Physics-Inspired Adaptions to Low-Parameter Neural Network Weather Forecast Systems

SEBASTIAN SCHER\textsuperscript{a,b,c} AND GABRIELE MESSORI\textsuperscript{b,c,d,e}

\textsuperscript{a} Know-Center Research GmbH, Graz, Austria
\textsuperscript{b} Department of Meteorology, Stockholm University, Stockholm, Sweden
\textsuperscript{c} Bolin Centre for Climate Research, Stockholm University, Stockholm, Sweden
\textsuperscript{d} Department of Earth Sciences, Uppsala University, Uppsala, Sweden
\textsuperscript{e} Centre of Natural Hazards and Disaster Science, Uppsala University, Uppsala, Sweden

(Manuscript received 9 June 2023, in final form 3 November 2023, accepted 7 November 2023)

ABSTRACT: Recently, there has been a surge of research on data-driven weather forecasting systems, especially applications based on convolutional neural networks (CNNs). These are usually trained on atmospheric data represented on regular latitude–longitude grids, neglecting the curvature of Earth. We assess the benefit of replacing the standard convolution operations with an adapted convolution operation that takes into account the geometry of the underlying data (SphereNet convolution), specifically near the poles. Additionally, we assess the effect of including the information that the two hemispheres of Earth have “flipped” properties—for example, cyclones circulating in opposite directions—into the structure of the network. Both approaches are examples of physics-informed machine learning. The methods are tested on the WeatherBench dataset, at a resolution of \(-1.4^\circ\), which is higher than many previous studies on CNNs for weather forecasting. For most lead times up to day +10 for 500-hPa geopotential and 850-hPa temperature, we find that using SphereNet convolution or including hemisphere-specific information individually leads to improvement in forecast skill. Combining the two methods typically gives the highest forecast skill. Our version of SphereNet is implemented flexibly and scales well to high-resolution datasets but is still significantly more expensive than a standard convolution operation. Finally, we analyze cases with high forecast error. These occur mainly in winter and are relatively consistent across different training realizations of the networks, pointing to flow-dependent atmospheric predictability.

KEYWORDS: Atmosphere; Forecasting; Statistical forecasting; Artificial intelligence; Machine learning; Neural networks

1. Introduction

Weather forecasting has for decades been dominated by numerical models built on physical principles, the so-called numerical weather prediction models (NWP). These models have seen a constant increase in skill over time (Bauer et al. 2015). Recently, however, there has been a surge of interest in data-driven weather forecasting in the medium-range (~2–14 days ahead). These have often, but not exclusively, used neural networks (e.g., Scher 2018; Scher and Messori 2019b; Dueben and Bauer 2018; Weyn et al. 2019, 2020; Faranda et al. 2021; Scher and Messori 2020; Rasp and Thuerey 2021; Bi et al. 2022; Keisler 2022; Pathak et al. 2022; Chen et al. 2023; Lam et al. 2023; Ben-Bouallegue et al. 2023), also in combination with physics-based models (e.g., Arcomano et al. 2022). A historic overview of paradigms in weather prediction is outlined in Balaji (2020). The use of convolutional neural networks (CNNs; e.g., Scher 2018; Scher and Messori 2019b; Weyn et al. 2019; Rasp and Thuerey 2021) or of a local network that is shared across the domain (Dueben and Bauer 2018), dominated in the early data-driven approaches. What these methods have in common is that they use global data on a regular latitude–longitude grid. This leads to distortions, especially close to the poles (Coors et al. 2018). However, a standard convolution or shared local architecture does not take such distortion into account since it uses a filter whose size is a fixed number of grid points (e.g., \(3 \times 3\)). Therefore, the area that the filter sees is not the same close to the equator and close to the poles. Weyn et al. (2020) have proposed a solution to this problem via working on a different grid. Specifically, they regard the data to a “cubed sphere” consisting of six different regions. Then, they use a standard convolution operation on each side of the cubed sphere. Additionally, they do not share the weights of the filters globally [as in the original architecture proposed by Scher (2018) and adapted to real-world data by Weyn et al. (2019)], but instead use an independent convolution operation for the different sides of the cubed sphere. The weights are shared only for the two polar parts of the cubed sphere but then “flipped” from one pole to the other to account for the different direction of rotation.

There are also other possibilities for including the spherical nature of Earth in neural network (NN)–based weather prediction models. Most of the currently best-performing models use either transformer-based methods (Bi et al. 2022) or graph neural networks (GNNs) (Keisler 2022; Lam et al. 2023). The approaches with GNNs consider the spherical nature of Earth via mapping into an abstract feature space on an icosahedral grid (Keisler 2022), or on grids on a multimesh representation (Lam et al. 2023). The transformer architecture of Bi et al. (2022) uses a “3D Earth-specific transformer.” In practice, the spherical nature is dealt with an Earth-specific positional bias in the transformer.
A drop-in replacement for convolution layers is presented in Esteves et al. (2023). They use a highly optimized form of spherical convolution based on spherical harmonics and showed it can successfully be integrated into machine learning (ML) models for weather forecasting, having a skill comparable to state-of-the-art architectures.

In this paper, we present an alternative approach to incorporate the spherical nature of Earth into CNNs. We use the SphereNet architecture, which has previously been proposed for classification tasks on 360° images (Coors et al. 2018). Additionally, we test two different approaches for including information on the characteristics of the two hemispheres into our networks. All these approaches can be seen as variants of “informed machine learning” (von Rueden et al. 2020; Kashinath et al. 2021), in which prior knowledge is included into the machine learning pipeline. In our case, the prior knowledge is that Earth is spherical (in contrast to the regular data that we provide), and that the dynamics of the two hemispheres are—to some extent—“flipped” relative to each other. This information is directly encoded into the structure of the neural network.

Many of the latest transformer and GNN-based weather predictions models come with high technical and computational demands. While these architectures can outperform CNNs, we see a benefit in testing the limits of CNNs with a limited number of parameters, especially for research groups who may not have enough computational capacity to run very large transformer or GNN models. Therefore, the aim of this paper is not to create the best data-driven weather forecasts (and indeed we note that there are published architectures that perform more skillful predictions). Instead, we present two possible improvements to older CNN-based models and disentangle their individual contributions to forecast skill. The improvements presented here are conceptually simple, and stem from physical reasoning. We believe that the experiments presented here will support future development of physics-oriented ML weather prediction models, despite the fact that large GNN and transformer-based models currently have superior skill. We use a test bed reanalysis data from the WeatherBench dataset (Rasp et al. 2020), at a resolution of up to 1.4°. This is higher than many previous studies that used CNNs. Finally, we include an analysis of the events with the highest forecast errors. These are important from an end-user point of view, and in NWP models they have elicited significant attention. The occurrence of unusually bad forecasts (“forecast busts”) in NWP models is connected with specific weather situations (Rodwell et al. 2013; Lillo and Parsons 2017), and more generally, different weather situations have different predictability (Ferranti et al. 2015; Matsueda and Palmer 2018). We analyze whether the poorest forecasts of our data-driven forecast systems are, as in NWP, associated with specific weather situations.

### 2. Methods

#### a. Data

We use data from WeatherBench (Rasp et al. 2020). This is a dataset specifically designed for benchmarking machine learning-based weather forecasts. The subset we use consists of ERA5 reanalysis data, regridded to a regular latitude–longitude grid with two different resolutions: 2.8125° [low-resolution (lres)] and 1.40625° [high-resolution (hres)]. The following input variables are used: temperature at 850 hPa, geopotential at 300, 500, 700, and 1000 hPa, as well as top-of-the-atmosphere incident solar radiation. As evaluation variables, we use geopotential at 500 hPa (z500) and temperature at 850 hPa (t850). As additional time-invariant input variables, the land–sea mask and orography from ERA5 are used. We use the period 1979–2016 for training and validation, and 2017/18 for evaluation (as proposed in WeatherBench). Following WeatherBench, all results presented in this paper are on the independent evaluation data that is not used for training and validation. The temporal resolution of the data is 6 h. An overview of the input data is given in Table 1.

#### b. SphereNet convolution

In normal convolution, for each grid point, a fixed number of grid points in the vicinity are sampled (e.g., a $3 \times 3$ box centered on the grid point). For data on a globe (such as global atmospheric data) represented on a regular grid, this leads to distortions except very close to the equator. Indeed, a fixed neighborhood defined via the number of grid points corresponds to rectangles of differing size, depending on latitude. One way to remedy this is through the use of adapted convolution filters. Coors et al. (2018) proposed the SphereNet architecture in neural networks for image detection in spherical images.

With this method, instead of a box of fixed size in gridpoint space, each grid point is assigned a rectangle with fixed size in real space. Since the positions of available grid points and the target coordinates in this fixed-size box do not necessarily coincide, the target can also be an interpolation of up to four grid points. The principle is sketched in Fig. 1a.

In normal convolution operations in neural networks, the kernel is always made up of exact grid points (e.g., a $3 \times 3$ box corresponding to nine grid points). Our approach requires interpolation, and thus our method also allows the use of a non-integer number of grid points in the kernel. For example, we could use the distances $-1, -0.5, 0, 0.5,$ and 1 along the longitude direction, creating a kernel of five grid points. These five points then cover, with equal spacing, the distance that is

### Table 1. Overview of input data variables and spatial and temporal resolutions.

<table>
<thead>
<tr>
<th>Input variable</th>
<th>Spatial resolution lres/hres</th>
<th>Temporal resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperature at 850 hPa</td>
<td>2.8125°/1.40625°</td>
<td>6 h</td>
</tr>
<tr>
<td>Temperature at 500 hPa</td>
<td>2.8125°/1.40625°</td>
<td>6 h</td>
</tr>
<tr>
<td>Geopotential at 850 hPa</td>
<td>2.8125°/1.40625°</td>
<td>6 h</td>
</tr>
<tr>
<td>Geopotential at 500 hPa</td>
<td>2.8125°/1.40625°</td>
<td>6 h</td>
</tr>
<tr>
<td>Top-of-atmosphere incident radiation</td>
<td>2.8125°/1.40625°</td>
<td>6 h</td>
</tr>
<tr>
<td>Orography</td>
<td>2.8125°/1.40625°</td>
<td>Time-invariant</td>
</tr>
<tr>
<td>Land–sea mask</td>
<td>2.8125°/1.40625°</td>
<td>Time-invariant</td>
</tr>
</tbody>
</table>
normally covered by three kernel points (this distance that is, in kilometers, the distance of three grid points at the equator).

An alternative method for adapting convolution operators to Earth’s approximately spherical geometry that would seem promising in our case was proposed by Boomsma and Frellsen (2017), who presented convolution on a cubed sphere, where the sphere is divided into 6 parts. This is the same approach as in Weyn et al. (2020), but with weight sharing across all 6 parts. The main limitation of this approach is that translational invariance of the convolution filters is lost. Specifically, in our application, the filters would not necessarily be aligned along the longitude and latitude circles. This is a potential problem, as there is physical meaning to these (a gradient in a certain variable along the latitude dimension is physically not the same as along the longitude dimension).

Cohen et al. (2018) compute spherical convolution via FFTs, resulting in full rotational invariance. As noted by Coors et al. (2018) this is not always a desired property, as it assumes that all directions (e.g., in our case along longitude, along latitude or any other direction) are equivariant. As we discussed above, in our setting this is not physically meaningful. Eder et al. (2019) present an approach that generalizes the method from Coors et al. (2018) to any type of structured data. Jiang et al. (2019) present an approach that works for unstructured grids.

Due to the limitations of the methods from Boomsma and Frellsen (2017) and Cohen et al. (2018) discussed above, and the fact that for our setting we do not need the generalized approaches from Eder et al. (2019), and Jiang et al. (2019), we opted for using the approach from Coors et al. (2018). We will refer to this approach as SphereNet convolution.

**IMPLEMENTATION**

Since Coors et al. (2018) have not provided details on their technical implementation, and since their code is not publicly available, we have designed our own implementation of SphereNet. In this section we use the word “tensor” as it is used in computational packages such as TensorFlow, thus interchangeably with “array.” Therefore, not everything referred to as a tensor here is necessarily a tensor in the strict mathematical sense of the term.

We have implemented SphereNet with the following steps (the channel dimension of the neural network is omitted here for simplification):

1) We start with a (fixed) filter kernel \( K \) of length \( n \), consisting of \( n \) pairs of latitude–longitude distances \( \Delta p_i = (\Delta y_i, \Delta x_i) \), corresponding to grid points at the equator. A \( 3 \times 3 \) kernel without fractional distances for example would be \([-1, -1], (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -1), (1, 0), (1, 1)]\).

2) For each of the \( N \times M \) input grid points \( p = (x, y) \) in the regular grid, we compute \( n \) pairs of (potentially noninteger) coordinates \( p' = (y', x') \), corresponding to the \( n \) points in the kernel \( K \), transformed for the current position of \( p \) on the globe with the following equations;
x and y are in gridpoint coordinates, and $\phi$ and $\theta$ are in radians:

$$\phi' = \arcsin \sin \phi + \frac{\Delta \phi \sin \nu \cos \phi}{\rho},$$

(1)

$$\theta' = \theta + \arctan \left( \frac{\Delta x \sin \nu \cos \phi \cos \nu - \Delta y \sin \phi \sin \nu}{\rho} \right),$$

(2)

$$\rho = \sqrt{(\Delta \rho)^2 + (\Delta \theta)^2},$$

(3)

$$v = \arctan \rho,$$

(4)

$$\Delta \phi = \Delta y \frac{\pi}{N},$$

(5)

$$\Delta \theta = \Delta x \frac{2\pi}{M},$$

(6)

$$y' = \phi' \frac{M}{\pi} + \frac{N - 1}{2}, \text{ and}$$

(7)

$$x' = \theta' \frac{M}{2\pi},$$

(8)

with latitude $\phi$ of the central point. The transformed coordinates $\phi'$ and $\theta'$ are in regular latitude-longitude coordinates, and $x'$ and $y'$ the corresponding coordinate indices on the regular latitude-longitude grid. The transformed coordinate indices for each grid point and kernel points are combined in a coordinate tensor $\hat{A}$ of shape $N \times M \times n$.

3) The input $x$ data are flattened to $x_{\text{flat}}$ with shape $L = N \times M$, and the coordinate tensor $\hat{A}$ is flattened to a tensor of shape $L \times n$, with the coordinates transformed to flattened coordinates.

4) A sparse interpolation tensor $\hat{L}$ of size $L \times L$ is created, and filled with the target coordinates in such a way that multiplying the flattened input data $x_{\text{flat}}$ with the interpolation tensor results in the expanded input data $x_{\text{exp}} = \hat{L} x$ with shape $L \times n$. Here, $\hat{L}$ is implemented as a sparse TensorFlow tensor. This implementation allows the use also on very large grids (large $L$), as only the nonzero components are kept in memory.

5) On $x_{\text{exp}}$, a standard 1D convolution with kernel size $n$ (as implemented in major neural network libraries such as TensorFlow) can now be applied, resulting in $x_{\text{out}}$ with shape $L \times n$. Steps 1 and 2, and the computation of the interpolation tensor, need to be performed only once (when setting up the network). The tensor $\hat{L}$ is stored in memory for all subsequent operations.

At grid points close to the poles, kernel points can “pass” through the pole. For these points, not only the longitude, but also the latitude is adjusted. For example, on a $1^\circ \times 1^\circ$ grid, a kernel point that without this adjustment would correspond to the impossible point $90.5^\circ \text{N} - 0^\circ$ will be set as $89.5^\circ \text{N} - 180^\circ$. With this, the “polar problem” of regular grids is eliminated.

c. Neural network architecture

We use a neural network architecture based on that proposed in Weyn et al. (2020), namely, a U-Net architecture. Weyn et al. (2020), however, do not use data on a regular grid, but on a cubed sphere, consisting of several regular grids. They further use two sets of weights, across six different regions. We use the same architecture, but with each of their special convolution layers replaced by a standard convolution, SphereNet convolution and/or hemisphere-wise convolution (see below). The network structure is shown in Fig. 1b). Our networks are implemented in TensorFlow (Abadi et al. 2016) using the dataset application programming interface (API) with TensorFlow record files, resulting in an implementation that should also scale to datasets with higher resolution than the ones used here.

In addition to the input variables from ERA5 discussed in section 2a, day of the year (doy) and local hour of the day (hod; different for each longitude band) are used as additional input variables. Since these are “circular” variables, each of them is converted to two variables. Using two variables should make it easier for the model to learn the circular relationship (e.g., that doy 1 and doy 365 are adjacent days):

$$\text{doy1} = \sin \left( \frac{2\pi}{365} \text{doy} \right),$$

(9)

$$\text{doy2} = \cos \left( \frac{2\pi}{365} \text{doy} \right),$$

(10)

$$\text{hod1} = \sin \left( \frac{2\pi}{24} \text{hod} \right), \text{ and}$$

(11)

$$\text{hod2} = \cos \left( \frac{2\pi}{24} \text{hod} \right).$$

(12)

These four scalars are extended to the grid resolution of the data and added as additional channels. The input of the networks is composed of two time steps of the six ERA5 inputs (resulting in 12 input channels), but the additional four variables (doy1, doy2, hod1, hod2) are provided only once, resulting in $12 + 4 = 16$ input channels. The outputs of the networks are two time steps of the input variables, without the additional variables (thus 12 channels). One forecast step is made of two consecutive passes through the network, via feeding the output back to the input, resulting in a 24-h forecast. For details, see Weyn et al. (2020).

For consecutive forecasts (longer than 24 h), hod is not updated, since each forecast step is 24 h. We also choose not to update doy, since the forecast length of 10 days is very short compared to seasonal variations.

1) Base architecture

Our base architecture without SphereNet convolution uses normal convolution with wrapping on the sides. Along the longitude direction the convolution is “wrapped” around, so there is no artificial boundary. At the poles the grid is wrapped over the pole: the northern neighbor of a point on
the northernmost latitude band is the point on the same latitude band but with 180° shifted longitude. The kernel size of the convolutions is $3 \times 3$. Note that for the different resolution datasets, this corresponds to different spatial extents in the input data.

2) SPHERENET CONVOLUTION ARCHITECTURE

The SphereNet convolution architecture is the same as the base architecture, except that each convolution operation is replaced by a SphereNet convolution operation. Since the convolution deals both with the poles and the longitude wrap, no padding is applied. We use a $3 \times 3$ kernel, just as in the base architecture.

3) HEMISPHERIC CONVOLUTION

We use two related approaches for incorporating the fact that there are two hemispheres into the networks. In the first, we use separate (independent) convolution operations (with separate weights) for each hemisphere. The data is split at the equator. For the architecture without SphereNet convolution, the first row of the other hemisphere is added as padding for the boundary of the convolution, and the padding at the pole is handled in the same way as in the base architecture. When using SphereNet convolution together with hemispheric convolution this is not necessary, as this is included in the interpolation for the SphereNet convolution. Then, on each hemisphere, a convolution operation is performed. These will be referred to as “hemconv” and “sphereconv_hemconv,” respectively. In the second approach, the same convolution operation is used for both hemispheres, with the filter “flipped” along the latitude dimension for the second hemisphere. This will be referred to as “hemconv_shared” and “sphereconv_hemconv_shared.” This approach is a variant of the inclusion of “invariances” into the neural network in the terminology of von Rueden et al. (2020). The rationale behind this flipping approach is that many properties of the atmospheric circulation have flipped properties on one hemisphere compared to the hemisphere. For example, many variables—including temperature and geopotential—have significant pole-to-equator gradients that change sign between the hemispheres. Additionally, midlatitude weather systems spin in opposite directions in the two hemispheres, which manifests itself in the temperature and geopotential fields, among others. Note that when sharing the weights, but not flipping the weights, one would end up with the base architecture again. For both hemconv and hemconv_shared (and sphereconv_hemconv and sphereconv_hemconv_shared, respectively), for each convolution operation, the same convolution depth (=number of filters) is used. Therefore, the networks of hemconv and sphereconv_hemconv have twice the number of parameters compared to the other architectures.

4) ADDITIONAL EXPERIMENTS

In addition to using all architectures described above, we made two additional experiments on the low-resolution data. In the first experiment we added latitude and longitude grids as constant input features to the input data. Latitude was scaled to $[0, 1]$, while the longitude $\theta$ (in radians) was, just as day of the year and hour of the day, presented as two variables:

$$\text{lon1} = \sin(\theta), \quad \text{and} \quad \text{lon2} = \cos(\theta). \quad (13)$$

This approach we call base_latlon. In the second experiment, called hemconv_halfsize, we use hemispheric convolution without filter sharing, but with half the number of parameters per hemisphere compared to the base architecture. With this, the network in total has the same number of parameters as the base-architecture and all other architectures except for hemconv_shared and sphereconv_hemconv_shared. An overview of network architectures and the number of neural network parameters is given in Table 2.

5) NETWORK TRAINING

For the training, the data from WeatherBench is converted to the tensorflow-record format. Each network is trained four times with different random seeds to account for the randomness in the training. Each training realization is evaluated separately, and throughout the paper the average of the errors and skill scores is shown. The same architecture is used for both high- and low-resolution data. Only the input size is adjusted according to the resolution. Since the architecture is a pure convolution architecture, the number of parameters (weights) is independent of the input size, and thus both for high-resolution and low-resolution the same number of parameters is used (336 040 for architectures with same weights for both hemispheres, and 671 816 for the architectures with independent weights for each hemisphere). We train the networks first for 100 epochs, and then over an additional 50 epochs with early stopping (stopping after no increase in skill at 10% of the training data left out for validation). The latter step is done to prevent overfitting of the model.

d. Forecast evaluation

We use both root-mean-square error (RMSE) and anomaly correlation coefficient (ACC), which are also the two evaluation measures used in WeatherBench.

RMSE is defined as

$$\text{RMSE} = (\text{fc} - \text{truth})^2, \quad (16)$$

with the overbar representing latitude-weighted area and time mean, and ACC as

\begin{table}
\centering
\caption{Overview of network architectures.}
\begin{tabular}{ll}
\hline
Architecture & Parameters \\
\hline
basenet & 336 040 \\
sphereconv & 336 040 \\
hemconv & 671 816 \\
hemconv_halfsize & 336 040 \\
hemconv_shared & 336 040 \\
sphereconv_hemconv & 671 816 \\
sphereconv_hemconv_shared & 336 040 \\
basenet_latlon & 336 040 \\
\hline
\end{tabular}
\end{table}
with the correlation computed with latitude weights and clim being the time mean over all forecasts. For further details on the calculations, we refer the readers to Rasp et al. (2020).

3. Results

a. Evaluation of the different forecast architectures

We start by looking at global average RMSE and ACC, shown in Figs. 2 and 3 and Table 3. An alternative version of Fig. 2 is shown in the appendix. The upper panels of the figures show absolute values, whereas the lower panels show the differences relative to the base architecture. The base architecture has the lowest skill at all lead times and for all resolutions, skill metrics and variables considered here (we do not include here the two additional experiments described in section 2c), except for lead times of 6–8 days for RMSE of the low-resolution 850-hPa temperature. Sphereconv improves over the base architecture for all cases. This holds for all lead times, both resolutions and both for RMSE and ACC. In many cases, the improvement is, however, relatively small.
The hemconv architecture also improves systematically on the base architecture, with the exception of the aforementioned lead times of 6–8 days for RMSE of the low-resolution 850-hPa temperature. Depending on the variable, resolution and lead time, it can also perform better than the sphereconv architecture on the high-resolution data. Taking as reference RMSE at lead times of 3 or 5 days (Table 3), sphereconv systematically outperforms hemconv. Combining the two (sphereconv_hemconv), the forecast performance generally improves further over both sphereconv and hemconv taken individually. The hemispheric convolution architecture with shared (flipped) weights (hemconv_shared) generally outperforms the hemispheric convolution architecture with independent weights. The same holds when comparing SphereNet convolution combined with hemisphere-wise convolution with shared weights (sphereconv_hemconv_shared) to sphereconv_hemconv. While no single method is the best for all cases...
considered, in most cases combining SphereNet convolution with hemisphere-wise convolution and sharing the flipped weights leads to the best results. This is illustrated in Table 3, where the lowest RMSE for each variable, lead time, and resolution (not considering IFS) is highlighted in bold.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Geopotential at 500 hPa, day 3 (m² s⁻²)</th>
<th>Geopotential at 500 hPa, day 5 (m² s⁻²)</th>
<th>Temperature at 850 hPa, day 3 (K)</th>
<th>Temperature at 850 hPa, day 5 (K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>hres base</td>
<td>542 (7)</td>
<td>845 (40)</td>
<td>2.87 (0.09)</td>
<td>4.11 (0.16)</td>
</tr>
<tr>
<td>hres hemconv</td>
<td>519 (15)</td>
<td>785 (24)</td>
<td>2.73 (0.07)</td>
<td>3.83 (0.11)</td>
</tr>
<tr>
<td>hres hemconv_sharedweights</td>
<td>492 (14)</td>
<td>767 (26)</td>
<td>2.61 (0.02)</td>
<td>3.68 (0.10)</td>
</tr>
<tr>
<td>hres sphereconv</td>
<td>487 (6)</td>
<td>773 (8)</td>
<td>2.64 (0.07)</td>
<td>3.73 (0.13)</td>
</tr>
<tr>
<td>hres sphereconv_hemconv</td>
<td>455 (13)</td>
<td>736 (28)</td>
<td>2.57 (0.02)</td>
<td>3.68 (0.04)</td>
</tr>
<tr>
<td>hres sphereconv_hemconv_shared</td>
<td><strong>415 (6)</strong></td>
<td><strong>696 (11)</strong></td>
<td><strong>2.45 (0.05)</strong></td>
<td><strong>3.53 (0.08)</strong></td>
</tr>
<tr>
<td>lres base</td>
<td>514 (14)</td>
<td>788 (26)</td>
<td>2.75 (0.08)</td>
<td>3.81 (0.12)</td>
</tr>
<tr>
<td>lres base_latlon</td>
<td>520 (17)</td>
<td>806 (34)</td>
<td>2.73 (0.03)</td>
<td>3.82 (0.07)</td>
</tr>
<tr>
<td>lres hemconv</td>
<td>490 (4)</td>
<td>773 (6)</td>
<td>2.69 (0.07)</td>
<td>3.80 (0.10)</td>
</tr>
<tr>
<td>lres hemconv_halfsize</td>
<td>535 (12)</td>
<td>819 (27)</td>
<td>2.82 (0.11)</td>
<td>3.88 (0.16)</td>
</tr>
<tr>
<td>lres hemconv_sharedweights</td>
<td>458 (18)</td>
<td>737 (33)</td>
<td>2.55 (0.05)</td>
<td>3.63 (0.07)</td>
</tr>
<tr>
<td>lres sphereconv</td>
<td>480 (13)</td>
<td>762 (20)</td>
<td>2.62 (0.08)</td>
<td>3.70 (0.14)</td>
</tr>
<tr>
<td>lres sphereconv_hemconv</td>
<td>455 (6)</td>
<td>727 (12)</td>
<td>2.56 (0.02)</td>
<td>3.62 (0.02)</td>
</tr>
<tr>
<td>lres sphereconv_hemconv_shared</td>
<td><strong>416 (14)</strong></td>
<td><strong>699 (35)</strong></td>
<td><strong>2.50 (0.07)</strong></td>
<td><strong>3.61 (0.09)</strong></td>
</tr>
<tr>
<td>IFS T42</td>
<td>489</td>
<td>743</td>
<td>3.09</td>
<td>3.83</td>
</tr>
<tr>
<td>IFS T63</td>
<td>268</td>
<td>463</td>
<td>1.85</td>
<td>2.52</td>
</tr>
<tr>
<td>Operational IFS</td>
<td>154</td>
<td>334</td>
<td>1.36</td>
<td>2.03</td>
</tr>
</tbody>
</table>

We now turn to the spatial distribution of RMSE. We focus on z500 as this provides a more direct link to the atmospheric circulation than temperature (Fig. 4). Figure 4a shows the error of the base networks at different lead times for the high-resolution data, with increasing lead time from upper left to
lower right. The error pattern follows the typical error patterns of medium range NWP forecasts, with lowest predictability in the storm-track regions and in the Southern Hemisphere (e.g., Scher and Messori 2019a). As expected, the error grows with increasing lead time, with no dramatic changes in the spatial patterns.

More interesting is the difference between the spherconv and the base architecture (Fig. 4b) and between hemconv_sharedweights and the base architecture (Fig. 4c). Both clearly improve the forecasts in the high latitudes, and, in particular, in the Southern Hemisphere, with increasing improvement with increasing lead times. We finally consider the difference between sphereconv_hemconv_shared and hemconv_shared (Fig. 4d). Up to forecast day 5, the SphereNet convolution clearly improves the forecasts around both poles. For longer lead times, the SphereNet convolution still provides an improvement, but this now appears to be largest in the midlatitudes. Results are similar for \textit{t850} (Fig. A3), except that the forecast improvements in the midlatitudes for both sphereconv and hemconv_shared are larger relative to those in the poles than for \textit{z500}. Moreover, sphereconv_hemconv_shared presents a slightly lower skill in some mid-latitude regions compared to hemconv_shared.

Finally, we turn to the two additional experiments carried out on the low-resolution data: including latitude and longitude as additional constant channels (base_latlon), and hemispheric convolution without shared weights, but with half the number of weights per convolution (hemconv_halfsize), thus in total the same number of weights as the base architecture. Adding longitude and latitude information, surprisingly, deteriorates forecast performance for \textit{z500}, while it slightly increases forecast skill of \textit{t850} at intermediate lead times (second-to-last bars in Fig. 3). We, however, note that we have not tuned our model for these additional inputs, and we cannot exclude that the number of parameters and therefore the capacity of the network is not large enough to exploit this additional information. The results with the hemispheric convolution with same number of weights as the base architecture are also intriguing (last bars in Fig. 3). The forecast skill deteriorates compared to the base architecture, except for ACC at longer lead times, while for RMSE it deteriorates at all lead times. This result shows that the skill increase brought by the nonshared hemispheric convolution reported above actually seems to come from the higher number of parameters, and not from the separate convolutions. Conversely, the improved skill seen in the architecture with shared but flipped parameters clearly shows that the flipping provides an advantage.

\textbf{b. Analysis of events with largest forecast errors}

We now look at the forecasts within the upper 5\% of RMSE (forecast “busts”) for the Northern Hemisphere (NH) for a lead time of 3 days for the sphereconv_hemconv_shared architecture (the architecture that, as discussed in section 3a, generally displays the highest forecast skill). For each of the four training realizations, the percentile is computed individually. When comparing the initialization dates of the worst forecasts, \textasciitilde28\% are exactly the same dates for all training realizations, \textasciitilde46\% occur in at least three of the four members, and \textasciitilde66\% in at least two of the four members (Fig. 5a). This is much higher than expected by chance if the events were randomly distributed. Events with large errors are more common in boreal winter than in summer (Fig. 5b), in line...
with the performance of operational NWP models. Similarly, repeating the analysis on Southern Hemisphere data shows that events with large errors are more common in austral winter (Fig. A4).

Finally, Fig. 5c shows composite z500 anomalies at all initialization times for which at least one of the spheronconv_hemconv_shared high-resolution networks had an error $> 95\%$ in the NH. The anomaly is computed with respect to the mean over 2017/18 (the evaluation period), and separately for each month. There are positive anomalies east of Greenland, in central Russia, and in the middle of the North Pacific, and negative anomalies in northern Canada, the eastern coast of Asia, and central Europe. There is also an anomaly dipole between the west coast of North America and the eastern Pacific, at lower latitudes than the other anomaly centers. These collectively constitute a circumpolarwave-4 pattern. The results are qualitatively similar for other architectures (Figs. A5–A7), and for other lead times (Figs. A8–A11). This indicates that the skill of the network forecasts is dependent on the atmospheric configuration, just as in NWP forecasts (e.g., Ferranti et al. 2015; Matsueda and Palmer 2018).

c. Computational performance

Replacing standard convolution with SphereNet convolution introduces a significant amount of additional computations. While the use of sparse TensorFlow tensors (see section 2b) for the interpolation tensor $L$ avoids large memory requirements, the computation time of the spheronconv network for the high-resolution data compared to the base architecture is roughly a factor of 8 higher on a CPU with two cores (9.3 vs 1.1 s), and by a factor of 150 higher on an NVIDIA Tesla v100 graphical processing unit (GPU; 3.3 s vs 23 ms). The computational overhead might be reduced by optimizing the implementation for certain computational architectures, such as in Esteves et al. (2023), who optimized their code for tensor processing units (TPUs). Using hemispheric-wise convolution, on the other hand, does not introduce any significant performance overhead (with shared weights $\sim 4\%$; with separate weights none at all).

4. Discussion and conclusions

In this paper, we have tested two approaches to improve data-driven weather forecasts with CNNs. The aim was not to develop the best possible neural network–based weather forecasts (and indeed we note that there are publications presenting data-driven forecasts with a higher skill than ours), but to assess the effect of specific changes to a conventional neural network architecture on forecast skill. First, we have tested replacing standard convolution operations with a convolution operation that takes into account the (near-)spherical shape of Earth. Second, we tested integrating basic meteorological knowledge into the structure of the networks, namely, that the dynamics of the two hemispheres are different. This is done in two ways: in the first case, we hardcode into the network that the dynamics of one hemisphere are “flipped” with respect to the other hemisphere. This is implemented by flipping the weights of the network. In the second case, we use independent weights for each hemisphere, thus leaving the network free to learn potential differences in dynamics. These methods (and combinations of them) were tested on the ERA5 data from the WeatherBench dataset (Rasp et al. 2020). We used a neural network architecture previously proposed by Weyn et al. (2020) and adapted it to our convolution methods. We found that both the SphereNet convolution and the hemispheric information improve the forecasts, but in subtly different ways. Both SphereNet convolution and flipped hemispheric information lead to the largest improvements close to the poles, and less so in other regions. However, adding the SphereNet convolution to the flipped hemisphere-specific information leads to relatively uniform improvements in the midlatitudes when compared to hemispheric information alone. The midlatitudes are the regions where the largest forecast errors appear in the first place. For most lead times, skill metrics, resolutions and variables considered here, combining SphereNet convolution with hemisphere-specific information using shared flipped weights leads to the best forecasts. The experiments with hemispheric convolution without weight sharing show some interesting results. While hemispheric convolution without weight sharing showed a skill increase, the additional experiment with keeping the number of parameters the same as in the base architecture showed that this skill increase actually seems to come mainly from the additional parameters, and not the hemispheric convolution. This, however, clearly shows that the skill increase in the hemispheric convolution with shared convolution comes from the weight-sharing and flipping, showing the value of this physics-inspired approach.

Finally, we have found that initial conditions causing the largest forecast errors are relatively consistent across different training realizations of the same network, and across different network architectures and lead times. This indicates that, as for conventional NWP forecasts, forecast errors of the neural networks are at least partly flow-dependent. In other words, the networks struggle to make skillful forecasts when initialized from specific atmospheric states. Previous work has shown that quantifications of an “intrinsic” atmospheric predictability only partly match the empirical forecast errors from an NWP initialized from that atmospheric state (Scher and Messori 2018; Hochman et al. 2019, 2021, 2022). We do not investigate here whether the atmospheric configurations leading to high forecast errors for the CNNs used here match configurations with low “intrinsic” atmospheric predictability. Indeed, the error could also depend on the error in the reanalysis product used for training or on deficiencies in the training of the networks themselves.

Possible improvements to the methods presented here could be the following:

- To split up the convolution for smaller regions instead of at hemispheric scale (similar to Weyn et al. 2020), which could lead to the shared flipped weights no longer presenting a forecast skill advantage over independent convolutions.
- To include locally connected layers. Here each grid point in a layer is also a combination of the inputs from a certain kernel (e.g., $3 \times 3$), but the weights are not shared across the domain.
• To add more prior information into the structure of the network, for example on the vertical structure of the atmosphere.

In this study, we used an existing architecture, and changed the convolution types. Due to limited computational resources, we did not perform any additional hyperparameter tuning. An additional methodological improvement would therefore be to tune each individual architecture separately. Related to this, one could also compare scores for the training set both between architectures and to those on the test set, to verify whether any over or under fitting in occurring for the different models.

Our approach differs from previous studies in the field in several respects. The closest previous work is that by Weyn et al. (2020), who split up the world into six regions—two polar and four equatorial regions—with each region being represented by a local grid. On these local grids they used standard convolution operations. This method still leads to distortions, as even a subregion of Earth’s surface cannot be represented on a local regular grid with complete accuracy. In addition, this method also needs padding at the edge of each of the six regions, which introduces some ambiguity at the corners. Weyn et al. (2020) further use two sets of weights, one for all four equatorial regions and the other one for the two polar regions. The weights for the polar regions are “flipped” between the two poles. This is similar to our idea of flipped weights to provide hemispheric information to the CNN. There are also several differences between the approach we present here and that of Weyn et al. (2020) from a practical point of view. The method of Weyn et al. (2020) needs data-preprocessing (regridding) but can then use standard neural network operations. In our approach, the standard data can be used, but the SphereNet convolution introduces a computational overhead in every pass through the network.

The percent increase in runtime needed for the networks with SphereNet convolution is higher on a GPU than on CPU, which could indicate that the implementation using sparse TensorFlow tensors—which we adopt here—is not optimized for GPUs. For small input sizes, an alternative would be to use standard TensorFlow tensors (still filled sparsely but represented as a full tensor (array) in memory). However, for the full-resolution ERA5 data (0.25° resolution, namely, 3600 × 1801 grid points on a regular latitude–longitude points), this would not be feasible with current computers due to memory limitations. Indeed, the interpolation tensor would then have a size of 6 483 600 × 6 483 600.

The methods used and presented in this study all generate single deterministic forecasts. In many weather forecasting settings, however, probabilistic forecasts are wanted. The typical way to do this with numerical weather prediction models is to use ensemble forecasting, which entails performing multiple model runs with slightly different initial conditions, slightly different model formulations, stochastic components, or a combination of these (e.g., Leutbecher and Palmer 2008). While dedicated probabilistic ML techniques exist, such as Gaussian process regression (Ebden 2015) or probabilistic neural networks (Mohebali et al. 2020), the at least conceptually simpler approach is to apply the concept of ensemble NWP to ML models as well. Scher and Messori (2020) compared three methods of generating ensembles from deterministic neural network prediction systems. The first one—using multiple models, each trained with different initial seeds—has a computational cost that scales linearly with ensemble size both in the training stage and the prediction stages. Using random initial perturbations requires no additional training, and scales linearly in the prediction phase with no significant additional overhead, as generating the random perturbations is computationally cheap. The final and most advanced method uses singular value decomposition to find optimal initial perturbations. This introduces a one-time overhead per prediction for generating the singular vectors for the initial perturbations, and apart from this the computational cost again scales linearly with ensemble size. In principle, the improvements to conventional CNNs proposed here would be combined with any of the above approaches, although the one requiring training the model multiple times may become computationally demanding given the additional overhead from the SphereNet convolution.

To conclude, in this study we have tested some simple improvements to conventional convolutional neural network architectures for weather forecasting. These are based on a convolution operator accounting for Earth’s (near-)spherical geometry and on providing the network with knowledge that the climate dynamics of the two hemispheres differ. We did not seek to outperform state-of-the-art data-driven weather forecasts in terms of forecast skill but instead sought to test the limits of CNNs with a relatively small number of parameters. This can be beneficial for research groups who may not have enough computational capacity to run very large transformer or GNN models. The improvements presented here stem from physical reasoning and may support future development of physics-oriented ML weather prediction models.

Acknowledgments. S.S. was funded by the Dept. of Meteorology of Stockholm University. G.M. was supported by the European Union’s H2020 research and innovation programme under ERC Grant 948309 (CEN-Eproject) and by the Swedish Research Council Vetenskapsrådet (Grant 2016–03724). This work was partly supported by the “DDAI” COMET Module within the Competence Centers for Excellent Technologies (COMET) Programme, funded by the Austrian Federal Ministry (BMK and BMDW), the Austrian Research Promotion Agency (FFG), the province of Styria (SFG) and partners from industry and academia. The COMET Programme is managed by FFG. The computations and data handling were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) and the Swedish National Infrastructure for Computing (SNIC) at the High Performance Computing Center North (HPC2N) and National Supercomputer Centre (NSC), partially funded by the Swedish Research Council through Grant Agreements 2022-06725 and 2018-05973.
Data availability statement. The software developed for this study and the intermediate data underlying the plots are available on the first author’s GitHub repository (https://github.com/sipposip/physics-informed-ML-NWP) as well as on Zenodo (https://zenodo.org/record/8344872 with https://doi.org/10.5281/zenodo.8344872). Additionally, the trained models are also stored in the Zenodo archive as well. The ERA5 WeatherBench data used as input data can be freely obtained via WeatherBench (https://github.com/pangeo-data/WeatherBench) and, alternatively, as raw data from the Copernicus data store at https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-single-levels?tab=overview.

APPENDIX

Additional Analyses

Figures A1 and A2 show the data from Figs. 2 and 3 of the main text but in a different visualization format. Figures A3–A11 show analysis for additional variables and for different model architectures.

**Fig. A1.** As in Fig. 2, but as line plots.
Fig. A2. As in Fig. 3, but as line plots.
Fig. A3. As in Fig. 4, but for t850 (K).

Fig. A4. As in Fig. 5, but for the Southern Hemisphere.
**FIG. A5.** As in Fig. 5, but for the base architecture.

**FIG. A6.** As in Fig. 5, but for the hemconv_shared architecture.
**Fig. A7.** As in Fig. 5, but for the sphereconv architecture.

**Fig. A8.** As in Fig. 5, but for lead time of 1 day.
Fig. A9. As in Fig. 5, but for lead time of 5 days.

Fig. A10. As in Fig. 5, but for lead time of 7 days.
REFERENCES


Fig. A11. As in Fig. 5, but for lead time of 9 days.


