Lecture 6: Advanced Transformers

Advanced deep learning

Rémy Sun

remy.sun@inria.fr

Course organization

- Goal: In-depth understanding of important Deep Learning staples
  - Reinforce what you have already seen
  - Introduce state of the art models

- This is a hands-on course in pytorch
  - Minimal math
    - Enough to understand
  - Quite a bit of coding
  - Get comfortable with the standard pipeline

Course organization: 10 Lectures

UNIVERSITÉ CÔTE D'AZUR

- L1-2: Overview of Deep Learning (F. Precioso)
- L3-4: Fundamentals of Deep Learning (R. Sun)

- **L5-6: Transformers (R. Sun)**
- L7: Large models (LLMs, VLMs, Generators) (R. Sun)

- L8: Tricks of the trade (R. Sun)

- L9: Ethics of AI (F. Precioso)

- L10: Intro to generative models (P-A. Mattei)

Today!

- Goal: Solidify understanding of basic transformer blocks
  - Attention in particular

- See how transformers can be used in practice
  - Natural Language Processing
  - Image processing
  - Object detection
  - Semantic segmentation
  - Pretraining procedure on large data

Before we start

How does pytorch work?

- Create neural network
  - Use the torch.nn modules
    - Can use torch.nn.Modules
    - Or create a new class inheriting from torch.nn modules

- The training loop is

How does pytorch work?

- Create neural network
  - Use the torch.nn modules
    - Can use torch.nn.Modules
    - Or create a new class inheriting from torch.nn modules

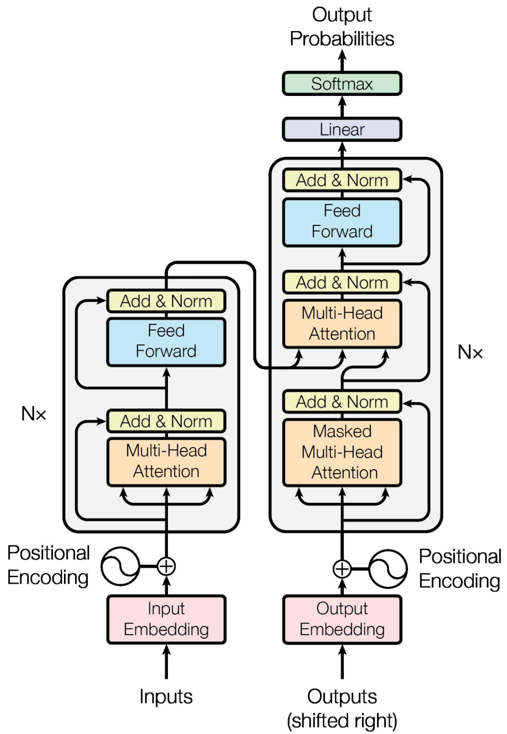
- The training loop is

```
yhat = model(X)
L,acc = loss_accuracy(loss,yhat,Y)
optim.zero_grad()
L.backward()
optim.step()
```

Refresher on Transformers

A new type of neural layer for everying

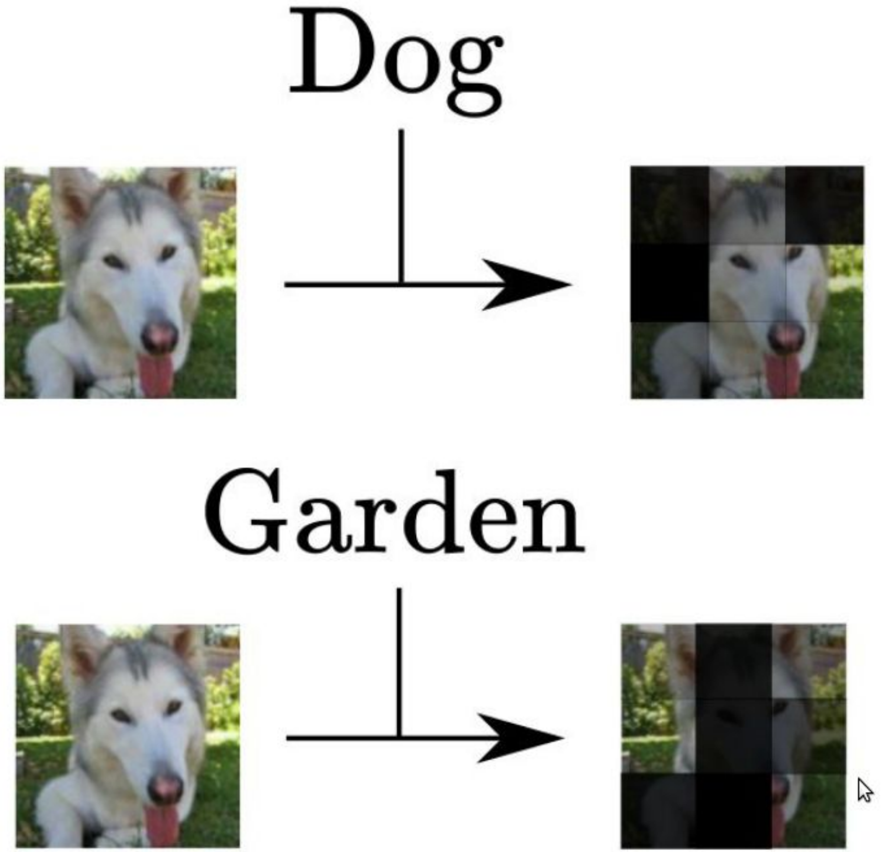
- What is the state of the art in
  - Computer vision?
    - ~~CNNs~~ -> **Transformers**
  - Natural language processing?
    - ~~RNNs~~ -> **Transformers**
  - Time series?
    - ~~RNNs or TCNs~~ -> **Transformers**
  - Multimodal problems?
    - ~~Hybrid?~~ -> **Transformers**



- Transformers use keeps increasing over time

Main idea

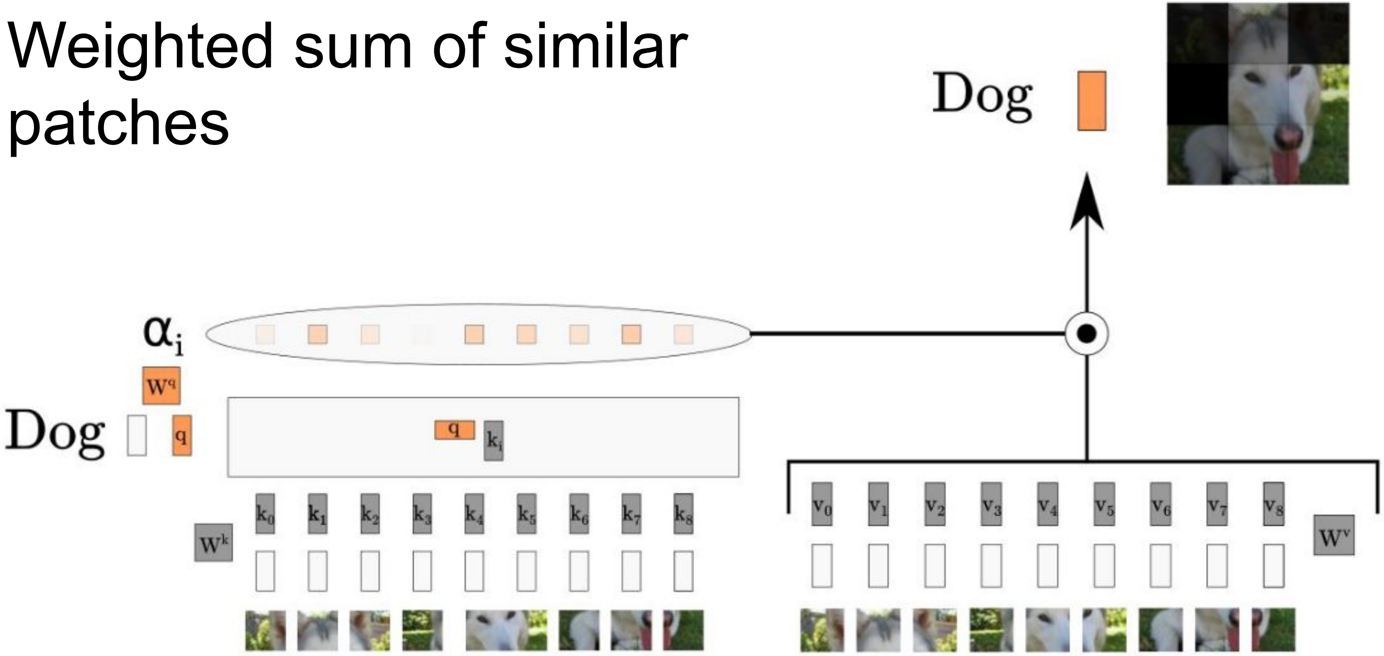
UNIVERSITÉ CÔTE D'AZUR



- What is a Dog?
    - It is a 4 legged animal with fur and ears and eyes and a head and …
    - It is on this part of that picture.

Attended representation

- Weighted sum of similar patches

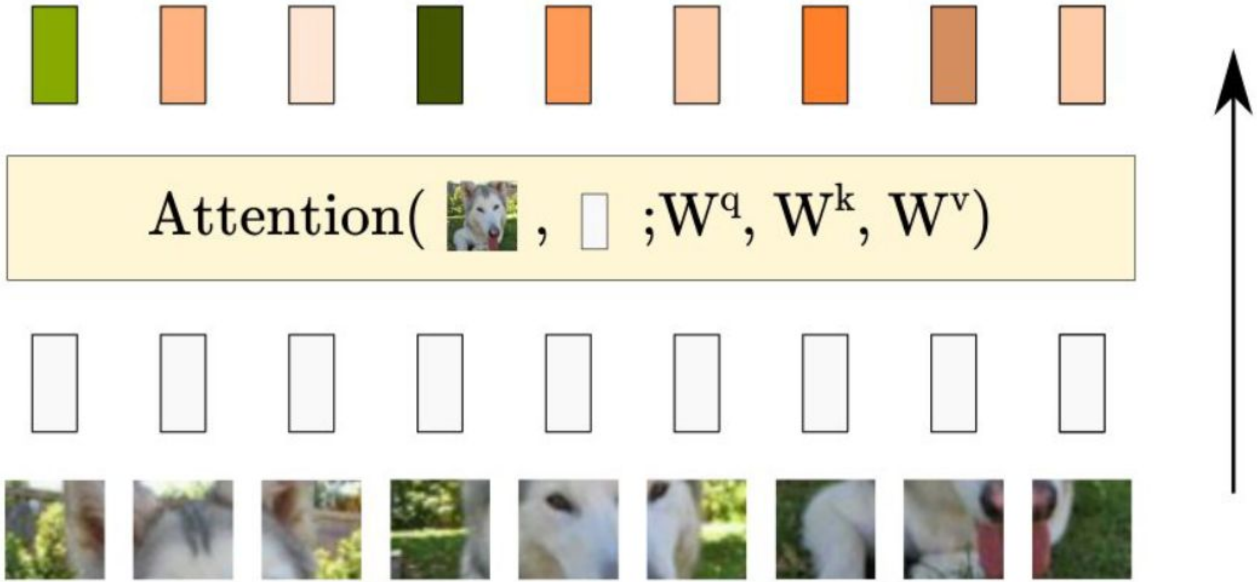


Attended representation


```python
def scaled_dot_product(q, k, v, mask=None):
    d_k = q.size()[-1]

    #######################
    ### YOUR CODE HERE! ###
    #######################

    # Compute attn_logits
    attn_logits = torch.matmul(q, k.transpose(-2, -1))
    attn_logits /= math.sqrt(d_k)

    # Apply mask if not None
    if mask is not None:
        attn_logits = attn_logits.masked_fill(mask == 0, - 1e14)

    # Pass through softmax
    attention = F.softmax(attn_logits, dim=-1)

    # Weight values accordingly
    output_values = torch.matmul(attention, v)

    #######################
    ###       END       ###
    #######################

    return output_values, attention
```

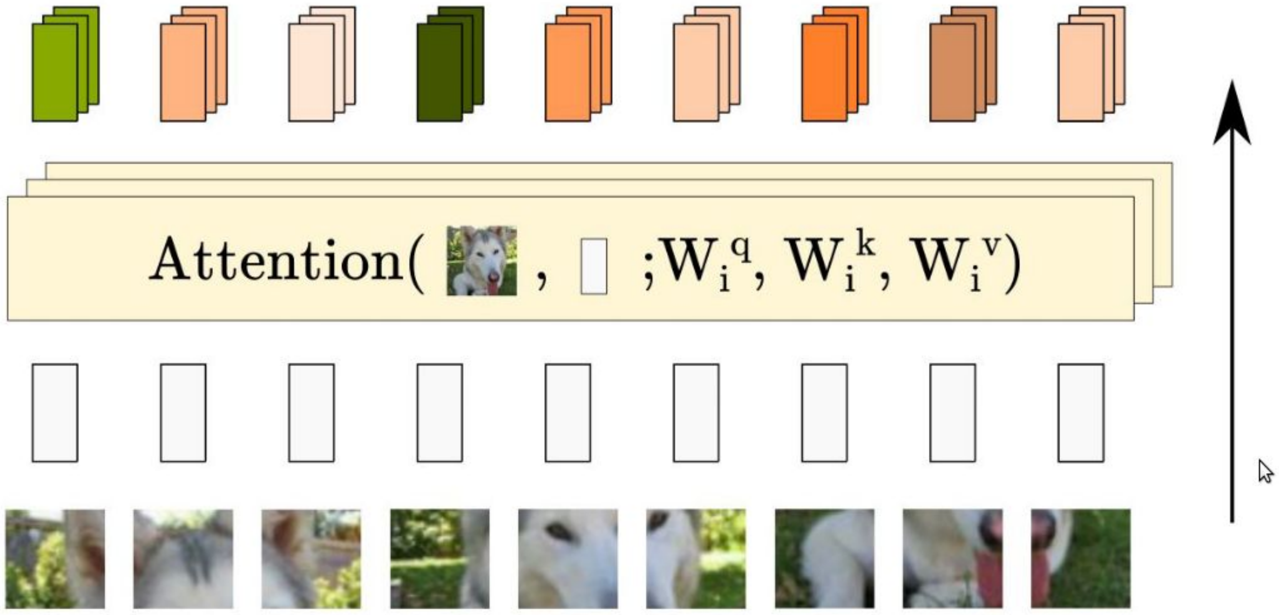
```python
def forward(self, x, mask=None, return_attention=False):

    #######################
    ### YOUR CODE HERE! ###
    #######################

    batch_dim, seq_length, input_dim = x.shape

    # Compute linear projection for qkv and separate heads
    # QKV: [Batch, Head, SeqLen, Dims]
    qkv = self.qkv_proj(x) # Batch x SeqLen x Hidden_dim * 3
    qkv = qkv.reshape(batch_dim, seq_length, self.num_heads, 3* self.head_dim)
    qkv = qkv.permute(0, 2, 1, 3)
    q, k, v = qkv.chunk(3, dim=-1)


    # Apply Dot Product Attention to qkv ()
    attention_values, attention = scaled_dot_product(q, k, v)

    # Concatenate heads to [Batch, SeqLen, Embed Dim]
    attention_values = attention_values.reshape(batch_dim, seq_length, self.embed_dim)

    # Output projection
    o = self.o_proj(attention_values)

    #######################
    ###       END       ###
    #######################

    if return_attention:
        return o, attention
    else:
        return o
```

- You can compare a sequence to itself!



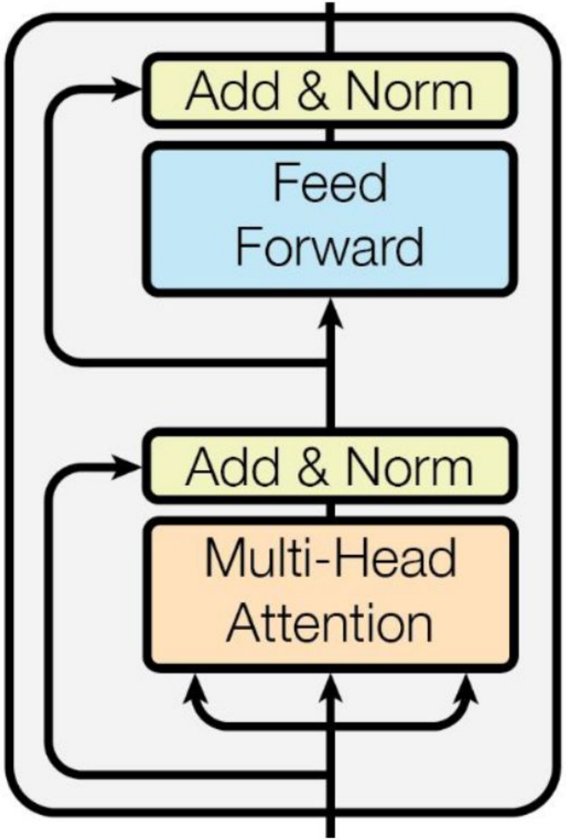
MultiHead

- And even have multiple interpretations



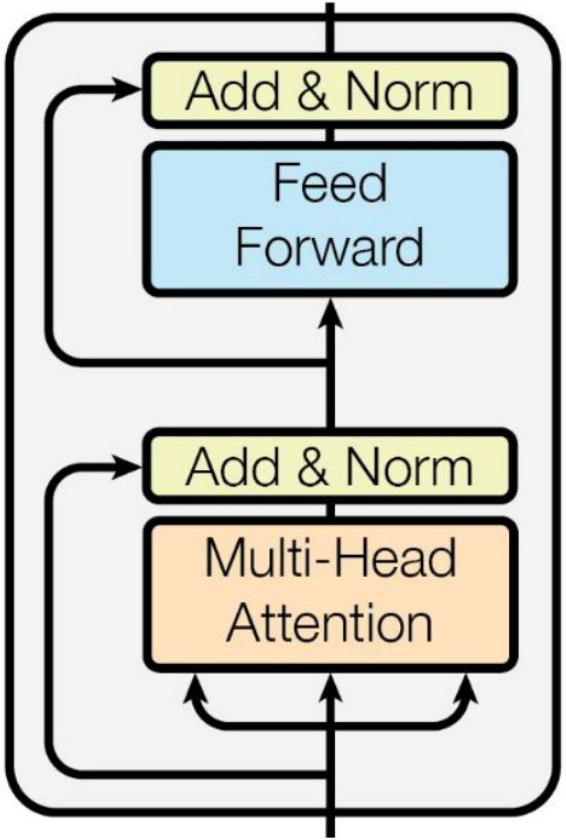
- Basic encoder block
  - MultiHead attention
  - Skip connection
  - Small MLP applied to each token
  - Skip connection

- Stacked in a transformer



Transformer (Encoder) Block

```python
class EncoderBlock(nn.Module):
    def __init__(self, input_dim, num_heads, dim_feedforward, dropout=0.0):
        """
        Args:
            input_dim: Dimensionality of the input
            num_heads: Number of heads to use in the attention block
            dim_feedforward: Dimensionality of the hidden layer in the MLP
            dropout: Dropout probability to use in the dropout layers
        """
        super().__init__()

        # Create Attention layer
        self.self_attn = MultiheadAttention(input_dim, input_dim, num_heads)

        # Create Two-layer MLP with dropout
        self.mlp = nn.Sequential(
            nn.Linear(input_dim, input_dim*2),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(2*input_dim, input_dim)
        )
        # Layers to apply in between the main layers (Layer Norm and Dropout)
        self.norm = nn.Sequential(
            nn.LayerNorm(input_dim),
            nn.Dropout(dropout)
        )

    def forward(self, x, mask=None):
        # Compute Attention part
        attn=self.self_attn(x)
        x=self.norm(attn+x)

        # Compute MLP part
        x = self.norm(x+self.mlp(x))

        return x
```



A freeform operator

- Attention can implement a lot of different operations
  - Little built-in bias
    - As opposed to CNNs
  - Adapts to data
    - Can change depending on the input

- A lot of work has been done to “discover” good operators
  - To little avail
  - But attention kind of does that!

Very strong expressive power

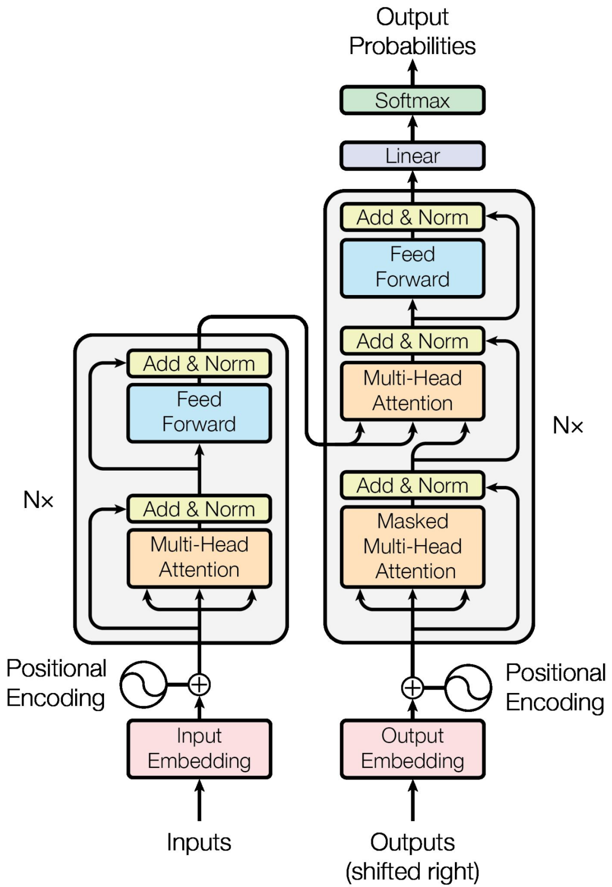
UNIVERSITÉ CÔTE D'AZUR

- Transformers are able to leverage large datasets
  - Because they can learn more adapted relations
  - Becoming more and more adopted
  - Scale very well
- Emerging as the dominant neural network type
- Drawbacks
  - Quadratic cost with the number of tokens
  - Need lots of data or strong regularization

Transformers for Natural Language Processing

Seminal success: Vaswani et al. 2017

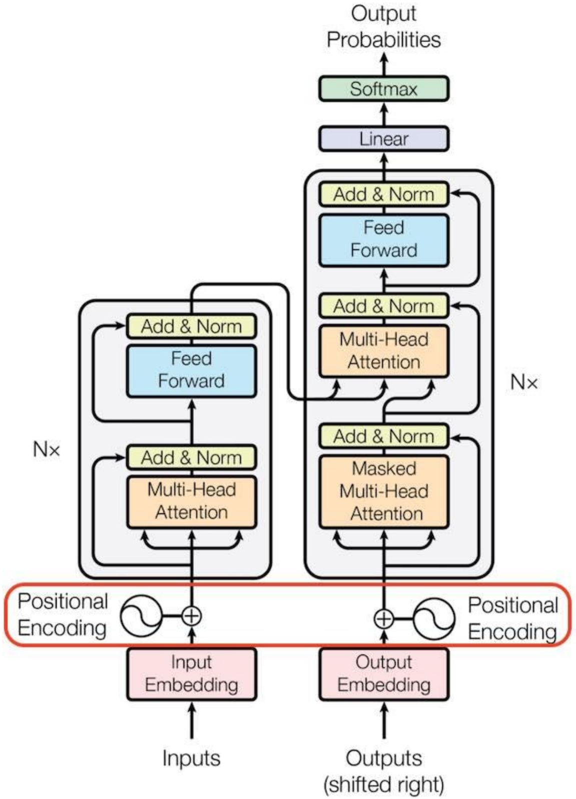
- Completely does away with recurrent units
  - Attention as a first class citizen!
  - Introduces element wise MLP for transform
- Transformer
  - Transforms the input throughout the layers
  - Also to blame for BERT, ELMO, DALL-E, ...



Positional encoding

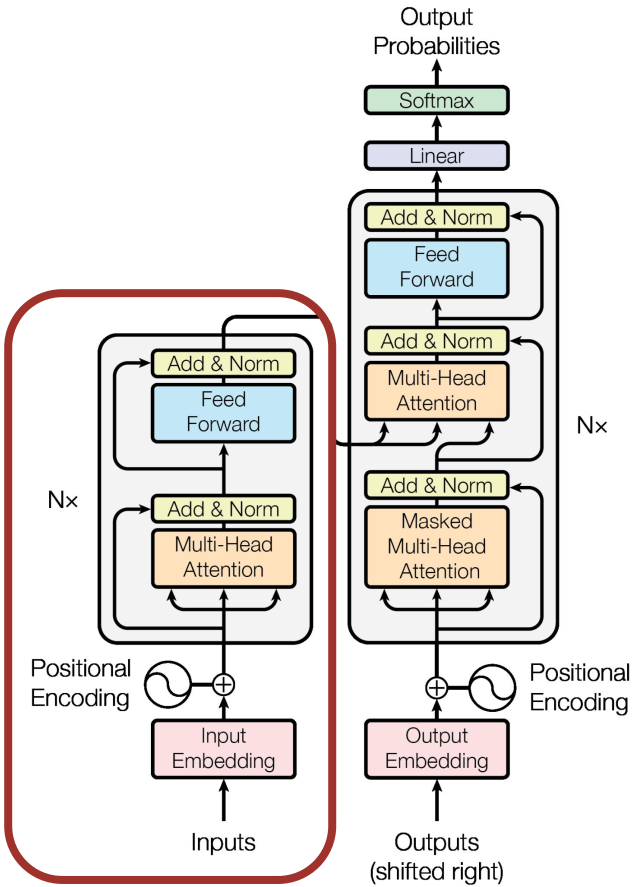
- What is the order of the tokens?
  - Treated as a set
  - Permutation invariant

- How do keep positional info?
  - Masking
  - Add a positional encoding
    - Sine encoding
    - Learned encoding



Transformer Encoder

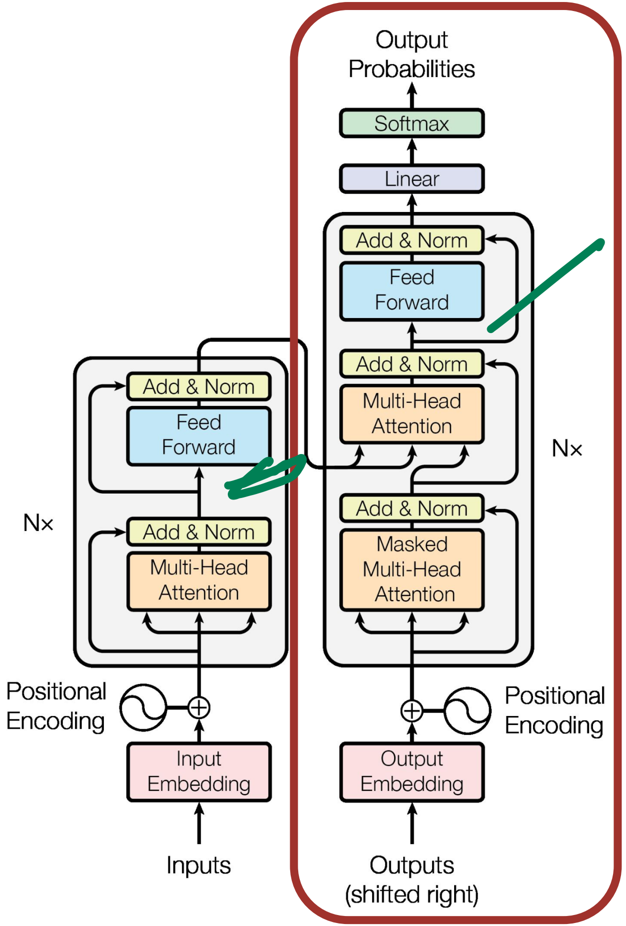
- Sometimes used on its own for downstream tasks
- Extracts useful features
- Simple framework
  - Self attention
  - Element-wise MLP



Transformer Decoder

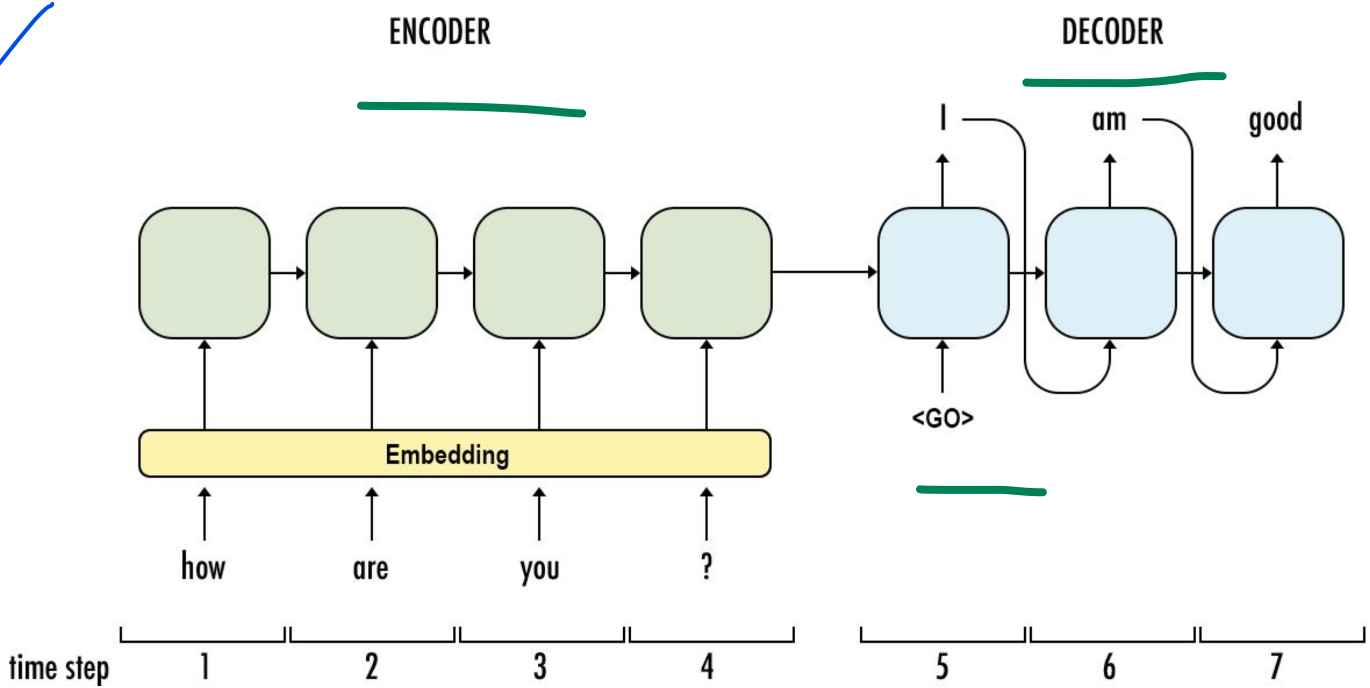
UNIVERSITÉ CÔTE D'AZUR

- Takes input queries and outputs word predictions
- Used on its own for language generation
- Simple framework
    - Masked self-attention
    - (Cross attention) with encoder features
    - Element-wise MLP



Autoregressive prediction

- Predict next token, feed it back into the decoder

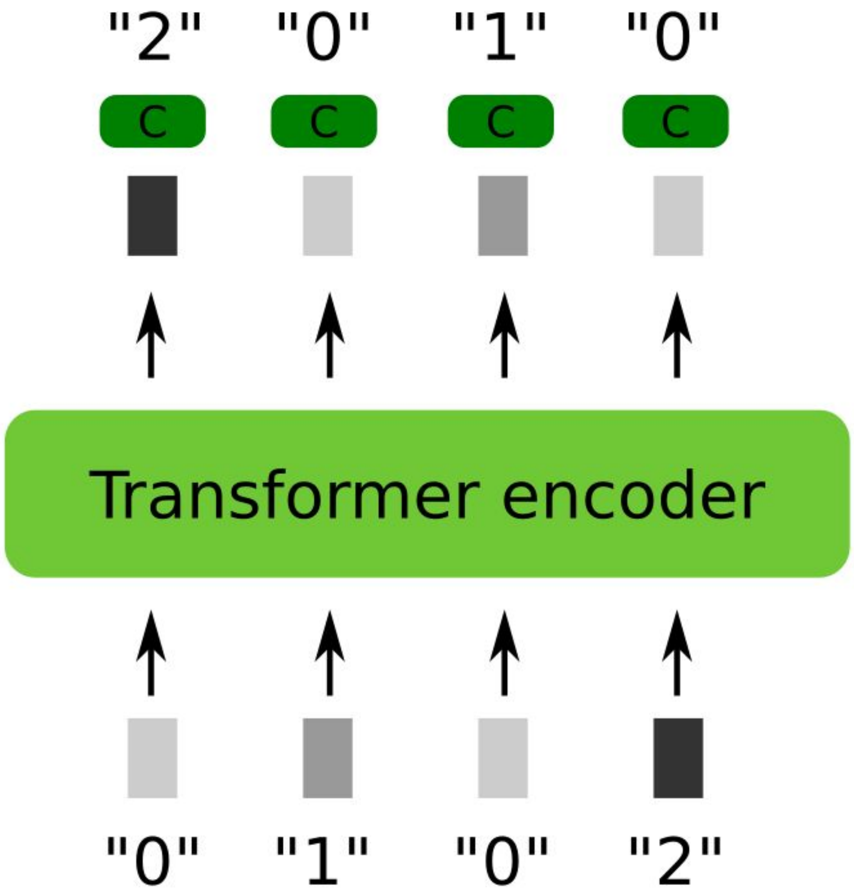


Takeaway

- Similar transformer blocks can be used
  - To extract information
  - To decode information
- Often used for language modeling with auto-regressive predictions
- State of the art on Language for a long time!

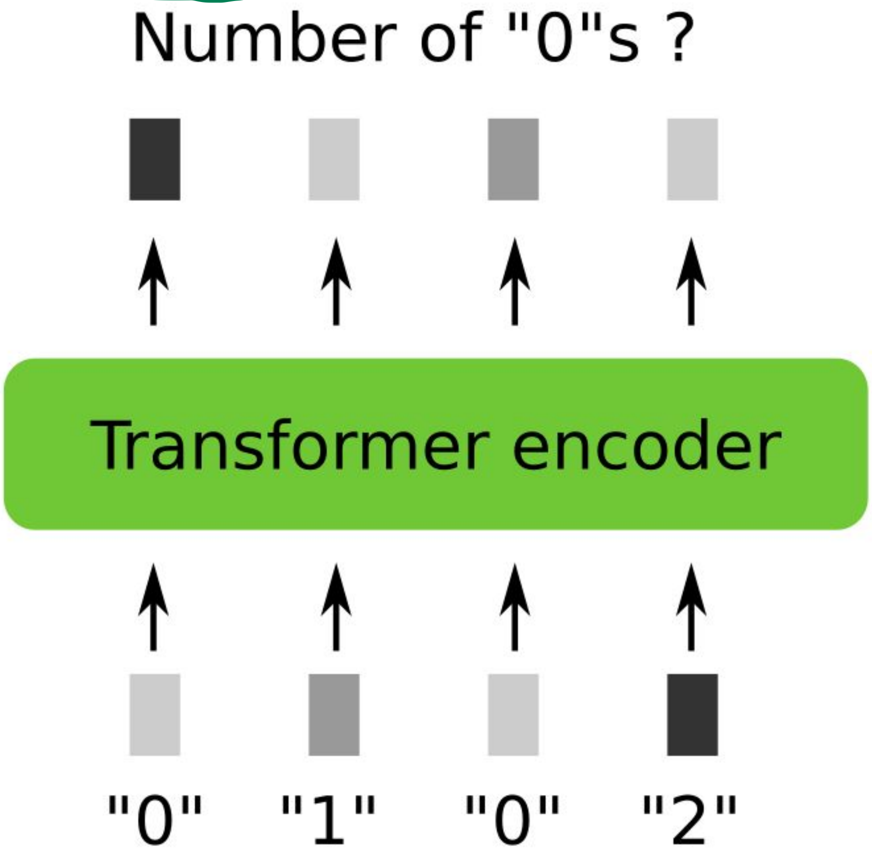
Classification token

Token level predictions



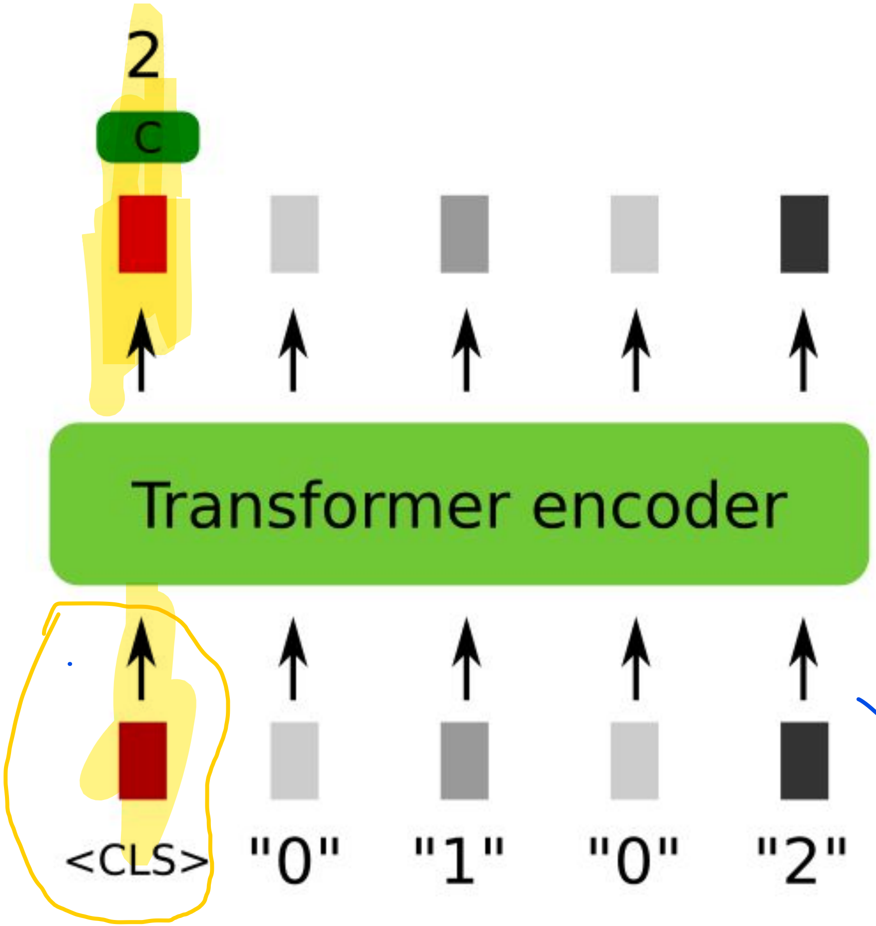
- Rich token features
  - Due to attention!
- Predictions for each token
  - Invert sequence here
- Predict with linear layers

Global predictions



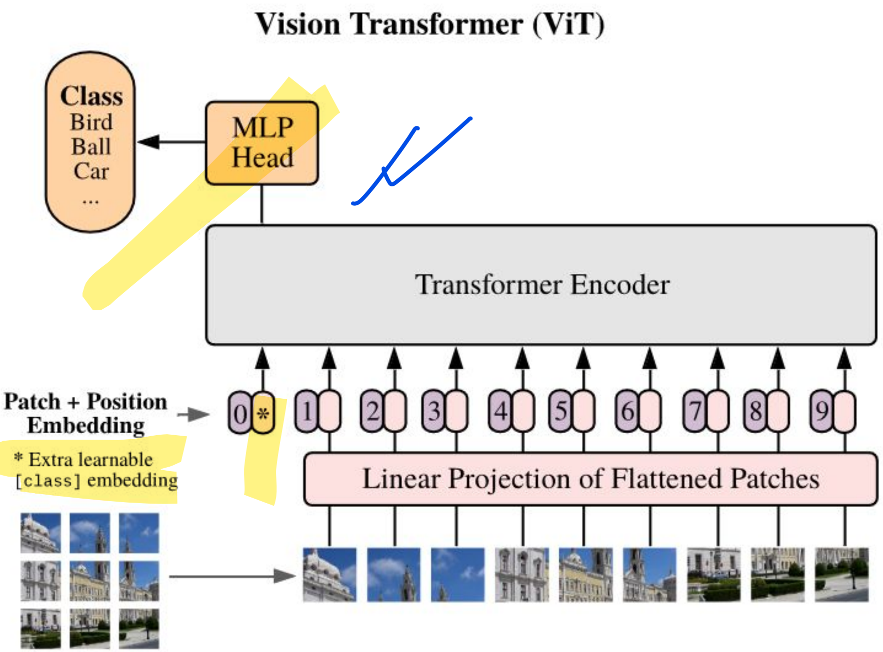
- Rich token features
  - Due to attention!
- One prediction to get
  - Number of 0s
- What to use for prediction?
  - Linear layer?
    - On what?

Classification token



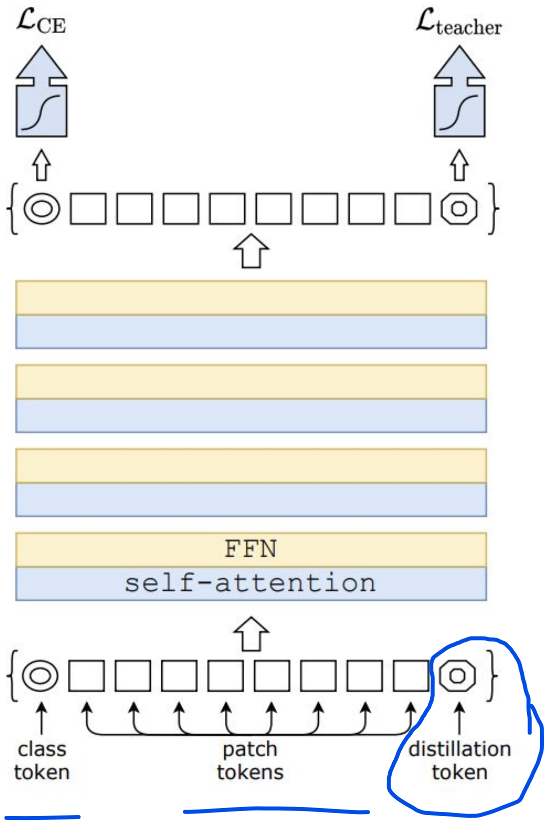
- Rich token features
  - Due to attention!
- Question/Classification token
  - "What is the class"
- Accumulate relevant info from other tokens

Vision Transformer (Dosovitskiy et al, 2021)



- Token = Patch
  - Rich features
- Accumulate relevant info from other tokens
- State of the Art for Computer Vision
  - Data hungry

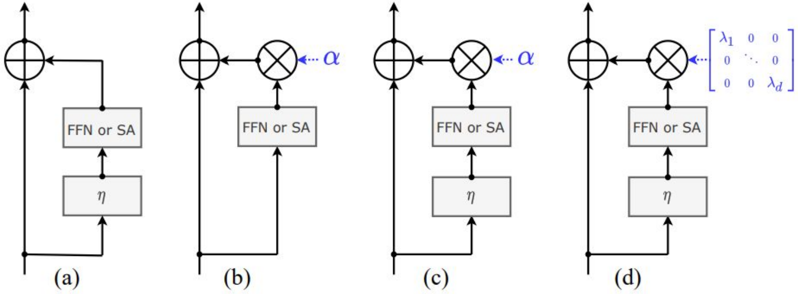
Data efficient Image T. (Touvron et al. 2021)

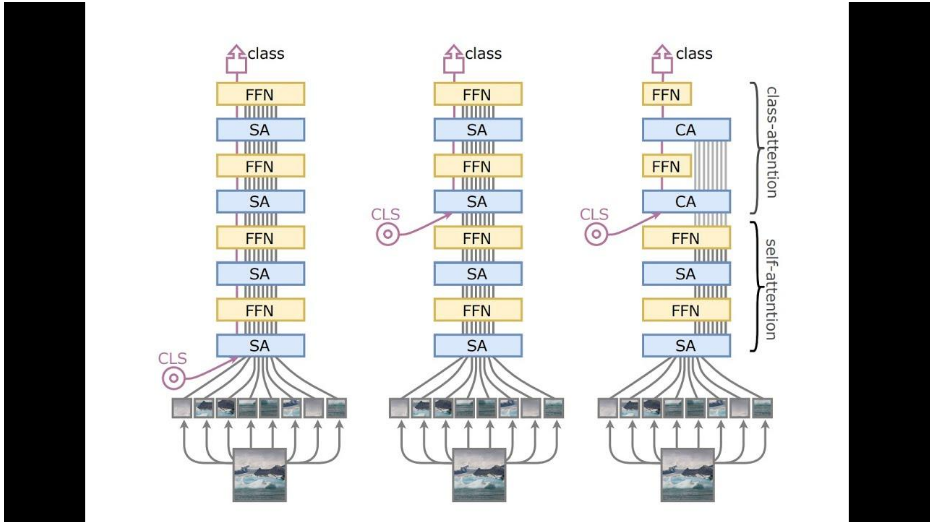


- State of the Art for Computer Vision
  - Data hungry
- Solution: Strong regularization
  - Works with "only" ImageNet data

Class att. image T. (Touvron et al. 2021)

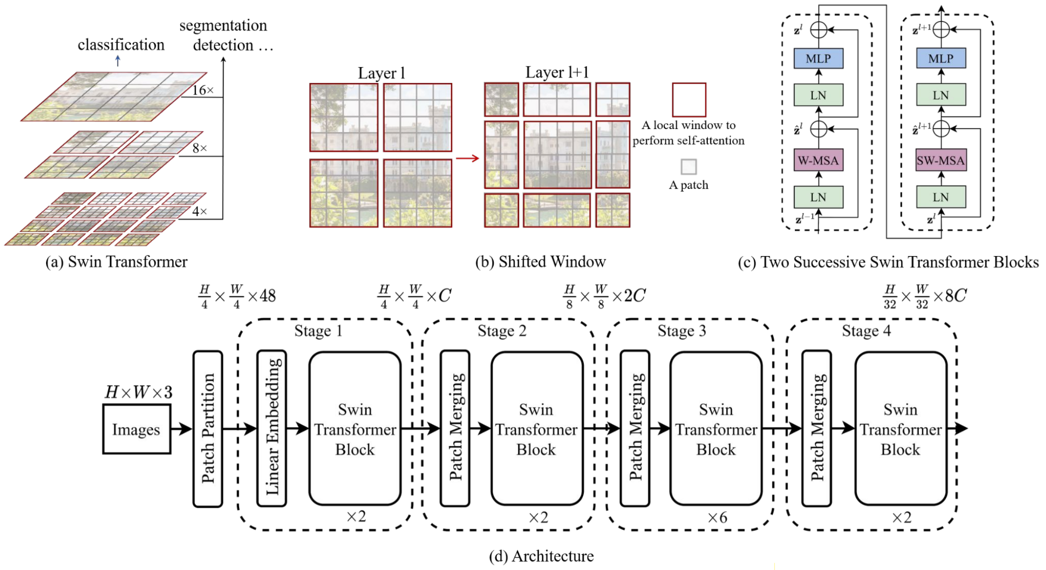
UNIVERSITÉ CÔTE D'AZUR





- Difficulty to have deep transformers
- Issue with residual scaling
- No need to start with cls token
  - Class attention

Swin (Liu et al. 2021)



- More efficient attention types (polynomial costs wrt to number of tokens)
  - Factorization of attention

Takeaway

UNIVERSITÉ CÔTE D'AZUR

- Global tasks require different aggregation
- Dedicated classification token
  - "What does the data say for <task>?"
- Linear classifiers on token features
- Can be extended to images with image patches
  - Data hungry

Pre-training in Transformers

Pretraining neural networks

- Problem with supervised training
  - We need labels for training
  - Labels are hard to get

- What if we train without manual labels?
  - “Free” training
  - Huge amounts of available data!

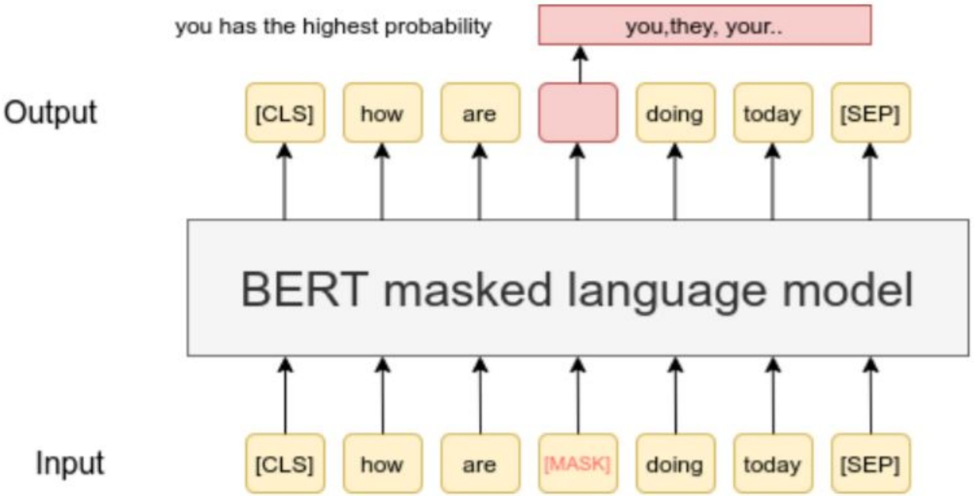
Pretraining neural networks

- Problem with supervised training
  - We need labels for training
  - Labels are hard to get

- What if we train without manual labels?
  - “Free” training
  - Huge amounts of available data!

How?

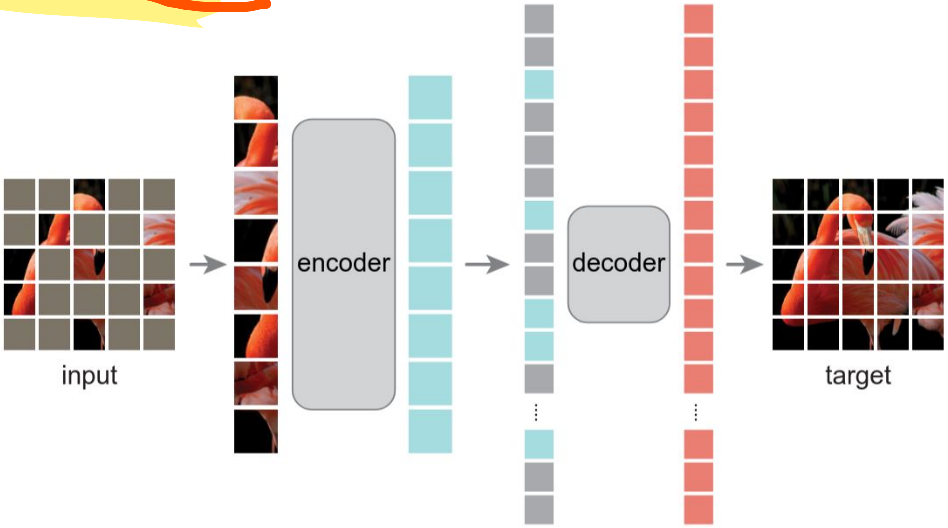
BERT pretraining



- Pseudo-objective
  - Mask part of the sentence
  - Try to predict the masked part

The [CLS] token specifically doesn't represent any actual word; rather, it's a placeholder that captures a summary of the entire input sequence.

Masked auto-encoders



- Also works for images!
- Originally introduced in BeIT paper

Vision Transformers as backbones

- Extract rich features from images
  - Including long range dependencies
- Masked token prediction for pretraining
  - Only works with transformers
  - Much better than traditional contrastive training
- Replace CNNs as backbones for downstream tasks
  - Segmentation
  - Detection

Vision Transformers as backbones

UNIVERSITÉ CÔTE D'AZUR

- Extract rich features from images
  - Including long range dependencies

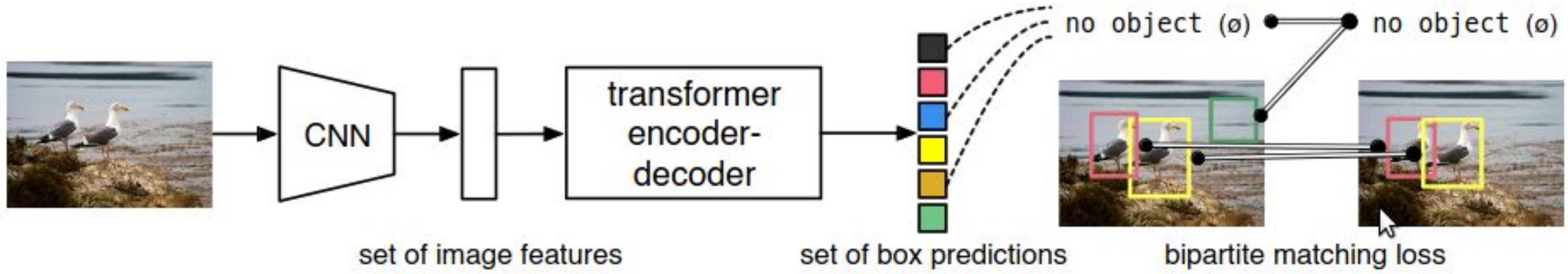
- Masked token prediction for pretraining
  - Only works with transformers
  - Much better than traditional contrastive training

- Replace CNNs as backbones for downstream tasks
  - Segmentation
  - Detection

Can we do a bit more?

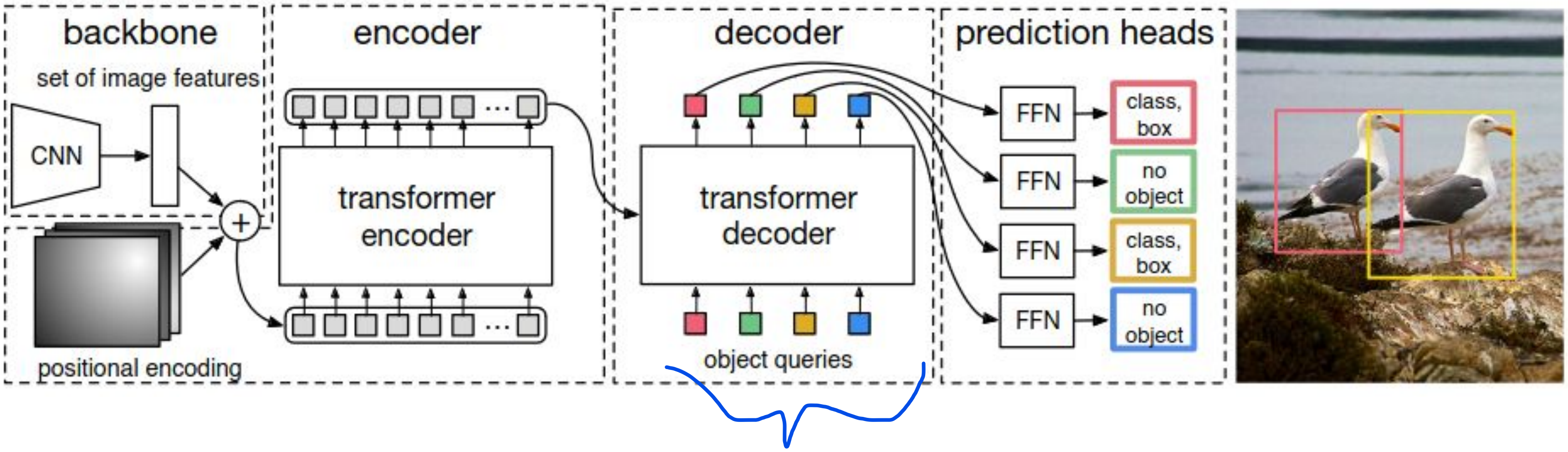
Learnable detection prompts

Intro



- Object Detection
- Transformer based
- N detection proposals
    - Set loss optimization
- One of the first "working" image transformers
    - Lots of issues
- Large legacy for other problems: DETR-like models

Overview

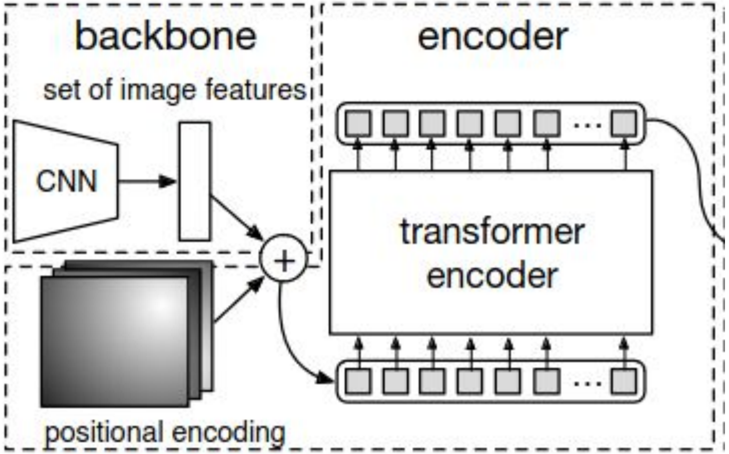


- Embed patches -> Encode -> Decode queries -> Predict
  - Empty predictions sometimes

- OK performance

Encoder

- CNN Backbone
  - Possibly linear
- Positional embedding
- Get good image representations

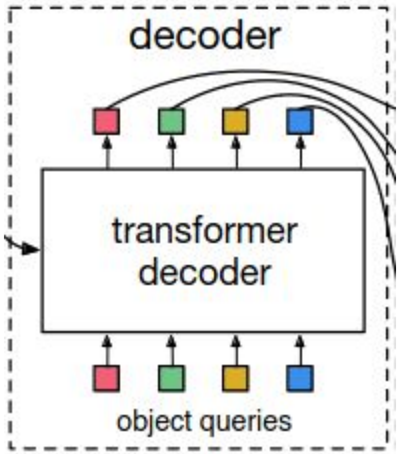


Decoder

- Cross attention Queries/patches
  - Learned queries

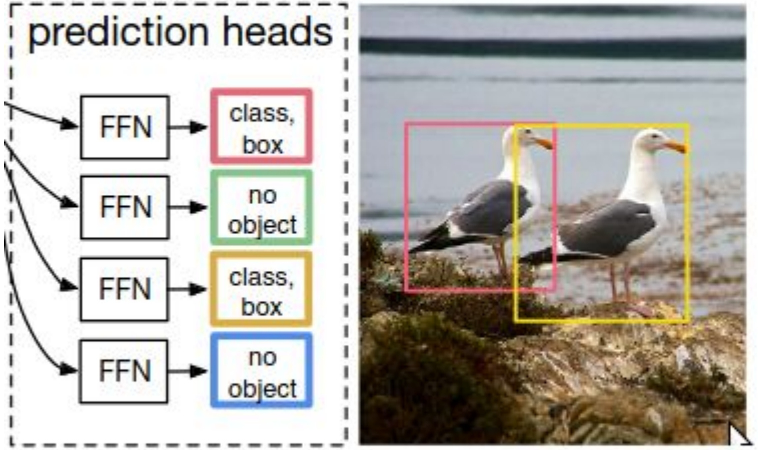
- 1 query = 1 type of proposal

- More queries than needed

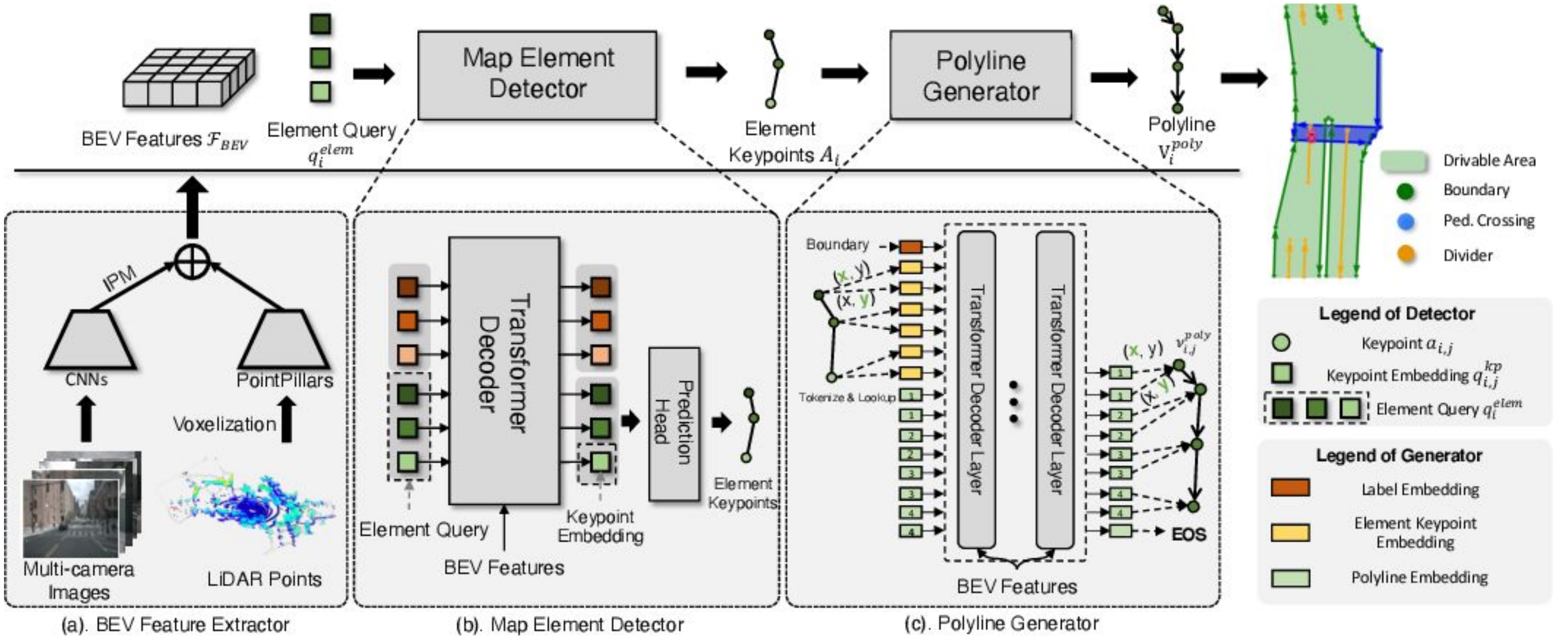


Predict

- Translate queries into proposals
  - Possibly empty
- Optimize with set loss
  - Hungarian alg



Example of downstream applications



Takeaway

UNIVERSITÉ CÔTE D'AZUR

- Detection requires predicting multiple boxes in an image
  - Huge variability in box predictions
  - Varying number of boxes

- Relies on learnable box queries/tokens
  - Box queries accumulate image information
  - Each box specializes separately

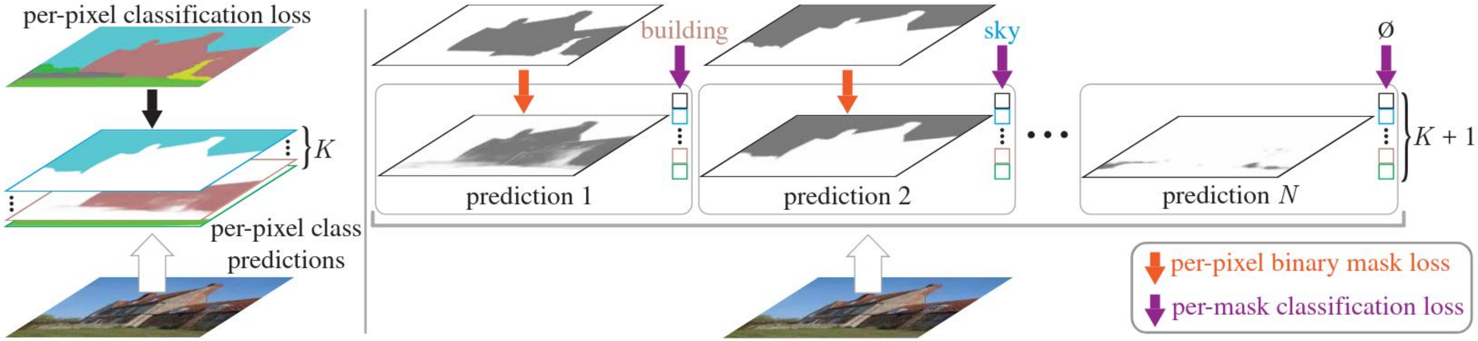
- Predict object class and box coordinates with Linear layers

Learnable segmentation prompts

Intro


void
traffic light
road
sidewalk
building
wall
fence
pole
traffic sign
vegetation
rider
car
terrain
truck
sky
person
bus
train
motorcycle
bicycle

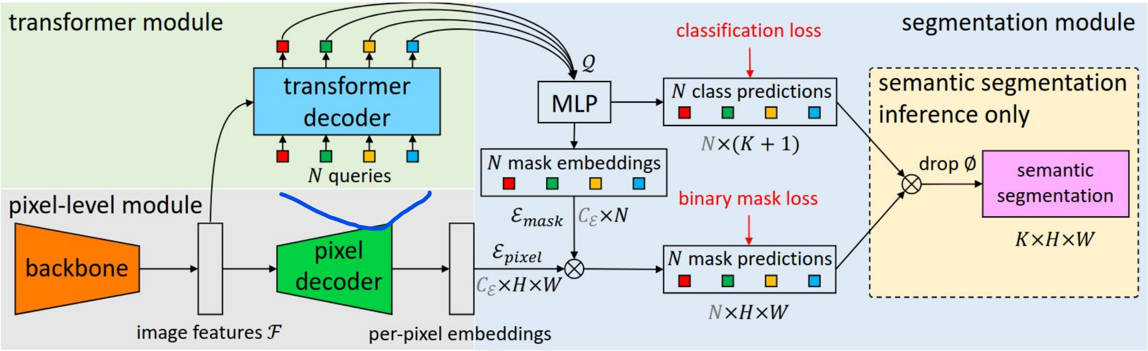
Mask based segmentation



- Basic solution
  - Classify the pixels
- Semantic segmentation
- Transformer based

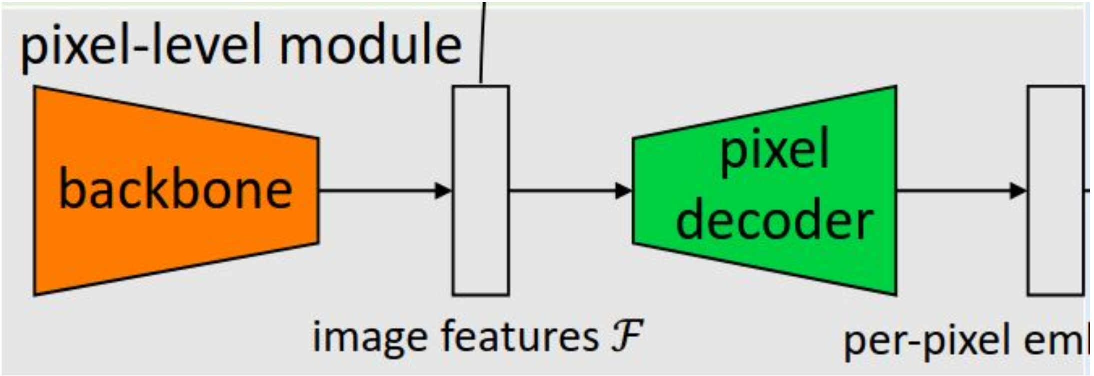
- Advanced solution
  - Maskformer
  - “Detect” masks
- Large legacy for other problems: Maskformer-like

Maskformer framework



- Per pixel features
  - Combined with decoded mask queries for area
- Mask query decoding
  - Decoded both for mask class and area.

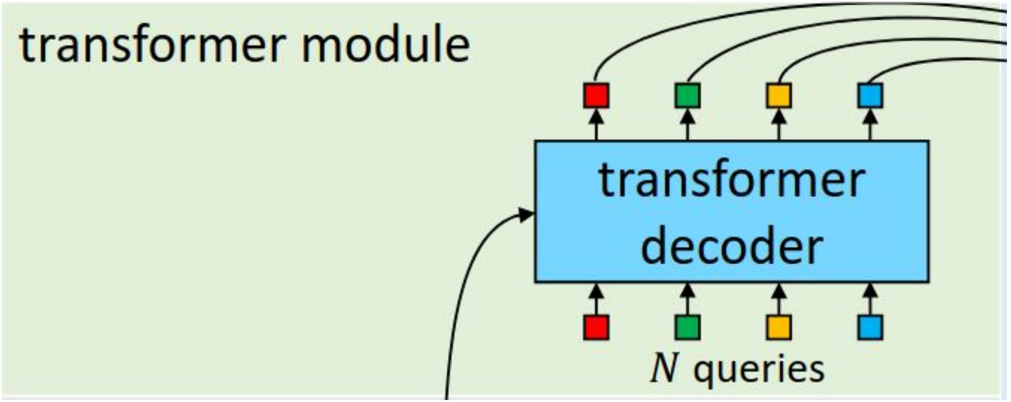
Pixel level module



- Encode features -> decode pixel level predictions
  - What you would naturally expect

- Works well enough on its own

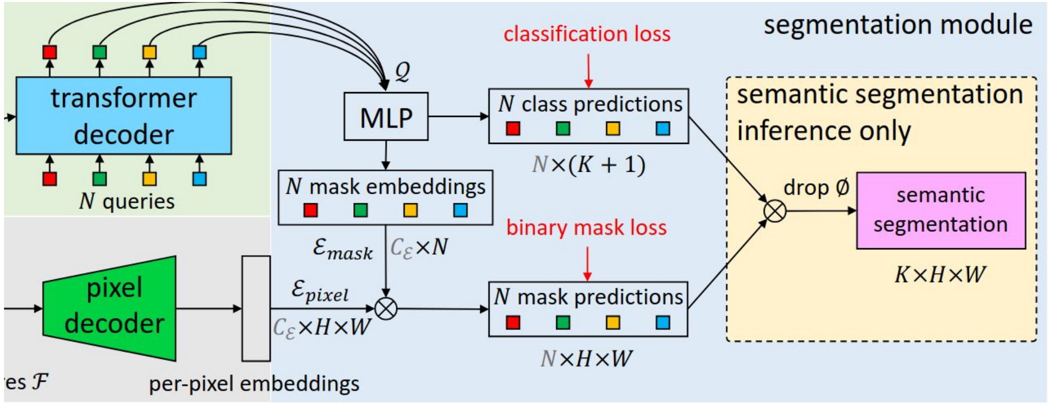
Mask query module



- Encode features -> decode learned mask queries
  - Proposals of "what" is seen

- Very similar to DETR models

Mask query module



- Classify detected masks into semantic classes
  - Like DETR
- Combine detected masks with decoded pixels before predicting mask area

Takeaway

UNIVERSITÉ CÔTE D'AZUR

- Segmentation can be performed very similarly to detection
  - Decode entity queries with image
  - More subtly per pixel prediction area
- Transformers and attention is a very powerful tool
- Transformers can acquire very good input representations
  - With lots of data
  - Masked token prediction is very good for pre-training

- (Re-)Implement
  - Alignment score
  - Attention block
  - Transformer predictor
- Task: Count number of 0s
  - With attention visualization!
- Bonus Task: Detect 0s