# Supplementary Information

## Neural Networks

In supervised learning, a neural network predicts the output data y from an input data x using $y = f(x, \theta)$. In this model, $\theta$ are the parameters of the model, and f is a large and layer-structured model. A basic layer is a non-linear function that follows a linear function. This non-linear function (known as activation function) enables the modelling of complex relations in the network. In addition, to provide suitability for computer hardware and feasibility of fast evaluation of the networks (regardless of their size), linear functions and layer structure are employed. For instance, consider the following example of a layer $l$ with an input of $a$, and an output of $c$ (1):

$$c_j = g(\sum_i w_{i,j} a_i + b_j) \tag{1}$$

In this equation, $b_j$ and $w_{i,j}$ are parameters of the model, $i, j$ are indices employed to distinguish individual nodes in layer $l$ and layer $l+1$, respectively, and g represents the activation function. Moreover, $c_j s$ are outputs (*i.e.*, activations of the node $j$) and are later employed in a similar manner to calculate the activations of the layer $l+2$. The activation $c_j$ can be assumed an unnormalized probability for a feature specific to the node $j$. In a layer, nodes are generally arranged in a 3D tensor. However, if the input is inherently 2D (*e.g.*, an image), the following layers will have two dimensions in accordance with the image dimensions. In this paper, they will be referred to as spatial or image dimensions, while the remaining layers will be referred to as channel dimensions.

$b_j$ and $w_{i,j}$ are learned through the evaluation of the model on the test data set $(\bar{y}, \bar{x})$ and through the minimization of the error $l(\bar{y}, f(\bar{x}, \theta), \theta)$ for $\theta$. In this error, $l$ is a loss function that describes the distance between $\bar{y}$ and $f(\bar{x}, \theta)$ on $\theta$ similar to prior instances. In almost all cases, an Stochastic Gradient Descent (SGD) based optimizer is employed. Often, a layer's input can be assumed shift-invariant (*e.g.*, for image inputs). In such cases, the utilisation of convolutional layers seems logical, where the weights are shared so that if the relationship between $a_i$ and $c_j$ in the spatial dimension is similar to $a_l$ and $c_k$, then $w_{i,j} = w_{l,k}$ and $b_{i,j} = b_{l,k}$. In addition, if $a_i$ is spatially far from $c_j$, then it can be assumed that $w_{i,j} = 0$, which will limit the field of view for $c_j$.

## 2D and 3D Convolutional Neural Networks

The function of a convolutional neural network (CNN) is to learn the mapping between input data and output data[1]. A 2D-CNN processes input images by first applying a sequence of matrix multiplications called kernel filters with a sample size $(P_i, Q_i)$ in the $i^{th}$ layer and then summing the results. Identifying the most important features in that image is the objective[2]. The output $V_{ij}^{ab}$ centred at (a,b) for the $j^{th}$ feature map and the $i^{th}$ layer can be stated in (2).

$$V_{ij}^{ab} = tanh\left(\sum_m \sum_{x=0}^{P_i-1} \sum_{y=0}^{Q_i-1} W_{ijm}^{xy} V_{(i-1)m}^{(a+x)(b+y)} + b_{ij}\right) \tag{2}$$

The non-linearity operation is tanh used on the kernel output and the biases is $b_{ij}$. $W_{ijm}^{x\,y}$ is the output of kernels centered at $(x, y)$ for the $k^{th}$ feature map. Also, $m$ is the indexing parameter over a feature maps set connected to the current feature map in the $(i-1)^{th}$ layer. The output of a convolutional layer is typically a feature map that is reflective of the learned features from the input image. Therefore, when $R_i$ represents the third dimension of the kernels, the output centered at $(a, b, c)$ for the $j^{th}$ feature map and the $i^{th}$ layer is formulated in 3D-CNN (3). The kernel output is $W_{ijm}^{x\,yz}$ centred at $(x, y, z)$ for the $k^{th}$ feature map. Also, $m$ represents feature maps indexes.

$$V_{ij}^{abc} = tanh\left(\sum_m \sum_{x=0}^{P_i-1} \sum_{y=0}^{Q_i-1} \sum_{z=0}^{R_i-1} W_{ijm}^{xyz}\ V_{(i-1)m}^{(a+x)(b+y)(c+z)} + b_{ij}\right) \tag{3}$$

## Capsule Neural Networks

It has been suggested that the novel artificial neural network known as CapsNet can approach the biological representation of the human brain in terms of its hierarchical connections in a more precise way[3]. Modules, represented as capsules in the brain, are highly proficient at controlling different visual stimuli and encoding information[4]. Pooling, which is usually employed for dimension reduction in CNN networks, does not exhibit this characteristic. Each capsule in a CapsNet layer forward propagates

| | | CapsNet | Traditional neural networks |
|---|---|---|---|
| Input from previous capsule/neuron | | Vector ($u_i$) | Scalar ($x_i$) |
| Operation | Affine Transform | $\hat{u}_{j\|i} = W_{ij}u_i$ | - |
| | Weighting<br>Sum | $s_j = \sum_t c_{ij}\,\hat{u}_{j\|i}$ | $a_j = \sum_i w_i\,x_i + b$ |
| | Nonlinear activation | $v_j = \frac{\|s_j\|^2}{1+\|s_j\|^2}\frac{s_j}{\|s_j\|}$ | $h_j = f(a_j)$ |
| Output | | Vector ($v_j$) | Scalar ($h_j$) |

**Table 1.** A comparison between traditional neural networks and CapsNet

data to the capsule in the layer above it, resulting in a hierarchical structure. Table 3 displays a comparison between CapsNets and neural networks.

Traditional neural networks have an output equal to the sum of the weighted neurons $a_j$ computed in the last layer. The output $h_j$, which is scalar, is calculated with a nonlinear activation function. However, in CapsNet, instead of treating each neuron individually, a capsule treats the entire group of neurons as a whole (vector). To do so, it uses vector capsules instead of scalar output feature indicators (activation vectors $u_i$). The activation vectors with weighted coefficients are multiplied by a matrix $W_{ij}$. Then, the vectors $s_j$ in the preceding layer are weighted. Finally, the vector is scaled between zero and unit length using summing and a squashing function ($v_j$).

Figure 1 displays the layers that comprise the original CapsNet model, which are the convolution layers, the primary capsule layer, the DigitCaps, and the fully connected layer. The model receives a handwritten image of digits. Conv1 and Conv2 are two Convolution layers with the same kernels and different strides. In these Convolution layers, the Rectified Linear Unit (ReLU) is used as the nonlinearity. Therefore, the feature maps that are generated by Conv1 and Conv2 are different. By modifying the feature maps, the primary capsule layer is created. It functions as the Capsule's input layer, and its purpose is to construct the corresponding vector structure. The output of this layer's reshaping is used as a vector input in the subsequent layer. DigitCaps are then used to determine the loss function for the classification objective (encoder component), while the fully connected layers used for image reconstruction (decoder part) serve as network regularisations that avoid overfitting. Between primary capsules and DigitCaps, the DR method is used to update the necessary calculations and parameters between full connections. As for updating the parameters, CapsNets use both DR and the traditional back-propagation technique[3].

## References


1. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2323, DOI: 10.1109/5.726791 (1998).
2. Ji, S., Xu, W., Yang, M. & Yu, K. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis Mach. Intell.* **35**, 221–231, DOI: 10.1109/TPAMI.2012.59 (2013).
3. Sabour, S., Frosst, N. & Hinton, G. E. Dynamic routing between capsules. vol. 2017-Decem, 3857–3867, DOI: 10.48550/arxiv.1710.09829 (2017).
4. Hinton, G. E., Krizhevsky, A. & Wang, S. D. Transforming auto-encoders. *Lect. Notes Comput. Sci.* **6791 LNCS**, 44–51, DOI: 10.1007/978-3-642-21735-7_6 (2011).
5. Khodadadzadeh, M., Ding, X., Chaurasia, P. & Coyle, D. A hybrid capsule network for hyperspectral image classification. *IEEE J. Sel. Top. Appl. Earth Obs. Remote. Sens.* **14**, 11824–11839, DOI: 10.1109/JSTARS.2021.3126427 (2021).

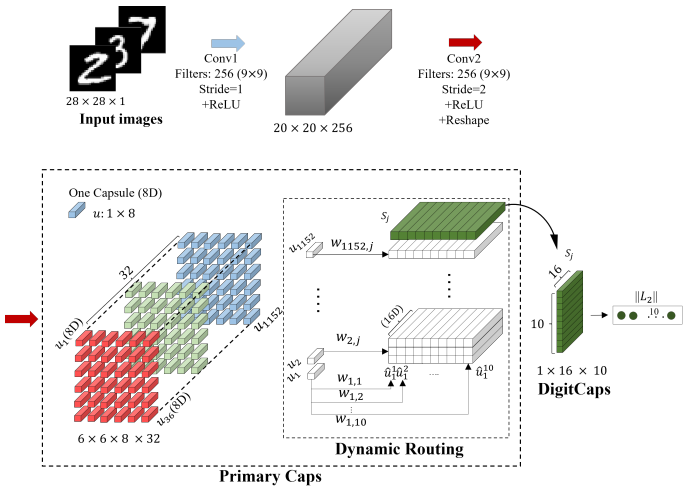


**Figure 1.** Structure of the CapsNet[5]. As an input image, the model receives a handwritten digit from the MNIST database of handwritten digits [67]. For 10 distinct classes, the system learns to encode images of size 28x28x1 into a 16D vector of instantiation parameters (DigitCaps). To accomplish the extraction of local features, the convolution layers are traditional convolution layers with a rectified linear unit (ReLU) activation function. There are two layers of convolution first: conv1 uses a 9x9 convolutional kernel and ReLU with a stride of 1, second: conv2 uses a 9x9 convolutional kernel and ReLU with a stride of 2. The first conversion yields 256 20x20 feature maps, whereas the second yields 256 scalar-filled 6x6 $(32x8x6x6)$ feature maps. This result is converted to produce 32 6$x$6 maps with 8D vectors. 1152 prediction × 10 classes equals 11520 weighted matrices $W_{ij}$.

**Algorithm 1:** Training and Backpropagation

**for** numbers of epochs
  **for** iterations (batches)
    **for** numbers of routing (r)

$$b_{i,j}(initialized)^{(r=1)} = 0$$
$$b_{i,j} \rightarrow c_{i,j} \quad \text{; Coupling coefficient (Softmax)}$$
$$S_j{}^{(r)} = \sum_i c_{i,j} \,.\, \hat{u}_i^{j\,(r)} \quad \text{; Weighted sum}$$
$$V_j{}^{(r)} = Squash\left(S_j{}^{(r)}\right) \quad \text{; Product vector}$$
$$= Squash\left(\sum_i c_{i,j} \,.\, \hat{u}_i^{j\,(r)}\right)$$

Dot Product: $\hat{u}_i^j \,.\, V_j{}^{(r)}$

$$b_{i,j}{}^{Updated} \leftarrow b_{i,j} + \hat{u}_i^j \,.\, V_j{}^{(r)}$$
$$\leftarrow b_{i,j} + \hat{u}_i^{j\,T} \,.\, V_j{}^{(r)}$$
$$\leftarrow b_{i,j} + \hat{u}_i^{j\,T} \,.\, Squash\left(\sum_i c_{i,j} \,.\, \hat{u}_i^{j\,(r)}\right)$$

**return** $V_j$ *(end for routings)*

The loss function for the correct category:
$L_J = k.max(0, 0.9 - \|V_j\|)$

Backpropagation starts:
Weighted matrices: $w_{i,j}{}^{updated} \leftarrow w_{i,j}$
Convolution layers: $filters^{updated} \leftarrow filters$
**return** (end for iterations)
**return** (end for epochs)

**Figure 2.** Routing by Agreement Algorithm[3].