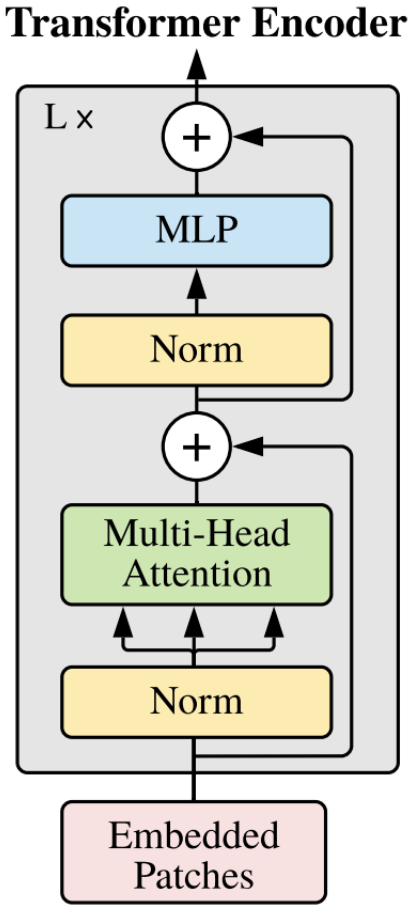
Vision Transformers

Class organization

- I have created a slack channel
- There are quizzes every week, which will build up to a substantial part of your grade

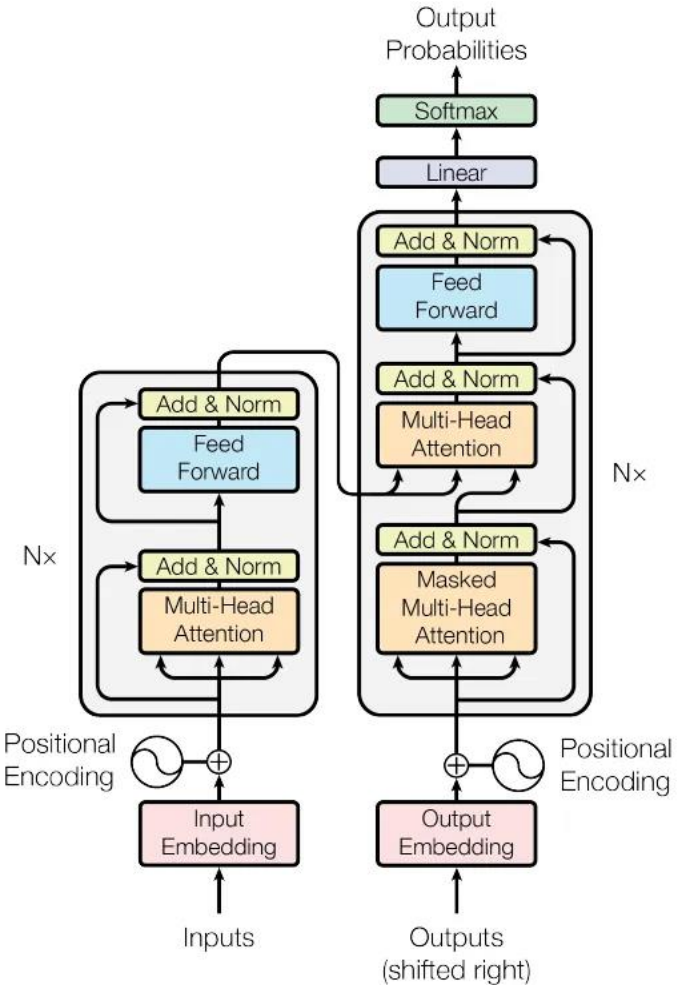
Recap: Transformer basics

- Self-Attention computes a weighted similarity matrix between all input tokens
- This process is replicated in parallel ("multi-head attention") to capture different relationships
  - Bark is a dog and he is cute:
    - Bark is a dog
    - Bark is cute
    - Bark is a he
- Layer normalization and residual connections
- Multi-layer perceptron then performs classification
- Repeat structure L times

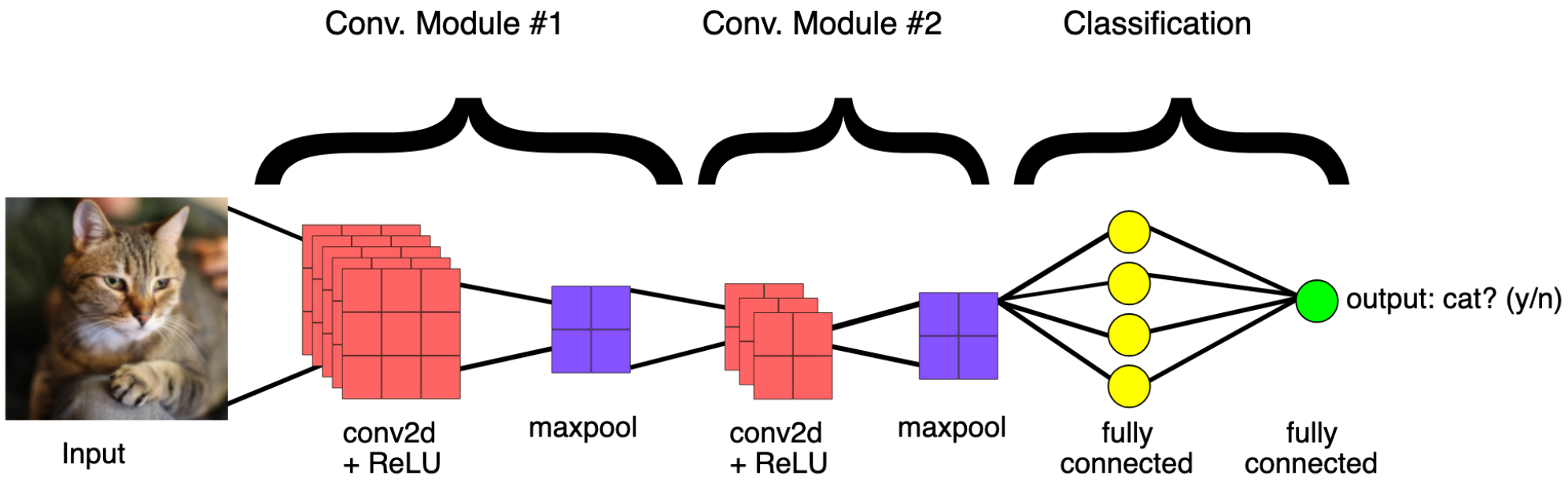


Recap: Transformer

- The transformer trains faster (vs. recurrent network) and is *parallelizable*
- Previously unheard of 100 billion parameters became doable to train
- Large language models consist of two parts
  - Transformer Decoder (Masked self-attention)
  - Transformer Encoder (Full self-attention)

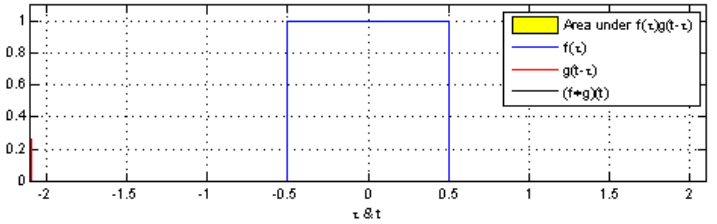


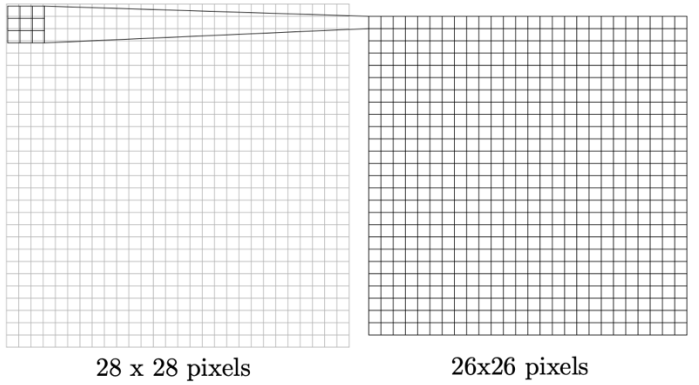
Other application: image classification

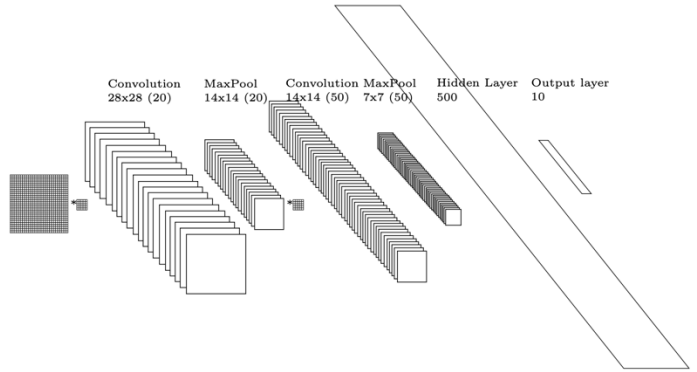


Excurse: Convolutional Neural Networks

- CNNs offer “inductive bias”
- Translation Equivarence
  - Pro: it does not matter where a feature is on an image
  - Con: its hard to learn spatial relations between features
- Locality
  - Pro: Relevant information is next to each other, e.g. an edge, a corner
  - Con: its hard to capture cues that rely on information that is far apart

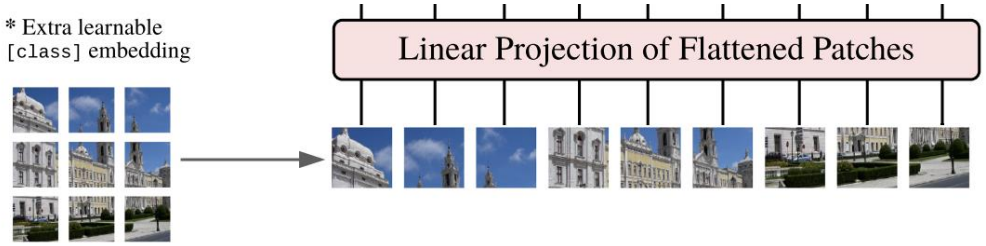






An Image is Worth 16x16 words

- Image *recognition*
- Image with **C** colors is split into sequence each **P** x **P** pixels wide
- Original paper: P=16
- An image of WxH pixels has **N**=W*H/P^2 patches
- x is a single patch of PxP pixels
- **E** is a matrix of size C*P*P x D
- D is the dimension of the embedding
- 1x768 * (768*D) -> 1xD
- X_{class} is a learned class embedding (the same for every class)



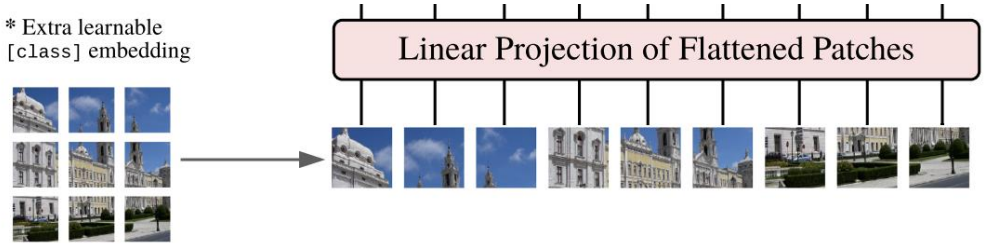
$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}};\ \mathbf{x}_p^1\mathbf{E};\ \mathbf{x}_p^2\mathbf{E};\ \cdots\ ;\ \mathbf{x}_p^N\mathbf{E}]$$

$$\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}$$

https://arxiv.org/pdf/2010.11929.pdf

An Image is Worth 16x16 words

- Example with P=16, C=3:
- Each patch is 16*16*3=768 values
- ViT-Large uses D=1024
- Embedding E is a 768x1024 matrix



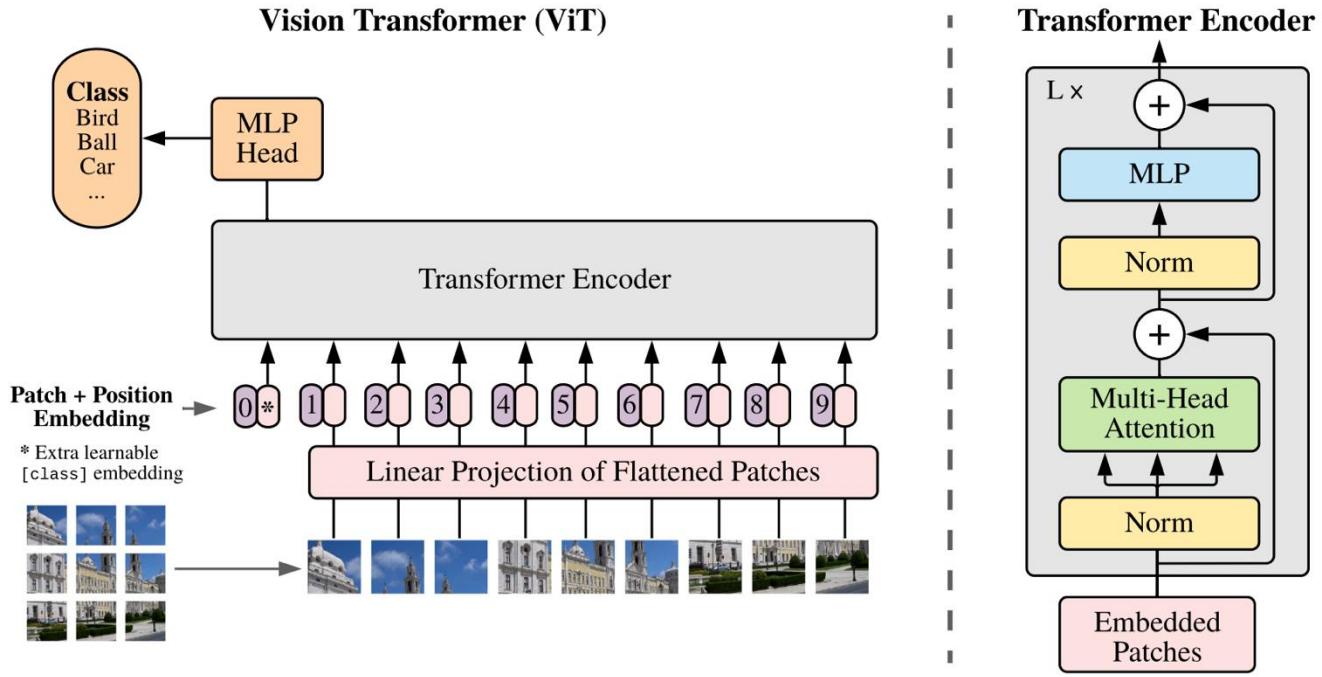
$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}};\ \mathbf{x}_p^1\mathbf{E};\ \mathbf{x}_p^2\mathbf{E};\ \cdots\ ;\ \mathbf{x}_p^N\mathbf{E}]$$

$$\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}$$

https://arxiv.org/pdf/2010.11929.pdf

Full architecture

- Add position encoding and class embedding
- Encoder just like for text
- Final MLP head for classification on x_{class} token



$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \cdots; \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \qquad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D},\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \qquad \ell = 1 \ldots L$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \qquad \ell = 1 \ldots L$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0)$$

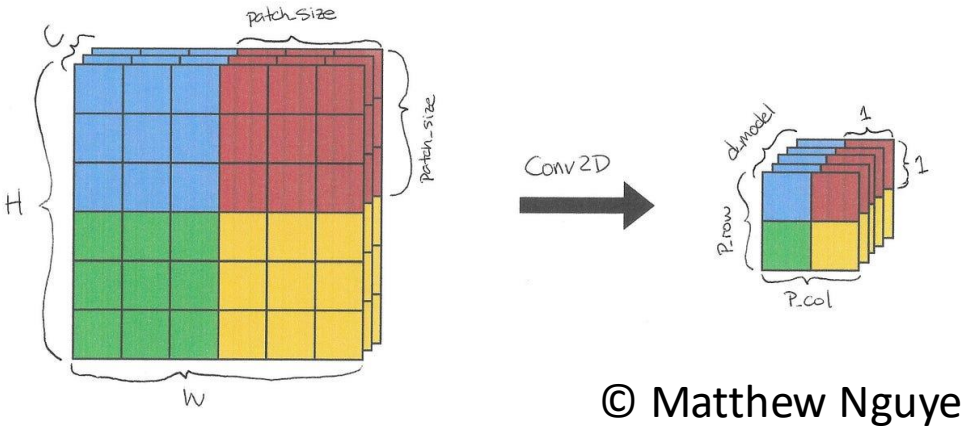
Trick: combine patching and linear projection

- 2D Convolution
- Kernel size is the same as patch size
- Stride length is the same as patch size
- Convolution reduces to matrix multiplication

```python
class PatchEmbedding(nn.Module):
  def __init__(self, d_model, img_size, patch_size, n_channels):
    super().__init__()

    self.d_model = d_model # Dimensionality of Model
    self.img_size = img_size # Image Size
    self.patch_size = patch_size # Patch Size
    self.n_channels = n_channels # Number of Channels

    self.linear_project = nn.Conv2d(self.n_channels, self.d_model, kernel_size=self.patch_size, stride=self.patch_size)

  # B: Batch Size
  # C: Image Channels
  # H: Image Height
  # W: Image Width
  # P_col: Patch Column
  # P_row: Patch Row
  def forward(self, x):
    x = self.linear_project(x) # (B, C, H, W) -> (B, d_model, P_col, P_row)

    x = x.flatten(2) # (B, d_model, P_col, P_row) -> (B, d_model, P)

    x = x.transpose(-2, -1) # (B, d_model, P) -> (B, P, d_model)

    return x
```

2D Convolution as Matrix multiplication

- W,H : width and height of image
- C : number of channels
- patch_size: kernel width/height
- d_model: desired model dimension
- Desired operation: x @ E
  - 1x patch_size^2 @ patch_size^2 x d_model
- The convolution accomplishes this by using d_model kernels of patch_size^2

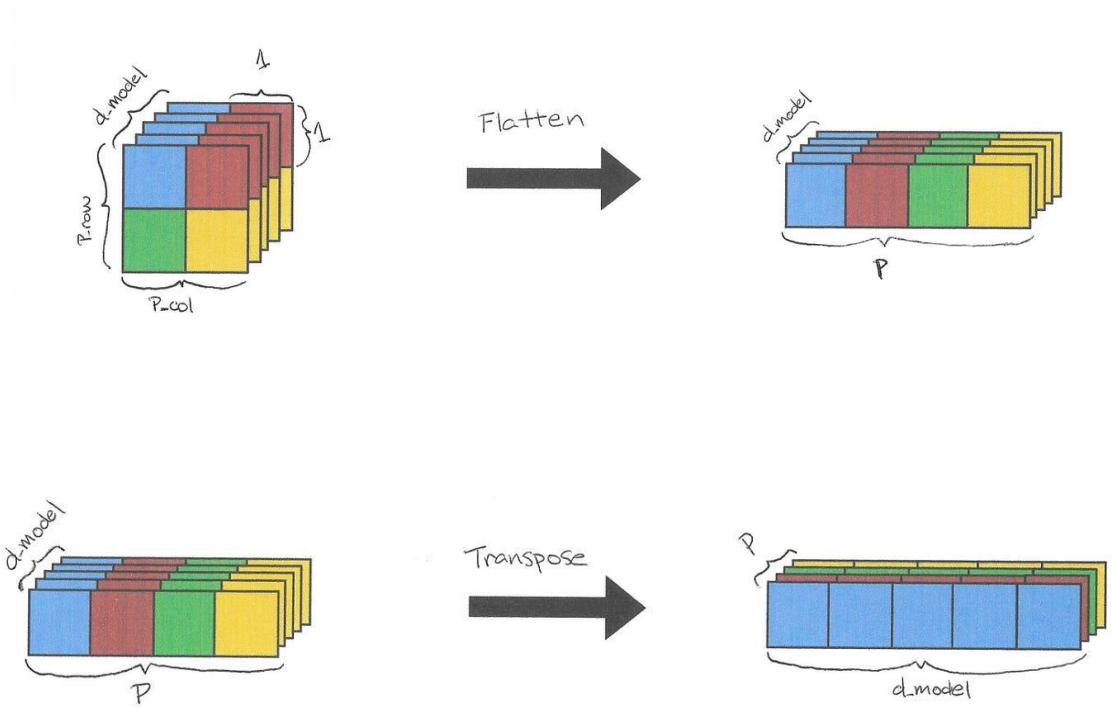


2D Convolution as Matrix Multiplication

- To obtain the transformed patches we need to

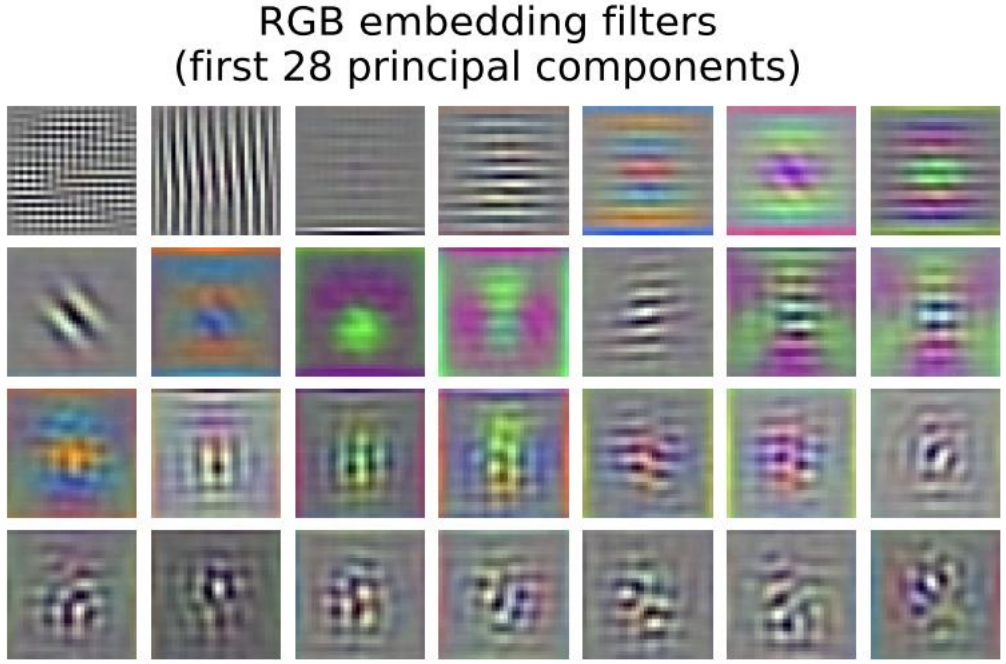
1. Flatten
2. Transpose

```
x = self.linear_project(x) # (B, C, H, W) -> (B, d_model, P_col, P_row)
x = x.flatten(2) # (B, d_model, P_col, P_row) -> (B, d_model, P)
x = x.transpose(-2, -1) # (B, d_model, P) -> (B, P, d_model)
return x
```



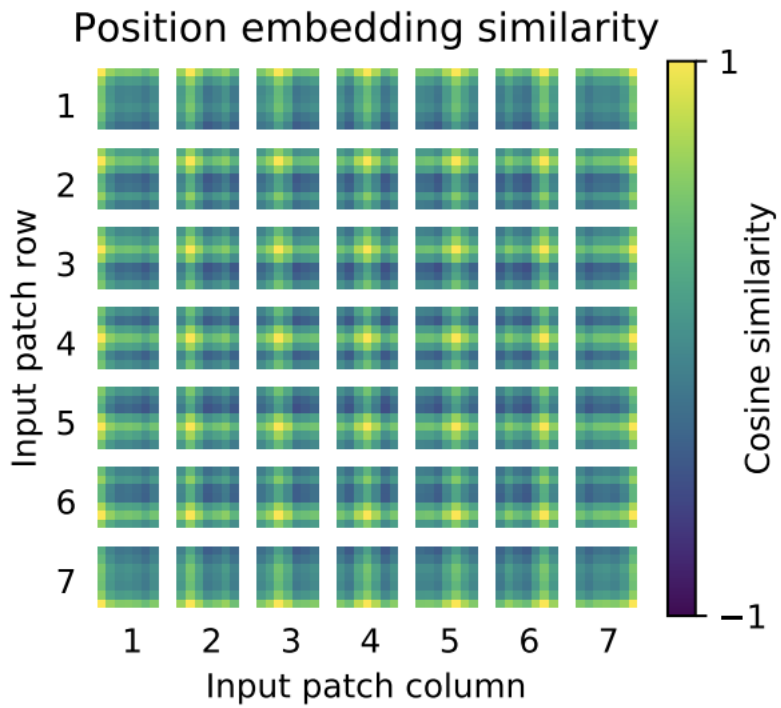
What does the E matrix do?

- Each column of E is one filter
- Principal Component Analysis of the columns of E
- Example:
  - Image patch is 16x16x3 = 1x768
  - Dimension is 1024
  - E is 768x1024
  - Output is 1x1024
- “The components resemble plausible basis functions for a low-dimensional representation of the fine structure within each patch.”



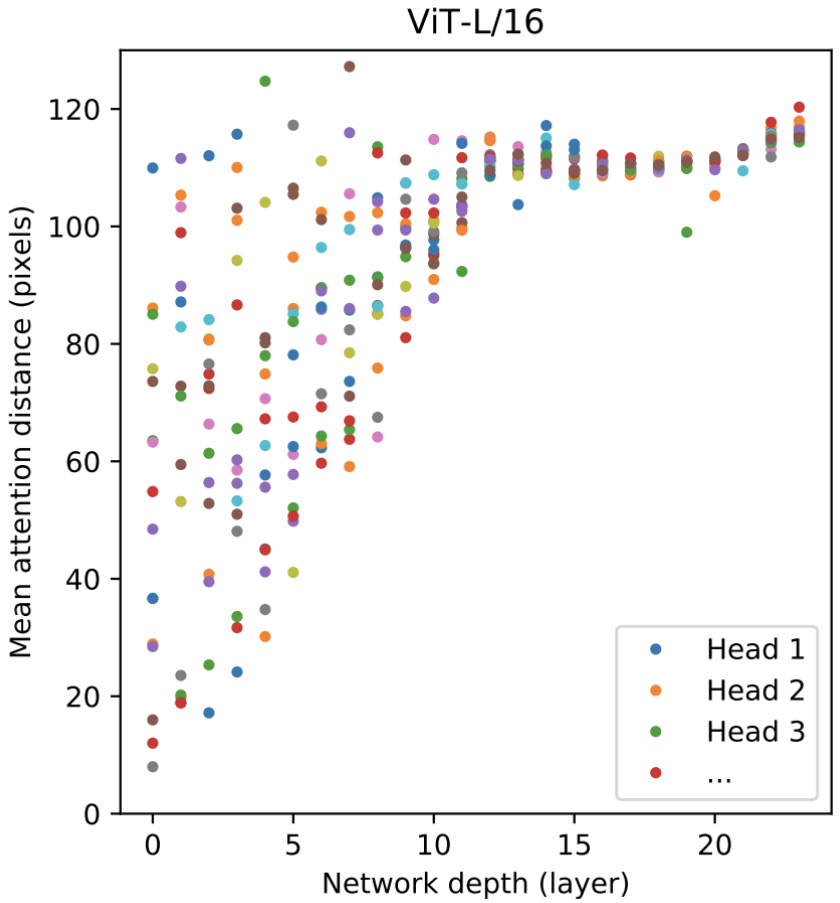
How does positional encoding turn out?

- Positional encoding is also learned
- Closer patches tend to have more similar position embeddings
- Patches in the same row/column have similar embeddings



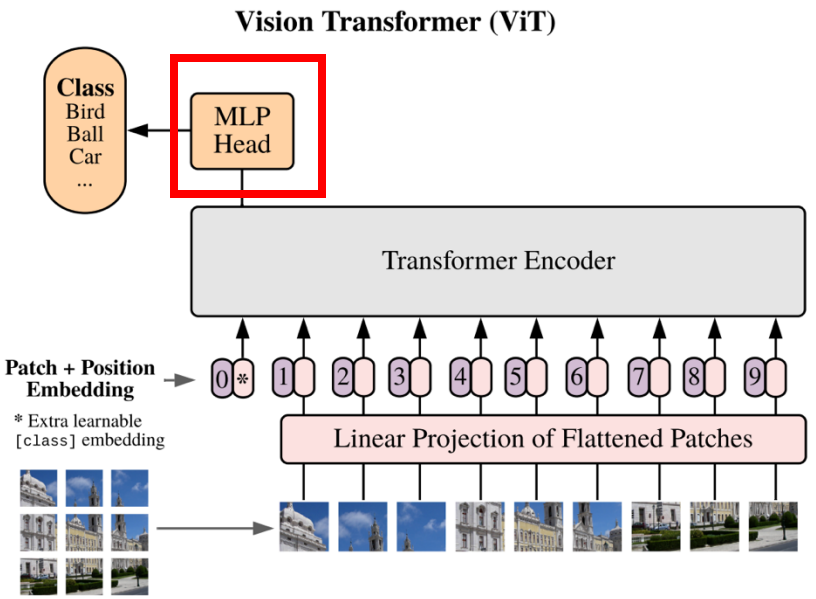
How far does self-attention look?

- Initially, attention looks at both far and near (in pixel distance) information
- The deeper the layer, the more emphasis is on relations across the image
- Analogous to CNNs, except for CNNs not looking far in lower layers (inductive bias?)



Finetuning

- Remove pre-trained classification head at the top
- Add D x K fresh prediction head
  - D: Transformer dimension
  - K: Number of classes to finetune
- What is kept are
  - Transformer embedding E
  - Weights for key, query and value
  - All MLPs below the top
- Finetuning often at higher resolution
  - Same patch size, longer sequence
  - Interpolate position embeddings to span the larger area



Evaluation

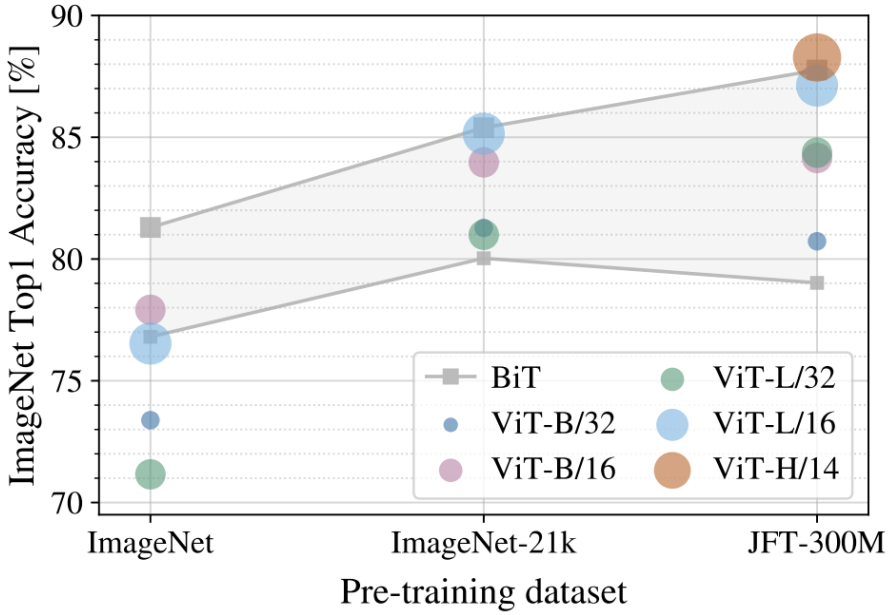
- Pretrained on
  - ImageNet: 1k classes and 1.3M images
  - ImageNet-21k: 21k classes and 14M images
  - JFT: 18k classes and **303M** images
- Note
  - JFT dataset is proprietary
  - TPU Hardware is not for sale
- ViT outperforms all other models at fraction of computational time

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

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

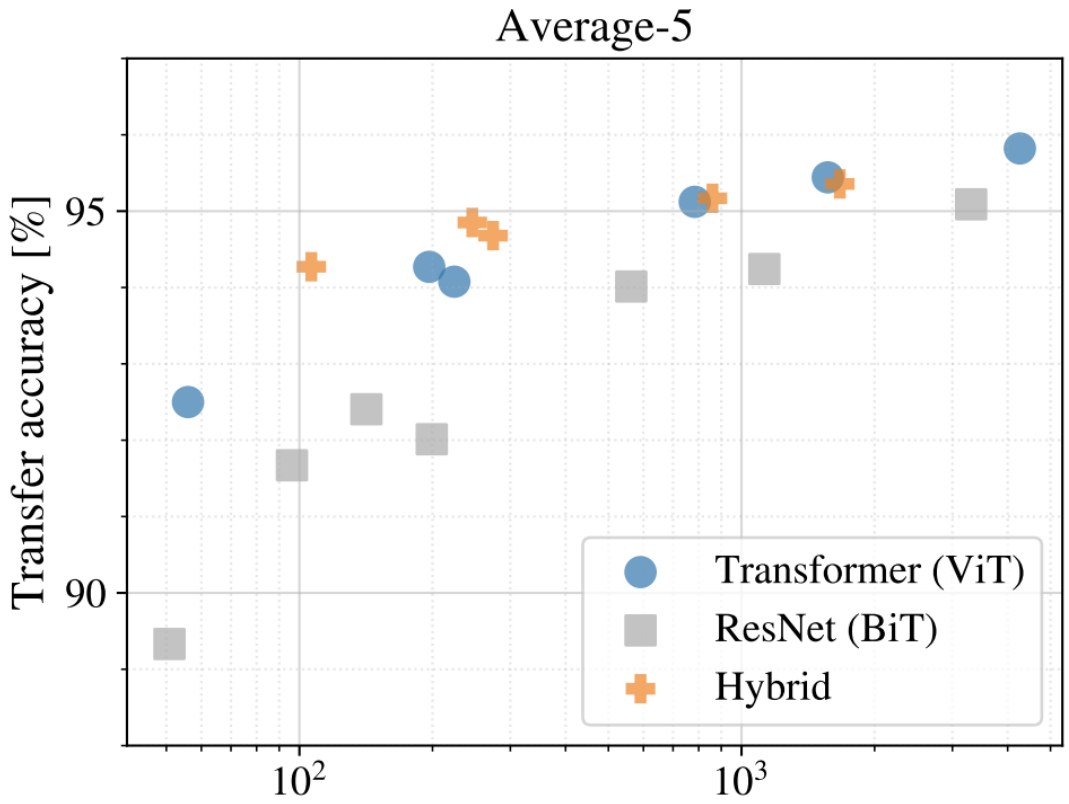
Notes

- ViT underperforms ResNet on Midsize datasets, but outperforms on really large datasets 14M-300M images
- Inductive bias of CNN helps with small datasets, but transformer learns the relevant structures with enough data
- Transformer-only wasn't new. Key insight was to use the efficient NLP implementations directly.
- Prior papers with the same idea go back 2+ years, Google DeepMind picked up on it and did it at scale



Combining vision transformers and convolution?

- Hybrid networks perform better than ViT or BiT *when trained less*
- The advantage vanishes as training goes on



Practice: Inspect a Basic ViT model

- Goal: bring its accuracy up
- Exploratory tasks:
  - where is the E matrix?
  - visualize positional embeddings
  - make positional embeddings trainable
  - What happens when removing the cls token.

```
correct = 0
total = 0

with torch.no_grad():
  for data in test_loader:
    images, labels = data
    images, labels = images.to(device), labels.to(device)

    outputs = transformer(images)

    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()
  print(f'\nModel Accuracy: {100 * correct // total} %')
```

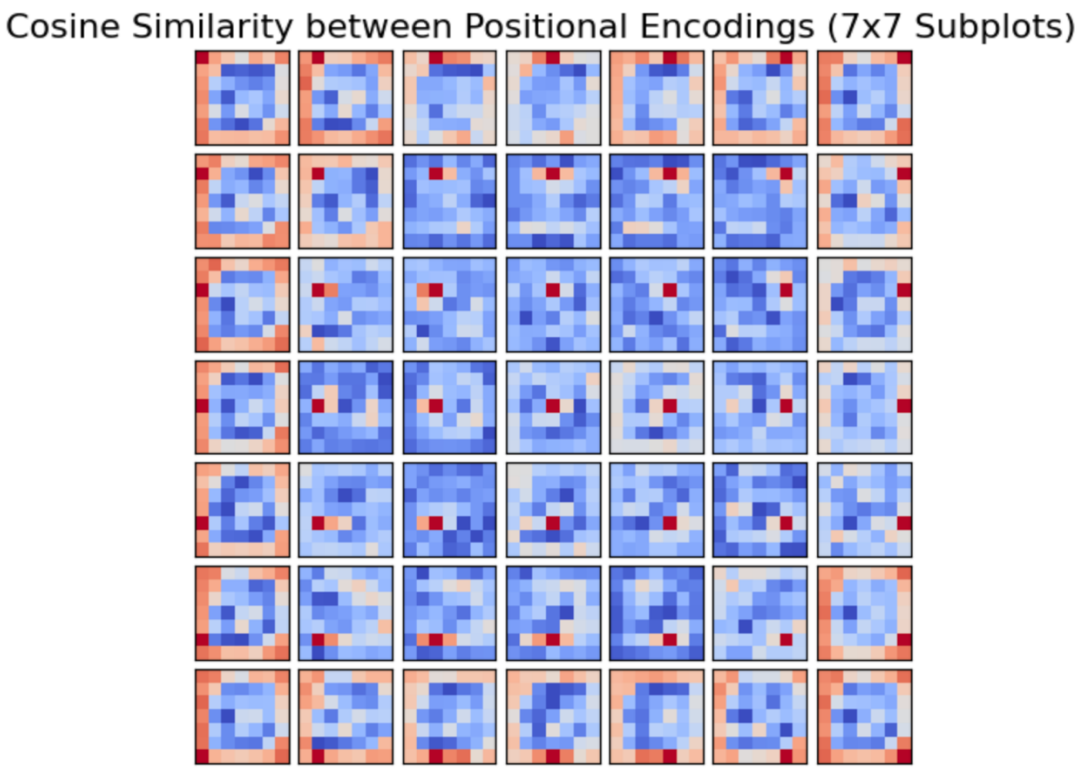
```
Model Accuracy: 98 %
```

Solution

- Parameters
  - d_model = 64 # Increased model dimension for better feature learning
  - n_classes = 10 # No change, 10 classes for MNIST digits
  - img_size = (28,28) # Matching the original MNIST size
  - patch_size = (4,4) # Reasonable for 28x28 input
  - n_channels = 1 # MNIST is grayscale
  - n_heads = 4 # Better division of attention with 64-dimensional model
  - n_layers = 2 # Two layers should suffice for MNIST
  - batch_size = 256 # Larger batch size for faster training
  - epochs = 10 # More epochs to allow convergence
  - alpha = 0.001 # Smaller learning rate for better training stability
- Add dropout to self-attention module (but not too many)
- Use weight decay (penalizes large weights)
- More training

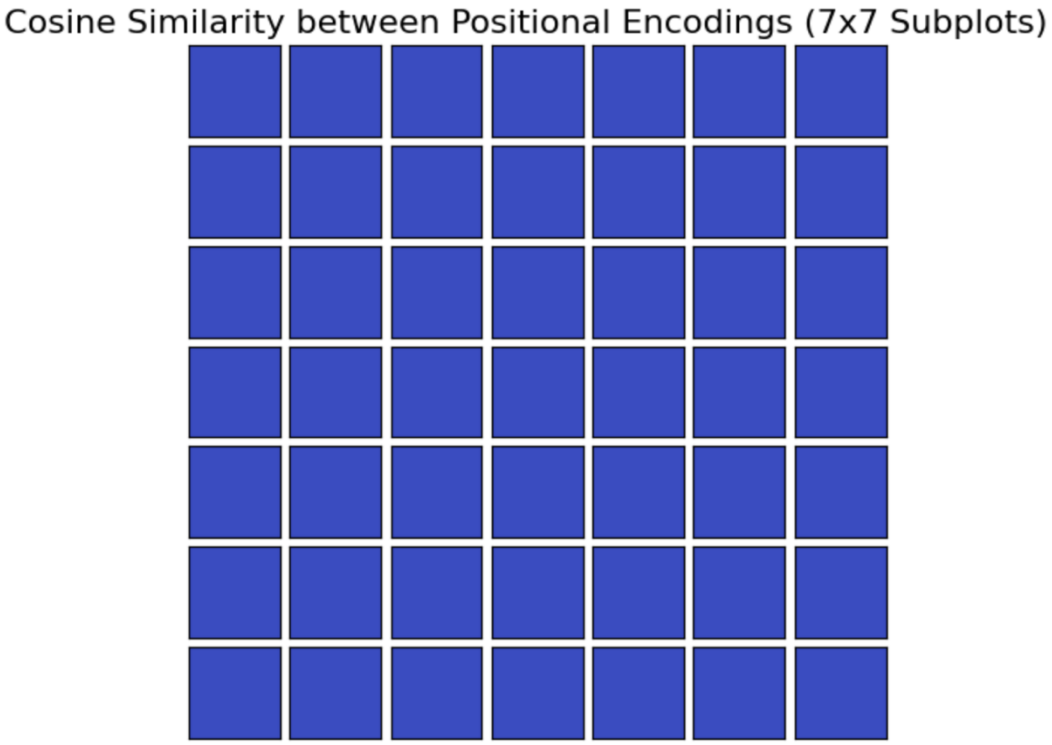
Learning positional encodings

After 1 epoch

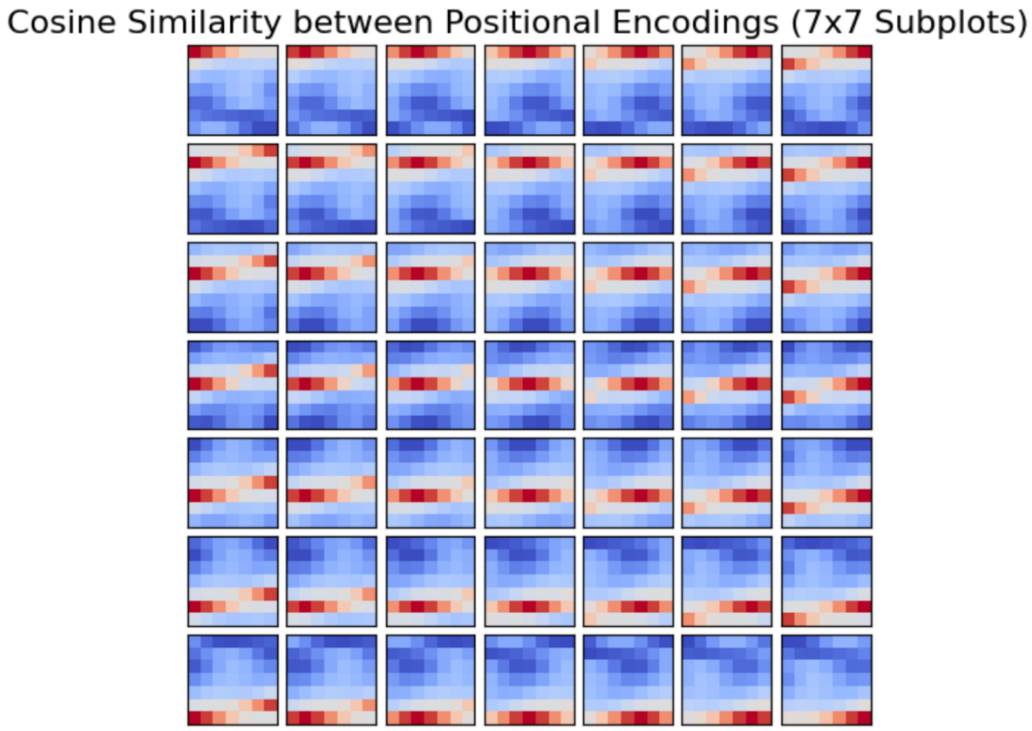


After 50 epochs

Positional Encodings do Matter



68% after 10 epochs



95% after 10 epochs

Next week

- Contrastive Image-Language Pretraining (CLIP)
- Combining images and text