Chapter

Advances in Convolutional Neural Networks

Wen Xu, Jing He, Yanfeng Shu and Hui Zheng

Abstract

Deep Learning, also known as deep representation learning, has dramatically improved the performances on a variety of learning tasks and achieved tremendous successes in the past few years. Specifically, artificial neural networks are mainly studied, which mainly include Multilayer Perceptrons (MLPs), Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). Among these networks, CNNs got the most attention due to the kernel methods with the weight sharing mechanism, and achieved state-of-the-art in many domains, especially computer vision. In this research, we conduct a comprehensive survey related to the recent improvements in CNNs, and we demonstrate these advances from the low level to the high level, including the convolution operations, convolutional layers, architecture design, loss functions, and advanced applications.

Keywords: deep learning, CNNs, kernel methods, weight sharing, comprehensive survey

1. Introduction

Convolutional Neural Networks (CNNs) are specially designed to handle data that consists of multiple arrays/matrixes such as an image composed of three matrixes in RGB channels [1]. The key idea behind CNNs is the convolution operation, which is to use multiple small kernels/filters to extract local features by sliding over the same input. Each kernel can output a feature map and all the feature maps are concatenated together, this is also known as a convolutional layer and it is the core component in a CNN. Note that these concatenated maps can be further processed by the next layer. To reduce the computational cost, the pooling operation such as maximum pooling is usually applied on these feature maps. A typical CNN is usually structured as a series of layers, including multiple convolutional layers and a few of fully connected layers. For example, the famous LeNet [2] consists of two convolutional layers and three fully connected layers, and the pooling operation is used after each convolutional layer.

In addition to building a neural network, a loss function is essential to measure the model performance. Therefore, the process of training a CNN model is transformed into an optimization problem, which normally seeks to minimize the value of the loss function over the training data. Specifically, a gradient-descent based algorithm is usually adopted to iteratively optimize the parameters in a CNN.

Figure 1 shows the high-level abstraction of CNNs in this survey. Specifically, we firstly introduce two types of convolution operations in Section 2. Then four

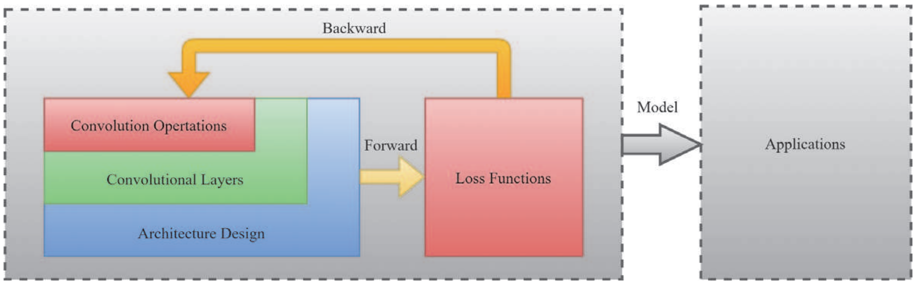

Figure 1.
High-level abstraction of convolutional neural networks in this survey.

methods are summarized for constructing convolutional layers in CNNs in Section 3. In Section 4, we group the current CNN architectures into three types: encoder, encoder-decoder and GANs. Next, we discuss two main types of loss functions in Section 5. In Section 6, we give the advanced applications based on the three types of CNN structures. Finally in Section 7, we conclude this research and give future trends.

2. Convolution operations

The main reason why CNNs are so successful on a variety of problems is that kernels (also known as filters) with fixed numbers of parameters are adopted to handle spacial data such as images. In particular the weight sharing mechanism can help reduce the number of parameters for low computational cost while remaining the spacial invariance properties. In general, there are mainly two types of convolution operations, including basic convolution and transposed convolution.

2.1 Basic convolution and dilated kernels

As shown on the left in **Figure 2**, convolution operation essentially is a linear model for the local spacial input. Specifically, it only performs the sum of element-wise dot products between the local input and the kernels (usually including a bias), and output a value after an activation function. Each kernel slides overall spacial locations in the input with a fixed step. The result is that we can get an 1-channel feature map. Note that there are generally many kernels in one convolutional layer, and all of the output feature maps are concatenated together, e.g., if the number of kernels used in this convolutional layer is D_O, we can get an $O \in \mathcal{R}^{3\times3\times D_O}$ feature map.

While the kernel size of 3 × 3 is widely used in current CNNs, we may need large receptive fields in the input for observing more information during each convolution operation. However, if we directly increase the size of kernels such as

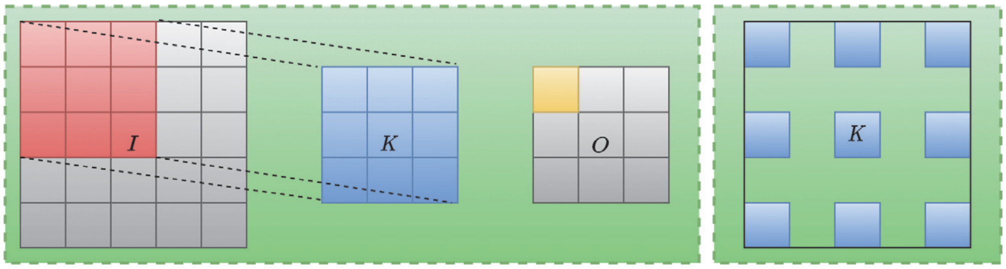

Figure 2.
Left: *A demonstration of basic 2D convolution with a 3 × 3 ×* D_I *kernel (stride = 1, padding = 0),* $I \in \mathcal{R}^{5\times5\times D_I}$ *is the spacial input and* $O \in \mathcal{R}^{3\times3}$ *is the 1-channel output feature map.* ***Right:*** *A dilated kernel for increasing the receptive fields in the input, where the empty space between each element represents 0.*

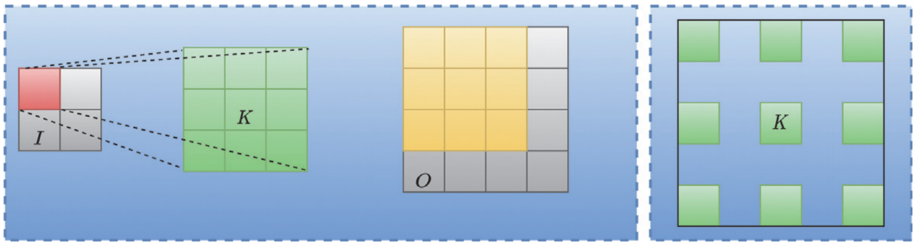

Figure 3.
Left: *A demonstration of transposed 2D convolution with a* $3 \times 3 \times D_I$ *kernel (stride = 1, padding = 0),* $I \in \mathcal{R}^{2\times 2\times D_I}$ *is the spacial input and* $O \in \mathcal{R}^{4\times 4}$ *is the 1-channel output feature map. Note that the receptive fields in O can overlap and we normally sum the values where output overlaps.* ***Right:*** *A dilated kernel for increasing the receptive fields in the input, where the empty space between each element represents 0.*

$K = 9 \times 9 \times D_I$, where D_I is the depth of input, the total number of parameters will increase dramatically and the computational cost will be prohibitive. In practical, as shown on the right in **Figure 2**, we can insert zeros between each element in the kernels and get dilated kernels. For example, dilated kernels have been applied in many tasks such as image segmentation [3], translation tasks [4] and speech recognition [5].

2.2 Transposed convolution and dilated kernels

Normally the size of output feature maps generated from the basic convolution is smaller than the input space (i.e., the dimension of input I is $5 \times 5 \times D_I$ and the dimension of output O is 3×3 in **Figure 2**), which results in high-level abstraction by using multiple convolutional layers. Transposed convolution can be seen as a reverse idea from basic convolution. Its primary purpose is to obtain an output feature map that is bigger than the input space. As shown on the left in **Figure 3**, the size of the input I is $2 \times 2 \times D_I$, after transposed convolution, we can have a 4×4 feature map O. Specifically, during transposed convolution, each output filed in O is just the kernel multiplied by the scalar value of one element in I.

Similarly, we can still use dilated kernels in transposed convolution. The main reason why we need transposed convolution is that it is the fundamental idea to construct a decoder network, which is used to map a latent space into an output image, such as the decoders in U-Net [6] and GANs. Specifically, the transposed convolution is widely used in tasks such as model visualization [7], image segmentation [6], image classification [8] and image super-resolution [9].

3. Convolutional layers

The core components in CNNs are convolutional layers. In the last section, we have demonstrated two types of convolution operations and they are the main idea to construct convolutional layers. In this part, we summarize the main methods in deep learning for building convolutional layers, including basic convolutional layers, convolutional layers with shortcut connection, convolutional layers with mixed kernels and convolutional capsule layers.

3.1 Basic convolutional layers

Recall that there are normally D_O kernels in one convolutional layer, where D_O also denotes the depth of the output feature map. In other words, the number of channels in the output map depends on the number of kernels used in the convolutional layer. More formally, we can denote it as

$$O = \sum_{i=1}^{D_O} I * K_i \tag{1}$$

where $*$ represents the convolution operation which has been addressed above, $\sum$ denotes the concatenation operation and $O \in \mathcal{R}^{W_O H_O D_O}$ is the output feature map. After convolution operation, a no-linear activation function is applied on each element in the concatenated feature map, which can be denoted as

$$O = \sigma(O) \tag{2}$$

While there are many variants related to the activation function, the typical ones which are widely adopted are ReLU $\sigma(x) = \max(0, x)$, tanh $\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$. Note that the non-linear activation functions are essential for building multi-layer networks, as it shows that a two-layer network with enough neurons can uniformly approximate any functions, which is also known as universal approximation theorem [10].

Note that after convolution operation, the width and height of the output feature map $O \in \mathcal{R}^{W_O H_O D_O}$ are usually close to the width and height of the input $I \in \mathcal{R}^{W_I H_I D_I}$. To further reduce the dimensions of the output feature maps for reducing computational cost, the pooling operation is widely used in the current CNNs. Specifically, for 2D pooling operation, two main hyper-parameters are involved: the filter size $F \times F$ and stride S. And after pooling operation, the width of the feature map O is reduced to $W_O = (W_O - F)/S + 1$ and the height of the feature map O is $H_O = (H_O - F)/S + 1$. In brief, we can have

$$O = pool(O) \tag{3}$$

where $pool()$ denotes the pooling operation discussed above. Typical pooling operations includes max-pooling and average-pooling. A general choice to conduct pooling operation is to use $stride = 2$ with 2×2 filter, which means that each 4 pixels in the 2D feature map O will be compressed into 1 pixel. Using a toy example, suppose that there are only four pixels $O = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, then $pool_{max}(O) = [4]$ or $pool_{avg}(O) = [2.5]$.

3.2 Convolutional layers with shortcut connection

It is true that deep neural networks normally can learn better representation from the data than shallow neural networks. However, stacking more layers in a CNN can lead to the problems of vanishing or exploding gradients, which make the networks hard to optimize. A simple and effective way to address this problem is to use shortcut connections, which can help directly transform the information from the previous layer to the current layer in a network.

$$O = \sigma\left(\left(\sum_{i=1}^{D_O} I_{current} * K_i\right) \oplus I_{previous}\right) \tag{4}$$

Note that $\oplus$ can denote two types of operations.

- **Element-Wise Sum:** Each element in $I_{current}$ is added by the corresponding element in $I_{previous}$, which means that the dimensions of $I_{current}$ and $I_{previous}$ must be the same, and the result is that we can get an output O of the same size. This type of operation is well known as **identity shortcut connection** and it is the core idea in ResNet [11, 12]. The main advantage is that it does not add any extra parameters or computational complexity. The disadvantage is due to its inflexible.

- **Concatenation:** We can concatenate the current output and previous input together. Suppose the size of the current output feature map is WHD_O and the size of the previous input is WHD_I, after concatenation, we can have a concatenated feature map O with a size of $WH(D_O + D_I)$. Note that the widths and heights of input and output must be the same. The advantage is that we can remain the information from the previous layers. The disadvantage is that we have to use extra parameters to handle the concatenated feature map O. (i.e., the depth of kernels for processing feature map O is $(D_O + D_I)$.) Specifically, this type of convolutional layers is broadly adopted in networks for image segmentation such as U-Net [6].

3.3 Convolutional layers with mixed kernels

So far we have demonstrated that we normally use many convolutional kernels with the same size in one convolutional layer such as 3 × 3. To enlarge the receptive field, we may adopt the dilated kernels instead. However, it is difficult to know what size of kernels we should use in a CNN. Naturally, we may apply different sizes of kernels in each convolutional layer. E.g., both 1 × 1, 3 × 3 and 5 × 5 kernels are adopted in one convolutional layer. More formally, we define one convolutional layer with mixed kernels as

$$O = \sigma\left(\sum_{i=1}^{D_O^1} I * K_i^{1\times1} + \sum_{i=1}^{D_O^2} I * K_i^{3\times3} + \sum_{i=1}^{D_O^3} I * K_i^{5\times5} + pool(I)\right) \tag{5}$$

where pool(I) denotes the pooling operation such as max-pooling. Therefore, the size of the output feature map is $W_O H_O \left(D_O^1 + D_O^2 + D_O^3 + D_I\right)$.

However, if we directly add different sizes of kernels in one convolutional layer, the computational cost involved will increase sharply. In the inception module [13, 14], a 1 × 1 convolutional layer is applied before 3 × 3 and 5 × 5 convolutional layers in order to reduce the convolutional cost.

3.4 Convolutional capsule layers

> *"The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster."—Geoffrey Hinton.*

In general, pooling operation is essential to reduce the size of output feature maps so that we can obtain high-level abstractions from input by stacking multiple convolutional layers in a CNN. However, the cost is that some information in the feature maps has been abandoned such as conducting max-pooling.

In 2017 [15], Hinton et al. proposed an alluring version of convolutional architectures, which is known as capsule networks, followed by the updated versions in 2018 [16] and 2019 [17]. The convolutional capsule layers in capsule networks are very similar to the traditional convolutional layers. The main difference is that each capsule (i.e., an element in convolutional feature maps) has a weight matrix W_{ij} (i.e., the sizes are 8×16 in [15] and 4×4 in [16] respectively).

4. Architecture design

Although numerous variants of CNN architectures for solving different tasks are proposed from the deep learning community every year, their essential components and over-all structures are very similar. We group the recent classic network structures into three main types, including encoder, encoder-decoder and GANs.

4.1 Encoder

In 1990, LeCun et al. proposed a seminal network called LeNet [2], which help establish the modern CNN structure. Since then, many new methods and compositions are proposed to handle the difficulties encountered in training deep networks for challenging tasks such as objective detection and recognition in computer vision. Some representative works in recent years are AlexNet [18], ZFNet [7], VGGNet [19], GoogleNet [13], ResNet [11], Inception [14]. As mentioned earlier, new methods for constructing convolutional layers in these networks are proposed, e.g., shortcut connection [11] and mixed kernels [14, 20].

In general, the above-mentioned networks can all be regarded as encoders, in which each input such as an image is encoded into a high-level feature representation, as shown on the left in **Figure 4**. And this encoded representation can be further used for, such as image classification, object detection etc. In some literatures, an encoder is also called as a feature extractor. Specifically, the basic convolutional layers are the main components for constructing an encoder network, by stacking multiple layers, each layer in the network can learn high-level abstractions from previous layers [1]. More formally, an encoder network can be written as

$$Z = \mathcal{F}_{encoder}(X; \Theta) \tag{6}$$

where X is the input, Θ is the parameters to learn (e.g., kernels and bias) in the network and Z denotes the encoded representation such as a vector.

4.2 Encoder-decoder

In some specific tasks such as image segmentation [20], our goal is to map an input image to a segmented output image rather than an abstraction. An encoder-decoder structure is specifically designed for solving this type of task. There are

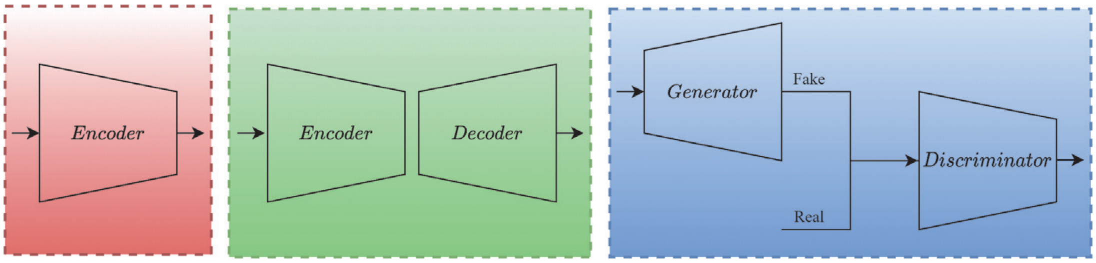

Figure 4.
Left: *An encoder network.* ***Middle:*** *An encoder-decoder network.* ***Right:*** *Generative adversarial networks.*

many possible ways to implement an encoder-decoder structure, and many variants have also been proposed to improve the drawbacks in the last few years. A naive version of encoder-decoder network which was introduced in [20] can be denoted as

$$Z = \mathcal{F}_{encoder}(X; \theta_{encoder}) \tag{7}$$

$$\hat{X} = \mathcal{F}_{decoder}(Z; \theta_{decoder}) \tag{8}$$

where $\mathcal{F}_{encoder}$ denotes an encoder CNN to map an input sample to a representation Z and $\mathcal{F}_{decoder}$ represents a decoder CNN to reconstruct the input sample with Z. Specifically, CNN encoders usually conducts basic convolution operations (i.e., Section 2.1) and CNN decoders perform transposed convolution operations (i.e., Section 2.2).

As shown in the middle in **Figure 4**, an encoder-decoder network is still one complete network and we can train it with an end-to-end method. Note that there are generally many convolutional layers in each coder network, which results that it can be challenging to train a deep encoder-decoder network directly. Recall that the shortcut connection is often adopted to address the problems in deep CNNs. Naturally, we can add connections between the encoder and the decoder. An influential network based on this idea is U-Net [6], which is widely applied in many challenging domains such as medical image segmentation. The above two equations can also be rewritten as a composition of two functions.

$$\hat{X} = \mathcal{F}_{decoder} \circ \mathcal{F}_{encoder}(X; \Theta) = \mathcal{F}_{autoencoder}(X; \Theta) \tag{9}$$

Specifically, in unsupervised learning, an encoder-decoder network is also well known as autoencoder. And there are many variants of autoencoders proposed in recent years, some famous ones including variational autoencoder [21], denoising variational autoencoder [22] and conditional variational autoencoder [23, 24].

4.3 GANs

Since generative adversarial networks were firstly proposed by Goodfellow et al. [25] in 2014, this type of architectures for playing two-player minimax game has been most extensively studied. Partly because it is an unsupervised learning method and we can obtain a fancy generator network which can help generate fake examples from a latent space (i.e., a vector with some random noise). On the right in **Figure 4** shows the basic structure of GANs, in which a generator network can map some input noise into a fake example and make it look as real as possible and a discriminator network always tries to identify the fake sample from its input. By iteratively training the two players, they can both improve their methods. More formally, we can have

$$\hat{Y} = \mathcal{D}(\mathcal{G}(L; \theta_{\mathcal{G}}), X_{real}, \theta_{\mathcal{D}}) \tag{10}$$

where $\mathcal{G}$ denotes the generator function and $\mathcal{D}$ represents the discriminator function. L is the latent space input in the generator, and its output is a fake example. X_{real} is the real samples we have collected. And $\hat{Y} \in [0, 1]$ is the predicted result of the discriminator to show whether the input is real or fake.

As shown in **Table 1**, numerous variants of GANs architectures can be found in the recently published literatures and we broadly summarize these representative networks according to their published time. Note that the fundamental methods behind these architectures are very similar.

Name	Year	Summary
GANs [25]	2014	The original version of GANs, where $\mathcal{G}$ and $\mathcal{D}$ are implemented with fully connected neural networks.
Conditional GANs [26]	2014	Labels are included in $\mathcal{G}$ and $\mathcal{D}$.
Laplacian Pyramid GANs [27]	2015	CNNs with the laplacian pyramid method.
Deep Convolutional GANs [28]	2015	Transposed convolutional layers are used to construct $\mathcal{G}$.
Bidirectional GANs [29]	2016	An extra encoder was adopted based on the traditional GANs.
Semi-supervised GANs [30]	2016	The $\mathcal{D}$ can also classify the real samples while distinguishing the real and fake.
InfoGANs [31]	2016	An extra classifier was added into the GANs.
Energy-based GANs [32]	2016	The $\mathcal{D}$ was replaced with an encoder-decoder network.
Auxiliary Classifier GANs [33]	2017	An auxiliary classifier was used in the $\mathcal{D}$.
Progressive GANs [34]	2017	Progressive steps are adopted to explain the networks.
BigGANs [35]	2018	A large GANs with self-attention module and hinge loss.
Self-attention GANs [36]	2019	The self-attention mechanism is proposed to build $\mathcal{G}$ and $\mathcal{D}$.
Label-noise Robust GANs [37]	2019	A noise transition model is included in $\mathcal{D}$.
AutoGANs [38]	2019	The neural architecture search algorithm is used to obtain $\mathcal{G}$ and $\mathcal{D}$.
Your Local GANs [39]	2020	A new local sparse attention layer was proposed.
MSG-GANs [40]	2020	There are connections from $\mathcal{G}$ to $\mathcal{D}$.

Table 1.
Representative architectures of GANs in recent years.

5. Loss functions

Before introducing the loss functions, we need to understand that the ultimate goal to train a neural network $\mathcal{F}(X;\Theta)$ is to find a suitable set of parameters Θ so that our model can achieve good performance on the unseen samples (i.e., test dataset). The typical way to search Θ in machine learning is to use loss functions as a criterion during training. In other words, training neural networks is equivalent to optimizing the loss functions by back-propagation. Accurately, a loss function outputs a scalar value which is regarded as a criterion for measuring the difference between the predicted result and the true label over one sample. And during training, our goal is to minimize the scalar value over m training samples (i.e., cost function). Therefore, as shown in **Figure 1**, loss functions play a significant role in constructing CNNs.

$$\mathcal{J} = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}_i \tag{11}$$

where $\mathcal{L}_i$ denotes a loss function for the training sample i, and $\mathcal{J}$ is often known as cost function, which is just the mean of the sum of the losses over m training samples (i.e., usually a batch of m training samples is fed into a CNN during each iteration of training).

Note that there are numerous variants of loss functions used in the deep learning literature. However, the fundamental theories behind them are very similar. We group them into two categories, namely Divergence Loss Functions and Margin Loss Functions. And we also introduce six typical and classic loss functions that are commonly used for training neural networks.

5.1 Divergence loss functions

Divergence loss functions denote a family of loss functions based on computing the divergences between the predicted results and true labels, mainly including Kullback-Leibler Divergence, Log Loss, Mean Squared Error.

5.1.1 Kullback-Leibler divergence

Before introducing the Kullback–Leibler divergence, we need to understand that the fundamental goal of deep learning is to learn a data distribution Q over the training dataset so that Q is close to the true data distribution P. Back in 1951, Kullback-Leibler divergence was proposed to measure the difference between two distributions on the same probability space [41]. It is defined as

$$\begin{aligned} D_{KL}(P\|Q) &= \sum_X P(X) log P(X) - \sum_X P(X) log Q(X) \\ &= \sum_X P(X)\, log\, \frac{P(X)}{Q(X)} \end{aligned} \tag{12}$$

where $D_{KL}(P\|Q)$ denotes the Kullback–Leibler divergence from Q to P. $\sum_X P(X) log P(X)$ is the entropy of P and $\sum_X P(X) log Q(X)$ is the cross entropy of P and Q. There is also a symmetrized form of the Kullback–Leibler divergence, which is known as the Jensen–Shannon divergence. It is a measure of the similarity between P and Q.

$$JSD(P\|Q) = \frac{1}{2} D_{KL}\left(P\|\frac{P+Q}{2}\right) + \frac{1}{2} D_{KL}\left(Q\|\frac{P+Q}{2}\right) \tag{13}$$

Specifically, $JSD(P\|Q) = 0$ means the two distributions are the same. Therefore, if we minimize the Jensen-Shannon divergence, we can make the distribution Q close to the distribution P. More Specifically, if Q denotes the distribution on data, and P represents the distribution which is learned by a CNN model. By minimizing the divergence, we can learn a model which is close to the true data distribution. This is the main idea of GANs. The loss function of GANs is defined as

$$\min_{\mathcal{G}} \quad \max_{\mathcal{D}} : \quad \mathbb{E}_{X\sim P(X)}\, log\, \mathcal{D}(X;\Theta_{\mathcal{D}}) + \mathbb{E}_{L\sim Q(L)}\, log\, (1 - \mathcal{D}(\mathcal{G}(L;\Theta_{\mathcal{G}});\Theta_{\mathcal{D}})) \tag{14}$$

where $\mathcal{G}$ denotes the generator and $\mathcal{D}$ denotes the discriminator. And our goal is to try to make $Q(\mathcal{G}(L))$ close to $P(X)$. In other words, when the generative distribution of fake examples is close to the distribution of real samples, the discriminator cannot distinguish between the fake and the real.

5.1.2 Log loss

Log loss is widely used in the current deep neural networks due to its simplicity and power. The binary log loss function is defined as

$$\mathcal{L}_{binary} = -Ylog\left(\hat{Y}\right) - (1 - Y)\,log\,\left(1 - \hat{Y}\right) \quad (Y \in [0, 1]) \tag{15}$$

where $Y \in [0, 1]$ denotes the binary label for a sample and $\hat{Y}$ is the predicted result, (i.e., given a training sample with its corresponding label $\{X, Y\}$, we can have an output predicted result with an encoder network $\hat{Y} = \mathcal{F}_{encoder}(X, \Theta)$.)

When the learning task is multi-class classification, each sample label is normally encoded with the one-hot-encoding format, which can be denoted as $Y = \left[y_1, y_2, \dots, y_{nclass}\right]^{\mathrm{T}}$, i.e., if the label is 3, then only $y_3 = 1$ and the others are all given the value of 0. Therefore, the log loss for one sample can be written as

$$\mathcal{L}_{log} = -\sum_{i=1}^{nclass} 1\left\{y_i = 1\right\}\,log\,\left(\hat{y}_i\right) \tag{16}$$

where $\hat{y}_i$ is the predicted result for the true label y_i. $1\left\{y_i = 1\right\}$ denotes the indicator function, which means that its output is 1 if $y_i = 1$, otherwise it outputs 0.

We may wonder why the log loss is a reasonable choice. Informally, let Y denotes the data distribution and $\hat{Y}$ denotes the distribution leaned by our model, then based on Kullback–Leibler divergence, we can have

$$D_{KL}\left(Y \| \hat{Y}\right) = \sum YlogY - \sum Ylog\hat{Y} \tag{17}$$

And our goal is to minimize the divergence between Y and $\hat{Y}$ so that the distribution obtained by our model is close to the true data distribution. Because the term $\sum YlogY$ is the entropy related to data, and we only need to optimize the cross entropy term $-\sum Ylog\hat{Y}$. Therefore, log loss is also well known as cross-entropy loss.

5.1.3 Mean squared error

Probably the mean squared error is one of the most familiar loss functions as it is really like the least square loss function. It directly calculates the difference between the predicted result and the true label, which is denoted as

$$\mathcal{L}_{mean} = -\frac{1}{2}\left(Y - \hat{Y}\right)^2 \tag{18}$$

One example which can help us deeply understand the mean squared error is that minimize the mean squared loss of a linear regression model is equivalent to maximum likelihood. In other words, this is a method to optimize the parameters of our model so that the distribution learned by our model is most probable under the observed training data. Therefore, the fundamental goal is still the same as above, which is to make the model distribution and the data distribution as close as possible.

5.2 Margin loss functions

Margin loss functions represent a family of margin maximizing loss functions. The typical functions include Hinge Loss, Contrastive Loss and Triplet Loss. Unlike the divergence loss functions, margin loss functions calculate the relative distances between outputs and they are more flexible in terms of training data.

5.2.1 Hinge loss

Hinge loss is well known to train Support Vector Machine classifiers. Specifically, there are two main types of hinge losses. The first type is for each sample with only one correct label, it is denoted as

$$\mathcal{L}_{hinge} = \sum_{i \neq k}^{nclass} \max\left(0, \Delta + \hat{y}_i - \hat{y}_k\right)^p \quad \left(y_k = 1, \sum_{i \neq k}^{nclass} y_i = 0\right) \tag{19}$$

where y_i denotes each element in the one-hot-encoding label, y_k is the correct class. $\hat{y}_i$ represents the predicted result of our neural network for each class. $\Delta = 1$ is the standard choice for the margin. If $p = 1$, the above loss denotes the standard Hinge loss, and if $p = 2$, it is the Squared Hinge loss.

However, in real tasks such as attribute classification, each samples can have multiple correct labels. e.g., a photo posted on Facebook may include a set of hashtags. Therefore, the second type for multiple labels is

$$\mathcal{L}_{hinge} = \sum_{i=1}^{nclass} \max\left(0, \Delta - \delta(y_i = 1)\hat{y}_i\right)^p \tag{20}$$

where $\delta(y_i = 1) = +1$ if $y_i = 1$, otherwise $\delta(y_i = 1) = -1$. $\Delta = 1$ is still the common choice for the margin and $p = 1$ or $p = 2$.

5.2.2 Contrastive loss

Contrastive loss is specially designed for measuring the similarity of a pair of training samples. Considering two pairs of samples $\{X_a, X_p\}$ and $\{X_a, X_n\}$, where X_a is known as an anchor sample and X_p denotes the positive sample and X_n represents the negative sample, Specifically, if the pair $\{X_a, X_p\}$ is matching, then the loss for the pair is the distance between their outputs from the network $d(Z_a, Z_p)$. While if the pair $\{X_a, X_n\}$ is not matching and the distance of their outputs from the model is small than the pre-defined margin $(0, \Delta - d(Z_a, Z_n)) > 0$, then we need also to calculate the loss. Formally, we can have

$$\mathcal{L}_{contrastive} = \begin{pmatrix} d(Z_a, Z_p) & \text{if matched pair} \\ \max\left(0, \Delta - d(Z_a, Z_n)\right) & \text{if unmatched pair} \end{pmatrix} \tag{21}$$

where d can be the Euclidean distance, (i.e., $d(Z_a, Z_p)) = \|Z_a - Z_n\|_2)$. Alternatively, the above equation can be rewritten as

$$\mathcal{L}_{contrastive} = yd(Z_a, Z_p) + (1 - y)\max\left(0, \Delta - d(Z_a, Z_n)\right) \tag{22}$$

where $y = 1$ if the given pair is matching, otherwise $y = 0$. Δ is the margin which can affect the loss calculating for the unmatched pairs.

5.2.3 Triplet loss

Triplet loss looks similar to the contrastive loss, but it is a measure of the difference between the matched pair and the unmatched pair. Considering three samples $\{X_a, X_p, X_n\}$, the Triplet loss is denoted as

$$\mathcal{L}_{triplet} = \max\left(0, \Delta + d\left(Z_a, Z_p\right) - d\left(Z_a, Z_n\right)\right) \tag{23}$$

Note that minimize the loss function is equivalent to minimizing the distances of matched pairs and maximizing the distances of unmatched pairs.

6. Advanced applications

One of the most exciting areas in deep learning is that we can apply neural networks to a numerous number of applications that cannot be solved well or be handled by the traditional machine learning method. In this section, we summarize the typical advances that CNNs has achieved based on the three types of CNN structures.

6.1 Applications with encoders

6.1.1 Image classification

A basic task in machine learning is classification, which is the problem of identifying to which of a list of labels a new sample belongs, such as the well-known CIFAR-10 dataset, in which there are 10 categories of images and the goal is to train a model for correctly classifying an unseen image based on observing the training dataset. In particular, CNNs have made many breakthroughs on large scale image datasets such as the ImageNet challenge [18]. As mentioned in Section 4.1, the classic encoders such as AlexNet [18], ZFNet [7], VGGNet [19], GoogleNet [13], ResNet [11], Inception [14] are regarded as the milestones in the past few years. The successes of these encoders are all based on supervised learning, which means that manual labelling is essential for the dataset such as the ImageNet dataset [42]. Specifically, a labeled dataset is normally divided into training and test dataset (may also include a validation dataset), and our goal is to achieve good performance on the test dataset after training a neural network with the training dataset, and the pre-trained model can be further used for classifying new images that are from the same data distribution space.

Classification can also be treated as a fundamental problem in machine learning, the successes of these encoders on image classification also help establish the foundation for many other applications. Specifically, we can utilize an encoder to extract high-level representation from the low-level input image, and the obtained representation can be further used for many other applications.

6.1.2 Object detection

In addition to image classification, object detection is also very important in computer vision. Image classification gives us the answer to what a given image is, and object detection is about telling us the specific positions of objects in an image. Specifically, the goal is to train an encoder to output a suitable bounding box and associated class probabilities for each object in a given image. Two typical methods are widely used in the current computer vision, including YOLO [43] and SSD [44]. The core idea of YOLO is that object detection is treated as an regression problem, which means that each image is divided into multiple grids and each grid cell outputs a pre-defined number of bounding boxes, the corresponding confidence for each box and class probabilities [43]. Since the first version of YOLO was proposed, the updated versions have also been proposed. SSD is a more simple method, which utilizes a set of default boxes with different aspect ratios, and each box outputs the shape offsets and the class confidences [44].

6.1.3 Pose estimation

The multiple levels of representations learned in the multiple layers of CNNs can also be used for solving the task of human-body pose estimation. Specifically, there are mainly two types of approaches, including regression of body joint coordinates and heat-map for each body part. In 2014, a framework called DeepPose [45] was introduced to learn pose estimation by a deep CNN, in which estimating human-body pose is equivalent to regressing the body joint coordinates. There are also some extension works based on this method, such as a process called iterative error feedback [46], which encompasses both the input and output spaces of CNN for enhancing the performance. In 2014, Tompson et al. [47] propose a hybrid architecture which consists of a CNN and a Markov Random Field, in particular the output of the CNN for an input image is a heat-map. Some recent works based on the heat-map method such as [48], in which a multi-context attention mechanism was proposed to incorporate with CNNs.

6.2 Applications with encoder-decoders

6.2.1 Image restoration

The operation of image restoration is to recover a damaged or corrupt image for the clean image such as image denoising and super-resolution. Therefore, a natural way to implement this idea is to utilize a pre-trained encoder-decoder network, where the encoder can map a noise image into a high-level representation, and the decoder can transform the representation into an original image. For example, Mao et al. [49] apply a deep convolutional encoder-decoder network for image restoration, in particular the shortcut connection method is adopted between the encoder and decoder, which has been demonstrated in Section 3.2. And the transposed convolution is used for constructing the decoder network, as mentioned in Section 2.2. Similar work in [50] has also been introduced for image restoration, in which a residual method is used in the network (i.e., in Section 3.2).

6.2.2 Image segmentation

The task of image segmentation is to map an input image into a segmented output image. The encoder-decoder networks have been developed dramatically in recent years and achieve a significant impact on computer vision. Specifically, there are mainly two types of tasks including semantic segmentation and instance segmentation. In 2015, Long et al. [20] firstly showed that an end-to-end fully CNN can achieve state-of-art in image segmentation tasks. Similar work has also been introduced in [6] in 2015, in which a U-Net architecture is proposed for medical image segmentation, and the main advance in this architecture is that the shortcut connection method is also used between the encoder and decoder network. Since then, a series of papers based on these two methods have been published. In particular nowadays the U-Net based architectures are widely used for the medical image diagnosis.

6.2.3 Image captioning

One of the exciting applications achieved by CNNs is image captioning, which is to describe the content of an input image with natural language. The basic idea is as follows: Firstly, a pre-trained CNN encoder is used to extract some high-level features from an input image. Secondly, these features are typically fed into an recurrent neural network for generating a sentence. For example, Li et al. [51] proposed a fully convolutional localization network for extracting representation from images and the decoder for generating captions is LSTM. Recently, attention mechanism has been widely used for sequence processing and achieved significant improvements such as machine translation, Huang et al. [52] introduce an encoder-decoder framework, where an attention module is used in the encoder and decoder respectively. Specifically, the encoder is a CNN based network.

6.2.4 Speech processing

Note that speech signals exhibit spectral variations and correlations, CNNs are very suitable to reduce them. Therefore, CNNs can also be utilized for the task of speech processing, such as speech recognition. Sainath1 et al. [53] applied deep CNNs for large vocabulary speech tasks. In [54–56], the CNNs are used for speech recognition. And the fundamental methods are very similar, both of them use the CNNs to extract features from the raw input, and then these features are fed into an decoder for the specific learning tasks.

6.3 Applications with GANs

6.3.1 Image generation

The most typical application of GANs is to generate fake examples. Recall that there normally are two dependent networks in GANs, including $\mathcal{G}$ and $\mathcal{D}$. Once the training process is finished, we can utilize $\mathcal{G}$ to generate fake samples from the training dataset.

Generating fake samples can be regarded as data augmentation, which means that these fake data can be further used to train models. Note that deep learning is also well known as a data-driven approach. In particular most of the advances that deep neural networks achieved are based on supervised learning. Specifically, the current successful neural network models usually consist of millions of parameters. And annotated data is essential to optimize these parameters for guaranteeing the model accuracy when conducting supervised learning. However, manually labeling data is time-consuming and expensive, especially in some specific domains such as medicine. Even more severe is that it can be hard to collect enough data due to the privacy concerns. There are numerous works to utilize GANs for enhancing model performance. E.g., in [57], a semi-supervised framework based on GANs is applied to semantic segmentation in order to address the lack of annotations. [58] is a work of utilizing synthetic medical images for enhancing the performance of liver lesion classification.

Despite the successes of GANs, generating high-resolution, diverse samples is still a challenging task. In [35], they introduce the progressive GANs which can generate high-resolution human faces. Another impressive work to generate realistic photographs is BigGANs [36].

6.3.2 Image translation

Another interesting application derived from GANs is image translation. While there are many specific applications, we summarize them into three categories, including translation of image to image, translation of text to image and translation of image to super-resolution.

Image to Image: The task of image-to-image translation is to learn a mapping $\mathcal{G}(X) \rightarrow Y$. E.g., Isola et al. [59] apply conditional GANs for an image-to-image task and achieve impressive results such as mapping sketches to photographs, black-white photographs to color etc. Another typical work is the CycleGANs [60], which can transfer a style of an image into another.

Text to Image: One of the interesting works from GANs is to synthesis a realistic image based on some text descriptions. E.g., "There is a little bird with red feather." Some representative works include: Reed et al. [61] introduce a text-conditional convolutional GANs. Zhang et al. [62] apply a StackGANs to synthesize high-quality images from text.

Super Resolution: The task of super-resolution is to map a low-resolution image to a high-resolution image. In 2017, ledig et al. [63] propose a framework named as SRGAN, which is regarded as the first work that has the ability to generate photo-realistic images for 4X upscaling factors. Specifically, the loss functions used in their framework consist of an adversarial loss and a content loss. In particular the content loss can help remain the original content from the input images.

6.3.3 Image editing

Image editing is regarded as a fundamental problem in computer vision. The emergence of GANs has also brought new chances for this task. In the past few years, GANs have been developed for image editing, such as image inpainting and image matting.

Image inpainting: The task of image inpainting is to recover an arbitrary damaged region in an image. Specifically, we can utilize the algorithm to learn the content and style of the image and generate the damaged part based on the input image, such as [64], in which they introduce a context encoder for natural image inpainting. And in [65, 66], their works mainly focus on human face completion.

Image matting: The goal of image matting is to separate the foreground object from the background in an image. This technique can be used for a wide range of applications such as photo editing and video post-production. And there are also some representative works such as [67, 68].

7. Summary and future trends

In this research, we have conducted a hierarchically-structured survey of the main components in CNNs from the low level to the high level, namely, convolution operations, convolutional layers, architecture design, loss functions. In addition to introducing the recent advances of these aspects in CNNs, we have also discussed the advanced applications based on the three types of architectures including encoder, encoder-decoder and GANs, from which we can see that CNNs have made numerous breakthroughs and achieved state-of-the-art in computer vision, natural language processing and speech recognition, especially these fantastic results based on GANs.

From the above analyses, we can summarize that the current development tendencies in CNNs mainly focus on designing new architectures and loss functions. Because these two aspects are the core parts when applying CNNs into various types of tasks. On the other hand, the fundamental ideas behind these various applications are very similar, as summarized above.

However, there are still many disadvantages in the current deep learning. The first problem is the requirement of large-scale datasets, in particular constructing a labeled dataset is very time-consuming and expensive such as in the medical domain. Therefore, we need to pay much more attention to semi-supervised learning and unsupervised learning. The second disadvantage is the high computational cost related to training deep CNNs, as the current standard CNN structures become deeper and deeper and they usually consists of millions of parameters. The third issue is that applying CNNs into tasks is not an easy job and it usually requires professional skills and experiences, because training a network involves a lot of hyper-parameters to tune, such as the number of kernels in each layer, the size of kernels, the total number of layers, learning rate etc.

Future work should focus on deep learning theory as the solid theory for supporting the current neural models is lacking. Unlike other machine learning algorithms such as support vector machines that have obvious mathematical logic, it is usually very hard to totally understand why a deep network can achieve such an excellent performance on a task. Therefore, based on the current developments of deep learning, we give three trends on which we need to work in the future: Neural Topologies such as the graph neural networks, Uncertainty Estimation such as Bayesian neural networks and Privacy Preservation.

Acknowledgements

This work is supported by China Scholarship Council and Data61 from CSIRO, Australia.

Conflict of interest

The authors declare no conflict of interest.

Author details

Wen Xu[1,2], Jing He[1*], Yanfeng Shu[2] and Hui Zheng[1]

1 Swinburne University of Technology, Australia

2 Data61, CSIRO, Australia

*Address all correspondence to: jinghe@swin.edu.au

References

[1] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;**521**(7553): 436-444

[2] LeCun Y, Boser BE, Denker JS, Henderson D, Howard RE, Hubbard WE, et al. Handwritten digit recognition with a back-propagation network. In: Advances in Neural Information Processing Systems. 1990. pp. 396-404

[3] Yu F, Koltun V. Multi-Scale Context Aggregation by Dilated Convolutions. arXiv preprint arXiv:1511.07122. November 23, 2015

[4] Kalchbrenner N, Espeholt L, Simonyan K, Oord AV, Graves A, Kavukcuoglu K. Neural Machine Translation in Linear Time. arXiv preprint arXiv:1610.10099. October 31, 2016

[5] Sercu T, Goel V. Dense prediction on sequences with time-dilated convolutions for speech recognition. arXiv preprint arXiv:1611.09288. November 28, 2016

[6] Ronneberger O, Fischer P, Brox T. U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention. Cham: Springer; 2015. pp. 234-241

[7] Zeiler MD, Fergus R. Visualizing and understanding convolutional networks. In: European Conference on Computer Vision. Cham: Springer; 2014. pp. 818-833

[8] Zhang Y, Lee K, Lee H. Augmenting supervised neural networks with unsupervised objectives for large-scale image classification. In: International Conference on Machine Learning. 2016. pp. 612-621

[9] Dong C, Loy CC, He K, Tang X. Image super-resolution using deep convolutional networks. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2015;**38**(2):295-307

[10] Cybenko G. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems. 1992;**5**(4):455

[11] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. pp. 770-778

[12] He K, Zhang X, Ren S, Sun J. Identity mappings in deep residual networks. In: European Conference on Computer Vision. Cham: Springer; 2016. pp. 630-645

[13] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al. Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015. pp. 1-9

[14] Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016. pp. 2818-2826

[15] Sabour S, Frosst N, Hinton GE. Dynamic routing between capsules. In: Advances in Neural Information Processing Systems. 2017. pp. 3856-3866

[16] Hinton GE, Sabour S, Frosst N. Matrix capsules with EM routing. In: International Conference on Learning Representations. 2018

[17] Kosiorek A, Sabour S, Teh YW, Hinton GE. Stacked capsule autoencoders. In: Advances in Neural Information Processing Systems. 2019. pp. 15512-15522

[18] Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, et al. Imagenet large scale visual recognition challenge. International Journal of Computer Vision. 2015;**115**(3):211-252

[19] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556. September 4, 2014

[20] Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015. pp. 3431-3440

[21] Kingma DP, Welling M. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114. December 20, 2013

[22] Im DI, Ahn S, Memisevic R, Bengio Y. Denoising criterion for variational auto-encoding framework. In: Thirty-First AAAI Conference on Artificial Intelligence. 2017

[23] Kingma DP, Mohamed S, Rezende DJ, Welling M. Semi-supervised learning with deep generative models. In: Advances in Neural Information Processing Systems. 2014. pp. 3581-3589

[24] Sohn K, Lee H, Yan X. Learning structured output representation using deep conditional generative models. In: Advances in Neural Information Processing Systems. 2015. pp. 3483-3491

[25] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, et al. Generative adversarial nets. In: Advances in neural information processing systems. 2014. pp. 2672-2680

[26] Mirza M, Osindero S. Conditional Generative Adversarial Nets. arXiv preprint arXiv:1411.1784. November 6, 2014

[27] Denton EL, Chintala S, Fergus R. Deep generative image models using a laplacian pyramid of adversarial networks. In: Advances in Neural Information Processing Systems. 2015. pp. 1486-1494

[28] Radford A, Metz L, Chintala S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv preprint arXiv:1511.06434. November 19, 2015

[29] Donahue J, Krähenbühl P, Darrell T. Adversarial Feature Learning. arXiv preprint arXiv:1605.09782. May 31, 2016

[30] Odena A. Semi-Supervised Learning with Generative Adversarial Networks. arXiv preprint arXiv:1606.01583. June 5, 2016

[31] Chen X, Duan Y, Houthooft R, Schulman J, Sutskever I, Abbeel P. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In: Advances in Neural Information Processing Systems. 2016. pp. 2172-2180

[32] Zhao J, Mathieu M, LeCun Y. Energy-Based Generative Adversarial Network. arXiv preprint arXiv: 1609.03126. September 11, 2016

[33] Odena A, Olah C, Shlens J. Conditional image synthesis with auxiliary classifier gans. In: International Conference on Machine Learning. 2017. pp. 2642-2651

[34] Karras T, Aila T, Laine S, Lehtinen J. Progressive Growing of Gans for Improved Quality, Stability, and Variation. arXiv preprint arXiv: 1710.10196. October 27, 2017

[35] Brock A, Donahue J, Simonyan K. Large Scale Gan Training for High Fidelity Natural Image Synthesis. arXiv preprint arXiv:1809.11096. September 28, 2018

[36] Zhang H, Goodfellow I, Metaxas D, Odena A. Self-attention generative adversarial networks. In: International Conference on Machine Learning. 2019. pp. 7354-7363

[37] Kaneko T, Ushiku Y, Harada T. Label-noise robust generative adversarial networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019. pp. 2467-2476

[38] Gong X, Chang S, Jiang Y, Wang Z. Autogan: Neural architecture search for generative adversarial networks. In: Proceedings of the IEEE International Conference on Computer Vision. 2019. pp. 3224-3234

[39] Daras G, Odena A, Zhang H, Dimakis AG. Your local GAN: Designing two dimensional local attention mechanisms for generative models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020. pp. 14531-14539

[40] Karnewar A, Wang O. Msg-gan: Multi-scale gradients for generative adversarial networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020. pp. 7799-7808

[41] Kullback S, Leibler RA. On information and sufficiency. The annals of mathematical statistics. 1951;**22**(1): 79-86

[42] Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L. Imagenet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. IEEE; 2009. pp. 248-255

[43] Redmon J, Divvala S, Girshick R, Farhadi A. You only look once: Unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016. pp. 779-788

[44] Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, et al. Ssd: Single shot multibox detector. In: European Conference on Computer Vision. Cham: Springer; 2016. pp. 21-37

[45] Toshev A, Szegedy C. Deeppose: Human pose estimation via deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2014. pp. 1653-1660

[46] Carreira J, Agrawal P, Fragkiadaki K, Malik J. Human pose estimation with iterative error feedback. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016. pp. 4733-4742

[47] Tompson JJ, Jain A, LeCun Y, Bregler C. Joint training of a convolutional network and a graphical model for human pose estimation. In: Advances in Neural Information Processing Systems. 2014. pp. 1799-1807

[48] Chu X, Yang W, Ouyang W, Ma C, Yuille AL, Wang X. Multi-context attention for human pose estimation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017. pp. 1831-1840

[49] Mao X, Shen C, Yang YB. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. In: Advances in Neural Information Processing Systems. 2016. pp. 2802-2810

[50] Zhang Y, Tian Y, Kong Y, Zhong B, Fu Y. Residual dense network for image restoration. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2020

[51] Johnson J, Karpathy A, Fei-Fei L. Densecap: Fully convolutional localization networks for dense captioning. In: Proceedings of the IEEE Conference on Computer Vision and

Pattern Recognition. 2016. pp. 4565-4574

[52] Huang L, Wang W, Chen J, Wei XY. Attention on attention for image captioning. In: Proceedings of the IEEE International Conference on Computer Vision. 2019. pp. 4634-4643

[53] Sainath TN, Mohamed AR, Kingsbury B, Ramabhadran B. Deep convolutional neural networks for LVCSR. In: 2013 IEEE international conference on acoustics, speech and signal processing. IEEE; 2013. pp. 8614-8618

[54] Abdel-Hamid O, Mohamed AR, Jiang H, Deng L, Penn G, Yu D. Convolutional neural networks for speech recognition. IEEE/ACM Transactions on audio, speech, and language processing. 2014;**22**(10): 1533-1545

[55] Amodei D, Ananthanarayanan S, Anubhai R, Bai J, Battenberg E, Case C, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In: International Conference on Machine Learning. 2016. pp. 173-182

[56] Zhang Y, Pezeshki M, Brakel P, Zhang S, Bengio CL, Courville A. Towards End-to-End Speech Recognition with Deep Convolutional Neural Networks. arXiv preprint arXiv: 1701.02720. January 10, 2017

[57] Souly N, Spampinato C, Shah M. Semi supervised semantic segmentation using generative adversarial network. In: Proceedings of the IEEE International Conference on Computer Vision. 2017. pp. 5688-5696

[58] Frid-Adar M, Diamant I, Klang E, Amitai M, Goldberger J, Greenspan H. GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification. Neurocomputing. 2018; **321**:321-331

[59] Isola P, Zhu JY, Zhou T, Efros AA. Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017. pp. 1125-1134

[60] Zhu JY, Park T, Isola P, Efros AA. Unpaired image-to-image translation using cycle-consistent adversarial networks. In: Proceedings of the IEEE International Conference on Computer Vision. 2017. pp. 2223-2232

[61] Reed S, Akata Z, Yan X, Logeswaran L, Schiele B, Lee H. Generative Adversarial Text to Image Synthesis. arXiv preprint arXiv: 1605.05396. May 17, 2016

[62] Zhang H, Xu T, Li H, Zhang S, Wang X, Huang X, et al. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In: Proceedings of the IEEE International Conference on Computer Vision. 2017. pp. 5907-5915

[63] Ledig C, Theis L, Huszár F, Caballero J, Cunningham A, Acosta A, et al. Photo-realistic single image super-resolution using a generative adversarial network. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017. pp. 4681-4690

[64] Pathak D, Krahenbuhl P, Donahue J, Darrell T, Efros AA. Context encoders: Feature learning by inpainting. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016. pp. 2536-2544

[65] Yeh RA, Chen C, Yian Lim T, Schwing AG, Hasegawa-Johnson M, Do MN. Semantic image inpainting with deep generative models. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017. pp. 5485-5493

[66] Li Y, Liu S, Yang J, Yang MH. Generative face completion. In:

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017. pp. 3911-3919

[67] Xu N, Price B, Cohen S, Huang T. Deep image matting. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017. pp. 2970-2979

[68] Lutz S, Amplianitis K, Smolic A. Alphagan: Generative Adversarial Networks for Natural Image Matting. arXiv preprint arXiv:1807.10088. July 26, 2018