# Contents

# Chapter 3
# Application to neural networks

## Contents

## Introduction

TODO:

## 3.1 Layer representations

Let $\mathcal{L} = (g, h)$ a neural network layer, where $g : I \to O$ is its linear part, $h : O \to O$ is its activation function, $I$ and $O$ are its input and output spaces, which are tensor spaces.

### 3.1.1 Neural interpretation of tensor spaces

Recall from Definition **??** that a tensor space has been defined such that its canonical basis is a cartesian product of canonical bases of vector spaces. Let $I = \bigotimes_{k=1}^{p} \mathbb{V}_k$ and $O = \bigotimes_{l=1}^{q} \mathbb{U}_l$. Their canonical bases are denoted $\mathbb{v}_k = (\mathbb{v}_k^1, \dots, \mathbb{v}_k^{n_k})$ and $\mathbb{u}_l = (\mathbb{u}_l^1, \dots, \mathbb{u}_l^{n_l})$.

*Remark.* Note that a tensor space is isomorph to the signal space defined over its canonical basis.

More precisely, we have the following relation.

**Lemma 1. Relation between tensor and signal spaces**
Let $\mathbb{V}$, $\mathbb{U}$ vector spaces, and $\mathbb{v}$, $\mathbb{u}$ their canonical bases. Let $\mathbb{T}$ a tensor space. $\otimes$ and $\times$ denote tensor and cartesian products. Then,

(i) $\mathbb{V} \cong \mathcal{S}(\mathbb{v})$

(ii) $\mathbb{V} \otimes \mathbb{U} \cong \mathcal{S}(\mathbb{v} \times \mathbb{u})$

(iii) $\mathbb{V} \otimes \mathbb{T} \cong \mathcal{S}_{\mathbb{T}}(\mathbb{v})$

where $\mathcal{S}_{\mathbb{U}}$ are signals taking values in $\mathbb{U}$ (and $\mathcal{S}$ are real-valued signals).

*Proof.* (i) Given $x \in \mathbb{V}$, define $\widetilde{x} \in \mathcal{S}(\mathbb{v})$ such that $\forall i, \widetilde{x}[\mathbb{v}^i] = x[i]$. The mapping $x \mapsto \widetilde{x}$ is a linear isomorphism.

(ii) $\widetilde{x}[\mathbb{v}^i, \mathbb{u}^j] = x[i, j]$

(iii) $\widetilde{x}[\mathbb{v}^i] = x[i, :, \dots, :]$

□

Let $d \leq n_k$ and $e \leq n_l$. Define $V$ and $U$ as the cartesian products $V = \bigtimes_{k=1}^{d} \mathbb{v}_k$ and $U = \bigtimes_{l=1}^{e} \mathbb{u}_l$. Thanks to Lemma 1, we can identify the input and output spaces as $I = \mathcal{S}(V) \otimes \bigotimes_{k=d+1}^{p} \mathbb{V}_k$ and $O = \mathcal{S}(U) \otimes \bigotimes_{l=e+1}^{q} \mathbb{U}_l$. As $\mathcal{S}(\mathbb{v}) \otimes \mathbb{T} = \mathcal{S}_{\mathbb{T}}(\mathbb{v})$, an object of $V$ or $U$ can be interpreted as the representation of a *neuron* which can take multiple values.

In what follows, without loss of generality, we will make the simplification that a neuron can only take a single value (we don't consider input channels and feature maps yet). We'll thus consider that $I = \mathcal{S}(V)$ and $O = \mathcal{S}(U)$, where $V$ is the set of *input neurons*, and $U$ is the set of *output neurons.*

## 3.1.2 Propagational interpretation

Let $\mathcal{L} = (g, h)$, recall that $g : \mathcal{S}(V) \to \mathcal{S}(U)$ is characterized by a connectivity matrix $W$ such that, $g(x) = Wx$.

*Remark.* Using the mapping defined in the proof of Lemma 1, for notational conveniency, we'll abusively consider $x$ as a vector (enventually reshaped from a tensor), and $W$ as an object of a binary tensor product for its indexing (*i.e.* $W[u, v] := W[i, j]$ where $u = \mathbb{u}^i$ and $v = \mathbb{v}^j$).

**Definition 2. Propagation graph**
The *propagation graph* $P = \langle (V, U), E_P \rangle$ of a layer $\mathcal{L} = (W, h)$ is the bipartite graph that has the connectivity matrix $W$ for bipartite adjacency matrix.

An example is depicted on Figure 1.

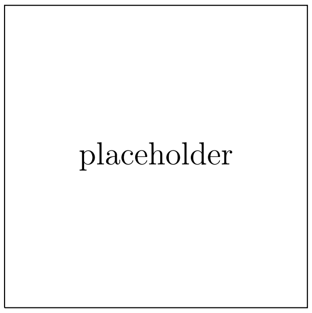


Figure 1: A propagation graph

The propagation graph defines an input topological space $\mathcal{T}_V$, and an output topological space $\mathcal{T}_U$.

**Definition 3. Topological space**
A *topological space* is a pair $\mathcal{T} = (X, \mathcal{O})$, where $X$ is a set of points, $\mathcal{O}$ is a set of sets that is closed under intersection (the *open sets*), and such that every point $x \in X$ is associated with a set $\mathcal{N}_x \in \mathcal{O}$, called its *neighborhood.*

Hence, the *neural topologies* $\mathcal{T}_V$ and $\mathcal{T}_U$ are defined as

1. $\mathcal{T}_V = (V, \mathcal{O}(U))$, with $\forall v \in V, \mathcal{N}_v = \{u \in U, v \overset{P}{\sim} u\}$
2. $\mathcal{T}_U = (U, \mathcal{O}(V))$, with $\forall u \in U, \mathcal{N}_u = \{v \in V, v \overset{P}{\sim} u\}$

In particular, given an output neuron $u \in U$, a neighborhood $\mathcal{N}_u$ is also called a *receptive field*, that we denote $\mathcal{R}_u$.

### 3.1.3 Graph representation of the input space

Let's consider that the input neurons $V$ have a (possibly edge-less) graph structure $G = \langle V, E \rangle$. We define an edge-constrained layer as follows.

**Definition 4. Edge-constrained layer**
A layer $\mathcal{L} : G = \langle V, E \rangle \to U$, is said to be *edge-constrained* (EC) if:

1. There is a one-to-one correspondence $\pi : V_\pi \to U$, where $V_\pi \subset V$.

2. Given an output neuron $u$, an input neuron $v$ is in its receptive field, if and only if, $v$ and the $\pi$-fiber of $u$ are connected in $G$,
$$i.e.\ \forall u \in U, v \in \mathcal{R}_u \Leftrightarrow v \overset{E}{\sim} \pi^{-1}(u)$$

Note that (EC) convolutions are (EC) layers. We have the following characterization.

**Proposition 5. (EC) Characterization with receptive fields**
Let a layer $\mathcal{L} : V \to U$, $V_\pi \subset V$, and a one-to-one correspondence $\pi : V_\pi \to U$. There exists a graph $G = \langle V, E \rangle$ for which $\mathcal{L}$ is (EC), if and only if, the receptive fields are *intertwined* ($i.e.\ \forall a, b \in V_\pi, a \in \mathcal{R}_{\pi(b)} \Leftrightarrow b \in \mathcal{R}_{\pi(a)}$).

*Proof.* $\Rightarrow$ : Thanks to $a \in \mathcal{R}_{\pi(b)} \Leftrightarrow a \overset{E}{\sim} b \Leftrightarrow b \in \mathcal{R}_{\pi(a)}$

$\Leftarrow$ :If the receptive fields are intertwined, then the relation defined as $a \sim b \Leftrightarrow a \in \mathcal{R}_{\pi(b)}$ is symmetric, and thus can define an edge set.

□

Therefore, any layer that has its receptive fields intertwined, admits an *underlying* graph structure. For example, a 2-d convolution operator can be rewritten as an (EC*) convolution on a lattice graph, and as an (EC) convolution on a grid graph.

Figure 2 depicts an underlying graph and its corresponding propagation graph.

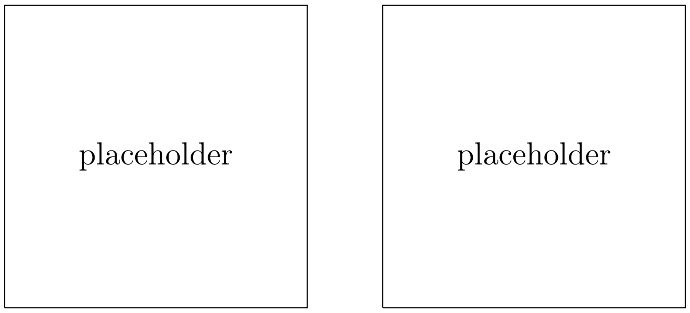


Figure 2: Underlying graph Vs Prop graph

### 3.1.4 General representation with weight sharing

Weight sharing refers to the fact that some parameters of the connectivity matrix $W$ are equal, and stay equal after each learning iteration. In other words they are tied together. From a propagational point of view, this amounts to label the edges of the propagation graph $P$ with weights, where weights can be used multiple times to label these edges. Supposed $W$ is of shape $m \times n$ and there are $\omega$ weights in the kernel used to label the edges. Given a input neuron $i$ and an output neuron $j$, the edge labelling can be expressed as:

$$W[j,i] = \theta[h] = \theta^T a \tag{1}$$

where $\theta$, the weight kernel, is a vector of size $\omega$ and $a$ is a one-hot vector (full of 0s except for one 1) of size $\omega$ with a 1 at the index $h$ corresponding to the weight that labels the edge $i \sim j$ ; or $a$ is the *zero* vector in case $i \nsim j$.

This equation (1) can be rewritten as a tensor contraction under Einstein summation convention, by noticing that $a$ depends on $i, j$ and by defining a tensor $S$ such that $a = S[:, i, j]$, as follows:

$$W_{ij} = \theta_k {S^k}_{ij} \tag{2}$$

Therefore, the linear part of $\mathcal{L}$ can be rewritten as:

$$g(x)_j = \theta_k S^k{}_j{}^i x_i \tag{3}$$

If we consider that the layer $\mathcal{L}$ is duplicated with input channels and feature maps, then $\theta$, $x$ and $g(x)$ become tensors, denoted $\Theta$, $X$ and $g(X)$. Usually, for stochastic gradient descent, $X$ and $g(X)$ are also expanded with a further rank corresponding to the batch size and we obtain:

$$g(X) = \Theta S X \text{ where } \begin{cases} W_{pq}{}^{ij} = \Theta_{pq}{}^k S_k{}^{ij} \\ g(X)_{jq}{}^b = W_{jq}{}^{ip} X_{ip}{}^b \end{cases} \tag{4}$$

| index | size | description |
|---|---|---|
| $i$ | $n$ | input neuron |
| $j$ | $m$ | output neuron |
| $p$ | $N$ | input channel |
| $q$ | $M$ | feature map |
| $k$ | $\omega$ | kernel weight |
| $b$ | $B$ | batch instance |

Table 1: Table of indices

*Remark.* Note that the expression $\Theta S X$ is written regardless of the ordering of the tensors ranks and is defined by index juggling.

Also, note that it is associative and commutative. This can be seen by the index symmetry of (5), which rewrites (4), and where the sum symbols $\Sigma$ and scalar values commute:

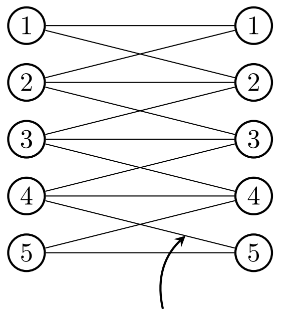
$$\Theta S X[j,q,b] = \sum_{k=1}^{\omega} \sum_{p=1}^{P} \sum_{i=1}^{n} \Theta[k,p,q]\, S[k,i,j]\, X[i,p,b] \tag{5}$$

We call $S$ the *weight sharing scheme* of the layer $\mathcal{L}$.

**Definition 6. Ternary representation**
The *ternary representation* of a layer $\mathcal{L} : X \mapsto Y$, with activation function $h$, is the equation $Y = h\left(\Theta S X\right)$, as defined in (4), where $\Theta$ is the *weight kernel*, and $S$ is called the *weight sharing scheme*.

*Remark.* In (1), we defined $a = S[:, i, j]$ as a one-hot vector when $i \sim j$, as its role is to select a weight in $\theta = \Theta[p, q, :]$. However, $a$ can also do this selection *linearly*, so in fact it is not necessarily a one-hot vector.

Figure 3 depicts an example of how the equation (4) labels the edges of $P$.



$$W[p, q, i = 4, j = 5] = \sum_{k=1}^{\omega} \Theta[p, q, k] S[k, 4, 5]$$

Figure 3: Example of a propagation graph $P$ for a given input channel $p$ and feature map $q$. The edge $4 \sim 5$ is labelled with a linear combination of kernel weights from $\Theta[p, q, :]$. In the usual case, $S[k, 4, 5]$ is a one-hot vector that selects a single kernel weight: $\exists h, W[p, q, 4, 5] = \Theta[p, q, h]$.

The ternary representation uncouples the roles of $\Theta$ and $S$ in $W$, and is the most general way of representing any kind of partially connected layer with weight sharing.

## 3.2 Study of the ternary representation

In this section, we study the ternary representation, which is the general representation with weight sharing we obtained above.

### 3.2.1 Genericity

The ternary representation can represent any kind of layer. For example,

- To obtain a fully connected layer, one can choose $\omega$ to be of size $nm$ and $S$ the matrix of vectors that contains all possible one-hot vectors.
- To obtain a convolutional layer, one can choose $\omega$ to be the size of the kernel. $S$ would contain one-hot vectors. A stride $> 1$ can be obtained by removing the corresponding dimensions. If the convolution is a classical convolution, or is supported by a Cayley subgraph (see Chapter **??**), then $S$ would be circulant along the input neurons rank in the canonical basis.
- Any partially connected layer with (or without) weight sharing can be obtained with appropriate construction of $S$.

### 3.2.2 Efficient implementation under sparse priors

**What is the fastest way to compute $\Theta SX$ ?**

As the equation (4) is associative and commutative, there are three ways to start to calculate it: with $\Theta S$, $SX$, or $\Theta X$, which we will call *middle-stage tensors*. The computation of a middle-stage tensor is the bottleneck to compute (4) as it imposes to compute significantly more entries than for the second tensor contraction. In Table 2, we compare their shapes. We refer the reader to Table 1 for the denomination of the indices.

| tensor | shape |
|---|---|
| $\Theta$ | $\omega \times P \times Q$ |
| $S$ | $\omega \times n \times m$ |
| $X$ | $n \times P \times B$ |
| $\Theta S$ | $n \times m \times P \times Q$ |
| $SX$ | $\omega \times m \times P \times B$ |
| $\Theta X$ | $\omega \times n \times Q \times B$ |
| $\Theta SX$ | $m \times Q \times B$ |

Table 2: Table of shapes

In usual settings, we want to have $\omega \ll n$ and $\omega \ll m$, which means that we have weight kernels of small sizes (for example in the case of images, convolutional kernel are of size significantly smaller than that of the images). Also, the number of input channels $P$ and of feature maps $Q$ are roughly in the same order, with $P < Q$ more often than the contrary. It turns out that in practice, the size of $\Theta S$ is significantly bigger than the size of $SX$ and of $\Theta X$, and the size of $SX$ is usually the smallest.

**How to exploit $S$ sparsity ?**

Also, in usual settings, $S$ is sparse as $S[:, i, j]$ are one-hot vectors. So computing $SX$ should be faster that computing $\Theta X$, provided we exploit the sparsity. Although $S$ is very sparse as it contains at most a fraction $\frac{1}{w}$-th of non-zero values, it is only sparse along the first rank, which makes implementation with sparse classes of common deep learning libraries not optimized. So we proceed differently. The idea is to use a non-sparse tensor $X_{\text{LRF}}$ that has a rank that indexes local receptive fields (LRF), and another rank that indexes elements of these LRF, in order to lower the computation to a dense matrix multiplication (or dense tensor contraction) which is already well optimized. This approach, proposed in Chellapilla et al., 2006, is also exploited in the cudnn primitives (Chetlur et al., 2014) to efficiently implement the classical convolution.

### The LRF representation

In our case, it turns out that $X_{\text{LRF}}$ can be exactly $SX$, as given fixed $b$, $p$, and $j$, $SX[:, j, p, b]$ corresponds to entries of the input signal $X[:, p, b]$ restrained to a LRF $\mathcal{R}_j$ of size $\omega$. Therefore,

$$\exists\, \text{LRF}_j = [i_1, \ldots, i_\omega] \ s.t.\ SX[:, j, p, b] = X[\text{LRF}_j, p, b] \tag{6}$$

The elements of $\text{LRF}_j$ can be found by doing a lookup in the one-hot vectors of $S$, provided each kernel weight occurs exactly once in each LRF. We have:

$$R_j[k] = i_k \ s.t.\ S[:, i_k, j][k] = 1 \tag{7}$$

This lookup needs not be computed each time and can be done beforehand. Finally, if we define $\text{LRF} = [\text{LRF}_1, \ldots, \text{LRF}_m]$, (6) gives:

$$SX = X[\text{LRF}, :, :] \tag{8}$$

The equation (8) is computed with only $\omega \times m$ assignations and can be simply implemented with automatic differentiation in commonly used deep learning libraries.

### Benchmarks

To support our theoretical analysis, we benchmark three methods for computing the tensor contraction $SX$:

- naively using dense multiplication,
- using sparse classes of deep learning libraries,
- using the LRF based method we described above.

We run the benchmarks under the assumptions that $S[:, i, j]$ are one-hot vectors, and that a weight occur exactly once in each LRF (as it is the case for convolutions supported by a Cayley subgraph). For each method, we make 100 runs of computations of $SX$, with $S$ and $X$ being randomly generated according to the assumptions. In Table 3, we report the mean time and standard deviation. The values of the hyperparameters were each time $n = m = N = M = B = 100$, and $\omega = 10$. The computations were done on graphical processing units (GPU).

| Method | Time |
|---|---|
| Naive | *todo*$\mu s \pm$ *todo* |
| Sparse | *todo*$\mu s \pm$ *todo* |
| LRF | ***todo***$\mu s \pm$ ***todo*** |

Table 3: Benchmark results

As expected, the LRF method is faster.

### 3.2.3 Influence of symmetries

In the case of images, or other signals over a grid, the grid structure of the domain defines the weight sharing scheme $S$ of the convolution operation. For example, for a layer $\mathcal{L} : X \mapsto Y = h(\Theta SX)$, and given fixed $b, p, q$, the classical convolution can be rewritten as

$$y[j] = h\left(\sum_{k=1}^{\omega} \theta[k] \sum_{i=1}^{n} S[k, i, j]\, x[i]\right) \tag{9}$$

$$= h\left(\sum_{k=1}^{\omega} \theta[k]\, x_{\text{LRF}}[k, j]\right) \tag{10}$$

Where $x_{\text{LRF}}[k, j]$ can be obtained by matching (10) with the expression given by the definition (see Definition **??**). So, $S[k, :, j]$ is a one-hot vector that specifies which input neuron in the LRF of $y$ is associated with the $k$-th kernel weight, and the index of the 1 is determined by $x_{\text{LRF}}[k, j]$.

### Visual construction

Visually, constructing $S$ amounts to move a rectangular grid over the pixel domain, as depicted on Figure 4 where each point represents the center of a pixel, and the moving rectangle represents the LRF of its center $j$. Each of its squares represents a kernel weight $\theta[k]$ which are associated with the pixel $x_{\text{LRF}}[k, j]$ that falls in it.

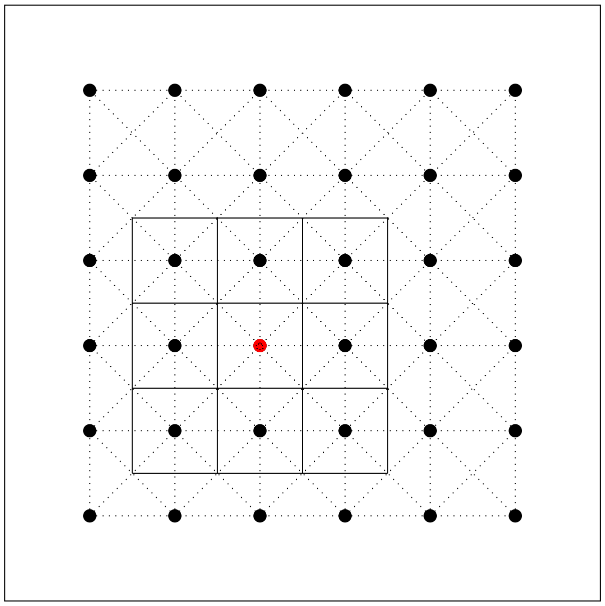


Figure 4: Weight assignement in convolution on pixel domains

The case of images is very regular, in the sense that every pixels are regularly spaced out, so that obtained $S$ is circulant along its last two ranks. This is a consequence of the translational symmetries of the input domain, which underly the definition of the convolution, as seen in Chapter **??**.

- What happens if we loose these symmetries?

To answer this question, we make the following experiment (Vialatte et al., 2016):

1. We distort the domain by moving the pixels randomly. The radial displacement is uniformly random with the angle, and its radius follows a gaussian distribution $\mathcal{N}(0, \sigma)$.

2. Then we compare performances of shallow CNNs, for which $S$ is constructed similarly than with the above visual construction, for different values of $\sigma$.

The visual construction of $S$ on distorded domain is depicted by Figure 5.

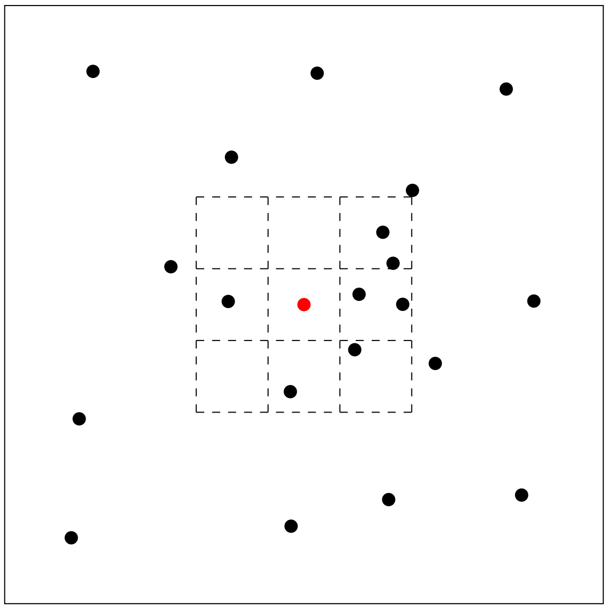


Figure 5: Weight assignement in generalized convolution on distorded domains

We run a classification task with standard hyperparameters on a toy dataset (we used MNIST, LeCun et al., 1998). The results are reported in Figure 6

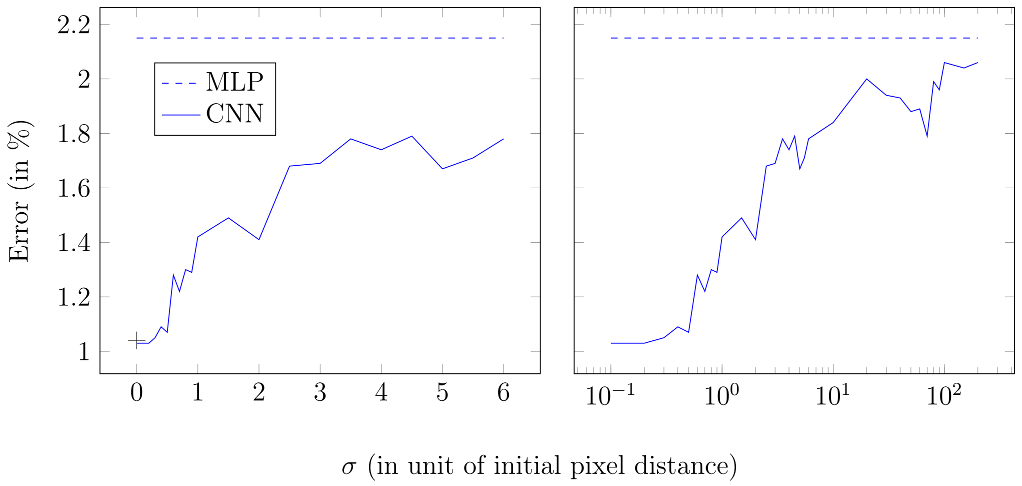


Figure 6: Error in function of the standard deviation $\sigma$, for generalized CNNs and an MLP, each with 500 weights.

The bigger is $\sigma$, the less accurate are the symmetries of the input domain, up to a point where the ternary representation becomes almost equivalent to a dense layer. The results illustrate nicely this evolution, and stress out the importance of trying to leverage symmetries when defining new convolutions.

### 3.2.4 Learning the weight sharing scheme

## 3.3 Extending CNNs using EC symmetries on graph domains

# Bibliography

Chellapilla, Kumar, Sidd Puri, and Patrice Simard (2006). “High performance convolutional neural networks for document processing”. In: *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft (cit. on p. 13).

Chetlur, Sharan, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer (2014). “cudnn: Efficient primitives for deep learning”. In: *arXiv preprint arXiv:1410.0759* (cit. on p. 13).

LeCun, Yann, Corinna Cortes, and Christopher JC Burges (1998). *The MNIST database of handwritten digits* (cit. on p. 17).

Vialatte, Jean-Charles, Vincent Gripon, and Grégoire Mercier (2016). “Generalizing the convolution operator to extend cnns to irregular domains”. In: *arXiv preprint arXiv:1606.01166* (cit. on p. 16).