Input Selection for Bandwidth-Limited Neural Network Inference*

Stefan Oehmcke† Fabian Gieseke‡†

Abstract

Data are often accommodated on centralized storage servers. This is the case, for instance, in remote sensing and astronomy, where projects produce several petabytes of data every year. While machine learning models are often trained on relatively small subsets of the data, the inference phase typically requires transferring significant amounts of data between the servers and the clients. In many cases, the bandwidth available per user is limited, which then renders the data transfer to be one of the major bottlenecks. In this work, we propose a framework that automatically selects the relevant parts of the input data for a given neural network. The model as well as the associated selection masks are trained simultaneously such that a good model performance is achieved while only a minimal amount of data is selected. During the inference phase, only those parts of the data have to be transferred between the server and the client. We propose both instance-independent and instance-dependent selection masks. The former ones are the same for all instances to be transferred, whereas the latter ones allow for variable transfer sizes per instance. Our experiments show that it is often possible to significantly reduce the amount of data needed to be transferred without affecting the model quality much.

1 Introduction.

The data volumes have increased dramatically in various domains. Often, centralized storage servers/clusters are used to accommodate the collected data, which are then accessed by many users. This is the case in remote sensing, where current projects produce petabytes of satellite data every year [19,31]. The application of a machine learning model in this field to, e. g., monitor changes on a global scale or to search for objects, often requires "scanning" all the data and, thus, induces the transfer of large amounts of data between the server and the client that executes the model [29]. Similar data-intensive scenarios can be found in other disciplines as well. For instance, astronomers also resort to centralized storage servers such as the ones for the Sloan Digital Sky Survey [1]. The data transfer between such servers and (thousands of) clients is already restricted today (e. g., via a limited bandwidth per user) and will become a serious bottleneck for projects such as the Large Synoptic Sky Survey [13] or the Square Kilometre Array [9], which will be fully operational within the next few years and which will yield petabytes of data every month.

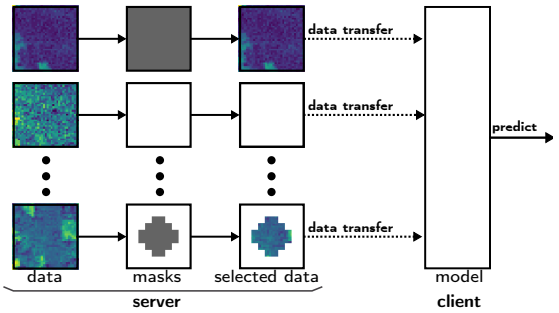


Figure 1: End-to-end training of input selection masks and a task model to achieve both, a minimal amount of data selected by the masks as well as a good model performance. During the inference phase, only the selected parts have to be transferred.

While the reduction of the training and inference runtimes have received considerable attention [5, 8, 11, 16, 25, 34], comparatively little work has been done regarding the transfer of data induced by such server/client based scenarios. In this work, we propose a framework that allows to automatically select those parts of the input data that are relevant for a given task and an associated model. In particular, we aim at scenarios, in which large amounts of data reside on a public storage server and where it is, in general, *not* possible for the user to execute code on the server side. Our framework allows to learn masks that are adapted to the specific transfer capabilities offered by the server (e. g., if the server permits to select only certain channels or parts of the images), which can then be used to significantly reduce the amount of data needed to be transferred.[1] The masks as well as the model are

*The code is available at `https://github.com/StefOe/selection-masks`

†Department of Computer Science, University of Copenhagen

‡Department of Information Systems, University of Münster

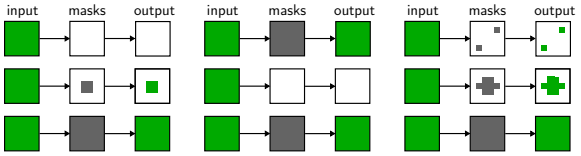
[1]For instance, the *Planet API* allows to select input channels or to "clip" data before transmission.

optimized simultaneously in an end-to-end way to achieve both a minimal amount of data being selected by the masks and a good model performance, see Figure 1. During the inference phase, only the selected parts have to be transferred. In addition to such "static" masks, we also consider scenarios where reduced versions of the data (e. g., thumbnails) are available on the server side that serve as basis for a dynamic selection of relevant input data for a given instance. Our experiments show that both the static and dynamic selection masks can be used to significantly reduce the amount of data that must be transferred during the inference phase without sacrificing much of the model performance.

2 Background.

The transfer of data during the inference phase is an active field of research [3, 18, 23, 25, 30, 33]. Two different lines of approaches exist: (a) feature extraction, where the original features get modified, and (b) feature selection, where a subset of the original features is chosen. In this work, we consider feature selection scenarios, where the user can select and slice the data, but is not able to perform any computations on the server side, which excludes the use of feature extraction schemes. For example, a neural network cannot be applied on the server. Recently, unsupervised approaches based on autoencoders have been proposed that aim at selecting a pre-defined number of relevant input pixels. However, these methods loose the spatial information present in the data during the selection process [3, 10]. LassoNet [18] has been proposed as a supervised and unsupervised feature selection scheme. It is, however, only applicable to fully-connected networks and not to the more prominent convolutional networks.

We conduct a gradient-driven search to find suitable weight assignments for the selection masks (defined below). As detailed below, an exhaustive search for finding an optimal feature selection is computationally intractable. An alternative to our approach are greedy schemes that, e. g., incrementally select input channels or pixels (i. e., similar to forward/backward feature selection methods). However, these approaches also quickly become computationally infeasible in case many features (e. g., pixels or channels) are given. Another approach is based on iteratively selecting features according to a pre-calculated ranking, for instance based on the feature importance values induced by random forests [4, 7] or based on a principal feature analysis [22]. However, these methods neglect the feature structure, in particular spatial correlations, which is vital for many tasks. In addition, they usually yield sub-optimal



(a) `block` (b) `channel` (c) `pixel`

Figure 2: Proposed masks to select data. While the final masks are discrete, differentiable surrogates are used during the training process.

results since they are not trained simultaneously with the task model in an end-to-end fashion.

Our framework is different from these works in the sense that we conduct supervised search with a more general and flexible class of selection schemes (e. g., channel-, block-, or pixel-wise selection, see below). Furthermore, instead of fixing the number of selected features beforehand, our approach iteratively reduces the number of features through gradient information updates, which provide more choices for the trade-off between accuracy and transfer cost. There is also no restriction to specific network architectures (e.g., fully-connected networks), meaning any architecture can be used after the selection. In addition to static scenarios, our approach allows dynamic selection of required input data per instance, which is a novel approach.

3 Automatic Input Selection.

The goal of our framework is to reduce the amount of data that needs to be transferred in order to apply a given neural network model for a specific task. For the sake of simplicity, we focus on image data and classification in this work. Our approach can, however, also be applied to other types of data such as video, time series, or unstructured data as well as other types of tasks, such as regression or segmentation.

3.1 Input Selection Masks.

Let $\boldsymbol{m}^D$ be a *selection mask* that allows to choose certain parts of an image $\boldsymbol{x} \in \mathbb{R}^{w \times h \times k}$ with width w, height h, and number of channels k, such as specific input channels or individual pixels, see Figure 2. The generic selection scheme presented here is `block` selection, where the input data are divided into "blocks". A partition in $b_w \times b_h$ blocks is achieved via a mask of the form $\boldsymbol{m}^D \in \{0,1\}^{b_w \times b_h \times k}$. As an example, a mask $\boldsymbol{m}^D$ with $\boldsymbol{m}^D_{[2,1,3]} = 1$ would correspond to selecting the second block in the first row of the third channel, whereas $\boldsymbol{m}^D_{[2,2,1]} = 0$ would correspond to deselecting the second block in the second row of the first channel. The case $b_w = w$

and $b_h = h$ yields `pixel` selection masks of the form $\boldsymbol{m}^D \in \{0,1\}^{w \times h \times k}$, which allow to select individual pixels per channel. Furthermore, the case $b_w = 1$ and $h_h = 1$ yields `channel` selection masks, where a mask $\boldsymbol{m}^D \in \{0,1\}^{1 \times 1 \times k}$ allows to select specific channels of the image $\boldsymbol{x}$.

3.2 Learning Static Masks.

Training such a discrete mask that is well-suited for a given network and all the instances available for a specific task can be challenging. In case a single mask is applied for all instances, we call the mask *static* (since it remains the same regardless of the particular input image). Let $T = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\} \subset X \times Y$ be a training set consisting of images $\boldsymbol{x}_i \in X = \mathbb{R}^{w \times h \times k}$ with associated class labels $y_i \in Y = \{1, \ldots, C\}$, where C is the number of classes. Masking is implemented by element-wise multiplication $\boldsymbol{m}^D \odot \boldsymbol{x}$ of the mask $\boldsymbol{m}^D$ with a given input image $\boldsymbol{x}$, which sets deselected blocks to zero. In order to apply this step, we need to ensure that if the input image $\boldsymbol{x}$ and the corresponding mask $\boldsymbol{m}^D$ have different shapes (i.e., $b_w \neq w$ or $b_h \neq h$), the first two axes are broadcasted (via nearest neighbor interpolation), so it always yields a mask $\boldsymbol{m}^D \in \{0,1\}^{w \times h \times k}$ with the same shape as the input.

The goal of the training process is to find suitable weight assignments for both, the selection mask $\boldsymbol{m}^D$ and the neural network $f : X \to Y$ that is being considered for the task at hand. Simultaneous training of mask and network is achieved by minimizing the loss function

$$\mathcal{L}_{f,\boldsymbol{m}^D}(\hat{y}, y) = \mathcal{L}_f(\hat{y}, y) + \lambda \cdot \mathcal{Q}\left(\boldsymbol{m}^D\right), \qquad (1)$$

where $\hat{y} = f(\boldsymbol{m}^D \odot \boldsymbol{x})$ is the prediction for a given image $\boldsymbol{x}$ with associated class label y, $\mathcal{L}_f$ a user-defined task/model loss function (e.g., cross-entropy), and $\mathcal{Q}$ a *mask loss* that penalizes selections made by the mask $\boldsymbol{m}^D$. For the mask loss, various choices are possible. In this work, we consider the standardized L1-loss:

$$\mathcal{Q}\left(\boldsymbol{m}^D\right) = \frac{\sum_{i=1}^{b_w} \sum_{j=1}^{b_h} \sum_{l=1}^{k} \boldsymbol{m}^D_{[i,j,l]}}{b_w b_h k} \qquad (2)$$

It's constant gradient helps to gradually deactivate mask entries. The parameter $\lambda \in \mathbb{R}^+$ increases the weight of $\mathcal{Q}$, which we adapt heuristically to overcome stagnation (see Section 3.2.3). This mask loss is a direct representation of how much data is selected and needs to be transferred (e.g., 0.25 is 25% of the data).

3.2.1 Optimizing Discrete Masks.

Naturally, search schemes that aim at finding optimal discrete weight assignments for the mask w.r.t. $\mathcal{L}_{f,\boldsymbol{m}^D}$ by testing all possible assignments are computationally infeasible. Simple greedy approaches such as forward/backward selection of channels become computationally very demanding and are, thus, generally ill-suited (especially for pixel- or block-wise selection). While adapting the selection mask $\boldsymbol{m}^D$ directly via gradient descent is, in general, possible, doing so yields imprecise gradient information due to its binary value space.

We therefore introduce a mask model $m : X \to X$ with a trainable weight matrix $\boldsymbol{m} \in \mathbb{R}^{b_w \times b_h \times k}$ that outputs the selected input $\hat{\boldsymbol{x}}$. In the static case considered so far, the associated weights of the model m are independent of the particular (image) instance; for the dynamic case, the weights will depend on the particular instances, see Section 3.3. In both cases, the model output is given by

$$m(\boldsymbol{x}) = \boldsymbol{m}^D \odot \boldsymbol{x} = \lfloor \sigma(\boldsymbol{m}) \rceil \odot \boldsymbol{x} \qquad (3)$$

during the forward pass of the training process, where the rounding operator $\lfloor \cdot \rceil$ discretizes the weights (e.g., $\lfloor 0.2 \rceil = 0$ and $\lfloor 0.8 \rceil = 1$) and where $\sigma : \mathbb{R} \to \mathbb{R}$ is the sigmoid function with $\sigma(z) = \frac{1}{1+\mathrm{e}^{-z}}$ that is applied in an element-wise fashion to the weight vector $\boldsymbol{m}$. For the backward pass, a real-valued surrogate is used to obtain a continuous local gradient for the model m (that depends on the trainable weight vector $\boldsymbol{m}$). This surrogate ignores the rounding operator (since it is not differentiable) and only resorts to the derivative of the sigmoid function, i.e.,

$$\frac{d\, m}{d\, m_j} = \frac{\mathrm{e}^{-m_j}}{(1+\mathrm{e}^{-m_j})^2} x_j, \qquad (4)$$

for the individual weights m_j of the weight vector $\boldsymbol{m}$. This enables effective backpropagation since the sigmoid function is a differentiable function that can be used as surrogate for the rounding operator.[2]

3.2.2 Training.

Our training procedure for learning a suitable mask and network model is given by `LearnMasks` in Algorithm 1: The weights associated with

[2] More precisely, one can consider $\bar{\sigma}(z) = \sigma(\frac{z}{\tau})$, which approximates the rounding operator $\lfloor \cdot \rceil$ for $\tau \to 0$. This approximation of the non-differentiable function is similar to the one used in [24]; no random noise has to be added in this case though. There are also parallels to straight through estimators used in model pruning and compression [6, 12, 35], but instead of using an identity function and clipping the gradient to be between 0 and 1, we utilize Equation 4. For all experiments reported in this work, we used $\tau = 1$, i.e., a normal sigmoid function was used as approximation.

Algorithm 1: `LearnMasks`(f, m, T)

Input: neural network f, mask model m, and training set T

1 $\boldsymbol{m} \leftarrow$ `InitAllMasks()`
2 $\lambda \leftarrow$ `InitLambda()`
3 **for** $i \leftarrow 1$ **to** n_{epoch} **do**
4 **for** $j \leftarrow 1$ **to** n_{batch} **do**
5 $\boldsymbol{x}, y \leftarrow$ `GetBatch`(T)
6 $\hat{\boldsymbol{x}} \leftarrow m(\boldsymbol{x})$
7 $\hat{y} \leftarrow f(\hat{\boldsymbol{x}})$
8 $\mathcal{L}_{f,\boldsymbol{m}^D} \leftarrow \mathcal{L}_f(\hat{y}, y) + \lambda \mathcal{Q}(\boldsymbol{m}^D)$
9 $f, m \leftarrow$ `Optimize` $(f, m, \mathcal{L}_{f,\boldsymbol{m}^D})$
10 $\lambda \leftarrow$ `AdaptLambda`$(\lambda, \mathcal{Q}(\boldsymbol{m}^D))$
11 `ModelCheckpoint`$(f, m, \mathcal{Q}(\boldsymbol{m}^D))$

the mask model as well as the trade-off parameter λ are initialized in Line 1 and 2, respectively. Both the mask model m and the network f are trained simultaneously by iterating over a pre-defined number n_{epoch} of epochs, each being split into n_{batch} mini-batches (for the sake of clarity, we consider a batch size of 1). For each batch, a discrete mask $\boldsymbol{m}^D$ is computed and element-wisely multiplied ($\odot$) with the input $\boldsymbol{x}$ via the mask model m to return the masked image $\hat{\boldsymbol{x}}$, see again Equation (3). The induced prediction $\hat{y}$ is then used to compute the task loss $\mathcal{L}_f(\hat{y}, y)$. Both, the task loss $\mathcal{L}_f(\hat{y}, y)$ and the mask loss $\mathcal{Q}(\boldsymbol{m}^D)$ are optimized simultaneously via the procedure `Optimize` in Line 9 using standard optimizers such as stochastic gradient descent or Adam [14]. The influence of the mask loss is gradually increased by adapting λ after each epoch via the procedure `AdaptLambda` (see Section 3.2.3). Throughout the overall process, the procedure `ModelCheckpoint` assesses the mask losses of the intermediate models and stores them according to user-defined criteria.

3.2.3 Initialization, Parameters, and Post-Training.

We initialize the weight matrix $\boldsymbol{m}$ of the mask model m via the `InitAllMasks` procedure. Note that the decision boundary for the discrete mask is 0.5 due to the rounding operation applied in Equation (3) for discretization. Therefore, we initialize the weight matrix $\boldsymbol{m}$ with values larger than 0 since $\sigma(0) = 0.5$. This ensures that the mask model m initially retains all blocks, thereby allowing the prediction model to operate on the complete data at the beginning. This, in turn, ensures that the prediction model provides useful gradient information about the less important parts of the input. If parts of the input are removed too fast, the prediction model may degrade too quickly. Thus, we initialize the entries of the weight matrix $\boldsymbol{m}$ with random values drawn from a normal distribution $\mathcal{N}(\mu, \sigma)$, where μ is a constant larger than 1 and σ a small positive value ($\mu = 2$ and $\sigma = 0.01$ in all our experiments).

The procedure `InitLambda` initializes λ, which determines the trade-off between the task loss $\mathcal{L}_f$ and the mask loss $\mathcal{Q}$. Initially, λ is set to a small value (e. g., $\lambda = 0.1$) to ensure that the input data is not directly removed at the beginning of the training process. The influence of λ is then gradually increased until n_{epoch} epochs have been processed. Since the range of possible values for the model loss $\mathcal{L}_f$ is generally unknown a priori, we resort to a scheduler that increases λ through `AdaptLambda` in Line 8 of Algorithm 1 in case the overall error $\mathcal{L}_{f,\boldsymbol{m}^D} = \mathcal{L}_f + \lambda\mathcal{Q}$ has not decreased for a specific amount p of epochs. Thus, the λ-scheduler behaves similarly to standard learning schedulers. However, instead of decreasing the learning rate, the value for λ is increased by a user-defined factor λ_{fac} (e. g., $\lambda_{fac} = 1.1$).

During training, the intermediate models and masks are stored via the procedure `ModelCheckpoint` and the best-performing combination can be selected in the end. It is also possible to store the best-performing models for each interval of a user-defined partition of the amount of data to be removed (e. g., (5%, 10%, ..., 100%)) to account for varying bandwidth availability. In general, the performance of the models can also be slightly improved by continuing training the weights of the model f for several epochs without changing the mask weights anymore (post training).

3.3 Learning Dynamic Masks.

The approach outlined above is used to obtain task-specific input selection masks that are the same for all instances. In addition to such "static" masks, we introduce scenarios in which one additionally has access to a thumbnail/reduced version $\bar{\boldsymbol{x}}$ for each input image $\boldsymbol{x}$ (see Section 4). As described next, we use these reduced versions to obtain instance-based selection masks with a small constant overhead. Compared to the static masks, these dynamic masks yield further reductions in data transfer.

Figure 3 outlines the dynamic mask approach: The basic idea is to decide which parts of a given instance $\boldsymbol{x}$ need to be transferred based on the thumbnail $\bar{\boldsymbol{x}}$. To that end, the instance-based masks are generated via a separate thumbnail model $g : \overline{X} \rightarrow \mathbb{R}^{b_w \times b_h \times k}$ that receives a thumbnail $\bar{\boldsymbol{x}}$ and outputs instance-dependent mask weights (e. g., g could be a small convolutional neural network). Effectively, we replace the

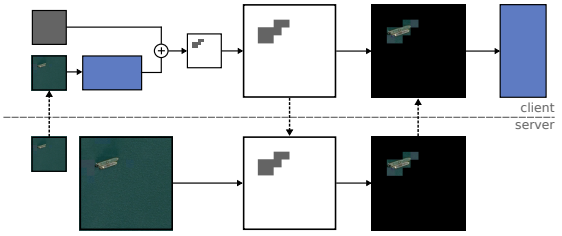


Figure 3: The thumbnail model g (left blue rectangle) identifies the relevant parts of input data based on a thumbnail $\bar{\boldsymbol{x}}$. Only those parts of the image $\boldsymbol{x}$ have to be transferred and are processed by the task model f (right blue rectangle).

weight matrix $\boldsymbol{m}$ of the mask model m by the output of the thumbnail model g in (3) and (4), i. e.:

$$\boldsymbol{m} = g(\bar{\boldsymbol{x}}) + c \tag{5}$$

Here, c is a positive constant (e. g., $c = 1$). Further, the weights of the thumbnail model g are initialized in such a way that $g(\bar{\boldsymbol{x}}) \approx \mathbf{0}$ at the beginning. This ensures that the masks select all data initially. These instance-based masks have the same size as the thumbnails and are, hence, expanded (e. g., via nearest neighbors interpolation) to the dimensions of the original input $\boldsymbol{x}$.

As for the static case, both the original model f and the thumbnail model g are trained simultaneously (on the client). The discretization approach also remains the same. Note that the thumbnail model g has to conduct a conceptually simpler task than the task model f: It only has to identify those parts of the input data $\boldsymbol{x}$ that are potentially relevant for f; it does not have to address the final learning task.

During the inference phase, the thumbnail, the instance-dependent mask, and the selected data must be transferred, which creates a small constant overhead for each instance. However, depending on the complexity of the instance, a dynamically selected input image may require significantly fewer selections than a static mask, which must be well suited for all possible input images. This higher reduction can often outweigh the price for the small constant overhead. In addition, each input image is selected by masks whose block size is smaller than the original input, which allows efficient block-wise encoding of the data to be transmitted. As shown in Section 4.2, such dynamic selections significantly reduce data transfer costs compared to a static selection.

4 Experiments.

We considered several classification datasets and network architectures, see Table 1. In addition to the well-known `cifar10`, `mnist`, fashion-`mnist` (`f-mnist`), and `svhn` datasets [15,17,26,32], we resorted to three more datasets from remote sensing and astronomy, respectively: For each instance of `remote`, one is given an image with 36 channels originating from six multi-spectral satellite image bands available for six different dates [27]. The learning goal of `remote` is to predict the type of change occurring in the central pixel of each image. The dataset `galaxy10` is dedicated to detecting different types of galaxies based on RGB images from the Sloan Digital Sky Survey [2] with labels from Galaxy Zoo [21]. Both `remote` and `galaxy10` are typical datasets for their respective domain, with the target objects being located in the centers of the images. Finally, we considered the `ship` dataset of the Airbus Ship Detection Challenge. We simplified the task from segmentation to classification (i. e., the task was to detect if a ship is visible in the image or not), halved the sizes of the original images (to obtain images of size 384×384), and used undersampling to balance the classes. In contrast to the other datasets, the relevant information is not necessarily in the middle of an image, which renders static selection approaches less useful.

Table 1: Datasets and Models

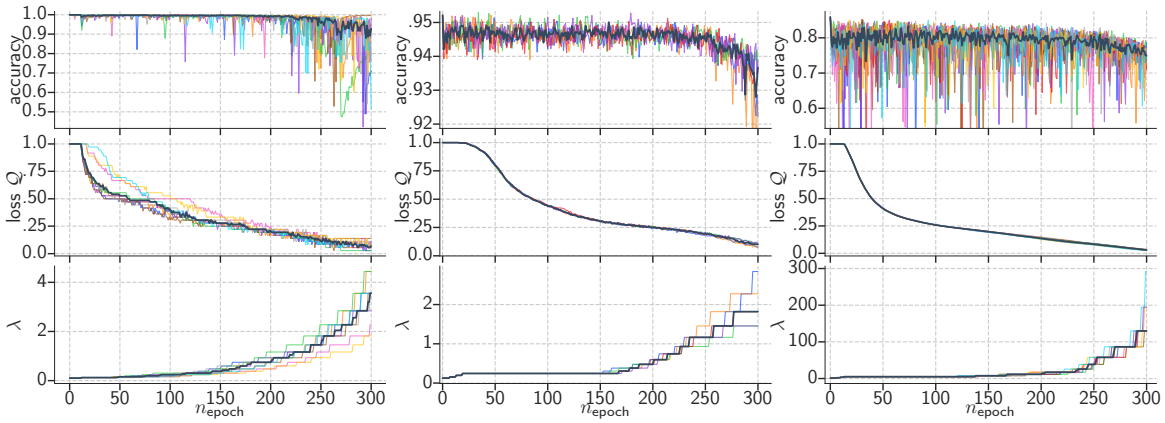
Dataset	#train	#hold-out	#class	w	h	c	model	lr
`remote`	24694	24694	12	35	35	36	AllConvNet	1e−3
`galaxy10`	19606	2179	10	69	69	3	ResNet20	1e−4
`cifar10`	50000	10000	10	32	32	3	ResNet20	1e−5
`mnist`	60000	10000	10	28	28	1	LeNet5	1e−3
`f-mnist`	60000	10000	10	28	28	1	LeNet5	1e−3
`svhn`	73257	26032	10	32	32	3	ResNet20	1e−5
`ship`	175548	17008	2	768	768	3	Squeezenet	1e−5

For all experiments, we considered a fixed amount of epochs and monitored the classification accuracy and mask loss $\mathcal{Q}$ on the hold-out set, see Table 1. The mask loss is a direct measure of how much raw information needs to be transferred. Depending on the application, further reductions might be achieved by using compression algorithms after the input selection. Each experiment was conducted $n_{runs} = 10$ times. We used pre-trained networks and optimizer parameters.

Our implementation is in PyTorch (version 1.5). Except for the trade-off parameter λ and the learning rate for the mask model m, all parameters were fixed (e.g. batch size of 128). The Adam [14] optimizer with AMSGrad [28] and specific learning rates per model and dataset were employed, see Table 1. More implementation details are available in the appendix.

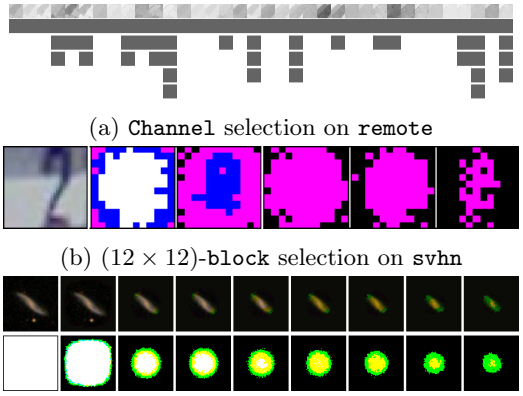
4.1 Static Selection Masks.

We start by demonstrating the basic functionality of the different types of selection masks, an evaluation of the parameter λ, and a comparison



(a) `channel` on `remote` (b) (12×12)-`block` on `svhn` (c) `pixel` on `galaxy10`

Figure 4: The black line is the average value of the runs and the individual runs are displayed in different colors. Note that the fluctuations are induced by the training process during which the input data are partially ignored. Performance at $n_{\text{epoch}} = 0$ represents performance of the pre-trained network.



(a) `Channel` selection on `remote`

(b) (12×12)-`block` selection on `svhn`

(c) `pixel` selection on `galaxy10`

Figure 5: In Figure (a), the selection process is sketched, whereby each row represents a different epoch (from top to bottom: example instance, 0, 50, 100, 150, 200) and each column represents a channel. For Figure (b), the example image is provided (left) along with the mask development during the training process. Figure (c) shows the progression of the image instance (i.e. the selected pixels of the image; at the end, central pixels from the green channel are selected).

with other static selection approaches.

4.1.1 Selection Schemes.

We first evaluated the behaviour of the `channel` selection scheme ($b_w = 1$ and $b_h = 1$) on `remote` to select the most relevant of the 36 input channels. Figure 4a shows the outcome of the iterative selection process induced by Algorithm 1. Only if less than 25% of the channels were selected, the accuracy started to drop. The evolution of this mask selection can be seen in Figure 5a.

The `block` selection experiment was conducted on `svhn`. The considered mask selected 12×12 blocks ($b_w = 12$ and $b_h = 12$) from each image per channel, see again Figure 2. Figure 5b shows an instance and the progress w.r.t. n_{epoch} for the RGB channels (purple: red and blue; yellow: green and red; white: all selected). Figure 4b shows the outcome of the iterative selection: Smooth optimization and a noticeable loss in accuracy around $\mathcal{Q} = 25\%$. To investigate the behaviour of `pixel`-wise selection, we considered the `galaxy10` dataset. The mask loss $\mathcal{Q}$ here reflects the summation over the selected pixels, where each pixel had a weight of $\frac{1}{w \times h \times k}$. The results of the iterative selection are provided in Figure 4c, whereas Figure 5c shows an instance and the development of the masks w.r.t. n_{epoch}.

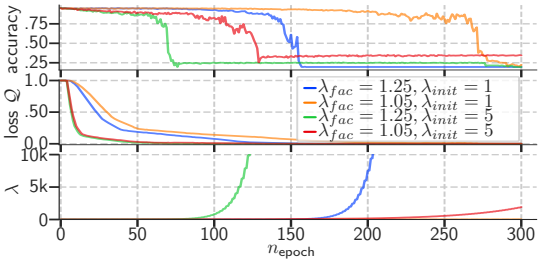
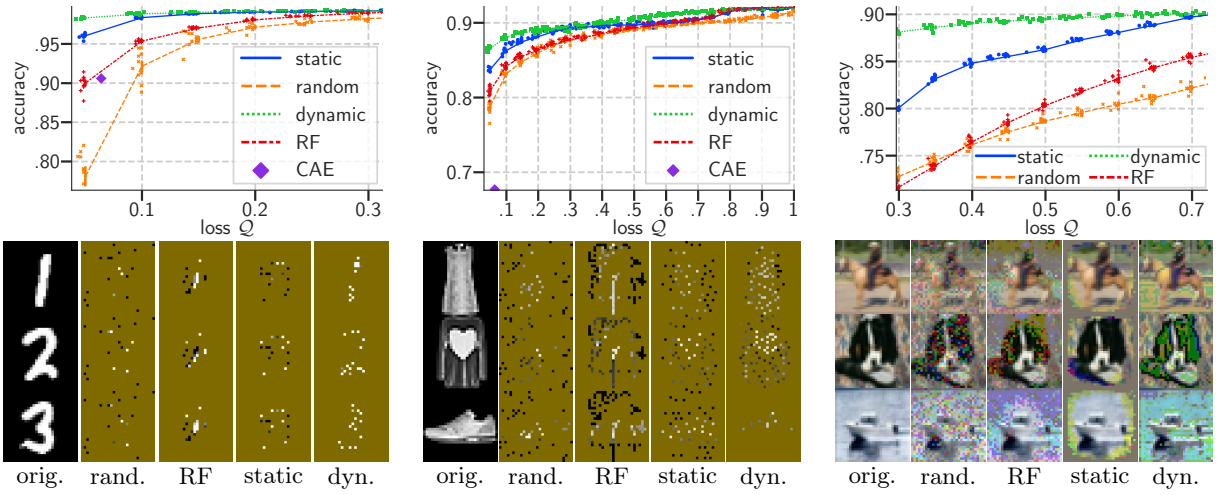


Figure 6: Influence of λ

4.1.2 Influence of λ.

The initial assignment λ_{init} for λ as well as the factor λ_{fac} generally have a significant impact on the outcome. Figure 6 shows the influence of four different configurations given the `svhn` dataset. A large λ_{init} (red and green line) led to the mask loss $\mathcal{Q}$ quickly decreasing, but it also caused a lower accuracy compared to the smaller λ_{init} values. A smaller initial value for λ generally led to



(a) `mnist` (b) `f-mnist` (c) `cifar10`

Figure 7: The top row shows the behaviour w.r.t. the mask loss $\mathcal{Q}$. The bottom row shows instances for different $\mathcal{Q}$: `mnist` (left) $\sim$ 2.42%, `f-mnist` (middle) $\sim$ 6%, and `cifar10` (right) $\sim$ 66%. Dark yellow color filling corresponds to removed pixels for `mnist` and `f-mnist`. For `cifar10` grey indicates completely removed pixel while tints indicate that channels for a pixel are disabled (e.g., green tinted pixel indicates missing red and blue).

the selection process taking less input data away at the beginning. Accordingly, a large λ_{fac} led to a faster decrease w.r.t. $\mathcal{Q}$.

4.1.3 Comparison with Baselines.

We compared our static selection approach with two direct competitors: random (backward) selection and the selection w.r.t. the feature importance values of a random forest (RF) [4]. In particular, we compared the approaches in the context of pixel-wise selections on `cifar10`, `mnist`, and `f-mnist`. To simulate a fair training process, we first created a ranking for each pixel based on randomness and the feature (i.e., pixel) importance values induced by an extra trees classifier [7] with 100 trees, respectively. Given the same task model f and the same amount of epochs, we then iteratively removed pixels to obtain the same loss $\mathcal{Q}$ as our static approach.

Figure 7 shows the results (the runs appear to be "clustered" since we saved the best accuracy in steps of 0.05-$\mathcal{Q}$ intervals): In can be seen that our static mask selection quickly diverges in accuracy from the random selection with decreasing $\mathcal{Q}$. The RF-based selection performed better than the random selection approach, but was outperformed by our approach, which shows the benefits of optimizing both the mask weights and the model weights simultaneously. We also compared our results with the concrete autoencoder (CAE) [3] approach, which selects a fixed (arbitrary) amount of pixels without retaining spatial structure and LassoNet [18], which iteratively removes features but can only be applied to fully-connected networks. The CAE approach achieved a self-reported accuracy of about 90.6% on `mnist` with a mask loss $\mathcal{Q}$ of 0.064 (50 out of 784 pixels) and the LassoNet paper reports 87.3% at the same mask loss. In comparison, our static selection approach yielded an accuracy of 96.11% given a lower mask loss $\mathcal{Q} = 0.047$. Similarly, on `f-mnist`, the CAE achieved a self-reported accuracy of 67.7% given a mask loss $\mathcal{Q}$ of 0.064, while our static mask yielded an accuracy of 83.86% given a mask loss of $\mathcal{Q} = 0.045$. It is worth stressing that the CAE and LassoNet approach resort to either random forest or fully-connected networks applied to the selected pixels, i.e., they cannot make use of predictors with spatial awareness.

4.2 Dynamic Selection Masks.

Finally, we conducted experiments for instance-based mask selections. For the sake of comparison, we also considered dynamic mask selection on `cifar10`, `f-mnist`, and `mnist` to illustrate the additional benefits of the instance-based selection over the other competitors mentioned above, see again Figure 7. The thumbnail model g used was a small linear layer. The $\mathcal{Q}$ reported is the average over all instance-dependent losses per batch. The static selection approach starts dropping in performance at a mask loss $\mathcal{Q}$ of 0.125, while the dynamic mask approach could retain the accuracy. A similar effect can be observed on `f-mnist` and `cifar10`, although the differences are much more apparent

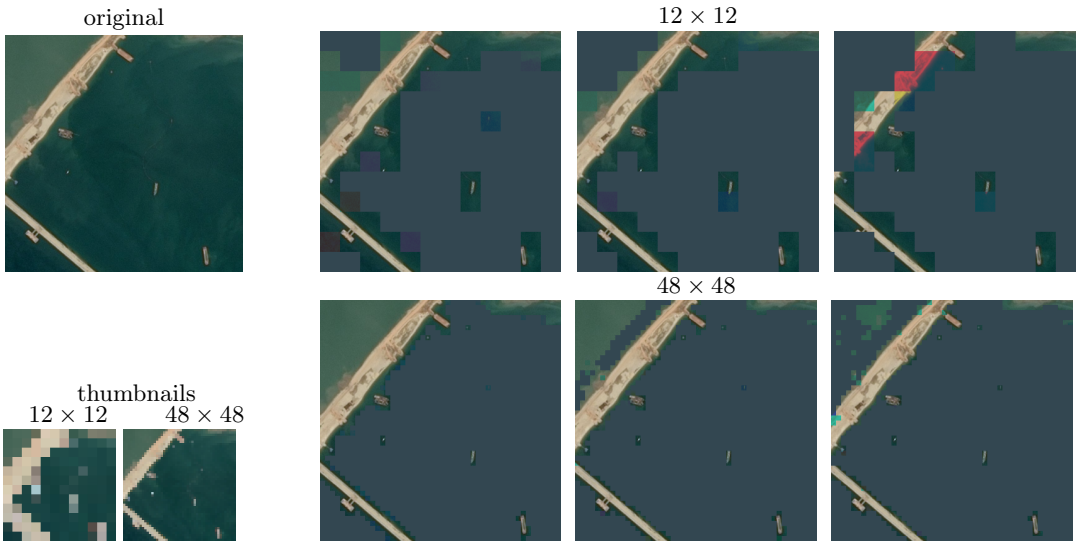
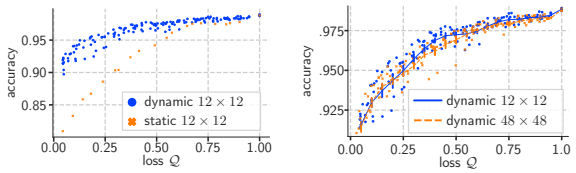


Figure 8: An examples of the instance-based selection for the dataset. Missing pixels were filled with average color based on the instances given in the training set (image size: 384 × 384 pixels).

for `cifar10`. Overall, since the dynamic approach yielded instance-based masks, less pixels were selected. It is worth stressing that for the dynamic approach, one has to transfer the selection mask per instance. For pixel-wise selection, this might only pay off if a few pixels are selected (since the coordinates per selected pixel need to be transferred as well). This problem is alleviated by selecting blocks, as shown next.

The potential of the dynamic selection approach is demonstrated on the `ship` dataset. Here, we considered thumbnails of size 12 × 12 and 48 × 48, respectively, which are processed by the thumbnail model g to identify the relevant parts of the input data per instance, see again Figure 3. We used a small convolutional network as thumbnail model g.[3] In Figure 9 (a), the difference between dynamic and static block selection is shown (12 × 12-block masks). Static selection quickly becomes inferior for this dynamic task. In contrast, both, the 12 × 12 and the 48 × 48 thumbnail selection performed well, see Figure 9 (b). The $\mathcal{Q}$ loss per block was $\frac{1}{12\times12\times k}$ and $\frac{1}{48\times48\times k}$, respectively. Overall, only a fraction of the original data needs to be transferred. For instance, using the model with $\mathcal{Q} \approx 0.025$ and 12 × 12 thumbnails, one would on average transfer less than $100 \cdot (0.025 + 2 \cdot \frac{144}{384\cdot384}) \approx 2.7\%$ of the original data per instance. Figure 8 shows two instances along with the (dynamic) masks for different mask loss $\mathcal{Q}$. The unimportant blocks (e. g., water or ground) were removed, but blocks containing ships or landmass remained.

[3] Composed of one selective kernel convolution [20] with 2 convolutional layers, 8 filters per layer, kernel size 3, a dilation rate of 1 and 2, respectively, followed by point-wise convolution.



(a) static vs. dynamic (b) 12 × 12 vs. 48 × 48

Figure 9: Comparing dynamic and static mask selection as well as different block sizes on the `ship` dataset.

5 Conclusions.

The transfer of data between servers and clients can become a major bottleneck during the inference phase of a neural network. We propose a framework that allows to automatically select the relevant parts of the input data needed by a model. Our approach resorts to various types of selection masks (`channel`, `pixel`, and `block`) that are optimized together with any given task model during the training phase. In addition to static selection, we also introduce instance-based selection to deal with tasks with shifting feature importance such as segmentation. Our experiments show that it is often possible to achieve a good accuracy with significantly less input data that needs to be transferred.

In addition, our mask selections are inferred from the task model. Hence, the selections eventually made also provides insights into what is considered relevant by the model and can, therefore, help to identify and understand learned patterns and potential biases.

Acknowledgement.

We acknowledge support from the Independent Research Fund Denmark through the grant *Monitoring Changes in Big Satellite Data via Massively-Parallel Artificial Intelligence* (9131-00110B). We also acknowledge support by the Villum Foundation through the project *Deep Learning and Remote Sensing for Unlocking Global Ecosystem Resource Dynamics (DeReEco)*.

References

[1] D. Aguado, H. Jönsson, H. Zou, and et al. The fifteenth data release of the sloan digital sky surveys: First release of manga-derived quantities, data visualization tools, and stellar library. *Astrophysical Journal, Sup. Series*, 240(2), 2019.

[2] H. Aihara, C. A. Prieto, D. An, S. F. Anderson, É. Aubourg, E. Balbinot, T. C. Beers, A. A. Berlind, S. J. Bickerton, D. Bizyaev, et al. The eighth data release of the sloan digital sky survey: first data from sdss-iii. *The Astrophysical Journal Sup. Series*, 193(2):29, 2011.

[3] M. F. Balın, A. Abid, and J. Zou. Concrete autoencoders: Differentiable feature selection and reconstruction. In *ICML*, pages 444–453. PMLR, 2019.

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] A. Coates, B. Huval, T. Wang, D. J. Wu, B. C. Catanzaro, and A. Y. Ng. Deep learning with COTS HPC systems. In *ICML*, volume 28, pages 1337–1345. JMLR.org, 2013.

[6] S. Gao, F. Huang, J. Pei, and H. Huang. Discrete model compression with resource constraint for deep neural networks. In *CVPR*, June 2020.

[7] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[8] A. Gordon, E. Eban, O. N. andcit Bo Chen, H. Wu, T. Yang, and E. Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *CVPR*, pages 1586–1595. IEEE, 2018.

[9] K. Grange, B. Alachkar, S. Amy, D. Barbosa, P. Bommimmeni, and P. Boven. Square kilometre array: The radio telescope of the xxi century. *Astronomy Reports*, 61(4):288–296, 2017.

[10] K. Han, Y. Wang, C. Zhang, C. Li, and C. Xu. Autoencoder inspired unsupervised feature selection. In *ICASSP*, pages 2941–2945. IEEE, 2018.

[11] S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural networks. In *NeurIPS*, pages 1135–1143, Cambridge, MA, USA, 2015. MIT Press.

[12] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *NeurIPS*, volume 29. Curran Associates, 2016.

[13] Ž. Ivezić, S. M. Kahn, J. A. Tyson, B. Abel, E. Acosta, R. Allsman, D. Alonso, Y. AlSayyad, S. F. Anderson, J. Andrew, and et al. LSST: From Science Drivers to Reference Design and Anticipated Data Products. *The Astrophysical Journal*, 873:111, 2019.

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[15] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[16] A. Kumar, S. Goyal, and M. Varma. Resource-efficient machine learning in 2 KB RAM for the IoT. In *ICML*, volume 70, pages 1935–1944. PMLR, 2017.

[17] Y. LeCun, C. Cortes, and C. Burges. MNIST handwritten digit database. *AT&T Labs*, 2010.

[18] I. Lemhadri, F. Ruan, L. Abraham, and R. Tibshirani. Lassonet: A neural network with feature sparsity. *JMLR*, 22(127):1–29, 2021.

[19] J. Li and D. P. Roy. A global analysis of sentinel-2a/2b and landsat-8 data revisit intervals and implications for ter. monitoring. *Remote Sensing*, 9(902), 2017.

[20] X. Li, W. Wang, X. Hu, and J. Yang. Selective kernel networks. In *CVPR*, pages 510–519. IEEE, 2019.

[21] C. J. Lintott, K. Schawinski, A. Slosar, K. Land, S. Bamford, D. Thomas, M. J. Raddick, R. C. Nichol, A. Szalay, D. Andreescu, et al. Galaxy zoo: morphologies derived from visual inspection of galaxies from the sloan digital sky survey. *Monthly Notices of the Royal Astronomical Society*, 389(3):1179–1189, 2008.

[22] Y. Lu, I. Cohen, X. S. Zhou, and Q. Tian. Feature selection using principal feature analysis. In *ACM Multimedia*, pages 301–304, 2007.

[23] Y. Lu, Y. Fan, J. Lv, and W. S. Noble. Deeppink: reproducible feature selection in deep neural networks. In *NeurIPS*, pages 8676–8686, 2018.

[24] C. J. Maddison, D. Tarlow, and T. Minka. A* sampling. In *NeurIPS*, pages 3086–3094, 2014.

[25] F. Nan, J. Wang, and V. Saligrama. Pruning random forests for prediction on a budget. In *NeurIPS*, pages 2334–2342. Curran Associates, 2016.

[26] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NeurIPS Workshop*, 2011.

[27] A. V. Prishchepov, V. C. Radeloff, M. Dubinin, and C. Alcantara. The effect of landsat ETM/ETM+ image acquisition dates on the detection of agricultural land abandonment in eastern europe. *Remote Sensing of Environment*, 126:195 – 209, 2012.

[28] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of adam and beyond. In *ICLR*, 2018.

[29] M. Reichstein, G. Camps-Valls, B. Stevens, M. Jung, J. Denzler, N. Carvalhais, and Prabhat. Deep learning and process understanding for data-driven earth system science. *Nature*, 566(7743):195–204, 2019.

[30] L. Theis, W. Shi, A. Cunningham, and F. Huszár. Lossy image compression with compressive autoencoders. In *ICLR*. OpenReview.net, 2017.

[31] M. A. Wulder, J. G. Masek, W. B. Cohen, T. R. Loveland, and C. E. Woodcock. Opening the archive: How free data has enabled the science and monitoring promise of landsat. *Remote Sensing of Environment*, 122(Sup. C):2 – 10, 2012. Landsat Legacy Special Issue.

[32] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, cs.LG/1708.07747, 2017.

[33] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi. Scaling for edge inference of deep neural networks. *Nature Electronics*, 1(4):216–222, 2018.

[34] Z. E. Xu, M. J. Kusner, K. Q. Weinberger, and M. Chen. Cost-sensitive tree of classifiers. In *ICML*, volume 28, pages 133–141. JMLR.org, 2013.

[35] Y. Zhang, S. Gao, and H. Huang. Exploration and estimation for model compression. In *ICCV*, pages 487–496, 2021.

A Reproducibility

We provide all the source code used for our experiments via an open GitHub repository.[4] For training the models and assessing their quality, different machines, and GPU devices (NVIDIA K20, K40, GTX1080, and V100) were used.

The chosen parameters for our static and dynamic experiments are summarized in Table 3a and 3b, respectively. The scripts with our parameter settings to run the corresponding experiments can be found in the code repository (directory `exp_scripts`). The data transformations applied during training are listed in Table 2.

Table 2: Dataset augmentations and transforms

dataset	vert. flip	horiz. flip	crop	normalize
`mnist`				✓
`f-mnist`				✓
`cifar10`		✓	✓	✓
`svhn`		✓	✓	✓
`galaxy10`	✓	✓	✓	✓
`remote`	✓	✓	✓	✓
`ship`	✓	✓	✓	✓

To run the experiments, the necessary requirements need to be installed (e. g., via `pip install -r requirements.txt`). The individual experiments can be started via the command line. For instance, the `block` experiment on `svhn` (using the normal "any" selection) can be started via:

```
python create_mask.py
--dataset svhn  --mask-type
static
--any-granularity subCXLIVdrant
--lambda-patience 5
--lambda-init 0.125
--lambda-factor 1.25 --n-epochs
300
--use-warmup-net 1 --lr 0.001
--n-repeats 10
```

By executing this command, a corresponding log file and checkpoints in the directory `runs` will be created. Note that the flag `--use-warmup-net 1` enables the use of pre-trained models and

[4] `https://github.com/StefOe/selection-masks`

optimizers. The mask type (i. e., random, static, or dynamic) can be changed via the flag `--mask-type`, which is, together with the flag `--dataset`, the only required parameter. For instance, the dynamic mask experiment for `mnist` can be reproduced via the following command:

```
python create_mask.py
--dataset mnist --mask-type
dynamic
--dynamic-mask linear
--any-granularity subpixel
--lambda-patience 5
--lambda-init 0.0005
--lambda-factor 1.5 --n-epochs
400
--use-warmup 1 --lr 0.0005
--n-repeats 10
```

The remaining experiments can be started in a similar fashion. An overview over the flags and options can be obtained via `python create_mask.py --help`.

(a) Parameters used for static experiments

granularity	dataset	λ_{init}	λ_{fac}	p	lr mask	epochs	any-init
`channel`	`remote`	0.1	1.25	10	0.001	300	3
`pixel`	`cifar10`	1	1.05	5	0.001	300	3
	`mnist`	0.0005	1.5	5	0.005	400	3
	`f-mnist`	0.0005	1.5	5	0.005	400	3
	`galaxy10`	1	1.5	2	0.001	300	3
`block`							
(12×12)	`svhn`	0.125	1.25	5	0.001	300	3

(b) Parameters used for dynamic experiments

granularity	mask model g	dataset	λ_{init}	λ_{fac}	p	lr mask	epochs
`pixel`	linear	`mnist`	0.0005	1.5	5	0.0005	400
		`f-mnist`	0.0005	1.5	2	0.0005	400
	convatt	`cifar10`	0.1	1.15	10	0.001	300
`block`							
(12×12)	convatt	`ship`	0.025	1.15	9	0.0005	400
(48×48)	convatt	`ship`	0.025	1.15	9	0.0005	400

B Broader Impact

We expect that such selection masks will play an important role for many data-intensive domains to alleviate the data transfer problem between centralized storage servers and clients: In many cases, the collected data are stored on centralized storage servers. The transfer of data between such servers and the users has already become a severe bottleneck. For instance, practitioners in remote sensing already resort to reduced versions of the data since a full data transfer would be too time-consuming (i. e., instead of the full multi-spectral image data, reduced versions are often considered, such as channels computed via the so-called Normalized Difference Moisture Index (NDMI)). A similar situation is given in astronomy, where the data transfer will become a key bottleneck in future with projects producing exabytes of data per year [1,9,13]. Today's storage servers already provide advanced APIs to select or crop the data prior to the transmission (e. g., the Planet API mentioned above can be used to select pieces of the data beforehand; similarly, services such as the Copernicus Open Access Hub allow to select subsets of the data and also provide previews of the data). The methods presented in this work offer the potential to alleviate the data transfer bottleneck in such domains, which, in the end, will lead to less resources that have to be spent for the overall infrastructure (e. g., more powerful servers, faster network connections, ...). Naturally, while our work addresses the two aforementioned application domains, the approaches developed are generally applicable to other domains as well (e. g., IoT data, medical image data, ...). We therefore believe that the results presented in this work will affect a broad range of domains and applications.