Introduction to deep learning

Lihi Shiloh-Perl and Raja Giryes

Abstract Deep Learning (DL) has made a major impact on data science in the last decade. This chapter introduces the basic concepts of this field. It includes both the basic structures used to design deep neural networks and a brief survey of some of its popular use cases.

1 General overview

Neural Networks (NN) have revolutionized the modern day-to-day life. Their significant impact is present even in our most basic actions, such as ordering products on-line via Amazon's Alexa or passing the time with on-line video games against computer agents. The NN effect is evident in many more occasions, for example, in medical imaging NNs are utilized for lesion detection and segmentation [40, 5], and tasks such as text-to-speech [38, 120] and text-to-image [101] have remarkable improvements thanks to this technology. In addition, the advancements they have caused in fields such as natural language processing (NLP) [24, 144, 77], optics [114, 42], image processing [110, 143] and computer vision (CV) [10, 34] are astonishing, creating a leap forward in technology such as autonomous driving [13, 79], face recognition [109, 134, 23], anomaly detection [64], text understanding [54] and art [35, 53], to name a few. Its influence is powerful and is continuing to grow.

The NN journey began in the mid 1960's with the publication of the Perceptron [105]. Its development was motivated by the formulation of the human neuron activity [80] and research regarding the human visual perception [49]. However, quite quickly, a deceleration in the field was experienced, which lasted for almost three decades. This was mainly the result of lack of theory with respect to the possibility of training the (single-layer) perceptron and a series of theoretical results

School of Electrical Engineering, Tel Aviv University, e-mail: {lihishiloh@mail ,raja@tauex}.tau.ac.il

that emphasized its limitations, where the most remarkable one is its inability to learn the XOR function [82].

This *NN ice age* came to a halt in the mid 1980's, mainly with the introduction of the multi-layer perceptron (MLP) and the backpropagation algorithm [107]. Furthermore, the revolutionary convolutional layer was presented [68], where one of its notable achievements was successfully recognizing hand-written digits [67].

While some other significant developments have happened in the following decade, such as the development of the long-short memory machine (LSTM) [46], the field experienced another deceleration. Questions were arising with no adequate answers especially with respect to the non-convex nature of the used optimization objectives, overfitting the training data, and the challenge of vanishing gradients. These difficulties led to a second *NN winter*, which lasted two decades. In the meantime, classical machine learning techniques were developed and attracted much academic and industry attention. One of the prominent algorithms was the newly proposed Support Vector Machine (SVM) [17], which defined a convex optimization problem with a clear mathematical interpretation [16]. These properties increased its popularity and usage in various applications.

The 21st century began with some advancements in neural networks in the areas of speech processing and Natural Language Processing (NLP). Hinton *et al.* [45] proposed a method for layer-wise initial training of neural networks, which leveraged some of the challenges in training networks with several layers. However, the great NN *tsunami* truly hit the field with the publication of *AlexNet* in 2012 [62]. In this paper, Krizhevsky *et al.* presented a neural network that achieved state-of-the-art performance on the ImageNet [22] challenge, where the goal is to classify images into 1000 categories using 1.2 Million images for training and 150000 images for testing. The improvement over the runner-up, which relied on hand crafted features and one of the best classification techniques of that time, was notable - more than 10%. This caused the whole research community to understand that neural networks are way more powerful than what was thought and they bear a great potential for many applications. This led to a myriad of research works that applied NNs for various fields showing their great advantage.

Nowadays, it is safe to say that almost every research field has been affected by this NN *tsunami* wave, experiencing significant improvements in abilities and performance. Many of the tools used today are very similar to the ones used in the previous phase of NN. Indeed, some new regularization techniques such as batch-normalization [50] and dropout [121] have been proposed. Yet, the key-enablers for the current success is the large amounts of data available today that are essential for large NN training, and the developments in GPU computations that accelerate the training time significantly (sometimes even leading to ×100 speed-up compared to training on a conventional CPU). The advantages of NN is remarkable especially at large scales. Thus, having large amounts of data and the appropriate hardware to process them, is vital for their success.

A major example of a tool that did not exist before is the Generative Adversarial Network (GAN [39]). In 2014, Goodfellow *et al.* published this novel framework for learning data distribution. The framework is composed of two models, a generator

and a discriminator, trained as adversaries. The generator is trained to capture the data distribution, while the discriminator is trained to differentiate between generated ("fake") data and real data. The goal is to let the generator synthesize data, which the discriminator fails to discriminate from the real one. The GAN architecture is used in more and more applications since its introduction in 2014. One such application is the rendering of real scene images were GANs have proved very successful [36, 151]. For example, Brock *et al.* introduced the BigGAN [7] architecture that exhibited impressive results in creating high-resolution images, shown in Fig. 1. While most GAN techniques learn from a set of images, recently it has been successfully demonstrated that one may even train a GAN just using one image [112]. Other GAN application include inpainting [73, 145], retargeting [115], 3D modeling [1], semi-supervised learning [31], domain adaptation [47] and more.

Fig. 1: Class-conditional samples generated by a GAN, [7].

While neural networks are very successful, the theoretical understanding behind them is still missing. In this respect, there are research efforts that try to provide a mathematical formulation that explains various aspects of NN. For example, they study NN properties such as their optimization [124], generalization [52] and expressive power [108, 88].

The rest of the chapter is organized as follows. In Section 2 the basic structure of a NN is described, followed by details regarding popular loss functions and metric learning techniques used today (Section 3). We continue with an introduction to the NN training process in Section 4, including a mathematical derivation of backpropagation and training considerations. Section 5 elaborates on the different optimizers used during training, after which Section 6 presents a review of common regularization schemes. Section 7 details advanced NN architecture with state-of-the-art performances and Section 8 concludes the chapter by highlighting some current important NN challenges.

2 Basic NN structure

The basic building block of a NN consists of a linear operation followed by a non-linear function. Each building block consists of a set of parameters, termed weights and biases (sometimes the term weights includes also the biases), that are updated in the training process with the goal of minimizing a pre-defined loss function.

Assume an input data $\mathbf{x} \in \mathbb{R}^{d_0}$, the output of the building block is of the form $\psi(\mathbf{Wx} + \mathbf{b})$, where $\psi(\cdot)$ is a non-linear function, $\mathbf{W} \in \mathbb{R}^{d_1 \times d_0}$ is the linear operation and $\mathbf{b} \in \mathbb{R}^{d_1}$ is the bias. See Fig. 2 for an illustration of a single building block.
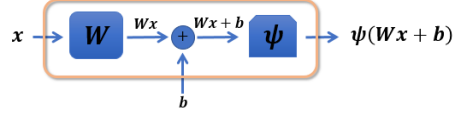

Fig. 2: NN building block consists of a linear and a non-linear elements. The weights $\mathbf{W}$ and biases $\mathbf{b}$ are the parameters of the layer.
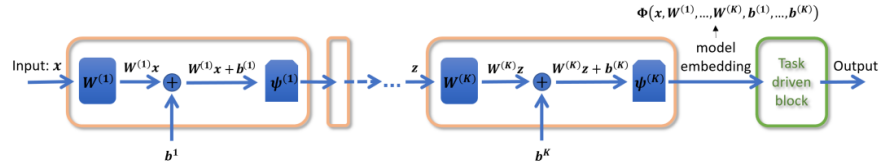

Fig. 3: NN layered structure: concatenation of N building blocks, e.g., model layers.

To form an NN model, such building blocks are concatenated one to another in a layered structure that allows the input data to be gradually processed as it propagates through the network. Such a process is termed the (feed-)forward pass. Following it, during training, a backpropagation process is used to update the NN parameters, as elaborated in Section 4.1. In inference time, only the forward pass is used.

Fig. 3 illustrates the concatenation of K building blocks, e.g., layers. The intermediate output at the end of the model (before the "task driven block") is termed the *network embedding* and it is formulated as follows:

$$\Phi(\mathbf{x}, \mathbf{W}^{(1)}, ..., \mathbf{W}^{(K)}, \mathbf{b}^{(1)}, ..., \mathbf{b}^{(K)}) = \psi(\mathbf{W}^{(K)}...\psi(\mathbf{W}^{(2)}\psi(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)})... + \mathbf{b}^{(K)}). \quad (1)$$

The final output (prediction) of the network is estimated from the network embedding of the input data using an additional task driven layer. A popular example is the case of classifications, where this block is usually a linear operation followed by the *cross-entropy* loss function (detailed in Section 3).

When approaching the analysis of data with varying length, such as sequential data, a variant of the aforementioned approach is used. A very popular example for such a neural network structure is the Recurrent Neural Network (RNN [51]). In a vanilla RNN model, the network receives at each time step just a single input but with a feedback loop calculated using the result of the same network in the previous time-step (see an illustration in Fig. 4). This enables the network to "remember" information and support multiple inputs and producing one or more outputs.

More complex RNN structures include performing bi-directional calculations or adding gating to the feedback and the input received by the network. The most known complex RNN architecture is the Long-Term-Short-Memory (LSTM) [46, 37], which adds gates to the RNN. These gates decide what information from the current input and the past will be used to calculate the output and the next feedback, as well as what information to mask (i.e., causing the network to forget). This enables an easier combination of past and present information. It is commonly used for time-series data in domains such as NLP and speech processing.
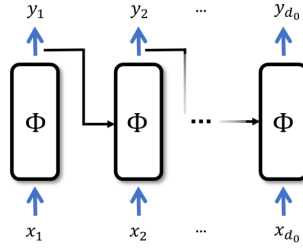

Fig. 4: Recurrent NN (RNN) illustration for time series data. The feedback loop introduces time dependent characteristics to the NN model using an element-wise function. The weights are the same along all time steps.

Another common network structure is the *Encoder-Decoder* architecture. The first part of the model, the encoder, reduces the dimensions of the input to a compact feature vector. This vector functions as the input to the second part of the model, the decoder. The decoder increases its dimension, usually, back to the original input size. This architecture essentially learns to compress (encode) the input to an efficiently small vector and then decode the information from its compact representation. In the context of regular feedforward NN, this model is known as autoencoder [119] and is used for several tasks such as image denoising [102], image captioning [133], feature extraction [132] and segmentation [2]. In the context of sequential data, it is used for tasks such as translation, where the decoder generates a translated sentence from a vector representing the input sentence [126, 14].

2.1 Common linear layers

A common basic NN building block is the Fully Connected (FC) layer. A network composed of a concatenation of such layers is termed Multi-Layer Perceptron (MLP) [106]. The FC layer connects every neuron in one layer to every neuron in the following layer, i.e. the matrix $\mathbf{W}$ is dense. It enables information propagation from all neurons to all the ones following them. However it may not maintain spatial information. Figure 5 illustrates a network with FC layers.

The convolutional layer [66, 68] is another very common layer. We discuss here the 2D case, where the extension to other dimension is straight-forward. This layer applies one or multiple convolution filters to its input with kernels of size $W \times H$. The output of the convolution layer is commonly termed a *feature map*.

Each neuron in a feature map receives inputs from a set of neurons from the previous layer, located in a small neighborhood defined by the kernel size. If we apply this relationship recursively, we can find the part of the input that affects each neuron at a given layer, i.e., the area of visible context that each neuron sees from the input. The size of this part is called the *receptive field*. It impacts the type and size of visual features each convolution layer may extract, such as edges, corners and even patterns. Since convolution operations maintain spatial information and are translation equivariant, they are very useful, namely, in image processing and CV.

If the input to a convolution layer has some arbitrary third dimension, for example 3-channels in an RGB image ($C = 3$) or some $C > 1$ channels from an output of a hidden layer in the model, the kernel of the matching convolution layer should be of size $W \times H \times C$. This corresponds to applying a different convolution for each input channel separately, and then summing the outputs to create one feature map. The convolution layer may create a multi-channel feature map by applying multiple filters to the input, i.e., using a kernel of size $W \times H \times C_{in} \times C_{out}$, where C_{in} and C_{out} are the number of channels at the input and output of the layer respectively.

2.2 Common non-linear functions

The non-linear functions defined for each layer are of great interest since they introduce the non-linear property to the model and can limit the propagating gradient from vanishing or exploding (see Section 4).

Non-linear functions that are applied element-wise are known as *activation functions*. Common activation functions are the Rectified Linear Unit (ReLU [20]), leaky ReLU [141], Exponential Linear Unit (ELU) [15], hyperbolic tangent (tanh) and sigmoid. There is no universal rule for choosing a specific activation function, however,
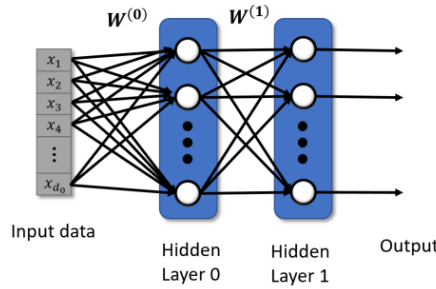

Fig. 5: Fully-connected layers.

ReLUs and ELUs are currently more popular for image processing and CV while sigmoid and tanh are more common in speech and NLP. Fig. 6 presents the response of the different activation functions and Table 1 their mathematical formulation.
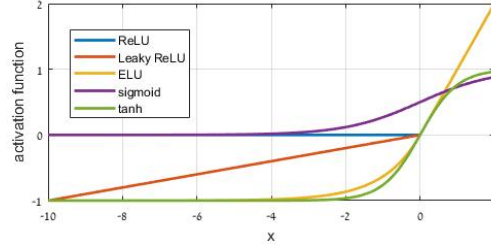

Fig. 6: Different activation functions. Leaky ReLU with $\alpha = 0.1$, ELU with $\alpha = 1$.

Table 1: Mathematical expressions for non-linear activation functions.

Function	Formulation $s(x)$	Derivative $\frac{ds(x)}{dx}$	Function output range
ReLU	$\begin{cases} 0, & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases}$	$\begin{cases} 0, & \text{for } x < 0 \\ 1, & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$
Leaky ReLU	$\begin{cases} \alpha x, & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases}$	$\begin{cases} \alpha, & \text{for } x < 0 \\ 1, & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
ELU	$\begin{cases} \alpha(e^x - 1), & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases}$	$\begin{cases} \alpha e^x, & \text{for } x < 0 \\ 1, & \text{for } x \geq 0 \end{cases}$	$[-\alpha, \infty)$
Sigmoid	$\frac{1}{1+e^{-x}}$	$\frac{e^{-x}}{(1+e^{-x})^2}$	$(0, 1)$
tanh	$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$	$1 - \tanh^2(x)$	$(-1, 1)$

Another common non-linear operations in a NN model are the *pooling* functions. They are aggregation operations that reduce dimensionality while keeping dominant features. Assume a pooling size of q and an input vector to a hidden layer of size d, $\mathbf{z} = [z_1, z_1, ..., z_d]$. For every $m \in [1, d]$, the subset of the input vector $\tilde{\mathbf{z}} = [z_m, z_{m+1}, ..., z_{q+m}]$ may undergo one of the following popular pooling operations:

1. Max pooling: $g(\tilde{\mathbf{z}}) = \max_i \tilde{\mathbf{z}}$
2. Mean pooling: $g(\tilde{\mathbf{z}}) = \frac{1}{q} \sum_{i=m}^{q+m} z_i$
3. ℓ_p pooling: $g(\tilde{\mathbf{z}}) = \sqrt[p]{\sum_{i=m}^{q+m} z_i^p}$

All pooling operations are characterized by a stride, s, that effectively defines the output dimensions. Applying pooling with a stride s, is equivalent to applying the

pooling with no stride (i.e., $s = 1$) and then sub-sampling by a factor of s. It is common to add zero padding to $\mathbf{z}$ such that its length is divisible by s.

Another very common non-linear function is the *softmax*, which normalizes vectors into probabilities. The output of the model, the embedding, may undergo an additional linear layer to transform it to a vector of size $1 \times N$, termed *logits*, where N is the number of classes. The logits, here denoted as $\mathbf{v}$, are the input to the softmax operation defined as follows:

$$\text{softmax}(v_i) = \frac{e^{v_i}}{\sum_{j=1}^{N} e^{v_j}}, \quad i \in [1, ..., N]. \tag{2}$$

3 Loss functions

Defining the loss function of the model, denoted as $\mathcal{L}$, is critical and usually chosen based on the characteristics of the dataset and the task at hand. Though datasets can vary, tasks performed by NN models can be divided into two coarse groups: (1) regression tasks and (2) classification tasks.

A *regression* problem aims at approximating a mapping function from input variables to a continuous output variable(s). For NN tasks, the output of the network should predict a continues value of interest. Common NN regression problems include image denoising [148], deblurring [84], inpainting [142] and more. In these tasks, it is common to use the Mean Squared Error (MSE), Structural SIMilarity (SSIM) or ℓ_1 loss as the loss function. The MSE (ℓ_2 error) imposes a larger penalty for larger errors, compared to the ℓ_1 error which is more robust to outliers in the data. The SSIM, and its multiscale version [149], help improving the perceptual quality.

In the *classification* task, the goal is to identify the correct class of a given sample from pre-defined N classes. A common loss function for such tasks is the *cross-entropy* loss. It is implemented based on a normalized vector of probabilities corresponding to a list of potential outcomes. This normalized vector is calculated by the softmax non-linear function (Eq. (2)). The cross-entropy loss is defined as:

$$\mathcal{L}_{CE} = -\sum_{i=1}^{N} y_i \log(p_i), \tag{3}$$

where y_i is the ground-truth probability (the label) of the input to belong to class i and p_i is the model prediction score for this class. The label is usually binary, i.e., it contains 1 in a single index (corresponding to the true class). This type of representation is known as *one-hot encoding*. The class is predicted in the network by selecting the largest probability and the log-loss is used to increase this probability.

Notice that a network may provide multiple outputs per input data-point. For example, in the problem of image semantic segmentation, the network predicts a class for each pixel in the image. In the task of object detection, the network outputs a list of objects, where each is defined by a bounding box (found using a

regression loss) and a class (found using a classification loss). Section 7.1 details these different tasks. Since in some problems, the labelled data are imbalanced, one may use weighted softmax (that weigh less frequent classes) or the focal loss [72].

3.1 Metric Learning

An interesting property of the log-loss function used for classification is that it implicitly cluster classes in the network embedding space during training. However, for a clustering task, these vanilla distance criteria often produce unsatisfactory performance as different class clusters can be positioned closely in the embedding space and may cause miss-classification for samples that do not reside in the specific training set distribution.

Therefore, different metric learning techniques have been developed to produce an embedding space that brings closer intra-class samples and increases inter-class distances. This results in better accuracy and robustness of the network. It allows the network to be able to distinguish between two data samples if they are from the same class or not, just by comparing their embeddings, even if their classes have not been present at training time.

Metric learning is very useful for tasks such as face recognition and identification, where the number of subjects to be tested are not known at training time and new identities that were not present during training should also be identified/recognized (e.g., given two images the network should decide whether these correspond to the same or different persons).

An example for a popular metric loss is the *triplet loss* [109]. It enforces a margin between instances of the same class and other classes in the embedding feature space. This approach increases performance accuracy and robustness due to the large separation between class clusters in the embedding space. The triplet loss can be used in various tasks, namely detection, classification, recognition and other tasks of unknown number of classes.

In this approach, three instances are used in each training step i: an anchor $\mathbf{x}_i^a$, another instance $\mathbf{x}_i^p$ from the same class of the anchor (positive sample), and a sample $\mathbf{x}_i^n$ from a different class (negative class). They are required to obey the following inequality:

$$\left\|\Phi(\mathbf{x}_i^a) - \Phi(\mathbf{x}_i^p)\right\|_2^2 + \alpha < \left\|\Phi(\mathbf{x}_i^a) - \Phi(\mathbf{x}_i^n)\right\|_2^2, \tag{4}$$

where $\alpha < 0$ enforces the wanted margin from other classes. Thus, the triplet loss is defined by:

$$\mathcal{L} = \sum_i \left\|\Phi(\mathbf{x}_i^a) - \Phi(\mathbf{x}_i^p)\right\|_2^2 - \left\|\Phi(\mathbf{x}_i^a) - \Phi(\mathbf{x}_i^n)\right\|_2^2 + \alpha. \tag{5}$$

Fig. 7 presents a schematic illustration of the triplet loss influence on samples in the embedding space. This illustration also exhibits a specific triplet example, where the positive examples are relatively far from the anchor while negative examples are

relatively near the anchor. Finding such examples that violate the triplet condition is desirable during training. They may be found by on-line or off-line searches known as *hard negative mining*. A preprocessing of the instances in the embedding space is performed to find violating examples for training the network.

Finding the "best" instances for training can, evidently, aid in achieving improved convergence. However, searching for them is often time consuming and therefore alternative techniques are being explored.
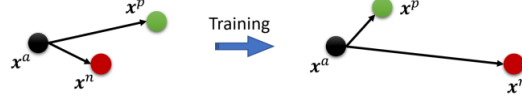

Fig. 7: Triplet loss: minimizes the distance between two similar class examples (anchor and positive), and maximizes the distance between two different class examples (anchor and negative).

An intriguing metric learning approach relies on 'classification'-type loss functions, where the network is trained given a fixed number of classes. Yet, these losses are designed to create good embedding space that creates margin between classes, which in turn provides good prediction of similarity between two inputs. Popular examples include the Cos-loss [134], Arc-loss [23] and SphereFace [76].

4 Neural network training

Given a loss function, the weights of the neural network are updated to minimize it for a given training set. The training process of a neural network requires a large database due to the nature of the network (structure and amount of parameters) and GPUs for efficient training implementation.

In general, training methods can be divided into supervised and unsupervised training. The former consists of labeled data that are usually very expensive and time consuming to obtain. Whereas the latter is the more common case and does not assume known ground-truth labels. However, supervised training usually achieves significantly better network performance compared to the unsupervised case. Therefore, a lot of resources are invested in labeling datasets for training. Thus, we focus here mainly on the supervised setting.

In neural networks, regardless of the model task, all training phases have the same goal: to minimize a pre-defined error function, also denoted as the loss/cost function. This is done in two stages: (a) a feed-forward pass of the input data through all the network layers, calculating the error using the predicted outputs and their ground-truth labels (if available); followed by (b) backpropogation of the errors through the network to update their weights, from the last layer to the first. This process is performed continuously to find the optimized values for the weights of the network.

The backpropagation algorithm provides the gradients of the error with respect to the network weights. These gradients are used to update the weights of the network. Calculating them based on the whole input data is computationally demanding and therefore, the common practice is to use subsets of the training set, termed *mini-batches*, and cycle over the entire training set multiple times. Each cycle of training over the whole dataset is termed an *epoch* and in every cycle the data samples are used in a random order to avoid biases. The training process ends when convergence in the loss function is obtained. Since most NN problems are not convex, an optimal solution is not assured. We turn now to describe in more details the training process using backpropagation.

4.1 Backpropogation

The backpropagation process is performed to update all the parameters of the model, with the goal of decreasing the loss function value. The process starts with a feed-forward pass of input data, $\mathbf{x}$, through all the network layers. After which the loss function value is calculated and denoted as $\mathcal{L}(\mathbf{x}, \mathbf{W})$, where $\mathbf{W}$ are the model parameters (including the model weights and biases, for formulation convenience). Then the backpropagation is initiated by computing the value of: $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$, followed by the update of the network weights. All the weights are updated recursively by calculating the gradients of every layer, from the final one to the input layer, using the chain rule.

Fig. 8: Simple classification model example, consisting of a two layered fully-connected model.

Denote the output of layer l as $\mathbf{z}^{(l)}$. Following the chain rule, the gradients of a given layer l with parameters $\mathbf{W}^{(l)}$ with respect to its input $\mathbf{z}^{(l)}$ are:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l-1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l)}(\mathbf{W}^{(l)}, \mathbf{z}^{(l-1)})}{\partial \mathbf{z}^{(l-1)}}, \tag{6}$$

and the gradients with respect to the parameters are:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l)}(\mathbf{W}^{(l)}, \mathbf{z}^{(l-1)})}{\partial \mathbf{W}^{(l)}}. \tag{7}$$

These two formulas of the backpropagation algorithm dictate the gradients calculation with respect to the parameters for each layer in the network and, therefore, the
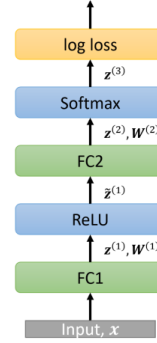
optimization can be performed using gradient-based optimizers (see Section 5 for more details).

To demonstrate the use of the backpropagation technique for the calculation of the network gradients, we turn to consider an example of a simple classification model with two-layers: a fully-connected layer with a ReLU activation function followed by another fully-connected layer with softmax function and log-loss. See Fig. 8 for the model illustration.

Denote by $\mathbf{z}^{(3)}$ the output of the softmax layer and assume that the input $\mathbf{x}$ belongs to class k (using one-hot encoding $y_k = 1$). The log-loss in this case is:

$$\mathcal{L} = -\sum_i \log\left(z_i^{(3)}\right) y_i = -\log\left(\frac{\exp\left(z_k^{(2)}\right)}{\sum_i \exp\left(z_i^{(2)}\right)}\right) = -z_k^{(2)} + \log\left(\sum_j \exp z_j^{(2)}\right). \quad (8)$$

For all $i \neq k$, the gradient of the error with respect to the softmax input $z_i^{(2)}$ is

$$\frac{\partial \mathcal{L}}{\partial z_i^{(2)}} = \frac{\exp\left(z_i^{(2)}\right)}{\sum_j \exp\left(z_j^{(2)}\right)} \equiv g_i. \quad (9)$$

Notice that this implies that we need to decrease the value of $z_i^{(2)}$ (the ith-logit) proportionally to the probability the network provides to it. While for the correct label, $i = k$, the derivative is:

$$\frac{\partial \mathcal{L}}{\partial z_k^{(2)}} = -1 + \frac{\exp\left(z_k^{(2)}\right)}{\sum_j \exp\left(z_j^{(2)}\right)} = g_k - 1, \quad (10)$$

which implies that the value of the logit element associated with the true label should be increased proportionally to the mistake the network is currently doing in the prediction.

The output $\mathbf{z}^{(2)}$ is a product of a fully-connect layer. Therefore, it can be formulated as follows:

$$\mathbf{z}^{(2)} = \mathbf{W}^{(2)}\tilde{\mathbf{z}}^{(1)}, \quad (11)$$

where $\tilde{\mathbf{z}}^{(1)}$ is the output of the ReLu function. Following the backpropagation rules we get that for this layer, the derivative with respect to its input is:

$$\frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{z}}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(2)}} \cdot \frac{\partial \mathbf{z}^{(2)}(\mathbf{W}^{(2)}, \tilde{\mathbf{z}}^{(1)})}{\partial \tilde{\mathbf{z}}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(2)}} \cdot \mathbf{W}^{(2)}, \quad (12)$$

whereas, the derivative with respect to its parameters is:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(2)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(2)}} \cdot \frac{\partial \mathbf{z}^{(2)}(\mathbf{W}^{(2)}, \tilde{\mathbf{z}}^{(1)})}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(2)}} \cdot \tilde{\mathbf{z}}^{(1)}. \quad (13)$$

The ReLU operation has no weight to update, but affects the gradients. The derivative of this stage follows:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{z}}^{(1)}} \cdot \frac{\partial \tilde{\mathbf{z}}^{(1)}(\mathbf{W}^{(1)}, I)}{\partial \mathbf{z}^{(1)}} = \begin{cases} 0, & \text{if } \mathbf{z}^{(1)} < 0 \\ \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{z}}^{(1)}}, & \text{otherwise.} \end{cases} \tag{14}$$

The final derivative with respect to the input $\partial \mathcal{L}/\partial \mathbf{x}$ is calculated similar to Eq. (12).

4.2 Training considerations

There are several considerations that should be addressed when training a NN. The most infamous is the *overfitting*, i.e., when the model too closely fits to the training dataset but does not generalize well to the test set. When this occurs, high training data precision is achieved, while the precision on the test data (not used during training) is low [129]. For this purpose, various regularization techniques have been proposed. We discuss some of them in Section 6.

A second consideration is the vanishing/exploding gradients occurring during training. Vanishing gradients are a result of multiplications with values smaller than one during their calculation in the backpropagation recursion. This can be resolved using activation functions and batch normalization detailed in Section 6. On the other hand, the gradients might also explode due to derivatives that are significantly larger than one in the backpropogation calculation. This makes the training unstable and may imply the need for re-designing the model (e.g., replace a vanilla RNN with a gated architecture such as LSTMs) or the use of gradient clipping [91].

Another important issue is the requirement that the training dataset must represent the true distribution of the task at hand. This usually enforces very large annotated datasets, which necessitate significant funding and manpower to obtain. In this case, considerable efforts must be invested to train the network using these large datasets, commonly with multiple GPUs for several days [62, 58]. One may use techniques such as domain adaptation [138] or transfer learning [128] to use already existing networks or large datasets for new tasks.

5 Training optimizers

Training neural networks is done by applying an optimizer to reach an optimal solution for the defined loss function. Its goal is to find the parameters of the model, e.g., weights and biases, which achieve minimum error for the training set samples: $(\mathbf{x}_i, y_i)$, where y_i is the label for the instance $\mathbf{x}_i$. For a loss function $\mathcal{L}(\cdot)$, the objective reads as:

$$\sum_i \mathcal{L}(\Phi(\mathbf{x}_i, \mathbf{W}), y_i), \tag{15}$$

for ease of notation, all model parameters are denoted as $\mathbf{W}$. A variety of optimizers have been proposed and implemented for minimizing Eq. 15. Yet, due to the size of the network and training dataset, mainly first-order methods are being considered,

i.e. strategies that rely only on the gradients (and not on second-order derivatives such as the Hessian).

Several gradient based optimizers are commonly used for updating the parameters of the model. These NN parameters are updated in the opposite direction of the objective function's gradient, $g_{\{GD, \mathcal{T}(t)\}}$, where $\mathcal{T}(t)$ is a randomly chosen subgroup of size $n' < n$ training samples used in iteration t (n is the size of the training dataset). Namely, at iteration t the weights are calculated as

$$\mathbf{W}(t) = \mathbf{W}(t-1) - \eta \cdot g_{\{GD, \mathcal{T}(t)\}}, \tag{16}$$

where η is the learning rate that determines the size of the steps taken to reach the (local) minimum and the gradient step, $g_{\{GD, \mathcal{T}(t)\}}$ is computed using the samples in $\mathcal{T}(t)$ as

$$g_{\{GD, \mathcal{T}(t)\}} = \frac{1}{n'} \sum_{i \in \mathcal{T}(t)} \nabla_W \mathcal{L}(\mathbf{W}(t); \mathbf{x}_i; y_i), \tag{17}$$

where the pair $(\mathbf{x}_i, y_i)$ is a training example and its corresponding label in the training set, and $\mathcal{L}$ is the loss function. However, needless to say that calculating the gradient on the whole dataset is computationally demanding. To this end, Stochastic Gradient Descent (SGD) is more popular, since it calculates the gradient in Eq. (17) for only one randomly chosen example from the data, i.e., $n' = 1$.

Since the update by SGD depends on a different sample at each iteration, it has a high variance that causes the loss value to fluctuate. While this behavior may enable it to jump to a new and potentially better local minima, it might ultimately complicates convergence, as SGD may keep overshooting. To improve convergence and exploit parallel computing power, mini-batch SGD is proposed in which the gradient in Eq. (17) is calculated with $n' > 1$ (but not all the data).

An acceleration in convergence may be obtained by using the history of the last gradient steps, in order to stabilize the optimization. One such approach uses adaptive momentum instead of a fixed step size. This is calculated based on exponential smoothing on the gradients, i.e:

$$\begin{aligned} M(t) &= \beta \cdot M(t-1) + (1-\beta) \cdot g_{\{SGD, \mathcal{T}(t)\}}, \\ \mathbf{W}(t) &= \mathbf{W}(t-1) - \eta M(t), \end{aligned} \tag{18}$$

where $M(t)$ approximates the 1st moment of $g_{\{SGD, \mathcal{T}(t)\}}$. A typical value for the constant is $\beta \sim 0.9$, which implies taking into account the last 10 gradient steps in the momentum variable $M(t)$ [95]. A well-known variant of Momentum proposed by Nestrov *et al.* [85] is the Nestrov Accelerated Gradient (NAG). It is similar to Momentum but calculates the gradient step as if the network weights have been already updated with the current Momentum direction.

Another popular technique is the Adaptive Moment Estimation (ADAM) [61], which also computes adaptive learning rates. In addition to storing an exponentially decaying average of past squared gradients, $V(t)$, ADAM also keeps an exponentially decaying average of past gradients, $M(t)$, in the following way:

$$M(t) = \beta_1 M(t-1) + (1 - \beta_1)g_t,$$
$$V(t) = \beta_2 V(t-1) + (1 - \beta_2)g_t^2, \tag{19}$$

where g_t is the gradient of the current batch, β_1 and β_2 are ADAM's hyperparameters, usually set to 0.9 and 0.999 respectively, and $M(t)$ and $V(t)$ are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively. Hence the name of the method - Adaptive Moment Estimation. As $M(t)$ and $V(t)$ are initialized as vectors of 0âĂŹs, the authors of ADAM observe that they are biased towards zero, especially during the initial time steps. To counteract these biases, a bias-corrected first and second moment are used: $\hat{M}(t) = M(t)/(1 - \beta_1(t))$ and $\hat{V}(t) = V(t)/(1 - \beta_2(t))$. Therefore, the ADAM update rule is as follows:

$$\mathbf{W}(t+1) = \mathbf{W}(t) - \frac{\eta}{\sqrt{\hat{V}(t) + \epsilon}}\hat{M}(t). \tag{20}$$

ADAM has two popular extensions: AdamW by Loshchilov *et al.* [78] and AMSGrad by Redddi *et al.* [97]. There are several additional common optimizers that have adaptive momentum, such as AdaGrad [29], AdaDelta [146] or RMSprop [21]. It must be noted that since the NN optimization is non-convex, the minimal error point reached by each optimizer is rarely the same. Thus, speedy convergence is not always favored. In particular, it has been observed that Momentum leads to better generalization than ADAM, which usually converges faster [60]. Thus, the common practice is to make the development with ADAM and then make the final training with Momentum.

(a) Original image

(b) Flip augmentation

(c) Crop and scale augmentation

(d) Noise augmentation

Fig. 9: Different image augmentations.

6 Training regularizations

One of the great advantageous of NN is their ability to generalize, i.e., correctly predict unseen data [52]. This must be ensured during the training process and is accomplished by several regularization methods, detailed here. The most common are weight decay [63], dropout [121], batch normalization [50] and the use of data augmentation [116].

Weight decay is a basic tool to limit the growth of the weights by adding a regularization term to the cost function for large weights, which is the sum of squares of all the weights, i.e., $\sum_i |W_i|^2$.

The key idea in *dropout* is to randomly drop units (along with their connections) from the neural network during training and thus prevent units from co-adapting too much. The percentage of dropped units is critical since a large amount will result in poor learning. Common values are $20\% - 50\%$ dropped units.

Batch normalization is a mean to deal with changes in the distribution of the model's parameters during training. The layers need to adapt to these (often noisy) changes between instances during training. Batch normalization causes the features of each training batch to have a mean of 0 and a variance of 1 in the layer it is being applied. To normalize a value across a batch, i.e. to batch normalize the value, the batch mean, μ_B, is subtracted and the result is divided by the batch standard deviation, $\sqrt{\sigma_B^2 + \epsilon}$. Note that a small constant ϵ is added to the variance in order to avoid dividing by zero. The batch normalizing transform of a given input, $\mathbf{x}$, is:

$$\text{BN}(\mathbf{x}) = \gamma \left(\frac{\mathbf{x} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \right) + \beta. \tag{21}$$

Notice the (learnable) scale and bias parameters γ and β, which provides the NN with freedom to deviate from the zero mean and unit variance. BN is less effective when used with small batch sizes since in this case the statistics calculated per each is less accurate. Thus, techniques such as group normalization [139] or Filter Response Normalization (FRN) [118] have been proposed.

Data augmentation is a very common strategy used during training to artificially "increase" the size of the training data and make the network robust to transformations that do not change the input label. For example, in the task of classification a shifted cat is still a cat; see Fig. 9 for more similar augmentation. In the task of denoising, flipped noisy input should result in a flipped clean output. Thus, during training the network is trained also with the transformed data to improve its performance.

Common augmentations are randomly flipping, rotating, scaling, cropping, translating, or adding noise to the data. Other more sophisticated techniques that lead to a significant improvement in network performance include mixup [147], cutout [26] and augmentations that are learned automatically [18, 71, 19].

7 Advanced NN architectures

The basic building blocks, which compose the NN model architecture, are used in frequently innovative structures. In this section, such known architectures with state-of-the-art performance are presented, divided by tasks and data types: detection and segmentation tasks are described in Section 7.1, sequential data handling is elaborated in Section 7.2 and processing data on irregular grids is presented in Section 7.3. Clearly, there are many other use-cases and architectures, which are not mentioned here.

7.1 Deep learning for detection and segmentation

Many research works focus on detecting multiple objects in a scene, due to its numerous applications. This problem can be divided into four sub-tasks as follows, where we refer here to image datasets although the same concept can be applied to different domains as well.

1. *Classification and localization*: The main object in the image is detected and then localized by a surrounding bounding box and classified from a pre-known set.
2. *Object detection*: Detection of all objects in a scene that belong to a pre-known set and then classifying and providing a bounding box for each of them.
3. *Semantic segmentation*: Partitioning the image into coherent parts by assigning each pixel in the image with its own classification label (associated with the object the pixel belongs to). For example, having a pixel-wise differentiation between animals, sky and background (generic class for all object that no class is assigned to) in an image.
4. *Instance segmentation*: Multiple objects segmentation and classification from a pre-known set (similar to object detection but for each object all its pixels are identified instead of providing a bounding box for it).

Today, state-of-the-art object detection performance is achieved with architectures such as Faster-RCNN [103, 135], You Only Look Once (YOLO) [98, 99, 100], Single Shot Detector (SSD) [75] and Fully Convolutional One-Stage Object Detection (FCOS) [150]. The object detection models provide a list of detected bounding boxes with the class of each of them.

Segmentation tasks are mostly implemented using fully convolutional network. Known segmentation models include UNet [104], Mask-RCNN [44] and Deeplab [11]. These architecture have the same input/output spatial size since the output represents the segmentation map of the input image.

Both object detection and segmentation tasks are analyzed via the Intersection over Union (IoU) metric. The IoU is defined as the ratio between the intersection area of the object's ground-truth pixels, B_g, with the corresponding predicted pixels, B_p, and the union of these group of pixels. The IoU is formulated as:

$$\text{IoU} = \frac{\text{Area}\{B_g \cap B_p\}}{\text{Area}\{B_g \cup B_p\}}. \tag{22}$$

As this measure evaluate only the quality of the bounding box, a mean Average Precision (mAP) is commonly used to evaluate the models performance. The mAP is defined as the ratio of the correctly detected (or segmented) objects, where an object is considered to be detected correctly if there is a bounding box for it with the correct class and a IoU greater than 0.5 (or another specified constant).

Another common evaluation metric is the F1 score, which is the harmonic average of the precision and the recall values. See Eq. (24) below. They are calculated using the following definitions that are presented for the case of semantic segmentation:

- True Positive (TP): the predicted class of a pixel matches it ground-truth label.
- False Positive (FP): the predicted pixel of an object was falsely determined.
- False Negative (FN): a ground-truth pixel of an object was not predicted.

Now that they are defined, the *precision*, *recall* and F1 are given by:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{23}$$

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{24}$$

7.2 Deep learning on sequential data

Sequential data are composed of time-sensitive signals such as the output of different sensors, audio recordings, NLP sentences or any signal that its order is of importance. Therefore, this data must be processed accordingly.

Initially, sequential data was processed with Recurrent NN (RNN) [51] that has recurrent (feedback) connections, where outputs of the network at a given time-step serve as input to the model (in addition to the input data) at the next time-step. This introduces the time dependent feature of the NN. A RNN is illustrated in Fig. 4.

However, it was quickly realized that during training, vanilla RNNs suffer from vanishing/exploding gradients. This phenomena, originated from the use of finite-precision back-propagation process, limits the size of the sequence.

To this end, a corner stone block is used: the Long-Short-Term-Memory (LSTM [46]). Mostly used for NLP tasks, the LSTM is a RNN block with gates. During training, these gates learn which part of the sentence to forget or to memorize. The gating allow some of the gradients to backpropagate unchanged, which aids the vanishing gradient symptom. Notice that RNNs (and LSTMs) can process a sentence in a bi-directional mode, i.e., process a sentence in two directions, from the beginning to the end and vice verse. This mechanism allows a better grasp of the input context by the network. Examples for popular research tasks in NLP data include question answering [96], translation [65] and text generation [41].

Sentences processing. An important issue in NLP is representing words in preparation to serve as network input. The use of straight forward indices is not effective since there are thousands of words in a language. Therefore, it is common to process text data via *word embedding*, which is a vector representation of each word in some fixed dimension. This method enables to encapsulate relationships between words.

A classic methodology to calculate the word embedding is *Word2Vec* [81], in which these vector representations are calculated using a NN model that learn their context. More advanced options for creating efficient word representations include BERT [25], ELMO [92], RoBERTa [77] and XLNet [144].

Audio processing. Audio recordings are used for multiple interesting tasks, such as speech to text, text to speech and speech processing. In the audio case, the common input to speech systems is the Mel Frequency Cepstral Coefficient (MFCC) or a Short Time Fourier Transform (STFT) image, as opposed to the audio raw-data. A milestone example for speech processing NN architecture is the *wavenet* [89]. This architecture is an autoregressive model that synthesizes speech or audio signals. It is based on dilated convolutional layers that have large receptive fields, that allow efficient processing. Another prominent synthesis model for sequential data is the Tacotron [113].

The attention model. As mentioned in Section 2, one may use RNN for translation using the encoder decoder model, which encodes a source sentence into a vector, which is then decoded to a target language. Instead of relying on a compressed vector, which may lose information, the *attention models* learn where or what to focus on from the whole input sequence. Introduced in 2015 [3], attention models have shown superior performance over encoder-decoder architectures in tasks such as translation, text to speech and image captioning. Recently, it has been suggested to replace the recurrent network structure totally by the attention mechanism, which results with the *transformers network* models [131].

7.3 Deep learning on irregular grids

A wide variety of data acquisition mechanisms do not represent the data on a grid as is common with images data. A prominent example is 3D imaging (e.g. using LIDAR), where the input data are represented as points in a 3D space with or without color information. Processing such data is not trivial as standard network components, such as convolutions, assume a grid of the data. Therefore, they cannot be applied as is and custom operations are required. We focus our discussion here on the case of NN for 3D data.

Today, real-time processing of 3D scenes can be achieved with advanced NN models that are customized to these irregular grids. The different processing techniques for these irregular grid data can be divided by the type of representation used for the data:

1. **Points processing.** 3D data points are processed as points in space, i.e., a list of the point coordinates is given as the input to the NN. A popular network for

this representation is *PointNet* [93]. It is the first to efficiently achieve satisfactory results directly on the point cloud. Yet, it is limited by the number of points that can be analyzed, computational time and performance. Some more recent models that improves its performance include PointNet++ [94], PointCNN [69], DGCNN [136]. Strategies to improve its efficiency have been proposed in learning to sample [28] and RandLA-Net [48].

2. **Multi-view 2D projections.** 3D data points are projected (from various angles) to the 2D domain so that known 2D processing techniques can be used [70, 56].
3. **Volumetric (voxels).** 3D data points are represented in a grid-based *voxel* representation. This is analogous to a 2D representation and is therefore advantageous. However, it is computationally exhaustive [140] and losses resolution.
4. **Meshes.** Mesh represents the 3D domain via a graph that defines the connectivity between the different points. Yet, this graph has a special structure such that it creates the surface of the 3D shape (in the common case of triangular mesh, the shape surface is presented by a set of triangles connected to each other). In 2015 Masci *et al.* [6] have shown it is possible to learn features using DL on meshes. Since then, a significant advancement has been made in mesh processing [43, 83].
5. **Graphs.** Graph representations are common for representing non-linear structured data. Some works have proposed efficient NN models for 3D data points on a grid-based graph structure [122, 86].

8 Summary

This chapter provided a general survey of the basic concepts in neural networks. As this field is expanding very fast, the space is too short to describe all the developments in it, even though most of them are from the past eight years. Yet, we briefly mention here few important problems that are currently being studied.

1. **Domain adaptation and transfer learning.** As many applications necessitate data that is very difficult to obtain, some methods aim at training models based on scarce datasets. A popular methodology for dealing with insufficient annotated data is *domain adaptation*, in which a robust and high performance NN model, trained on a source distribution, is used to aid the training of a similar model (usually with the same goal, e.g., in classification the same classes are searched for) on data from a target distribution that are either unlabelled or small in number [33, 90, 117]. An example is adapting a NN trained on simulation data to real-life data with the same labels [130, 47]. On a similar note, *transfer learning* [128, 27] can also be used in similar cases, where in addition to the difference in the data, the input and output tasks are not the same but only similar (in domain adaptation the task is the same and only the distributions are different). One such example, is using a network trained on natural images to classify medical data [4].
2. **Few shot learning.** A special case of learning with small datasets is *few-shot learning* [137], where one is provided either with just semantic information of the

target classes (zero-shot learning), only one labelled example per class (1-shot learning) or just few samples (general few-shot learning). Approaches developed for these problems have shown great success in many applications, such as image classification [125, 111, 123], object detection [57] and segmentation [8].

3. **On-line learning.** Various deep learning challenges occur due to new distributions or class types introduced to the model during a continuous operation of the system (post-training), and now must be learnt by the model. The model can update its weights to incorporate these new data using *on-line learning* techniques. There is a need for special training in this case, as systems that just learn based on the new examples may suffer from a reduced performance on the original data. This phenomena is known as catastrophic forgetting [59]. Often, the model tends to forget the representation of part of the distribution it already learned and thus it develops a bias towards the new data. A specific example of on-line learning is *incremental learning* [9], where the new data is of different classes than the original ones.

4. **AutoML.** When approaching real-life problems, there is an inherent pipeline of tasks to be preformed before using DL tools, such as problem definition, preparing the data and processing it. Commonly, these tasks are preformed by specialists and require deep system understating. To this end, the *autoML* paradigm attempts to generalize this process by automatically learning and tuning the model used [32]. A particular popular task in autoML is *Neural Architecture Search (NAS)* [30]. This is of interest since the NN architecture restricts its performance. However, searching for the optimal architecture for a specific task, and from a set of pre-defined operations, is computationally exhaustive when performed in a straight forward manner. Therefore, on-going research attempts to overcome this limitation. An example is the DARTS [74] strategy and its extensions [87, 12] where the key contribution is finding, in a differentiable manner, the connections between network operations that form a NN architecture. This framework decreases the search time and improves the final accuracy.

5. **Reinforcement Learning.** To date, the most effective training method for decision based actions, such as robot movement and video games, is *Reinforcement Learning* (RL) [55, 127]. In RL, the model tries to maximize some pre-defined award score by learning which action to take, from a set of defined actions in specific scenarios.

To summarize, being able to efficiently train deep neural networks has revolutionized almost every aspect of the modern day-to-day life. Examples span from bio-medical applications through computer graphics in movies and videos to international scale applications of big companies, such as Google, Amazon, Microsoft, Apple and Facebook. Evidently, this theory is drawing much attention and we believe there is still much to unravel, including exploring and understanding the NN's potential abilities and limitations.

The next chapters detail Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), generative models and autoencoders. All are very important paradigms that are used in numerous applications.

References

1. Achlioptas, P., Diamanti, O., Mitliagkas, I., Guibas, L.: Learning representations and generative models for 3D point clouds. In: J. Dy, A. Krause (eds.) Proceedings of the 35th International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 80, pp. 40–49. PMLR, StockholmsmÃďssan, Stockholm Sweden (2018)
2. Atlason, H.E., AskellLove, Sigurdsson, S., Gudnason, V., Ellingsen, L.M.: Unsupervised brain lesion segmentation from mri using a convolutional autoencoder. In: Medical Imaging 2019: Image Processing, vol. 10949, p. 109491H. International Society for Optics and Photonics (2019)
3. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. In: 3rd International Conference on Learning Representations, ICLR (2015)
4. Bar, Y., Diamant, I., Wolf, L., Greenspan, H.: Deep learning with non-medical training used for chest pathology identification. In: Medical Imaging 2015: Computer-Aided Diagnosis, vol. 9414, pp. 215 – 221. International Society for Optics and Photonics, SPIE (2015)
5. Ben-Cohen, A., Diamant, I., Klang, E., Amitai, M., Greenspan, H.: Fully convolutional network for liver segmentation and lesions detection. In: Deep learning and data labeling for medical applications, pp. 77–85. Springer (2016)
6. Boscaini, D., Masci, J., Melzi, S., Bronstein, M.M., Castellani, U., Vandergheynst, P.: Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks. Comput. Graph. Forum **34**, 13–23 (2015)
7. Brock, A., Donahue, J., Simonyan, K.: Large scale GAN training for high fidelity natural image synthesis. In: International Conference on Learning Representations (ICLR) (2019)
8. Caelles, S., Maninis, K.K., Pont-Tuset, J., Leal-Taixé, L., Cremers, D., Van Gool, L.: One-shot video object segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 221–230 (2017)
9. Castro, F.M., Marín-Jiménez, M.J., Guil, N., Schmid, C., Alahari, K.: End-to-end incremental learning. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 233–248 (2018)
10. Chen, L.C., Zhu, Y., Papandreou, G., Schroff, F., Adam, H.: Encoder-decoder with atrous separable convolution for semantic image segmentation. In: Proceedings of the European conference on computer vision (ECCV), pp. 801–818 (2018)
11. Chen, L.C., Zhu, Y., Papandreou, G., Schroff, F., Adam, H.: Encoder-decoder with atrous separable convolution for semantic image segmentation. In: ECCV (2018)
12. Chen, X., Xie, L., Wu, J., Tian, Q.: Progressive darts: Bridging the optimization gap for nas in the wild. arXiv preprint arXiv:1912.10952 (2019)
13. Chen, Z., Zhang, J., Tao, D.: Progressive lidar adaptation for road detection. IEEE/CAA Journal of Automatica Sinica **6**(3), 693–702 (2019)
14. Cho, K., van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder–decoder approaches. In: Workshop on Syntax, Semantics and Structure in Statistical Translation, pp. 103–111. Association for Computational Linguistics (2014)
15. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). CoRR (2015)
16. Cortes, C., Vapnik, V.: Support vector networks. Machine Learning **20**, 273–297 (1995)
17. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press (2000). DOI 10.1017/CBO9780511801389
18. Cubuk, E.D., Zoph, B., Mane, D., Vasudevan, V., Le, Q.V.: Autoaugment: Learning augmentation strategies from data. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
19. Cubuk, E.D., Zoph, B., Shlens, J., Le, Q.V.: Randaugment: Practical automated data augmentation with a reduced search space. arXiv (2019)
20. Dahl, G.E., Sainath, T.N., Hinton, G.E.: Improving deep neural networks for lvcsr using rectified linear units and dropout. In: ICASSP, pp. 8609–8613. IEEE (2013)

21. Dauphin, Y.N., de Vries, H., Chung, J., Bengio, Y.: Rmsprop and equilibrated adaptive learning rates for non-convex optimization. CoRR (2015)
22. Deng, J., Dong, W., Socher, R., jia Li, L., Li, K., Fei-fei, L.: Imagenet: A large-scale hierarchical image database. CVPR (2009)
23. Deng, J., Guo, J., Xue, N., Zafeiriou, S.: Arcface: Additive angular margin loss for deep face recognition. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
24. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding (2018)
25. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT (2019)
26. DeVries, T., Taylor, G.W.: Improved regularization of convolutional neural networks with cutout. arXiv (2017)
27. Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T.: Decaf: A deep convolutional activation feature for generic visual recognition. In: E.P. Xing, T. Jebara (eds.) Proceedings of the 31st International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 32, pp. 647–655. PMLR, Bejing, China (2014)
28. Dovrat, O., Lang, I., Avidan, S.: Learning to sample. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
29. Duchi, J., Hazan, E., yORAM Singer: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**, 2121–2159 (2011)
30. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. J. Mach. Learn. Res. **20**, 55:1–55:21 (2018)
31. van Engelen, J.E., Hoos, H.H.: A survey on semi-supervised learning. Machine Learning (2019). DOI 10.1007/s10994-019-05855-6. URL `https://doi.org/10.1007/s10994-019-05855-6`
32. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, R. Garnett (eds.) Advances in Neural Information Processing Systems 28, pp. 2962–2970. Curran Associates, Inc. (2015). URL `http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf`
33. Ganin, Y., Lempitsky, V.: Unsupervised domain adaptation by backpropagation. arXiv preprint arXiv:1409.7495 (2014)
34. Gao, C., Gu, D., Zhang, F., Yu, Y.: Reconet: Real-time coherent video style transfer network. In: Asian Conference on Computer Vision, pp. 637–653. Springer (2018)
35. Gatys, L.A., Ecker, A.S., Bethge, M.: Image style transfer using convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2414–2423 (2016)
36. Gatys, L.A., Ecker, A.S., Bethge, M.: Image style transfer using convolutional neural networks. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 2414–2423 (2016)
37. Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to forget: Continual prediction with lstm. ICANN (1999)
38. Gibiansky, A., Arik, S., Diamos, G., Miller, J., Peng, K., Ping, W., Raiman, J., Zhou, Y.: Deep voice 2: Multi-speaker neural text-to-speech. In: Advances in neural information processing systems, pp. 2962–2970 (2017)
39. Goodfellow, I., Jean Pouget-Abadieand, M.M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, K.Q. Weinberger (eds.) Advances in Neural Information Processing Systems 27, pp. 2672–2680. Curran Associates, Inc. (2014)
40. Greenspan, H., van Ginneken, B., Summers, R.M.: Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique. CVPR **35**, 1153 – 1159 (2016)

41. Guo, J., Lu, S., Cai, H., Zhang, W., Yu, Y., Wang, J.: Long text generation via adversarial training with leaked information. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
42. Haim, H., Elmalem, S., Giryes, R., Bronstein, A.M., Marom, E.: Depth estimation from a single image using deep learned phase coded mask. IEEE Transactions on Computational Imaging **4**(3), 298–310 (2018)
43. Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., Cohen-Or, D.: Meshcnn: A network with an edge. ACM Transactions on Graphics (TOG) **38**(4), 90 (2019)
44. He, K., Gkioxari, G., Dollár, P., Girshick, R.B.: Mask r-cnn. IEEE International Conference on Computer Vision (ICCV) pp. 2980–2988 (2017)
45. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Comput. **18**(7), 1527–1554 (2006). DOI 10.1162/neco.2006.18.7.1527. URL http://dx.doi.org/10.1162/neco.2006.18.7.1527
46. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997). DOI 10.1162/neco.1997.9.8.1735
47. Hoffman, J., Tzeng, E., Park, T., Zhu, J.Y., Isola, P., Saenko, K., Efros, A., Darrell, T.: CyCADA: Cycle-consistent adversarial domain adaptation. In: J. Dy, A. Krause (eds.) Proceedings of the 35th International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 80, pp. 1989–1998. PMLR, StockholmsmÃďssan, Stockholm Sweden (2018)
48. Hu, Q., Yang, B., Xie, L., Rosa, S., Guo, Y., Wang, Z., Trigoni, N., Markham, A.: Randla-net: Efficient semantic segmentation of large-scale point clouds. arXiv preprint arXiv:1911.11236 (2019)
49. Hubel, D.H., Wiesel, T.N.: Receptive fields of single neurons in the cat's striate cortex. Journal of Physiology **148**, 574–591 (1959)
50. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on Machine Learning, vol. 37, pp. 448–456 (2015)
51. Jain, L.C., Medsker, L.R.: Recurrent Neural Networks: Design and Applications, 1st edn. CRC Press, Inc., Boca Raton, FL, USA (1999)
52. Jakubovitz, D., Giryes, R., Rodrigues, M.R.D.: Generalization Error in Deep Learning, pp. 153–193. Springer International Publishing, Cham (2019)
53. Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: European conference on computer vision, pp. 694–711. Springer (2016)
54. Kadlec, R., Schmid, M., Bajgar, O., Kleindienst, J.: Text understanding with the attention sum reader network. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 908–918. Association for Computational Linguistics, Berlin, Germany (2016). DOI 10.18653/v1/P16-1086
55. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of artificial intelligence research **4**, 237–285 (1996)
56. Kalogerakis, E., Averkiou, M., Maji, S., Chaudhuri, S.: 3d shape segmentation with projective convolutional networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 6630–6639 (2016)
57. Karlinsky, L., Shtok, J., Harary, S., Schwartz, E., Aides, A., Feris, R., Giryes, R., Bronstein, A.M.: Repmet: Representative-based metric learning for classification and few-shot object detection. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
58. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of GANs for improved quality, stability, and variation. In: International Conference on Learning Representations (2018). URL https://openreview.net/forum?id=Hk99zCeAb
59. Kemker, R., McClure, M., Abitino, A., Hayes, T.L., Kanan, C.: Measuring catastrophic forgetting in neural networks. In: Thirty-second AAAI conference on artificial intelligence (2018)
60. Keskar, N.S., Socher, R.: Improving generalization performance by switching from adam to sgd. arXiv preprint arXiv:1712.07628 (2017)

61. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR (2014)
62. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (eds.) Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc. (2012)
63. Krogh, A., Hertz, J.A.: A simple weight decay can improve generalization. In: J.E. Moody, S.J. Hanson, R.P. Lippmann (eds.) Advances in Neural Information Processing Systems 4, pp. 950–957. Morgan-Kaufmann (1992). URL http://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization.pdf
64. Kwon, D., Kim, H., Kim, J., Suh, S.C., Kim, I., Kim, K.J.: A survey of deep learning-based network anomaly detection. Cluster Computing **22**(1), 949–961 (2019). DOI 10.1007/s10586-017-1117-8
65. Lample, G., Conneau, A., Denoyer, L., Ranzato, M.: Unsupervised machine translation using monolingual corpora only. arXiv preprint arXiv:1711.00043 (2017)
66. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural Comput. **1**(4), 541–551 (1989). DOI 10.1162/neco.1989.1.4.541. URL http://dx.doi.org/10.1162/neco.1989.1.4.541
67. LeCun, Y., Boser, B.E., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W.E., Jackel, L.D.: Hand-written digit recognition with a back-propagation network. NIPS (1990)
68. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE, pp. 2278–2324 (1998)
69. Li, Y., Bu, R., Sun, M., Wu, W., Di, X., Chen, B.: Pointcnn: Convolution on x-transformed points. In: NeurIPS (2018)
70. Li, Y., Pirk, S., Su, H., Qi, C.R., Guibas, L.J.: Fpnn: Field probing neural networks for 3d data. In: D.D. Lee, M. Sugiyama, U.V. Luxburg, I. Guyon, R. Garnett (eds.) Advances in Neural Information Processing Systems 29, pp. 307–315. Curran Associates, Inc. (2016). URL http://papers.nips.cc/paper/6416-fpnn-field-probing-neural-networks-for-3d-data.pdf
71. Lim, S., Kim, I., Kim, T., Kim, C., Kim, S.: Fast autoaugment. In: Advances in Neural Information Processing Systems (NeurIPS) (2019)
72. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: Proceedings of the IEEE international conference on computer vision, pp. 2980–2988 (2017)
73. Liu, G., Reda, F.A., andx Ting-Chun Shih, K.J.S., Tao, A., Catanzaro, B.: Image inpainting for irregular holes using partial convolutions. In: The European Conference on Computer Vision (ECCV) (2018)
74. Liu, H., Simonyan, K., Yang, Y.: DARTS: Differentiable architecture search. In: International Conference on Learning Representations (2019)
75. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: ECCV (2016)
76. Liu, W., Wen, Y., Yu, Z., Li, M., Raj, B., Song, L.: Sphereface: Deep hypersphere embedding for face recognition. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 6738–6746 (2017)
77. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 (2019)
78. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: ICLR (2017)
79. Ma, W.C., Wang, S., Hu, R., Xiong, Y., Urtasun, R.: Deep rigid instance scene flow. In: CVPR (2019)
80. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics **5**(4), 115–133 (1943). DOI 10.1007/BF02478259
81. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K.Q. Weinberger (eds.) Advances in Neural Information Processing Systems 26, pp. 3111–3119.

Curran Associates, Inc. (2013). URL `http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf`

82. Minsky, M., Papert, S.: Perceptrons: An Introduction to Computational Geometry. MIT Press, Cambridge, MA, USA (1969)

83. Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., Bronstein, M.M.: Geometric deep learning on graphs and manifolds using mixture model cnns. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR, pp. 5425–5434 (2017). DOI 10.1109/CVPR.2017.576

84. Nah, S., Kim, T.H., Lee, K.M.: Deep multi-scale convolutional neural network for dynamic scene deblurring. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3883–3891 (2017)

85. Nesterov, Y.E.: A method for solving the convex programming problem with convergence rate o $(1/k^2)$. In: Dokl. akad. nauk Sssr, vol. 269, pp. 543–547 (1983)

86. Niepert, M., Ahmed, M., Kutzkov, K.: Learning convolutional neural networks for graphs. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML'16, pp. 2014–2023. JMLR.org (2016). URL `http://dl.acm.org/citation.cfm?id=3045390.3045603`

87. Noy, A., Nayman, N., Ridnik, T., Zamir, N., Doveh, S., Friedman, I., Giryes, R., Zelnik-Manor, L.: Asap: Architecture search, anneal and prune. arXiv preprint arXiv:1904.04123 (2019)

88. Ongie, G., Willett, R., Soudry, D., Srebro, N.: A function space view of bounded norm infinite width re{lu} nets: The multivariate case. In: International Conference on Learning Representations (ICLR) (2020)

89. van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K.: Wavenet: A generative model for raw audio. In: Arxiv (2016). URL `https://arxiv.org/abs/1609.03499`

90. Pan, S.J., Tsang, I.W., Kwok, J.T., Yang, Q.: Domain adaptation via transfer component analysis. IEEE Transactions on Neural Networks **22**(2), 199–210 (2010)

91. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: S. Dasgupta, D. McAllester (eds.) Proceedings of the 30th International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 28, pp. 1310–1318. PMLR, Atlanta, Georgia, USA (2013)

92. Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L.: Deep contextualized word representations. In: Proc. of NAACL (2018)

93. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 77–85 (2016)

94. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. arXiv preprint arXiv:1706.02413 (2017)

95. Qian, N.: On the momentum term in gradient descent learning algorithms. Neural Networks **12**(1), 145–151 (1999)

96. Radford, A., Sutskever, I.: Improving language understanding by generative pre-training. In: arxiv (2018)

97. Reddi, S.J., Kale, S., Kumar, S.: On the convergence of adam and beyond. In: International Conference on Learning Representations (ICLR) (2018)

98. Redmon, J., Divvala, S.K., Girshick, R.B., Farhadi, A.: You only look once: Unified, real-time object detection. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 779–788 (2015)

99. Redmon, J., Farhadi, A.: Yolo9000: Better, faster, stronger. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 6517–6525 (2016)

100. Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. ArXiv **abs/1804.02767** (2018)

101. Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., Lee, H.: Generative adversarial text to image synthesis. In: Proceedings of the 33rd International Conference on International

Conference on Machine Learning - Volume 48, ICMLâĂŹ16, p. 1060âĂŞ1069. JMLR.org (2016)

102. Remez, T., Litany, O., Giryes, R., Bronstein, A.M.: Class-aware fully convolutional gaussian and poisson denoising. IEEE Transactions on Image Processing **27**(11), 5707–5722 (2018)

103. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, R. Garnett (eds.) Advances in Neural Information Processing Systems 28, pp. 91–99. Curran Associates, Inc. (2015)

104. Ronneberger, O., P.Fischer, Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: Medical Image Computing and Computer-Assisted Intervention (MICCAI), *LNCS*, vol. 9351, pp. 234–241. Springer (2015)

105. Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review pp. 65–386 (1958)

106. Ruck, D.W., Rogers, S.K.: Feature Selection Using a Multilayer Perceptron. Journal of Neural Network Computing **2**(July 1993), 40–48 (1990)

107. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning Representations by Back-propagating Errors. Nature **323**(6088), 533–536 (1986). DOI 10.1038/323533a0

108. Safran, I., Eldan, R., Shamir, O.: Depth separations in neural networks: What is actually being separated? In: Conference on Learning Theory (COLT), pp. 2664–2666. PMLR (2019)

109. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. CVPR pp. 815–823 (2015)

110. Schwartz, E., Giryes, R., Bronstein, A.M.: Deepisp: Toward learning an end-to-end image processing pipeline. IEEE Transactions on Image Processing **28**(2), 912–923 (2019). DOI 10.1109/TIP.2018.2872858

111. Schwartz, E., Karlinsky, L., Shtok, J., Harary, S., Marder, M., Kumar, A., Feris, R., Giryes, R., Bronstein, A.: Delta-encoder: an effective sample synthesis method for few-shot object recognition. In: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (eds.) Advances in Neural Information Processing Systems 31, pp. 2845–2855. Curran Associates, Inc. (2018)

112. Shaham, T.R., Dekel, T., Michaeli, T.: Singan: Learning a generative model from a single natural image. In: The IEEE International Conference on Computer Vision (ICCV) (2019)

113. Shen, J., Pang, R., Weiss, R.J., Schuster, M., Jaitly, N., Yang, Z., Chen, Z., Zhang, Y., Wang, Y., Skerrv-Ryan, R., Saurous, R.A., Agiomvrgiannakis, Y., Wu, Y.: Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In: International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4779–4783 (2018)

114. Shiloh, L., Eyal, A., Giryes, R.: Efficient processing of distributed acoustic sensing data using a deep learning approach. J. Lightwave Technol. **37**(18), 4755–4762 (2019)

115. Shocher, A., Bagon, S., Isola, P., Irani, M.: Ingan: Capturing and retargeting the "dna" of a natural image. In: The IEEE International Conference on Computer Vision (ICCV) (2019)

116. Shorten, C., Khoshgoftaar, T.M.: A survey on image data augmentation for deep learning. Journal of Big Data **6**(1), 60 (2019). DOI 10.1186/s40537-019-0197-0

117. Shu, R., Bui, H.H., Narui, H., Ermon, S.: A DIRT-T approach to unsupervised domain adaptation. In: 6th International Conference on Learning Representations, ICLR (2018)

118. Singh, S., Krishnan, S.: Filter response normalization layer: Eliminating batch dependence in the training of deep neural networks. arXiv (2019)

119. Sønderby, C.K., Raiko, T., Maaløe, L., Sønderby, S.K., Winther, O.: Ladder variational autoencoders. In: Advances in neural information processing systems, pp. 3738–3746 (2016)

120. Sotelo, J., Mehri, S., Kumar, K., Santos, J.F., Kastner, K., Courville, A.C., Bengio, Y.: Char2wav: End-to-end speech synthesis. In: ICLR (2017)

121. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)

122. Such, F.P., Sah, S., Domínguez, M., Pillai, S., Zhang, C., Michael, A., Cahill, N.D., Ptucha, R.W.: Robust spatial filtering with graph convolutional neural networks. IEEE Journal of Selected Topics in Signal Processing **11**, 884–896 (2017)

123. Sun, Q., Liu, Y., Chua, T.S., Schiele, B.: Meta-transfer learning for few-shot learning. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
124. Sun, R.: Optimization for deep learning: theory and algorithms. arXiv preprint arXiv:1912.08957 (2019)
125. Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P.H., Hospedales, T.M.: Learning to compare: Relation network for few-shot learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1199–1208 (2018)
126. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems, pp. 3104–3112. Curran Associates, Inc. (2014)
127. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
128. Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., Liu, C.: A survey on deep transfer learning. In: V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, I. Maglogiannis (eds.) Artificial Neural Networks and Machine Learning – ICANN 2018, pp. 270–279. Springer International Publishing, Cham (2018)
129. Tetko, I.V., Livingstone, D.J., Luik, A.I.: Neural network studies. 1. comparison of overfitting and overtraining. Journal of chemical information and computer sciences **35**(5), 826–833 (1995)
130. Tzeng, E., Hoffman, J., Saenko, K., Darrell, T.: Adversarial discriminative domain adaptation. IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 2962–2971 (2017)
131. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is all you need. In: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (eds.) Advances in Neural Information Processing Systems 30, pp. 5998–6008. Curran Associates, Inc. (2017). URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf
132. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th international conference on Machine learning, pp. 1096–1103 (2008)
133. Vinyals, O., Toshev, A., Bengio, S., Erhan, D.: Show and tell: A neural image caption generator. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)
134. Wang, H., Wang, Y., Zhou, Z., Ji, X., Li, Z., Gong, D., Zhou, J., Liu, W.: Cosface: Large margin cosine loss for deep face recognition. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (2018)
135. Wang, X., Shrivastava, A., Gupta, A.: A-fast-rcnn: Hard positive generation via adversary for object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2606–2615 (2017)
136. Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph cnn for learning on point clouds. ACM Transactions on Graphics (TOG) (2019)
137. Wang, Y., Yao, Q.: Generalizing from a few examples: A survey on few-shot learning. ArXiv (2019)
138. Wilson, G., Cook, D.J.: A survey of unsupervised deep domain adaptation. In: arxiv (2018)
139. Wu, Y., He, K.: Group normalization. In: The European Conference on Computer Vision (ECCV) (2018)
140. Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3d shapenets: A deep representation for volumetric shapes. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 1912–1920 (2014)
141. Xu, B., Wang, N., Chen, T., Li, M.: Empirical evaluation of rectified activations in convolutional network. ArXiv (2015)
142. Yang, C., Lu, X., Lu, Z., Shechtman, E., Wang, O., Li, H.: High-resolution image inpainting using multi-scale neural patch synthesis. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6721–6729 (2017)

143. Yang, W., Zhang, X., Tian, Y., Wang, W., Xue, J., Liao, Q.: Deep learning for single image super-resolution: A brief review. IEEE Transactions on Multimedia **21**(12), 3106–3121 (2019). DOI 10.1109/TMM.2019.2919431

144. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., Le, Q.V.: Xlnet: Generalized autoregressive pretraining for language understanding (2019)

145. Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., Huang, T.S.: Free-form image inpainting with gated convolution. In: The IEEE International Conference on Computer Vision (ICCV) (2019)

146. Zeiler, M.D.: Adadelta: An adaptive learning rate method. ArXiv **abs/1212.5701** (2012)

147. Zhang, H., Cisse, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. In: International Conference on Learning Representations (2018). URL `https://openreview.net/forum?id=r1Ddp1-Rb`

148. Zhang, K., Zuo, W., Chen, Y., Meng, D., Zhang, L.: Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. IEEE Transactions on Image Processing **26**(7), 3142–3155 (2017)

149. Zhao, H., Gallo, O., Frosio, I., Kautz, J.: Loss functions for image restoration with neural networks. IEEE Transactions on Computational Imaging **3**, 47–57 (2017)

150. Zhi Tian Chunhua Shen, H.C., He, T.: Fcos: Fully convolutional one-stage object detection. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), ICCV '19. IEEE Computer Society (2019)

151. Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. 2017 IEEE International Conference on Computer Vision (ICCV) pp. 2242–2251 (2017)