Chapter 6
Convolutional Neural Networks (CNN)

Convolutional neural networks (CNN or ConvNet) are a specific type of neural networks for processing grid-like data such as images and time series. In healthcare applications, the CNN models are widely used in automatic feature learning and disease classification from medical images, for example, automatic classification of skin lesions [45], detection of diabetic retinopathy [61], and COVID X-ray classification [120]. In addition, CNN demonstrated great performance in disease detection using biosignals such as Electrocardiography (ECG) [155] and electroencephalogram (EEG) [7]. More recently, researchers also applied CNN models on structured EHR data [18, 177] and clinical text [3, 114].

6.1 CNN Intuition

CNN are neural networks comprised of layers of **convolutions** often with additional nonlinear activation and **pooling** layers, followed by fully connected layers. Unlike DNN models that connect each neuron to all neurons in the next layer, CNN models focus on local properties. In particular, they use convolutions over the input layer to create **local connections** such that each region in the input is connected to a unit in the next layer. The convolution outputs are called **feature maps**, which will be discussed in more detail in the next section. After convolution, the **pooling** layers progressively reduce the size of the feature map.

The design of convolution and pooling layers leverages the local properties of grid-like data (e.g., images or time-series signals), namely **translational invariance** and **compositionality**.

- Translational invariance means statistics of one local region (e.g., a section in the image or a fragment of ECG series) are similar to the other local regions. Thus features learned from a local region can be used in all regions.

C. Xiao, J. Sun, *Introduction to Deep Learning for Healthcare*,
https://doi.org/10.1007/978-3-030-82184-5_6

- Compositionality refers to the intrinsic hierarchical structure within image or signal data: lower-level features such as a pixel in an image or a beat in ECG signals can be composed into higher-level representation such as a scene or a rhythm pattern.

CNNs often repeat convolution and pooling layers multiple times and later employ fully connected layers to fuse all local features to produce more abstract features for classification tasks. Different layers can be stacked as deep networks.

6.2 Architecture of CNN

This section describes the components of the CNN architecture, namely, convolution layers, pooling layers, and fully connected layers. The related notations are defined in Table 6.1.

6.2.1 Convolution Layer: 1D

Intuition A convolution layer is the building block that distinguishes CNNs from other deep learning models. For a 1D convolution, the idea is to extract various local patterns over 1D signals such as EEG. The way to do that is to define those patterns with *filters* (also called kernels) and then to repeatedly apply those filters over the signals to generate features called a *feature map*. Then multiple convolution operations can be applied in sequence to generate more and more sophisticated features.

Notations The input and output of a convolution layer are called *feature maps*, which are denoted by $\boldsymbol{F}^{(l)}$. In particular, the feature map from layer-l $\boldsymbol{F}^{(l)}$ will be the input to the convolution operation at layer-l to generate the output feature map $\boldsymbol{F}^{(l+1)}$. The convolution operation itself is specified by a set of m *filters* each

Table 6.1 Notations for convolutional neural networks

Notation	Definition
$\boldsymbol{F}^{(l)} \in \mathbb{R}^{T\times n}$	1D feature map at layer l
$\boldsymbol{F}^{(l)} \in \mathbb{R}^{T_1\times T_2\times n}$	2D feature map at layer l
T, $T_1 \times T_2$	Size of 1D signals or 2D images
n	The number of input channels or depth
m	The number of filters or output channels
k_1, $k_1 \times k_2$	1D or 2D filter size
$\boldsymbol{W} \in \mathbb{R}^{k_1\times m\times n}$	Weight tensor in 1D convolution
$\boldsymbol{W} \in \mathbb{R}^{k_1\times k_2\times m\times n}$	Weight tensor in 2D convolution
$\boldsymbol{b}^{(l)} \in \mathbb{R}^m$	Bias vector

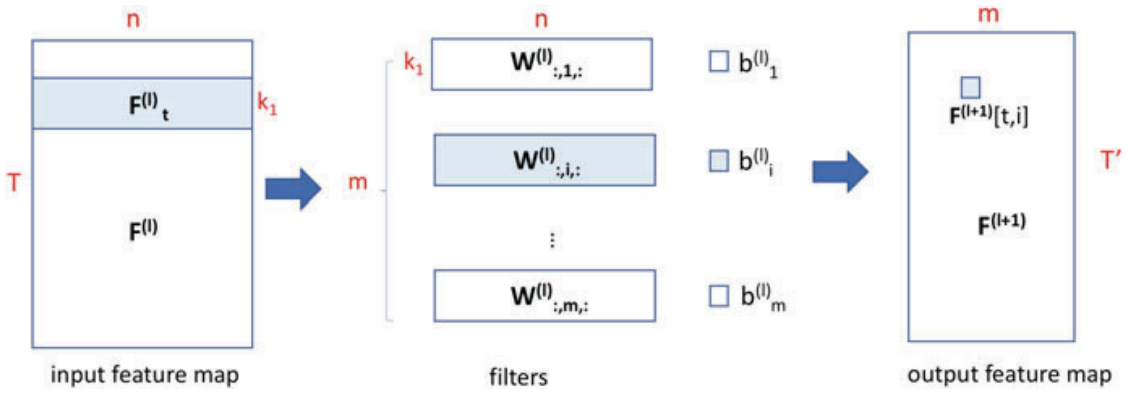

Fig. 6.1 Illustration of general 1D convolution of m filters over input feature map $\boldsymbol{F}^{(l)} \in \mathbb{R}^{T \times n}$ to generate output feature map $\boldsymbol{F}^{(l+1)}$. Filter i is specified by the weight matrix $\boldsymbol{W}_{:,i,:} \in \mathbb{R}^{k_1 \times n}$. The entire set of filters is specified by a 3D tensor $\boldsymbol{W} \in \mathbb{R}^{k_1 \times m \times n}$. $\boldsymbol{F}_t^{(l)} \in \mathbb{R}^{k_1 \times n}$ is the t-th patch of the input feature map

parameterized by a weight matrix $\boldsymbol{W}_{:,i,:} \in \mathbb{R}^{k_1 \times n}$ and a bias term $b_i^{(l)}$ (i.e., a scalar) where filter index $i = 1$ to m. The entire convolution is specified by a 3D weight tensor $\boldsymbol{W} \in \mathbb{R}^{k_1 \times m \times n}$ and a bias vector $\boldsymbol{b}^{(l)} \in \mathbb{R}^m$. Here k_1 is the filter size, m the number of filters (also the number of output channels), n is the number of input channels, For example, if we apply 50 filters of size 100 on a 6-channel EEG data, we have $k_1 = 100$ (filter size), $m = 50$ (number of filters) and $n = 6$ (number of channels) (Fig. 6.1).

Convolution Each element of the output feature map is computed as the following.

$$\boldsymbol{F}^{(l+1)}[t, i] = \boldsymbol{W}_{:,i,:}^{(l)} \otimes \boldsymbol{F}_t^{(l)} + b_i^{(l)}$$

where i is the filter index and $\boldsymbol{F}_t^{(l)} \in \mathbb{R}^{k_1 \times n}$ is the t-th patch when we slide a filter over the input, and $\boldsymbol{F}^{(l+1)}[t, i]$ is the (t, i) element of the output. Here $\otimes$ refers to the element-wise multiplication followed by summation: $\boldsymbol{W}_{:,i,:}^{(l)} \otimes \boldsymbol{F}_t^{(l)} = \sum_{a=1}^{k} \sum_{b=1}^{n} \boldsymbol{W}_{a,i,b}^{(l)} \cdot \boldsymbol{F}_t^{(l)}[a, b]$. This is analogous of the inner product but generalized to an operation between two matrices. Furthermore, it is quite common to apply a nonlinear activation such as ReLU on the output feature map. In this case, we will have

$$\boldsymbol{F}^{(l+1)}[t, i] = \sigma(\boldsymbol{W}_{:,i,:}^{(l)} \otimes \boldsymbol{F}_t^{(l)} + b_i^{(l)})$$

where σ is the activation function.

Stride One important parameter is the *stride size* S, which determines how often we apply the filter operations over the input. When $S = 1$, it means we slide the filter one element at a time. When $S = 2$, we will slide the filter two elements at

time, which means we will apply the filter every other element over the input. The large stride size leads to smaller output and vice versa.

Padding Another important detail for convolution is how to deal with the elements on the boundary. Two common strategies are *padding* or no padding. The idea of padding is to add extra values (usually zeroes) outside the input data boundary to ensure convolution can be applied to elements near the boundary. For example, if the input is of length T and the filter is of size $2P + 1$, we can pad P zeroes on each side of the input to have an output of length T (assuming stride 1).

Now we summarize the convolution layer of CNN with its parameters and hyperparameters. A convolution layer accepts input data of size $T \times n$. The convolution layer has the following hyperparameters, including the number of filters m, filter size k_1, stride size S, and padding size P. Then the layer will produce an output whose size is given by $T' \times m$ where $T' = (T - k_1 + 2P)/S + 1$.

6.2.2 Convolution Layer: 2D

Intuition When we process images, a 2D convolution becomes essential. The intuition is to construct various local pattern extractors (*filters*) and then apply them everywhere on an image to generate meaningful features. Again convolution layers can be stacked to generate more and more sophisticated features. For example, the lower level filters detect simple patterns such as edges and dots, while higher-level filters find different objects such as faces. For 2D convolution, we consider a simple example shown in Fig. 6.2 where in the middle is a 5×5 binary input matrix. On the left is a 3×3 filter. We will perform a convolution of the filter matrix over the input matrix. It is more common to have multiple input channels and multiple filters (hence multiple output channels) when applying the convolution operation.

Notations The input and output feature maps are denoted by $\boldsymbol{F}^{(l)}$ and $\boldsymbol{F}^{(l+1)}$, respectively. The 2D convolution operation is also specified by a set of m *filters*

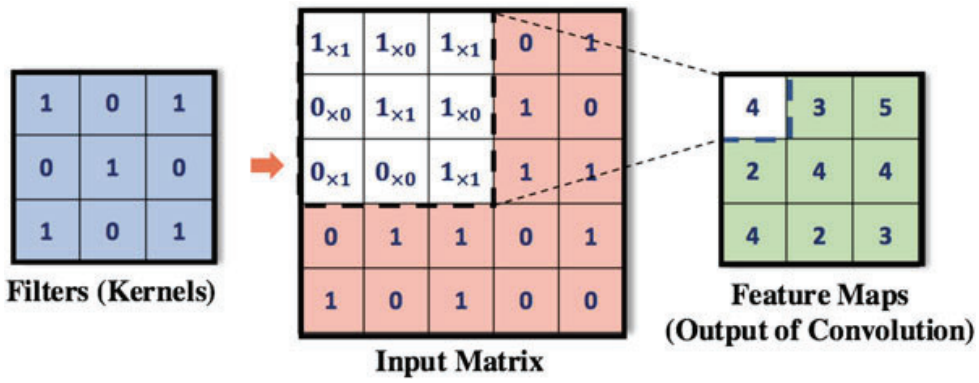

Fig. 6.2 Filters slide over an input image to produce feature maps

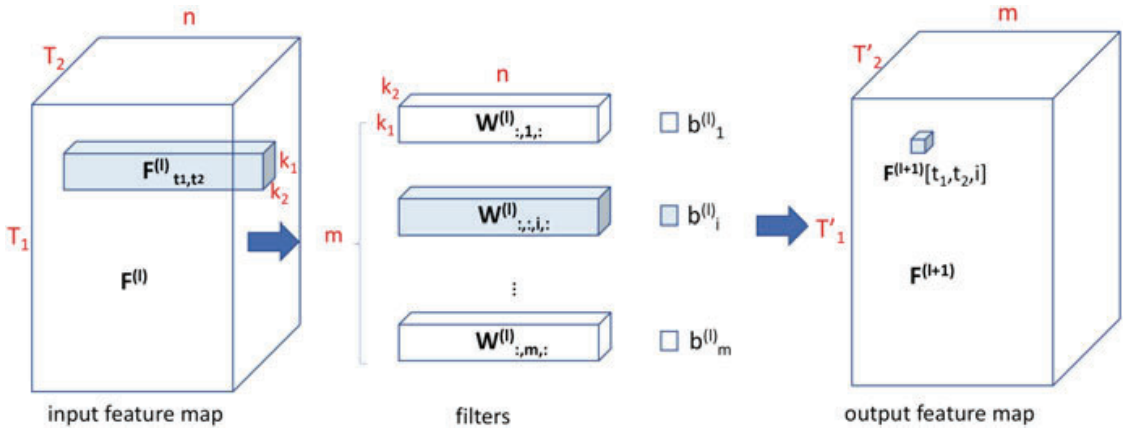

Fig. 6.3 Illustration of general 2D convolution of m filters over input feature map $\boldsymbol{F}^{(l)} \in \mathbb{R}^{T_1 \times T_2 \times n}$ to generate output feature map $\boldsymbol{F}^{(l+1)}$. A patch $\boldsymbol{F}^{(l)}_{t_1,t_2} \in \mathbb{R}^{k_1 \times k_2 \times n}$ performs element-wise multiplication followed by summation with each filter. The filter i is specified by the weight tensor $\boldsymbol{W}_{:,:,i,:} \in \mathbb{R}^{k_1 \times k_2 \times n}$ with the filter size $k_1 \times k_2$. The entire set of filters is specified by a 4D tensor $\boldsymbol{W} \in \mathbb{R}^{k_1 \times k_2 \times m \times n}$

each parameterized by a 3D weight tensor $\boldsymbol{W}_{:,:,i,:} \in \mathbb{R}^{k_1 \times k_2 \times n}$ and a bias term $b_i^{(l)}$ where filter index $i = 1$ to m. The entire convolution is specified by a 4D weight tensor $\boldsymbol{W} \in \mathbb{R}^{k_1 \times k_2 \times m \times n}$ and a bias vector $\boldsymbol{b}^{(l)} \in \mathbb{R}^m$. Here $k_1 \times k_2$ is the filter size, m the number of filters (also the number of output channels), n is the number of input channels, For example, if we apply 50 filters of size 3×3 on a 3-channel RGB image, we have $k_1 = k_2 = 3$, $m = 50$ and $n = 3$ (Fig. 6.3).

Convolution The convolution operation for 2D input is similar the 1D case but with one more index:

$$\boldsymbol{F}^{(l+1)}[t_1, t_2, i] = \sigma(\boldsymbol{W}^{(l)}_{:,:,i,:} \otimes \boldsymbol{F}^{(l)}_{t_1,t_2} + b_i^{(l)})$$

where $\boldsymbol{F}^{(l)}_{t_1,t_2} \in \mathbb{R}^{k_1 \times k_2 \times n}$ is a patch that operates with filter i via element-wise multiplication followed by summation, $\boldsymbol{W}^{(l)}_{:,:,i,:} \in \mathbb{R}^{k_1 \times k_2 \times n}$ and $b_i^{(l)}$) are the weight tensor and the bias term for filter i, respectively. And $\boldsymbol{F}^{(l+1)}[t_1, t_2, i]$ is an element of the output feature map and σ is the activation function.

In summary, a 2D convolution layer accepts input data of size $T_1 \times T_2 \times n$ where $T_1 \times T_2$ is the image size and n is the number of input channels or its depth. For example, a color medical image of size 1024×1024 is a 3D tensor of size $1024 \times 1024 \times 3$ (given RGB color channels). The convolution layer has the following hyperparameters including number of filters m, filter size $k_1 \times k_2$, stride S and number of paddings P. Then the layer will produce an output feature map of size $T_1' \times T_2' \times m$ where $T_1' = (T_1 - k_1 + 2P)/S + 1$ and $T_2' = (T_2 - k_2 + 2P)/S + 1$.

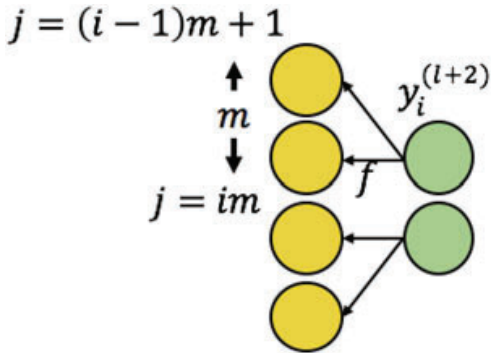

Fig. 6.4 Pooling layers of the CNN

6.2.3 *Pooling Layer*

Pooling is to subsample a feature map by aggregation. The pooling layer often follows after the convolution or activation step to produce translational invariance local features, called a pooled feature map. For example, as illustrated in Fig. 6.4, to produce one unit $y_i^{(l+2)}$ in the pooled feature map, the pooling operation focuses on one pooling region with size m in $(l+1)$th layer, indexed from $(i-1)m+1$ to im. Let $f(\cdot)$ be the pooling function such that $y_i^{(l+2)} = f([o_{(i-1)m+1}^{(l+1)}, \ldots, o_{im}^{(l+1)}])$, where $o_j^{(l+1)}$ are the output from the convolution layer. Here are several common pool functions:

- **Mean pooling** computes average values from the pooling region:

$$f([o_{(i-1)m+1}^{(l+1)}, \ldots, o_{im}^{(l+1)}]) = \frac{1}{m}\left(\sum_{j=(i-1)m+1}^{im} o_j^{(l+1)}\right)$$

- **Max pooling** picks the maximum value from the pooling region:

$$f([o_{(i-1)m+1}^{(l+1)}, \ldots, o_{im}^{(l+1)}]) = \max_{(j=(i-1)m+1):(j=im)} o_j^{(l+1)}$$

- **Sum pooling** sums up all values from the pooling region:

$$f([o_{(i-1)m+1}^{(l+1)}, \ldots, o_{im}^{(l+1)}]) = \sum_{j=(i-1)m+1}^{im} o_j^{(l+1)} \tag{6.1}$$

Regardless of the pooling function we use, pooling layers will reduce the size of the feature map. The local connections and tied weights followed by pooling will result in translation-invariant local features. We can repeat convolution and pooling multiple times and later employ fully connected layers to fuse all local features and output more abstract features for classification tasks.

To summarize the pooling layer of 2D CNN has the following parameters and hyperparameters: A pooling layer accepts input data of size is $T_1 \times T_2 \times n$. A pooling

layer has the following hyperparameters, including size $F \times F$ and stride S. Then it will produce an output of size $T_1' \times T_2' \times m$ where $T_1' = (T_1 - F)/S + 1$, $T_2' = (T_1 - F)/S + 1$. And the number of output channels remains the same as the number of input channels.

6.2.4 Fully Connected Layer

The fully connected layers are often added after all convolution and pooling layers. The reasons for adding the fully connected layers are.

- First, after a series of convolutional and pooling layers, we now have high-level features of the input data. And the fully connected layers work as classifiers to use these features for mapping the input image into various classes based on the training dataset.
- Second, the fully connected layers can perform flexible nonlinear combination to integrate features to provide better classification results.

A softmax activation function is often used in the final output layer for multiclass classification tasks, e.g., classifying whether a chest X-ray corresponds to COVID-19, viral pneumonia, bacteria pneumonia, or healthy cases like in [120].

6.3 Backpropagation Algorithm in CNN*

In the following, we will describe forward computation and backward propagation of CNN models for 1D signals. The generalization of 2D images is straightforward. The idea is very similar to DNN, but more complicated due to the convolutional and pooling operations.

6.3.1 Forward and Backward Computation for 1D Data

To present the forward computation and backward propagation, we consider a simplified 1D CNN shown in Fig. 6.5. The 1D CNN has an input feature map $\boldsymbol{x}^{(l-1)}$. We first transform the input feature map using a convolution filter $\boldsymbol{w}^{(l)}$ to produce the output feature map $\boldsymbol{x}^{(l)}$. We then apply a nonlinear activation function such as ReLU over $\boldsymbol{x}^{(l)}$ to output $\boldsymbol{o}^{(l+1)}$. Next we apply a pooling function $f()$ on the output $\boldsymbol{o}^{(l+1)}$ to produce a pooled feature map $\boldsymbol{y}^{(l+2)}$. Fully connected layers can follow these steps to generate final predictions. In the following, we omit the fully connected layers since their computations as they have been described in Chap. 4.

Forward Computation For this simple CNN, the forward computation is given by the following equations.

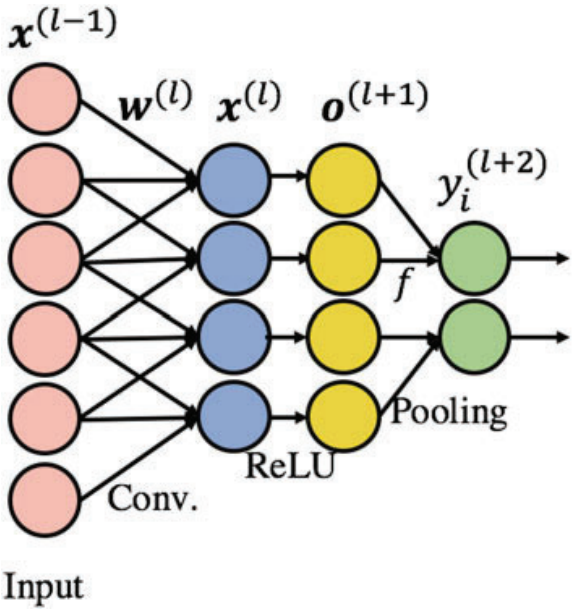

Fig. 6.5 Forward computation of a simple 1D CNN

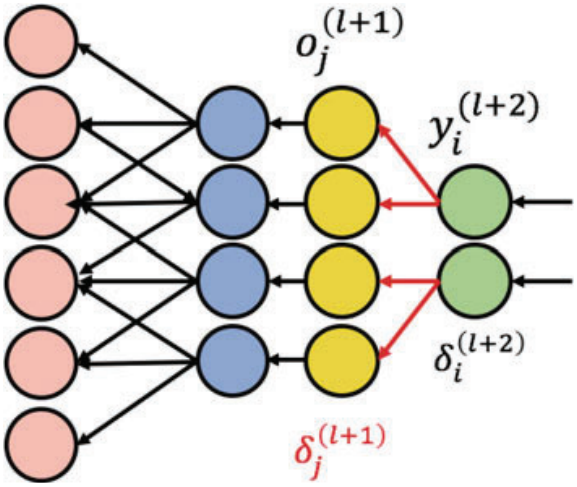

Fig. 6.6 Backward propagation for a pooling layer

$$x_i^{(l)} = \sum_m w_m^{(l)} x_{i+m}^{(l-1)} + b^{(l)}$$

$$\boldsymbol{o}^{(l+1)} = ReLU(\boldsymbol{x}^{(l)})$$

$$y_i^{(l+2)} = f([o_j^{(l+1)}]|j \in (i-1)m+1 : im)$$

where $f()$ is the pooling function.

Backpropagation for Pooling Layers For the backward propagation, we start from the pooling layer, illustrated in Fig. 6.6. We define $\delta_j^{(l+1)} = \dfrac{\partial L}{\partial o_j^{(l+1)}}$ to measure how much node j that is corresponding to the ith pooling region of $(l+1)$th layer (e.g., the region from $(i-1)m+1$ to im) was responsible for errors in the output

of that region. Likewise, we also let $\delta_i^{(l+2)} = \dfrac{\partial L}{\partial y_j^{(l+2)}}$ measure how much node i of $(l+2)$th layer was responsible for errors. Then the gradient descent updating rule for the units of the layer before pooling layer, denoted as $o_j^{(l+1)}$, can be expressed as

$$o_j^{(l+1)} = o_j^{(l+1)} - \eta \frac{\partial L}{\partial o_j^{(l+1)}}$$

where η is the learning rule. For updating this formula we will need to estimate $\delta_j^{(l+1)} = \dfrac{\partial L}{\partial o_j^{(l+1)}}$. By chain rule, we have the following derivation.

$$\begin{aligned}
\delta_j^{(l+1)} &= \frac{\partial L}{\partial o_j^{(l+1)}} \\
&= \frac{\partial L}{\partial y_i^{(l+2)}} \frac{\partial y_i^{(l+2)}}{\partial o_j^{(l+1)}} \\
&= \delta_i^{(l+2)} \frac{\partial y_i^{(l+2)}}{\partial o_j^{(l+1)}}
\end{aligned}$$

where $\dfrac{\partial y_i^{(l+2)}}{\partial o_j^{(l+1)}}$ has different formulation depends on the pooling function. They are summarized as follows:

- For mean pooling, we have $\dfrac{\partial y_i^{(l+2)}}{\partial o_j^{(l+1)}} = \frac{1}{m}$;
- For sum pooling we have $\dfrac{\partial y_i^{(l+2)}}{\partial o_j^{(l+1)}} = 1$;
- For max pooling, for the unit $j = \arg\max(o_j^{(l+1)})$, we have $\dfrac{\partial y_i^{(l+2)}}{\partial o_j^{(l+1)}} = 1$, otherwise 0. That is the gradient passed back from layer $l+2$ is only towards that neuron which achieved the max. All other neurons have zero gradient.

Backpropagation for Convolution Layers Next we will describe how to perform backpropagation of a convolution layer. For the example in Fig. 6.7, we define $\delta_j^{(l-1)} = \dfrac{\partial L}{\partial x_j^{(l-1)}}$ to measure how much node j in the $(l-1)$th layer (the feature map) was responsible for errors in the output. By chain rule, we have the equations below.

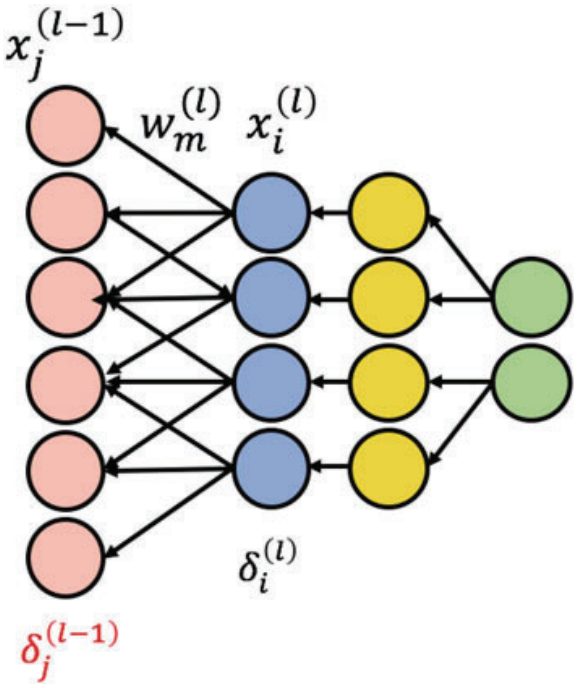

Fig. 6.7 Backward propagation for a convolution layer

$$\begin{aligned}
\delta_j^{(l-1)} &= \frac{\partial L}{\partial x_j^{(l-1)}} \\
&= \frac{\partial L}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial x_j^{(l-1)}} \\
&= \delta_i^{(l)} \frac{\partial x_i^{(l)}}{\partial x_j^{(l-1)}} \\
&= \delta_i^{(l)} \frac{\partial (\sum_m w_m^{(l)} x_{i+m}^{(l-1)} + b^{(l)})}{\partial x_j^{(l-1)}} \\
&= \delta_i^{(l)} \sum_m w_m^{(l)}
\end{aligned}$$

Next we analyze the gradient of the weight vector $\frac{\partial L}{\partial w_m^{(l)}}$. By the chain rule, we have the equations below.

$$\begin{aligned}
\frac{\partial L}{\partial w_m^{(l)}} &= \frac{\partial L}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial w_m^{(l)}} \\
&= \delta_i^{(l)} \frac{\partial x_i^{(l)}}{\partial w_m^{(l)}}
\end{aligned}$$

$$= \delta_i^{(l)} \frac{\partial(\sum_k w_k^{(l)} x_{i+k}^{(l-1)} + b^{(l)})}{\partial w_m^{(l)}}$$
$$= \delta_i^{(l)} \sum_m x_{i+m}^{(l-1)}.$$

Similarly, we can find the gradient of the bias term $\frac{\partial L}{\partial b^{(l)}}$:

$$\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial b^{(l)}}$$
$$= \delta_i^{(l)} \frac{\partial x_i^{(l)}}{\partial b^{(l)}}$$
$$= \delta_i^{(l)} \frac{\partial(\sum_k w_k^{(l)} x_{i+k}^{(l-1)} + b^{(l)})}{\partial b^{(l)}}$$
$$= \delta_i^{(l)}.$$

Once the gradients are identified, the update rules are simply

$$w_m^{(l)} = w_m^{(l)} - \eta \frac{\partial L}{\partial w_m^{(l)}}$$

and

$$b^{(l)} = b^{(l)} - \eta \frac{\partial L}{\partial b^{(l)}}.$$

where η is the learning rate.

6.3.2 *Special CNN Architectures*

Many different CNN architectures have been proposed over the years. Next, we introduce a few popular CNN architectures. Most of them have been compared on the ILSVRC ImageNet challenge.[1] Different architectures are compared based on the top 5 classification error for the competition, which is the percentage of mistakes in the model's top 5 predictions. The ImageNet dataset has 1000 classes and includes many images that are hard to be distinguished from others. From Fig. 6.8, we

[1] http://image-net.org.

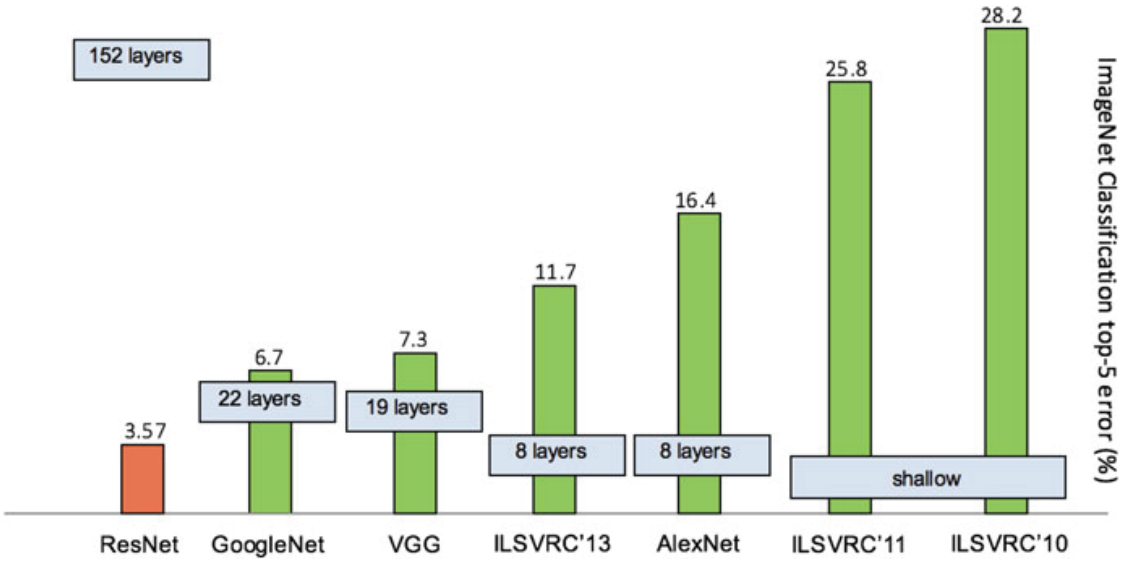

Fig. 6.8 The trend of CNN model performance on the ImageNet challenge

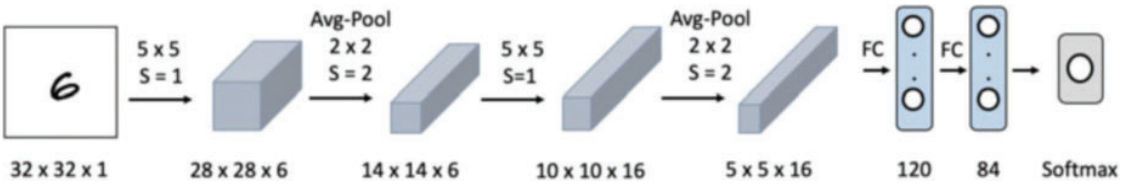

Fig. 6.9 The LeNet architecture

observe CNN variants become more and more accurate in image classification. Meanwhile, architectures become much larger and deeper.

LeNet

LeNet [96] is an earlier CNN structure that contains seven levels of a convolutional network and classifies digits. In the original work, it was applied to recognize handwritten numbers on checks. As shown in Fig. 6.9, LeNet takes 32-by-32 grayscale images as input and first apply six 5×5 convolutional filters with stride 1 and no padding to produce a feature map of $28 \times 28 \times 6$. A 2×2 average pooling filter with stride 2 is applied to produce a feature map of $14 \times 14 \times 6$. After that, another 16 5×5 convolutional filters with stride 1 and no padding are applied to generate a feature map of $10 \times 10 \times 16$. Then another average pooling layer is introduced, followed by two fully connected layers to generate the final softmax classification.

LeNet has about 60k parameters. To process higher resolution images, LeNet needs more and larger convolutional layers. Given that LeNet was proposed before the popularity of GPU, the available computation hardware limited its architecture.

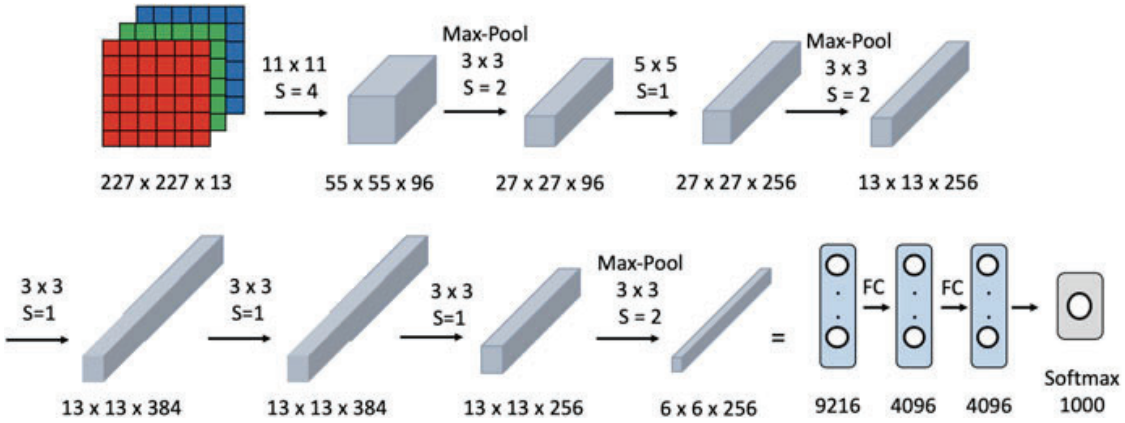

Fig. 6.10 The AlexNet architecture

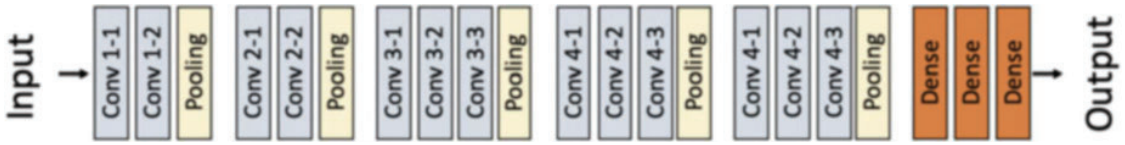

Fig. 6.11 The VGG16 architecture

AlexNet

The AlexNet was architecturally similar to LeNet [90] but is deeper and has more filters per layer, and has stacked convolutional layers. AlexNet introduces the ReLU activation following convolution layers. It has about 60 million parameters. AlexNet is the first deep learning model that won the ImageNet challenge in 2012 with a top-5 error rate of 15.3%, compared to the second place top-5 error rate of 26.2%. After that, all the winning models of the ImageNet challenge are deep learning models (Fig. 6.10).

VGG

VGG16 is a convolutional neural network model proposed in [136]. VGG16 improved AlexNet by replacing large filters (e.g., 11×11 and 5×5) with multiple small filters one after another. Figure 6.11 shows the detailed architecture of VGG16. VGG16 also introduces the 1×1 filter, which performs a linear combination of input channels followed by a nonlinear activation (ReLU function). It has 138 million parameters. For example, the VGG16 model takes over 533MB storage, which can be prohibitively expensive for training and inference.

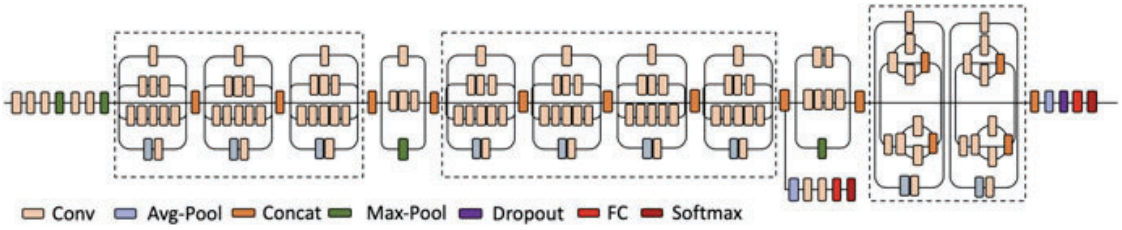

Fig. 6.12 Inception V3 architecture

GoogLeNet Inception Net

GoogleLeNet Inception architecture is another popular CNN architecture with several important ideas [146] as shown in Fig. 6.12:

- **Multiple resolutions:** Instead of stacking convolution layers, inception architectures introduce parallel paths of multiple convolution filters of different sizes to capture patterns at a different resolution. The combination of multiple filters together serves as the building blocks called *inception cells*, which will be stack up to form the CNN architecture.
- **Stacking up smaller filters:** To reduce computational cost and the number of parameters, it is more efficient to stack multiple smaller filters to approximate a larger filter. For example, a 5×5 filter can be approximated by connecting two 3 filters. Similarly, a 3×3 filter can be approximated by a 3×1 filter followed by 1×3 filter. Like VGG16, 1×1 filters are also used to adjust the number of output channels by combining the input channels.
- **Auxiliary loss:** Adding auxiliary loss in the middle of the network improves the model performance. This idea will be significantly enhanced in ResNet via the notion of skipped connection.

The number of parameters of Inception architecture is about 5 million for V1 and 23 million for V3.

ResNet

Residual Neural Network (ResNet) [66] overcomes a troubling phenomenon when training a very deep neural network called *degradation problem*. That is, when the depth of the neural network increases the performance of the network decreases. The intuitive explanation of why degradation happens is that each layer has to construct a brand new set of features without reusing the features learned from previous layers. To use features from previous layers, ResNet proposes "skip connections." that allows input directly passing through without any transformation. As a result, the new features learned at any layer model the residual that the input features have not already captured. Formally, this effect is modeled by a residual block as shown in Fig. 6.13.

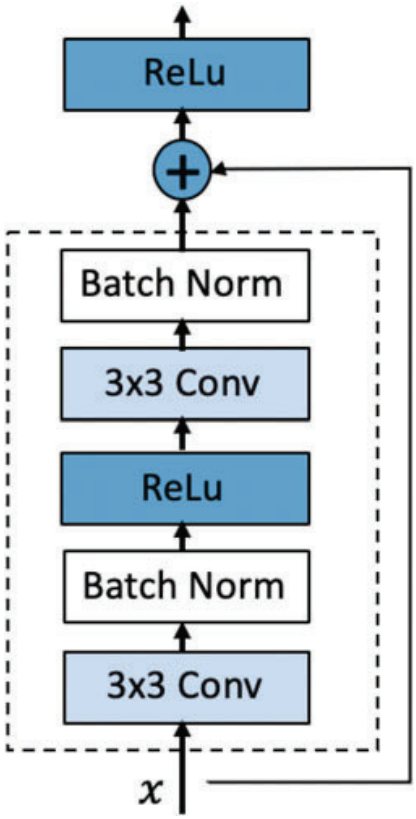

Fig. 6.13 Residual block

The residual learning has analogous effects on auxiliary loss or gated units. Thanks to this technique, ResNet can train a very deep neural network (e.g., 152 layers) while still achieving high accuracy. The number of parameters of a ResNet model is actually moderate. For example, the popular architecture ResNet-50 has 25 million parameters.

DenseNet

Densely connected convolutional network (DenseNet) [75] generalizes the idea of skipped connections by connecting the output of a layer as input to all later layers. In this way, DenseNet can maximally reuse features from all layers. Thanks to the feature reuse, DenseNet can perform well with a small number of filters (e.g., 12 filters per layer). As a result, it leads to fewer parameters than ResNet. DenseNet also repeats computational units called dense blocks to construct deeper networks, as shown Fig. 6.14. Thanks to dense connections, the gradient can flow more easily through the deep network.

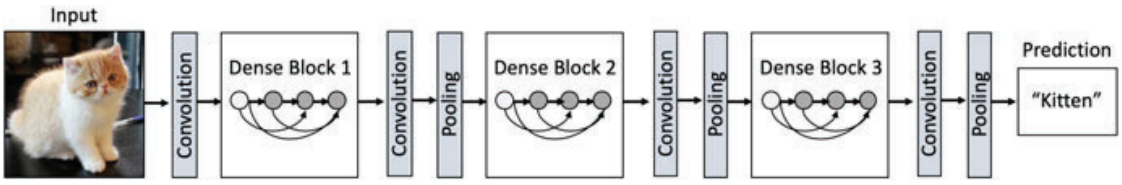

Fig. 6.14 DenseNet architecture

6.4 Case Study: Diabetic Retinopathy Detection

Problem How do the CNN models perform on detecting diabetic retinopathy from retinal fundus images? How does the model performance compare to manual grading by ophthalmologists? A systematic study is presented in [61] to address these questions.

Data The training data were macula-centered retinal fundus images obtained from EyePACS in the United States and India among patients who received diabetic retinopathy screening. Ophthalmologists graded all images in the development and clinical validation sets for the presence of diabetic retinopathy. Diabetic retinopathy severity (none, mild, moderate, severe, or proliferative) was graded according to the International Clinical Diabetic Retinopathy scale. The data are summarized in Table 6.2. Note that the labeled training/development set is very large, over 128K, which gives the edge for training a complex CNN model.

Method To make classification, a convolutional neural network was adopted to compute diabetic retinopathy severity from the intensities of the pixels in a fundus image. It first combines nearby pixels into local features, then aggregates those into global features. Although the network does not explicitly detect lesions (e.g., hemorrhages, microaneurysms), it likely learns to recognize them using the local features. The specific neural network used in this work is the Inception v3 architecture [147].

Results CNN models showed high sensitivity and specificity on the tasks. In 2 validation sets of 9963 images and 1748 images, at the operating point selected for high specificity, the algorithm had 90.3 and 87.0% sensitivity and 98.1 and 98.5% specificity for detecting referable diabetic retinopathy, defined as moderate or worse diabetic retinopathy or referable macular edema by the majority decision of a panel of at least 7 US board-certified ophthalmologists. At the operating point selected for high sensitivity, the algorithm had 97.5 and 96.1% sensitivity and 93.4 and 93.9% specificity in the two validation sets. Another important result in this work is to illustrate the importance of training sample size and labels' reliability. Figure 6.15a shows the increasing trend of model performance as the size of training data. In this task, 50k training images seem to be sufficient. Figure 6.15b shows the increasing performance trend as more grades per image are assigned. More grades per image lead to more reliable labels, which in turn can improve the model performance.

Table 6.2 Data set summary of diabetic retinopathy detection study

Characteristics	Dev set	EyePACS-1 validation data	Messidor-2 validation data
No. of images	128,175	9963	1748
No. of ophthalmologists	54	8	7
No. of grades per image	3–7	8	7
Grades per ophthalmologists, medium	2021 (304–8366)	8906 (8744–9360)	1745 (1742–1748)
No. of unique patients	69,573	4997	874
Age (mean)	55.1 (11.2)	54.4 (11.3)	57.6 (15.9)
% of female	59.9	62.2	42.6
% of fully gradable images	75.1	88.4	99.8
# images for both diabetic retinopathy and macular edema	118,419	8788	1745
No diabetic retinopathy	53,759	7252	1217
Mild diabetic retinopathy	30,637	842	264
Severe diabetic retinopathy	5298	54	28
Proliferative diabetic retinopathy	4359	95	25
Referable diabetic macular edema	18,224	272	125
Referable diabetic retinopathy	33,246	683	254

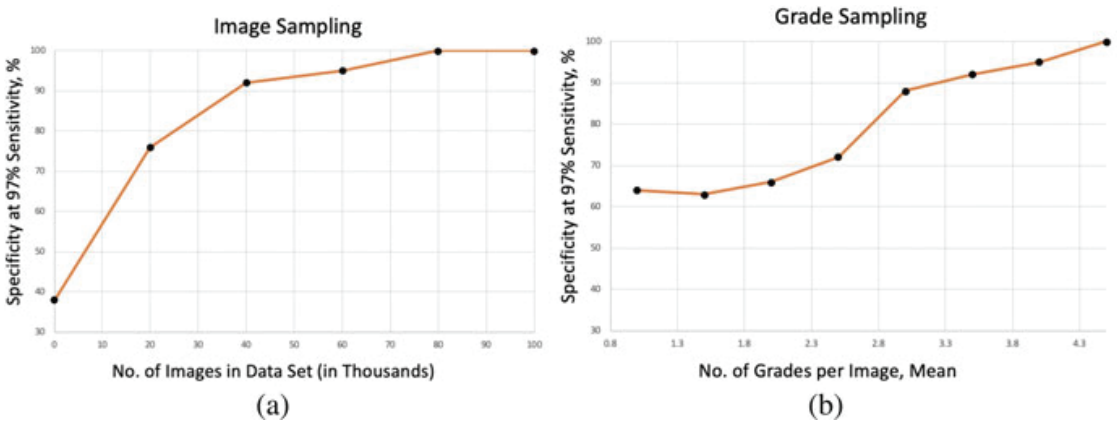

Fig. 6.15 Model performance for diabetic retinopathy detection as the training data size and number of grades per image vary

6.5 Case Study: Skin Cancer Detection

Problem Another famous example is to use CNN to assist skin cancer detection.

Data and Method In [44], researchers trained a CNN using a dataset of 129,450 clinical skin images. In particular, the dataset is composed of dermatologist-labeled images organized in a tree-structured taxonomy of 2032 diseases, in which the individual diseases form the leaf nodes. The images come from 18 different clinician-curated, open-access online repositories, and clinical data from Stanford University Medical Center. The dataset is split into 127,463 training and validation images and 1942 biopsy-labeled test images. CNNs were applied for two diagnosis tasks: classification of keratinocyte carcinomas versus benign seborrheic keratoses; and malignant melanomas versus benign nevi. The first case represents the identification of the most common cancers. The second represents the identification of the deadliest skin cancer. An Inception V3 model was pretrained on approximately 1.28 million images (1000 object categories) from the 2014 ImageNet Large Scale Visual Recognition Challenge. To initialize with the pretrained model, the CNN model was further trained (i.e., finetuning) on the skin images. The CNN model is trained using 757 disease classes.

Results The performance was compared against 21 board-certified dermatologists. The CNN achieves performance on par with all tested experts across both tasks, demonstrating an artificial intelligence capable of classifying skin cancer with a level of competence comparable to dermatologists. The detailed results are shown in Fig. 6.16, where the CNN model outperforms the average dermatologists. Figure 6.17 shows the t-SNE plot of the last layer of the CNN model, where a clear separation of images from different classes can be observed.

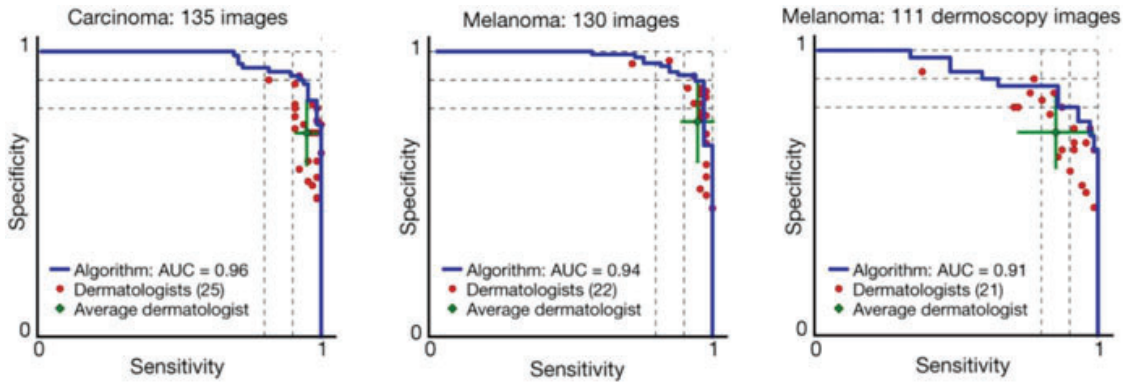

Fig. 6.16 Performance of skin cancer detection: These figures show the performance comparison on 3 test tests where 21–25 dermatologists also scored the same test set. The red dots indicate the individual doctors' performance, and green cross is the average performance of the dermatologists. The blue curve is the ROC curve of the CNN model, which is higher than the performance of the average dermatologists

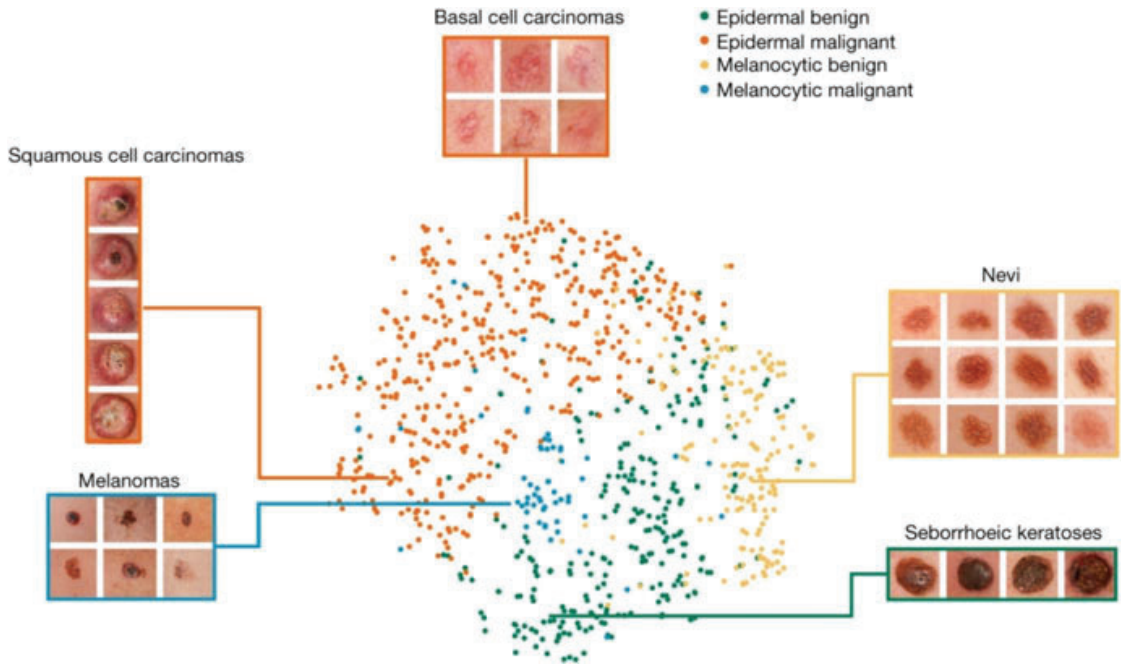

Fig. 6.17 The t-SNE plot of the last layer of CNN model can clearly cluster the images based on different types of skin cancers

6.6 Case Study: Automated Surveillance of Cranial Images for Acute Neurologic Events

Problem How to perform accurate and efficient screen head CT images for acute neurologic events? Authors of [156] introduce a 3D-CNN model trained with weak supervision provided by the NLP program.

Data The data used in this work includes 37,236 imaging studies and 96,303 radiology reports from the Icahn School of Medicine AI Consortium (AISINAI).

All imaging studies and radiology reports were annotated with silver labels based on diagnostic terms derived from the Universal Medical Language System (UMLS) concept of universal identifiers. The NLP labels for UMLS concept universal identifiers were aggregated into higher-level disease types. A small number of images (180) are used to create the test set, manually reviewed, and labeled. The rest are used as training and validation, which are labeled using the silver labels based on extracted UMLS identifiers.

Method To identify whether an image contained acute neurological illnesses, ResNet-50 architecture followed by a 3D-CNN was trained to predict disease acuity. In their simulation, the studies were marked as critical or non-critical by the deep learning model. Then images are ordered in a work queue according to the probability of a critical finding. A random order was used as the baseline for comparison. For the test data, 180 images were randomly queried from the dataset of silver-standard labels to achieve an approximately 1:1 split of noncritical to critical studies. Gold-standard labels were obtained through manual review, where radiologists manually assigned each disease's acuity as critical or non-critical. The remaining studies with silver labels were split into training and validation sets with an 80 : 20 ratio. The prevalence of critical findings of the silver labels is 7.6, 0.8, and 0.7% at thresholds of 4, 8, and 10 UMLS critical findings per note, respectively.

Result There are two sets of results. One set is on model prediction performance such as AUC, accuracy, sensitivity, and specificity. The other set is about a trial simulation for triage based on image interpretation.

Model Performance Since the model is trained with silver labels, their performance is tested on silver label data and, ultimately, the small gold standard test set. For predicting the silver labels of critical findings on the training set, the 3D-CNN achieved a high AUC of 0.88. However, the model performance on the gold standard test set is significantly lower. The DL model achieved an AUC of 0.73. The sensitivity and specificity of the model were 0.79 and 0.48. In comparison, the physicians' average sensitivity and specificity are 0.79 (s.d. 0.04) and 0.85 (s.d. 0.07), which is much higher, especially in specificity. This result suggests that human physicians are still significantly better in this task than DL models, probably because of the limited quality of silver labels used for model training.

Trial Simulation Authors simulated a randomized control trial (RCT) of image interpretation. In the simulation, both the DL model and radiologists were assessed regarding how quickly they recognized and provided notification of a critical finding. Since queue prioritization and rapid assessment are the focus, the model was assessed for its runtime. The average time for the DL model to score an image and potentially raise the alarm was 1.2 s versus 177 s for humans. The triaged queue constructed by the DL model had a significantly lower distribution of queue positions for critical studies compared with noncritical studies (two-sided Wilcoxon rank-sum test, $P = 0.01$, $n = 180$). This seems to confirm the DL model's potential utilization in triage, even with limited prediction accuracy.

6.7 Case Study: Detection of Lymph Node Metastases from Pathology Images

Problem Accurately detecting metastases in whole-slide pathology images is an important task for diagnosing breast cancer. Currently, the task was performed by pathologists, who are expensive and time-consuming. In [42] the authors reported a research competition (CAMELYON16) to develop automated solutions for detecting lymph node metastases.

Data The training data used in the analysis was a dataset of whole-slide images from 2 centers in the Netherlands (110 with nodal metastases, 160 without nodal metastases). Algorithm performance was evaluated in an independent test set of 129 whole-slide images (49 with and 80 without metastases). A panel of 11 pathologists also evaluated the same test set with time constraint (WTC—2 h to evaluate 129 test images) from the Netherlands to ascertain the likelihood of nodal metastases for each slide in a flexible 2-h session, simulating routine pathology workflow, and also by 1 pathologist without time constraint (WOTC).

Method Many methods are proposed in the context of the CAMELYON16 competition. Several CNN architectures were used to analyze whole-slide pathology images. There are two tasks in this competition:

- **Task 1 Tumor localization** identifies the locations of individual metastases on the images. The algorithms were evaluated for their ability to identify specific metastatic foci in a whole-slide image. The metric for task 1 is specialized. In particular, a set of locations need to be scored based on a true-positive fraction, which is from 6 predefined false-positive rates: 1/4 (meaning 1 false-positive result in every 4 images), 1/2, 1, 2, 4, and 8 false-positive findings per whole-slide image;
- **Task 2 Image classification** classifies the whole images without the need to specify the locations of metastases. The evaluation metric for task 2 is the standard metric of the area under the receiver operating curve (AUC). Task 2 evaluated the algorithms' ability to discriminate between 49 whole-slide images with SLN metastases vs. 80 without SLN metastases (control). In this case, identification of specific foci within images was not required. Participants provided a confidence score, using the same rating schema as task 1, indicating the probability that each whole-slide image contained any evidence of SLN metastasis from breast cancer.

Next, we describe the winning method from MGH and MIT [42]. First, a threshold-based algorithm is used to differentiate the tissue regions and the background. Then CNN models are only trained on the tissue regions. Millions of 256×256 patches are sampled from both positive and negative regions to construct large samples to train the CNN models. Various CNN architectures are used with different accuracy. Similarly, different magnification levels (40x, 20x, and 10x) are tested. GoogLeNet and 40x magnification led to the best performance for patch level

classification. Based on the patch level classification, a set of false-positive patches are added into the training set as the "hard negative" patches. Data augmentation such as rotation, random cropping, and addition of color noise are also added to enrich the training data's diversity.

Result The best performing algorithm [42] used a GoogLeNet architecture, which outperformed all with a score of 0.807 (95%CI, 0.732–0.889) for task 1, and an AUC of 0.994 (95%CI, 0.983–0.999) for task 2. This AUC exceeded the pathologists' mean performance with the time constraint (mean AUC, 0.810 [$range$, 0.738 – 0.884]) in the diagnostic simulation exercise.

6.8 Case Study: Cardiologist-Level Arrhythmia Detection and Classification in Ambulatory ECG

Problem Computerized electrocardiogram (ECG) interpretation is an important problem. Authors in [65] propose a deep learning model for detecting 12 rhythm classes from ECG data.

Data One major reason that limits deep learning in ECG analysis is the lack of appropriate training data. In this work, a large ECG dataset was constructed with expert annotation with a broad range of ECG rhythm classes. The training dataset consists of a single lead EEG with 91,232 ECG records from 53,549 patients.
The ECG data were recorded by a single-lead, patch-based ambulatory ECG monitor that continuously records data from a single vector (modified Lead II) at 200Hz. The mean and median wear time was 10.6 and 13.0 days, respectively. Mean age was 69 ± 16 years, and 43% were women. The model was evaluated on a test dataset that consisted of 328 ECG records collected from 328 unique patients, which was annotated by a consensus committee of expert cardiologists. Annotations are done by certified ECG technicians, which specified the start time and the end time of all rhythms. Rare rhythms such as AVB are oversampled in the training set. The ground truth labels are assigned with the consensus from the cardiologist committees of 3 members each.

Method This work develops a deep neural network (DNN) to classify 12 rhythm classes using 91,232 single-lead ECGs from 53,549 patients who used a single-lead ambulatory ECG monitoring device. A CNN model is developed, as shown in Fig. 6.18. The model takes a 30 s ECG record of 200 Hz (6000-dimensional vectors) and outputs probability score distribution over 12 classes: (1) Atrial Fibrillation, (2) Atrio-ventricular Block, (3) Bigeminy, (4) Ectopic Atrial Rhythm, (5) Idioventricular Rhythm, (6) Junctional Rhythm, (7) Noise, (8) Sinus Rhythm, (9) Supraventricular Tachycardia, (10) Trigeminy, (11) Ventricular Tachycardia, and (12) Wenckebach. The model is trained and applied in a sliding window fashion and outputs the 12-dimensional prediction every 256 elements (every 1.28 s). The architecture is based on ResNet.

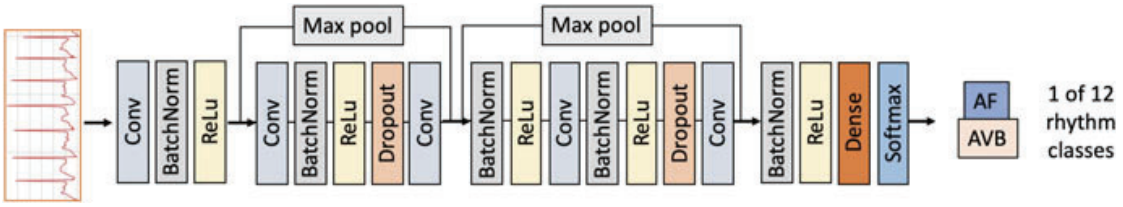

Fig. 6.18 Thirty seconds ECG record of 200 Hz is processed through this CNN model for 12 class classification

Results The proposed model was validated against an independent test dataset annotated by a consensus committee of board-certified practicing cardiologists. The DNN achieved an average area under the receiver operating characteristic curve (ROC) of 0.97. The average F1 score, which is the harmonic mean of the positive predictive value and sensitivity, for the DNN (0.837) exceeded that of average cardiologists (0.780). With specificity fixed at the average specificity achieved by cardiologists, the sensitivity of the DNN exceeded the average cardiologist sensitivity for all rhythm classes.

6.9 Case Study: COVID X-Ray Image Classification

Problem Nothing is probably more surprising and devastating to the whole world than the COVID-19 pandemic in 2020. Accurate and efficient diagnosis and triaging of COVID-19 can be of great value for hospitals to fight the pandemic. Researchers of [120] studied the possibility of differentiating chest x-ray images of COVID-19 against other pneumonia and healthy patients using deep neural networks. Our main task is to differentiate between chest x-ray images of COVID-19 and other pneumonia and healthy patients, which can be considered multiple classification tasks.

Data We combined two publicly available datasets:

- **COVID Chest X-ray (CCX) dataset:** We obtain the COVID-19 pneumonia images from the CCX dataset, which contains a few X-ray images from the other classes.
- **Kaggle Chest X-ray (KCX) dataset:** This dataset contains three types of X-ray images: normal, bacterial pneumonia, and non-COVID-19 viral pneumonia.

All KCX images and most CCX images are based on an anteroposterior (AP) or posteroanterior (PA) view. We only included images with both AP and PA view for consistency purposes in our final experimental dataset. The dataset comprises a total of 5508 chest radiography images across 2874 independent patient cases. All images in KCX data are all in AP/PA view. CCX data has 131 images, of which 119 are AP/PA images. We only selected the 119 AP/PA images in our experiments,

including 100 covid-related pneumonia, 11 viral pneumonia, 7 bacterial pneumonia, and 1 normal patient Chest X-ray images. Because AP and PA are two different types of X-ray views, we introduced horizontal flips and random noise to convert PA into AP view.

Method We construct the X-ray imaging data from two publicly available sources including include 5508 chest x-ray images across 2874 patients with four classes: normal, bacterial, non-COVID-19 viral pneumonia, and COVID-19. To identify COVID-19, we propose a Focal Loss Based Neural Ensemble Network (FLANNEL), a flexible module to ensemble several convolutional neural networks (CNN) models and fuse with a focal loss for accurate COVID-19 detection on class imbalance data.

Stage-1 Base Learner Training The convolutional neural networks (CNNs) have been widely used in image classification and get huge successes. Here, we choose 5 popular and state-of-the-art CNN classification models as base learners to model the COVID-19 identification task. The following models were chosen due to their flexibility and high performance with general image classification.

- Inception v3: It is the third edition of Google's Inception Convolutional Neural Network[146].
- VGG19: The model architecture is from the VGG group with batch normalization and consists of 19 layers, where 16 Convolutional layers and 3 Fully Connected layers [136].
- ResNeXt101: This 101-layer architecture is designed by the ResNeXt group [168].
- Resnet152: This is a 152-layer Deep Residual Neural Network that learns the residual representation functions instead of directly learning the signal representations [66].
- Densenet161: This is a Densely Connected Convolutional Network with 161 layers [75].

Stage-2 Ensemble Model Learning We take the linear combination of the base learner predictions using the resulting base learner weights (denoted as $\boldsymbol{w}$) to obtain the final prediction $\hat{y} = \text{Softmax}(\sum_i w_i P_i)$.

However, heavy class imbalances in the data during training will overwhelm and dominate the gradient, making optimal parameter updates difficult. To overcome this class imbalance challenge, we adapt *focal loss*. Originally, focal Loss is a loss function proposed for binary classification tasks, where the well-classified examples are down-weighted and can focus on learning the hard imbalanced examples [16]. Here, we extend focal loss to multiclass classification in our model to address these imbalance issues. For each image, we define the focal loss as:

$$L(\hat{y}, y) = \sum_{m=1}^{M} -\alpha_m y_m (1 - \hat{y}_m)^{\gamma} \log(\hat{y}_m)$$

Table 6.3 Performance comparison on F1-score: class-specific F1-score is calculated using one class vs. the rest strategy

	COVID	Viral pneumonia	Bacterial pneumonia	Normal	Macro-F1
InceptionV3	0.5904(0.27)	0.5864(0.05)	0.8056(0.01)	0.8771(0.04)	0.7149(0.09)
VGG19	0.6160(0.06)	0.5349(0.04)	0.7967(0.02)	0.8691(0.03)	0.7042(0.02)
ResNeXt101	0.6378(0.12)	0.5649(0.03)	0.7959(0.01)	0.8537(0.02)	0.7140(0.03)
Resnet152	0.6277(0.11)	0.5506(0.02)	0.7988(0.01)	0.8700(0.01)	0.7110(0.03)
Densenet161	0.6880(0.07)	0.5930(0.02)	0.8017(0.01)	0.8953(0.01)	0.7445(0.02)
Voting	0.7684(0.04)	0.6005(0.03)	0.8214(0.03)	0.9079(0.01)	0.7745(0.01)
FLANNEL	0.8168(0.03)	0.6063(0.02)	0.8267(0.00)	0.9144(0.01)	0.7910(0.01)

where M is the number of classes, $(1-\hat{y}_m)^\gamma$ is the modulating factor with a focusing parameter γ, and α_m represents a weight parameter for class m.

Results FLANNEL consistently outperforms baseline models on the COVID-19 identification task in all metrics. Compared with the best baseline, FLANNEL shows a higher macro-F1 score with a 6% relative increase on the Covid-19 identification task where it achieves 0.7833±0.07 in Precision, 0.8609±0.03 in Recall, and 0.8168±0.03 F1 score (Table 6.3). Ensemble learning that combines multiple independent basis classifiers can increase robustness and accuracy. In summary, FLANNEL effectively combines state-of-the-art CNN classification models and tackle class imbalance with Focal loss to achieve better performance on Covid-19 detection from X-rays.

6.10 Exercises

1. If you are given a dataset of 100 X-ray images, would you still use CNN models? If so, which architecture would you try? And why?
2. What about you are given 10,000 images, which CNN models would you try?
3. Question 1 Which of the following is NOT true about convolutional neural networks (CNN)?

   (a) Compared to fully connected neural network, CNN has more local connections.
   (b) CNN enables weight sharing through filters.
   (c) CNN utilizes pooling to try to address the translation invariance challenge.
   (d) CNN usually has much more parameters than fully connected neural networks.

4. Given input sequence = [1, 2, 3, 4, 5, 6] and a filter [1 2 1], what is the output of the convolution operation with stride 2 with no padding?

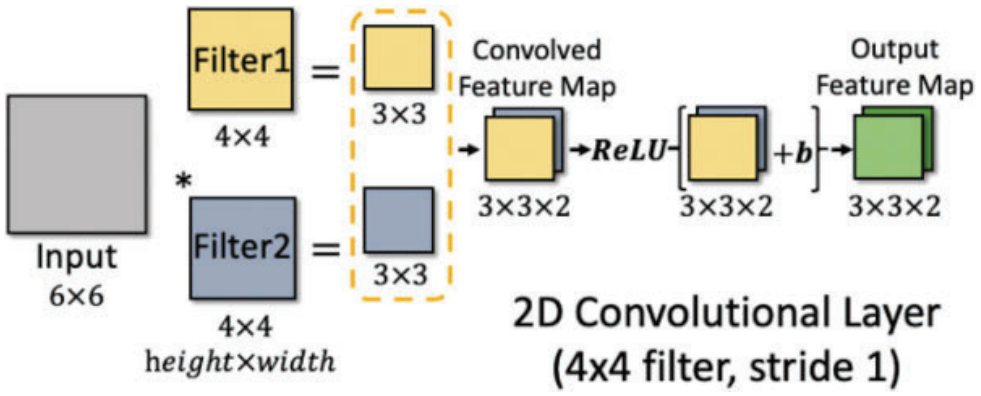

Fig. 6.19 CNN exercise

5. What is the size of the output feature map given 6x6 input and two filters of 4x4 with stride 1 and no padding as illustrated in Fig. 6.19?
6. What is the number of parameters for a convolutional layer given 6x6 input and two filters of 4x4 with stride 1 and no padding as illustrated in Fig. 6.19?
7. What is the output of 1D max pooling over input [1, 2, 3, 2, 3, 5] with filter of size 2 and stride 2?
8. What is NOT true about AlexNet?

   (a) The first deep neural network that won the ImageNet classification challenge.
   (b) AlexNet uses larger filter of 11x11 which is computationally more efficient than the subsequent CNN models such as VGG and ResNet.
   (c) It has parameters in feedforward layers than the convolution layers.
   (d) It uses more computational power in the convolution operations than feedforward computation in fully connected layers.

9. What is NOT true about CNN architectures?

   (a) VGG introduces the idea of stacking smaller filters instead of a larger filter
   (b) Inception Net introduces the parallel paths.
   (c) Inception Net uses 1x1 filters to adjust the output size so that the dimensions across different parallel paths are the same
   (d) ResNet introduces skip connections to allow learning networks of variable depths.

10. Which of the following is true about CNN models?

    (a) CNN is a single model for imaging classification.
    (b) ResNet has more parameters than both VGG and Inception net.
    (c) CNN can be applied to both images and text data.
    (d) The size of a CNN model (number of model parameters) is proportional to the amount of computation (FLOPS).

11. Why did the CNN model succeed in diabetic retinopathy detection application?

    (a) Many standardized retina images exist.
    (b) Many labeled images exist.
    (c) Multiple labels are collected to generate gold standard labels.
    (d) This is a simple task.

12. What neural network architecture was used to detect skin cancer in [44]?