Convolutional Neural Networks

©2025 Codefinity. All Terms Reserved.

Was ist ein CNN und warum unterscheidet es sich von traditionellen neuronalen Netzen?

Ein **Convolutional Neural Network (CNN)** ist eine Form der künstlichen Intelligenz, die Computern hilft, Bilder zu "sehen" und zu verstehen. Im Gegensatz zu herkömmlichen neuronalen Netzen, die Bilder als Zahlenlisten verarbeiten, **analysieren CNNs Bilder abschnittsweise** und erkennen dabei Muster wie **Kanten, Formen und Texturen**. Dadurch sind sie wesentlich besser für die Verarbeitung von Bildern und Videos geeignet.

©2025 Codefinity. All Terms Reserved.

Merkmal	Traditionelle Neuronale Netzwerke	Convolutional Neural Networks
Konnektivität ⓘ	Vollständig verbunden (jedes Neuron ist mit allen Neuronen der nächsten Schicht verbunden)	Verwendet lokale rezeptive Felder und geteilte Gewichte
Parametereffizienz ⓘ	Niedrig (benötigt eine große Anzahl an Gewichten)	Hoch (weniger Parameter durch Gewichtsteilung)
Skalierbarkeit ⓘ	Hat Schwierigkeiten mit großen Bildern	Effizient für hochdimensionale Daten wie Bilder
Räumliches Bewusstsein ⓘ	Kann keine räumliche Hierarchie erfassen	Erhält räumliche Beziehungen in Bildern

Wie CNNs vom menschlichen Auge inspiriert sind

©2025 Codefinity. All Terms Reserved.

CNNs arbeiten ähnlich wie das **menschliche Gehirn bei der Bildverarbeitung**. Wenn wir etwas betrachten, senden unsere Augen Informationen an das Gehirn, das zunächst einfache Formen wie **Kanten und Farben** erkennt. In tieferen Schichten werden diese Elemente dann zusammengesetzt, um **Objekte, Gesichter oder ganze Szenen** zu erfassen. CNNs folgen demselben Prinzip: Sie beginnen mit einfachen Merkmalen und bauen darauf auf, um komplexe Objekte zu erkennen.

Wie unsere Augen sich auf bestimmte Bereiche konzentrieren, **verarbeiten auch CNNs Bilder in kleinen Abschnitten**, was ihnen ermöglicht, Muster unabhängig von ihrer Position zu erkennen. Im Gegensatz zum Menschen benötigen CNNs jedoch **tausende gelabelte Bilder** zum Lernen, während Menschen Objekte oft schon nach wenigen Beispielen erkennen können.

Überblick über die wichtigsten Komponenten: Convolution, Pooling, Aktivierung und vollverbundene Schichten

Ein CNN besteht aus mehreren Schichten, von denen jede eine spezifische Rolle bei der Bildverarbeitung übernimmt:

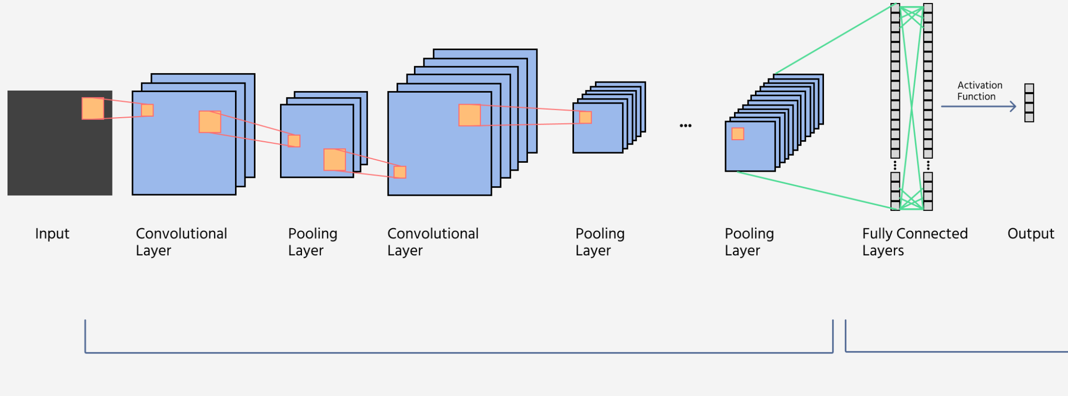
1. Faltungsschichten →

2. Aktivierungsfunktionen →

3. Pooling-Schichten →

4. Vollständig verbundene Schichten →

©2025 Codefinity. All Terms Reserved.



CNNs sind leistungsstark, weil sie automatisch Merkmale aus Bildern lernen können, anstatt dass Menschen jedes Detail programmieren müssen. Deshalb werden sie in **selbstfahrenden Autos, Gesichtserkennung, medizinischer Bildgebung** und vielen anderen realen Anwendungen eingesetzt.

©2025 Codefinity. All Terms Reserved.

🕮 Faltungsschichten

Verständnis von Convolution-Schichten

Convolution-Schichten bilden das Kernstück von **Convolutional Neural Networks (CNNs)**. Sie führen eine **Faltung** durch, bei der eine kleine Matrix, genannt **Filter (oder Kernel)**, über ein Bild gleitet, um **Kanten, Texturen und Formen** zu erkennen. Dies ermöglicht es CNNs, Bilder effizienter zu verarbeiten als herkömmliche Netzwerke.

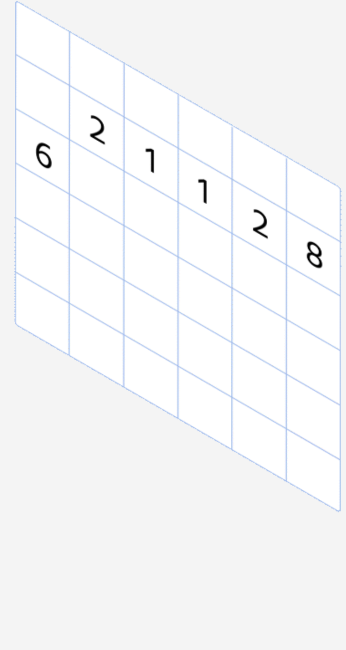
Anstatt ein gesamtes Bild auf einmal zu analysieren, teilen CNNs es in kleinere Abschnitte auf und erkennen Merkmale auf verschiedenen Ebenen. **Frühe Schichten** erkennen einfache Muster wie **Kanten**, während **tiefere Schichten** komplexe Strukturen erfassen.

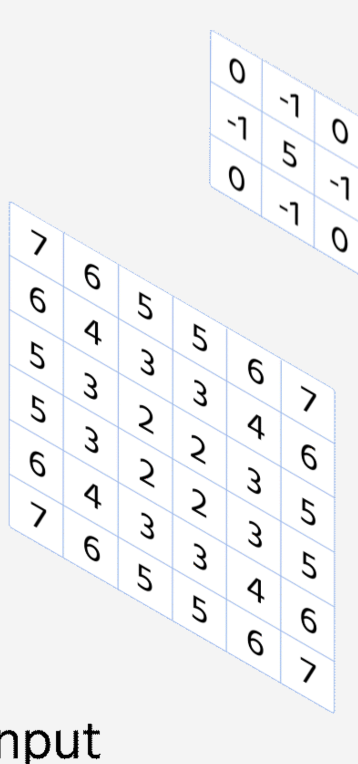
Funktionsweise der Faltung

Die Faltung beinhaltet das Verschieben eines **Filters (Kernels)** über ein Bild und umfasst folgende Schritte:

1. **Anwendung des Kernels** in der oberen linken Ecke des Bildes.
2. **Elementweise Multiplikation** zwischen Kernel und Pixelwerten.
3. **Summieren der Produkte** zur Erzeugung eines Ausgabepixels.
4. **Verschieben des Kernels** entsprechend dem **Stride** und Wiederholung des Vorgangs.
5. **Erzeugung einer Feature-Map**, die erkannte Muster hervorhebt.

©2025 Codefinity. All Terms Reserved.





Mehrere Filter ermöglichen es CNNs, verschiedene Merkmale wie vertikale Kanten, Kurven und Texturen zu erfassen.

Filter (Kerne):

Filter spielen eine entscheidende Rolle bei der Extraktion bedeutungsvoller Muster aus Bildern. Verschiedene Filtertypen sind auf die Erkennung unterschiedlicher Merkmale spezialisiert:

©2025 Codefinity. All Terms Reserved.

- **Kantenerkennungsfilter**: Erkennen Objektgrenzen durch das Aufspüren abrupter Intensitätsänderungen (z. B. Sobel-, Prewitt- und Laplace-Filter);

- **Texturfilter**: Erfassen sich wiederholende Muster wie Wellen oder Gitter (z. B. Gabor-Filter);

- **Schärfungsfilter**: Verstärken Bilddetails durch die Betonung hochfrequenter Komponenten;

- **Weichzeichnungsfilter**: Reduzieren Rauschen und glätten Bilder (z. B. Gaußscher Weichzeichner);

- **Relief-Filter**: Heben Kanten hervor und erzeugen einen 3D-Effekt durch Betonung der Tiefe.

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Gaussian Blur

$$\frac{1}{16} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Sharpen

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Edge Detection

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Jeder Filter wird darauf trainiert, spezifische Muster zu erkennen, und trägt zum Aufbau hierarchischer Merkmalsrepräsentationen in tiefen CNNs bei.

©2025 Codefinity. All Terms Reserved.

Faltungsschichten verwenden **denselben Filter über das gesamte Bild**, wodurch die Anzahl der Parameter reduziert und CNNs effizient werden. Spezialisierte lokal verbundene Schichten hingegen nutzen **verschiedene Filter** für unterschiedliche Bildbereiche, wenn dies erforderlich ist.

Durch das Stapeln von Faltungsschichten extrahieren CNNs detaillierte Muster und sind dadurch leistungsstark für Bildklassifikation, Objekterkennung und Aufgaben der maschinellen Bildverarbeitung.

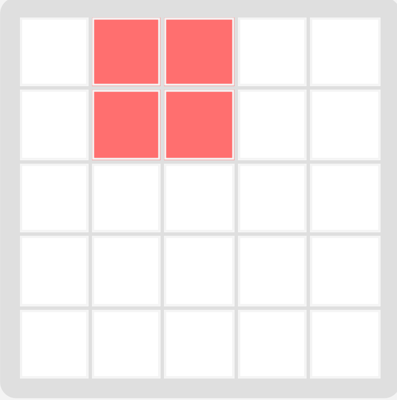
Hyperparameter:

- **Stride**: bestimmt, wie weit der Filter pro Schritt verschoben wird;
- **Padding**: fügt Pixel hinzu, um die Ausgabengröße zu steuern ("same padding" erhält die Größe, "valid padding" verringert sie);
- **Anzahl der Filter (Tiefe)**: mehr Filter verbessern die Merkmalsextraktion, erhöhen jedoch den Rechenaufwand.

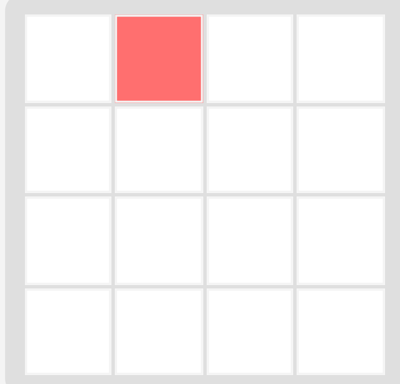
©2025 Codefinity. All Terms Reserved.

codefinity

Input size (5×5)



Output size (4×4)



©2025 Codefinity. All Terms Reserved.

Fahre mit der Maus über die Matrizen, um die Kernelposition zu ändern.

Hinweis

Example: For a `24×24` grayscale image using a `3×3` kernel with `64 filters`, the output size is `22×22×64`, computed as: $$ (W - F + 1) \times (H - F + 1) \times D =\= (24 - 3 + 1) \times (24 - 3 + 1) \times 64 =\= 22\times22\times64, $$

Where:

- H: size of the filter (kernel) = 3 (assuming a square 3×3 kernel);
- D: number of filters (depth of the output) = 64.

Vor dem nächsten Kapitel

Obwohl **Faltungsschichten** die **Ausgabegröße verringern** können, besteht ihr Hauptzweck in der **Merkmalextraktion und nicht in der Dimensionsreduktion**. **Pooling-Schichten** hingegen reduzieren explizit die Dimensionalität, während wichtige Informationen erhalten bleiben, was die Effizienz in tieferen Schichten gewährleistet.

©2025 Codefinity. All Terms Reserved.

📖 Pooling-Schichten

Zweck des Poolings

Pooling-Schichten spielen eine entscheidende Rolle in Convolutional Neural Networks (CNNs), indem sie die räumlichen Dimensionen von Feature-Maps reduzieren und gleichzeitig wesentliche Informationen beibehalten. Dies unterstützt:

- **Dimensionsreduktion**: Verringerung der Rechenkomplexität und des Speicherbedarfs;
- **Merkmalsbewahrung**: Erhalt der relevantesten Details für nachfolgende Schichten;
- **Überanpassungsvermeidung**: Reduzierung des Risikos, Rauschen und irrelevante Details zu erfassen;
- **Translationsinvarianz**: Erhöhung der Robustheit des Netzwerks gegenüber Variationen der Objektpositionen innerhalb eines Bildes.

Arten des Poolings

Pooling-Schichten arbeiten, indem sie ein kleines Fenster über die Feature-Maps bewegen und Werte auf unterschiedliche Weise aggregieren. Die wichtigsten Arten des Poolings sind:

Max-Pooling

- Wählt den **maximalen** Wert aus dem Fenster aus;
- Bewahrt dominante Merkmale und verwirft kleinere Variationen;
- Wird häufig verwendet, da es scharfe und markante Kanten erhält.

Average Pooling

©2025 Codefinity. All Terms Reserved.

- Berechnet den **Durchschnittswert** innerhalb des Fensters;
- Sorgt für eine geglättete Feature-Map durch Reduzierung extremer Schwankungen;
- Wird seltener als Max-Pooling verwendet, ist aber in bestimmten Anwendungen wie der Objekterkennung vorteilhaft.

Example: Kernel of size 2x2; stride=(2,2)

Max Pooling

Take **highest** value from the area covered by the kernel

Convolved Feature (4x4)

3	2	0	0
0	7	1	3
5	2	3	0
0	9	2	3

Max values

Output

7	

Average Pooling

Calculate the **average** value from the area covered by the kernel

Convolved Feature (4x4)

3	2	0	0
0	7	1	3
5	2	3	0
0	9	2	3

Average values

Output

3	

Globales Pooling

©2025 Codefinity. All Terms Reserved.

- Anstelle eines kleinen Fensters wird über die **gesamte Feature-Map** gepoolt;
- Es gibt zwei Arten von globalem Pooling:
- **Globales Max-Pooling**: Nimmt den Maximalwert über die gesamte Feature-Map;
- **Globales Durchschnitts-Pooling**: Berechnet den Durchschnitt aller Werte in der Feature-Map.
- Häufig in vollständig konvolutionalen Netzwerken für Klassifizierungsaufgaben verwendet.

Hinweis

Beim Pooling wird kein Kernel auf die Eingangsdaten angewendet, sondern die **Informationen werden durch eine mathematische Operation** (Max oder Durchschnitt) vereinfacht.

Vorteile des Poolings in CNNs

Pooling verbessert die Leistung von CNNs auf verschiedene Weise:

- **Translationsinvarianz**: Kleine Verschiebungen in einem Bild verändern die Ausgabe nicht drastisch, da das Pooling sich auf die wichtigsten Merkmale konzentriert;
- **Reduzierung von Overfitting**: Vereinfacht Merkmalskarten und verhindert übermäßiges Auswendiglernen der Trainingsdaten;
- **Verbesserte Recheneffizienz**: Die Verkleinerung der Merkmalskarten beschleunigt die Verarbeitung und reduziert den Speicherbedarf.

Pooling-Schichten sind ein grundlegender Bestandteil von CNN-Architekturen und stellen sicher, dass Netzwerke bedeutungsvolle Informationen extrahieren und gleichzeitig Effizienz und Generalisierungsfähigkeit bewahren.

©2025 Codefinity. All Terms Reserved.

Flattening

Übergang von der Merkmalsextraktion zur Klassifikation

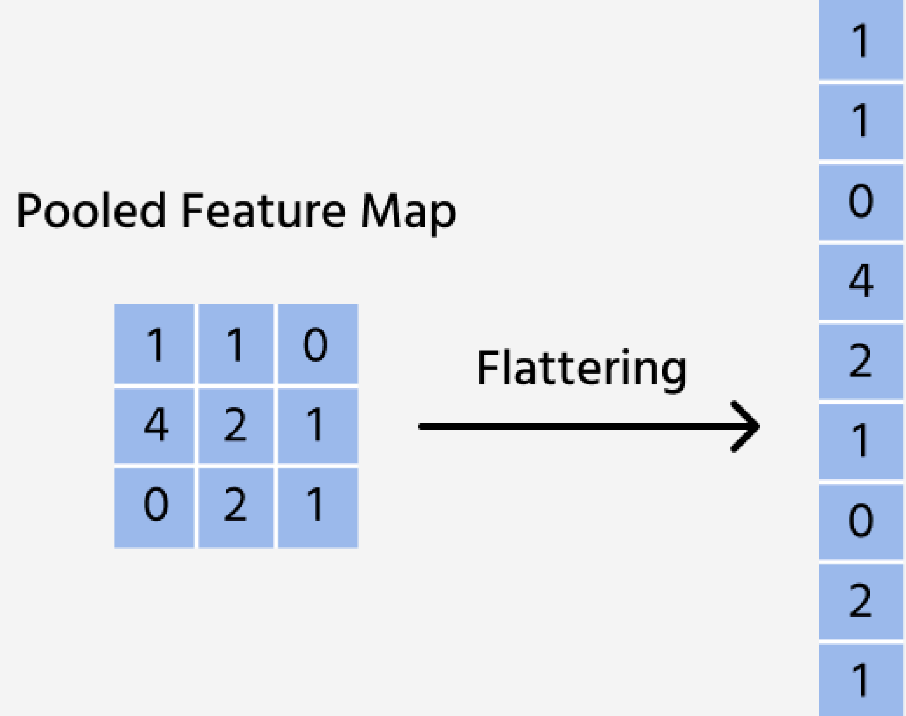
Nachdem Faltungs- und Pooling-Schichten wesentliche Merkmale aus einem Bild extrahiert haben, folgt in einem Convolutional Neural Network (CNN) der Schritt der Klassifikation. Da vollständig verbundene Schichten eine eindimensionale Eingabe benötigen, müssen die mehrdimensionalen Merkmalskarten in ein für die Klassifikation geeignetes Format umgewandelt werden.

Umwandlung von Merkmalskarten in einen 1D-Vektor

Flattening ist der Prozess, bei dem die Ausgaben der Faltungs- und Pooling-Schichten in einen einzigen langen Vektor umgeformt werden. Hat eine Merkmalskarte die Dimensionen `X × Y × Z`, wandelt Flattening diese in ein `1D array` der Länge `X × Y × Z` um.

Beispielsweise wird eine finale Merkmalskarte mit den Dimensionen `7 × 7 × 64` durch Flattening in einen `(7 × 7 × 64) = 3136-dimensional`-dimensionalen Vektor umgewandelt. Dadurch können die vollständig verbundenen Schichten die extrahierten Merkmale effizient verarbeiten.

©2025 Codefinity. All Terms Reserved.



Bedeutung des Flattening vor der Übergabe an vollverbundene Schichten

Vollverbundene Schichten arbeiten mit einer Standardstruktur neuronaler Netze, bei der jedes Neuron mit jedem Neuron der nächsten Schicht verbunden ist. Ohne Flattening kann das Modell die räumliche Struktur der Feature-Maps nicht korrekt interpretieren. Flattening gewährleistet:

- **Korrekte Überleitung** von der Merkmalsextraktion zur Klassifikation;
- **Nahtlose Integration** mit vollverbundenen Schichten;
- **Effizientes Lernen** durch Erhaltung der extrahierten Muster für die finale Entscheidungsfindung.

Durch das Flattening der Feature-Maps können CNNs hochrangige Merkmale, die während der Faltung und des Poolings gelernt wurden, nutzen und so eine präzise Klassifikation von Objekten innerhalb eines Bildes ermöglichen.

©2025 Codefinity. All Terms Reserved.

Warum Aktivierungsfunktionen in CNNs entscheidend sind

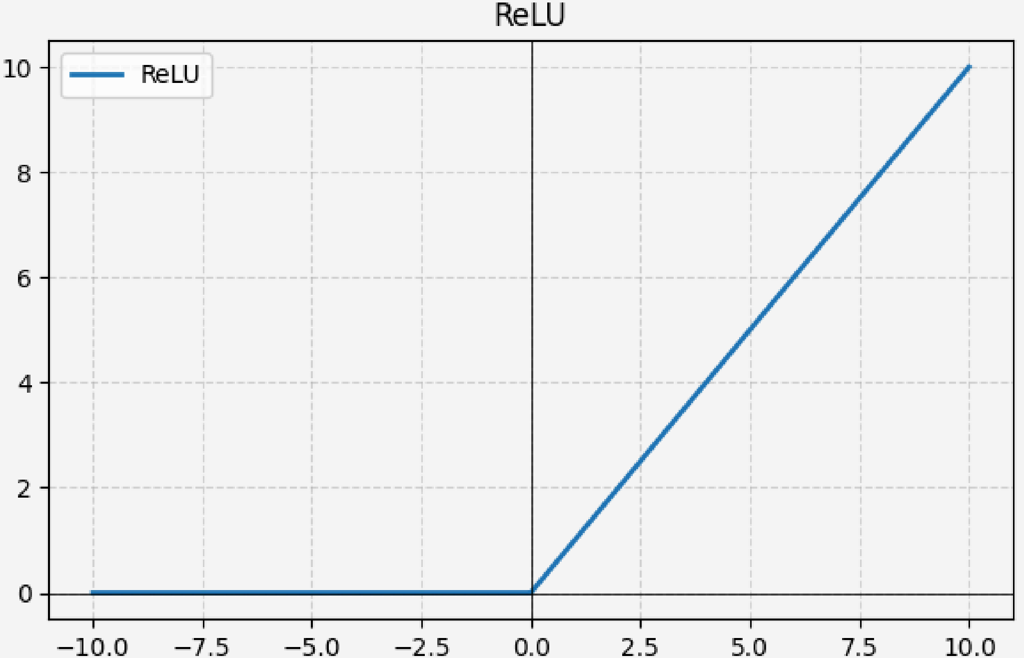
Aktivierungsfunktionen führen Nichtlinearität in CNNs ein und ermöglichen es ihnen, komplexe Muster zu erlernen, die über die Fähigkeiten eines einfachen linearen Modells hinausgehen. Ohne Aktivierungsfunktionen hätten CNNs Schwierigkeiten, komplexe Zusammenhänge in den Daten zu erkennen, was ihre Effektivität bei der Bildklassifikation und -erkennung einschränkt. Die Wahl der richtigen Aktivierungsfunktion beeinflusst Trainingsgeschwindigkeit, Stabilität und Gesamtleistung.

Häufig verwendete Aktivierungsfunktionen

- **ReLU (Rectified Linear Unit):** Die am häufigsten verwendete Aktivierungsfunktion in CNNs. Sie gibt nur positive Werte weiter und setzt alle negativen Eingaben auf Null, was sie recheneffizient macht und das Verschwinden von Gradienten verhindert. Allerdings können einige Neuronen durch das "Dying ReLU"-Problem inaktiv werden;

$$f(x) = \max(0, x)$$

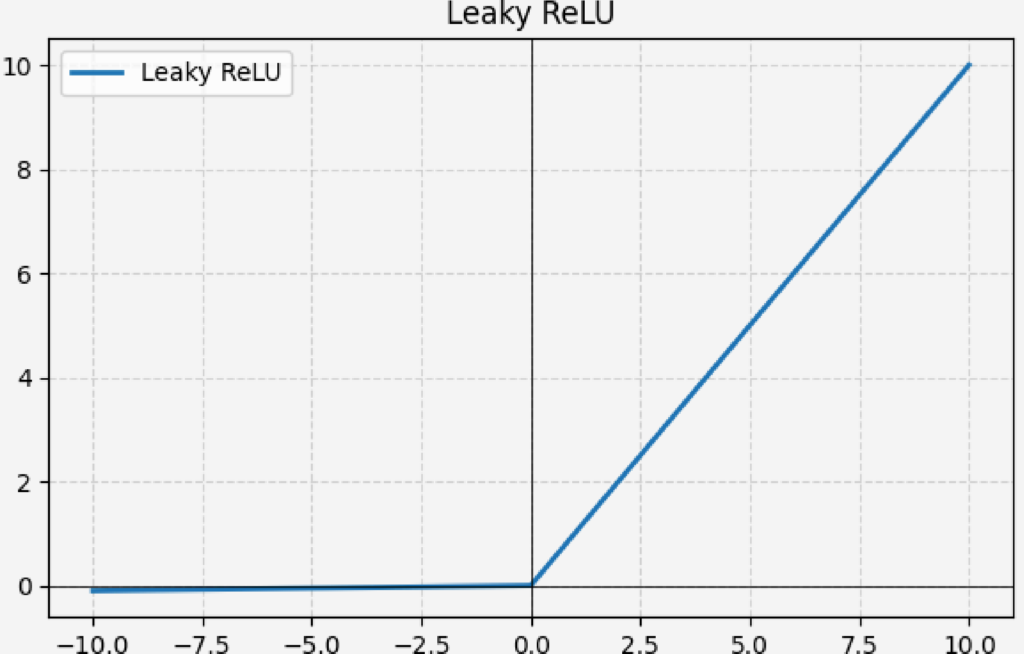
©2025 Codefinity. All Terms Reserved.



- **Leaky ReLU:** eine Variante der ReLU, die kleine negative Werte zulässt, anstatt sie auf Null zu setzen, wodurch inaktive Neuronen vermieden und der Gradientenfluss verbessert werden;

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$

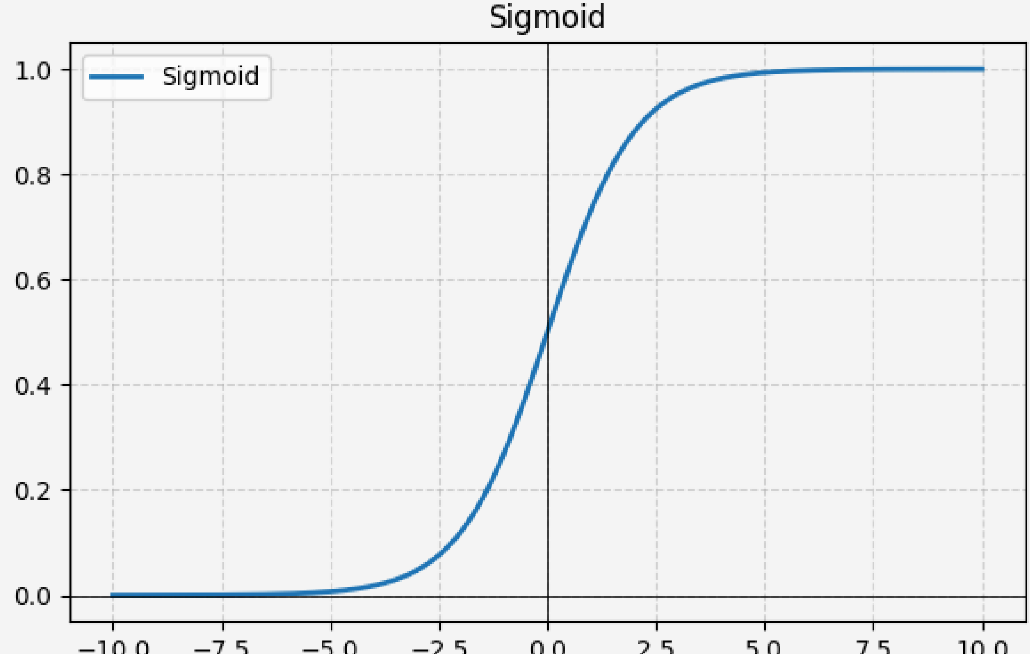
©2025 Codefinity. All Terms Reserved.



- **Sigmoid:** komprimiert Eingabewerte in einen Bereich zwischen 0 und 1 und ist daher nützlich für binäre Klassifikation. Allerdings tritt bei tiefen Netzwerken das Problem verschwindender Gradienten auf;

$$f(x) = \frac{1}{1 + e^{-x}}$$

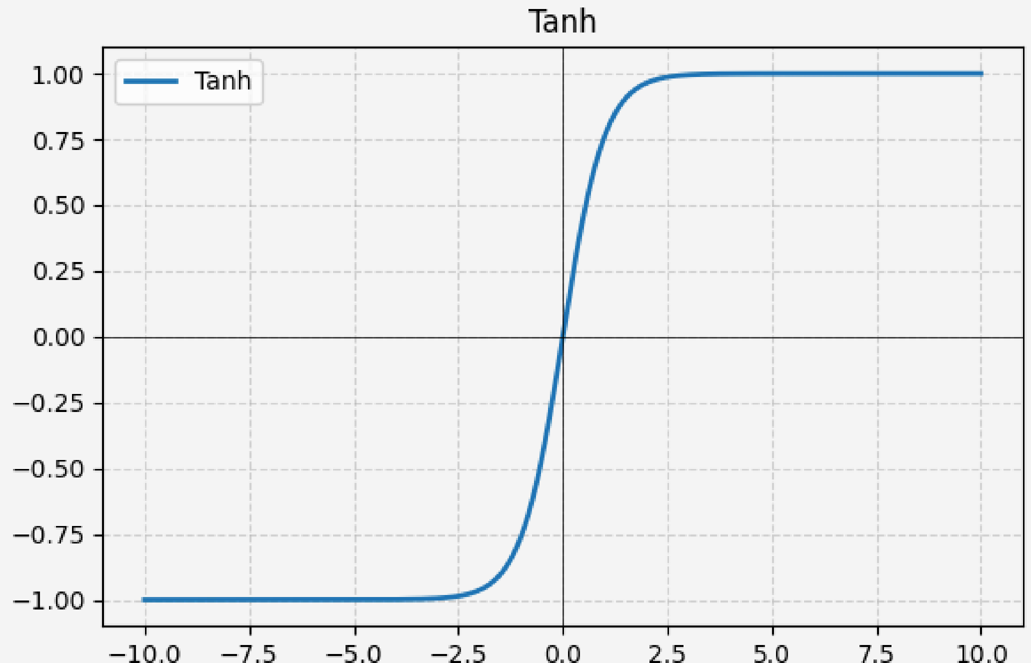
©2025 Codefinity. All Terms Reserved.



- **Tanh:** ähnlich wie Sigmoid, gibt jedoch Werte zwischen -1 und 1 aus und zentriert die Aktivierungen um Null;

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

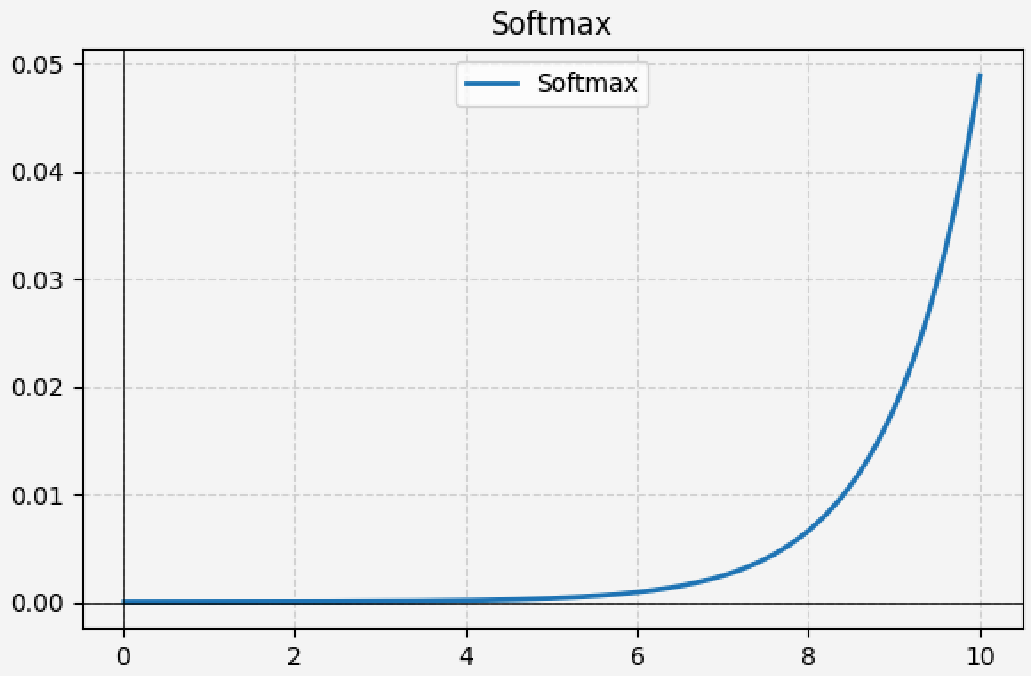
©2025 Codefinity. All Terms Reserved.



- **Softmax:** Wird typischerweise in der letzten Schicht für Mehrklassenklassifikation verwendet. Softmax wandelt die Rohwerte des Netzwerks in Wahrscheinlichkeiten um und stellt sicher, dass deren Summe eins ergibt, was die Interpretierbarkeit verbessert.

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

©2025 Codefinity. All Terms Reserved.



Auswahl der richtigen Aktivierungsfunktion

ReLU ist aufgrund ihrer Effizienz und starken Leistung die Standardwahl für versteckte Schichten, während **Leaky ReLU** vorzuziehen ist, wenn das Problem der Inaktivität von Neuronen auftritt. **Sigmoid** und **Tanh** werden in tiefen CNNs meist vermieden, können jedoch in bestimmten Anwendungen weiterhin nützlich sein. **Softmax** bleibt für Aufgaben der Mehrklassenklassifikation unverzichtbar und sorgt für klare, auf Wahrscheinlichkeiten basierende Vorhersagen.

Die Auswahl der passenden Aktivierungsfunktion ist **entscheidend** für die Optimierung der CNN-Leistung, das Gleichgewicht zwischen Effizienz und die Vermeidung von Problemen wie **verschwindenden oder explodierenden Gradienten**. Jede Funktion trägt auf einzigartige Weise dazu bei, wie ein Netzwerk visuelle Daten verarbeitet und daraus lernt.

©2025 Codefinity. All Terms Reserved.

📖 Überblick Über Beliebte CNN-Modelle

Convolutional Neural Networks (CNNs) haben sich erheblich weiterentwickelt, wobei verschiedene Architekturen die Genauigkeit, Effizienz und Skalierbarkeit verbessert haben. Dieses Kapitel behandelt fünf zentrale CNN-Modelle, die das Deep Learning maßgeblich geprägt haben: **LeNet**, **AlexNet**, **VGGNet**, **ResNet** und **InceptionNet**.

LeNet: Die Grundlage der CNNs

Eine der ersten Architekturen für Convolutional Neural Networks, 1998 von Yann LeCun für die Erkennung handgeschriebener Ziffern vorgeschlagen. Sie legte das Fundament für moderne CNNs durch die Einführung zentraler Komponenten wie Convolutions, Pooling und vollständig verbundener Schichten. Weitere Informationen zum Modell finden Sie in der 🔗 Dokumentation.

Wichtige Architekturmerkmale

Convolutional layers + Tanh
Two convolutional layers with tanh activation to extract spatial features.

Average pooling
Dimensionality reduction applied after each convolutional layer using average pooling.

Fully connected layers
One or two fully connected layers to interpret features and perform classification.

Output layer (Softmax)
Final layer used to classify input into categories, such as digits 0–9.

AlexNet: Durchbruch im Deep Learning

©2025 Codefinity. All Terms Reserved.

Eine bahnbrechende CNN-Architektur, die den ImageNet-Wettbewerb 2012 gewann. AlexNet zeigte, dass tiefe Convolutional Networks herkömmliche Machine-Learning-Methoden bei der großskaligen Bildklassifikation deutlich übertreffen können. Es führte Innovationen ein, die heute zum Standard im modernen Deep Learning gehören. Weitere Informationen zum Modell finden Sie in der 🔗 Dokumentation.

Wichtige Architekturmerkmale

Deep convolutional stack

Five convolutional layers with ReLU activation functions to enable faster training.

Max pooling

Applied after certain convolutional layers to reduce spatial dimensions and computation.

Dropout regularization

Used in fully connected layers to mitigate overfitting during training.

GPU acceleration

Model was trained across two GPUs in parallel, enabling deeper and more complex architectures.

Fully connected layers

Three dense layers at the end, culminating in a softmax output for classification.

VGGNet: Tiefere Netzwerke mit einheitlichen Filtern

Entwickelt von der Visual Geometry Group in Oxford, betonte VGGNet Tiefe und Einfachheit durch die Verwendung einheitlicher 3×3-Faltungskerne. Es zeigte, dass das Stapeln kleiner Filter in tiefen Netzwerken die Leistung erheblich steigern kann, was zu weit verbreiteten Varianten wie VGG-16 und VGG-19 führte. Weitere Informationen zum Modell finden Sie in der 🔗 Dokumentation.

©2025 Codefinity. All Terms Reserved.

Wichtige Architekturmerkmale

Uniform 3x3 convolutions

Stacked small filters to extract complex features while maintaining a consistent structure.

Deep architecture

16 or 19 layers in VGG-16 and VGG-19, respectively, showing the power of network depth.

Max pooling

Applied after groups of convolutional layers to reduce spatial dimensions.

Fully connected layers

Three dense layers before the final softmax output.

High parameter count

Large number of weights, leading to strong performance but high memory and computation demands.

ResNet: Lösung des Tiefenproblems

ResNet (Residual Networks), eingeführt von Microsoft im Jahr 2015, adressierte das Problem des verschwindenden Gradienten, das beim Training sehr tiefer Netzwerke auftritt. Traditionelle tiefe Netzwerke haben Schwierigkeiten mit der Trainingseffizienz und Leistungsverschlechterung, aber ResNet überwand dieses Problem durch **Skip Connections** (residuales Lernen). Diese Abkürzungen ermöglichen es, Informationen bestimmte Schichten zu umgehen, wodurch sichergestellt wird, dass Gradienten effektiv weitergegeben werden. ResNet-Architekturen wie **ResNet-50** und **ResNet-101** ermöglichten das Training von Netzwerken mit Hunderten von Schichten und verbesserten die Genauigkeit der Bildklassifikation erheblich. Weitere Informationen zum Modell finden Sie in der Dokumentation.

Wichtige Architekturmerkmale

©2025 Codefinity. All Terms Reserved.

Residual blocks

Pairs of convolutional layers with identity skip connections to enable gradient flow across layers.

Deep architecture

Variants with 50, 101, or more layers that remain trainable thanks to residual learning.

Batch normalization + ReLU

Applied within residual blocks to stabilize and accelerate training.

Global average pooling

Replaces fully connected layers before the final classification layer to reduce overfitting.

Identity mapping

Allows the model to learn residual functions instead of direct mappings, improving convergence.

InceptionNet: Multi-Skalen-Merkmalextraktion

InceptionNet (auch bekannt als **GoogLeNet**) basiert auf dem Inception-Modul und bietet eine **tiefe, aber effiziente** Architektur. Anstatt Schichten sequenziell zu stapeln, verwendet InceptionNet **parallele Pfade**, um Merkmale auf verschiedenen Ebenen zu extrahieren. Weitere Informationen zum Modell finden Sie in der Dokumentation.

Wichtige Optimierungen umfassen:

- **Faktorisierte Faltungen** zur Reduzierung des Rechenaufwands;
- **Hilfsklassifikatoren** in Zwischenschichten zur Verbesserung der Trainingsstabilität;
- **Globales Durchschnittspooling** anstelle vollständig verbundener Schichten, wodurch die Anzahl der Parameter reduziert wird, während die Leistung erhalten bleibt.

©2025 Codefinity. All Terms Reserved.

Diese Struktur ermöglicht es InceptionNet, **tiefer als frühere CNNs wie VGG** zu sein, ohne die Rechenanforderungen drastisch zu erhöhen.

Wichtige Architekturmerkmale

Inception modules
Parallel convolutional and pooling operations for multi-scale feature extraction.

Factorized convolutions
Decomposed larger convolutions into smaller ones (e.g., 5×5 into two 3×3) to reduce computation.

Auxiliary classifiers
Intermediate softmax outputs during training to help gradients propagate in deep networks.

Global average pooling
Used instead of fully connected layers at the end, significantly reducing parameter count.

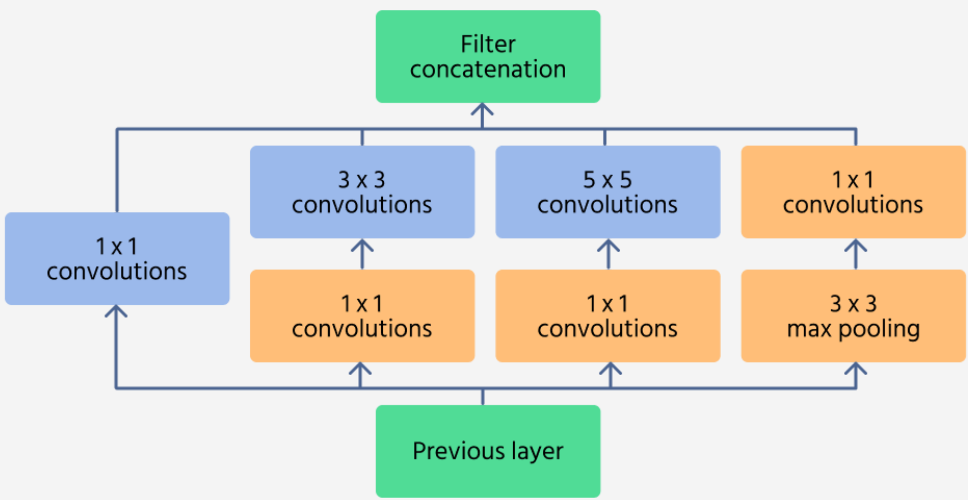
Deep but efficient
Achieved high performance with fewer parameters and lower memory usage compared to VGG.

Inception-Modul

Das **Inception-Modul** ist die zentrale Komponente von InceptionNet und wurde entwickelt, um Merkmale auf mehreren Skalen effizient zu erfassen. Anstatt nur eine einzelne Faltung anzuwenden, **verarbeitet das Modul die Eingabe parallel mit mehreren Filtergrößen (`1×1, 3×3, 5×5`)**. Dadurch kann das Netzwerk sowohl **feine Details als auch große Muster** in einem Bild erkennen.

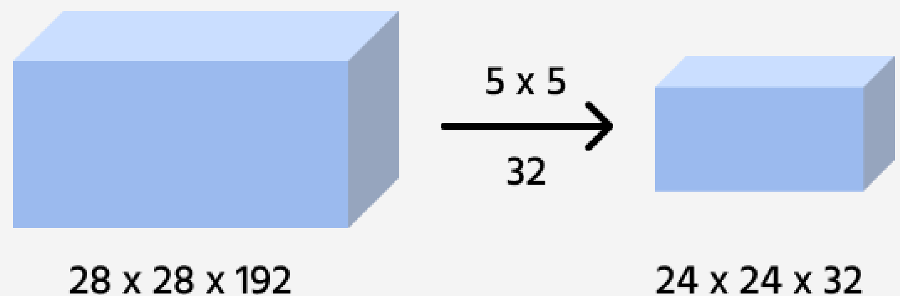
©2025 Codefinity. All Terms Reserved.

Um die Rechenkosten zu senken, werden `1×1 convolutions` vor der Anwendung größerer Filter eingesetzt. Diese **reduzieren die Anzahl der Eingangskanäle** und machen das Netzwerk effizienter. Zusätzlich helfen **Max-Pooling-Schichten** innerhalb des Moduls, wesentliche Merkmale zu erhalten und gleichzeitig die Dimensionalität zu kontrollieren.



Beispiel

Betrachten Sie ein Beispiel, um zu veranschaulichen, wie **die Reduzierung der Dimensionen die Rechenlast verringert**. Angenommen, es sollen `28 × 28 × 192 input feature maps` mit `5 × 5 × 32 filters` gefaltet werden. Dieser Vorgang würde etwa **120,42 Millionen** Berechnungen erfordern.

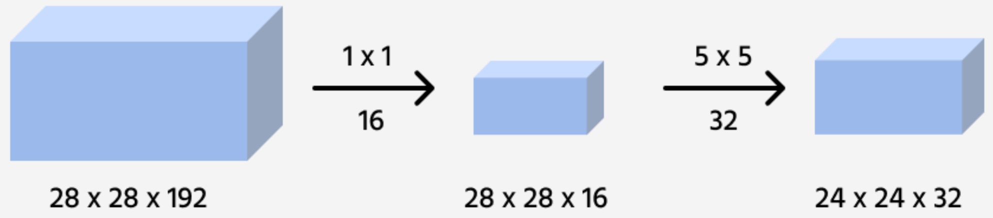


©2025 Codefinity. All Terms Reserved.

Hinweis

Number of operations = (28*28*192) * (5*5*32) = 120,422,400 operations

Führen wir die Berechnungen erneut durch, aber dieses Mal fügen wir eine `1×1 convolutional layer` hinzu, bevor wir die `5×5 convolution` auf die gleichen Eingabe-Feature-Maps anwenden.



Hinweis

Number of operations for 1x1 convolution = (28*28*192) * (1*1*16) = 2,408,448 operations

Number of operations for 5x5 convolution = (28*28*16) * (5*5*32) = 10,035,200 operations

Total number of operations 2,408,448 + 10,035,200 = 12,443,648 operations

©2025 Codefinity. All Terms Reserved.

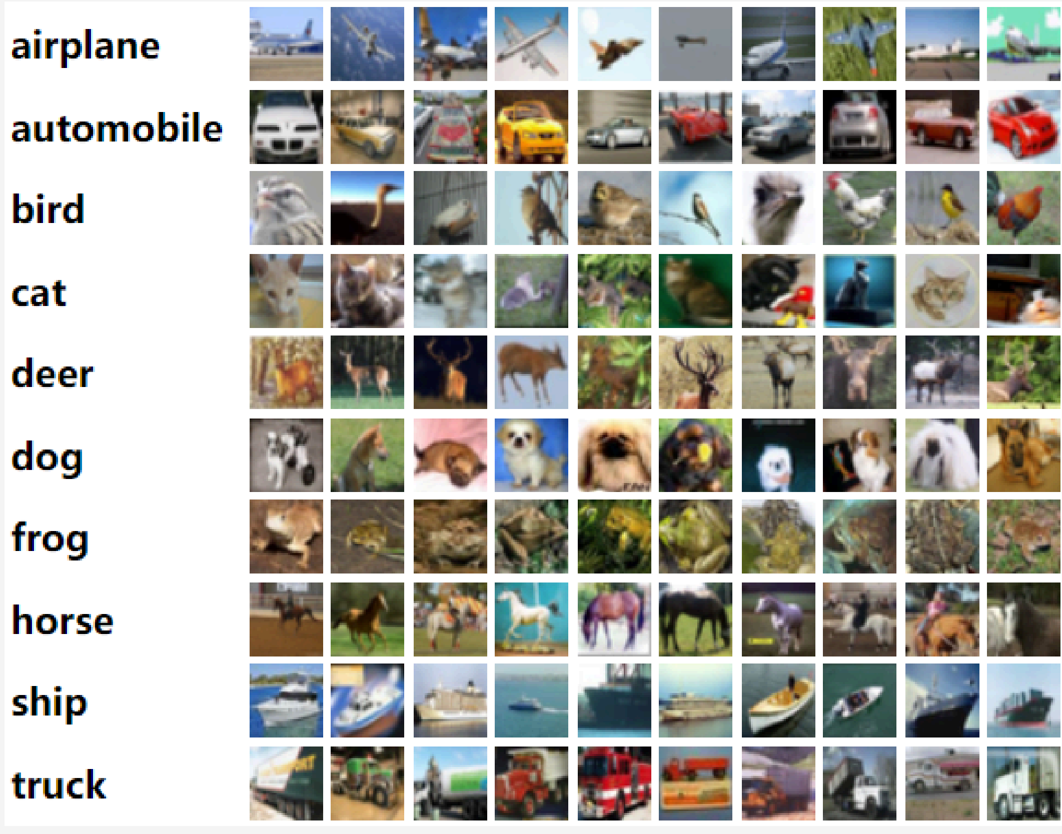
Jede dieser CNN-Architekturen hat eine entscheidende Rolle bei der Weiterentwicklung der Computer Vision gespielt und Anwendungen in **Gesundheitswesen**, **autonomen Systemen**, **Sicherheit** und **Echtzeit-Bildverarbeitung** beeinflusst. Von den grundlegenden Prinzipien von **LeNet** bis zur Multi-Skalen-Merkmalextraktion von **InceptionNet** haben diese Modelle die Grenzen des Deep Learning kontinuierlich erweitert und den Weg für noch fortschrittlichere Architekturen in der Zukunft geebnet.

©2025 Codefinity. All Terms Reserved.

🕮 Herausforderung: Aufbau eines Convolutional Neural Network

Convolutional Neural Networks (CNNs) werden aufgrund ihrer Fähigkeit, hierarchische Merkmale zu extrahieren, häufig für die Bildklassifikation eingesetzt. In dieser Aufgabe wird ein **VGG-ähnliches CNN** mit TensorFlow und Keras auf dem `CIFAR-10`-Datensatz implementiert und trainiert. Der Datensatz besteht aus **60.000 Bildern** (`32×32×3`), die zu **10 verschiedenen Klassen** gehören, darunter Flugzeuge, Autos, Vögel, Katzen und weitere.

Dieses Projekt führt durch die **Ladung des Datensatzes, die Vorverarbeitung der Bilder, die Definition des CNN-Modells, das Training sowie die Auswertung der Modellleistung**.



1. Datenvorverarbeitung für CNNs

©2025 Codefinity. All Terms Reserved.

Vor dem Training eines CNN ist die Vorverarbeitung der Daten ein entscheidender Schritt, um eine bessere Leistung und schnellere Konvergenz zu gewährleisten. Gängige Vorverarbeitungsmethoden sind:

- **Normalisierung**: Bei dieser Methode werden die Pixelwerte der Bilder von einem Bereich zwischen **0 und 255** auf einen Bereich zwischen **0 und 1** skaliert. Dies wird häufig als `x_train / 255.0, x_test / 255.0` umgesetzt;

- **One-Hot-Encoding**: Labels werden häufig in **One-Hot-codierte Vektoren** für Klassifikationsaufgaben umgewandelt. Dies erfolgt typischerweise mit der Funktion `keras.utils.to_categorical`, die ganzzahlige Labels **(z. B. 0, 1, 2, usw.)** in einen One-Hot-codierten Vektor transformiert, wie zum Beispiel `[1, 0, 0, 0]` für ein Klassifikationsproblem mit **4 Klassen**.

2. Aufbau der CNN-Architektur

Eine CNN-Architektur besteht aus **mehreren Schichten**, die verschiedene Aufgaben zur Merkmalsextraktion und Vorhersage übernehmen. Zentrale CNN-Schichten können wie folgt implementiert werden:

Faltungsschicht (Conv2D)

```
keras.layers.Conv2D(filters, kernel_size, activation='relu',
padding='same', input_shape=(height, width, channels))
```

Hinweis

Den Parameter `input_shape` muss man nur in der Eingabeschicht angeben.

©2025 Codefinity. All Terms Reserved.

Pooling-Schicht (MaxPooling2D)

```
keras.layers.MaxPooling2D(pool_size=(2, 2))
```

Flatten-Schicht

```
keras.layers.Flatten()
```

Dense-Schicht

```
layers.Dense(units=512, activation='relu')
layers.Dense(10, activation='softmax')
```

Hinweis

Die finale Dense-Schicht besitzt in der Regel eine Anzahl von Einheiten, die **der Anzahl der Klassen entspricht**, und verwendet eine **Softmax-Aktivierungsfunktion**, um eine Wahrscheinlichkeitsverteilung über die Klassen auszugeben.

3. Modellkompilierung

Nach der Definition der Architektur muss das Modell kompiliert werden. In diesem Schritt werden die Verlustfunktion, der Optimierer und die Metriken festgelegt, die das Modell während des Trainings steuern. Die folgenden Methoden werden häufig bei CNNs verwendet:

Optimierer (Adam)

©2025 Codefinity. All Terms Reserved.

Der Optimierer passt die Gewichte des Modells an, um die Verlustfunktion zu minimieren. Der Adam-Optimierer ist aufgrund seiner Effizienz und der Fähigkeit, die Lernrate während des Trainings anzupassen, sehr beliebt.

```
keras.optimizers.Adam()
```

Verlustfunktion (Categorical Crossentropy)

Für Mehrklassenklassifikation wird typischerweise die kategorische Kreuzentropie als Verlustfunktion verwendet. Dies kann wie folgt implementiert werden:

```
keras.losses.CategoricalCrossentropy()
```

Metriken

Die Modellleistung wird bei Klassifikationsaufgaben mit Metriken wie Genauigkeit, Präzision, Recall usw. überwacht. Diese können wie folgt definiert werden:

```
metrics = [
    keras.metrics.Accuracy(),
    keras.metrics.Precision(),
    keras.metrics.Recall()
]
```

Kompilieren

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy', # Or
keras.losses.CategoricalCrossentropy()
              metrics=metrics)
```

©2025 Codefinity. All Terms Reserved.

codefinity

4. Training des Modells

Das Training eines CNN umfasst das Einspeisen der Eingabedaten in das Netzwerk, das Berechnen des Verlusts und das Aktualisieren der Gewichte mittels Backpropagation. Der Trainingsprozess wird durch die folgenden zentralen Methoden gesteuert:

- **Modell anpassen**: Die Methode `fit()` wird verwendet, um das Modell zu trainieren. Diese Methode erhält die Trainingsdaten, die Anzahl der Epochen und die Batch-Größe. Zusätzlich kann ein optionaler Validierungsanteil angegeben werden, um die Leistung des Modells auf unbekannten Daten während des Trainings zu bewerten:

```
history = model.fit(x_train, y_train, epochs=10, batch_size=32,
validation_split=0.2)
```

- **Batch-Größe und Epochen**: Die Batch-Größe bestimmt die **Anzahl der Stichproben, die verarbeitet werden, bevor die Modellgewichte aktualisiert werden**, und die Anzahl der Epochen gibt an, **wie oft der gesamte Datensatz durch das Modell geleitet wird**.

5. Auswertung

Klassifikationsbericht

`sklearn.metrics.classification_report()` vergleicht **wahre und vorhergesagte** Werte aus dem Testdatensatz. Es enthält **Präzision, Recall und F1-Score** für jede Klasse. Die Methoden benötigen jedoch nur Klassenlabels, daher nicht vergessen, die Vektoren zurück in Labels umzuwandeln (`[0,0,1,0]` -> `2`):

©2025 Codefinity. All Terms Reserved.

codefinity

```
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred,axis = 1)
y_test_classes = np.argmax(y_test, axis = 1)

report = classification_report(y_test_classes, y_pred_classes,
target_names=class_names)
print(report)
```

Auswertung

Nachdem das Modell trainiert wurde, wird es auf dem Testdatensatz ausgewertet, um seine Generalisierungsfähigkeit zu beurteilen. Die Auswertung liefert Metriken, die bereits in der `.compile()` -Methode erwähnt wurden. Die Auswertung erfolgt mit `.evaluate()` :

```
results = model.evaluate(x_test, y_test, verbose=2, return_dict=True)
```

Konfusionsmatrix

Um weitere Einblicke in die Leistung des Modells zu erhalten, kann die **Konfusionsmatrix** visualisiert werden. Sie zeigt **die wahren Positiven, falschen Positiven, wahren Negativen und falschen Negativen Vorhersagen** für jede Klasse. Die Konfusionsmatrix kann mit TensorFlow berechnet werden:

```
confusion_mtx = tf.math.confusion_matrix(y_test_classes,
y_pred_classes)
```

Diese Matrix kann anschließend mithilfe von Heatmaps visualisiert werden, um zu beobachten, wie gut das Modell für jede Klasse arbeitet:

©2025 Codefinity. All Terms Reserved.

codefinity

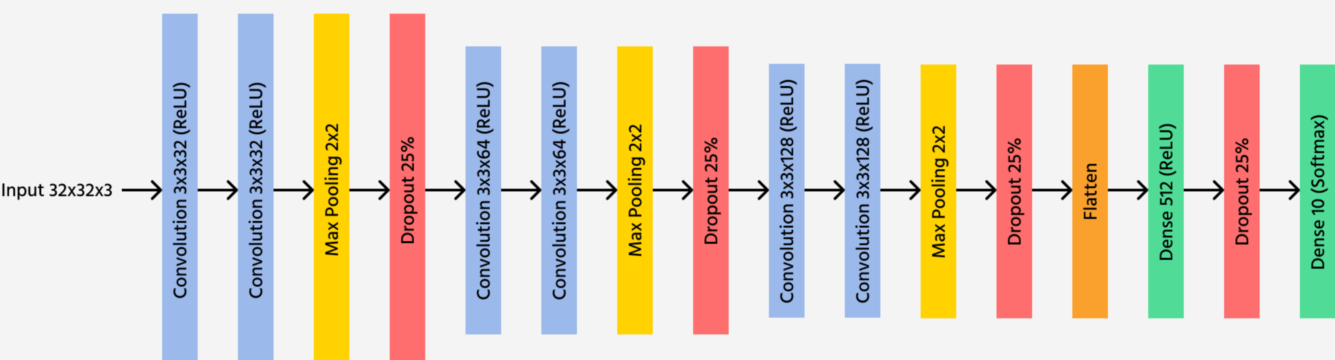
```
plt.figure(figsize=(12, 9))
c = sns.heatmap(confusion_mtx, annot=True, fmt='g')
c.set(xticklabels=class_names, yticklabels=class_names)

plt.show()
```

Aufgabe

1. Laden und Vorverarbeiten des Datensatzes - Importieren des CIFAR-10-Datensatzes aus Keras; - Normalisierung der Pixelwerte auf den Bereich `[0,1]` für bessere Konvergenz; - Umwandlung der Klassenlabels in das `one-hot encoded`-Format für die kategoriale Klassifikation.

2. Definition des CNN-Modells

Implementierung einer **VGG-ähnlichen CNN-Architektur** mit den folgenden Schlüsselschichten:



Convolutional Layers: - Kernelgröße: `3×3`; - Aktivierungsfunktion: `ReLU`; - Padding: `'same'`.

Pooling Layers: - Pooling-Typ: `max pooling`; - Pooling-Größe: `2×2`.

Dropout-Schichten (Verhindern von Overfitting durch zufälliges Deaktivieren von Neuronen): - Dropout-Rate: `25%`.

©2025 Codefinity. All Terms Reserved.

Flatten-Schicht – Umwandlung von 2D-Feature-Maps in einen 1D-Vektor für die Klassifikation.

Vollständig verbundene Schichten – Dense-Schichten für die finale Klassifikation, mit einer ReLU- oder Softmax-Ausgabeschicht.

Kompilieren des Modells mit: - `Adam optimizer` (für effizientes Lernen); - `Categorical cross-entropy` als Verlustfunktion (für Mehrklassenklassifikation); - `Accuracy metric` zur Leistungsbewertung (die Klassen sind ausgeglichen, weitere Metriken können optional hinzugefügt werden).

3. Training des Modells - Festlegen der Parameter `epochs` und `batch_size` für das Training (z. B. `epochs=20, batch_size=64`); - Festlegen des Parameters `validation_split`, um den **Prozentsatz der Trainingsdaten als Validierungsdaten** zu definieren, um die Modellleistung auf unbekannten Bildern zu verfolgen; - Speichern des Trainingsverlaufs zur Visualisierung von **Genauigkeits- und Verlusttrends**.

4. Auswertung und Visualisierung der Ergebnisse - Testen des Modells auf den **CIFAR-10-Testdaten** und Ausgeben der Genauigkeit; - Plotten von **Trainingsverlust vs. Validierungsverlust** zur Überprüfung auf Overfitting; - Plotten von **Trainingsgenauigkeit vs. Validierungsgenauigkeit** zur Sicherstellung des Lernfortschritts.

🔗 COLAB CNN PROJEKT

©2025 Codefinity. All Terms Reserved.