OmniPrint - Appendix

Haozhe Sun*, **Wei-Wei Tu**#+, **Isabelle Guyon***+
∗ LISN (CNRS/INRIA) Université Paris-Saclay, France
4Paradigm Inc, Beijing, China
+ ChaLearn, California, USA
omniprint@chalearn.org

A Datasheet for dataset for OmniPrint-meta[X] datasets

Motivation

For what purpose was the dataset created? Was there a specific task in mind? Was there a specific gap that needed to be filled? Please provide a description.

This dataset was created to be a drop-in replacement of Omniglot, which is more challenging. Omniglot can hardly push further the state-of-the-art since recent methods achieved almost perfect performances. Furthermore, Omniglot was not intended to be a realistic dataset: the characters were drawn online and do not look natural. The associated task would be the classical N-way-K-shot few-shot classification task [6, 27, 12].

Who created the dataset (e.g., which team, research group) and on behalf of which entity (e.g., company, institution, organization)?

Haozhe Sun created the dataset, under the supervision of Isabelle Guyon. The work was performed at LISN laboratory, Université Paris-Saclay, France, in the TAU team, as part of the HUMANIA project, funded by the French research agency ANR. ChaLearn also supported the development of the software.

Who funded the creation of the dataset? If there is an associated grant, please provide the name of the grantor and the grant name and number.

ANR (Agence Nationale de la Recherche, National Agency for Research, `https://anr.fr/`), grant number 20HR0134 and ChaLearn (`http://www.chalearn.org/`) a 501(c)(3) non-for-profit California organization.

Any other comments?

Composition

What do the instances that comprise the dataset represent (e.g., documents, photos, people, countries)? Are there multiple types of instances (e.g., movies, users, and ratings; people and interactions between them; nodes and edges)? Please provide a description.

The instances are 32×32 RGB images of synthetic printed characters.

How many instances are there in total (of each type, if appropriate)?

OmniPrint-meta[X] is a collection of five datasets. These 5 datasets, called OmniPrint-meta[1-5], share the same set of characters and data split and only differ in transformations and styles. For each

OmniPrint-meta[X] dataset, there are 1409 classes (characters) in total. Each class has 20 image instances. In consequence, each OmniPrint-meta[X] dataset has $1409 \times 20 = 28180$ images. There are $28180 \times 5 = 140900$ images in total.

Does the dataset contain all possible instances or is it a sample (not necessarily random) of instances from a larger set? If the dataset is a sample, then what is the larger set? Is the sample representative of the larger set (e.g., geographic coverage)? If so, please describe how this representativeness was validated/verified. If it is not representative of the larger set, please describe why not (e.g., to cover a more diverse range of instances, because instances were withheld or unavailable).

These datasets are synthesized from the data synthesizer OmniPrint, thus they can be viewed as a sample of instances from all the possible images given the nuisance parameters (fonts, styles, noises, etc.). OmniPrint-meta[X] are representative of such images because the synthesis parameters of each instance were uniformly sampled, no further selection was performed. The involved scripts are Arabic, Armenian, Balinese, Latin, Bengali, Devanagari, Ethiopic, Georgian, Greek, Gujarati, Hebrew, Hiragana, Katakana, Khmer, Lao, Mongolian, Myanmar, N'Ko, Oriya, Russian, Sinhala, Tamil, Telugu, Thai and Tibetan.

What data does each instance consist of? "Raw" data (e.g., unprocessed text or images) or features? In either case, please provide a description.

Each instance is a 32×32 RGB image. Each image contains one single character from a certain script, rendered in a particular way (background, foreground, distortions, noises).

Is there a label or target associated with each instance? If so, please provide a description.

Yes, there is a label (character) associated with each instance. Furthermore, the metadata is provided for each instance, which can also serve as labels for specific tasks. The metadata includes *e.g.,* the font, background, stroke width (if applicable), blur radius, margins, rotation angle, shear, text color, etc., and the alphabet of the character.

Is any information missing from individual instances? If so, please provide a description, explaining why this information is missing (e.g., because it was unavailable). This does not include intentionally removed information, but might include, e.g., redacted text.

No. All of the metadata is provided for each instance.

Are relationships between individual instances made explicit (e.g., users' movie ratings, social network links)? If so, please describe how these relationships are made explicit.

All relationships are contained in the labels and metadata, all provided.

Are there recommended data splits (e.g., training, development/validation, testing)? If so, please provide a description of these splits, explaining the rationale behind them.

Yes, there is a recommended data split in the context of N-way-K-shot learning, between meta-train, meta-validation and meta-test. For each of the 5 OmniPrint-meta[X] datasets, there are 1409 classes (characters), each class contains 20 image instances. The first 900 classes belong to meta-train, then 149 classes belong to meta-validation, the last 360 classes belong to meta-test. This data split is chosen in order to imitate the proportion of meta-train/meta-validation/meta-test of the popular Vinyals split [33] of Omniglot [16]. The recommended data split is provided via a data loader which forms the episodes of few-shot learning.

Are there any errors, sources of noise, or redundancies in the dataset? If so, please provide a description.

We intentionally introduced various transformations and noises to each image instance. The transformation parameter space is large so there is little chance that two instances are identical.

Is the dataset self-contained, or does it link to or otherwise rely on external resources (e.g., websites, tweets, other datasets)? If it links to or relies on external resources, a) are there guarantees that they will exist, and remain constant, over time; b) are there official archival versions of the complete dataset (i.e., including the external resources as they existed at the time the dataset was created); c) are there any restrictions (e.g., licenses, fees) associated with any of the external resources that might apply to a future user? Please provide descriptions of all external resources and any restrictions associated with them, as well as links or other access points, as appropriate.

The 5 datasets OmniPrint-meta[X] are self-contained. They will exist, and remain constant, over time once we release them after the NeurIPS 2021 meta-learning challenge.

Does the dataset contain data that might be considered confidential (e.g., data that is protected by legal privilege or by doctor-patient confidentiality, data that includes the content of individuals' non-public communications)? If so, please provide a description.

The OmniPrint-meta[X] datasets were considered confidential before the NeurIPS 2021 meta-learning challenge, they have been publicly released.

Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety? If so, please describe why.

No.

Does the dataset relate to people? If not, you may skip the remaining questions in this section.

No.

Does the dataset identify any subpopulations (e.g., by age, gender)? If so, please describe how these subpopulations are identified and provide a description of their respective distributions within the dataset.

No.

Is it possible to identify individuals (i.e., one or more natural persons), either directly or indirectly (i.e., in combination with other data) from the dataset? If so, please describe how.

No.

Does the dataset contain data that might be considered sensitive in any way (e.g., data that reveals racial or ethnic origins, sexual orientations, religious beliefs, political opinions or union mem- berships, or locations; financial or health data; biometric or genetic data; forms of government identification, such as social security numbers; criminal history)? If so, please provide a description.

No.

Any other comments?

Collection Process

How was the data associated with each instance acquired? Was the data directly observable (e.g., raw text, movie ratings), reported by subjects (e.g., survey responses), or indirectly inferred/derived from other data (e.g., part-of-speech tags, model-based guesses for age or language)? If data was reported by subjects or indirectly inferred/derived from other data, was the data validated/verified? If so, please describe how.

Each instance is synthesized by OmniPrint. Each instance is an image and is directly observable.

What mechanisms or procedures were used to collect the data (e.g., hardware apparatus or sensor, manual human curation, software program, software API)? How were these mechanisms or procedures validated?

The data are synthesized using the data synthesizer OmniPrint. The involved Unicode characters were manually selected from the Unicode standard, which constitutes a set of characters from several languages around the world. The involved fonts were downloaded from a manually-defined list of URLs, the downloaded fonts were then filtered by a python program in order to filter corrupted fonts. Several distortions and noises were involved, including affine and perspective transformations, random elastic transformations, natural background, foreground text filling, etc.

If the dataset is a sample from a larger set, what was the sampling strategy (e.g., deterministic, probabilistic with specific sampling probabilities)?

The data is synthesized by a data synthesizer OmniPrint. The sampling is uniformly random in the given transformation parameter space.

Who was involved in the data collection process (e.g., students, crowdworkers, contractors) and how were they compensated (e.g., how much were crowdworkers paid)?

The data is synthesized by a computer software. However the design and implementation of the software, the choice of characters and fonts involve the authors of this paper.

Over what timeframe was the data collected? Does this timeframe match the creation timeframe of the data associated with the instances (e.g., recent crawl of old news articles)? If not, please describe the timeframe in which the data associated with the instances was created.

The five datasets were synthesized on May 22, 2021.

Were any ethical review processes conducted (e.g., by an institutional review board)? If so, please provide a description of these review processes, including the outcomes, as well as a link or other access point to any supporting documentation.

N/A

Does the dataset relate to people? If not, you may skip the remainder of the questions in this section.

No.

Did you collect the data from the individuals in question directly, or obtain it via third parties or other sources (e.g., websites)?

N/A

Were the individuals in question notified about the data collection? If so, please describe (or show with screenshots or other information) how notice was provided, and provide a link or other access point to, or otherwise reproduce, the exact language of the notification itself.

N/A

Did the individuals in question consent to the collection and use of their data? If so, please describe (or show with screenshots or other information) how consent was requested and provided, and provide a link or other access point to, or otherwise reproduce, the exact language to which the individuals consented.

N/A

If consent was obtained, were the consenting individuals provided with a mechanism to revoke their consent in the future or for certain uses? If so, please provide a description, as well as a link or other access point to the mechanism (if appropriate).

N/A

Has an analysis of the potential impact of the dataset and its use on data subjects (e.g., a data protection impact analysis)been conducted? If so, please provide a description of this analysis, including the outcomes, as well as a link or other access point to any supporting documentation.

N/A

Any other comments?

Preprocessing/cleaning/labeling

Was any preprocessing/cleaning/labeling of the data done (e.g., discretization or bucketing, tokenization, part-of-speech tagging, SIFT feature extraction, removal of instances, processing of missing values)? If so, please provide a description. If not, you may skip the remainder of the questions in this section.

No preprocessing/cleaning/labeling was performed. The datasets are made available as they were synthesized. No feature extraction or removal of instances was done.

Was the "raw" data saved in addition to the preprocessed/cleaned/labeled data (e.g., to support unanticipated future uses)? If so, please provide a link or other access point to the "raw" data.

N/A

Is the software used to preprocess/clean/label the instances available? If so, please provide a link or other access point.

N/A

Any other comments?

Uses

Has the dataset been used for any tasks already? If so, please provide a description.

No, however a variant of these datasets will be used by the NeurIPS 2021 meta-learning challenge.

Is there a repository that links to any or all papers or systems that use the dataset? If so, please provide a link or other access point.

Yes, the link is https://github.com/SunHaozhe/OmniPrint-datasets. This repository is also used to announce any necessary information related to the OmniPrint datasets *e.g.,* potential changes of the dataset hosting address.

What (other) tasks could the dataset be used for?

Besides few-shot learning classification tasks, the five OmniPrint-meta[X] datasets can be used for classification tasks of a large number of characters, and for transfer learning (each dataset being used either as a source domain or a target domain). Furthermore, as the metadata can serve as labels, other kinds of classification or regression problems can also be considered *e.g.,* classification of fonts, classification of languages, regression of rotation angle, regression of horizontal shear, etc. Finally, the datasets can be used to study disentangling the label (class character) from the nuisance variables (font, style, distortions).

Is there anything about the composition of the dataset or the way it was collected and preprocessed/cleaned/labeled that might impact future uses? For example, is there anything that a future user might need to know to avoid uses that could result in unfair treatment of individuals or groups (e.g., stereotyping, quality of service issues) or other undesirable harms (e.g., financial harms, legal risks) If so, please provide a description. Is there anything a future user could do to mitigate these undesirable harms?

The datasets can be used without further considerations.

Are there tasks for which the dataset should not be used? If so, please provide a description.

Not that we know of.

Any other comments?

Distribution

Will the dataset be distributed to third parties outside of the entity (e.g., company, institution, organization) on behalf of which the dataset was created? If so, please provide a description.

The datasets are made available to everyone via the Internet.

How will the dataset will be distributed (e.g., tarball on website, API, GitHub)? Does the dataset have a digital object identifier (DOI)?

The OmniPrint-meta[X] datasets are publicly released via Kaggle Datasets. The digital object identifier (DOI) is 10.34740/kaggle/dsv/2763401. The access information and any necessary updates are announced via https://github.com/SunHaozhe/OmniPrint-datasets.

When will the dataset be distributed?

The datasets have been released after the NeurIPS 2021 meta-learning challenge.

Will the dataset be distributed under a copyright or other intellectual property (IP) license, and/or under applicable terms of use (ToU)? If so, please describe this license and/or ToU, and provide a link or other access point to, or otherwise reproduce, any relevant licensing terms or ToU, as well as any fees associated with these restrictions.

The datasets OmniPrint-meta[1-5] are distributed via Kaggle datasets. They are licensed under a Creative Commons license CC BY 4.0 https://creativecommons.org/licenses/by/4.0/. This comes with the following guarantee disclaimer: Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors, whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

Have any third parties imposed IP-based or other restrictions on the data associated with the instances? If so, please describe these restrictions, and provide a link or other access point to, or otherwise reproduce, any relevant licensing terms, as well as any fees associated with these restrictions.

No.

Do any export controls or other regulatory restrictions apply to the dataset or to individual instances? If so, please describe these restrictions, and provide a link or other access point to, or otherwise reproduce, any supporting documentation.

No.

Any other comments?

Maintenance

Who is supporting/hosting/maintaining the dataset?

The authors of this paper are responsible for supporting the datasets.

How can the owner/curator/manager of the dataset be contacted (e.g., email address)?

The preferred way to contact the maintainers is to raise issues on https://github.com/SunHaozhe/OmniPrint-datasets. In case of emergency, the authors of this paper can be contacted via email: omniprint@chalearn.org.

Is there an erratum? If so, please provide a link or other access point.

Any necessary information or updates will be accessible via https://github.com/SunHaozhe/OmniPrint-datasets.

Will the dataset be updated (e.g., to correct labeling errors, add new instances, delete instances)? If so, please describe how often, by whom, and how updates will be communicated to users (e.g., mailing list, GitHub)?

No. New needs will be met by synthesizing new datasets.

If the dataset relates to people, are there applicable limits on the retention of the data associated with the instances (e.g., were individuals in question told that their data would be retained for a fixed period of time and then deleted)? If so, please describe these limits and explain how they will be enforced.

N/A

Will older versions of the dataset continue to be supported/hosted/maintained? If so, please describe how. If not, please describe how its obsolescence will be communicated to users.

Any necessary information or updates will be accessible via https://github.com/SunHaozhe/OmniPrint-datasets.

If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so? If so, please provide a description. Will these contributions be validated/verified? If so, please describe how. If not, why not? Is there a process for communicating/distributing these contributions to other users? If so, please provide a description.

Users are free to extend or augment the dataset for their purposes. They can also use the data synthesizer OmniPrint to directly synthesize new datasets.

Any other comments?

B Experimental details of the few-shot learning use case

This section provides the experimental details of Section 4.1 of the main paper.

B.1 Data split

We split the data into 900 characters for meta-train, 149 characters for meta-validation, 360 characters for meta-test. The full details are provided with the code. The implementation of the few-shot learning data loader which forms the few-shot learning episodes is inspired by [18] which is under MIT License.

B.2 Evaluation and reproducibility

MAML [6] and Prototypical Networks [27] were trained during 300 epochs, where each epoch is defined to be 6 batches of episodes, each batch contains 32 episodes. During meta-training, the model checkpoints were evaluated on meta-validation episodes every 5 epochs. Only the checkpoint that has the highest accuracy on meta-validation episodes during training is selected to be tested on meta-test episodes.

The backbone neural network architecture is the same for each combination of method and dataset except for the last fully-connected layer, if applicable. It is the concatenation of three modules of Convolution-BatchNorm-Relu-Maxpool.

The metric of interest is the average classification accuracy of 1000 randomly generated meta-test episodes (of the best checkpoint on meta-validation episodes). The reported accuracy and 95% confidence intervals in the main paper are computed with 5 independent runs (5 random seeds). The random seeds were fixed in advance, no cherry-picking was performed afterwards.

B.3 Baseline implementation and compute resources

The implementation of MAML baseline [6] uses the Higher library [9] of PyTorch [20]. It is adapted from [8] which is under Apache License Version 2.0. The implementation of Prototypical Networks [27] is adapted from [5] which is under MIT License.

The experiments were run on an internal cluster which is managed through SLURM [13]. The involved GPUs are Tesla K80, Tesla V100-PCIE-32GB, Tesla V100-SXM2-32GB. Each run uses one single GPU. The experiments involve 5 datasets OmniPrint-meta[1-5], 3 baseline methods (MAML, PyTorch, Naive), 4 settings (5-way-1-shot, 5-way-5-shot, 20-way-1-shot, 20-way-5-shot), 5 random seeds. The total amount of computation time is about 280 hours.

B.4 Hyperparameters

The baseline methods used the default or recommended hyperparameters of the original paper/code. A small number of hyperparameters *e.g.,* learning rates, were adjusted according to preliminary experiments. No large-scale hyperparameter optimization was performed.

While the full details are provided with the code, we highlight some important hyperparameters:

- **MAML** [6] 5 inner steps were used for meta-train, meta-validation and meta-test. The meta learner is optimized using Adam [15] with the learning rate 10^{-3}. The inner loops were optimized using SGD [23] with the learning rate 10^{-1}.
- **Prototypical Networks** [27] By following the original paper, each meta-train episode is a 60-way-K-shot regardless the meta-validation/meta-test setting. No learning rate decay was used. The backbone neural network was optimized using Adam [15] with the learning rate 5×10^{-4}.
- **Naive** The neural network for each meta-test episode was trained from scratch (random initialization) with 20 gradient steps. It was optimized using Adam [15] with the learning rate 10^{-4}.

B.5 Data synthesis

The background images used for OmniPrint-meta5 dataset were taken using a personal mobile phone.

C Fonts

Fonts are usually protected under their own licenses. We do not provide any warranty for this. Please be aware that some fonts cannot be redistributed or modified. This is the reason why we do not redistribute fonts with our code. However, we provide the font preparation scripts that we used. These fonts were downloaded from a manually-collected list of URLs.

We provide the font preparation scripts. If some URLs fail, please consider re-run the scripts at a later time (possibly related to network problems). If some URLs continue to fail, please contact the authors of this paper (via GitHub Issues page or via email: omniprint@chalearn.org). On the other hand, the users are free to collect their own set of fonts depending on their needs.

We gathered a list of URLs and prepared scripts which automatically download, filter and format the fonts. These scripts also record metadata of these fonts. The workflow of the font preparation scripts can be summarized into 2 stages:

- **Downloading** Download files from the given URLs, logs will be generated to keep track of potential failures. After unzipping, reformat file names which handles decoding error, converts file names to lower case, remove invalid symbols and translate Chinese file names. Generate metadata about sources of each font: some URLs contain several fonts, the same font can also be downloaded from different URLs.
- **Building** Filter out corrupted or unwanted fonts and move all font files to the dedicated directory. Move all license files to the dedicated directory. Build the so-called index files for each alphabet. Each alphabet has an index file which contains a list of fonts that support all of the characters it contains. Generate the lists of variable fonts and save the metadata of fonts *e.g.,* family name, style name, the range of possible stroke width (if any), etc. into a csv file.

Importing new fonts is easy in OmniPrint.

1. Move new fonts to the directory *fonts/fonts/*
2. Optionally, update the index file under the directory *fonts/index/* if users want to randomly select fonts
3. Optionally, update the metadata of fonts under the directory *fonts/metadata/*
4. Users should not forget to include license files in the directory *fonts/licenses/*

If users want to collect their own set of fonts, please be aware that some fonts can produce false rendering (empty image, square as a placeholder or even random symbols) without reporting any warnings or errors.

D Pre-rasterization transformations

The rendering process of modern digital fonts (TrueType/OpenType) is divided into two phases by the rasterization. Digital fonts are originally stored as anchor points expressed in font units within the EM square. Before being able to be rendered into bitmaps, the anchor points are scaled to be aligned with the device pixel grid. The grid-fitting (also called hinting) and rasterization are performed by the FreeType engine (Figure 1).

Pre-rasterization transformations refer to direct manipulation of the anchor points of the digital font files. Modern fonts (*e.g.,* TrueType or OpenType) are made of straight line segments and quadratic Bézier curves, connecting anchor points. OmniPrint uses the low-level Freetype font rasterization engine [31] (Python binding [22] which is under BSD license), which makes direct manipulation of anchor points possible. With pre-rasterization transformations, one can deform the characters without incurring aberrations due to aliasing and generate some local deformations that would be

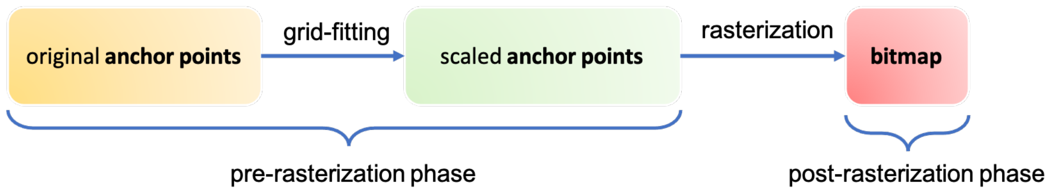

Figure 1: **Conversion process from TrueType/OpenType fonts to digital images.** In OmniPrint, pre-rasterization elastic transformation is performed on the original anchor points (yellow), linear transformations of anchor points are performed on the scaled anchor points (green).

difficult to achieve with post-rasterization transformations (digital image processing) *i.e.,* natural elastic transformation, variation of character proportion, structured deformation of specific characters, etc.

Algorithm 1: Pre-rasterization elastic transformation

Input: A sequence of characters S, a digital font F, a probability distribution D
Output: Rendered text image I

```
// C denotes characters, P denotes anchor points, the function load loads the
 initial anchor points of a digital font for a certain character.  The function
 enumerate returns the index as well as the value of an array.
// First pass to compute bounding box of the sequence
xmin, xmax, ymin, ymax = 0, 0, 0, 0
Initialize cache // In order to save random vibration
for C in S do
    for P in load(C, F) do
        xdelta ~ D
        ydelta ~ D
        P.x ← P.x + xdelta
        P.y ← P.y + ydelta
        cache.append( (xdelta, ydelta) )
        xmin, xmax, ymin, ymax ← update(xmin, xmax, ymin, ymax, P)
    end
end
I ← build_image(xmin, xmax, ymin, ymax)
// Second pass to render text
for i, C in enumerate(S) do
    for j, P in enumerate( load(C, F) ) do
        P.x ← P.x + cache[i][j][0]
        P.y ← P.y + cache[i][j][1]
    end
    I ← fill_image(I, C)
end
```

The implemented pre-rasterization transformations are listed as follows:

- **Elastic transformation (pre-rasterization)** corresponds to random vibration of independent anchor points. The pseudocode is shown in Algorithm 1. Of note is that elastic transformations are implemented in both pre-rasterization phase and post-rasterization phase, which can also be used together. All the elastic transformations mentioned in the main paper refer to pre-rasterization elastic transformation.
- **Stroke width variation** Variation of the stroke width *e.g.,* thinning or thickening of the strokes. Only variable fonts support stroke width variation, each variable font has its own continuous range of permissible stroke width.
- **Variation of character proportion** *e.g.,* variation of length of ascenders and descenders by some font units.

- **Linear transformations** Rotation, shear, scaling, stretch are assembled into a 2×2 matrix, see Equation 1. θ denotes the angle (in degree) of counter clockwise rotation, λ_1, λ_2 denote the shear parameters along horizontal axis and vertical axis respectively, s_1, s_2 denote the scaling (stretch) parameters along horizontal axis and vertical axis respectively. If $s_1 = s_2$, this corresponds to a scaling operation, otherwise this corresponds to a stretch operation along horizontal or vertical axes. The stretch along main diagonal axis and anti-diagonal axis by setting $\beta = \gamma \in \mathbb{R}$ or $\lambda_1 = \lambda_2 \in \mathbb{R}$ [26]. The four parameters $\alpha, \beta, \gamma, \delta$ allow inserting an arbitrary linear transform into the default linear transformation pipeline. Users are also allowed to directly set the values of a, b, d, e *i.e.,* the composed linear transformation matrix L.

$$
\begin{aligned}
L &= \begin{pmatrix} a & b \\ d & e \end{pmatrix} \\
&= \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & \lambda_1 \\ \lambda_2 & 1 \end{pmatrix} \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} \begin{pmatrix} s_1 & 0 \\ 0 & s_2 \end{pmatrix} \\
&= \begin{pmatrix} s_1((\alpha + \gamma\lambda_1)\cos\theta - (\alpha\lambda_2 + \gamma)\sin\theta) & s_2((\beta + \delta\lambda_1)\cos\theta - (\beta\lambda_2 + \delta)\sin\theta) \\ s_1((\alpha + \gamma\lambda_1)\sin\theta + (\alpha\lambda_2 + \gamma)\cos\theta) & s_2((\beta + \delta\lambda_1)\sin\theta + (\beta\lambda_2 + \delta)\cos\theta) \end{pmatrix}
\end{aligned}
\tag{1}
$$

In order to add new pre-rasterization transformations, users can edit the function *render_lt_text* in the script *freetype_text_generator.py*. More specifically, this function contains 2 passes over the sequence of characters to synthesize (a sequence containing a single character is a special case), the first pass computes the bounding box, the second pass performs the actual rendering. In each pass, users can loop over the anchor points of each character and perform the required transformations accordingly in the font unit space [3, 30, 1]. Algorithm 1 shows an example.

E Post-rasterization transformations

- **Translation** is performed, if any, when the foreground text is blended into the background.
- **Perspective transformations** can be used to imitate the effect of different camera viewpoints. A perspective transformation is generally parameterized by a 3×3 matrix in homogeneous coordinates. The homogeneous matrix coefficients are computed from 4 pairs of 2D points in the two projection planes by solving a linear system.
- **Morphological image processing** is a set of operations on the shape of the character and they operate on binary images (foreground vs background). In total, 7 morphological transformations are available via OpenCV [4]: morphological erosion, morphological dilation, morphological opening, morphological closing, morphological gradient, Top Hat, Black Hat.
  - **Morphological erosion** can be used to thin the stroke width in the post-rasterization phase. It erodes away the boundaries of foreground text and it can detach some previously connected strokes. The principle is to apply a 2D convolution, a pixel in the foreground text layer will be kept only if all the neighbor pixels are within the foreground area, otherwise it is eroded. The neighborhood is defined by a convolution kernel whose shape can be selected among rectangle, ellipse or cross-shaped.
  - **Morphological dilation** can be used to thicken the stroke width in the post-rasterization phase and join detached strokes, which is the opposite of morphological erosion. A pixel will be put into the foreground if at least one neighbor pixel is within the foreground area.
  - **Morphological opening** is the morphological erosion followed by morphological dilation. It can remove small pixel noises in the background, if any.
  - **Morphological closing** is the morphological dilation followed by the morphological erosion, which is the opposite of morphological opening. It can close small holes inside the foreground text, if any.
  - **Morphological gradient** is the difference between morphological dilation and morphological erosion of the input image. It can render hollow text in the post-rasterization phase.

  - **Top Hat** is the difference between the input image and the morphological opening of the input image.
  - **Black Hat** is the difference between the morphological closing of the input image and the input image.
- **Gaussian blur** is implemented using scikit-image [34]. In the synthesis pipeline, Gaussian blur is usually applied before downsampling to avoid aliasing.
- **Variation of contrast, brightness, color enhancement, sharpness** is implemented using Imgaug [14].
- **Elastic transformation (post-rasterization)** [25, 14] moves pixels locally around using displacement field. Depending on parameters, this transform can produce pixelated images or smooth deformation.
- **Foreground filling** Foreground text can be filled either by uniform color or by natural image/texture. The sampling distribution (Figure 2) of random color is from [36] (MIT License). When using random color for both foreground text and background, OmniPrint automatically ensures that foreground and background colors are visually distinguishable by thresholding the Delta E value (CIE2000). The computation of the Delta E value (CIE2000) is enabled by [29] (BSD-3-Clause License).
- **Text outline** can be generated and filled either by uniform color or by natural image/texture.
- **Background blending** can be done in two ways: (1) naively paste the foreground text onto the background while considering the mask; (2) Poisson Image Editing [21] which ensures seamless blending, this is particularly useful in case of natural background. The implementation is from [10], which is under Apache License 2.0. Background can be filled by uniform color, natural image/texture or uniform color augmented with a random regular polygon.

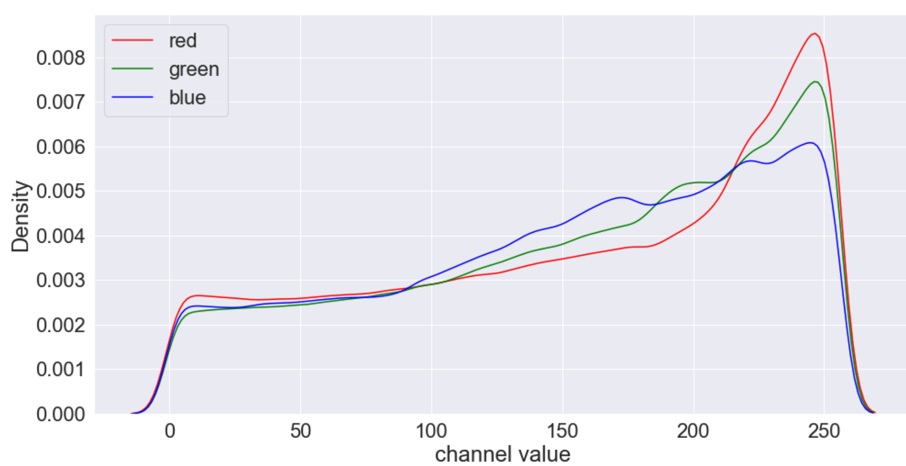

Figure 2: **Kernel density estimation of the marginal color distribution.** Each curve is the estimated distribution of one color channel.

New post-rasterization transformations can be added to the image synthesis pipeline. For example, if one wants to add a transformation called *my_transform*.

1. Create a Python script called *my_transform.py* under the directory *transforms*
2. Implement the desired functionalities in *my_transform.py*, which contains a function called *transform*. The first two positional parameters of the function *transform* should be the image and its corresponding mask (the mask is used for masking foreground text layer such that only the text itself will be pasted onto the background). The image is a RGB *PIL.Image.Image* object where text is black (0) and background is white (255). The mask is

a grayscale *PIL.Image.Image* object where text is white (255) and background is black (0). In principle, the mask should undergo the same operations as the image while taking into account the difference in image mode and black/white convention. The function *transform* can, of course, accept other parameters, which is usually the case. The output of the function *transform* is a tuple of size 2: the first is the transformed image, the second is the transformed mask.

3. Edit the script *__init__.py* under the directory *transforms*, add one line: *from transforms.my_transform import transform as my_transform*
4. Edit the script *data_generator.py* to insert the implemented transform at appropriate location. For example, *img, mask = my_transform(img, mask)*
5. It is recommended to edit the argument parsing function of the entry script *run.py*, which allows specifying parameters of the newly implemented transformation via command line. It is also recommended to wrap *img, mask = my_transform(img, mask)* under *data_generator.py* by something like *if args.get(my_transform) is not None:*, which allows to activate and deactivate the newly implemented transformation.

F Alphabets

Here we present the character selection criteria:

- For Latin script, we included basic uppercase and lowercase letters, all the variants in different European languages as well as the International Phonetic Alphabet. They are classified into basic Latin uppercase, basic Latin lowercase, Latin-1 Supplement, Latin Extended-A, Latin Extended-B, IPA letters and IPA for disordered speech and sinology, as defined in Unicode standard.
- Chinese characters, also known as CJK Unified Ideographs, are numerous and their usage in real life are extremely imbalanced. In consequence, we only included Chinese characters from Table of General Standard Chinese Characters [2]. These Chinese characters are divided into three levels containing 3500, 3000 and 1605 characters respectively. Characters in group 1 and 2 (the first 6500) are designated as common. Different from other writing systems, the distinction between simplified Chinese characters, traditional Chinese characters, Japanese Kanji and Korean Hanja is only handled by fonts in principle, because many of them share the same code points. The only way to distinguish them is the fonts' rendering. Generally, the fonts that were designed for simplified Chinese characters should never be used when rendering traditional Chinese text or Japanese text, and vice versa. Otherwise, it can be unintelligible or be unacceptable for native speakers. To avoid this overhead, we only aim to render simplified Chinese characters.
- For Japanese, all of Hiragana and Katakana are included. Note that each letter of these two scripts appears twice in the Unicode standard, one corresponds to the normal-sized version, the other is the smaller version. We only included the normal-sized versions.
- For Korean, there are up to 11172 unique syllabic blocks, we only included 2350 syllabic blocks which are assumed to be commonly used.
- All letters of Cyrillic script are not included. Only modern Russian alphabet is included, which consists of 66 upper case and lower case letters.
- Writing systems like Abjad (Arabic, Hebrew, etc.) and Abugida (Thai, Lao, Tibetan, Devanagari, Bengali, etc.) are only partly included. Typically, we only included consonants, independent vowels and digits of these languages. For these scripts (Khmer, Balinese, Bengali, Devanagari, Gujarati, Myanmar, Oriya, Sinhala, Tamil, Telugu, Tibetan, Thai and Lao.), dependent vowel signs were excluded, independent vowels were included if there are any.
- Even though the Mongolian script has been adapted to write languages such as Oirat and Manchu, we only included basic Mongolian letters and Mongolian digits.
- For the Arabic script, we only included the 29 Arabic letters. For the Hebrew script, we only included the 27 Hebrew letters.
- All of the Ethiopic syllables available in the Unicode standard are included.

- Common punctuations and symbols, ASCII digits, some musical symbols and some mathematical operators are also included. However, neither of the collected fonts fully support these musical symbols.

G Accessibility

The NeurIPS foundation shall not bear any responsibility. The diffusion of the code and data will be done by the authors, who will be responsible for maintaining them and resolving any dispute.

- The code of the OmniPrint data synthesizer will be made available on Github under an open source MIT license https://opensource.org/licenses/MIT. A specific guarantee disclaimer is associated with the license: `THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.`
- The datasets OmniPrint-meta[1-5] will be distributed via the UCI repository and/or Kaggle datasets. They will be licensed under a Creative Commons license CC BY 4.0 https://creativecommons.org/licenses/by/4.0/. This comes with the following guarantee disclaimer: `Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors, whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.`

In general, modern digital fonts are protected under their own licenses, we do not provide any warranty for this. Some fonts cannot be redistributed or modified. However, the users are free to collect or make their own fonts.

The code[1] and datasets[2] have been publicly released after the NeurIPS 2021 meta-learning challenge. The hosting platform of the datasets OmniPrint-meta[1-5] is Kaggle Datasets, DOI for datasets is 10.34740/kaggle/dsv/2763401, metadata is accessible on the dataset hosting page.

Kaggle Datasets make data available for an unlimited time period. The authors will verify that the data are properly accessible for at least three years and change venue in case of a problem. Likewise GitHub has no time limitations in terms of code hosting. The authors will maintain the code and address issues for at least three years. Users will be encouraged to post GitHub issues in case of problems and/or make pull requests.

Any information and updates regarding to the **release** and necessary **maintenance** will be communicated via the README of https://github.com/SunHaozhe/OmniPrint-datasets.

[1]https://github.com/SunHaozhe/OmniPrint

[2]https://github.com/SunHaozhe/OmniPrint-datasets

Table 1: **Unsupervised domain adaptation results** on OmniPrint-metaX-31. meta$A \rightarrow$ meta$B$ means the source domain is OmniPrint-meta$A$, the target domain is OmniPrint-meta$B$, where $A, B \in 3, 4, 5$. The 95% confidence intervals are computed with 8 random seeds.

	meta3→meta4	meta4→meta3	meta3→meta5	meta5→meta3	meta4→meta5	meta5→meta4
DAN [19, 32]	18.0 ± 2.4	3.2 ± 0.0	25.8 ± 1.7	3.5 ± 0.3	10.1 ± 16.0	10.7 ± 16.5
DANN [7]	72.2 ± 2.8	96.8 ± 0.5	65.6 ± 2.9	82.2 ± 2.7	79.8 ± 1.2	81.5 ± 2.1
DeepCoral [28]	22.9 ± 2.5	84.6 ± 1.5	28.6 ± 1.7	69.6 ± 2.5	57.0 ± 1.3	60.2 ± 1.0
DAAN [37]	22.3 ± 1.8	84.5 ± 2.1	25.1 ± 1.7	59.9 ± 5.9	50.9 ± 1.5	53.3 ± 2.3
DSAN [38]	$\mathbf{79.3 \pm 2.3}$	$\mathbf{96.9 \pm 0.3}$	$\mathbf{66.4 \pm 2.5}$	$\mathbf{93.5 \pm 0.8}$	$\mathbf{80.5 \pm 1.0}$	$\mathbf{82.8 \pm 1.9}$
Average	42.9	73.2	42.3	61.7	55.7	57.7
Median	22.9	84.6	28.6	69.6	57.0	60.2

Each dataset synthesized by OmniPrint shares the same folder structure. It contains two subfolders, the subfolder *data* contains the images in png format, the subfolder *label* contains a csv file, called *raw_labels.csv*, which stores the label (character class) as well as all the metadata of each image instance. The columns of *raw_labels.csv* may vary depending on involved transformations, the common columns include *image_name* which specifies which image instance this record is about, *text* which contains the rendered character to synthesize, *unicode_code_point* contains the Unicode code point (integer) of the character to synthesize, *font_file* which indicates the involved digital font, *background* which specifies which type of background is being used, *font_weight* which specifies the stroke width, *margin_bottom, margin_left, margin_right, margin_top* which indicate the proportion of each margin in the image and facilitate the construction of bounding boxes, *family_name, style_name* which show the family and font style to which the digital font belongs, etc.

The user manual of the data synthesizer OmniPrint and an example dataloader for the datasets OmniPrint-meta[1-5] are provided with the code.

H Experimental details of domain adaptation

This section provides the experimental details of Section 4.4 of the main paper.

H.1 Unsupervised domain adaptation methods

The 5 unsupervised domain adaptation algorithms are DAN [19, 32], DANN [7], DeepCoral [28], DAAN [37] and DSAN [38]. The implementation is from DeepDA [35] which is under MIT License.

H.2 Hyperparameters and compute resources

For each combination of task and algorithm, we run 10 epochs with 8 random seeds to get the confidence interval. The 8 random seeds were fixed in advance. The backbone neural network is Resnet50 [11]. The model is optimized using SGD [23] with 10^{-3} as the learning rate. No hyperparameter optimization was performed. The other experimental details are provided with the code at https://github.com/SunHaozhe/transferlearning. The experiments were run on an internal cluster with Tesla V100-PCIE-32GB, Tesla V100-SXM2-32GB. The total amount of computation time is about 182 hours.

The results are available in Table 1.

H.3 Unsupervised domain adaptation from Fake-MNIST to MNIST

We used OmniPrint to generate a dataset, called Fake-MNIST, which is similar to MNIST [17] and performed the unsupervised domain adaptation (the 5 DeepDA methods [35], see Appendix H.1) from Fake-MNIST to MNIST.

Only the test set of MNIST is involved in this experiment, which consists of 10000 images for the 10 digits. Fake-MNIST contains 3000 white-on-black character images for each of the 10 digits. Random pre-rasterization elastic transformation, horizontal shear, rotation and translation were used to synthesize Fake-MNIST. Figure 3 shows some example images from Fake-MNIST.

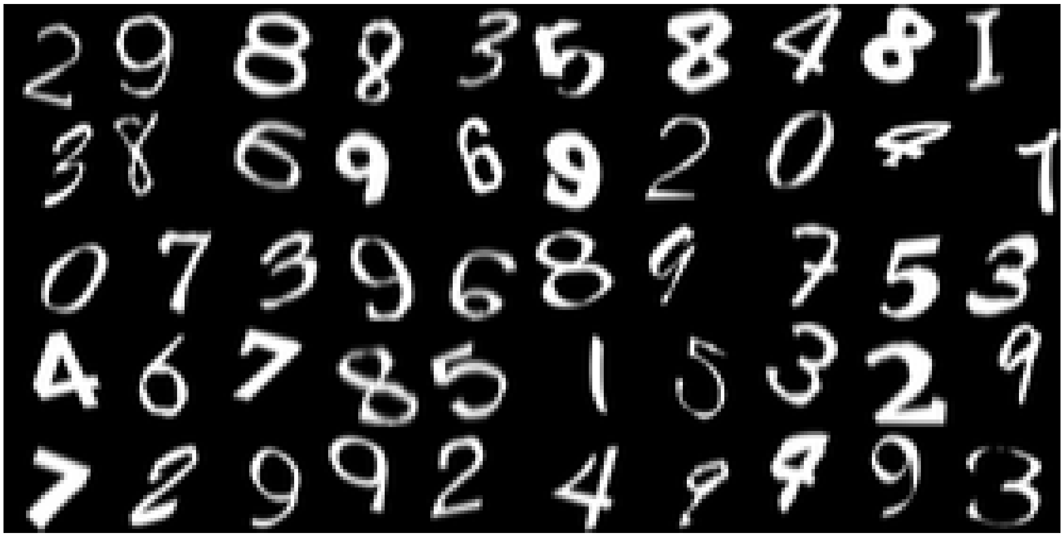

Figure 3: **Example images from Fake-MNIST.** Random pre-rasterization elastic transformation, horizontal shear, rotation and translation were used.

Table 2: **Unsupervised domain adaptation from Fake-MNIST to MNIST**. 95% confidence intervals are computed with 27 random seeds.

	DAN	DANN	DeepCoral	DAAN	DSAN	Average	Median
Fake-MNIST $\rightarrow$ MNIST	94.8 ± 0.1	98.0 ± 0.1	92.4 ± 0.2	93.3 ± 0.2	$\mathbf{98.2 \pm 0.1}$	95.34	94.8

While the synthesis parameters of Fake-MNIST were not optimized, the performance of the 5 unsupervised domain adaptation methods (Table 2) ranges from 92 to 98% accuracy, which is very honorable (current supervised learning results on MNIST are over 99%).

I Experimental details of few-shot learning experiments with metadata-based episodes

This section provides the experimental details of Section 4.2 of the main paper.

I.1 Metadata-based episode generation algorithm

The metadata-based episode generation algorithm is illustrated in Algorithm 2.

I.2 Data, hyperparameters and compute resources

The experiments used the same hyperparameters as Appendix B. The same character split was used (900 characters for meta-train, 149 characters for meta-validation, 360 characters for meta-test). The experiments were trained for 300 epochs, where each epoch is defined to be 6 batches of episodes, each batch contains 32 episodes. During meta-training, the model checkpoints were evaluated on meta-validation episodes every 5 epochs. Only the checkpoint having the highest accuracy on meta-validation episodes during training is selected to be tested on meta-test episodes. The backbone neural network is the concatenation of three modules of Convolution-BatchNorm-Relu-Maxpool. The reported accuracy and 95% confidence intervals were computed with 5 random seeds.

The experiments were run on an internal cluster, the involved GPUs are GeForce RTX 2080 Ti, Tesla V100-PCIE-32GB and Tesla V100-SXM2-32GB. The total amount of computation time is about 164 hours.

Algorithm 2: Metadata-based few-shot learning episode generation.

Input: Number of support images S, number of query images Q

`// Assuming that metadata consists of real numbers.`

for each episode **do**
- Randomly sample N classes $c_1, c_2, ..., c_N$
- **for** each class c_n **do**
  - Find all examples $E_{c_n} = \{e_1, e_2, ...\}$ of class c_n, the metadata m_i of each example $e_i \in E_{c_n}$ is a real-valued vector.
  - Compute the bounding box B_{c_n} of the metadata vectors m_i.
  - Randomly sample a centroid D within B_{c_n}.
  - Select the $(S + Q)$ nearest neighbors $M = \{m_x, m_y, ..., m_{(S+Q)}\}$ from all the metadata vectors $m_1, m_2, ...$
  - An example e_i is selected to be part of the episode if and only if $m_i \in M$, all the selected examples form the set $\hat{E}_{c_n,D}$
  - Randomly draw S examples from $\hat{E}_{c_n,D}$ to form the support set, the remaining examples serve as the query set.
- **end**

end

J Experimental details of the investigation of the influence of the number of meta-training episodes

This section provides the experimental details of Section 4.3 of the main paper.

J.1 Data

For this experiment, we generated a larger version of OmniPrint-meta3. It has the same synthesis parameters as OmniPrint-meta3 but has 200 images per class (OmniPrint-meta3 has 20 images per class).

J.2 Hyperparameters and compute resources

The experiments used the same hyperparameters as Appendix B. The same character split was used (900 characters for meta-train, 149 characters for meta-validation, 360 characters for meta-test). During meta-training, the model checkpoints were evaluated on meta-validation episodes every 960 episodes and at the end of meta-training. Only the checkpoint having the highest accuracy on meta-validation episodes during training is selected to be tested on meta-test episodes. The backbone neural network is the concatenation of three modules of Convolution-BatchNorm-Relu-Maxpool. The reported accuracy and 95% confidence intervals were computed with 5 random seeds.

The experiments were run on an internal cluster, the involved GPUs are GeForce RTX 2080 Ti, Tesla V100-PCIE-32GB. The total amount of computation time is about 80 hours.

K Experimental details of the regression task

This section provides the experimental details of Section 4.5 of the main paper.

K.1 Data

We generated two large datasets which are slightly easier than OmniPrint-meta3. Both datasets contain black-on-white characters (1409 characters with 200 images each). The first dataset has horizontal shear (horizontal shear parameter ranges from -0.8 to 0.8) but not rotation, the second dataset has rotation (rotation ranges from -60 degrees to 60 degrees) but not horizontal shear. Perspective transformations are not used. Some sample images are shown in Figure 4 and Figure 5. Each of the two generated datasets have 281800 images in total. 20% of the images (56360) were used for

Table 3: **Regression results**. The reported metric is the coefficient of determination R^2. 1.69×10^2, 1.69×10^3, 1.69×10^4 and 1.69×10^5 are the number of training images. 95% confidence intervals are computed with 3 random seeds.

	Backbone	1.69×10^2	1.69×10^3	1.69×10^4	1.69×10^5
Horizontal shear	small	0.3 $\pm$ 0.2	0.6 $\pm$ 0.0	0.8 $\pm$ 0.1	0.9 $\pm$ 0.0
	resnet18	-0.1 $\pm$ 0.2	0.6 $\pm$ 0.0	0.8 $\pm$ 0.0	0.9 $\pm$ 0.0
Rotation	small	-25.3 $\pm$ 50.4	-1.8 $\pm$ 0.4	-0.8 $\pm$ 1.7	0.3 $\pm$ 0.2
	resnet18	-1002.0 $\pm$ 3164.1	-0.9 $\pm$ 0.2	0.1 $\pm$ 0.1	0.5 $\pm$ 0.0

validation, 20% of the images (56360) were used for test. The remaining 169080 images were used for training.

Figure 4: **Shear dataset.** Horizontal shear parameter ranges from -0.8 to 0.8. Rotation and perspective transformations are not used.

Figure 5: **Rotation dataset.** Rotation angle ranges from -60 degrees to 60 degrees. Horizontal shear and perspective transformations are not used.

K.2 Hyperparameters and compute resources

We tested two neural networks. The first one, referred to as "small", is the concatenation of three modules of Convolution-BatchNorm-Relu-Maxpool, followed by a fully-connected layer within a scalar output. It contains 76097 trainable parameters. The second one is Resnet18 [11] pretrained on ImageNet [24]. We only train the last convolution layer and fully-connected layer of Resnet18 [11], it thus has 2360833 trainable parameters. The neural networks were optimized with MSE loss for 30 epochs using SGD [23], the initial learning rate was 10^{-3}, which is reduced by a factor of 10 when the validation loss has stopped decreasing for 5 epochs. The weight decay was 10^{-4}. The momentum was 0.9. Only the model having the highest accuracy on validation data during training is selected to be tested on test data. The 95% confidence intervals are computed with 3 random seeds.

The experiments were run on an internal cluster with GeForce RTX 2080 Ti. The total amount of computation time is about 48 hours.

The detailed results are reported in Table 3.

References

[1] Opentype font variations overview (opentype 1.8.4) - typography | microsoft docs. `https://docs.microsoft.com/en-us/typography/opentype/spec/otvaroverview`. (Accessed on 11/28/2020).

[2] Table of general standard chinese characters. `http://hanzidb.org/character-list/general-standard`. (Accessed on 12/21/2020).

[3] Truetype fundamentals (opentype 1.8.4) - typography | microsoft docs. `https://docs.microsoft.com/en-us/typography/opentype/spec/ttch01`. (Accessed on 11/22/2020).

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] Daniele Ciriello. orobix/prototypical-networks-for-few-shot-learning-pytorch: Implementation of prototypical networks for few shot learning (https://arxiv.org/abs/1703.05175) in pytorch. `https://github.com/orobix/Prototypical-Networks-for-Few-shot-Learning-PyTorch`. (Accessed on 06/04/2021).

[6] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *arXiv:1703.03400 [cs]*, July 2017.

[7] Yaroslav Ganin and Victor Lempitsky. Unsupervised Domain Adaptation by Backpropagation. *arXiv:1409.7495 [cs, stat]*, February 2015.

[8] Edward Grefenstette, Brandon Amos, Denis Yarats, Phu Mon Htut, Artem Molchanov, Franziska Meier, Douwe Kiela, Kyunghyun Cho, and Soumith Chintala. higher/maml-omniglot.py at master · facebookresearch/higher. `https://github.com/facebookresearch/higher/blob/master/examples/maml-omniglot.py`. (Accessed on 06/04/2021).

[9] Edward Grefenstette, Brandon Amos, Denis Yarats, Phu Mon Htut, Artem Molchanov, Franziska Meier, Douwe Kiela, Kyunghyun Cho, and Soumith Chintala. Generalized Inner Loop Meta-Learning. *arXiv:1910.01727 [cs, stat]*, October 2019.

[10] Ankush Gupta, Andrea Vedaldi, and Andrew Zisserman. Synthetic data for text localisation in natural images. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015.

[12] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-Learning in Neural Networks: A Survey. *arXiv:2004.05439 [cs, stat]*, November 2020.

[13] Morris A. Jette, Andy B. Yoo, and Mark Grondona. Slurm: Simple linux utility for resource management. In *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*, pages 44–60. Springer-Verlag, 2002.

[14] Alexander B. Jung, Kentaro Wada, Jon Crall, Satoshi Tanaka, Jake Graving, Christoph Reinders, Sarthak Yadav, Joy Banerjee, Gábor Vecsei, Adam Kraft, Zheng Rui, Jirka Borovec, Christian Vallentin, Semen Zhydenko, Kilian Pfeiffer, Ben Cook, Ismael Fernández, François-Michel De Rainville, Chi-Hung Weng, Abner Ayala-Acevedo, Raphael Meudec, Matias Laporte, et al. imgaug. `https://github.com/aleju/imgaug`, 2020. Online; accessed 01-Feb-2020.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017.

[16] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, December 2015.

[17] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[18] Liangqu Long. Maml-pytorch implementation. `https://github.com/dragen1860/MAML-Pytorch`, 2018.

[19] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I. Jordan. Learning Transferable Features with Deep Adaptation Networks. *arXiv:1502.02791 [cs]*, May 2015.

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

[21] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Trans. Graph.*, 22(3):313–318, July 2003.

[22] Nicolas P. Rougier. rougier/freetype-py: Python binding for the freetype library. `https://github.com/rougier/freetype-py`. (Accessed on 06/04/2021).

[23] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv e-prints*, page arXiv:1609.04747, September 2016.

[24] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *arXiv e-prints*, page arXiv:1409.0575, September 2014.

[25] P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pages 958–963, August 2003.

[26] Patrice Y. Simard, Yann A. LeCun, John S. Denker, and Bernard Victorri. Transformation Invariance in Pattern Recognition — Tangent Distance and Tangent Propagation. In Genevieve B. Orr and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, pages 239–274. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[27] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical Networks for Few-shot Learning. *arXiv:1703.05175 [cs, stat]*, June 2017.

[28] Baochen Sun and Kate Saenko. Deep CORAL: Correlation Alignment for Deep Domain Adaptation. *arXiv:1607.01719 [cs]*, July 2016.

[29] Gregory Taylor. gtaylor/python-colormath: A python module that abstracts common color math operations. for example, converting from cie l*a*b to xyz, or from rgb to cmyk. `https://github.com/gtaylor/python-colormath`. (Accessed on 06/06/2021).

[30] David Turner, Robert Wilhelm, Werner Lemberg, Alexei Podtelezhnikov, Toshiya Suzuki, Oran Agra, Graham Asher, David Bevan, Bradley Grainger, Infinality, Tom Kacvinsky, Pavel Kaňkovský, Antoine Leca, Just van Rossum, and Chia-I Wu. Freetype glyph conventions / vi. `https://www.freetype.org/freetype2/docs/glyphs/glyphs-6.html`. (Accessed on 04/20/2021).

[31] David Turner, Robert Wilhelm, Werner Lemberg, Alexei Podtelezhnikov, Toshiya Suzuki, Oran Agra, Graham Asher, David Bevan, Bradley Grainger, Infinality, Tom Kacvinsky, Pavel Kaňkovský, Antoine Leca, Just van Rossum, and Chia-I Wu. The freetype project. `https://www.freetype.org/index.html`. (Accessed on 11/25/2020).

[32] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep Domain Confusion: Maximizing for Domain Invariance. *arXiv:1412.3474 [cs]*, December 2014.

[33] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching Networks for One Shot Learning. *arXiv:1606.04080 [cs, stat]*, December 2017.

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

[35] Jindong Wang and Wenxin Hou. Deepda: Deep domain adaptation toolkit. `https://github.com/jindongwang/transferlearning/tree/master/code/DeepDA`.

[36] Kevin Wu. kevinwuhoo/randomcolor-py: A port of david merfield's randomcolor to python. `https://github.com/kevinwuhoo/randomcolor-py`. (Accessed on 06/06/2021).

[37] Chaohui Yu, Jindong Wang, Yiqiang Chen, and Meiyu Huang. Transfer Learning with Dynamic Adversarial Adaptation Network. *arXiv:1909.08184 [cs, stat]*, September 2019.

[38] Yongchun Zhu, Fuzhen Zhuang, Jindong Wang, Guolin Ke, Jingwu Chen, Jiang Bian, Hui Xiong, and Qing He. Deep Subdomain Adaptation Network for Image Classification. *IEEE Transactions on Neural Networks and Learning Systems*, 32(4):1713–1722, April 2021.