those layers that have a large number of channels C already, and also on those for which the constant of proportionality of C^2 in C^2WH is particularly large.

This is the case for most layers in the deconvolutional block. In the convolutional block, as seen in Figure 3.1, the image size gets progressively halved. This is conventionally done [74] in order to learn CNN features with a large *receptive field size*, which for a deep network means learning features that take the whole image into account. This implies that in the convolutional block, the cost in number of trainable parameters is proportional to

$$C^2WH/4^d, \tag{3.1}$$

where d is the depth measured in “blocks”, which are units of consecutive convolutional layers with a downsampling convolutional layer with stride 2 at the end. Thus reducing C for convolutional blocks is ineffective.

Deconvolutional blocks, a staple of image segmentation neural networks [75], on the other hand have the number of parameters proportional to

$$C(d)^24^d, \tag{3.2}$$

where d again measures depth from the start of the first deconvolutional block, in units

Name	Channels	Image size / stride
(input)	3	64×64
Conv+BN+ReLU	32	$64 \times 64/2$
InvertedResidual	16	32×32
InvertedResidual	32	32×32
InvertedResidual	32	32×32
InvertedResidual	32	32×32
InvertedResidual	32	$32 \times 32/2$
InvertedResidual	64	16×16
InvertedResidual	64	16×16
InvertedResidual	64	16×16
InvertedResidual	64	$16 \times 16/2$
InvertedResidual	128	8×8
InvertedResidual	128	8×8
InvertedResidual	128	8×8
InvertedResidual	128	$8 \times 8/2$
Conv+BN+ReLU	128	4×4
(extracted features)		
Conv ^T +BN	64	$8 \times 8 * 2$
Conv ^T +BN	64	$16 \times 16 * 2$
Conv ^T +BN	32	$32 \times 32 * 2$
Conv ^T +BN	32	64×64
Conv	21	64×64

Figure 3.1: Hand pose heatmap regression network for 21 heatmaps, one per keypoint.

of the number of stride-2 layers (see Figure 3.1). Here $C(d)$ is the a free parameter, the chosen number of layers at depth d . Following existing literature on convolutional-deconvolutional architectures [75], $C(d)$ is usually chosen to be inversely proportional to d , $C(d) \sim 1/d$. Finally, in a deconvolutional architecture the final output is an image of size $C(d')WH$, with W and H being the original image dimensions, and d' being the number of blocks,

$$d' = \log_4 W. \quad (3.3)$$

This analysis implies that the greatest savings can be made in the latter deconvolutional layers, where the number of learnable parameters is proportional to

$$C^2 d'^{-2} WH, \quad (3.4)$$

which is substantially larger than

$$C^2WH/4^d, \tag{3.5}$$

for the convolutional layers. This is the first reason that motivates sacrificing benchmark accuracy for performance, the benchmark accuracy decreases at a rate slower than C^2 , the number that determines performance.

The second reason is that, as discussed in 3.3, our ultimate goal is to use the internal bottleneck layer of the convolutional-deconvolutional network as a *feature extractor*. More precisely, if θ are the learnable network parameters, we first approximately solve the optimization problem

$$\bar{\theta} = \operatorname{argmin}_{\theta} \mathbb{E}_{x,y \in X} L(y, \operatorname{deconv}_{\theta}(\operatorname{conv}_{\theta}(x))) \tag{3.6}$$

where $x, y \in X$ are the training samples, X the empirical data distribution, and L the heatmap regression loss function.

Our system design in Section 3.3 relies on having access to learned features for the multi-headed neural network architectures. In our case we use $\operatorname{conv}_{\theta}$, which maps images in $\mathbb{R}^{3 \times 64 \times 64}$ to vectors in \mathbb{R}^{128} . Having learned the weights $\bar{\theta}$ according to 3.6, we fix the mapping $\operatorname{conv}_{\theta}$, the *learned feature extractors*, and use $\operatorname{conv}_{\theta}(x)$ as inputs to the siamese neural networks.

The primary concern in this procedure is, as always, whether the new neural network will overfit the available training data, which we discuss in Section 2.1. The technique we use for reducing overfitting [32] thus coincides with the technique for increasing performance as discussed above: reduce the number of channels C in $\operatorname{conv}_{\theta}(x)$.

3.1.2 Efficient Training

Similar to the previous section, a number of techniques widely used in state of the art research on image recognition have not yet been applied to hand pose estimation, leading to unnecessarily lower training performance. This means we present the application of techniques in both standard textbooks [32], and current image classification research [76].

Data augmentation

We used a number of standard data augmentation techniques [76, 32], in the following order:

- Resize image;
- Gaussian blur with a radius of 1 pixel (out of 64);
- Color jitter, modifying contrast, brightness, hue, and saturation by 40%;
- Image color normalization;
- White Gaussian noise of magnitude 0.01;
- Cutout regularization [77] using a square of 8×8 pixels (out of 64).

Training procedure

Following [76], we initialized all linear and convolutional layers using the Xavier initialization [78]. We initialized all batch normalization layers to the identity, except the batch normalization layers at the ends of residual blocks, which we initialized to zero following [76].

We trained the networks using a procedure consisting of 5 warmup epochs with a learning rate of 10^{-5} , followed by Nesterov accelerated gradient descent [79] using a learning rate schedule based on warm restarts [80],

$$\eta_k = \eta_+ + (\eta_- - \eta_+) \times \frac{1}{2} \left(1 + \cos 2\pi \frac{(k - k_{\text{last}})}{k_{\text{next}} - k_{\text{last}}} \right). \quad (3.7)$$

In [80] it was shown that using the warm restart schedule for stochastic gradient descent improved performance of vanilla stochastic gradient descent. Warm restarts were set, experimentally, at epochs 5 and 10, out of a total of 20 epochs, which was sufficient for convergence.

Varying the optimization method did not produce better results than stochastic gradient descent with warm restarts. Furthermore, we were unable to replicate the success reported in the literature for the more recent methods. Specifically, we found that Adam [81] did not outperform stochastic gradient descent with warm restarts, and

that AdaBound [82] performed more poorly than either Adam or stochastic gradient descent. AdaBound in particular results in both worse training error and worse generalization error. We were unable to determine the cause of this discrepancy, having used the standard implementations of SGD and Adam provided in PyTorch, as well as the author-published implementation of AdaBound. One potential source of this behaviour is the difference in the benchmarks used by different published works on large-scale optimization methods. We note that the authors of AdaBound report their results primarily on MNIST and CIFAR-10 [83], whereas the authors of SGDR worked mainly with CIFAR-100 [83]. It is plausible that the smaller size of the CIFAR-10 dataset, and the small number of classes in it, contribute to the findings regarding optimization methods being less easily generalizable to other, larger, and more complicated datasets.

The learning rate was set to a large initial value of 0.1. As discussed in [84], while large learning rates lead to much slower convergence, large learning rates have in fact a regularizing influence on the neural network. The target minimal learning rate was set at 10^{-4} , and our experiments showed that changing this value did not affect keypoint location accuracy.

In order to make the most efficient use of the available computing power, we used a large mini-batch size of 1024. This was made possible by our earlier choice of a smaller number of channels in the convolutional-deconvolutional layers, as the mini-batch size is severely limited by the available GPU memory. Typical mini-batch sizes for networks such as ResNet-50 [71] are, in comparison, 256, but can also be as low as 128 as in [73].

We did not use half-precision floating-point training. Although reported in [85] to be valuable for very large neural networks trained on ImageNet, we tested the difference using [86], and found the difference to be negligible (1.1ms vs 1.0ms, within standard error of each other in benchmarking).

We trained the networks on a machine with a single P100 GPU, at a rate of approximately 1 minute per epoch. With the choices made above, we found this is substantially better than the training performance of a more typical architecture ResNet-50 (used in [3, 4]), which resulted in 10 minutes per epoch.

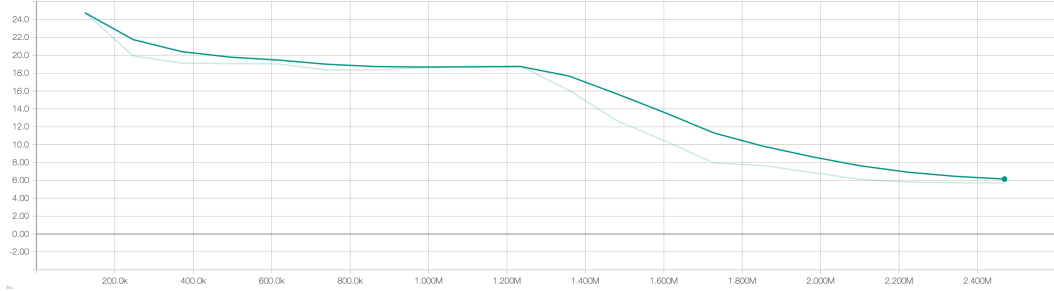


Figure 3.2: Test error convergence for hand keypoint heatmap regression, l_2 pixel distance. The model was refined further after this run.

Heatmap regression

When selecting the loss function for hand pose 2D keypoint estimation we compared heatmap regression to pixelwise binary cross entropy.

In heatmap regression, the network outputs a single real number per pixel, $\phi_{i,j}(x)$. The target function for heatmap regression is a Gaussian kernel centered around the true keypoint location of a small pre-determined size, which we chose to be 2 pixels (for an image of size 64 pixels). Thus the target heatmap is given by

$$y_{i,j} = e^{-((u_i - \hat{u})^2 + (v_j - \hat{v})^2) / 2r^2}, \quad (3.8)$$

and the loss function is

$$L = \frac{1}{WH} \sum_{i,j} (\sigma(\phi_{i,j}(x)) - y_{i,j})^2. \quad (3.9)$$

The alternative to this loss is the pixelwise binary cross entropy loss,

$$L = \frac{1}{WH} \sum_{i,j} \text{BCE}(\sigma(\phi_{i,j}(x)), y_{i,j}), \quad (3.10)$$

where $y_{i,j}$ must be normalized appropriately compared with the heatmap l_2 loss above. In our experiments we found negligible accuracy differences between l_2 heatmap regression and pixelwise binary cross entropy, and chose heatmap regression.



Figure 3.3: GANerated hand pose samples

3.1.3 Comparisons and Results

The test loss convergence, following this procedure is shown in Figure 3.2, and took 10 minutes total before converging. The inference times and l_2 pixel errors in predicted 2D keypoint locations are shown in Figure 3.4, and compared with existing work. Note that there are some discrepancies about how keypoint errors are reported, we chose mean l_2 pixelwise error, and converted other reported results where this measure was not directly available.

We further experimented with a progressively decreasing schedule for the r param-

	[2]	[3]	[4]	Ours
Number of channels	256	512	2048	128
Inference time (ms)	5	13	26	1
Average 2D keypoint error (px)	14	8	9	16

Figure 3.4: Comparison of performance for 2D hand pose estimation. Keypoint error is measured in pixels in a 256-pixel image. Data: [3, 2, 4] and our own work. Keypoint error numbers are re-computed using the data in the cited papers, to make them directly comparable as average l_2 errors measured in pixels.

eters, which controls how accurate the heatmap is expected to be, which is an approach inspired partly by [53]. We found this to be of no particular benefit, primarily because with the default setting of $r = 2\text{px}$ the network converged quickly, and thus progressively decreasing r to encourage the network to become more and more precise was not necessary. Setting $r = 1\text{px}$ did, however, cause the network to converge poorly, leading to much higher mean l_2 error at 30px (compared to 16px).

One particular property of the GANerated dataset [3], a sample of which is shown in Figure 3.3 is that hands, while unoccluded in our case, take extremely unnatural positions and finger angles. This serves to regularize the network and avoid overfitting to only the common hand poses.

3.2 Learning Embeddings on KMNIST using Siamese Networks

In order to apply the proposed method described in Section 1.2 to fingerspelling, we would like first to investigate whether the method works on a similar problem in a known domain, and then to investigate in more detail what the method actually learns about the dataset, and what features it learns. The latter cannot be done directly in the fingerspelling setting, as discussed in Section 2.1 because labelled fingerspelling datasets are expensive to collect and severely limited in the variety of gestures and signers represented. We therefore first look for a different dataset to establish that the method should work.

Looking for a dataset with a medium number of classes, we settled on the KMNIST dataset [35]. This dataset is illustrated in Figure 3.6. Each 28×28 image belongs to



Figure 3.5: Heatmap regression results, highlighted in red, for the downsampled GAN-generated dataset (64 pixels), *best seen in colour*. Highlighted are the base of the wrist the tip of the fingers (across columns).

one of 49 classes, and each class is a Japanese character. The variability within each class is significant: each image is a *handwritten* character, and therefore the images

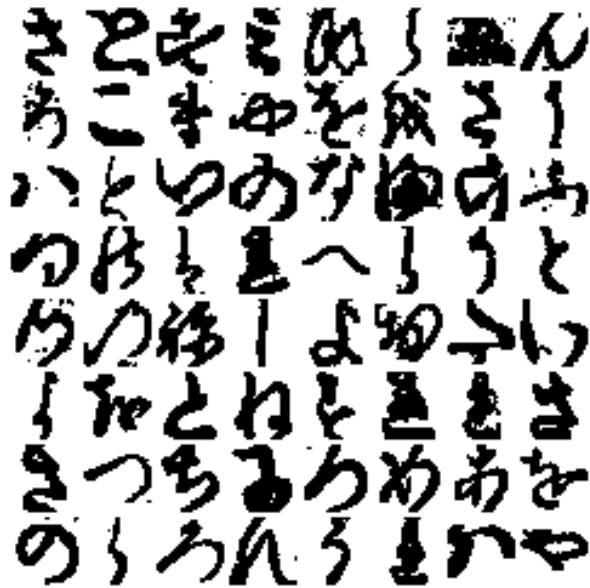


Figure 3.6: A random selection of samples from the KMNIST dataset. Note the amount of variability between characters in the same class.

differ in the style of stroke as well as the variation permitted by the Japanese writing system. This is a fascinating dataset, and we urge the reader to read [35] for further details.

There is a limited analogy with cross-signer fingerspelling. In fingerspelling, the exact gesture performed for a particular letter differs both within the alphabet to the extent permitted by the system, and also between signers, as different signers performing the same gestures produce only a small amount of variability by themselves. As discussed earlier, this is partially responsible for the substantially lower accuracy rates of fingerspelling recognition in the wild versus in the lab. Thus for fingerspelling, as discussed in 2.2 we treat the problem as if there is a known body of symbols with limited variability, followed by a larger unlabelled body of symbols with much greater variability, which we would like to classify based on knowing only a few labels per class, an reduction to the problem of *low-show learning*.

The analogy we propose for this particular experiment is to split the 49-character dataset by characters, specifically a 40-character training set, and a 9-character test

set. We then describe a procedure for learning a function ϕ ,

$$\phi : \mathbb{R}^{1 \times 28 \times 28} \mapsto \mathbb{R}^f, \quad (3.11)$$

where f is the number of features chosen to be with a small factor of $\log_2(49)$. This learning procedure results in a distance metric on the outputs of this function ϕ , with $\phi(x_1) \cdots \phi(x_2)$ measuring the similarity of images x_1 and x_2 .

Having done this on the 40-character training dataset, we find that ϕ can meaningfully distinguish the other 9 character from the testset, both by themselves, but also in the presence of the original 40 characters. This then leads into the presentation in Section 3.3.

3.2.1 Learning a Siamese Embedding for the KMNIST dataset

We used fundamentally the same neural network design for the KMNIST dataset as for the fingerspelling dataset. This ensures that the choices made with respect to neural network architecture and training are transferrable, and ensures we can draw conclusions about the neural networks and the learned embedding independently of the dataset quality issues already discussed in Section 2.1. We therefore present a precise description here, and present the changes made to accommodate the hand pose and fingerspelling domain separately in Section 3.3.

We begin by describing the basic classifier architecture based on MobileNetV2 [5], which is shown in full in Figure 3.7. Following [5], the basic block consists of Conv+BN+ReLU layers,

$$x \mapsto \min(6, \text{ReLU}(\text{BN}(\text{conv}_\theta(x))))). \quad (3.12)$$

Batch normalization is the primary method of regularization in this network design. The layers are then arranged in *inverted residual blocks*, as shown in Figure 3.8. We use the standard cross-entropy loss function. The network is trained with stochastic gradient descent with warm restarts [80], according to the procedure described in Section 3.1.2, same as for the hand pose network. Due to the lower complexity of the KMNIST dataset, this takes 30 seconds per epoch on an NVIDIA P100 GPU.

In [5] it was found that this architecture is superior in performance to other conven-

Name	Channels	Stride	Size
(input)	1	1	28×28
Conv+BN+ReLU	32	1	28×28
InvertedResidual	32	1	28×28
InvertedResidual	32	1	28×28
InvertedResidual	32	1	28×28
InvertedResidual	64	2	14×14
InvertedResidual	64	1	14×14
InvertedResidual	64	1	14×14
InvertedResidual	64	1	14×14
InvertedResidual	64	1	14×14
InvertedResidual	96	2	7×7
InvertedResidual	128	1	7×7
InvertedResidual	32	2	4×4
AveragePool	32		1×1
(extracted features)			
Dropout(0.5)	32		1×1
FullyConnectedLinear	$ C $		1×1

Figure 3.7: The basic classifier architecture, the convolutional block above the line, the classification block below.

Name	Channels	Stride	Size
(input x)	c		$w \times h$
Conv+BN+ReLU	$4c$	1	$w \times h$
DepthwiseConv+BN+ReLU	$4c$	stride	$w \times h$
Conv	c'	1	$w \times h$
BN			
(output y)			$w \times h$

$x + y$ if $c = c'$, otherwise y

Figure 3.8: The inverted residual block [5], shown with expand ratio 4, with c input channels and c' output channels.

tional architectures such as ResNet-18. We further compared the classifier architecture with one based on dual-path networks [73], and found that on the KMNIST dataset using a dual-path network did not offer any statistically significant advantages, resulting similar accuracies of $\sim 94\%$ while sacrificing performance. Network inference and training speed were explicit goals, allowing us to efficiently train and compare multiple approaches later on. The resulting neural network is of the same depth as the one in

[5], but uses fewer channels and works on smaller images to avoid overfitting to the KMNIST dataset with 40 classes versus ImageNet’s 1000. The number of channels and the number of features in the final convolutional layer was selected heuristically to minimize the number of learnable parameters while still achieving good accuracy of around 94%.

When reporting results, we found that taking the same network and training it multiple times (with random re-initializations) resulted in somewhat random generalization errors on the test set, of around 0.5%. For this reason we do not always follow the literature in reporting accuracies to the fourth digits of precision, as we found that the noise present in the training process is substantially larger than that.

Following the discussion in Section 2.2, the embedding we would like to compute is a learnable non-linear function

$$\phi : \mathbb{R}^{1 \times 28 \times 28} \mapsto \mathbb{R}^f \quad (3.13)$$

where $f \approx 10$ is the number of features, depending on some parameters θ , with the property that for two input images x_1 and x_2 , the quantity

$$\phi(x_1) \cdot \phi(x_2) \quad (3.14)$$

measures the similarity between x_1 and x_2 , and there is an additional constraint that the embedding lies on the unit sphere

$$\|\phi(x)\| = 1. \quad (3.15)$$

This embedding is independent of whether the labels y_1, y_2 of x_1, x_2 are known.

The basic motivation behind this approach can be seen by taking the number of features to be $f = 1$. In this case, ϕ is a simple linear discriminant, and the model predicts that images x_1, x_2 belong to the same or different class according to

$$\text{sign}(\phi(x_1) \cdot \phi(x_2)). \quad (3.16)$$

When the number of features is greater, but still small, on order of 10, the embedding distributes the input images on the unit sphere in \mathbb{R}^{10} , and the angle between

$\phi(x_1)$ and $\phi(x_2)$ measures whether the models predicts x_1 and x_2 to belong to the same or different class. Unlike in the linear discriminant case, the particular angle threshold cannot be set by hand, so we model the probability of $y_1 = y_2$ as

$$p_{1,2} = \sigma(\beta + \alpha^2 \phi(x_1) \cdot \phi(x_2)), \quad (3.17)$$

where $\phi(x)$ is normalized to the unit sphere.

The learnable parameters α, β directly determine the fraction of the 10-dimensional unit sphere that each sample x is “close to”. Since the model can overfit to classes by selecting a large α and a large negative β , resulting in very narrow clusters of points, we introduce a regularization loss on β by

$$L_\beta = \lambda_\beta |\beta|, \quad \lambda_\beta = 0.25. \quad (3.18)$$

The value of the regularization weight λ_β was selected to be 0.25 based on the model’s performance on the validation set. Typical learned values for α, β were around $(-1, +2)$.

To describe the loss function for learning the embedding, we first consider, as in Section 2.2, what happens to images in a single mini-batch when we do not know directly the classes y_i for the images x_i , but we do know whether the images pairwise belong to the same class, i.e., $y_{i,j} = \mathbf{1}_{y_i = y_j}$.

In the pure classification case we can use the cross-entropy loss function, which takes as inputs a class y_i and predicted class-wise log-odds $p_{i,c}$, and minimizes the cross-entropy between $p_{i,c}$ and $\mathbf{1}_i$. In the siamese case, there is no way to make the network output a probability distribution over classes, because the classes are not explicitly modelled.

Therefore in each mini-batch we take $y_{i,j} = \mathbf{1}_{y_i=y_j}$, and the pairwise similarity

$$p_{i,j} = \sigma(\beta + \alpha^2 \phi(x_i) \cdot \phi(x_j)), \quad (3.19)$$

and use binary cross entropy as the loss function,

$$L_{\text{BCE}} = \text{BCE}(p_{i,j}, y_{i,j}), \quad (3.20)$$

$$\text{BCE}(p, y) = -(\text{posy} \log p + (1 - y) \log(1 - p)). \quad (3.21)$$

The extra parameter $\text{pos} = |C| - 1$ accounts for the fact that there is a large imbalance between positive and negative examples given to the network during training. In a typical mini-batch, there will be $1/|C|$ positive pairs of images with the same class, so $\text{pos} = |C| - 1$. The total loss, then, is

$$L = L_{\text{BCE}} + L_{\beta}. \quad (3.22)$$

This loss function that is defined within a mini-batch has several implications. First, the network must be capable of being trained on a GPU with large mini-batches, and the parameters and their gradients must fit in memory. We found this to be a limitation with alternative network designs based on ResNet-50, specifically later for the fingerspelling dataset where the images are larger. With the MobileNet-based design we can use mini-batches of size 1024 and above, meaning there are enough positive example pairs in each mini-batch. Second, since the loss function is now random and defined per mini-batch, this has a regularization effect on the training process, similar to the discussion in [87] on how normalizing channel activations within a mini-batch leads to better generalization performance.

Having defined the loss function, we define the network architecture as shown in Figure 3.9, where ϕ_0 is the mapping learned in the first block of the classifier in Figure 3.7, the *extracted features*. This results in a multi-headed network architecture (Figure 3.10) where we learn one network for a single classification task on a dataset, then reuse a large block of it with a new network attached to its end to extend it to a second task, in this case learning the cosine similarity embedding.

In this case, primarily the block that computes the embedding is no longer a convolutional neural network because it starts with the small number of learned features from the first network block. We choose a dual-path network to compute the embedding, as it was reported in [73] to be a more efficient architecture for learning more complex features than residual blocks.

Having learned the feature extraction block ϕ_0 in the classifier network, we freeze its weights, and train only the part of the network that computes the embedding in Figure 3.9 using the procedure of Section 3.1.2.

Name	Channels	Stride	Size
(input)	1	1	28×28
(extracted features)	32	1	1
LinearDpLayer	48		
LinearDpLayer	64		
LinearDpLayer	80		
LinearDpLayer	96		
LinearDpLayer	112		
LinearDpLayer	128		
LinearDpLayer	144		
LinearDpLayer	160		
Dropout(0.5)	32		
FullyConnectedLinear	f		

Figure 3.9: Siamese embedding network, f is the number of features, typically $f = 10$.

(input image)	
extracted features $\phi_0(x)$ (Figure 3.7)	
classifier	embedding
$p_{i,c}$	$\phi(x_i)$

Figure 3.10: Multi-headed network architecture

Name	Channels
(input x)	c
Dropout(0.25)	c
FullyConnectedLinear	$4c$
BN+LeakyReLU	
FullyConnectedLinear, y	$c + \Delta c$
append($x + y[:c], y[c:]$)	

Figure 3.11: Linear dual-path layer, which creates Δc new channels.

Generalization to unseen characters

The loss function of Eq. 3.22 imposes a cosine similarity metric on the embedding $\phi(x)$. Let us now consider a completely dataset, the 9-character dataset that was not available to the network during training. Although it has not seen the individual full shapes during training, it nevertheless *has* seen most of the individual components that make up the shapes, namely the brush strokes, their edges and corners.

Given a set of labelled samples (x_i, y_i) , we take $S = 10$ *prototypes* from each class,

$\{\hat{x}_{i,c}\}$. To classify a new image x , we compute the similarities

$$p_{i,c} = \sigma(\beta + \alpha^2 \phi(x) \cdot \phi(x_{i,c})) \quad (3.23)$$

for each known prototype of each class, $S|C|$ in total. Given these similarity scores, we predict the class of x as

$$\arg \max_c \max_i p_{i,c}. \quad (3.24)$$

We investigated alternative choices, such as using

$$\arg \max_c \frac{1}{S} \sum_i p_{i,c}, \quad (3.25)$$

as well as the more complicated choice used in [31]

$$\arg \max_c \frac{\sum_i e^{\beta + \alpha^2 \phi(x) \cdot \phi(x_{i,c})}}{\sum_{i,c} e^{\beta + \alpha^2 \phi(x_{i,c}) \cdot \phi(x_{i,c})}}, \quad (3.26)$$

but we found no meaningful difference, likely due to the lower complexity of the dataset compared with ImageNet, and used the first, simpler choice.

Note that in this procedure the original class labels do not appear, and the model remembers only the features learned by the network on the pairwise comparisons within randomly-selected mini-batches. In particular, the labels y_i are not restricted to any particular classes except by accuracy.

This process introduces the same conflict as is common to all transfer learning problems [88]. Namely, in order for the network to perform well on the initial classification task, and to learn a useful embedding, it must learn the features present in the training dataset, without too much overfitting. The new classes on which the siamese embedding will be asked to do classification must satisfy two constraints: (a) be close enough to the original dataset that the learned features are *useful*, and (b) be far away enough from the original dataset to offer utility above training a simple classifier. Given the results presented in the next section on KMNIST, we believe that the KMNIST dataset satisfying these constraints only to some degree, as discussed below.

3.2.2 Results

Classification Accuracy

We first assess the classification accuracy of the learned embedding on the seen and unseen characters.

The first-stage classifier network on its own reached 94.7% accuracy on the test portion of the 40-character dataset. It is by design not capable of classifying unseen characters. By comparison the state of the art reported in [35] was achieved by a variant of ResNet-18, giving 97.33% accuracy. The lower accuracy of our classifier is a consequence of its simpler architecture.

As our goal is learning a more accurate metric embedding, we experimented with whether increasing specifically the classifier accuracy (using a deeper network) results in a better embedding, and found that it does not.

When using the learned embedding as a classifier, we must randomly select a number S of prototypes per class, as representatives of that class. Here we use $S = 10$ and $S = 25$. When evaluated on the test set of the seen 40 characters, the embedding classifier reaches 92.1% accuracy, somewhat lower than the pure classifier, and slightly higher (by 3%) than the default convolutional neural network benchmark reported in [35].

Focussing on the *unseen* characters, in Figures 3.12 and 3.13, we see that already 10 prototypes per class sufficiently cover the intra-class variability. Using 10 prototypes results in mean accuracy of 87.7%, which is 2% than the vanilla CNN benchmark trained on the full dataset. Furthermore, owing the slightly larger neural network, using 25 prototypes appears to be sufficient to cover most of the intra-class variability, giving 90.1% accuracy.

When using the embedding classifier on all the characters, seen and unseen, we get the confusion matrix presented in Figure 3.14. Two of the unseen character get misclassified often, with only 58 and 59% accuracy, but the other unseen characters are at a level comparable with the seen characters, around 80–90%.

How many dimensions

When we presented the method above, the dimensionality of the embedding was in fact a free parameter, which we set at 10. We believe that the correct range for this

89		1	1	2	6	
2	91		2	6		
	1	91				6
3		91		3	1	
7	1	76	12	1	1	
	2		2	94		
			1	1	85	11
		3	4	12	77	1
		9				90

Mean accuracy: 87.7%

Figure 3.12: Confusion matrix for the learned embedding on the 9 unseen characters. Using 10 prototypes per class.

92		2	1	2	4	
	95			5		
	1	89				8
		95		1	1	
3	1	85	9		1	
	2		4	92		
	1			89	7	
1	3	1	4	9	79	
	1	2	1			95

Mean accuracy: 90.1%

Figure 3.13: Confusion matrix for the learned embedding on the 9 unseen characters. Using 25 prototypes per class.

number is within a small integer factor of $\log_2 |C| = \log_2 49$, based on the heuristic grounds that each coordinate may provide one bit of information in its sign as the primary signal for classifying a sample. This estimate results in 5.6 coordinates, so we chose 10 as the number of features.

It is interesting, however, that using 12 features, results in higher classification accuracy on the unseen characters, 92.7%, see Figure 3.15, but in *lower* all-character mean accuracy, at 86.6%, as in Figure 3.16. We further investigate the impact of the number of embedding features in Section 3.3.

We believe that this result is due to overfitting. A heuristic explanation of this effect is that the input $1 \times 28 \times 28$ image is “compressed” by the network into a 10-

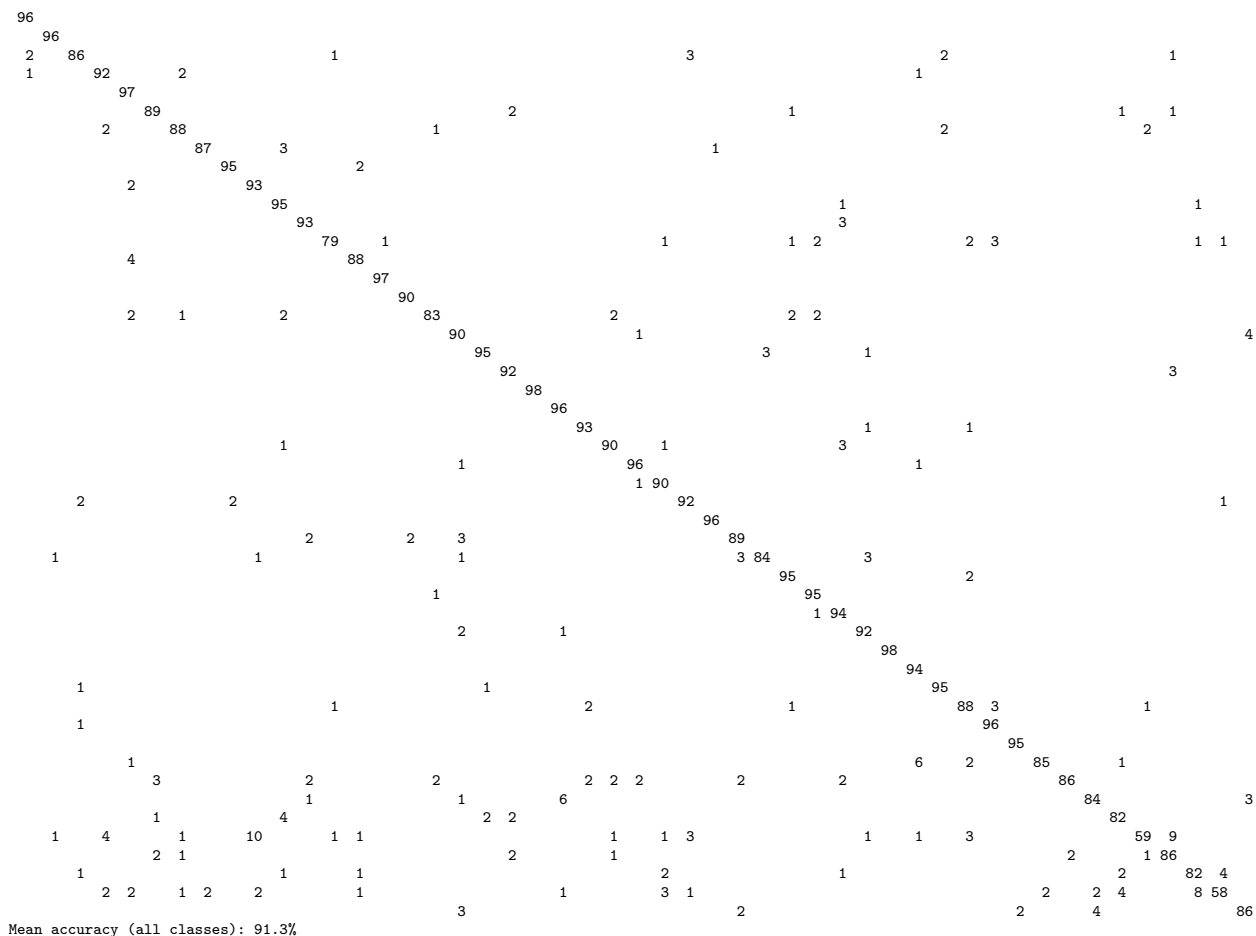


Figure 3.14: Confusion matrix for the learned embedding on all the seen and unseen characters together. Using 25 prototypes per class.

dimensional vector. Being essentially a compression method, there is pressure on it to preserve as much internal structure of the image as possible using only the available features. It follows, then, that increasing the number of available features allows it to overfit to the unnecessary details. Since the cosine similarity function we use to model the probability of two images belonging to the same class is unable to discard the unneeded extra dimensions, the new dimensions we add effectively add a source of white noise into the probability estimate

$$p_{1,2} = \sigma(\beta + \alpha^2 \phi(x_1) \cdot \phi(x_2)), \quad (3.27)$$

95			1	1						
	91		2	3		3		2		
		3	94						2	
			1	94			2	1		
			3		82	13	1			
	1				1	94	2			
		3			2	6	3	83	2	
								4	95	
				7						92

Mean accuracy: 92.7%

Figure 3.15: Confusion matrix for the learned embedding on the 9 unseen characters. Using 25 prototypes per class.

the noise being of magnitude $\alpha^2 \sim 4$.

This finding for this particular dataset conflicts to some extent with the success of using siamese networks for one-shot learning on the Omniglot dataset [36]. Recall that in that work, although the model was slightly different, using

$$p_{1,2} = \sigma \left(\sum_i \alpha_i |\phi_i(x_1) - \phi_i(x_2)| \right), \quad (3.28)$$

it is similar enough to the cosine similarity model we use. The Omniglot dataset contains thousands of characters, and [36] used 4096 features in the embedding vector, much greater than $\log_2 |C|$. We believe that the primary difference in behaviour between our choice of a small number of embedding features and their choice of a large number of embedding features is caused by the number of training classes. With a small number of training classes, the model must be explicitly stopped from overfitting, and choosing $2 \log_2 |C|$ features acts as a regularization method for the model. With a large number of classes, as in [36], or in the work on face recognition with siamese networks [57], the number of classes provides this regularization effect, and manual intervention in selecting the number of features is not needed.

In our experiments we found conventional regularization methods, specifically batch normalization [87], layer normalization [89], and dropout on the fully-connected layers [90] do not have this regularizing effect, and selecting a larger number of features leads to worse generalization errors.

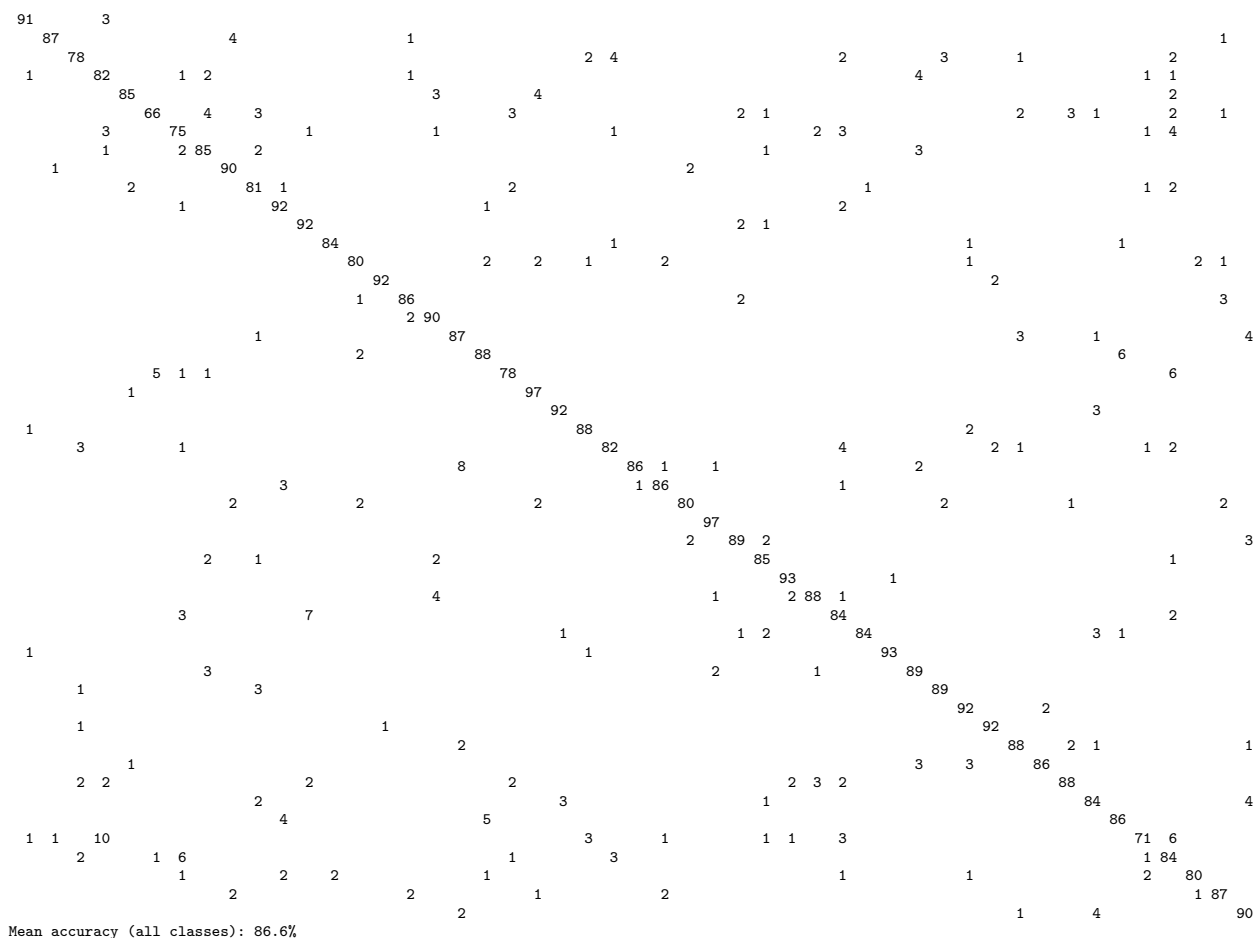


Figure 3.16: Confusion matrix for the learned embedding on all the seen and unseen characters together. Using 25 prototypes per class.

Kernel Principal Component Analysis for the Cosine Similarity Embedding

We next investigate whether we have successfully imposed the cosine similarity metric on the learned embedding. This is important because $\phi(x)$ maps samples x to the unit sphere \mathbb{R}^f , and, if successful, it evenly distributes the samples on that sphere with angles between two vectors measuring how likely they are to belong to the same class.

We test whether the embedding vectors actually come with the cosine similarity distance function using *kernel principal component analysis* [91]. Recall that kernel PCA takes a number of *unlabelled* samples x_i , and uses the kernel trick as follows. Let

$\Phi(x)$ maps the samples to an abstract high-dimensional *feature space*, and let

$$k(x_1, x_2) = \Phi(x_1) \cdot \Phi(x_2), \quad (3.29)$$

be the dot product in that space. Kernel PCA is specified using the kernel function k only, with Φ only defined implicitly through the definition of k . Kernel PCA, then, is equivalent to regular principal component analysis on the transformed matrix $K = (\Phi(x_i) \cdot \Phi(x_j))_{i,j}$, finding the most important linear features for the dataset in feature space through an eigendecomposition of K .

Note that in this procedure we already know that our embedding allows us to compute the similarity as $k(x_1, x_2) = \phi(x_1) \cdot \phi(x_2)$, which is equivalent to kernel PCA with the cosine similarity kernel. In particular, given a set of unlabelled samples x from either the 40-character training set or the 9-characters test set, we can use kernel PCA with nearest neighbours as an unsupervised clustering method. Taking the two most important PCA components we arrive at a two-dimensional visualization of the 10-dimensional feature space, the coordinates for sample x being given by

$$(\text{pca}_1(x), \text{pca}_2(x)), \quad (3.30)$$

where pca_i are the computed principal components for the matrix K .

Since the number of characters is quite high at 40, and the number of feature dimensions is much greater than the top two components that can be visualized, we select 10 characters at random and check that kernel PCA is able to successfully cluster them into regions of the 2D space. Because of the steep reduction in the number of dimensions, and limitations of kernel PCA as a dimensionality reduction method, using more than 10 characters results in an uninterpretable plot with too many classes, and using more than 10,000 samples results in unreasonable computation times because kernel PCA has time complexity $O(N^3)$ due to being based on singular value decomposition of the matrix K .

The result is shown in Figure 3.17. The primary conclusion we make is that kernel PCA is able to correctly use the same embedding with cosine similarity distance function. The reduction in the number of dimensions and the limitations of kernel PCA result in much worse inter-class decision boundaries than what we get when using our

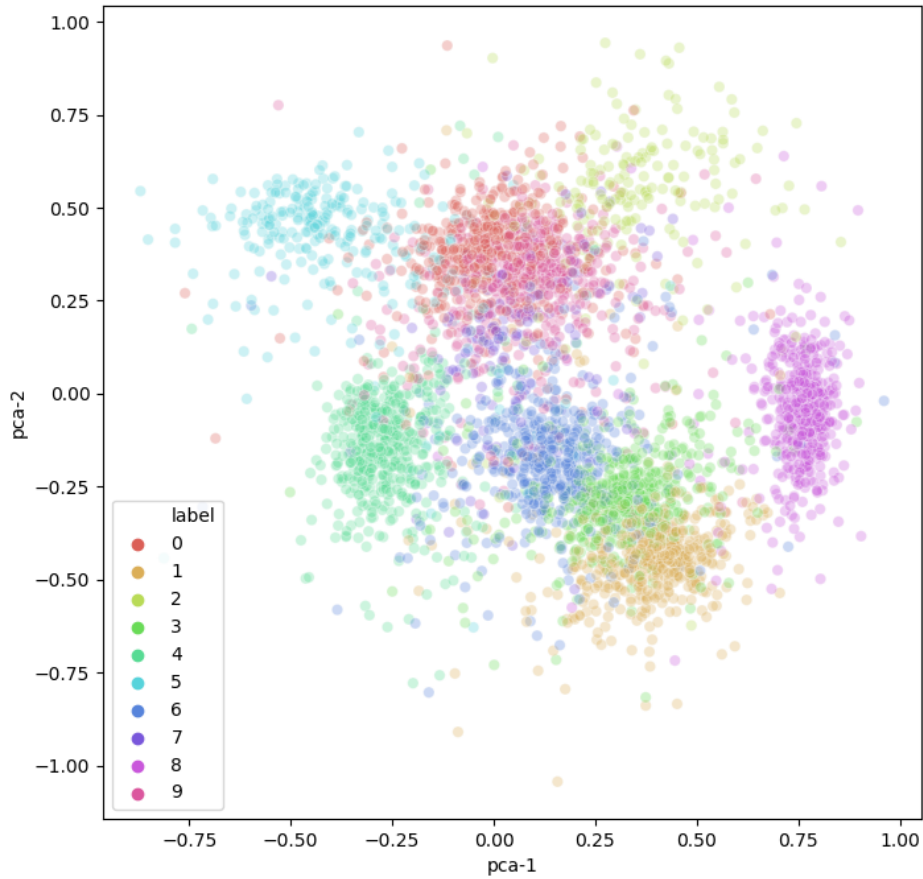


Figure 3.17: Kernel PCA applied to the cosine similarity embedding on 10 characters from the 40-character training set. *Best seen in colour pdf.*

siamese classifier directly. It would not make sense to learn the embedding, and then use a weak unsupervised clustering method like kernel PCA with nearest neighbours on the output, given that we already have trained a siamese classifier capable of low-shot classification. The primary purpose of kernel PCA, therefore, is to demonstrate that learning such an embedding may be useful as a transfer learning technique for using other unsupervised or semi-supervised methods that require nothing more than inputs with meaningful cosine similarity.

Adversarial example generation

The second approach we take to investigate the behaviour of this method is based on *adversarial example generation*. Adversarial example generation is a technique widely studied with respect to ImageNet [92]. We begin by first presenting the standard technique, then showing how it is applied to learn more about what features the siamese classifier learns.

In a conventional convolutional neural network with n layers, the network represents a sequence of non-linear functions, layers, $\phi_k(x)$,

$$x, \phi_1(x), \phi_2 = \phi_2(\phi_1(x)), \dots, \phi_n = \phi_n(\phi_{n-1}(\dots \phi_1(x))). \quad (3.31)$$

When looking at the l -th channel in the k -th layer, $\phi_{k,l}(x)$, we may ask what images cause the highest possible activation of this filter. If the filter were linear, this would be, straightforwardly, the gradient of the filter with respect to the input image,

$$\bar{x}_{i,j} = \frac{\partial \phi_{k,l}(x)}{\partial x_{i,j}}, \quad (3.32)$$

where $x_{i,j}$ is the (i, j) -th pixel of the image, and we follow the standard convention in the automatic differentiation literature using $\bar{x}_{i,j}$ for the derivative of the scalar output with respect to the input variable $x_{i,j}$. The output image that visualizes the channel $\phi_{k,l}$ can then be chosen to be

$$\sigma(\bar{x}_{i,j}), \quad (3.33)$$

or alternatively

$$\text{clamp}\left(\frac{|\bar{x}_{i,j}|}{\|\bar{x}\|}, 0, 1\right), \quad (3.34)$$

assuming the gradient is properly normalized.

The former illustrates which pixels cause the filter to activate the most, while the latter illustrates which pixels are the most salient to the filter. There is an important limitation to this approach discussed below.

Since the filter is not linear, due to the rectified linear unit non-linearities in the convolutional layers, it would not be meaningful to use $\bar{x}_{i,j}$ directly to find out what images activate the filter. In terms of heuristic dimensionality analysis, this is because

an image x that the filter responds to has units of pixel – intensity, whereas $\bar{x}_{i,j}$ has units of $\frac{\text{activation-intensity}}{\text{pixel-intensity}}$.

An iterative procedure then is to start with an empty image x_0 , and define a sequence of images

$$x_{m+1} = x_m + \eta \bar{x}_m, \quad (3.35)$$

or

$$x_{m+1} = x_m + \eta \text{sign}(\bar{x}_m), \quad (3.36)$$

where \bar{x}_m is the input-wise gradient described above.

The converse of this approach is *adversarial example generation*, which is to take a particular starting image x_0 with a known class y , and ask for the smallest possible perturbation that will cause the model to misclassify the image as a different class y' . This again is an iterative process, resulting in a sequence of one or more images, commonly one,

$$x_{m+1} = x_m + \eta \text{sign}(\bar{x}_m). \quad (3.37)$$

This time the gradient $\bar{x}_{m,ij} = \partial p_{y'} / \partial x_{m,ij}$ is taken of the output probability $p_{y'}$, the probability the model assigns for the input image belonging to class y' . This results in images such as those seen in [93].

For the siamese classifier we can adapt this approach as follows. Given a small number of images, $n = 8$, we compute the embedding $\phi(x_a)$ for each image, and the pairwise similarities $s_{ab} = \phi(x_a) \cdot \phi(x_b)$.

In order to ask which features of the image the model relies on to make its determination of similarity in the abstract embedding space, we can compute the gradient

$$\bar{x}_{a,ij}^{(b)} = \frac{\partial s_{ab}}{\partial x_{a,ij}}. \quad (3.38)$$

This results in an (a, b) -sized array of images given by $\sigma(\bar{x}_a)$, where each image is the input-wise gradient of the similarity between images a and b with respect to the image a .

This is shown in Figure 3.18. In this figure, the rows are the input images, the columns are the target images, taken to be the same set of images. Red pixels are the pixels highlighted by $\sigma(\bar{x}_a^{(b)})$ as described above.

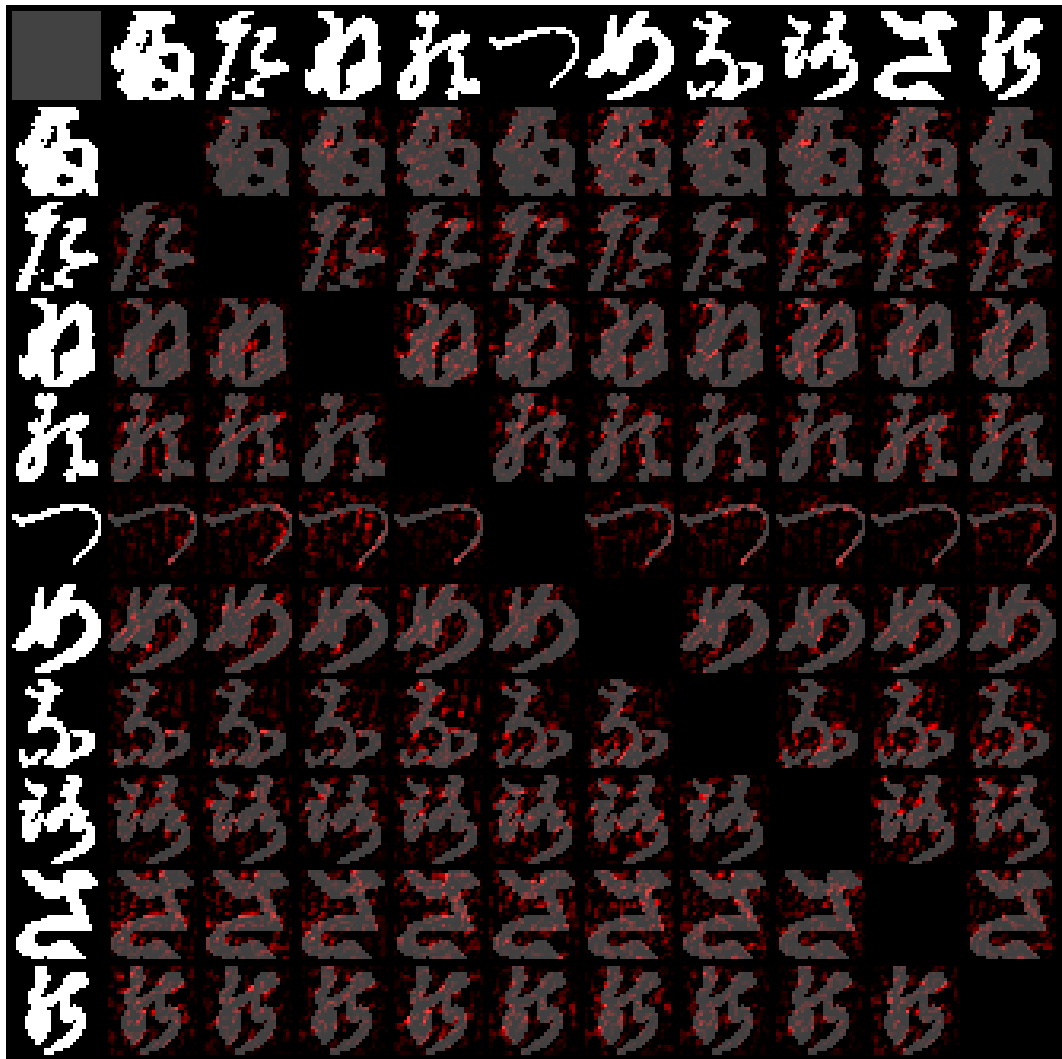


Figure 3.18: Pairwise similarity gradients for 10 randomly-selected characters. *Best seen in colour pdf.*

Looking at which pixels the pairwise similarity depends on the most, we can conclude that it appears to learn a number of orthogonal edge and corner detection filters. We may conclude that the filters are orthogonal based on the fact that imposed a cosine distance function on the embedding, not based on this image. For some characters (columns 3 and 4, in particular), it appears to learn the general shape of a character, matching it to the input image. For others (row 5), it focuses primarily on the character edges.

This implies that the siamese embedding learns, on the one hand, a set of orthogonal feature detectors that respond to edges and corners, and on the other hand, some feature detectors appear to match the shape of the target characters.

The primary limitation of this technique for elucidating what the embedding actually learns is that we are trying to do two things at once. On the one hand, we are trying to use the input-wise gradient $\bar{x}_{i,j}$ to learn which pixels a nonlinear feature detector responds to. On the other hand, this is the same technique that adversarial example generation uses to determine how to *mislead* the network into misclassifying the input image. This means there is tension, whether the highlighted pixels are really the ones that the feature detector genuinely responds to, or whether they are the ones that the feature detector is not robust to.

Furthermore, the information contained in the input-wise gradient is extremely noisy, easily seen in Figure 3.18. This is in line with what existing literature reports for the convolutional feature layers for ImageNet, for example, as seen in [93]. In line with existing literature, we therefore believe that the *simpler* visible features learned by our network represent, specifically edges, corners, and character-shaped filters, are the real filter responses, whereas the *noisier* gradients represent adversarial examples that merely mislead the network into misclassifying the images.

Fundamentally, this represents a limitation of artificial neural networks, as, for example, discussed in [94]. The features they learn are not directly interpretable. Using the siamese network architecture imposes *additional* structure (in our case, a distance function), causing them to become slightly more penetrable using even simple techniques like kernel PCA. However the individual filters remain uninterpretable, and the technique based on adversarial example generation is clearly insufficient to show what the filters actually learn beyond obvious low-level features.

3.3 Cross-Signer Fingerspelling Recognition

We now combine the two building blocks of Sections 3.1 and 3.2. We begin by reusing the features learned by the hand pose network of Section 3.1, Figure 3.1, and training a dual-path network to convert the features extracted by the convolutional block into an embedding. The embedding itself is computed using the same procedure as described in Section 3.2.1, where the place of the classifier network is taken by the convolutional

Name	Channels	Size / stride
(input)	3	64×64
(extracted features, Figure 3.1)	128	4×4
AveragePool	128	1×1
LinearDpLayer	128	
LinearDpLayer	144	
LinearDpLayer	160	
LinearDpLayer	176	
LinearDpLayer	192	
LinearDpLayer	208	
LinearDpLayer	224	
LinearDpLayer	256	
Dropout(0.25)	32	
FullyConnectedLinear	f	

Figure 3.19: Siamese embedding network architecture for fingerspelling, using f features in the embedding.

block of the hand pose network. The resulting architecture is shown in Figure 3.19.

The two key hyperparameters in the embedding network are the number of output features, as well as the number of features extracted by the hand pose network. One particular finding is that using 256 hand pose features instead of 128 causes the model to overfit on the fingerspelling dataset, which we believe is caused by the discrepancy in the sizes of the datasets, as the original dataset contains 140k images of extremely diverse hand poses, shown in Figure 3.3, whereas the fingerspelling dataset contains 50k images, which, as we describe below, contain too little variability to meaningfully capture all the hand poses possible in fingerspelling. As discussed in Section 2.1, there are significant limitations on the quality and quantity of datasets available for fingerspelling. The primary constraints for our work are that the datasets be varied and labelled, which is different from non-deep-learning-based work on fingerspelling, in which the condition is merely that they should be labelled. We selected two datasets, one an ASL fingerspelling dataset due to Pugeault [7], and the RWTH German fingerspelling database [6].



Figure 3.20: Samples from the RWTH dataset [6]

3.3.1 The RWTH Fingerspelling Dataset

A random selection of images from RWTH dataset is shown in Figure 3.20. This dataset consists of approximately 24 signers, each performing 35 gestures recorded with 2 cameras. The dataset requires further preprocessing to extract the hands from the images. For the purpose of preprocessing we used the hand detection work of [2], and several images on which this network failed to detect the hand can be seen in the figure (individual fingers, empty space).

The fundamental limitation of this dataset is the number of images. The subjects

hold the same fingerspelling gesture for approximately 10 seconds, and this is what is recorded on camera. Following the same procedure as [2], we make sure to not include exactly the same gestures in the train and test set, and extract the middle frame of each video, ultimately using one frame per video.

The primary issue with using multiple frames per video is how close the hand gestures are in the RWTH dataset. Comparing Figure 3.20 with Figure 3.22, we see much less variability. This is because the subjects were familiar with sign language and instructed by the lab to hold the correct gesture [6]. This results in an exceptionally clean dataset, but also misses much variability that both (a) is present in in-the-wild fingerspelling gestures, and (b) prevents rich models like deep neural networks from overfitting.

As a result of this limitation, taking one frame per video results in approximately 1.3k images, which we split into training and test data in a ratio of 4:1. The progression of the training loss is shown in Figure 3.21. The number of input images was significantly increased, as shown on the x -axis, using standard data augmentation techniques:

- Random cropping
- Random shearing
- Random rotations
- Color augmentation, brightness, contrast, hue, and saturation jitter
- Gaussian blur
- Small-magnitude white noise

We further regularized the network by implementing high-probability dropout at every layer, as this was shown to generate more robust “ensembles” of networks in [95].

These techniques were insufficient, and the network failed to learn a meaningful embedding, as shown in Figure 3.21. We further tested the same architecture by training a straightforward classifier, and this failed also, the test loss failed to decrease meaningfully below the level expected from random guessing.

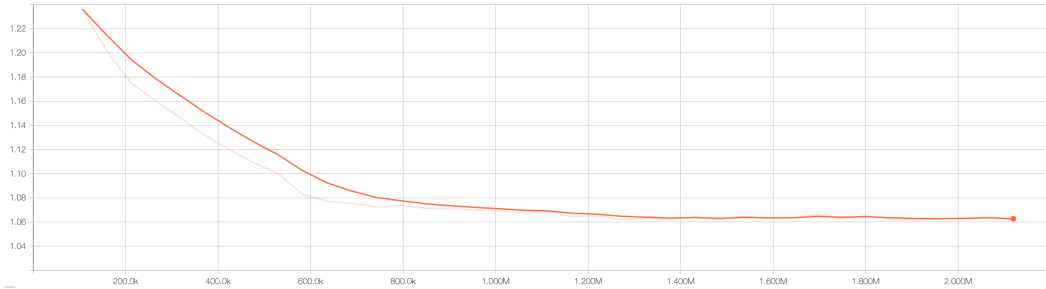


Figure 3.21: Non-convergence of the siamese embedding on the RWTH dataset, test loss shown.

3.3.2 The ASL Fingerspelling Dataset

In contrast to the RWTH dataset, the ASL dataset [7], was collected under different conditions. Random samples from it are shown in Figure 3.22. This dataset was collected under different conditions, and we first examine the main differences to understand the consequences for this work.

The dataset contains 6 signers, who were not proficient in sign language, and who were asked to perform the fingerspelling gestures in front of a webcam. Looking at the videos, it is clear that the subjects are deliberately varying their gestures, rotating and moving both the hands and the fingers. Compared with the RWTH dataset, this results in much noisier images.

For our purposes, however, the primary difference is the number of independent images available in the dataset. The RWTH contained gestures that were very similar frame-to-frame, and therefore the videos had to be aggressively pruned. The ASL dataset contains one video per signer per letter, for a total of 60k frames, but the frames are much more independent from each other. Using the same data augmentation techniques as in Section 3.3.1, this results in much more variability, and therefore less overfitting. The same network architecture, based on the same frozen hand pose network with a dual-path embedding block, converges as shown in Figures 3.23 and 3.24.

To correctly split the data into training and test sets, we must still split by signer, in order to meaningfully estimate the model’s cross-signer classification accuracy [18]. This means splitting the 6-signer dataset into a 5-signer training set and a 1-signer test set. The accuracy on the 6th signer is then reported as test accuracy. Note that



Figure 3.22: Samples from the ASL dataset [7]

despite the variability in the available frames, it is still not possible to split the dataset into training and test sets by picking the frames randomly because the frames while being variable will also be consecutive, leading to incorrect estimates of accuracy.

Comparison of results with benchmarks

As seen in Figure 3.25, the embedding classifier converges to 44.1% accuracy on the unseen signer. By varying the number of embedding features, we find that the model appears to reach at most 46% accuracy, as shown in Figure 3.26. The dependence

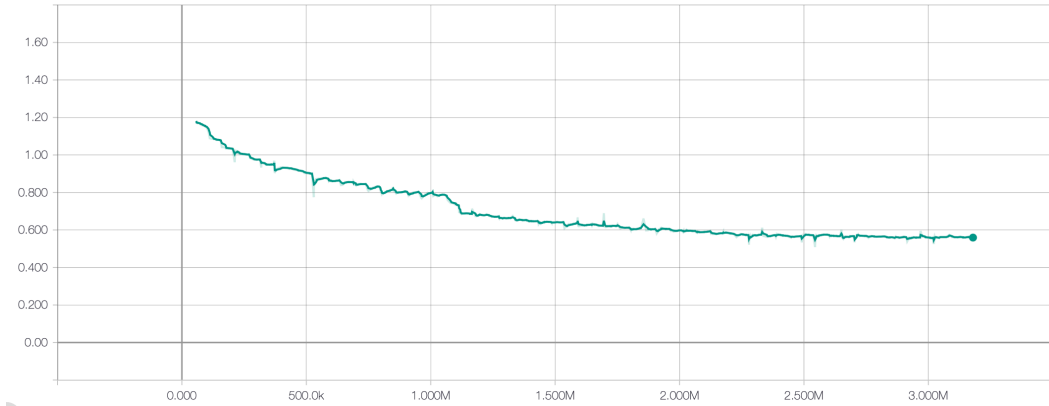


Figure 3.23: Training loss convergence for the siamese embedding ASL dataset.

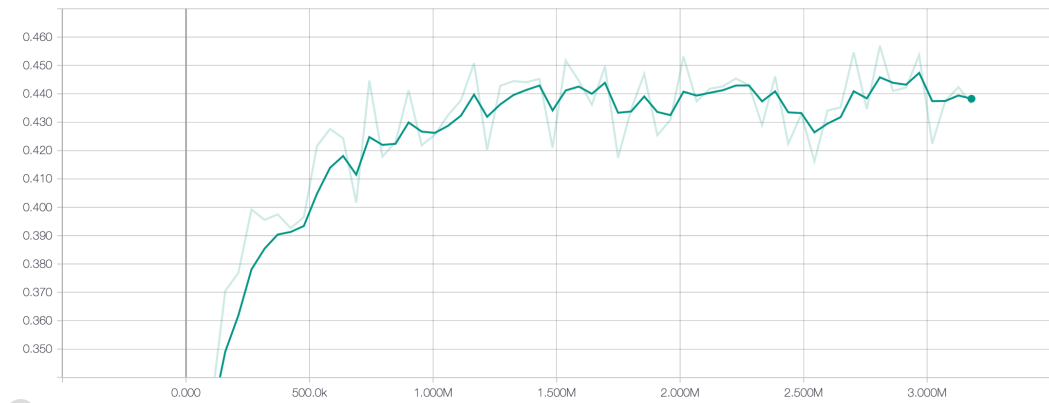


Figure 3.24: Test accuracy convergence for the siamese embedding ASL dataset.

Cross-signer	Ours	[7]	[18]	Same signers	Ours	[7]
Test accuracy	44.1%	35%	42.8%*	Test accuracy	72.4%	75%

Figure 3.25: Accuracies for cross-signer and non-cross-signer classification. *Figures marked with an asterisk were collected on different dataset, and are included for general comparison, as there are too few direct benchmark comparisons available otherwise.*

on the number of learned embedding features is similar to what we found in for the KMNIST dataset in Section 3.2, which is that there is an optimal number around approximately $2 \log_2 |C|$, and in the case of the ASL dataset we found the optimal number around 10 features. Once again, this is substantially lower than what is used in previous work such as [36].

To compare the model directly against [7], we also compute the same model’s

Number of features	Test accuracy (top)	Test accuracy (top-2)
6	32.3%	51.4%
8	43.2%	59.2%
10	46.6%	60.3%
12	45.8%	61.0%

Figure 3.26: Effect of varying the number of embedding features on the classification accuracy.

accuracy on the training set in Figure 3.25. The corresponding confusion matrix is shown in Figure 3.27. The main confused pairs are *a-t*, *c-e*, *e-o-s*, *g-h*, *k-v*, *m-n-s-t*, *p-q*. Because the train-test split of the dataset was along signers (signer E was the test signer), this is the confusion matrix on the training set, not a separately trained model on a purely random split of the dataset. Splitting the dataset random into training and test set would not in fact help estimate same-signer generalization error because the images in the data come from consecutive frames in videos and a classifier may overfit to recognize very similar frames. Since our model learns a 12-dimensional embedding, and therefore compresses the input data to a much greater extent than a typical deep neural network classifier, and since the model only sees $S = 10$ prototypes per letter (in this case, 10 prototypes per letter including all four available signers), the model is guarded against overfitting to this particular split. It is clear that the learned embedding does overfit to the particular signers in the signer-wise split of the dataset.

We would also like to note that the top-2 accuracy, is substantially higher, and is not consistent with random guessing for the fingerspelling gestures that the model gets wrong. While top-1 accuracy is the proper measure, the magnitude of top-2 accuracy suggests the embedding learns more general features of the input images.

As can be seen in the confusion matrix in Figure 3.28, the model struggles to learn to correctly classify the ambiguous shapes from Figure 2.1. The top confused pairs are shown in Figure 3.29. We can further compare the most ambiguous pairs with the hand pose PCA components represented in the MANO model [30], and it can easily be seen that the MANO model is not rich enough to fully distinguish between the harder pairs of shapes.

We further investigated whether the model trained on the ASL dataset would perform well on the RWTH dataset. We found, however, that the results on that particular test are no better than random chance. Furthermore, selecting $S = 10$ prototypes per

	a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y
a	67		1									2	4	3					21					
b		94	1			2											1							
c			66		26										4									
d				88					2	2							4							
e			1		62								1	15				15						
f		4				91																	3	
g							58	37							2									
h							25	70							4									
i									96															
k				5						78								3			1	11		
l											99													
m	2				2							38	34	2				10	12					
n	2				1							12	34	2				3	41					3
o	1		2		27							2	4	39	3			13	5					
p														3	7	68	16		3					
q															3	26	68							
r				3						5								75			10	6		
s					17							5	3	9				58	4					
t	6											12	30	3			2	7	37					
u		1								1								17			76	4		
v										5											3	84	5	
w																						7	92	
x																		1						95
y										2														98

Mean accuracy: 72.4%

Figure 3.27: Confusion matrix for the same-signer split of [7].

character class results in $S|C| = 10 \times 30$ prototypes for the entire dataset, which consists of only 1.3k images. Given that the proportion of the dataset necessary to extract prototypes is quite large at 23%, we must conclude our approach does not work at all on cross-dataset fingerspelling recognition.

We hypothesize this is because of the different image styles in Figures 3.22 and 3.20. As pointed out in [3], generating hand pose images that mimic real-world images significantly improves hand pose recognition performance. Therefore comparing Figures 3.22 and 3.20, we believe the primary reason why fingerspelling recognition learned on the ASL dataset would not transfer to the RWTH dataset is that it overfits to the particular image style in the ASL dataset.

	a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y
a	37			3	5						1	12	20	5	1	1		4	4				1	1
b		74				6				3		1	3				4		1	2				
c			2	56			9			1		4	14		3				4				1	
d	1			28	4				10	8	4				1		8			4	4		23	
e	6			2	21				1			11	6	15	7	3		11	7	1			2	2
f		8				79														1	2	5	1	
g	2		9		1	44	15			1		3	3	5	1	6		2	1				3	
h						11	78						1			2							1	1
i								70						1	3	1			2		1		1	15
k		1		3		2	1	3	11	43	6						4			2	18		3	
l	1							4	1	81											2		2	3
m	6				11							13	10	17	8	5		11	11	1			1	
n	12				10						1	10	18	9	4	2		18	9	2				
o	2		7	10		3	1					7	5	25	5	9		10	7	1			2	2
p	3				6	2		2				6	10	5	15	24		6	9				3	4
q	3		3		6	3						6	7	15	10	31		4	5				4	2
r		2	1	29	1				7	2							35			11	2		6	
s	2			2	6							5	9	13	3	2		37	12	2			1	
t	10	2		3	11						2	12	12	9	5	2		12	12	2			2	
u	1	4	3	6	2				5	2		4	5				14	1		44	4		2	
v		1		3					18								2			3	69	2		
w		1				3															9	86		
x	6		2	8	6	2		7	2	2	2	4	2	2	3	5	2	6	1				37	3
y	4				2			8					2	10										69

Mean accuracy: 45.8%

Figure 3.28: Confusion matrix for the learned embedding on the left-out signer from the ASL dataset.

To compare the results on the ASL fingerspelling dataset with the results on the KMNIST dataset more directly, we implemented the same two tests based on kernel PCA, and on adversarial example generation. Unfortunately, due to the limited accuracy of the siamese embedding classifier on the ASL dataset, we found the output of cosine-distance kernel uninterpretable, and therefore did not include it. This is due to the accuracy being substantially lower than for KMNIST, at 44% compared to $\sim 90\%$.

Furthermore, unlike for KMNIST, where the model learns moderately interpretable filters in its convolutional layers, the layers the model learns for both 2D hand pose estimation and for converting the outputs of those features to an embedding equipped

































































True	Mispredicted (%)	True	Mispredicted(%)
 (a)	 (20)  (12)	 (b)	
 (c)	 (14)	 (d)	 (23)  (10)
 (e)	 (15)  (11)  (11)	 (f)	
 (g)	 (15)	 (h)	 (11)
 (i)	 (15)	 (k)	 (11)  (18)
 (l)		 (m)	 (11)  (10)  (17)  (11)  (11)
 (n)	 (12)  (10)  (10)  (18)	 (o)	 (10)  (10)
 (p)	 (10)  (24)	 (q)	 (15)  (10)
 (r)	 (29)  (11)	 (s)	 (13)  (12)
 (t)	 (10)  (11)  (12)  (12)  (12)	 (u)	 (14)
 (v)	 (18)	 (w)	
 (x)		 (y)	 (10)

Figure 3.29: Top confused pairs (above 10% error rate) by the siamese embedding classifier. Symbols reproduced from [8] (public domain).

with a cosine similarity distance are too abstract. This is due to the model being required by the training process to perform substantially more work. While the KM-NIST model can reach sufficient accuracy by learning the shapes of characters, the hand pose model is required to reconstruct the positions of the hand’s 2D keypoints. This means the model not only has to learn the typical image filters required to locate salient points for detecting keypoint, it also has to distinguish the types of keypoints, such as producing the right heatmaps that distinguish the 4th finger from the 5th finger. This is a substantially more abstract task, and we did not find the visualization of input-wise gradients of the embedding, similar to Figure 3.18, to be interpretable, meaningful, or visually distinguishable from noise, and so do not include it here. As

discussed previously in the context of KMNIST, on which the gradient-based visualization of features is already unconvincing, this is fundamentally a problem with using convolutional neural networks to learn specifically *interpretable* features. The features are ultimately judged based on the loss function, the cross entropy loss or the heatmap regression loss, and it can be expected that the model may *overfit* to that particular loss from the point of view of the user who may want the model to be more directly interpretable. The current state of the art in the field of deep learning is not sufficiently advanced to produce even moderately interpretable learned CNN features [94].

Overall, the accuracy of the classifier is quite low. Training a classifier on this dataset, [7] report non-cross-signer accuracy of 75%, and while 45% accuracy is comparable to reported cross-signer accuracies in the literature, as noted in Section 2.1, it does not improve on the accuracy, and this particular low-shot learning approach that requires learning a metric embedding with siamese networks is quite complicated compared to the standard classifiers reported in the literature. We explained our motivation for picking this approach in 1, and found that the results are modest and do not compensate for the complexity of the method.

Chapter 4

Conclusions and Future Work

We have considered the problem of signer-independent fingerspelling recognition on static images. As discussed in Chapter 2, although the ability of neural networks to learn to recognize hand poses specifically is limited in specific cases, we found that this is not the limiting factor for fingerspelling recognition. Better, more accurate hand pose recognition would likely not result in better fingerspelling recognition. Rather, the limiting factor is the amount of labelled data that is available for training fingerspelling recognition models. We have discussed the limitations of available datasets in Section 2.1, and the dataset we trained our model on suffered from the same limitations.

Furthermore, our model based on learning an embedding with an imposed cosine similarity measure did not perform much better than similar models reported elsewhere in the literature. Our ability to effectively compare the model’s performance against suitable benchmarks is hampered by the fact that no such standard benchmarks are available in the fingerspelling literature, specifically for cross-signer recognition. For cross-signer recognition we reach error rates that are within several percent of similar cross-signer results reported elsewhere on other datasets. We have explained our motivation for using this particular low-shot learning-based approach in Section 1.2, but we must conclude that it does not offer much benefit above existing work.

Looking for the reasons of this poor performance, we have presented a study of the method’s performance on a separate KMNIST dataset, which is large and clean compared to the available datasets. We showed that the model correctly learns a cosine similarity embedding, and is capable of correctly classifying images new, previously

unseen, classes. The actual embedding learned by the model is much less interpretable on further analysis, and the question of what the model actually learns cannot be answered convincingly.

We have also established that the limiting factor of fingerspelling model accuracy is not the accuracy of the underlying hand pose estimation model. We found that it is possible to train smaller, faster neural networks for 2D hand pose estimation than reported in the literature with judicious application of known techniques for choosing the architecture and the training procedures.

As we have discussed in Section 2.3, one of the primary methods by which progress was made on the problem of 3D hand pose estimation was by generating new, larger, and higher-quality synthetic rendered datasets, which was driven by the lack of manually labelled datasets of real hand images. Generating a realistic 3D model of human hands that can cover the relatively wide range of human hand shapes is a hard task, as can be seen from the development of the MANO model. One of its primary limitations for the purposes of this work is that its representation of the range of hand pose is insufficient to cover the entire range of the fingerspelling alphabet.

As a direction for future work, it would be fruitful to examine whether a realistic model can be constructed with enough range to cover the gestures in sign language. Specifically, each hand pose would be annotated with ground truth 3D keypoint locations as in the currently available synthetic datasets, but the range of hand poses in the datasets can be much greater if the datasets were to include common fingerspelling poses. Currently they include random hand poses sampled uniformly at random from the space of anatomically permitted poses, of which fingerspelling poses form only a small subset. Such a dataset, without further improvements in model architecture may improve its performance on cross-signer fingerspelling recognition. In particular, this approach would avoid the much harder problem of weakly supervised or unsupervised learning on the abundance of in-the-wild fingerspelling data.

Bibliography

- [1] “American manual alphabet.” https://en.wikipedia.org/wiki/American_manual_alphabet. ix, 10
- [2] C. Zimmermann and T. Brox, “Learning to estimate 3d hand pose from single rgb images,” in *IEEE International Conference on Computer Vision (ICCV)*, 2017. <https://arxiv.org/abs/1705.01389>. ix, 12, 21, 22, 25, 33, 55, 56
- [3] F. Mueller, F. Bernard, O. Sotnychenko, D. Mehta, S. Sridhar, D. Casas, and C. Theobalt, “Generated hands for real-time 3d hand tracking from monocular rgb,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 49–59, 2018. ix, 3, 5, 12, 19, 21, 22, 25, 30, 33, 61
- [4] A. Boukhayma, R. d. Bem, and P. H. Torr, “3d hand shape and pose from images in the wild,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10843–10852, 2019. ix, 20, 22, 23, 25, 30, 33
- [5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018. ix, 25, 36, 37, 38
- [6] P. Dreuw, T. Deselaers, D. Keysers, and H. Ney, “Modeling image variability in appearance-based gesture recognition,” in *ECCV workshop on statistical methods in multi-image and video processing*, pp. 7–18, 2006. x, 9, 21, 54, 55, 56
- [7] N. Pugeault and R. Bowden, “Spelling it out: Real-time asl fingerspelling recognition,” in *2011 IEEE International conference on computer vision workshops (ICCV workshops)*, pp. 1114–1119, IEEE, 2011. x, 9, 54, 57, 58, 59, 61, 64

- [8] “Asl clip art.” https://wpclipart.com/sign_language/American_ABCs/index.html. x, 63
- [9] T. Pfister, J. Charles, M. Everingham, and A. Zisserman, “Automatic and efficient long term arm and hand tracking for continuous sign language tv broadcasts,” in *Proceedings of the British Machine Vision Conference 2012*, p. 4, - 2012. 1
- [10] H. Cooper, B. Holt, and R. Bowden, “Sign language recognition,” in *Visual Analysis of Humans*, pp. 539–562, Springer, 2011. 1
- [11] R. E. Mitchell, T. A. Young, B. BACHELDA, and M. A. Karchmer, “How many people use asl in the united states? why estimates need updating,” *Sign Language Studies*, vol. 6, no. 3, pp. 306–335, 2006. 1, 8
- [12] R. Poppe, “A survey on vision-based human action recognition,” *Image and vision computing*, vol. 28, no. 6, pp. 976–990, 2010. 1
- [13] D. Weinland, R. Ronfard, and E. Boyer, “A survey of vision-based methods for action representation, segmentation and recognition,” *Computer vision and image understanding*, vol. 115, no. 2, pp. 224–241, 2011. 1
- [14] S. Herath, M. Harandi, and F. Porikli, “Going deeper into action recognition: A survey,” *Image and vision computing*, vol. 60, pp. 4–21, 2017. 1
- [15] G. Guo and A. Lai, “A survey on still image based human action recognition,” *Pattern Recognition*, vol. 47, no. 10, pp. 3343–3361, 2014. 1
- [16] S. S. Rautaray and A. Agrawal, “Vision based hand gesture recognition for human computer interaction: a survey,” *Artificial intelligence review*, vol. 43, no. 1, pp. 1–54, 2015. 1
- [17] S. Mitra and T. Acharya, “Gesture recognition: A survey,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 3, pp. 311–324, 2007. 1, 3
- [18] B. Shi, A. M. Del Rio, J. Keane, J. Michaux, D. Brentari, G. Shakhnarovich, and K. Livescu, “American sign language fingerspelling recognition in the wild,”

- in *2018 IEEE Spoken Language Technology Workshop (SLT)*, pp. 145–152, IEEE, 2018. [2](#), [11](#), [12](#), [20](#), [57](#), [59](#)
- [19] D. I. Newble and R. A. Cannon, *A handbook for medical teachers*. Springer Science & Business Media, 2001. [2](#), [3](#)
- [20] M. Tytherleigh, T. Bhatti, R. Watkins, and D. Wilkins, “The assessment of surgical skills and a simple knot-tying exercise.,” *Annals of the Royal College of Surgeons of England*, vol. 83, no. 1, p. 69, 2001. [3](#)
- [21] J. D. Birkmeyer, J. F. Finks, A. O’reilly, M. Oerline, A. M. Carlin, A. R. Nunn, J. Dimick, M. Banerjee, and N. J. Birkmeyer, “Surgical skill and complication rates after bariatric surgery,” *New England Journal of Medicine*, vol. 369, no. 15, pp. 1434–1442, 2013. [3](#)
- [22] J. Causer, A. Harvey, R. Snelgrove, G. Arsenault, and J. N. Vickers, “Quiet eye training improves surgical knot tying more than traditional technical training: a randomized controlled study,” *The American Journal of Surgery*, vol. 208, no. 2, pp. 171–177, 2014. [3](#)
- [23] P. Panteleris and A. Argyros, “Back to rgb: 3d tracking of hands and hand-object interactions based on short-baseline stereo,” in *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, p. nil, 10 2017. [3](#)
- [24] S. Yuan, G. Garcia-Hernando, B. Stenger, G. Moon, J. Yong Chang, K. Mu Lee, P. Molchanov, J. Kautz, S. Honari, L. Ge, *et al.*, “Depth-based 3d hand pose estimation: From current achievements to future goals,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. [3](#)
- [25] R. Y. Wang and J. Popović, “Real-time hand-tracking with a color glove,” *ACM Transactions on Graphics*, vol. 28, no. 3, p. 1, 2009. [3](#)
- [26] B. Myanganbayar, C. Mata, G. Dekel, B. Katz, G. Ben-Yosef, and A. Barbu, “Partially occluded hands,” in *Asian Conference on Computer Vision*, pp. 85–98, Springer, 2018. [3](#), [20](#)
- [27] A. Mittal, A. Zisserman, and P. Torr, “Hand detection using multiple proposals,” in *Proceedings of the British Machine Vision Conference 2011*, p. 75, - 2011. [3](#)

- [28] J. H. Martin and D. Jurafsky, *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Pearson/Prentice Hall Upper Saddle River, 2009. 3, 4
- [29] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, “Building a large annotated corpus of english: The penn treebank,” 1993. 4
- [30] J. Romero, D. Tzionas, and M. J. Black, “Embodied hands: Modeling and capturing hands and bodies together,” *ACM Transactions on Graphics (TOG)*, vol. 36, no. 6, p. 245, 2017. 5, 22, 23, 60
- [31] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, *et al.*, “Matching networks for one shot learning,” in *Advances in neural information processing systems*, pp. 3630–3638, 2016. 6, 8, 42
- [32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016. 6, 14, 16, 28, 29
- [33] A. Halevy, P. Norvig, and F. Pereira, “The unreasonable effectiveness of data,” 2009. 6
- [34] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, “Revisiting unreasonable effectiveness of data in deep learning era,” in *Proceedings of the IEEE international conference on computer vision*, pp. 843–852, 2017. 6
- [35] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, “Deep learning for classical japanese literature,” 2018. 7, 24, 33, 35, 43
- [36] G. Koch, R. Zemel, and R. Salakhutdinov, “Siamese neural networks for one-shot image recognition,” in *ICML deep learning workshop*, vol. 2, 2015. 8, 17, 18, 46, 59
- [37] C. Valli and C. Lucas, *Linguistics of American sign language: an introduction*. Gallaudet University Press, 2000. 8
- [38] C. A. Padden and D. C. Gunsauls, “How the alphabet came to be used in a sign language,” *Sign Language Studies*, pp. 10–33, 2003. 9

- [39] N. Gkigkelos and C. Goumopoulos, “Greek sign language vocabulary recognition using kinect,” in *Proceedings of the 21st Pan-Hellenic Conference on Informatics*, p. 51, ACM, 2017. 9
- [40] M. Oliveira, H. Chatbri, Y. Ferstl, M. Farouk, S. Little, N. E. O’Connor, and A. Sutherland, “A dataset for irish sign language recognition,” 2017. 9
- [41] O. Koller, J. Forster, and H. Ney, “Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers,” *Computer Vision and Image Understanding*, vol. 141, pp. 108–125, 2015. 10
- [42] C. Neidle, S. Sclaroff, and V. Athitsos, “Signstream: A tool for linguistic and computer vision research on visual-gestural language data,” *Behavior Research Methods, Instruments, & Computers*, vol. 33, no. 3, pp. 311–320, 2001. 11
- [43] U. Von Agris and K.-F. Kraiss, “Towards a video corpus for signer-independent continuous sign language recognition,” *Gesture in Human-Computer Interaction and Simulation, Lisbon, Portugal, May, 2007*. 11
- [44] V. Athitsos, C. Neidle, S. Sclaroff, J. Nash, A. Stefan, A. Thangali, H. Wang, and Q. Yuan, “Large lexicon project: American sign language video corpus and sign language indexing/retrieval algorithms,” in *Workshop on the Representation and Processing of Sign Languages: Corpora and Sign Language Technologies (CSLT)*, pp. 11–14, 2010. 11
- [45] J. Zieren and K.-F. Kraiss, “Robust person-independent visual sign language recognition,” in *Iberian Conference on Pattern Recognition and Image Analysis*, pp. 520–528, Springer, 2005. 11
- [46] T. Kadir, R. Bowden, E.-J. Ong, and A. Zisserman, “Minimal training, large lexicon, unconstrained sign language recognition.,” in *BMVC*, pp. 1–10, 2004. 11
- [47] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, pp. 1097–1105, 2012. 12, 14

- [48] G. Lample, M. Ott, A. Conneau, L. Denoyer, and M. Ranzato, “Phrase-based & neural unsupervised machine translation,” *arXiv preprint arXiv:1804.07755*, 2018. **13**
- [49] M. Noroozi and P. Favaro, “Unsupervised learning of visual representations by solving jigsaw puzzles,” in *European Conference on Computer Vision*, pp. 69–84, Springer, 2016. **14**
- [50] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?,” in *Advances in neural information processing systems*, pp. 3320–3328, 2014. **14**
- [51] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, “Deep clustering for unsupervised learning of visual features,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 132–149, 2018. **14**
- [52] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, “The elements of statistical learning: data mining, inference and prediction,” *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005. **15**
- [53] T. Karras, T. Aila, S. Laine, and J. Lehtinen, “Progressive growing of gans for improved quality, stability, and variation,” *arXiv preprint arXiv:1710.10196*, 2017. **16, 33**
- [54] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *arXiv preprint arXiv:1411.1784*, 2014. **16**
- [55] Y.-X. Wang, R. Girshick, M. Hebert, and B. Hariharan, “Low-shot learning from imaginary data,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7278–7286, 2018. **16**
- [56] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, “Semi-supervised learning with deep generative models,” in *Advances in neural information processing systems*, pp. 3581–3589, 2014. **16**
- [57] M. Amodio and S. Krishnaswamy, “Travelgan: Image-to-image translation by transformation vector learning,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8983–8992, 2019. **16, 46**

- [58] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, “Signature verification using a” siamese” time delay neural network,” in *Advances in neural information processing systems*, pp. 737–744, 1994. 17
- [59] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, “Human-level concept learning through probabilistic program induction,” *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015. 17
- [60] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, “The omniglot challenge: A 3-year progress report,” *Current Opinion in Behavioral Sciences*, vol. 29, pp. 97–104, 2019. 17
- [61] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, “2d human pose estimation: New benchmark and state of the art analysis,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014. 19, 22
- [62] L. Sigal, A. O. Balan, and M. J. Black, “HumanEva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion,” *International journal of computer vision*, vol. 87, no. 1-2, p. 4, 2010. 19
- [63] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, “Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 7, pp. 1325–1339, 2013. 19
- [64] J. Zhang, J. Jiao, M. Chen, L. Qu, X. Xu, and Q. Yang, “3d hand pose tracking and estimation using stereo matching,” *arXiv preprint arXiv:1610.07214*, 2016. 20
- [65] S. Sridhar, F. Mueller, M. Zollhöfer, D. Casas, A. Oulasvirta, and C. Theobalt, “Real-time joint tracking of a hand manipulating an object from rgb-d input,” in *European Conference on Computer Vision*, pp. 294–310, Springer, 2016. 20
- [66] J. Tompson, M. Stein, Y. Lecun, and K. Perlin, “Real-time continuous pose recovery of human hands using convolutional networks,” *ACM Transactions on Graphics (ToG)*, vol. 33, no. 5, p. 169, 2014. 21

- [67] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, “Hand keypoint detection in single images using multiview bootstrapping,” in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1145–1153, 2017. 22
- [68] J. P. Lewis and K.-i. Anjyo, “Direct manipulation blendshapes,” *IEEE Computer Graphics and Applications*, vol. 30, no. 4, pp. 42–50, 2010. 23
- [69] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000. 23
- [70] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *CoRR*, vol. abs/1704.04861, 2017. 25
- [71] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016. 25, 30
- [72] X. Zhang, X. Zhou, M. Lin, and J. Sun, “Shufflenet: An extremely efficient convolutional neural network for mobile devices,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856, 2018. 25
- [73] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng, “Dual path networks,” in *Advances in Neural Information Processing Systems*, pp. 4467–4475, 2017. 25, 30, 37, 40
- [74] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014. 26
- [75] V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017. 26, 27
- [76] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, “Bag of tricks for image classification with convolutional neural networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 558–567, 2019. 28, 29

- [77] T. DeVries and G. W. Taylor, “Improved regularization of convolutional neural networks with cutout,” *arXiv preprint arXiv:1708.04552*, 2017. 29
- [78] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010. 29
- [79] Y. Nesterov, “Gradient methods for minimizing composite functions,” *Mathematical Programming*, vol. 140, no. 1, pp. 125–161, 2013. 29
- [80] I. Loshchilov and F. Hutter, “Sgdr: Stochastic gradient descent with warm restarts,” *arXiv preprint arXiv:1608.03983*, 2016. 29, 36
- [81] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014. 29
- [82] L. Luo, Y. Xiong, Y. Liu, and X. Sun, “Adaptive gradient methods with dynamic bound of learning rate,” *arXiv preprint arXiv:1902.09843*, 2019. 30
- [83] A. Krizhevsky, G. Hinton, *et al.*, “Learning multiple layers of features from tiny images,” tech. rep., Citeseer, 2009. 30
- [84] L. N. Smith, “Cyclical learning rates for training neural networks,” in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 464–472, IEEE, 2017. 30
- [85] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized neural networks: Training neural networks with low precision weights and activations,” *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017. 30
- [86] “Nvidia/apex, a pytorch extension: Tools for easy mixed precision and distributed training in pytorch.” <https://github.com/NVIDIA/apex>. 30
- [87] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015. 40, 46

- [88] M. Huh, P. Agrawal, and A. A. Efros, “What makes imagenet good for transfer learning?,” *arXiv preprint arXiv:1608.08614*, 2016. 42
- [89] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *arXiv preprint arXiv:1607.06450*, 2016. 46
- [90] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014. 46
- [91] B. Schölkopf, A. Smola, and K.-R. Müller, “Nonlinear component analysis as a kernel eigenvalue problem,” *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998. 47
- [92] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014. 50
- [93] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, “Synthesizing robust adversarial examples,” *arXiv preprint arXiv:1707.07397*, 2017. 51, 53
- [94] Y. LeCun, “Deep learning: Alchemy or science?,” Institute for Advanced Studies, 2019. 53, 64
- [95] D. Warde-Farley, I. J. Goodfellow, A. Courville, and Y. Bengio, “An empirical analysis of dropout in piecewise linear networks,” *arXiv preprint arXiv:1312.6197*, 2013. 56