To appear in Proc. of IEEE Intelligent Vehicles Symposium (IV), 2019, Paris, France.

Circular Convolutional Neural Networks for Panoramic Images and Laser Data

Stefan Schubert[1], Peer Neubert[1], Johannes Pöschmann[1] and Peter Protzel[1]
{firstname.lastname@etit.tu-chemnitz.de}

Abstract— **Circular Convolutional Neural Networks (CCNN) are an easy to use alternative to CNNs for input data with wrap-around structure like 360° images and multi-layer laserscans. Although circular convolutions have been used in neural networks before, a detailed description and analysis is still missing. This paper closes this gap by defining circular convolutional and circular transposed convolutional layers as the replacement of their linear counterparts, and by identifying pros and cons of applying CCNNs. We experimentally evaluate their properties using a circular MNIST classification and a Velodyne laserscanner segmentation dataset. For the latter, we replace the convolutional layers in two state-of-the-art networks with the proposed circular convolutional layers. Compared to the standard CNNs, the resulting CCNNs show improved recognition rates in image border areas. This is essential to prevent blind spots in the environmental perception. Further, we present and evaluate how weight transfer can be used to obtain a CCNN from an available, readily trained CNN. Compared to alternative approaches (e.g. input padding), our experiments show benefits of CCNNs and transfered CCNNs regarding simplicity of usage (once the layer implementations are available), performance and runtime for training and inference. Implementations for Keras with Tensorflow are provided online[2].**

I. Introduction

Convolutional Neural Networks (CNN) are a widely used and powerful tool for processing and interpreting image-like data in various domains like automotive, robotics or medicine. Compared to layer-wise fully connected networks, CNNs benefit from weight-sharing: Instead of learning a large set of weights from all input pixels to an element of the next layer, they learn few weights of a small set of convolutional kernels that are applied all over the image. The goal is to achieve shift equivariance: A trained pattern (e.g. to detect a car in a camera image) should provide strong response at the particular location of the car in the image, independent of whether this location is, e.g., in the left or right part of the image. However, this goal is missed at locations close to the image borders, where the receptive field of the convolution exceeds the input. Typically, these out-of-image regions are filled with zero-padding. During the repeated convolutions in a CNN, this zero-padding occurs at each layer and the effect of distorted filter responses grows from the image borders towards the interior. While this seems inevitable for imagery from pinhole-model cameras, this is not the case for panoramic data. In panoramic data, there is at least one dimension with wrap-around structure and without an inherent image border. However, feeding panoramic data as 2D images to a standard CNN artificially introduces such borders. Fig. 1 shows the example of object recognition using a car mounted Velodyne - its 360° degree depth and reflectance images are practically important examples for panoramic data. Ignoring the wrap-around connections by using standard CNNs creates blind spots near the image borders where we are unable to interpret the environment.

This paper discusses an extension of CNNs for wrap-around data: Circular Convolutional Neural Networks (CCNNs), which replace convolutional layers with circular convolutional layers. Each CCNN layer has a receptive field that is a connected patch in the image interior, but wraps around the appropriate image borders. Fig. 1 illustrates this concept as convolution on a ring.

For panoramic data, modeling the circular path in the convolutional NN is quite obvious and has been casually used in several existing works (e.g. recently [1], [2]). However, there are also many approaches that do not model the circular path in the network, but use surrogate techniques like input padding [3] (which increases computational time and memory consumption) or training on multiple circular shifted versions of the input data [4] (that increases the amount of training data and time without additional information gain, and might waste representational capacity for particular filters for border areas). Sometimes, the wrap-around connections are even not used at all [5].

Why are circular convolutions not used more frequently in CNNs for panoramic data? We see two major reasons in (1) missing literature with evidence and quantification of their benefits and (2) a lack of knowledge about their systematic design, particularly technical details on the implementation for nontrivial CNNs (e.g. those with transposed convolutional (deconvolutional) layers). This paper addresses these issues with the following contributions:

- Sec. III explains how circular convolution can be implemented in Circular Convolutional Layers and derives the novel Circular Transposed Convolutional Layer that extends the application of circular convolution to a wider range of neural network architectures, in particular many generative convolutional networks.
- Using these layers, Circular Convolutional Neural Networks (CCNN) can be as easily built and trained as CNNs (Sec. III). This is also demonstrated in Sec. V-B where we

[1]Chemnitz University of Technology, 09126 Chemnitz, Germany. This research has been funded by the European Social Fund (ESF) and Federal State of Saxony within the project VIVARE #100296259. We thank Intenta GmbH, Chemnitz, Germany for providing the labeled 360° LiDAR data.

[2]Open source code for Circular Convolutional and Transposed Convolutional Layers and MNIST example in Keras with Tensorflow: `http://www.tu-chemnitz.de/etit/proaut/ccnn`

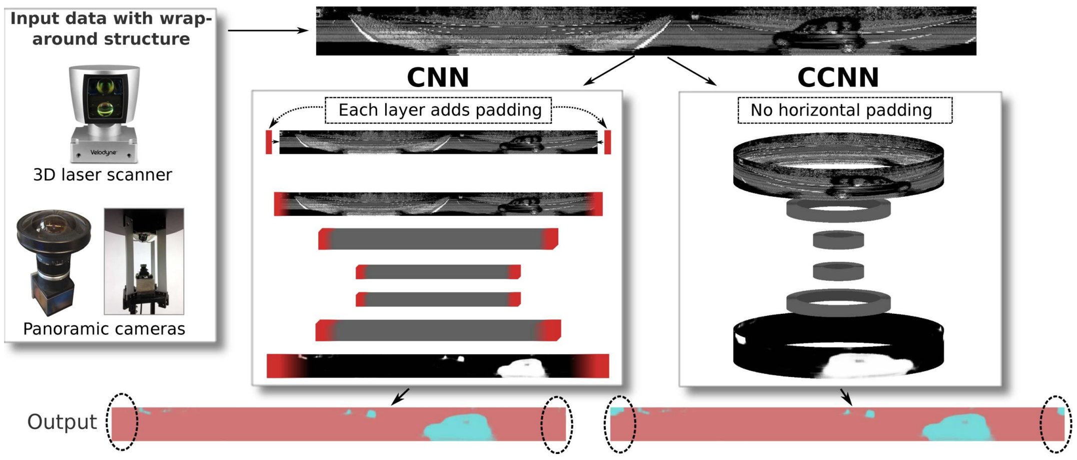

Fig. 1: The example image on the top right shows a reflectance map of a car mounted Velodyne laser scanner. The street in front of the car is located in the middle of the image, the street behind the car can be seen on the left and right borders. In this unwrapped panorama, it is hard to recognize the truck that is directly behind the car and split by the image border, half of it is at the left and the right image border. A standard CNN would apply zero-padding and is likely to miss the two halves of the truck. Even if the image is rotated to place the image border at a less important direction, there still remains a blind spot somewhere in the visual field. In a Circular Convolutional Neural Network, there is no such horizontal border and no such blind spot.

create CCNNs for two existing state-of-the-art semantic segmentation networks (DeconvNet [6] and Tiramisu [7]).

- Beside training from scratch, the proposed CCNN structure allows weight transfer from an available, readily trained CNN to create an according CCNN (sec. III-C). We evaluate this novel approach in Sec. V-D.
- Input padding is the most common surrogate approach to circular convolutions. Sec. IV discusses problems of this approach and provides a theoretical bound on the amount of required input padding to resemble the behavior of CCNNs. The effects of input padding of various amounts are experimentally evaluated and compared to CCNNs in Sec. V-E. We also experimentally evaluate the benefit of CCNNs over the above mentioned surrogate technique of training on shifted input data [4] in Sec. V-A.
- Sec. V-B qualitatively evaluates the performance gain of CCNNs compared to conventional CNNs on the important automotive recognition task from Fig. 1 using panoramic Velodyne laser data. We show that there is a blind spot for conventional CNNs near the image border and that this can be addressed by CCNNs. We further quantify the range of the positive effects towards the image interior. These experiments are done using the above mentioned CCNN versions of the state-of-the-art DeconvNet and Tiramisu networks.
- The implementation of CCNN layers for Keras with Tensorflow backend is provided online[2]. Changing a model from CNN to CCNN is as easy as changing the layer name from Conv2D to CConv2D.

II. Related Work

Although using a circular convolution in a CNN is not new (e.g. [1], [2]), comprehensive presentation, analysis and discussion of their advantages and drawbacks is still missing. In particular, to the best of our knowledge, prior works never introduced or applied circular transposed convolutional (deconvolutional) layers or used weight transfer between linear and circular convolutional layers.

In automotive applications and robotics, circular convolution is of particular practical interest for two types of data sources: 360° imagery (panoramic or spherical images) and depth data from rotating multi-layer LiDARs (e.g. Velodyne laser scanners):

Panoramic images A straightforward way of dealing with panoramic input images is to ignore the wrap-around connections, e.g., by feeding the whole image or cropped patches to a standard CNN [5]. A more advanced approach is to rotate (circular shift) the image to bring less important image content to the border or to align the views [8]. Lo et al. [4] augments the training data with a set of circularly shifted images to cope with circular data which caused a $36\times$ higher training effort. For object recognition, Shi et al. [3] create panoramic views from a cylindric projection of a 3D model around its principal axis. This cylindric projection is unwrapped and fed to a CNN. Border effects at the unwrapped cylinder image seam are addressed by padding the image on one side with an appropriate patch. Furthermore, they introduce a row-wise max-pooling layer to create rotation invariance. Such an increased input size comes at the cost of additional efforts for training and inference, e.g., in [3] the image size is increased by 2/3 of the image width. A multiplication in the frequency domain corresponds to a circular convolution in the spatial domain, thus implementing convolution using a Fourier transform could account for wrap-around structures. This technique is quite common for Correlation Filters [9] and occasionally also discussed in the context of neural networks for image processing [10].

Spherical images A typical source of panoramic images are projections of spherical images, e.g., from mirror-based omnidirectional cameras or vertically directed fisheye cameras. To avoid the projection, a straightforward approach is to select patches directly from the spherical image and feed them to a standard CNN [11]. Su et al. [12] reshape and learn convolutional kernels at different polar angles to apply them directly on the spherical image. In another direction, Ran et al. [13] train CNNs to work on holistic uncalibrated spherical images. Cheng et al. [1] project 360°-images onto a cube to generate six flat images; to address the wrap-around structure of the images, they use the cube-neighbors for padding to perform a circular convolution. However, in a post-processing step to get a saliency map they upsample an intermediate representation without any further circular consideration. Cohen et al. [14] propose a class of CNNs for spherical images that apply convolutions directly on the sphere and rotation group. They omit planar projections that would introduce space-varying distortions and achieve rotational weight sharing as well as rotation equivariance.

3D representations The sensor output of rotating multi-layer laser scanner, e.g., the popular Velodyne laser scanners, is a cylindrical projection of the distances to the closest obstacles. This can be processed in form of a panoramic image (with a typical resolution of, e.g., 64×864 pixels) or transfered into a 3D structure. Although there are deep learning approaches based on these 3D structures, e.g., using voxel grids [15], point clouds [16], or 3D voting [17], the additional dimension increases computational complexity and efficient ways to deal with the sparsity of 3D data (i.e. on GPUs) are still subject of ongoing research. Boomsma et al. [2] apply 3D kernels for spherical convolutions to molecular structures in a discretized spherical region. In their provided implementation, they use circular convolutions. However, there is no investigation of the performance gain when using circular convolutions.

III. Circular Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [18] are a special kind of Artificial Neural Networks (ANNs) for processing grid-like data like, e.g., time-series data (1D), RGB or depth images (2D), or Point Clouds (3D). The CNN's architecture concatenates at least one convolutional layer with other layer types like fully-connected layers, batch normalization, or activation layers. This architecture allows to compute complex features of the input data for tasks like object classification, detection, segmentation, or reinforcement learning. A CNN's convolutional layer computes a linear convolution between an incoming feature map F_l of size $(H \times W \times C)$ and a kernel K of size $((2 \cdot M + 1) \times (2 \cdot N + 1) \times C)$ with $H, W, C, M, N \in \mathbb{N}$ to return a feature map F_{l+1}. A stride $s = [s_i, s_j]$ can be used to downsample the input. With indexes starting at 0, linear convolution can be computed as follows:

$$\begin{aligned} F_{l+1}(i,j) &= (F_l * K)_s(i,j) \\ &= \sum_{c=0}^{C-1} \sum_{m=-M}^{M} \sum_{n=-N}^{N} F_l(s_i \cdot i - m, s_j \cdot j - n, c) \\ &\qquad \cdot K(m+M, n+N, c) \qquad (1) \end{aligned}$$

This can be repeated with different kernels to output F_{l+1} with multiple channels. If the distance of (y, x) to the image border is smaller than the kernel radius in the corresponding direction, then the kernel's receptive field reaches beyond the image border. In this case either no output is generated and the feature maps shrinks (usually called a *valid* convolution) or a padding of the input image is conducted (usually called a *same* convolution). Since shrinking feature maps would limit the potential number of stacked convolutional layers, typical convolutional layers pad the feature map borders of F_l with just enough zeros to keep the size of the output F_{l+1} constant. Of course, this zero padding introduces a distortion in the filter response. In the following, we present the *Circular Convolutional Layer* and the *Circular Transposed Convolutional Layer* (also Circular Deconvolutional Layer) that can be used to overcome this distortion for panoramic data. They are the building blocks of *Circular Convolutional Neural Networks* (CCNN). CCNNs are identical to classical CNNs except for replacing linear convolutional layers with their circular counterparts. In particular, the input and output data formats are the same. Accordingly, existing CNN-architectures can be applied to panoramic data by simply replacing linear convolutional layers with circular convolutional layers. CCNNs are designed to be equivariant towards circular image shifts. To fully exploit this feature, a classifier, for instance, should be built with a 1×1-convolution with a subsequent Global Average Pooling [19] rather than with fully-connected layers. The similarity to CNNs can be used to generate CCNNs from readily trained CNNs, this is discussed in the final paragraph of this section.

A. *Circular Convolutional Layers*

To handle data with wrap-around structure circular convolution rather than linear convolution can be applied. Circular Convolution keeps the input size constant and performs downsampling solely with stride s. It computes the convolution from Equation 1 on periodic data using modulo division on the indexes of circular dimensions. E.g., for an image with two circular dimensions:

$$\begin{aligned} &F_{l+1}^{circ}(i,j) = (F_l^{circ} * K)_s(i,j) \\ &= \sum_{c=0}^{C-1} \sum_{m=-M}^{M} \sum_{n=-N}^{N} F_l(\mathrm{mod}(s_i \cdot i - m), \mathrm{mod}(s_j \cdot j - n), c) \\ &\qquad \cdot K(m+M, n+N, c) \qquad (2) \end{aligned}$$

For efficient implementation, the proposed Circular Convolutional Layer avoids the computation of indexes using modulo division. Instead, it pads the borders of feature maps with just enough content from the opposite sides of this feature map to avoid a feature map shrinking. Thus, it replaces the zero-padding with the correct content from the image on each *individual* layer. This is in contrast to one very large padding at the input image (coined input padding), where the same operations on redundant data would be applied. The layer wise circular padding prevents a suppression of kernel weights during convolution at the borders through zero-padding and, accordingly, the kernels show the same behavior at a feature map's borders and interior.

The circular convolution can be applied to data of arbitrary dimensionality and can be combined with linear convolution. E.g., for typical panoramic images, a circular convolution can be used in horizontal direction and a linear convolution with zero padding in vertical direction. In the following, we list the algorithmic steps to implement such a combined Circular Convolutional Layer.

1) Given input $I \in \mathbb{R}^{h_I \times w_I}$, determine padding widths p_{left}, p_{right}, p_{top} and p_{bottom}: In case of stride $s = 1$, each padding width equals the kernel radius N. For stride $s > 1$, the padding widths can be different for distinct frameworks. E.g., for Tensorflow[3], the padding rules are:

$$p_h = \begin{cases} \max(k_h - s_h, 0), & \text{if } h_I \bmod s_h = 0 \\ \max(k_h - h_I \bmod s_h, 0), & \text{otherwise} \end{cases}$$

$$p_w = \begin{cases} \max(k_w - s_w, 0), & \text{if } w_I \bmod s_w = 0 \\ \max(k_w - w_I \bmod s_w, 0), & \text{otherwise} \end{cases}$$

$$\begin{aligned} p_{\text{top}} &= \lfloor p_h/2 \rfloor \\ p_{\text{bottom}} &= p_h - p_{top} \\ p_{\text{left}} &= \lfloor p_w/2 \rfloor \\ p_{\text{right}} &= p_w - p_{left} \end{aligned}$$

2) Cut a pad P_{left} with width p_{left} from the right side of the input and a pad P_{right} with width p_{right} from the left side of the input, and concatenate them with the input:

$$\begin{aligned} P_{\text{left}} &= I_{0:h_I-1,(w_I-p_{\text{left}}):w_I-1} \\ P_{\text{right}} &= I_{0:h_I-1,0:(p_{\text{right}}-1)} \\ I_{\text{padded left-right}} &= \begin{bmatrix} P_{\text{left}} & I & P_{\text{right}} \end{bmatrix} \end{aligned}$$

3) Zero-pad top and bottom of the concatenated image with p_{top} at the top and p_{bottom} at the bottom:

$$I_{\text{padded}} = \begin{bmatrix} 0_{\text{top}} \in 0^{p_{\text{top}} \times (p_{\text{left}} + w_I + p_{\text{right}})} \\ I_{\text{padded left-right}} \\ 0_{\text{bottom}} \in 0^{p_{\text{bottom}} \times (p_{\text{left}} + w_I + p_{\text{right}})} \end{bmatrix}$$

4) Run a *valid* linear convolution with stride s on the zero-padded concatenated feature map I_{padded}. This shrinks the preprocessed input so that the output size matches a *same* convolution on the input.
5) Return this convolved feature map.

B. Circular Transposed Convolutional Layers

The Circular Transposed Convolutional Layer (or Circular Deconvolutional Layer) is the analogue to a Transposed Convolutional (or Deconvolutional) Layer. It is designed for generative purposes and upsamples its input by stride $s > 1$. A transposed convolution can be computed as linear convolution: In case of 2D input, upsampling can be done by adding $s - 1$ zero-lines in-between every row and column, and at one border for each dimension. The upsampled input is linearly convolved with a kernel K to receive a transposed convolution. Again, to perform a *same* convolution, the upsampled input has to be zero-padded. This zero-padding involves border effects which can be avoided by circular convolution in the Circular Transposed Convolutional Layer. The following steps are required to implement a combined layer with circular transposed convolution in horizontal direction and standard linear transposed convolution in vertical direction (in accordance with the above circular convolutional layer):

1) Given input $I \in \mathbb{R}^{h_I \times w_I}$, run a (regular) *valid* transposed convolution on the input. The output shape is

$$\begin{aligned} h_{\text{tconv}} \times w_{\text{tconv}} &= (h_I \cdot s_h + p_h) \times (w_I \cdot s_w + p_w) \\ \text{with} \qquad p_h &= \max(k_h - s_h, 0) \\ p_w &= \max(k_w - s_w, 0) \end{aligned}$$

2) The result I^{tconv} of the above valid transposed convolution provides the correct values for the inner parts; for the areas near the border, the corresponding circular influence from the oposite borders is missing. Therefore, add the first p_w left columns to the last p_w right columns, and add the last p_w right columns to the first p_w left columns:

$$\begin{aligned} I^{\text{tconv}}_{0:h_{\text{tconv}}-1,0:p_w-1} &\leftarrow I^{\text{tconv}}_{0:h_{\text{tconv}}-1,0:p_w-1} \\ &\quad + I^{\text{tconv}}_{0:h_{\text{tconv}}-1,w_{\text{tconv}}-p_w-1:w_{\text{tconv}}-1} \\ I^{\text{tconv}}_{0:h_{\text{tconv}}-1,w_{\text{tconv}}-p_w-1:w_{\text{tconv}}-1} &\leftarrow I^{\text{tconv}}_{0:h_{\text{tconv}}-1,0:p_w-1} \end{aligned}$$

3) To obtain the desired output size $h_I \cdot s_h \times w_I \cdot s_w$, remove $\lfloor p_w/2 \rfloor$ columns from the left and $p_w - \lfloor p_w/2 \rfloor$ columns from the right as well as $\lfloor p_h/2 \rfloor$ rows from the top and $p_h - \lfloor p_h/2 \rfloor$ rows from the bottom.
4) Return this cropped feature map.

The experiments in Sec. V use such combined 2D-versions for panoramic images with horizontal wrap-around connections. However, the extension to other 2D-, 1D-, and 3D-versions is straightforward.

C. Weight Transfer from CNN to CCNN

For each CNN, a corresponding CCNN can be created by replacing all (transposed) convolutional layers with their circular counterparts. The network architecture and all hyper-paramters remain unchanged. This allows to directly reuse all weights of an already trained CNN by *copying* them to an accordingly created CCNN. In the interior of feature maps, Circular Convolutional Layer and Circular Transposed Convolutional Layer show the same behavior like their linear convolution counterpart. At the image borders, the benefits of the circular convolutions can be exploited. Both effects are evaluated in the results section V. However, dedicated CCNN training can additionally avoid wasting network capacity to learn kernels particularly for border regions.

IV. Why not simply padding the input?

Input padding is another way to avoid the negative effects of zero-padding: wrap-around dimensions are padded *once* with (a potentially large chunk of) their corresponding content before processing. A CNN with input padding (coined CNN-IP) is expected to achieve comparable results to CCNNs and might even be beneficial for parallelization. However, there are some important drawbacks:

(1) Runtime increases with increasing input padding. Sec. V-C provides runtime measurements. The deeper the network, the more padding is required; e.g., for a quite shallow network, [3] padded the input image with 66.7% of the original image.

[3]documentation: https://tensorflow.org/api_guides/python/nn#Convolution

The accordingly increasing memory consumption on a GPU could also reduce its parallelization capability.

(2) It is nontrivial to figure out the minimum required amount of input padding to avoid zero padding; an upper bound can be calculated with equation (3). It guarantees that there is no influence of zero padding on the relevant output:

$$\text{pad}_{\text{width}} \leq (k_0 - 1) + \sum_{l=1}^{L} (k_l - 1) \cdot \frac{\prod_{i=0}^{l-1} d_i}{\prod_{i=0}^{l} u_i} \qquad (3)$$

It includes strides (d... downsampling, u... upsampling, defaults to 1 in the opposite case), kernel sizes k, and number of layers L. The actual required $\text{pad}_{\text{width}}$ also depends on padding approaches of all the layers in the network and on implementation details of the used learning framework. E.g., convolutional and transposed convolutional layers do not show inverse behavior for the same stride and kernel size with valid-padding in backends like Tensorflow. As a consequence, feature maps from different depths cannot be concatenated without additional modification. A possible workaround is to apply different kinds of upsampling layers like *sub-pixel convolution* or *NN-resize convolution* (see [20]). Moreover, in frameworks like Keras the padding method cannot be chosen seperately for each dimension. Thus, for non-wrapping dimensions, an additional zero-padding has to be done manually which could further increase computation time and memory consumption.

(3) In case of same-padding or deconvolutional layers, the final output has to be cropped to achieve the original output size. It is not obvious where to crop and the exact location possibly depends on the used backend.

(4) In a CNN with input padding, feature maps of intermediate layers contain repeated content which might influence the behavior of some layer types like Batch Normalization or Dropout layers.

V. Experiments

In this section, we evaluate some properties of circular convolutional neural networks and compare their performance to standard CNNs (with layer-wise zero padding), and the surrogate techniques for panoramic date (standard CNNs trained on circularly shifted images and CNNs with input padding). In all experiments the 2D circular convolutional layers and 2D circular transposed convolutional layers perform a circular convolution solely in the horizontal direction; in the vertical direction, a linear convolution is employed as in regular convolutional layers. Circular and linear convolutions in all used networks perform a *same* convolution (size remains constant). For down- and upsampling, a stride s is applied.

We implemented the neural network training and evaluation in the framework Keras with the Tensorflow backend. Our circular convolution layers including the transposed convolutional layer provide the same interface as the layers offered by Keras, therefore, regular layers can be replaced easily with their circular version.

A. Evaluation of shift invariance of CCNNs for circular data

This section serves two goals: First, we evaluate the capability of the proposed CCNN to provide responses which are invariant under circular image shifts. Therefore, we define a circular MNIST [21] classification task of grayscale handwritten digits. The dataset provides 60000 training and 10000 test images of size 28×28, each with a ground truth label of the shown digit 0 to 9. We assume the images to be circular: If a digit is horizontally shifted within the image across the image borders, it appears on the opposite side as depicted in Fig. 3. We combine the CCNN with a subsequent Global Average Pooling [19] classifier. Since the classifier is invariant towards shift in the input data, the combination with a CCNN should not be affected by circular shifts as well. The second purpose of this section is the comparison to the surrogate approach of extending the training data with circular shifted versions of the training images.

For circular MNIST experiments, we use a shallow all convolutional network [22] for both CNN and CCNN: We concatenate four Convolutional layers, either regular for the CNN or circular for the CCNN, with k kernels of size 3×3 (identical in every layer); in addition, the second and fourth layer perform a downsampling with stride $s = 2$. For classification, 1×1-convolution kernels with a subsequent Global Average Pooling and softmax activation are applied as described by [19]. The training for both CNN and CCNN is conducted for three runs with $k = \{4, 8, 32\}$ kernels in each of the four layers. The Adam optimizer with the Keras default parameters is used for loss minimization. In order to investigate the shift variance or invariance, the trained models are evaluated over all possible circular shifts of the 10000 test images. We run two experiments:

1) The neural networks are trained for 28 epochs with the 60000 default training images and labels.
2) The neural networks are trained for 1 epoch with a training dataset containing all 60000 training images in every of the 28 possible circular shifts, resulting in $28 \cdot 60000 = 1,680,000$ images.

The results are depicted in Fig. 2. As expected, regular CNNs show a test accuracy drop for higher shifts in all experiments. However, the CNN with higher capacity ($k = 32$) learned to generalize quite well using the shifted training dataset. The maximum accuracy of both CNN and CCNN is almost identical in all experiments. The most important result is the consistency of the CCNN accuracy over the circular shifts. For smaller models, especially for $k = 4$, CCNNs perform slightly better and learn faster than CNNs on panoramic data (cf. learning curves in Fig. 2). Presumably this is due to the fact that the CCNN wastes no capacity to model border effects. The slight oscillation of the CCNN-curves is caused by strides $s > 1$ in the architecture (alternation of four values due to two strides > 1).

The second row in Fig. 2 evaluates the surrogate CNN approach with extended training data. All CNNs benefit from the additional data (note the changed axis scales), but the shift dependency is still visible. This holds particularly for the smallest network with only 4 kernels - presumably, its capacity is too small to fully exploit the extended training data. The CCNN retains its superior properties.

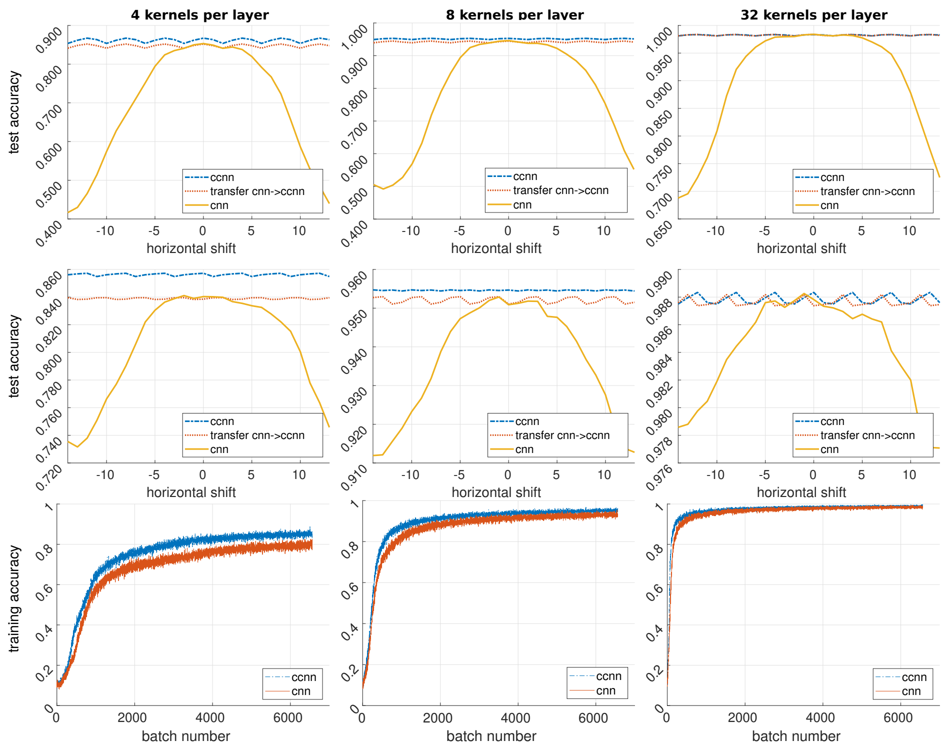

Fig. 2: Evaluation of the shift variance of CNNs and CCNNs trained on the circular MNIST dataset, and of weight transfer from trained CNNs to CCNNs (see Sec. V-D). The top and mid rows show the test accuracy of classification as a function of horizontal circular shift applied to all test images. The models are either trained on 1) the original 60000 training images (top row) or 2) on training images with all 28 possible circular shifts, i.e., $1,680,000$ training images (mid row). The models use either 4, 8, or 32 kernels in each convolutional layer (left to right). The bottom row shows the accuracy on the training set during training as a function of batch number. To address random initialization effects, all curves show the mean of the best 5 of 20 training runs per architecture. Note the different y-axis scalings.

B. *Performance gains with CCNNs near the image borders*

This section evaluates the performance gains near image borders when using a CCNN instead of a CNN. The following experiments build upon a real world dataset of car drives on highways recorded by a car equipped with a Velodyne HDL-64 3D laserscanner. This sensor provides panoramic depth and reflectivity images with 64 vertically aligned scanlines for a maximum distance of $120m$. The laserscanner spins with 20 Hz; we sample frames every 2 seconds, each consisting of 64×864 depth and reflectivity images. The dataset consists of 4093 training and 1164 test frames, each pixel is manually labeled $vehicle$ or $background$. Fig. 3 shows an example.

To perform a pixel-wise binary classification task we train two different neural network architectures with regular and circular convolutional layers:

1) **DeconvNet**: The DeconvNet from [6] is a Deconvolutional Neural Network with three convolutional layers for downsampling followed by three transposed convolutional layers for upsampling with shortcuts between layers of the same size. For training, the network is designed to output a classification for every upsampling layer. The network is implemented in *Keras* with *Tensorflow* backend. Except for a 5% dropout layer after the input to mimic laserscanner misclassifications, we copied the network's architecture including layers, filter number and size, and strides from [6]. The loss for vehicle misclassifications is multiplied by approx. 16, and the losses for the three outputs after each upsampling layer are weighted with $\{1, 0.7, 0.5\}$. The network is trained for 20 epochs with batch size 32 with the Adam optimizer; we used a scheduled learning rate: $\{$epoch $0-7$: 0.001, $8-10$: 0.0005, $11-14$: 0.00025, else: $0.00001\}$

2) **TiramisuNet**: The TiramisuNet from [7] is a quite deep Deconvolutional Neural Network with multiple blocks of downsampling and upsampling layers. A downsampling is obtained by a 1×1 convolution with subsequent max-pooling, an upsampling by a transposed convolution with stride 2. Before every downsampling layer and after every upsampling layer a densely-connected convolution [23] over multiple layers is performed. We partially adapted the 103 layer architecture described in the paper: Except for the bottleneck with 15 layers, we always use 5 convolutional layers in a densely-connected convolution block. Our TiramisuNet consists of 77 convolutional layers. Again, the loss for a vehicle misclassification is multiplied by approx. 16, and the network was trained with Adam optimizer for 5 epochs with batch size 2 and learning rate 0.001.

To evaluate the expected performance gain of CCNNs especially close to the image borders, we investigate the average precision for increasingly large areas around the borders: First, the classes of all pixels in all test images are predicted and stored. Next, average precisions are computed by comparing predictions and ground truth for an increasing area around the image border from the left and right side. The results are shown in Figure 4: In both experiments, there is a visible benefit of the CCNN over the CNN in areas near the image borders. For the better performing Tiramisu network, the gain is more than 20% (0.16 absolute) for a small area close to the border. When the area is increased

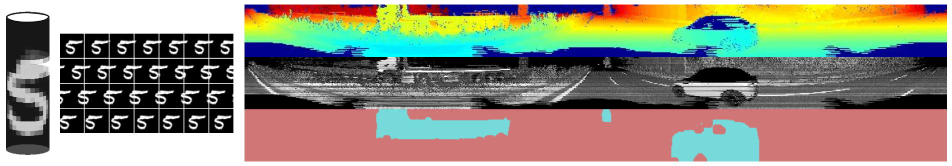
Fig. 3: *(left)* Visualization of a circular MNIST digit and all possible circular shifts. *(right)* An example frame from the dataset recorded with a *Velodyne HDL-64* 3D laserscanner on highways. Top: depth image; mid: reflectance image; bottom: label image with classes {vehicle, background}.

towards the image interior, the results converge to the CNN's performance. For the DeconvNet the difference is smaller than for the TiramisuNet which is much deeper; presumably, the benefit of circular convolutional layers is higher in deeper architectures since the influence of zero-padding spreads out more with increasing depth. Following equation (3), the maximum influence of zero-padding for DeconvNet is ≤ 60 pixel around the seam or ≤ 1706 pixel for TiramisuNet, respectively. In a defined setting with uniform weights and zero-bias, we could measure an actual impact of zero-padding of 46 pixel for DeconvNet and 864 pixel (full image width) for TiramisuNet. This can be observed for DeconvNet in Fig. 4 where CNN and the transfered CCNN perform equal from approximately 48 pixel. Note that this area is independent of the actual image width - the arbitrarily large interior of the image is not affected.

C. Runtime considerations

Circular convolutional layers require more copy operations than linear convolutional layers to replace the zero-padding with padding of the opposite input content. This results in longer training and inference times. We implemented our neural network models in the Keras framework with Tensorflow backend, and trained and tested all models with a *Nvidia GTX 1080 Ti* GPU. The deepest model from the experiments (TiramisuNet (see Sec. V-B)) requires for training with batch size 2 and 4093 images 655 seconds per epoch for a CNN (160ms/image) and 756 seconds per epoch for a CCNN (185ms/image); an increase of 16% in training time. For inference (to classify images pixel-wise), the TiramisuNet requires for 1164 images 59.1 seconds on a CNN-version (51ms/image) and 65.6 seconds on a CCNN (56ms/image); an increase of about 11%. According to Table I this corresponds to the runtime caused by input padding (CNN-IP) of about 15%. The required input padding to avoid the negative effect of zero padding is typically much larger (e.g. [3] increase the number of pixels by 66.7%, also see experiments in Fig. 5).

D. Transfer from trained CNN to CCNN

All experiments include results for weight transfer, i.e. training a CNN and copying the weights to CCNN without retraining. This could be reasonable for applying readily trained CNNs on circular data or to save the 16% additional training time for training a CCNN instead of a CNN. In the shift invariance experiments on MNIST in Fig. 2, the CNN's test accuracy drops for higher circular shifts whereas the transfered CCNNs show the same maximal accuracy and retain it across shifts. In the Velodyne data experiment, the transfered CCNNs show a performance gain next to the image border and converge to their parent CNN's average precision for higher distance to the border. In conclusion, the transfer improved the CNN's performance in every presented task without any additional training.

E. Comparison of CCNN and CNN-IP (with Input Padding)

Fig. 5 shows the achieved average precision in the area near the unfolded image border for the different network types including CNNs with varying input padding widths. Table I shows the corresponding input widths, training and testing times, and the average precision next to the border. The increasing input padding widths are chosen to fit the Tiramisu network architecture which halves the input five-times before upsampling; accordingly the input is a multiple of 2^5. We chose the different widths to cover the range from the default input width to an empirically determined reasonable maximum. In general, wider padding improves the performance. However, the pitfalls of using input padding described in Sec. IV can considerably degrade the performance. For example, CNN-IP with width 1504 performs worse than smaller input paddings. The increased CNN-IP input requires a cropping of the also increased output. In our experiments, we use the accordingly sized central part of the image. This is a reasonable but presumably not correct choice for this particular image width, caused by implementation details of the underlying learning framework and resulting in degraded performance. This example and their general better performance promote the usage of the described CCNN instead of input padding.

VI. Conclusion

In this work, we discussed Circular Convolutional Neural Networks as an extension of standard Convolutional Neural Networks. The described circular convolutional and circular transposed convolutional layers can be used to replace their linear counterparts in CNN-architectures for panoramic input images and laser data. On a circular MNIST dataset, we could show the CCNN's shift invariance: the trained CCNNs performed always equal or better than the CNN-counterpart, even if the CNN was additionally trained on shifted images; for models with less kernels the CCNN even outperformed the CNN for *unshifted* images which indicates more efficient exploitation of convolution kernels. On the practically more relevant multi-layer LiDAR dataset, we could show and quantify the performance gain of CCNNs in the area near the image border of panoramic images. The application to Velodyne scans from driving cars demonstrated the practical relevance to prevent blind spots. CCNNs perform better and are faster to train than approaches based on input padding or augmented training sets with circular shifts. However, they are slightly slower (training 16%, inference 11%) than standard CNNs *with* blind spots. For training, we demonstrated how weight transfer between CNN and CCNN can eliminate the

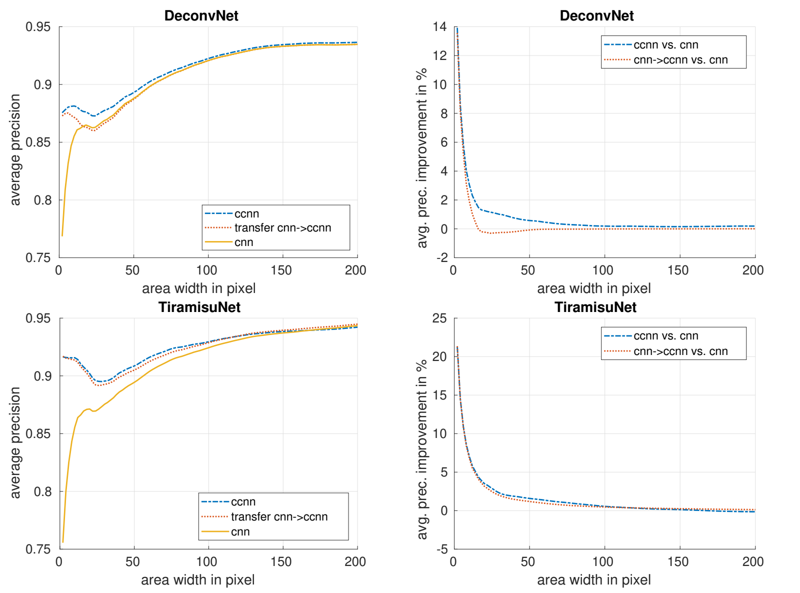

Fig. 4: Evaluation of the DeconvNet (top) and TiramisuNet (bottom) trained on the Velodyne dataset with linear or circular convolutional layers, and of trained CNNs transfered to CCNNs. *(left)* Average precision between model prediction and ground truth as a function of the area around the image border of the unfolded panoramic classification images. *(right)* Relative improvement of average precision of the trained or transfered CCNN over the CNN. Again, all curves show the mean of the best 5 of 20 trained models to address effects of random weight initializations. Presumably, the local minima at ~25 pixels are caused by the characteristics of the dataset (spatial distribution and appearance of objects around the seam); in not shown experiments we validated that this can be shifted by moving the image seam.

TABLE I: Comparison of training and testing time, and the average precision (AP) next to the image border. Each value represents the mean of five models to address random weight initialization. CNN* marks the same neural networks.

Network Type	**Input Width**	**Training Time**	**Testing Time**	**Border AP**
CCNN	**864**	185ms	56.4ms	**0.917**
CNN*	**864**	**160ms**	**50.8ms**	0.708
CNN*→CCNN	**864**	**160ms**	56.4ms	0.890
CNN-IP	928 (+7.4%)	64ms	52.7ms	0.845
CNN-IP	992 (+14.8%)	174ms	55.6ms	0.838
CNN-IP	1184 (+37%)	210ms	65.8ms	0.893
CNN-IP	1504 (+74%)	264ms	84.5ms	0.853
CNN-IP	1632 (+88.9%)	287ms	90.6ms	0.891

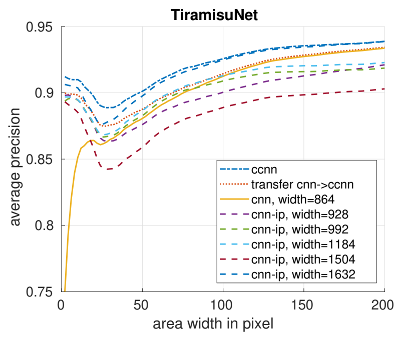

Fig. 5: Average precision as a function of the area around the image border of the unfolded panoramic classification images for different network types and CNNs with input padding (CNN-IP). All curves are mean of best 5 of 9 trained models.

runtime increase. It can also be used to convert available, readily trained CNNs for which retraining is not possible (e.g. due to unavailable training data or resources) into CCNNs. If blind spots are not acceptable, there is no reason to stick with techniques like input padding instead of using a Circular Convolutional Neural Network.

References

[1] H.-T. Cheng, C.-H. Chao, J.-D. Dong, H.-K. Wen, T.-L. Liu, and M. Sun, "Cube padding for weakly-supervised saliency prediction in 360° videos," in *IEEE CVPR*, 2018.

[2] W. Boomsma and J. Frellsen, "Spherical convolutions and their application in molecular modelling," in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 3433–3443.

[3] B. Shi, S. Bai, Z. Zhou, and X. Bai, "Deeppano: Deep panoramic representation for 3-d shape recognition," *IEEE Signal Processing Letters*, vol. 22, no. 12, pp. 2339–2343, Dec 2015.

[4] S.-C. B. Lo, H. Li, Y. Wang, L. Kinnard, and M. T. Freedman, "A multiple circular path convolution neural network system for detection of mammographic masses," *IEEE Tran. on Medical Imaging*, 2002.

[5] R. Arandjelović, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "NetVLAD: CNN architecture for weakly supervised place recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[6] V. Vaquero, I. del Pino, F. Moreno-Noguer, J. Solà, A. Sanfeliu, and J. Andrade-Cetto, "Deconvolutional networks for point-cloud vehicle detection and tracking in driving scenarios," in *2017 European Conference on Mobile Robots (ECMR)*, Sept 2017, pp. 1–7.

[7] S. Jégou, M. Drozdzal, D. Vázquez, A. Romero, and Y. Bengio, "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation," *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1175–1183, 2017.

[8] K. Sfikas, T. Theoharis, and I. Pratikakis, "Exploiting the panorama representation for convolutional neural network classification and retrieval," in *Eurographics Workshop on 3D Object Retrieval*, 2017.

[9] M. Danelljan, G. Häger, F. S. Khan, and M. Felsberg, "Learning spatially regularized correlation filters for visual tracking," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2015.

[10] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through ffts," *CoRR*, vol. abs/1312.5851, 2013.

[11] K. Sakurada and T. Okatani, "Change detection from a street image pair using CNN features and superpixel segmentation," in *BMVC*. BMVA Press, 2015, pp. 61.1–61.12.

[12] Y.-C. Su and K. Grauman, "Learning spherical convolution for fast features from 360° imagery," in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 529–539.

[13] L. Ran, Y. Zhang, Q. Zhang, and T. Yang, "Convolutional neural network-based robot navigation using uncalibrated spherical images," *Sensors*, vol. 17, no. 6, 2017.

[14] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling, "Spherical CNNs," in *International Conference on Learning Representations*, 2018.

[15] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2015.

[16] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017.

[17] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner, "Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks," in *ICRA*. IEEE, 2017, pp. 1355–1361.

[18] Y. LeCun, "Generalization and network design strategies," University of Toronto, Tech. Rep., 1989.

[19] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[20] A. P. Aitken, C. Ledig, L. Theis, J. Caballero, Z. Wang, and W. Shi, "Checkerboard artifact free sub-pixel convolution: A note on sub-pixel convolution, resize convolution and convolution resize," *CoRR*, vol. abs/1707.02937, 2017.

[21] Y. LeCun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard, "Handwritten digit recognition: applications of neural network chips and automatic learning," *IEEE Communications Magazine*, vol. 27, no. 11, 1989.

[22] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," in *ICLR (workshop)*, 2015.

[23] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *IEEE Conf. on Comp. Vision a. Pattern Recognition (CVPR)*, 2017.