AI-Document-Checker

Alin Bicer
Berre Nur Celik
Egesel Bozgeyik

(Juli 2025)

1. Vergleichung der Modelle

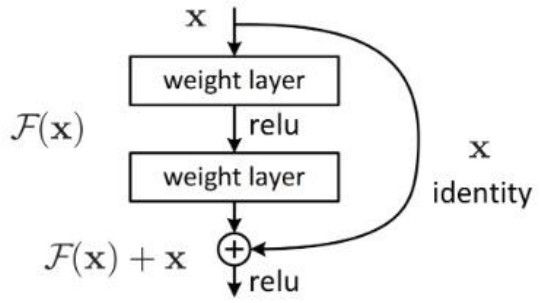
1.a. ResNet

Einleitung auf das Modell

Bis in die 2010er Jahre war Deep Learning noch nicht so weit entwickelt wie heute. Die Gründe dafür waren der Mangel an verfügbaren Daten und Hardware sowie die Schwierigkeit, Modelle zu trainieren. Im Jahr 2012 schufen die Entwickler AlexNet. AlexNet gewann den ImageNet-Wettbewerb mit einem Erdrutschsieg. Dieser 8-Schicht-CNN war damals revolutionär. Im Jahr 2014 wurden auch VGGNet und GoogLeNet eingeführt. Sie brachten das Deep Learning bei der Überwindung einiger Probleme wie der Überanpassung ein Stück weiter. Allerdings funktionierte die Erhöhung der Tiefe nicht immer, und in vielen Fällen schnitten die Modelle mit tieferen Schichten nicht besser ab. Im Jahr 2015 stellten 4 Forscher von Microsoft Research Asia ResNet vor. Dieses Modell, das für Residual Network steht, kann bis zu 152 Schichten umfassen und löste das Problem der Leistungsverschlechterung mit zunehmender Anzahl von Schichten in anderen Modellen. Mit diesem Modell gewannen die Forscher den 1. Platz im ImageNet-Wettbewerb 2015. Mit der Einführung dieses Modells wurden tiefere Modelle durch residuales Lernen möglich. In den folgenden Jahren bildete ResNet den Grundstein für viele neu entwickelte Modelle.

Im Gegensatz zu früheren Ansätzen geht es bei ResNet, wenn CNN zwischen den Schichten operiert, darum, wie sehr sich die Ausgabe von der Eingabe unterscheidet, um das Ergebnis der Funktionsoperation zu erreichen. Es lässt die Eingabe so, wie sie ist, und konzentriert sich nur auf das, was hinzugefügt werden muss. Mit diesem Ansatz wurde das Problem des Verschwindens des Gradienten gelöst, das bei früheren Modellen auftrat.

Theoretischer Hintergrund



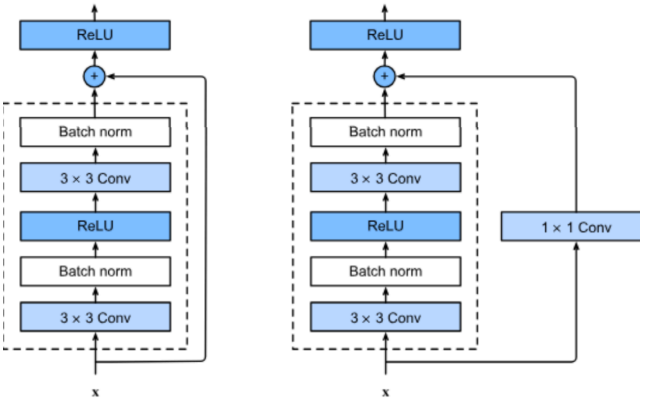
Die Idee: Bei früheren Lernmethoden bestand das Ziel darin, den Output direkt zu berechnen. Bei ResNet hingegen geht es nicht um die Berechnung von H(x)(Output), sondern darum, zu berechnen, wie stark wir von x abweichen.
Die Ausgabe dieses Netzes:
Ausgabe: F(x) + x
Diese Struktur wird Residual Learning genannt, weil nicht der Output direkt gelernt wird, sondern die Differenz zwischen Output und Input.

Das Lernen erfolgt im Allgemeinen durch Aktualisierung der Gewichte zwischen den Verbindungen. Diese Aktualisierung erfolgt mit Hilfe des Gradienten. Mathematisch gesehen ist der Gradient die Ableitung der Verlustfunktion in Bezug auf das Gewicht.
Neues Gewicht = Gewicht - (Gradient x Lernrate)
Dieser Vorgang wird bei jedem Schritt wiederholt, und auf diese Weise beginnt das Netz, bei jedem Schritt ein wenig genauer vorherzusagen.

Die Tatsache, dass diese Gradientenoperation mit der Ableitung durchgeführt wird, war das größte Problem bei den vor ResNet verwendeten Modellen. Denn die Ableitungsfunktionen wurden mit zunehmender Anzahl der Schichten zu kleineren Zahlen. Wenn diese Ableitungen zu einer Zahl im Bereich [0, 1) werden, wird der Gradient fast Null. Da die Gewichte in der ersten Schicht nicht aktualisiert werden konnten, war die Lernleistung in Netzen mit einer hohen Anzahl von Schichten sehr gering. Dieses Problem wird in der Literatur als Gradientenschwund bezeichnet.
Die Struktur, die ResNet Skip Connection nennt, unterbricht die Schrumpfungskette in dieser Ableitungskette. ResNet fügt jedem Block eine Abkürzung hinzu, die als Restverbindung bezeichnet wird. Allerdings lernt das Modell nur die Differenz.

Da der Gradient immer rückwärts durch x fließt, d. h. d(x)/d(x) = 1, selbst wenn df/dx sich 0 nähert, ist der Gesamtgradient nicht 0 und es wird immer ein gewisser Gradient übertragen. Dies garantiert das Lernen.

Kurz gesagt, ResNet lernt sowohl genauer als auch schneller, da es sich nur mit dem Lernen neuer Informationen befasst und die richtigen Informationen überspringt und mitnimmt.



Installation
Um ResNet zu installieren, muss zunächst die PyTorch-Bibliothek installiert werden.

```
pip install torch torchvision
```

Mit dem oben genannten Befehl wird die PyTorch-Bibliothek auf dem Computer installiert.
Dann wird ResNet in die Programmiersprache Python importiert. Je nach Rechenleistung des Computers können verschiedene Modelle von ResNet aufgerufen werden. Für Computer mit relativ geringer Rechenleistung kann beispielsweise das Modell ResNet18 verwendet werden, während für leistungsfähigere Computer das Modell ResNet50 eingesetzt werden kann.
Nehmen wir zum Beispiel an, dass wir ResNet50 verwenden und versuchen, eine Vorhersage zu treffen, indem wir das zuvor auf ImageNet trainierte Modell laden:

```
from torchvision.models import resnet50, ResNet50_Weights
```

```
weights = ResNet50_Weights.DEFAULT
model = resnet50(weights=weights)
model.eval()  # Prognosemodus
```

Dann nehmen wir die folgenden Änderungen vor, um das Bild, das wir dem Modell geben wollen, für das Modell geeignet zu machen.

```
from PIL import Image
from torchvision import transforms

image = Image.open("resim.jpg").convert("RGB")

transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

img_tensor = transform(image).unsqueeze(0)  # [1, 3, 224, 224]
```

Der folgende Python-Kontextmanager wird verwendet, um Vorhersagen zu treffen.

```
import torch.nn.functional as F
with torch.no_grad():
    output = model(img_tensor)
    probs = F.softmax(output[0], dim=0)
```

Die Vorhersagen des Modells können mit dem folgenden Codeblock ausgedruckt werden:

```
class_names = weights.meta["categories"]
top5 = torch.topk(probs, 5)

for idx, score in zip(top5.indices, top5.values):
    print(f"{class_names[idx]}: {score:.2%}")
```

Umfang und Komplexität (Für jede Aufgabe: Konzept, Installation Implementierung, mind. 3 exemplarische Ausführungen, Limits der Lsg.)
Aufbau (Einleitung, Erklärung der Ideen, Eval, Fazit), Rechtschreibung, …
Kenntnisse über Vorl. hinaus (z.B. Customizen fremder GitHub-P

1.b. EfficientNet

Einleitung auf das Modell

EfficientNet-B0 ist ein Convolutional Neural Network (CNN), das von Google AI im Jahr 2019 eingeführt wurde. Im Vergleich zu klassischen Modellen soll es eine höhere Genauigkeit bei geringerem Rechenaufwand bieten. Es ist die Basismodellvariante der EfficientNet-Familie (B0 bis B7) und wurde mit einem völlig neuen Architekturprinzip entwickelt. Compound Scaling. Das Ziel besteht darin, maximale Genauigkeit bei minimalem Rechenaufwand zu erreichen.

EfficientNet wurde von Tan und Le (2019) in „EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks" vorgestellt. Die damals verwendeten Modelle wurden in der Regel nur in Bezug auf Tiefe (mehr Schichten), Breite (mehr Kanäle) oder Eingabeauflösung (größere Eingabegröße) skaliert. Diese eindimensionalen Skalierungsmethoden führten jedoch zu ineffizienten und instabilen Ergebnissen. Um dieses Problem zu lösen, schlug EfficientNet eine Methode namens Compound Scaling vor. Dabei werden Tiefe, Breite und Eingabeauflösung des Modells gleichzeitig und in ausgewogener Weise skaliert, was zu einem wesentlich effizienteren Wachstum führt.

Zusammengesetzte Skalierung: Alle Dimensionen des Modells werden zusammen skaliert.

MBConv-Blöcke: Leichte und effiziente Struktur aus MobileNetV2.

Anstelle traditioneller tiefer Faltungsschichten verwendet EfficientNet diese Blöcke, die von der MobileNetV2-Architektur übernommen wurden. Diese Struktur verbessert die Effizienz des Modells erheblich, da sie ähnliche Genauigkeiten mit weniger Parametern erreicht. MBConv-Blöcke basieren auf einer dreistufigen Sequenz: Zunächst wird die Eingabe erweitert, dann in der Tiefe gefaltet und schließlich gequetscht, um die Ausgabe zu erzeugen. Diese Architektur senkt nicht nur die Rechenkosten, sondern verringert auch das Risiko einer Überanpassung. Dadurch ist es möglich, dass das Modell mit hoher Leistung auf kleineren Geräten (z. B. Mobiltelefonen oder eingebetteten Systemen) läuft. MBConv-Blöcke spielen daher eine Schlüsselrolle dabei, EfficientNet sowohl zu einer leichtgewichtigen als auch zu einer leistungsstarken Architektur zu machen.

Die Swish-Aktivierungsfunktion bietet im Vergleich zur klassischen ReLU (Rectified Linear Unit) ein sanfteres Lernen.

Anstelle der klassischen ReLU (Rectified Linear Unit) verwendet EfficientNet eine glattere und kontinuierlich differenzierbare Aktivierungsfunktion namens Swish. Die Swish-Funktion ist definiert als f(x) = x - sigmoid(x). Diese Struktur sorgt für einen stabileren Gradientenübergang, insbesondere bei sehr tiefen Netzen, und ermöglicht ein besseres Lernen, da das Problem des verschwindenden Gradienten reduziert wird.

Hohe Genauigkeit bei geringer Parameterzahl: Genauigkeit nahe an ResNet50 mit nur 5 Millionen Parametern.

Warum es für das Projekt als geeignet angesehen wurde?

Da es sich bei den Daten, die in unserem Projekt klassifiziert werden müssen, in der Regel um Screenshots der Benutzeroberfläche (z. B. Excel-Tabellen, Diagramme usw.) handelt, sind diese Bilder, obwohl sie strukturell nicht komplex sind, oft von geringer Auflösung. Dennoch müssen sie schnell und genau analysiert werden. Das Modell EfficientNet-B0 bietet dank seiner leichten Bauweise und hohen Verarbeitungseffizienz eine sehr geeignete und effiziente Lösung für diese Art von Daten.

Theoretischer Hintergrund

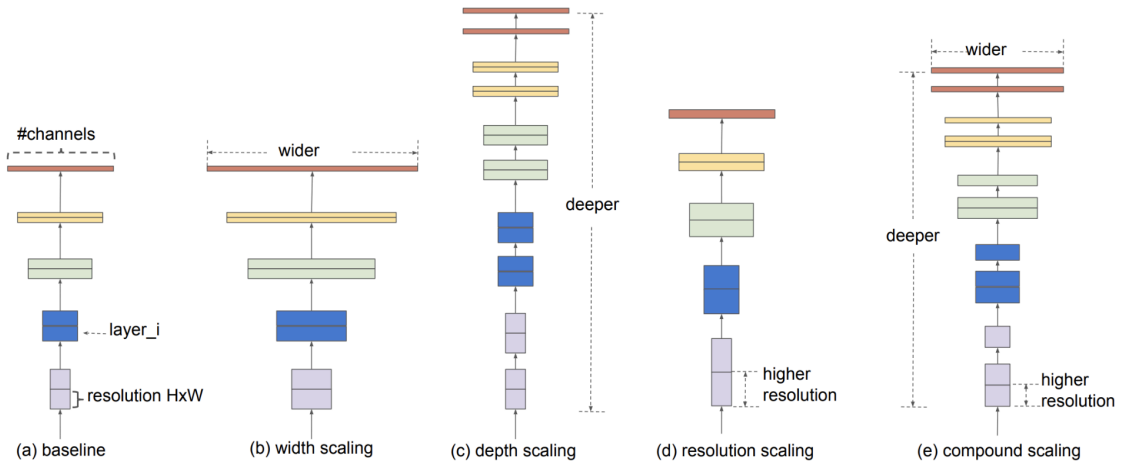


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

https://production-media.paperswithcode.com/methods/Screen_Shot_2020-06-06_at_10.45.54_PM.png

Abbildung 2: Vergleich von EfficientNets Compound Scaling mit klassischen Skalierungsansätzen.

(a): Ein grundlegendes CNN-Modell

(b), (c), (d): Klassische Methoden mit nur erhöhter Breite, Tiefe oder Auflösung.

(e): Der von EfficientNet vorgeschlagene Compound-Scaling-Ansatz, bei dem alle drei Dimensionen proportional wachsen.

Dieser Ansatz ermöglicht eine Vergrößerung des Modells bei gleichzeitiger Erhaltung der Effizienz und Ausgewogenheit der Ressourcenauslastung.

EfficientNet – Skalierungsformel (Compound Scaling)

Parametre	Anlamı
ϕ	Global scaling factor
α	Depth Coefficient
β	Width Coefficient
γ	Resolution Coefficient

Tiefe= α^ϕ
Breite= β^ϕ
Auflösung=γ^ϕ

ϕ -> Dies ist der Hauptsteuerungsparameter, der bestimmt, wie groß das Modell skaliert wird. Mit zunehmendem ϕ erhöhen sich sowohl die Anzahl der Schichten, die Anzahl der Kanäle als auch die Eingabeauflösung des Modells.

α -> Er bestimmt die Anzahl der Schichten des Modells, d. h. er steuert, wie viele Faltungsblöcke es gibt. Es kann komplexere Muster mit größerer Tiefe lernen.

β -> Bestimmt die Anzahl der Kanäle (Filter), die in jeder Ebene verwendet werden. Mit anderen Worten: Sie bestimmt, wie viele Merkmale gleichzeitig analysiert werden. Reichere Merkmalserfassungskapazität mit mehr Breite.

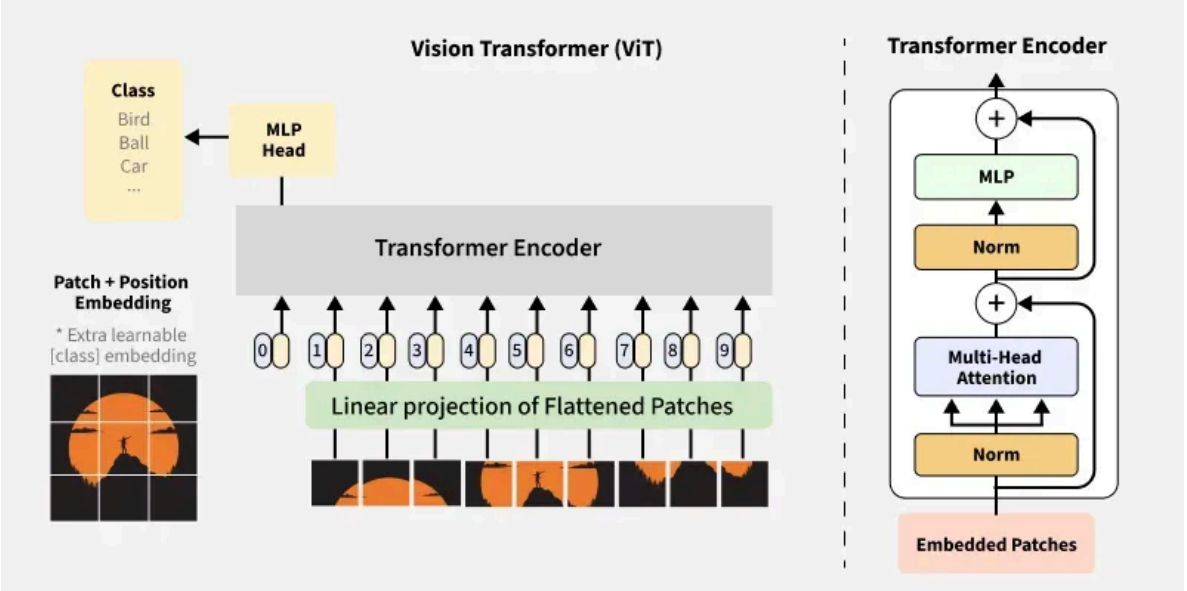
γ -> Bestimmt die Pixelgröße von Eingabebildern. Je höher die Auflösung, desto feinere Details können erkannt werden, aber der Rechenaufwand steigt.

1.c. Virtual Transformer

Einleitung auf das Modell

2020 wurde der Vision Transformer (ViT), ein neuartiges Modell zur Bildklassifikation, von der Google-Brain-Forschungsgruppe präsentiert. Es ist eine entscheidende Weiterentwicklung in der Anwendung von Transformer-Architekturen, die zuvor hauptsächlich im Bereich der natürlichen Sprachverarbeitung (NLP) verwendet wurden. Die in der NLP äußerst erfolgreichen Selbstaufmerksamkeitsmechanismen (Self-Attention) können auch auf Bilddaten angewendet werden, wie das bahnbrechende Paper „An Image is Worth 16x16 Words" demonstriert hat.

ViT formt ein Eingabebild in ein Feld fester Patch-Größen (z. B. 16 x 16 Pixel) um, von denen jedes als ein einzelnes Wort behandelt wird. Diese Fleckensymbole werden zusammen mit ihren Standortinformationen in das Transformer-Netzwerk eingefügt, wo ihre globale Beziehung über mehrere Schichten modelliert wird. ViT erlaubt daher eine globale Kontextualisierung in den frühen Phasen des Netzwerks, während traditionelle CNNs normalerweise nur lokale Muster untersuchen.



Theoretischer Hintergrund

Der Vision Transformer basiert auf der klassischen Transformer-Architektur, die eigentlich für Sprachverarbeitungsaufgaben wie die maschinelle Übersetzung entwickelt wurde. Das Herzstück dieses Modells ist die Self-Attention-Methode, mit der Beziehungen zwischen unterschiedlichen Token (in diesem Fall Bildausschnitte) unabhängig von ihrer Position modelliert werden können. Dadurch kann das Modell effizient lernen, welche Teile des Bildes miteinander in Beziehung stehen, was speziell für strukturierte Bilddaten wie Tabellen oder Diagramme wichtig ist.

Aus mathematischer Sicht funktioniert die Selbstaufmerksamkeit, wenn jedem Eingabe-Token drei Vektoren zugeordnet werden: Abfrage (Q), Schlüssel (K) und Wert (V). Die Gewichte der Selbstaufmerksamkeit werden durch das Skalarprodukt zwischen Q und K kalkuliert, durch die Wurzel der Dimension (d_k) normalisiert und dann mit Softmax skaliert:

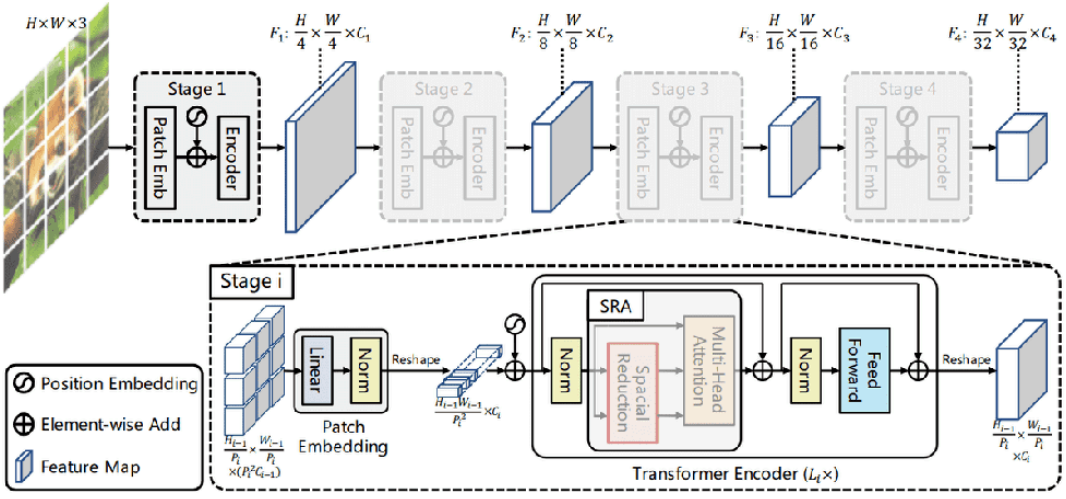
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Dieser Vorgang erlaubt es dem Modell, sich dynamisch auf relevante Regionen im Bildkontext zu konzentrieren. Der Transformer setzt sich aus mehreren dieser Schichten zusammen, kombiniert mit Schichtnormalisierung, Restverbindungen und MLP-Blöcken (Multi-Layer Perceptrons).

Die wichtigsten Modellparameter eines ViT sind:

- Patch-Größe: In der Regel 16x16 oder 32x32 Pixel. Je kleiner, desto feinkörniger die Analyse, aber auch desto größer der Rechenaufwand.
- Dimension der Einbettung: Gibt die Dimension an, in der die Patches eingebettet sind (z. B. 768 für ViT-Base).
- Tiefe: Anzahl der zusammenhängenden Transformatorenblöcke (z. B. 12 für ViT-Base).
- Anzahl der Aufmerksamkeitsköpfe: Gibt die Parallelisierung in der Selbstaufmerksamkeit an (z. B. 12 für ViT-Base).
- MLP-Dimension: Größe der versteckten Schicht im Feed-Forward-Netz innerhalb jedes Blocks.
- Positionskodierung: Erforderlich, da der Transformator selbst keine inhärente Reihenfolge kennt.

Eine Herausforderung bei der Anwendung von ViT ist der Bedarf an großen Datenmengen für ein erfolgreiches Training. Daher werden häufig vortrainierte Modelle benutzt (z. B. ImageNet-21k oder JFT-300M). Für unser Projekt bedeutet dies, dass wir ein fein eingestelltes ViT-Modell verwenden, das bereits grundlegende visuelle Konzepte gelernt hat und nur noch für unsere spezifischen Klassen angepasst werden muss.



Installation

Um Vision Transformer (ViT) zu verwenden, wird die Python-Bibliothek timm benötigt. Diese enthält eine Vielzahl an vortrainierten Vision-Modellen.

```
pip install timm
```

Außerdem werden torch, torchvision und PIL für die Bildverarbeitung benötigt:

```
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu
pip install timm matplotlib scikit-learn pandas opencv-python pdf2image
```

Implementierung

Modell laden

```
import timm
import torch

# Vortrainiertes ViT-Modell (ImageNet1k)
model = timm.create_model('vit_base_patch16_224', pretrained=True)
model.eval()
```

Bildvorverarbeitung

```
from torchvision import transforms
from PIL import Image

image = Image.open("resim.jpg").convert("RGB")

transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=(0.5, 0.5, 0.5),  # ViT verwendet oft diese Normalisierungswerte
        std=(0.5, 0.5, 0.5)
    )
])

img_tensor = transform(image).unsqueeze(0)  # [1, 3, 224, 224]
```

Vorhersage berechnen

```
import torch.nn.functional as F

with torch.no_grad():
    output = model(img_tensor)
    probs = F.softmax(output[0], dim=0)
```

Ergebnis anzeigen

```
# Klassenbezeichnungen laden (aus ImageNet)
import urllib.request

url = 'https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.txt'
class_names = urllib.request.urlopen(url).read().decode('utf-8').splitlines()

# Top-5 Vorhersagen anzeigen
top5 = torch.topk(probs, 5)

for idx, score in zip(top5.indices, top5.values):
    print(f"{class_names[idx]}: {score:.2%}")
```

Exemplarische Ausführung

```
tabby, tabby cat: 78.52%
Egyptian cat: 14.03%
tiger cat: 5.62%
Siamese cat: 1.21%
lynx, catamount: 0.42%
```

Mögliche Einschränkungen der Lösung

- Rechenleistung: Transformer-Modelle brauchen deutlich mehr Rechenleistung als einfache CNNs (wie ResNet18).
- Bildauflösung: ViT benötigt eine feste Eingabegröße (z. B. 224×224 Pixel).
- Allgemeinheit: Trotz des Vortrainings mit ImageNet können domänenspezifische Bilder (z. B. medizinische Bilder) falsch eingeschätzt werden.