

# Vorlesung 10: Transformer Netzwerke

BA-INF 153: Einführung in Deep Learning für Visual Computing

Prof. Dr. Reinhard Klein

Nils Wandel

Informatik II, Universität Bonn

03.07.2024

# Recap

## Letzte Woche: Generative Modelle

- Autoregressive Modelle
- GANs
- Flow-Based Modelle
- Diffusion Modelle

# Inhalte

- Transformer
- Image-GPT
- Vision Transformer (ViT)
- SWIN-Transformer
- CLIP

# Grundlagen: Natural Language Processing

- Die meisten Themen aus dem ersten Teil dieser Vorlesung sind Entwicklungen im Bereich des Natural Language Processing (NLP).
- Natural Language ist im Gegensatz zu formaler Sprache (z.B. formale Grammatiken, Programmiersprachen, etc) die von Menschen gesprochene oder geschriebene Sprache. (Weitere Anwendungen im Bereich von Visual Computing werden wir diskutieren, nachdem wir die Grundkonzepte eingeführt haben.)
- Üblicherweise wird Machine Learning im Bereich von NLP verwendet um natürliche Sprache zusammenzufassen, zu übersetzen oder in einen Zusammenhang zu stellen

## Beispiel

Betrachten wir folgendes NLP-Problem einer Übersetzungsaufgabe:

Deutsche Eingabe: "Wir schreiben den Artikel, der veröffentlicht werden soll jetzt endlich fertig."

Englische Ausgabe: "We finally finish writing the article to be published"

## Abschnitt 1

# Transformer

Aus "Attention is all you need" von Vaswani et al. (2017)

# Transformer-Netzwerk

Das Ziel von Transformer-Netzwerken ist es, Eingabe- auf Ausgabe-Sequenzen abzubilden um z.B. Übersetzungsaufgaben (z.B. Englisch-Deutsch) zu lösen.

Das Transformer Netzwerk ist ein Encoder-Decoder Netzwerk bestehend aus sich wiederholenden Blöcken, welche Multi-Head Attention Layer enthalten.

Eingabesequenzen werden als Menge von Eingabe-Tokens gehandhabt, welche ein posisional Encoding verwenden, um die Reihenfolge der Tokens zu kodieren.

Die Wahrscheinlichkeitsfunktion über die Ausgabe-Tokens wird zum Schluss durch ein Linear Layer mit Softmax Aktivierung berechnet.

Im folgenden sehen wir uns die einzelnen Blöcke genauer an.

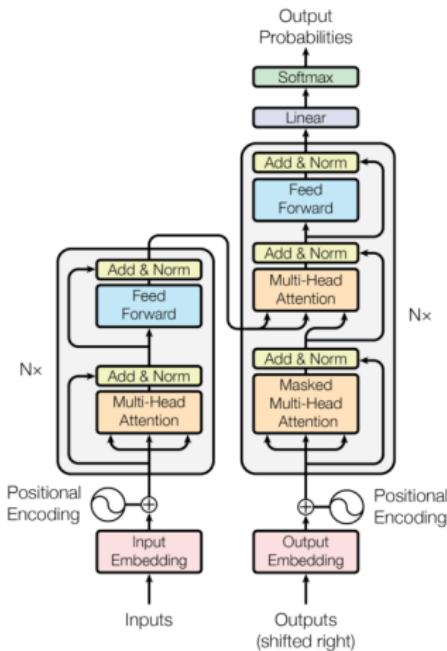


Figure 1: The Transformer - model architecture.

**Figure:** Encoder (links) und Decoder (rechts) der Transformer Architektur

# Transformer-Netzwerk

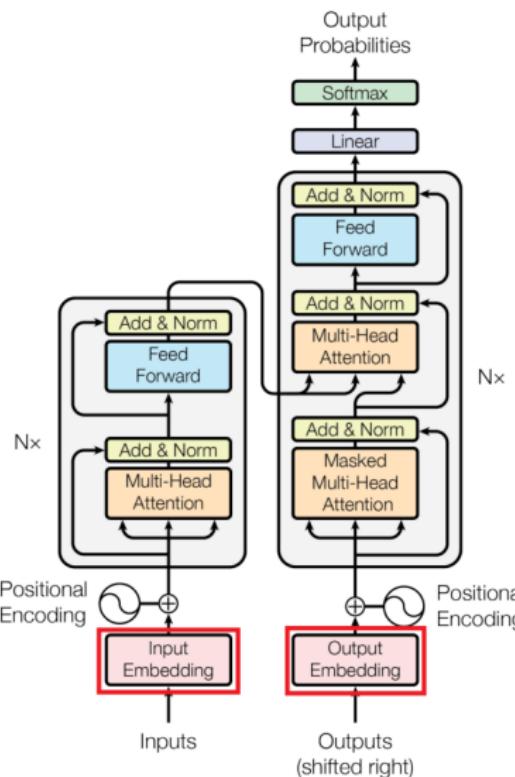


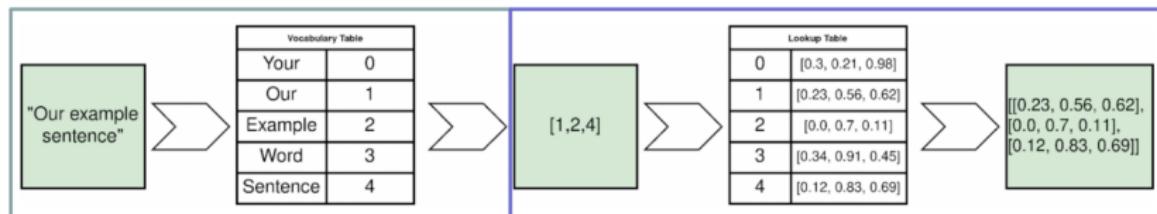
Figure 1: The Transformer - model architecture.

# Input / Output Embedding

Problem: Menschenlesbarer Text ist üblicherweise nicht besonders gut geeignet, um direkt als Eingabe verwendet zu werden, da einzelne Buchstaben einen geringen Informationsgehalt aufweisen und zu langen Eingabesequenzen führen würden.

Lösung: Embedding-Layer

- Das Embedding-Layer verwendet eine Lookup-Tabelle mit gelernten Einträgen (anstatt einer sonst in NN üblichen (nicht-)linearen Abbildung).
- Die Elemente aus der "eingebetteten" Sequenz werden auch als *Tokens* bezeichnet.



Pre-processing step performed once for every sequence.

Lookup is performed in every training step.

Figure: Input Embedding



# Input / Output Embedding

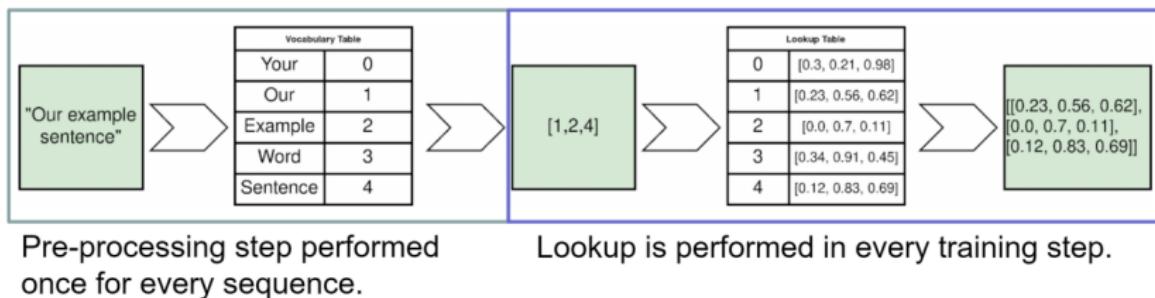


Figure: Input Embedding

Herangehensweise:

- Die Werte der Embedding-Vektoren in der Lookup-Tabelle werden gemeinsam mit den anderen Netzwerkparametern mit Gradient-Descent optimiert.
- Jede Sequenz resultiert in einem spärlichen Gradient über die Einträge der Lookup-Tabelle. D.h. nur die Einträge, welche in der Sequenz als Tokens verwendet wurden haben einen Gradient ungleich 0.

Die Ausgabe des Modells wird mit einem Linear Layer und Softmax berechnet und entspricht einer Wahrscheinlichkeitsfunktion über das definierte Vokabular.

# Transformer-Netzwerk

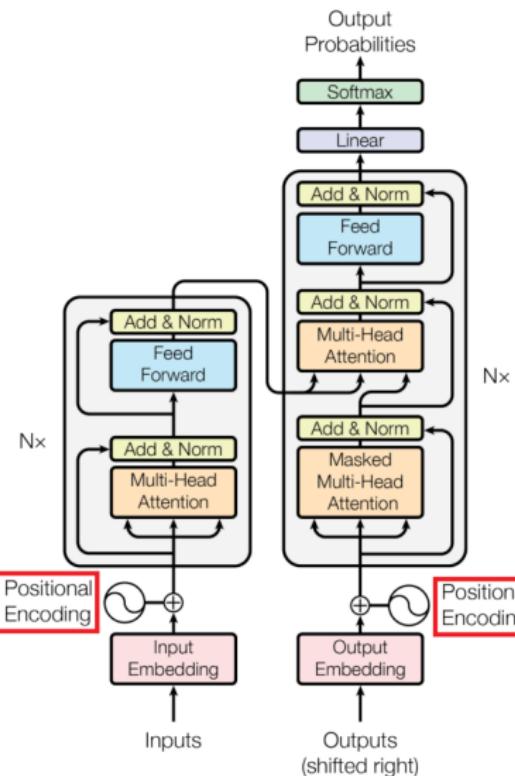


Figure 1: The Transformer - model architecture.

# Positional Encoding

Das Transformer-Netzwerk besitzt keine inherenten Informationen über die Ordnung der Eingabe-Tokens. Deshalb müssen diese "Orts"-Informationen durch ein zusätzliches positional Encoding Layer hinzugefügt werden.

- Sei  $d_{\text{model}}$  die Größe der Embedding-Vektoren und  $p$  die Position eines Worts in einer Eingabesequenz. Dann wird die Position kodiert, indem auf die Embedding-Vektoren ein  $d_{\text{model}}$ -dim Vektor mit Sinusoid-Encoding hinzugefügt wird:

$$\gamma(p) = \left( \dots, \sin\left(\frac{p}{10000^{2i/d_{\text{model}}}}\right), \cos\left(\frac{p}{10000^{2i/d_{\text{model}}}}\right), \dots \right)$$

- Das positional encoding könnte auch als lernbarer Parameter-Vektor gehandhabt werden und würde dann nahezu identische Ergebnisse liefern. Allerdings erlaubt das Sinusoid-Encoding auch Extrapolation für längere Eingabesequenzen und wurde deshalb von den Autoren der Transformer-Architektur bevorzugt.

# Transformer-Netzwerk

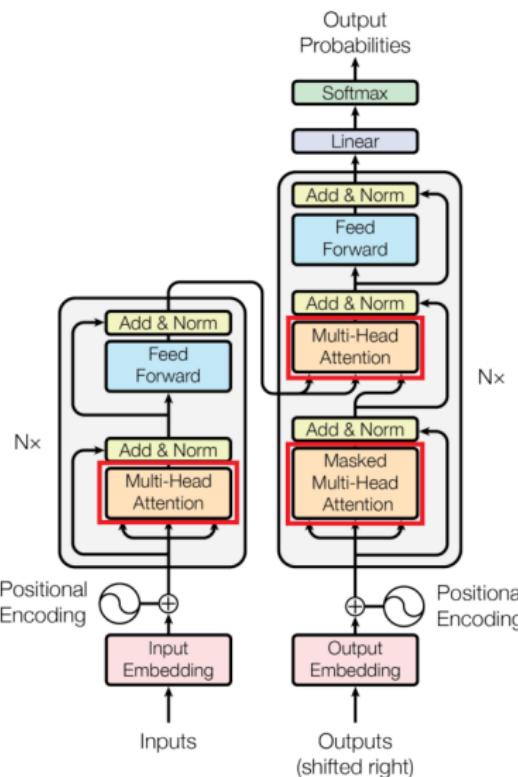
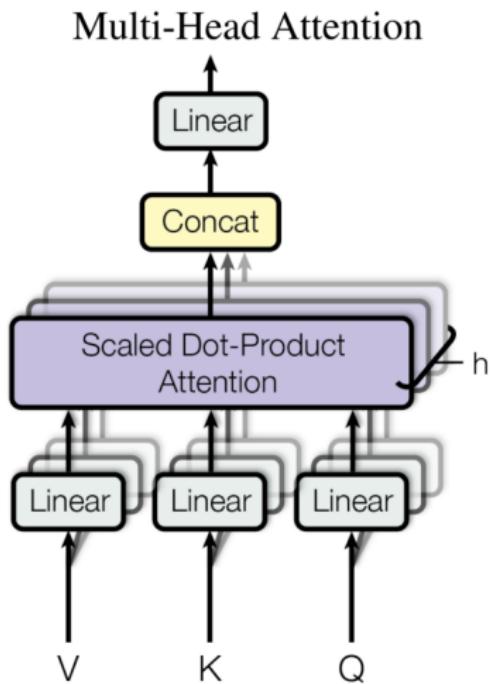


Figure 1: The Transformer - model architecture.

# Multi-Head Attention Mechanismus



**Figure:** Illustration des Multi-Head Attention Mechanismus. Quelle: Vaswani et al. (2017): Attention is All You Need

# Multi-Head Attention Mechanismus

Der Multi-Head Attention Mechanismus verwendet  $h$  (z.B.  $h = 8$ ) parallele Self-Attention "Köpfe". In jedem Attention-Head  $i$  werden die Eingaben linear mit Hilfe von  $W_i^Q$ ,  $W_i^K$ ,  $W_i^V$  auf Queries, Keys und Values projiziert und dann für die Attention verwendet:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

wobei

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

Die  $h$  Ausgaben der Attention-Heads werden dann konkateniert und auf die Model-Dimension mit Hilfe einer Matrix  $W^0$  projiziert:

$$\text{MultiHead}(Q, V, K) = \text{concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_h)W^0$$

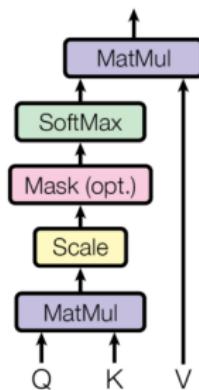
# Scaled Dot-Product Attention

Betrachten wir nun die Scaled Dot-Product Attention etwas genauer. Das Transformer-Netzwerk verwendet einen modifizierten multiplikativen Attention Mechanismus:

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

Hierbei werden die Attention-Werte mit  $\frac{1}{\sqrt{d_k}}$  skaliert um einen Performanceverlust aufgrund einer hohen Dimensionalität der Attention  $d_k$  zu vermeiden. (Eine grosse Dimensionalität erlaubt dem Skalarprodukt höhere Werte. Dadurch kann die Softmax-Funktion gesättigt werden und die Gradienten - ähnlich wie bei der  $\sigma$ -Funktion - verschwinden).

Scaled Dot-Product Attention



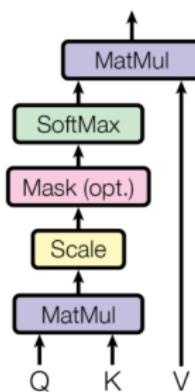
**Figure:** Scaled Dot-Product Attention. Quelle: Vaswani et al. (2017): Attention is All You Need

# Erhaltung der autoregressiven Eigenschaft durch Masking

Der Decoder sollte ein autoregressives Modell sein, welches nur von den vorherigen Eingaben abhängt.

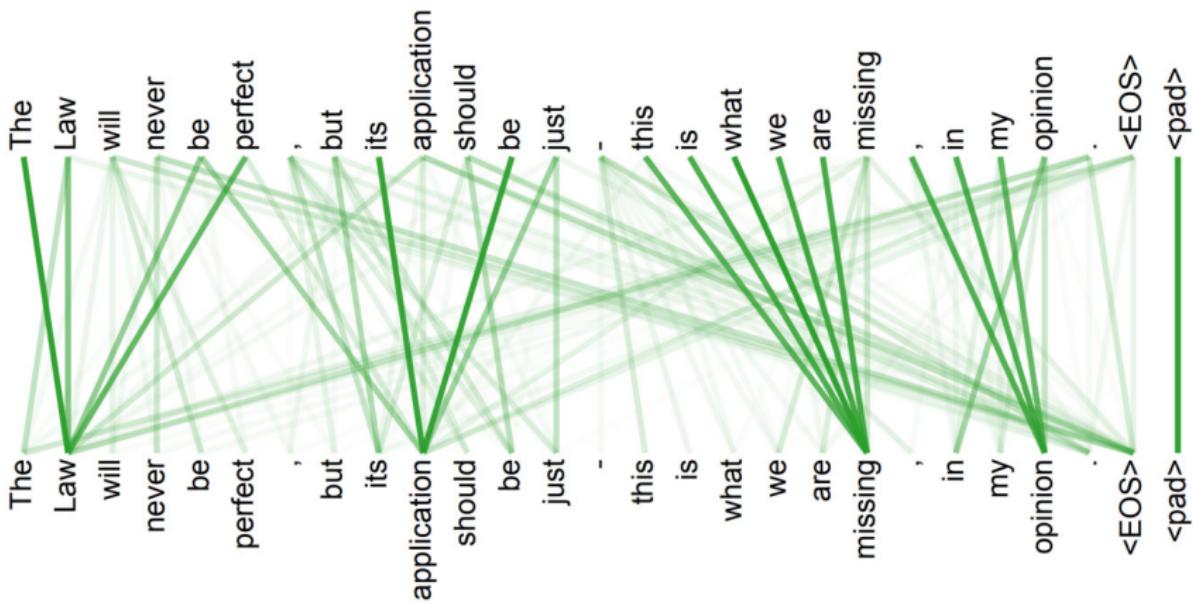
- Die Matrix  $Q$  und  $K$  kombinieren mehrere Queries und Keys und das äussere Produkt  $QK^T$  enthält die Skalarprodukte zwischen allen Query / Key - Kombinationen.
- Dies würde dem Decoder erlauben, Informationen aus vorher ungesesehenen Ausgaben zu erhalten, was die autoregressive Eigenschaft verletzen würde.
- Um diesen Informationsfluss zu unterbinden wird im Decoder zusätzlich eine *Maske* verwendet, welche die Skalarprodukte sämtlicher Query/Key Paare, die linksgerichteten Informationsfluss repräsentieren auf  $-\infty$  setzen und somit deren Attention score nach dem Softmax auf 0.

Scaled Dot-Product Attention



**Figure:** Scaled Dot-Product Attention. Quelle: Vaswani et al. (2017): Attention is All You Need

# Self-Attention Beispiel



**Figure:** Beispiel für Gewichte, die mit Hilfe des Self-Attention-Mechanismus Verbindungen zwischen Query-Tokens (oben) und Key-Tokens (unten) herstellen. Quelle: Vaswani et al. (2017): Attention is All You Need

# Self-Attention als Information Retrieval

- Self-Attention ist ein Grundkonzept in Transformer-Netzwerken und wird manchmal auch als Intra-Attention bezeichnet.
- Hierbei werden Gewichte berechnet, um die Eingabesequenz mit sich selbst zu vergleichen und eine gewichtete Summe über die Tokens der Sequenz zu erhalten. Dadurch wird eine Zusammenfassung der relevanten Tokens der Sequenz für jedes Element der Sequenz erstellt.

Der Attention Mechanismus kann auch als ein Information Retrieval Prozess interpretiert werden. In diesem Prozess wird eine Anfrage (Query)  $Q$  und ein Schlüssel (Key)  $K$  verglichen um eine passende Ausgabe basierend auf den Werten (Values)  $V$  zu berechnen.

# Transformer-Netzwerk

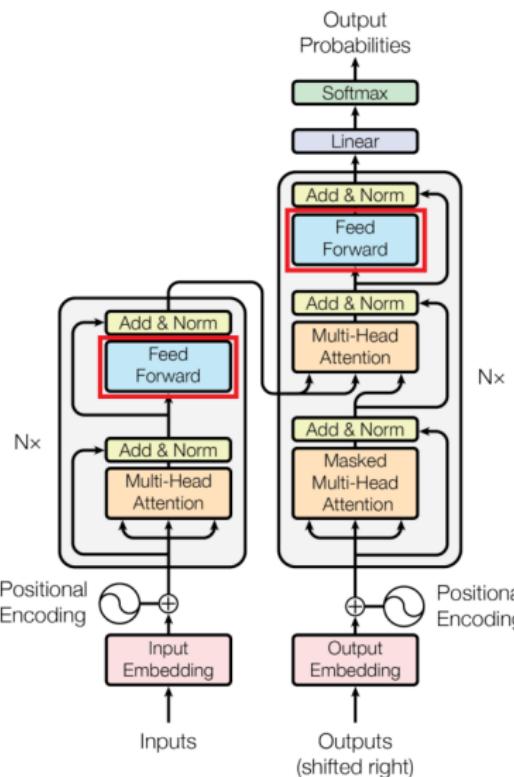
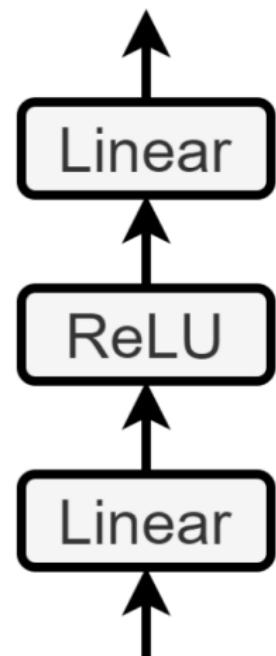


Figure 1: The Transformer - model architecture.

# Feed-Forward Netzwerk

- Das Feed-Forward Netzwerk ist ein 2-layer MLP mit einer zwischengeschalteten ReLU-Aktivierungsfunktion. Es bildet zunächst einen 512-dim Feature-Vektor auf einen 2048-dim Zwischenrepräsentation und diese dann wieder zurück auf eine 512-dim Repräsentation
- Dadurch können komplexere Features vor dem nächsten Multi-Head Attention-Block gelernt werden.



# Transformer-Netzwerk

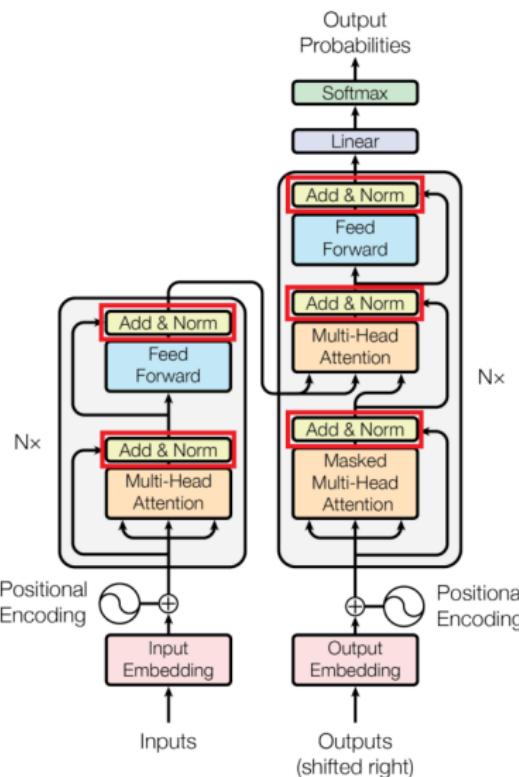
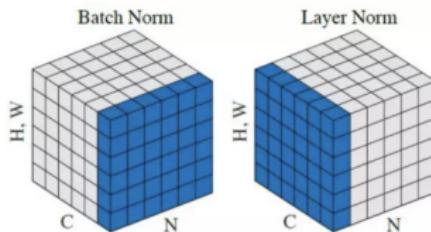


Figure 1: The Transformer - model architecture.

# Residual Connections und Layer Normalisierung

Ähnlich wie bei Resnets werden für bessere Gradienten residual Connections verwendet. Statt Batch-Normalisierung wird jedoch Layer-Normalisierung verwendet:

- Im Gegensatz zu Batch-Normalisierung, welche den Mean und Varianz für jeden Channel separat über den gesamten Batch berechnet, berechnet Layer-Normalisierung diese Werte über die Channels für jedes Sample im Batch separat.



**Figure:** Batch vs Layer Normalization. H/W entsprechen der Bildgröße (bzw der Sequenzlänge), C der Anzahl Channels (bzw  $d_{model}$ ) und N der Batchgröße.

# Scaling von Transformer-Netzwerken

Rank	Name	Model	URL	Score
1	ERNIE Team - Baidu	ERNIE		90.9
2	DeBERTa Team - Microsoft	DeBERTa / TuringNLVRv4		90.8
3	HFL iFLYTEK	MacALBERT + DKM		90.7
4	Alibaba DAMO NLP	StructBERT + TAPT		90.6
5	PING-AN Omni-Sinic	ALBERT + DAAF + NAS		90.6
6	T5 Team - Google	T5		90.3
7	liangzhu ge	Deberta (adv + ensemble)		90.3
8	Microsoft D365 AI & MSR AI & GATECHMT-DNN-SMART			89.9
9	Huawei Noah's Ark Lab	NEZHA-Large		89.8
10	Zihang Dai	Funnel-Transformer (Ensemble B10-10-H1024)		89.7

Die meisten Modelle der Top 10 im "General Language Understanding Evaluation" (kurz: GLUE) Benchmark basieren auf Transformern.<sup>1</sup>

<sup>1</sup>Quelle: <https://gluebenchmark.com/leaderboard> (13.07.2021)

# Transformer in Computer Graphics und Vision

Transformer sind mittlerweile die leistungsstärksten Modelle im Bereich von NLP und haben gezeigt, dass sie kontextuelle Zusammenhänge aussergewöhnlich gut modellieren können.

Somit stellt sich die Frage, ob sie auch im Bereich der Computer Grafik / Vision eingesetzt werden können?

⇒ Antwort: Ja!

Als nächstes: Image-GPT, ViT und SWIN-Transformer als erfolgreiche Anwendungsbeispiele von Transformer-Netzwerken in Computer Grafik und Vision-Problemen.

## Abschnitt 2

# Image-GPT

Aus "Generative Pretraining from Pixels" von Mark Chen et al. (2020)

# Image-GPT: Transformer für Klassifizierungsaufgaben

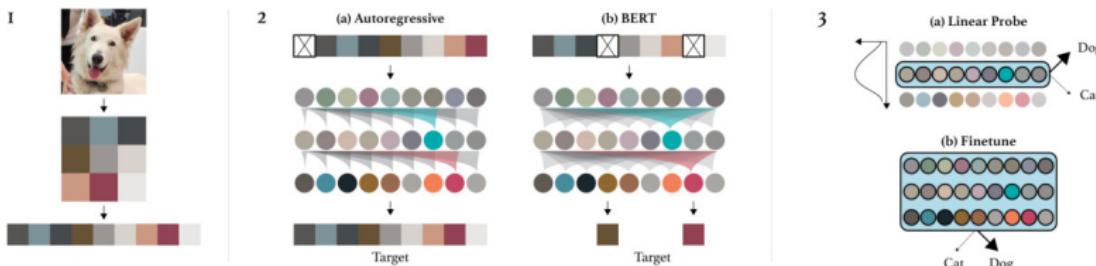


Figure 1. An overview of our approach. First, we pre-process raw images by resizing to a low resolution and reshaping into a 1D sequence. We then chose one of two pre-training objectives, auto-regressive next pixel prediction or masked pixel prediction. Finally, we evaluate the representations learned by these objectives with linear probes or fine-tuning.

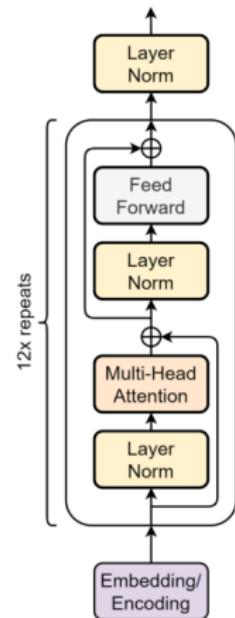
- Image-GPT<sup>2</sup> verwendet die Fähigkeiten eines Transformers um Repräsentationen zu lernen, welche sinnvolle Features aus Bildern für einen Klassifikator enthalten.
- Das Training von Image-GPT besteht aus 2 Schritten:
  - ① trainiere einen Transformer auf einem Datenset vor um sinnvolle Features zu extrahieren. Dazu kann entweder eine Autoregressive Zielfunktion oder eine "BERT"-Zielfunktion verwendet werden. (mehr dazu gleich)
  - ② verwende die extrahierten Features aus dem vortrainierten Netzwerk für einen (linearen) Klassifikator um Bilder zu klassifizieren.

<sup>2</sup>Chen et al. (2020): Generative Pretraining from Pixels

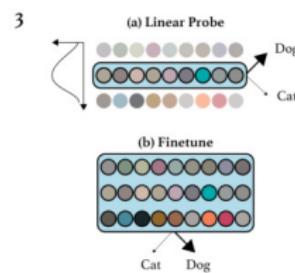
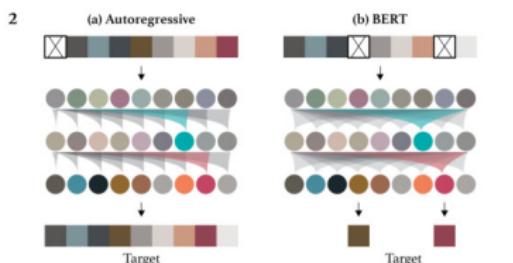
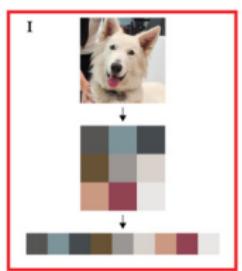
# Image-GPT: Architektur

Der Image GPT Transformer basiert auf der GPT-2 Architektur.

- Der Image GPT Transformer besitzt nur ein Decoder Netzwerk, welches die selben Layer wie die ursprüngliche Transformer-Architektur verwendet.
- Um die Skalierbarkeit des Models zu verbessern wird die Layer-Normalisierung innerhalb der Residual-Connections angewendet.



# Image-GPT: Image Embedding



# Image-GPT: Image Embedding

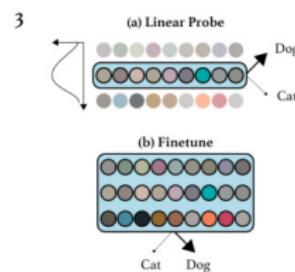
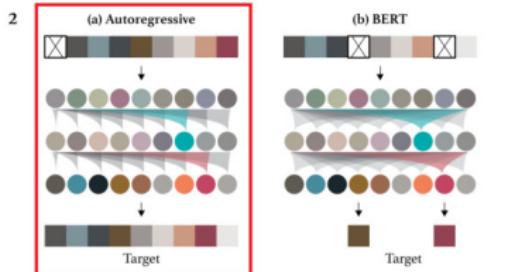
- Die Pixel im Bild müssen als Tokens interpretiert werden. Dazu können wir die Bilder mit Breite  $w$  und Höhe  $h$  zunächst in einen Vektor der Grösse  $n = w \cdot h$  packen.
- Da die Komplexität des Self-Attention Mechanismus  $O(n^2 \cdot d)$  ist, wobei  $d$  die Anzahl Bits zur Encodierung der Farbwerte ist, wird dieser Ansatz für grössere Auflösungen schnell unpraktisch (Bei einer Auflösung von  $256^2$  hätte jedes Layer  $256^4 = 4.3$  Milliarden Einträge).

# Image-GPT: Image Embedding

Diese lineare Skalierung in  $d$  sowie quadratische Skalierung in der Anzahl Pixel  $d$  macht eine Reduzierung der Farbtiefe sowie Auflösung nötig. Dazu können 2 Massnahmen ergriffen werden:

- Skaliere das Bild auf eine niedrigere Auflösung (z.B.  $32 \times 32$ ) herunter.  
Niedrigere Auflösungen als  $32 \times 32$  wurden nicht verwendet, da dann selbst Menschen die Bilder nicht mehr klassifizieren können.
- Reduziere die Farbtiefe auf 9 bits (z.B. durch k-means Clustering der RGB-Werte in 512 Bins). Diese 512 bins entsprechen dann (wie bei transformern) Embedding-Vektoren.
- Die Vektoren für das positional encoding werden als Parameter gelernt und zu den Embeddings hinzugefügt.

# Image-GPT: Autoregressive pre-trainingg



# Image-GPT: Autoregressive pre-training

Die erste pre-training Strategie für Image-GPT basiert auf einem autoregressiven Loss.

- Sei  $(x_1, \dots, x_n)$  eine Sequenz und  $\pi$  eine beliebige Permutation von  $[1 : n]$ . Dann ist die Wahrscheinlichkeit in einem Autoregressiven Modell gegeben durch:

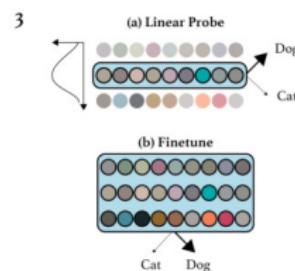
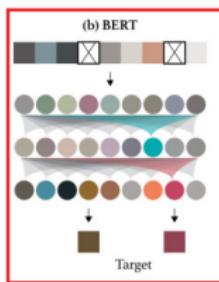
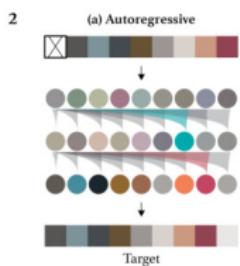
$$p_\theta(x) = \prod_{i=1}^n p_\theta(x_{\pi_i} | x_{\pi_{i-1}}, \dots, x_{\pi_1})$$

- In den meisten Fällen (inklusive Image-GPT) ist die Permutation trivial:  $\pi_i = i \forall i \in [1 : n]$
- Beim Autoregressiven Training wird der Transformer trainiert um die Likelihood der Trainings-sequenzen zu maximieren (bzw die negative log-likelihood zu minimieren):

$$L_{AR} = \mathbb{E}_{x \sim X} [-\log p_\theta(x)]$$

- Um die autoregressive Eigenschaft zu erhalten, wird - wie im Decoder der Transformer-Architektur - eine Maske auf die Attention-Scores angewendet.

# Image-GPT: BERT pre-training



# Image-GPT: BERT pre-training

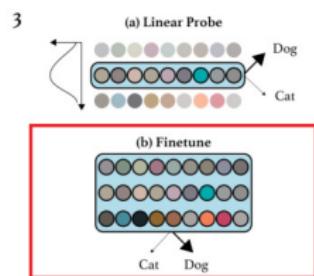
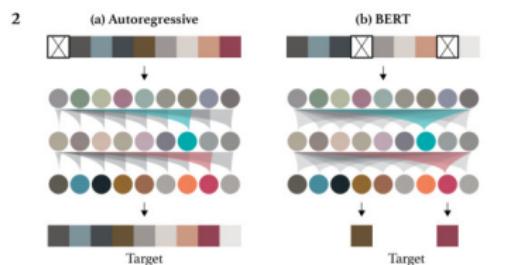
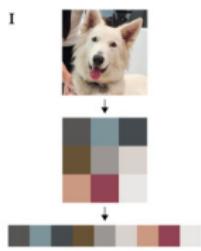
Die zweite pre-training Strategie basiert auf einer BERT (Bidirectional Encoder Representations from Transformers) Zielfunktion.

- Die ursprüngliche Idee von BERT im Kontext von NLP ist zu lernen, einen Lückentext zu füllen, bei dem 15 % der Worte fehlen. Hier sollen fehlende Pixel in einem Bild ausgefüllt werden.
- Dazu wird eine Maske  $M$  gesampled, welche ein Token  $i$  mit Wahrscheinlichkeit 0.15 enthält.
- Das Modell wird dann trainiert, die Likelihood der maskierten Tokens gegeben der unmaskierten Tokens zu maximieren:

$$L_{BERT} = \mathbb{E}_{x \sim X} \mathbb{E}_M \sum_{i \in M} [-\log p(x_i | x_{[1,n] \setminus M})]$$

- Während der Evaluation muss man ebenfalls eine Maske verwenden um einen Shift in der Verteilung über die Eingabewerte zu vermeiden. Allerdings kann eine solche Maske auch die Performance des Klassifikators verringern weshalb als Workaround 5 unabhängige Masken gezogen werden und dann die Klasse vorhergesagt wird, die im Durchschnitt mit der höchsten Wahrscheinlichkeit bewertet wurde.

# Image-GPT: Fine-Tuning



# Image-GPT: Fine-Tuning

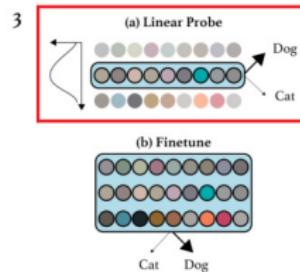
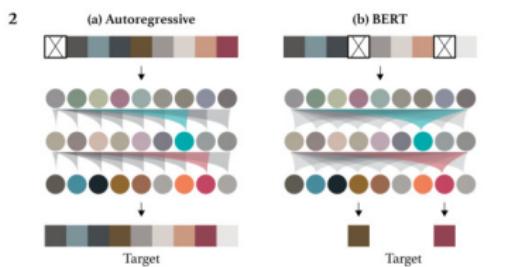
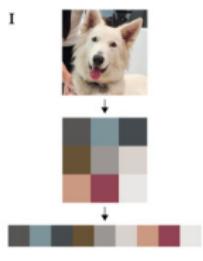
Nach dem pre-training wird das Transformer Netzwerk mit einem Klassifikator-Netzwerk erweitert.

- Die Features aus dem letzten Layer  $L$  werden mit Hilfe von Average-Pooling entlang der Sequenz-Dimension reduziert um einen  $d$ -dimensionalen Feature-Vektor  $f^L$  zu erhalten.
- Mit Hilfe eines linearen Klassifikators (entspricht Linear Layer + softmax) wird aus  $f^L$  dann eine Wahrscheinlichkeitsfunktion über die Ausgaben berechnet.

Das vortrainierte, erweiterte Netzwerk wird dann weiter trainiert (fine-tuning) um den kombinierten Loss  $L = L_{CLF} + L_{GEN}$  zu optimieren.

- $L_{CLF}$  ist der Kreuz-Entropie Loss zwischen den vorhergesagten und den ground truth Labels.
- $L_{GEN}$  ist entweder der autoregressive Loss oder der BERT Loss aus dem pre-training.

# Image-GPT: Linear Probing



# Image-GPT: Linear Probing

Linear Probing verwendet ebenfalls Average-Pooling über die Token-Dimension und einen linearen Klassifikator um die Labels zu erhalten. Unterscheidet sich jedoch in 2 Punkten:

- Die Features, welche im Klassifikator verwendet werden, müssen nicht aus den letzten Transformer Block stammen. Tatsächlich zeigen Experimente, dass die besten Features in den mittleren Layern des Netzes stecken (ca Layer 20).
- Nur der Klassifikator wird trainiert um die Kreuz-Entropie zu minimieren, wohingegen die Gewichte des Transformers konstant gelassen werden.

# Image-GPT: Pre-training Resultate

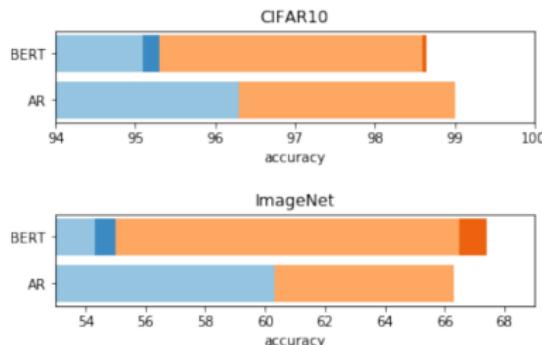
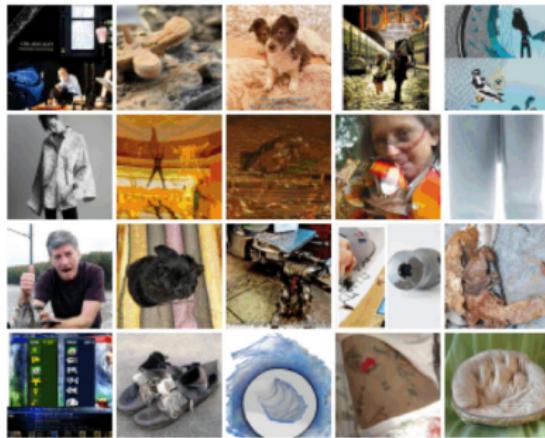


Figure 4. Comparison of auto-regressive pre-training with BERT pre-training using iGPT-L at an input resolution of  $32^2 \times 3$ . Blue bars display linear probe accuracy and orange bars display fine-tune accuracy. Bold colors show the performance boost from ensembling BERT masks. We see that auto-regressive models produce much better features than BERT models after pre-training, but BERT models catch up after fine-tuning.

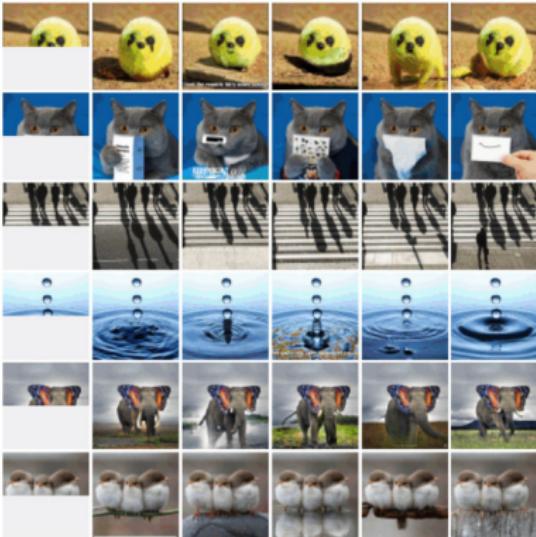
- Das Pre-training mit der BERT-Zielfunktion zeigt eine schlechtere Performanz als der autoregressive Loss, wenn es in Zusammenhang mit linear Probing verwendet wird.
- Das BERT-Version profitiert deutlich mehr vom fine-tuning und kann auf dem ImageNet Datenset sogar die Autoregressive Variante outperformen.

**Figure:** Quelle: Chen et al. (2020):  
Generative Pretraining from Pixels

# Image-GPT: Generative Resultate mit Autoregressiven Modell



**Figure:** Die generierten Bilder haben eine geringe Auflösung, zeigen jedoch kohärente Szenen. Quelle: Chen et al. (2020): Generative Pretraining from Pixels



**Figure:** Die In-Painting Resultate zeigen, dass das Modell ein Verständnis über die Statistik plausibler Szenen hat.

## Abschnitt 3

# Vision Transformer

Aus "An image is worth 16x16 words: Transformers for image recognition at scale"  
von Alexey Dosovitskiy et al. (2021)

# Vision Transformer - Architektur

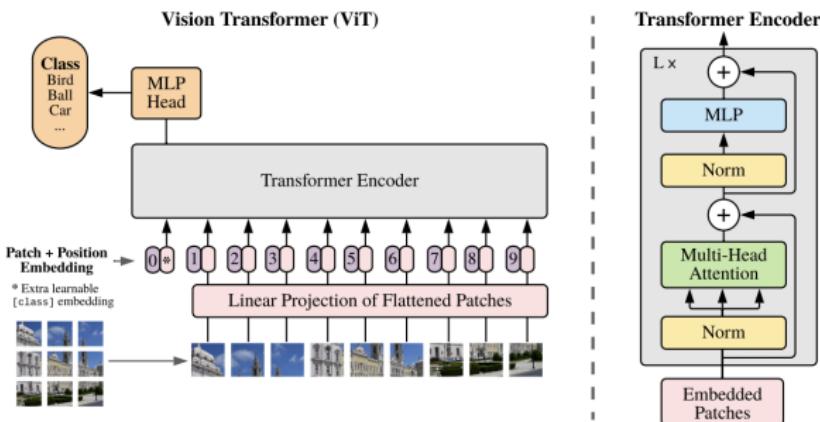


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable “classification token” to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

**Figure:** Vision Transformer Architektur. Quelle: "An image is worth 16x16 words: Transformers for image recognition at scale" von Alexey Dosovitskiy et al. (2021)

# Vision Transformer

## Architektur

- Forme Eingabebild  $x \in \mathbb{R}^{H \times W \times C}$  in 2D Patches  $x_p \in \mathbb{R}^{N \times P^2 \times C}$  um. Hierbei sind  $H, W$  die Bild-Höhe/Breite,  $C$  die Anzahl Channels,  $N = HW/P^2$  die Anzahl Patches (z.B.  $16 \times 16$ ) und  $P$  die Auflösung eines einzelnen Patches.
- Diese Patches werden dann mit einer linearen Projektion auf latente Vektoren ("Embeddings") mit Dimensionalität  $D$  abgebildet.
- Positional Embedding: Lernbare Vektoren, die zu den Patch Embeddings hinzugaddiert werden.
- Transformer Encoder: Entspricht einem standard Transformer, wie vorhin bereits besprochen.
- Class token: ist ein soziales Eingabe-Token, dessen Endzustand als Input für das MLP zur Klassifizierung verwendet wird.
- MLP head: Multi-layered perceptron zur Vorhersage der Class labels mit einem hidden layer während des pre-trainings auf einem grossen Datensatz bzw einem einzelnen linear layer während des fine-tunings auf einem kleineren Datensatz.

# Vision Transformer

## Training

- Pre-training auf großen Datensets (z.B. JFT-300M ist ein Google-internes Datenset mit 300 Millionen Bildern und über 1 Milliarden labels welche mit Hilfe von Algorithmen aus dem Internet extrahiert wurden)
- Fine-tuning auf kleineren Datensets (z.B. ImageNet oder CIFAR)

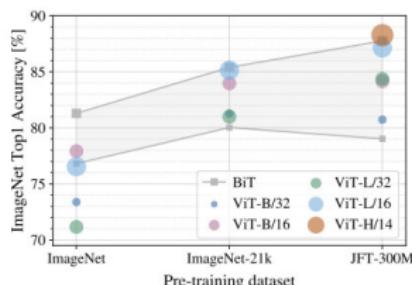


Figure 3: Transfer to ImageNet. While large ViT models perform worse than BiT ResNets (shaded area) when pre-trained on small datasets, they shine when pre-trained on larger datasets. Similarly, larger ViT variants overtake smaller ones as the dataset grows.

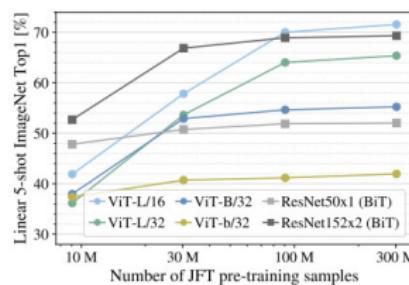


Figure 4: Linear few-shot evaluation on ImageNet versus pre-training size. ResNets perform better with smaller pre-training datasets but plateau sooner than ViT, which performs better with larger pre-training. ViT-b is ViT-B with all hidden dimensions halved.

**Figure:** Vision Transformer benötigen sehr viele Daten im pretraining um bessere Ergebnisse als CNNs zu liefern.

# Vision Transformer

## Resultate

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	<b>88.55</b> ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4 / 88.5*
ImageNet ReaL	<b>90.72</b> ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	<b>99.50</b> ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	—
CIFAR-100	<b>94.55</b> ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	—
Oxford-IIIT Pets	<b>97.56</b> ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	—
Oxford Flowers-102	99.68 ± 0.02	<b>99.74</b> ± 0.00	99.61 ± 0.02	99.63 ± 0.03	—
VTAB (19 tasks)	<b>77.63</b> ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	—
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

Table 2: Comparison with state of the art on popular image classification benchmarks. We report mean and standard deviation of the accuracies, averaged over three fine-tuning runs. Vision Transformer models pre-trained on the JFT-300M dataset outperform ResNet-based baselines on all datasets, while taking substantially less computational resources to pre-train. ViT pre-trained on the smaller public ImageNet-21k dataset performs well too. \*Slightly improved 88.5% result reported in Touvron et al. (2020).

**Figure:** Vision Transformer zeigen eine bessere Performance als CNNs trotz geringerem Rechenaufwand des Trainings

## Abschnitt 4

# SWIN-Transformer

Aus "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows"  
von Ze Liu et al. (2021)

# Swin Transformer: Hierarchische Vision Transformer

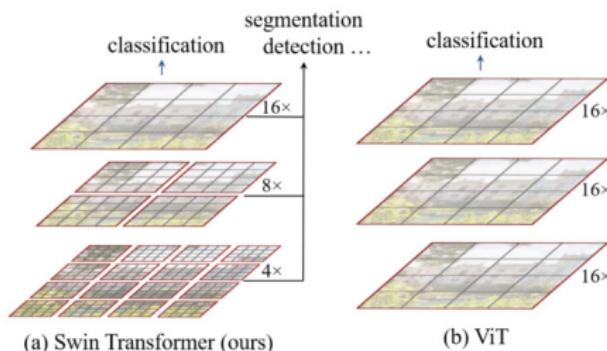


Figure 1. (a) The proposed Swin Transformer builds hierarchical feature maps by merging image patches (shown in gray) in deeper layers and has linear computation complexity to input image size due to computation of self-attention only within each local window (shown in red). It can thus serve as a general-purpose backbone for both image classification and dense recognition tasks. (b) In contrast, previous vision Transformers [19] produce feature maps of a single low resolution and have quadratic computation complexity to input image size due to computation of self-attention globally.

**Figure:** Swin-Transformer im Vergleich zu vorherigen Vision Transformern (wie z.B. Image-GPT). Quelle: Liu et al. (2021): Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

# Hierarchische Vision Transformer

Eine Lösung zum Problem der Berechnungskomplexität von Self-Attention Mechanismen könnte darin bestehen, die Berechnungskosten mit der Modellkapazität abzuwägen. Diese Strategie verfolgen Swin Transformer<sup>3</sup>:

- Die Berechnungskosten der Self-Attention werden mit Hilfe von nicht-überlappenden lokalen Fenstern limitiert, wodurch eine lineare Komplexität in der Bild-Grösse erzielt wird.
- Durch eine hierarchische Architektur, welche Verbindungen zwischen den lokalen Fenstern erlaubt, kann die Verringerung der Modell-Kapazität kompensiert werden.

<sup>3</sup>Liu et al. (2021): Swin Transformer: Hierarchical Vision Transformer using Shifted Windows 60

# Swin Transformer: Architektur

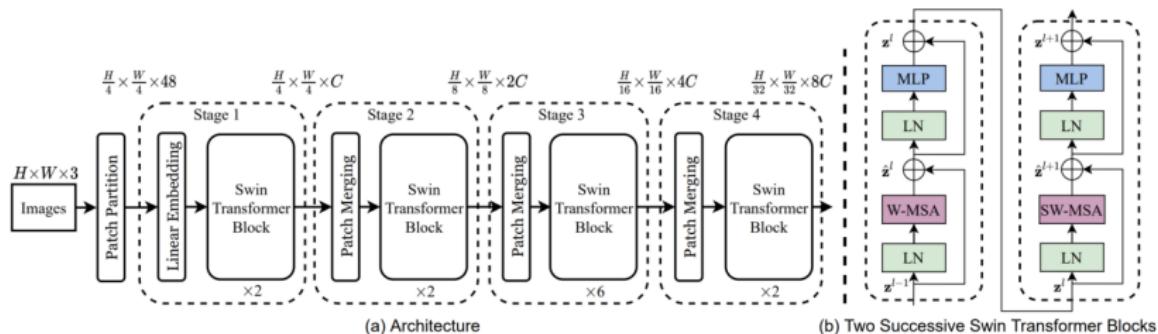


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

**Figure:** Architektur des Swin-Transformers. Quelle: Liu et al. (2021): Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

Zunächst wird die Auflösung der Bilder verringert indem sie in  $4 \times 4$  Blöcke unterteilt werden. Jeder Block mit  $4 \times 4$  Pixeln mit jeweils 3 RGB-Farbkanälen wird dann in einen  $4 \times 4 \times 3 = 48$ -dim Vektor gepackt. Danach wird dieser Vektor mit Hilfe des linear embedding-Layers auf die Modell-Dimension  $C$  projiziert, wobei  $C$  je nach Modell-Variante unterschiedlich gross sein kann (z.B. 96,128,192).

# Swin Transformer: Linear Embedding und Patch Merging

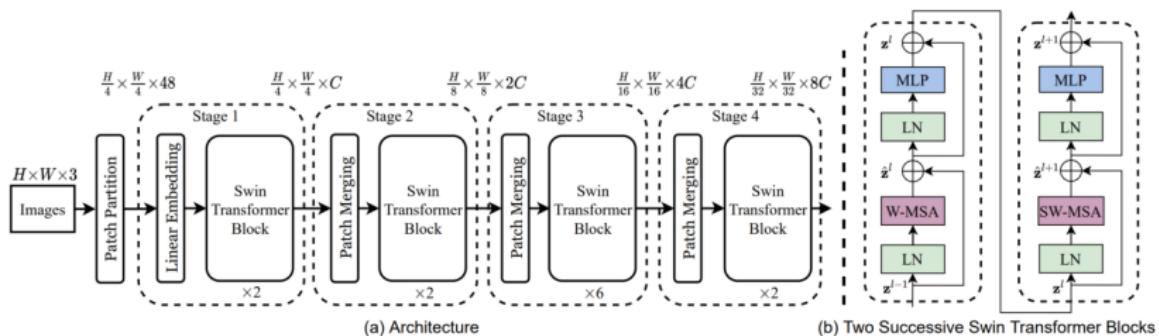


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

**Figure:** Architektur des Swin-Transformers. Quelle: Liu et al. (2021): Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

Danach werden in hierarchischen Stufen Patch-Merging und Swin Transformer Blöcke angewendet.

Patch-Merging konkateniert die Features der Grösse  $C$  aus einer  $2 \times 2$  Nachbarschaft in ein neues Feature der Grösse  $4C$  und projiziert dieses neue Feature dann mit Hilfe eines Linear Layers auf ein Feature der Länge  $2C$  herunter.

# Swin Transformer Block: Shifted Window

Die lokalen Fenster des wMSA (window Multi-Head Self-Attention) Mechanismus verbessern die Skalierbarkeit, reduzieren aber auch die Mächtigkeit des Modells, da Abhängigkeiten zwischen entfernten Pixeln durch die Fenstergröße limitiert werden.

Damit auch Abhängigkeiten zwischen den Fenstern modelliert werden können, wird alternierend zwischen Fensteraufteilungen gewechselt.

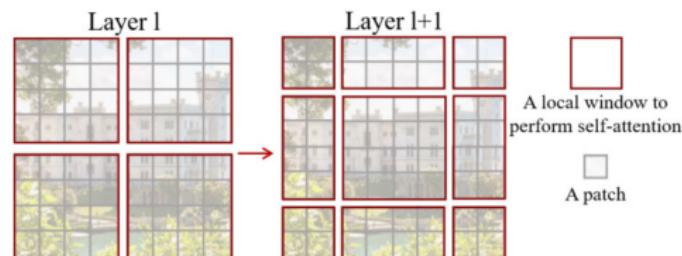


Figure 2. An illustration of the *shifted window* approach for computing self-attention in the proposed Swin Transformer architecture. In layer  $l$  (left), a regular window partitioning scheme is adopted, and self-attention is computed within each window. In the next layer  $l + 1$  (right), the window partitioning is shifted, resulting in new windows. The self-attention computation in the new windows crosses the boundaries of the previous windows in layer  $l$ , providing connections among them.

**Figure:** In dieser Illustration wurden Fenster der Grösse  $M \times M = 4 \times 4$  gewählt. Üblicherweise werden jedoch grössere Fenster mit  $M = 7$  verwendet.

# Self-attention in nicht-überlappenden Fenstern

- Der Swin-Transformer ersetzt den Multi-head Self-Attention (MSA) Mechanismus aus den Transformer-Netzwerken mit einem "window Multi-head Self-Attention" (wMSA) Mechanismus.
- Dieser wMSA Mechanismus unterteilt die Eingabe in  $M \times M$  Patches (wobei  $M = 7$ ). Jeder dieser Patches (Windows) wird durch einen separaten MSA bearbeitet.
- Gegeben einer Eingabe mit Breite  $w$  und Höhe  $h$ , dann ist die Komplexität von wMSA im Vergleich zu MSA nur linear von der Anzahl Pixel  $hw$  abhängig:

$$\Omega(\text{MSA}) = 4hwC^2 + 2(hw)^2C$$

$$\Omega(\text{wMSA}) = 4hwC^2 + 2M^2hwC$$

# Swin Transformer: Resultate

(a) Regular ImageNet-1K trained models				
method	image size	#param.	FLOPs	throughput (image / s)
RegNetY-4G [47]	224 <sup>2</sup>	21M	4.0G	1156.7
RegNetY-8G [47]	224 <sup>2</sup>	39M	8.0G	591.6
RegNetY-16G [47]	224 <sup>2</sup>	84M	16.0G	334.7
EffNet-B3 [57]	300 <sup>2</sup>	12M	1.8G	732.1
EffNet-B4 [57]	380 <sup>2</sup>	19M	4.2G	349.4
EffNet-B5 [57]	456 <sup>2</sup>	30M	9.9G	169.1
EffNet-B6 [57]	528 <sup>2</sup>	43M	19.0G	96.9
EffNet-B7 [57]	600 <sup>2</sup>	66M	37.0G	55.1
ViT-B/16 [19]	384 <sup>2</sup>	86M	55.4G	85.9
ViT-L/16 [19]	384 <sup>2</sup>	307M	190.7G	27.3
DeiT-S [60]	224 <sup>2</sup>	22M	4.6G	940.4
DeiT-B [60]	224 <sup>2</sup>	86M	17.5G	292.3
DeiT-B [60]	384 <sup>2</sup>	86M	55.4G	85.9
Swin-T	224 <sup>2</sup>	29M	4.5G	755.2
Swin-S	224 <sup>2</sup>	50M	8.7G	436.9
Swin-B	224 <sup>2</sup>	88M	15.4G	278.1
Swin-B	384 <sup>2</sup>	88M	47.0G	84.7
				84.2

Method	Backbone	val mIoU	test score	#param.	FLOPs	FPS
DANet [22]	ResNet-101	45.2	-	69M	1119G	15.2
DLab.v3+ [10]	ResNet-101	44.1	-	63M	1021G	16.0
ACNet [23]	ResNet-101	45.9	38.5	-	-	-
DNL [68]	ResNet-101	46.0	56.2	69M	1249G	14.8
OCRNet [70]	ResNet-101	45.3	56.0	56M	923G	19.3
UperNet [66]	ResNet-101	44.9	-	86M	1029G	20.1
OCRNet [70]	HRNet-w48	45.7	-	71M	664G	12.5
DLab.v3+ [10]	ResNeSt-101	46.9	55.1	66M	1051G	11.9
DLab.v3+ [10]	ResNeSt-200	48.4	-	88M	1381G	8.1
SETR [78]	T-Large <sup>†</sup>	50.3	61.7	308M	-	-
UperNet	DeiT-S <sup>†</sup>	44.0	-	52M	1099G	16.2
UperNet	Swin-T	46.1	-	60M	945G	18.5
UperNet	Swin-S	49.3	-	81M	1038G	15.2
UperNet	Swin-B <sup>‡</sup>	51.6	-	121M	1841G	8.7
UperNet	Swin-L <sup>‡</sup>	<b>53.5</b>	<b>62.8</b>	234M	3230G	6.2

Table 3. Results of semantic segmentation on the ADE20K val and test set. <sup>†</sup> indicates additional deconvolution layers are used to produce hierarchical feature maps. <sup>‡</sup> indicates that the model is pre-trained on ImageNet-22K.

Swin-Transformer liefern kompetitive Resultate zu State-of-the-Art CNNs. Dies ist erstaunlich, da es eine sehr neue Architektur ist, die im Vergleich zu CNNs keinen starken intrinsischen Prior für Translations-Invarianz besitzt.

## Abschnitt 5

# Contrastive Language-Image Pre-training (CLIP)

Aus "Learning Transferable Visual Models From Natural Language Supervision"  
von Alec Radford et al. (2021)

# CLIP

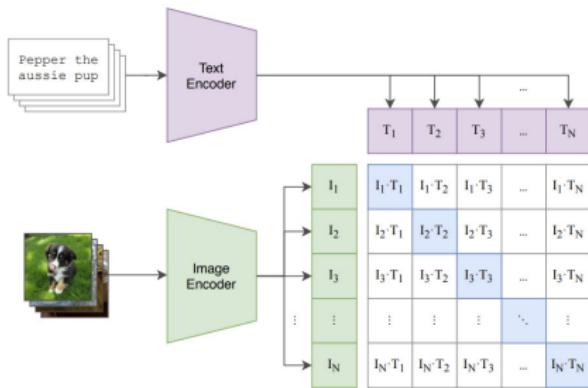
## Motivation

Können wir ein vortrainiertes Netzwerk ohne weiteres Training und ohne weitere Trainingsdaten (also "0-shot") für weitere Klassifizierungsaufgaben auf neuen Datensets verwenden?

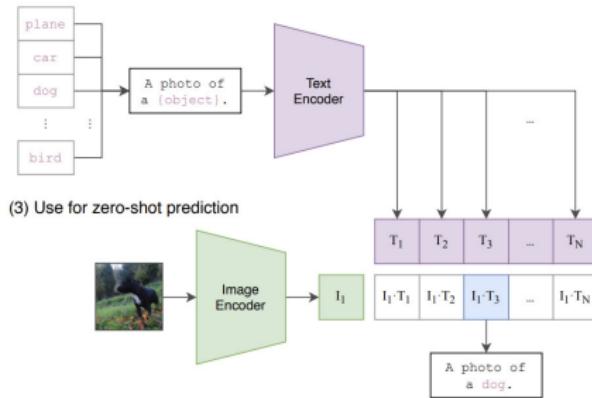
Antwort: Ja, wenn wir Sprach- und Bild-Modelle kombinieren!

# CLIP Architektur

(1) Contrastive pre-training



(2) Create dataset classifier from label text



(3) Use for zero-shot prediction

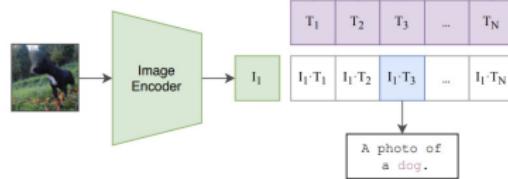


Figure 1. Summary of our approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset’s classes.

# CLIP Architektur

## Image Encoder

Verschiedene Versionen:

- Modifizierte ResNets
- Vision Transformer in unterschiedlichen Größen

## Text Encoder

Transformer - Netzwerk.

## Contrastive Training

Loss: In der Produkt-Matrix stehen  $N$  korrekte Paare und  $N^2 - N$  inkorrekte Paare. Daraus kann ein symmetric cross entropy loss berechnet werden, indem der Softmax und Cross-Entropy loss auf die Produkt-Matrix  $I_i T_j$  sowohl Zeilen- als auch Spaltenweise angewendet und gemittelt wird.

Das Training wurde auf einem datenset mit 400 Millionen (Bild,Text)-Paaren, welche aus dem Internet extrahiert wurden durchgeführt und war entsprechend aufwändig: "The largest ResNet model, RN50x64, took 18 days to train on 592 V100 GPUs while the largest Vision Transformer took 12 days on 256 V100 GPUs."

# CLIP Resultate

	aYahoo	ImageNet	SUN
Visual N-Grams	72.4	11.5	23.0
CLIP	<b>98.4</b>	<b>76.2</b>	<b>58.5</b>

Table 1. Comparing CLIP to prior zero-shot transfer image classification results. CLIP improves performance on all three datasets by a large amount. This improvement reflects many differences in the 4 years since the development of Visual N-Grams (Li et al., 2017).

Figure: [Quelle]

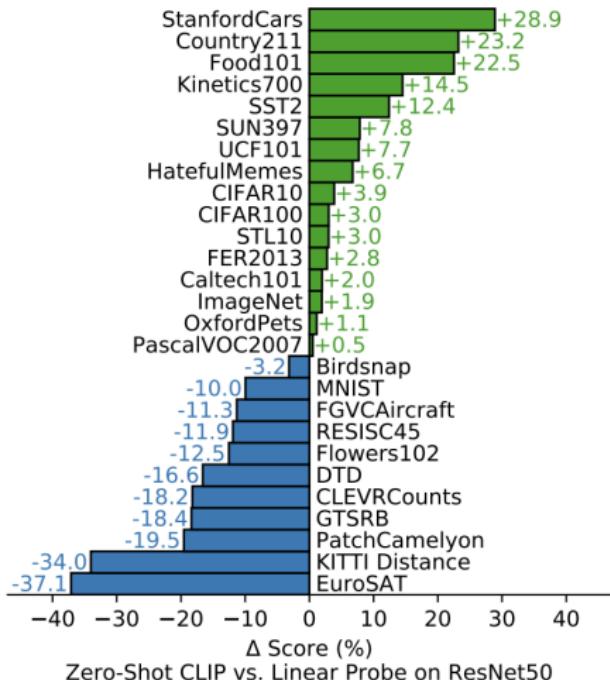


Figure 5. Zero-shot CLIP is competitive with a fully supervised baseline. Across a 27 dataset eval suite, a zero-shot CLIP classifier outperforms a fully supervised linear classifier fitted on ResNet-50 features on 16 datasets, including ImageNet.

# Nebenbemerkung: unCLIP (Dall-e 2)

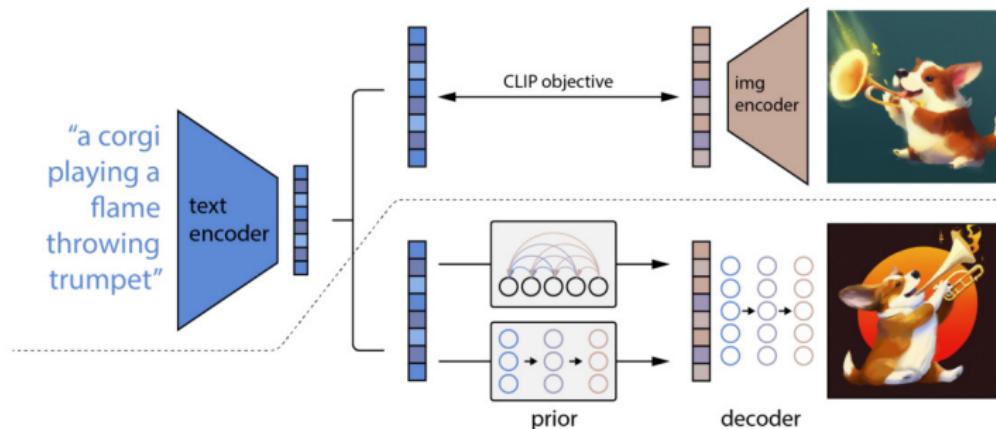


Figure 2: A high-level overview of unCLIP. Above the dotted line, we depict the CLIP training process, through which we learn a joint representation space for text and images. Below the dotted line, we depict our text-to-image generation process: a CLIP text embedding is first fed to an autoregressive or diffusion prior to produce an image embedding, and then this embedding is used to condition a diffusion decoder which produces a final image. Note that the CLIP model is frozen during training of the prior and decoder.

**Figure:** unCLIP (Dall-e 2) erlaubt Bilder aus Textbeschreibungen zu erzeugen. Dazu werden die text embeddings eines CLIP modells zunächst mit Hilfe eines Autoregressiven oder Diffusion Modells in image embeddings umgewandelt. Diese können dann verwendet werden, um ein Diffusion Modell zu konditionieren, welches das Ausgabebild produziert.  
[Quelle]