Lecture 4: Deep Learning

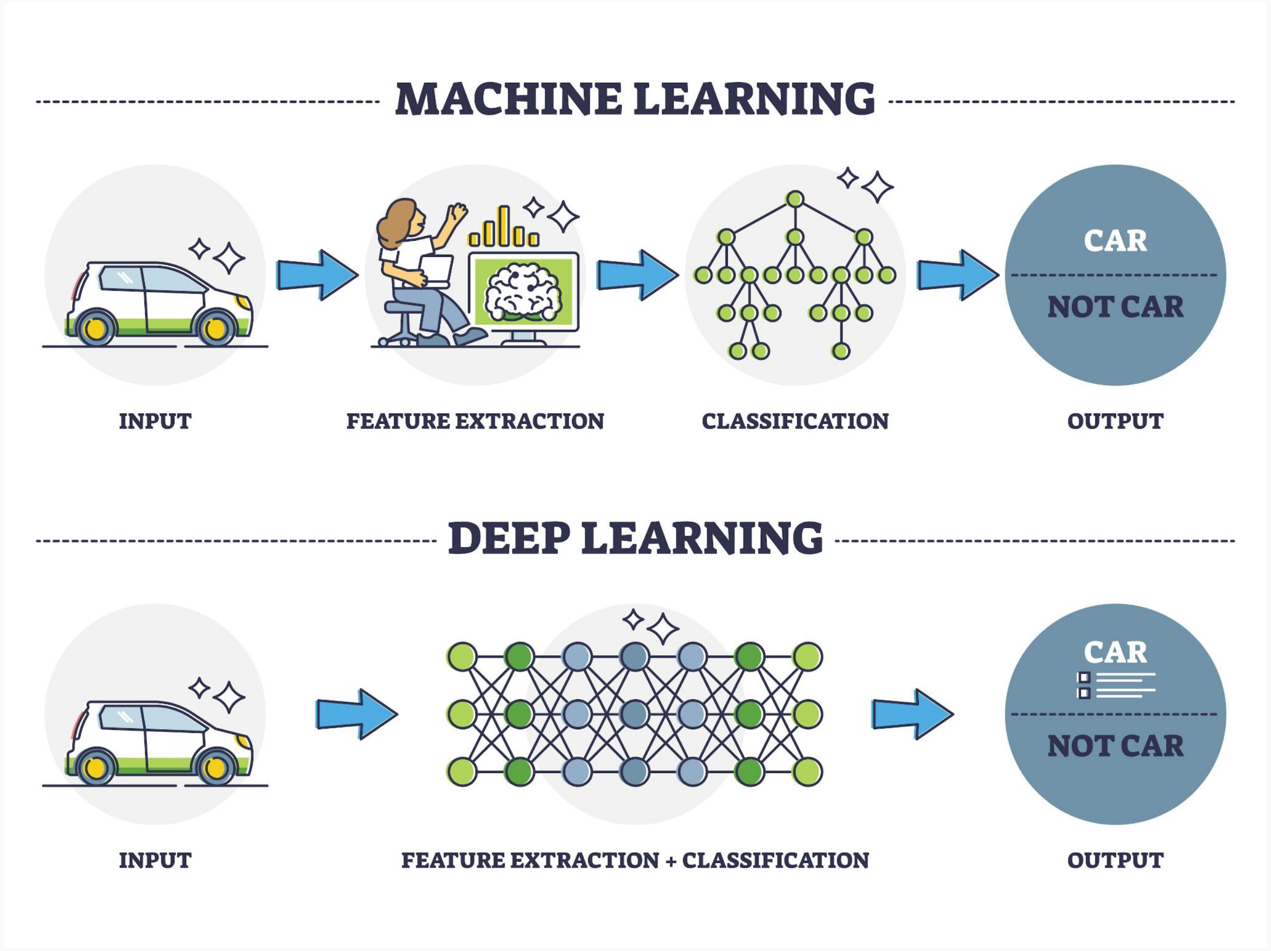
https://github.com/kaopanboonyuen/SC310005_ArtificialIntelligence_2025s1

Teerapong Panboonyuen
https://kaopanboonyuen.github.io

Introduction to Deep Learning:

- Deep Learning is a subset of Machine Learning based on artificial neural networks.
- It excels at learning from large amounts of data, especially unstructured data like images, audio, and text.
- In this lecture, we'll apply deep learning to image classification: Recognizing Thai Prime Ministers' faces.

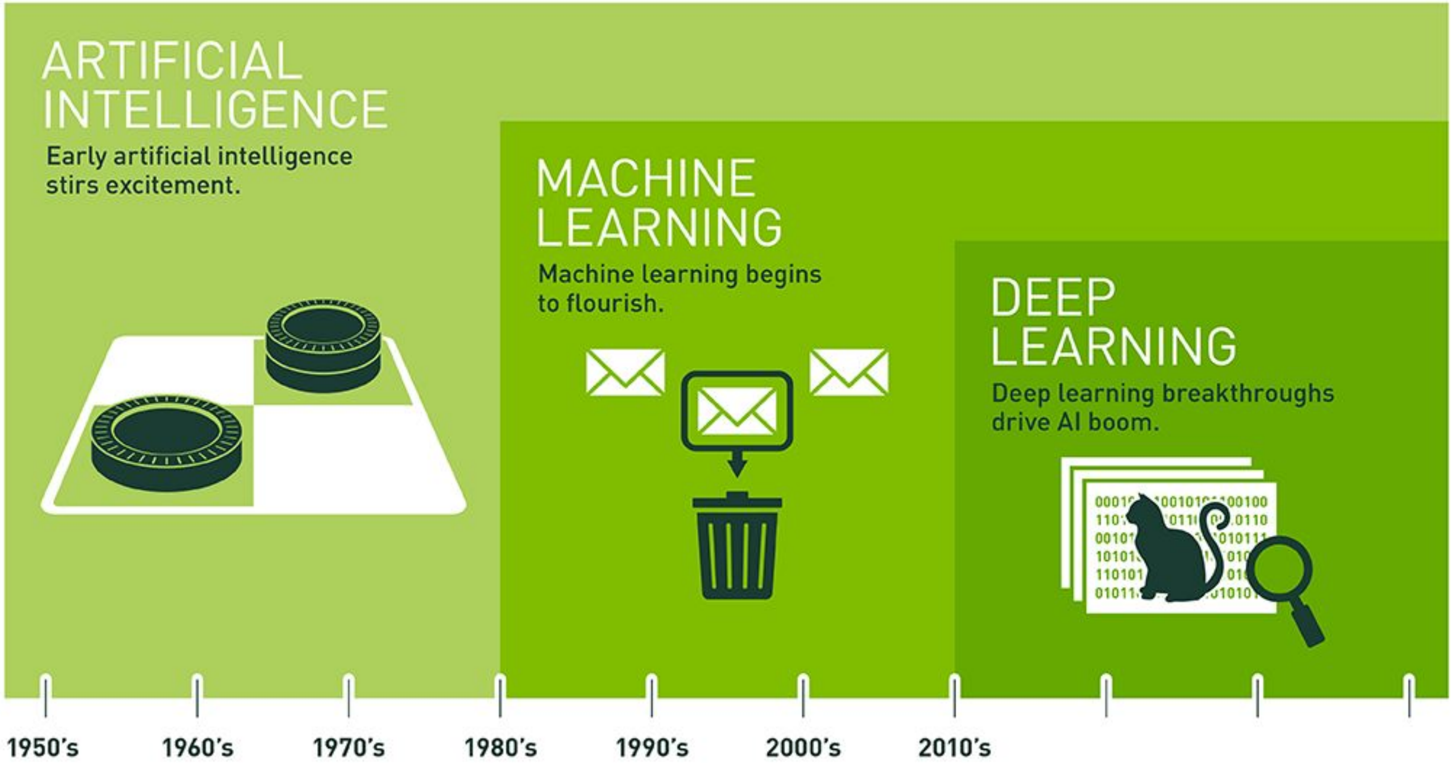
https://www.alltius.ai/glossary/what-is-deep-learning



Deep Learning Foundations

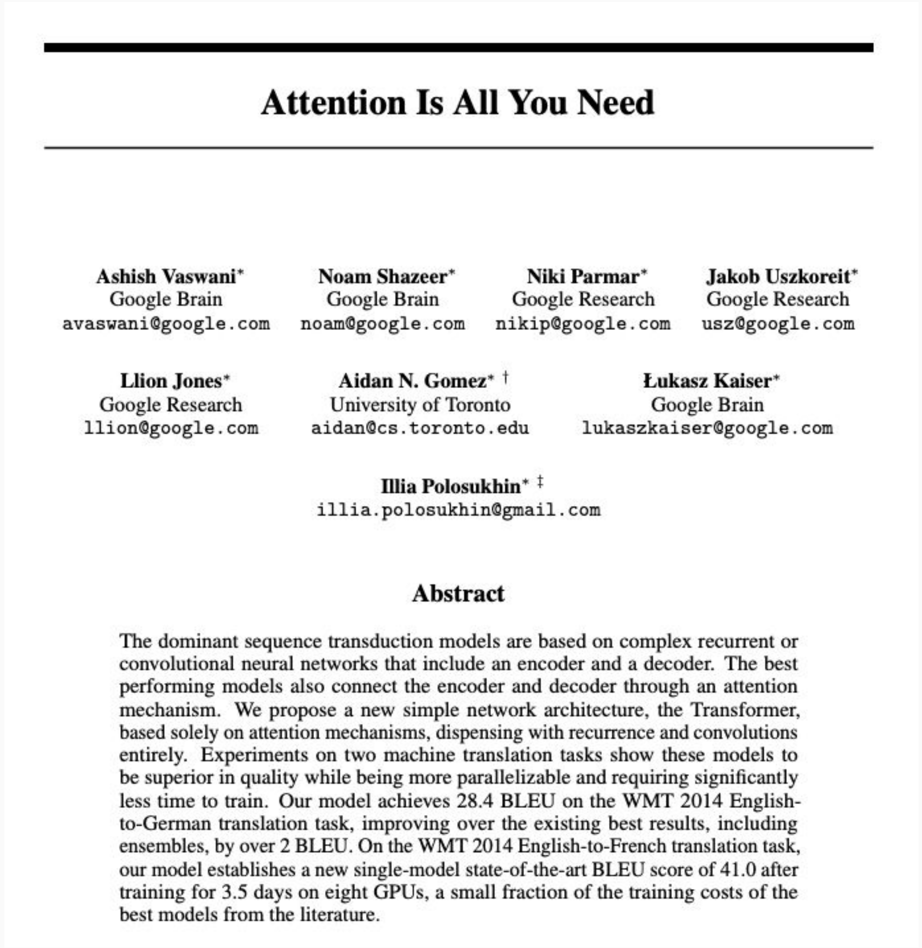
- **Neuron**: Basic computation unit.
- **Layer**: Stack of neurons.
- **Model**: A combination of layers.
- **Forward Pass**: Data flows through the model.
- **Loss Function**: Measures the error.
- **Backpropagation**: Updates weights using gradients.

https://developer.nvidia.com/deep-learning


ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf



Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[*] [†]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*] [‡]
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

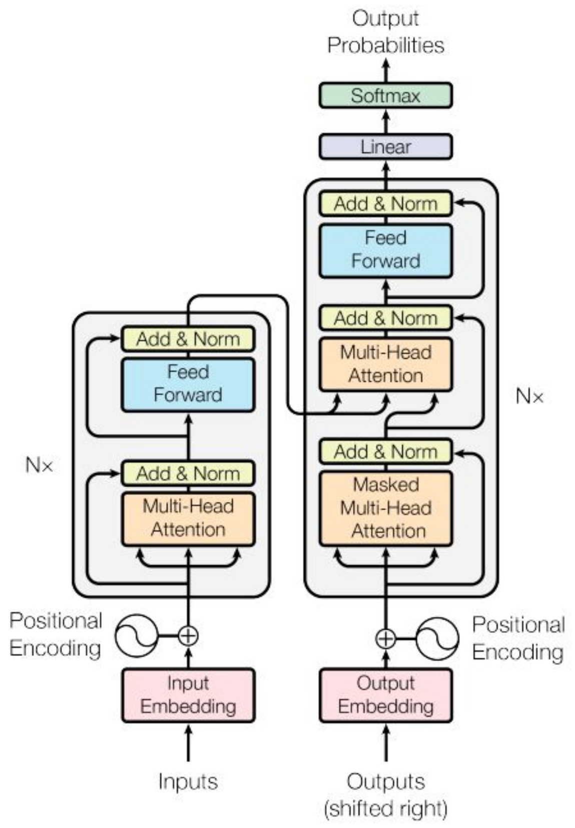
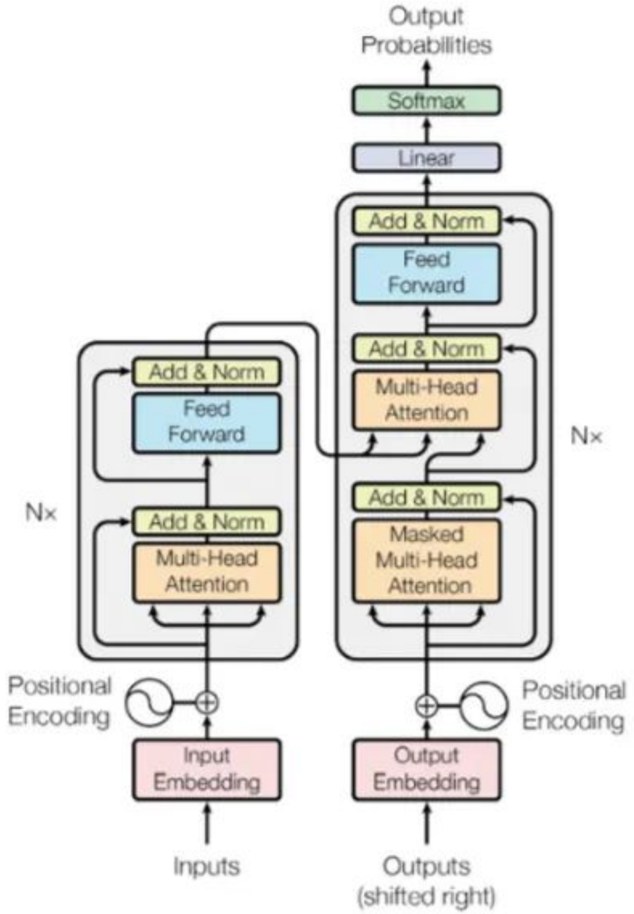


Figure 1: The Transformer - model architecture.

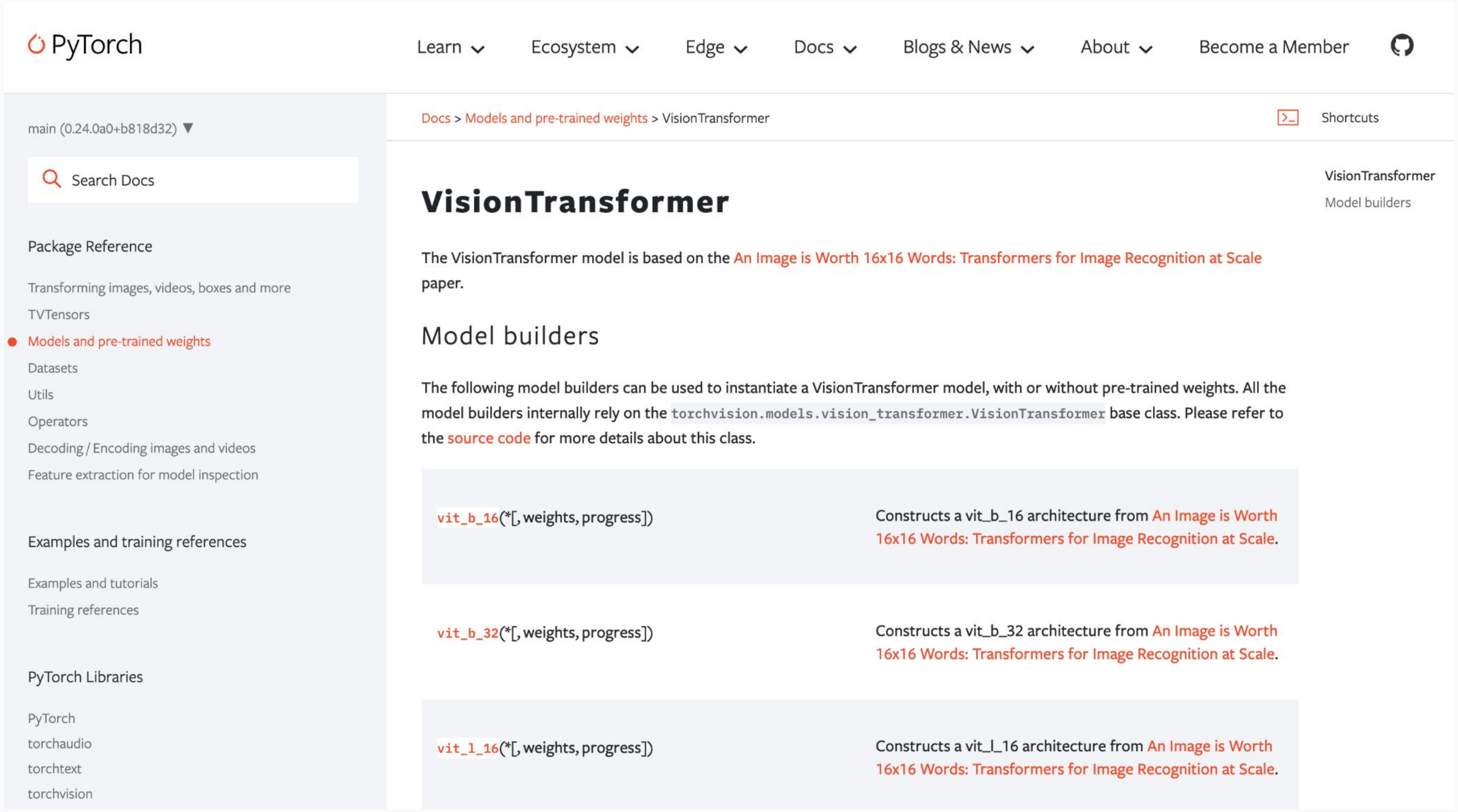
https://medium.com/@mervebdurna/how-transformers-work-a-beginner-friendly-guide-to-attention-is-all-you-need-683d660c6442

Transformer

Attention Is All You Need



https://docs.pytorch.org/vision/main/models/vision_transformer.html


PyTorch
Learn
Ecosystem
Edge
Docs
Blogs & News
About
Become a Member
main (0.24.0a0+b818d32) ▼
Search Docs
Package Reference
Transforming images, videos, boxes and more
TVTensors
Models and pre-trained weights
Datasets
Utils
Operators
Decoding / Encoding images and videos
Feature extraction for model inspection
Examples and training references
Examples and tutorials
Training references
PyTorch Libraries
PyTorch
torchaudio
torchtext
torchvision
Docs > Models and pre-trained weights > VisionTransformer
Shortcuts
VisionTransformer
Model builders

VisionTransformer

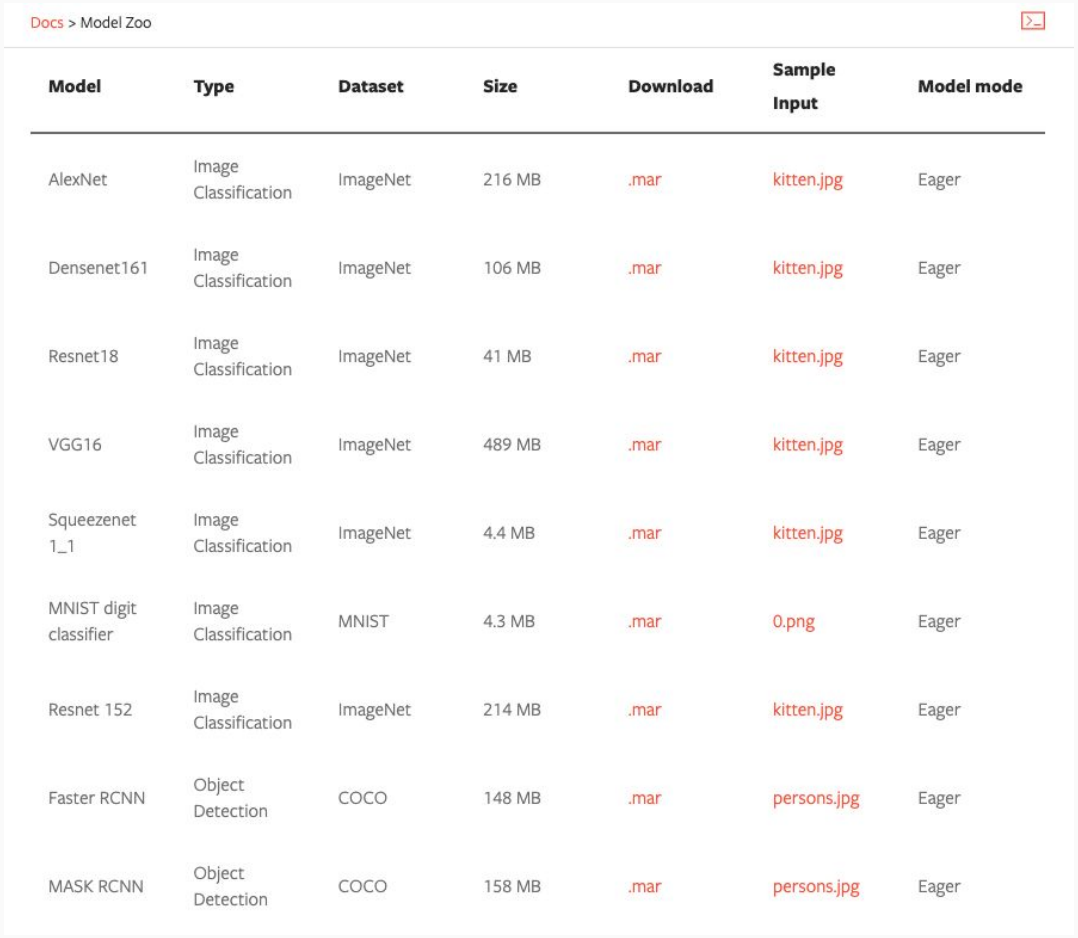
The VisionTransformer model is based on the An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale paper.

Model builders

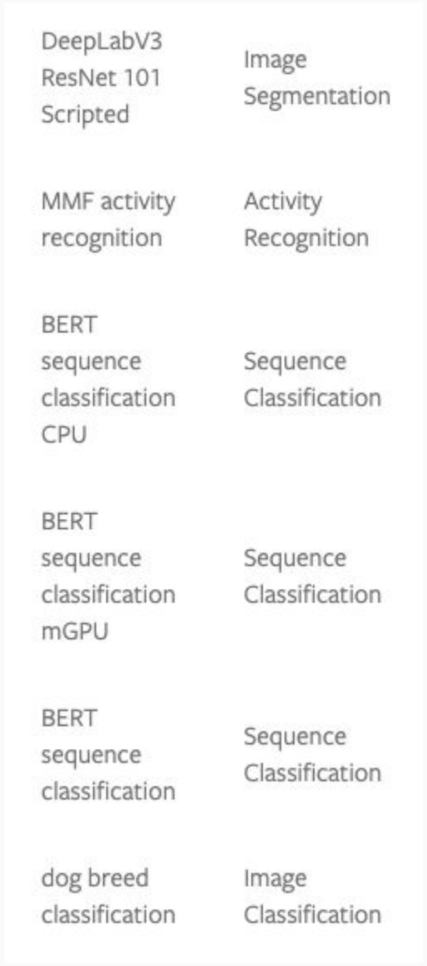
The following model builders can be used to instantiate a VisionTransformer model, with or without pre-trained weights. All the model builders internally rely on the `torchvision.models.vision_transformer.VisionTransformer` base class. Please refer to the source code for more details about this class.

`vit_b_16`(*[, weights, progress])	Constructs a vit_b_16 architecture from An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.
`vit_b_32`(*[, weights, progress])	Constructs a vit_b_32 architecture from An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.
`vit_l_16`(*[, weights, progress])	Constructs a vit_l_16 architecture from An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.

https://docs.pytorch.org/serve/model_zoo.html

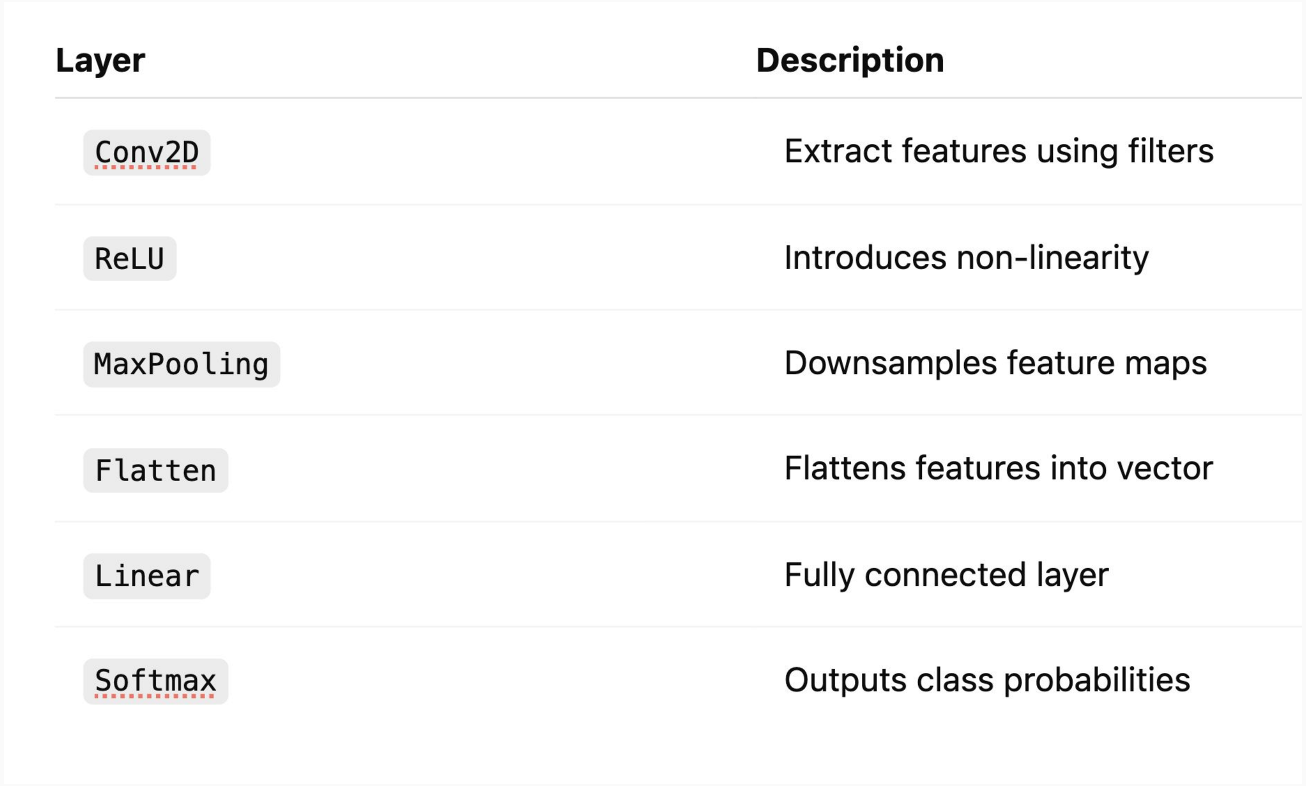

Docs > Model Zoo

Model	Type	Dataset	Size	Download	Sample Input	Model mode
AlexNet	Image Classification	ImageNet	216 MB	.mar	kitten.jpg	Eager
Densenet161	Image Classification	ImageNet	106 MB	.mar	kitten.jpg	Eager
Resnet18	Image Classification	ImageNet	41 MB	.mar	kitten.jpg	Eager
VGG16	Image Classification	ImageNet	489 MB	.mar	kitten.jpg	Eager
Squeezenet 1_1	Image Classification	ImageNet	4.4 MB	.mar	kitten.jpg	Eager
MNIST digit classifier	Image Classification	MNIST	4.3 MB	.mar	0.png	Eager
Resnet 152	Image Classification	ImageNet	214 MB	.mar	kitten.jpg	Eager
Faster RCNN	Object Detection	COCO	148 MB	.mar	persons.jpg	Eager
MASK RCNN	Object Detection	COCO	158 MB	.mar	persons.jpg	Eager



DeepLabV3 ResNet 101 Scripted	Image Segmentation
MMF activity recognition	Activity Recognition
BERT sequence classification CPU	Sequence Classification
BERT sequence classification mGPU	Sequence Classification
BERT sequence classification	Sequence Classification
dog breed classification	Image Classification

Core Layers in Deep Learning



Layer	Description
`Conv2D`	Extract features using filters
`ReLU`	Introduces non-linearity
`MaxPooling`	Downsamples feature maps
`Flatten`	Flattens features into vector
`Linear`	Fully connected layer
`Softmax`	Outputs class probabilities

What You’ll Build Today

- Face classification of Thai Prime Ministers
- Step-by-step tutorial using PyTorch
- Compare multiple architectures:
    - CustomCNN1 (Basic CNN)
    - CustomCNN2 (CNN + Self-Attention)
    - ResNet50, ResNet101, DenseNet121
    - Vision Transformer (ViT)

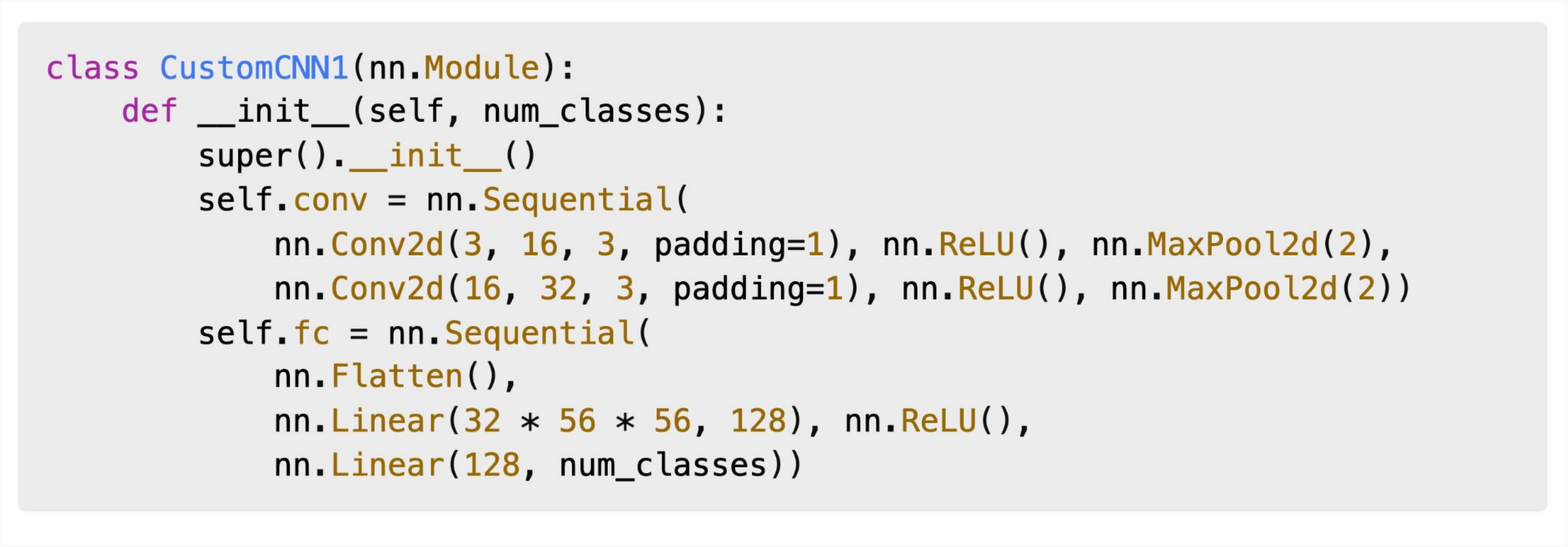
Dataset Overview

- 10 classes (one per Prime Minister)
- Face images of each
- Preprocessed into 224x224 pixels
- Train/Test split: 80/20

Image Transformations

- Resize (224x224)
- Convert to Tensor
- Normalize with mean 0.5, std 0.5

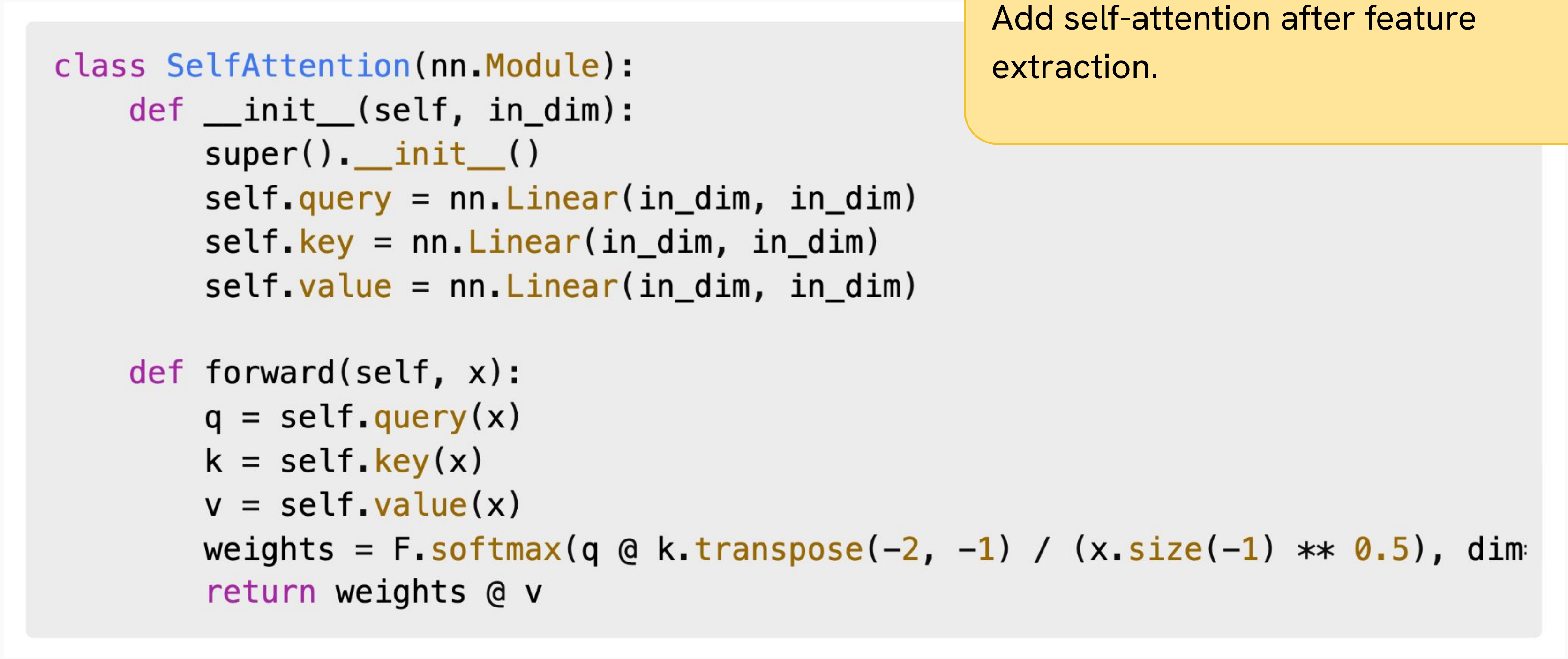
Custom Model 1 - CNN (Like LeNet-5)



```python
class CustomCNN1(nn.Module):
    def __init__(self, num_classes):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(3, 16, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(16, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2))
        self.fc = nn.Sequential(
            nn.Flatten(),
            nn.Linear(32 * 56 * 56, 128), nn.ReLU(),
            nn.Linear(128, num_classes))
```

Custom Model 2 - CNN + Self Attention

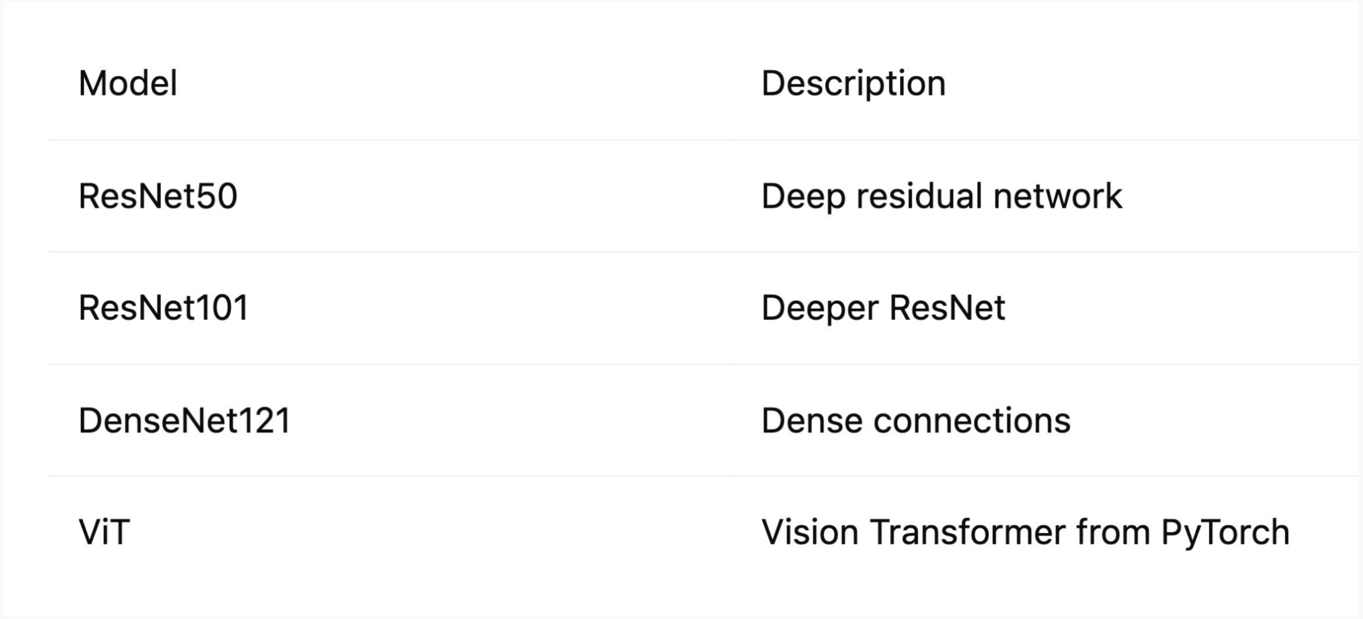
Add self-attention after feature extraction.



```
class SelfAttention(nn.Module):
    def __init__(self, in_dim):
        super().__init__()
        self.query = nn.Linear(in_dim, in_dim)
        self.key = nn.Linear(in_dim, in_dim)
        self.value = nn.Linear(in_dim, in_dim)

    def forward(self, x):
        q = self.query(x)
        k = self.key(x)
        v = self.value(x)
        weights = F.softmax(q @ k.transpose(-2, -1) / (x.size(-1) ** 0.5), dim
        return weights @ v
```

Pretrained Models

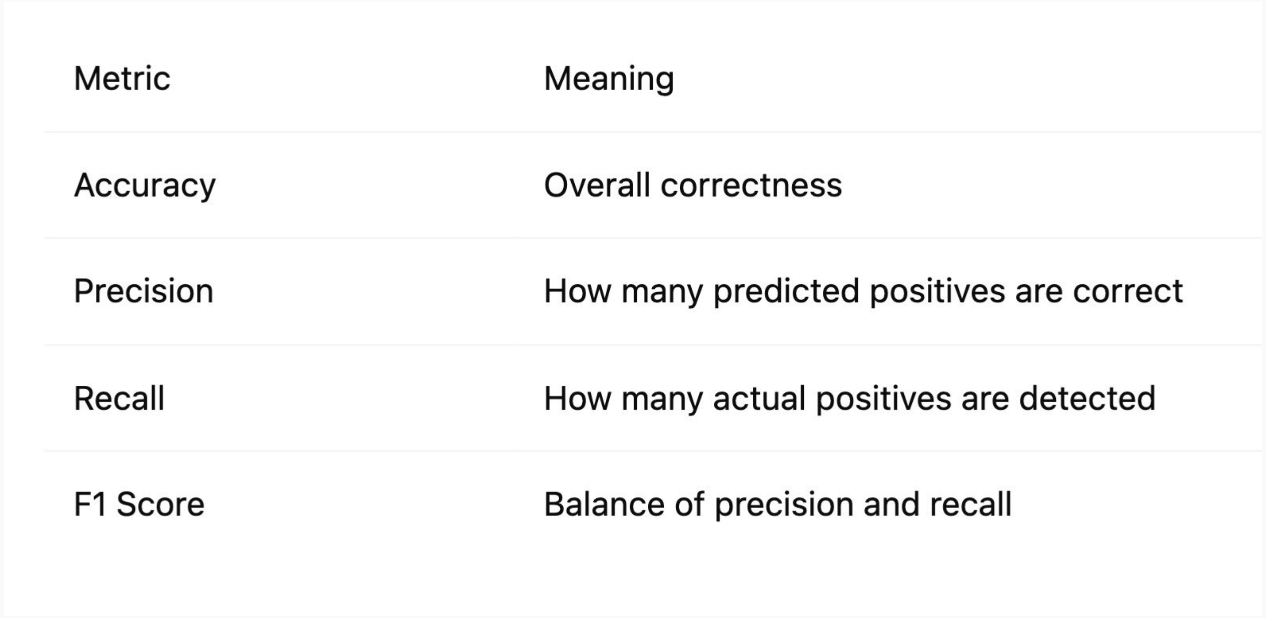


Model	Description
ResNet50	Deep residual network
ResNet101	Deeper ResNet
DenseNet121	Dense connections
ViT	Vision Transformer from PyTorch

Training Loop (Shared by All Models)

- Forward pass
- Compute loss (CrossEntropy)
- Backpropagation
- Optimizer step (Adam)
- Repeat for multiple epochs

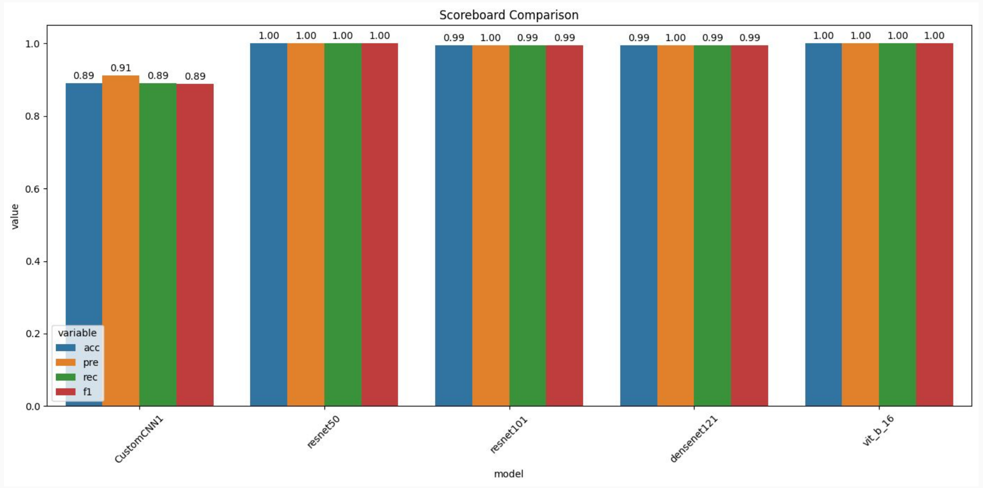
Evaluation Metrics



Metric	Meaning
Accuracy	Overall correctness
Precision	How many predicted positives are correct
Recall	How many actual positives are detected
F1 Score	Balance of precision and recall

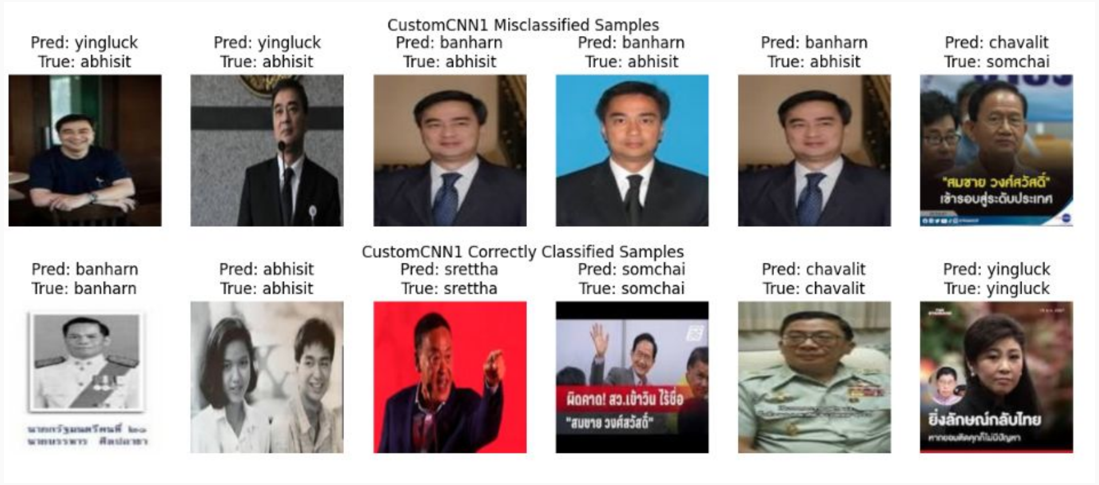
Scoreboard

- Compare all models on the same dataset
- Plot bar chart of Acc, Prec, Recall, F1
- Label each bar with the value



Error Analysis

- Visualize correctly and incorrectly classified images
- Helps understand model weaknesses
- Visual feedback = great teaching tool


CustomCNN1 Misclassified Samples
Pred: yingluck True: abhisit
Pred: yingluck True: abhisit
Pred: banharn True: abhisit
Pred: banharn True: abhisit
Pred: banharn True: abhisit
Pred: chavalit True: somchai
CustomCNN1 Correctly Classified Samples
Pred: banharn True: banharn
Pred: abhisit True: abhisit
Pred: srettha True: srettha
Pred: somchai True: somchai
Pred: chavalit True: chavalit
Pred: yingluck True: yingluck

Tricks to Boost Accuracy for Mini-Projects

📈 Image Augmentation

- Add variety to training images by flipping, rotating, zooming.
- Example:

```
transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ColorJitter(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])
```

Tips:

- 🧠 **Transfer Learning**
  - Start with a pretrained model and fine-tune it on your dataset.
- 🧹 **Clean Your Data**
  - Make sure labels are correct. Visualize samples often.
- 📦 **Early Stopping & Checkpoints**
  - Stop training when validation loss stops improving.
- 🔢 **Hyperparameter Tuning**
  - Try different learning rates, batch sizes, optimizers.

Tips:

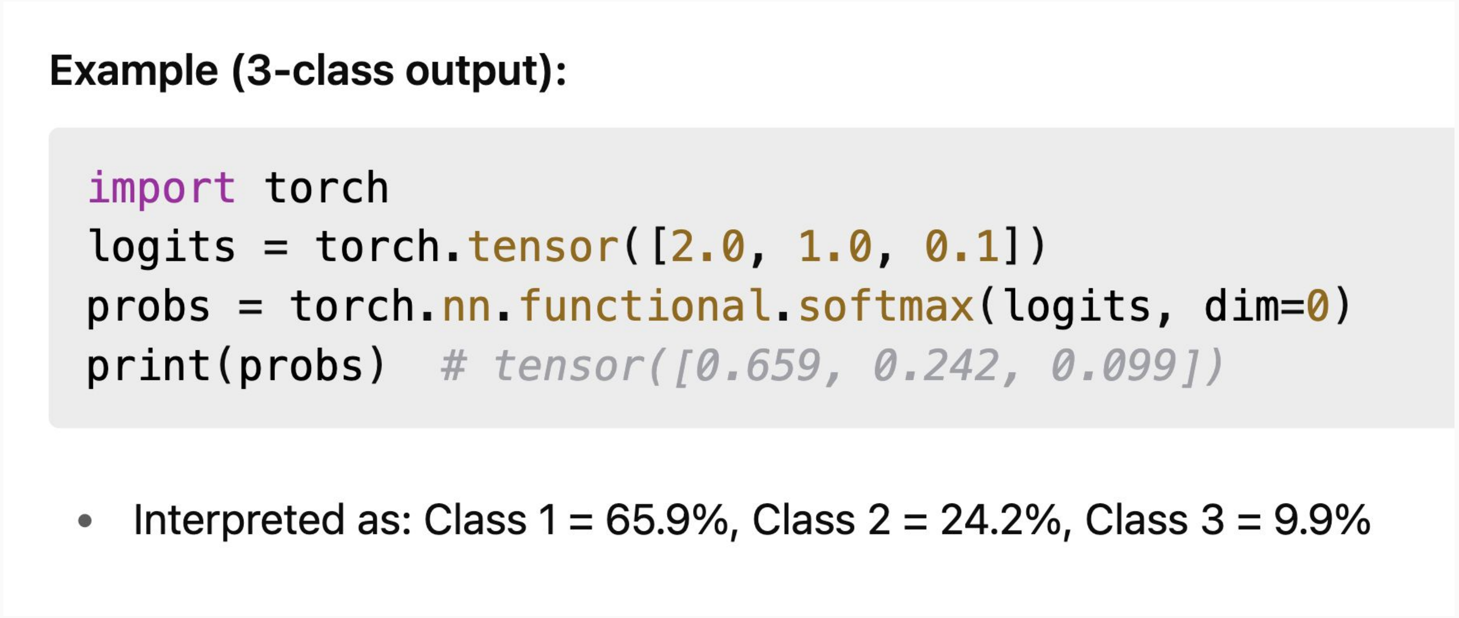
- 👁️‍🗨️ **Grad-CAM / Feature Maps**
  - Visualize where the model is focusing. Debug like a pro.
- 🧪 **Stratified Splits**
  - Always maintain class balance in train/test splits.
- 💬 **Ask the Model: Confidence Scores**
  - Use `softmax` to find out which predictions the model is unsure about.
- 🔄 **Data Balancing**
  - If classes are imbalanced, consider weighted loss or over/under sampling.

Save and Load Models

```
# Save
torch.save(model.state_dict(), 'best_model.pth')
# Load
model.load_state_dict(torch.load('best_model.pth'))
```

Understanding Softmax (with Example)

Softmax turns raw scores (logits) into probabilities.

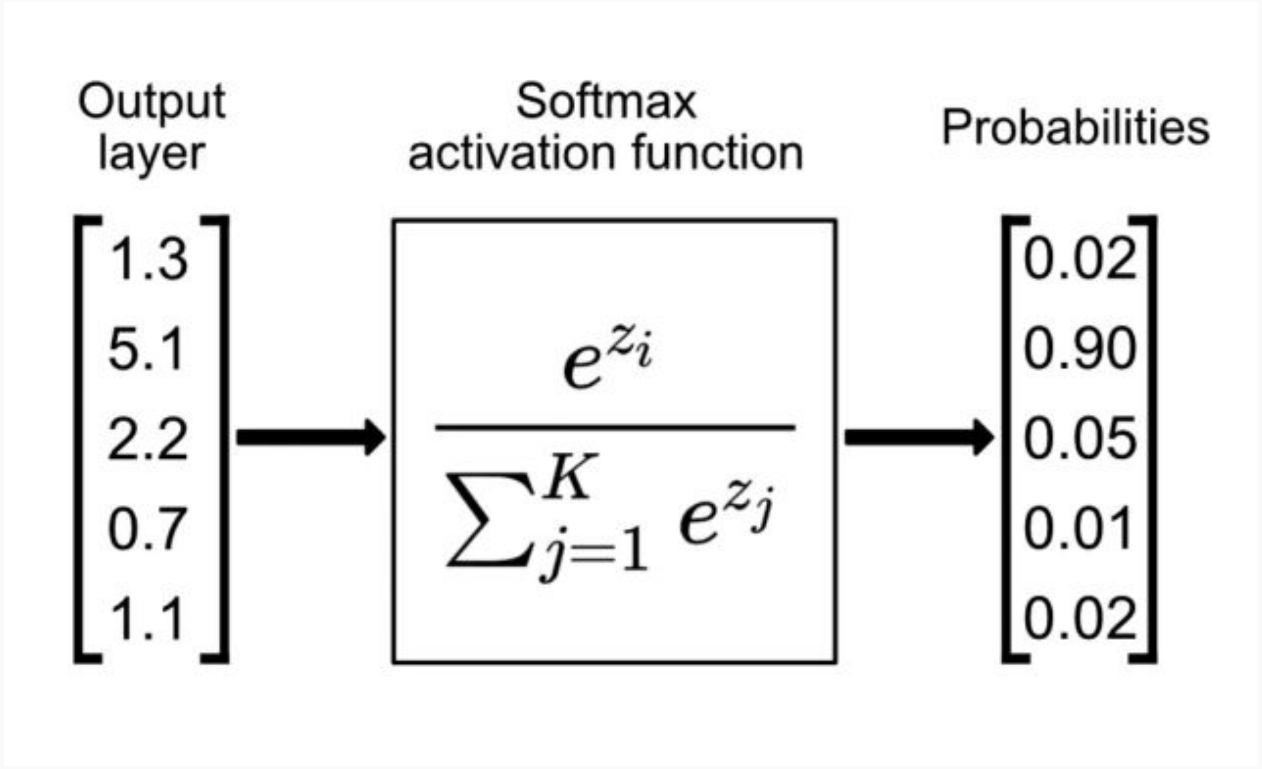


Example (3-class output):

```python
import torch
logits = torch.tensor([2.0, 1.0, 0.1])
probs = torch.nn.functional.softmax(logits, dim=0)
print(probs)  # tensor([0.659, 0.242, 0.099])
```

- Interpreted as: Class 1 = 65.9%, Class 2 = 24.2%, Class 3 = 9.9%

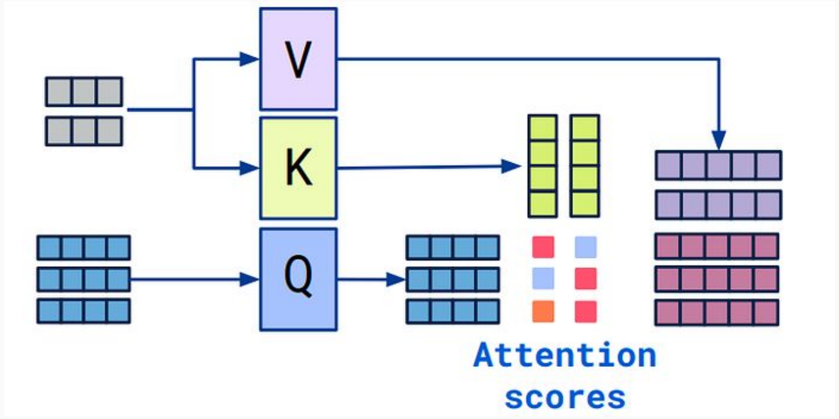
Understanding Softmax

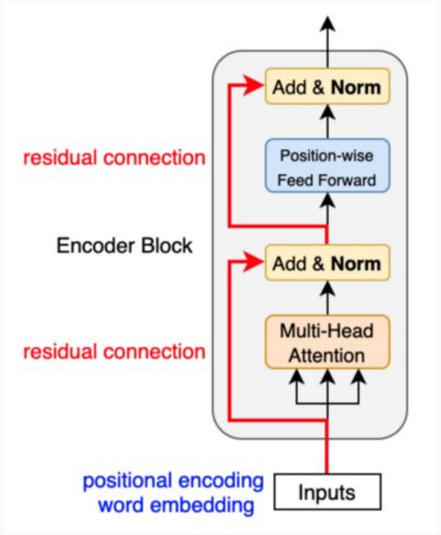

Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

https://www.linkedin.com/pulse/what-self-attention-impact-large-language-models-llm-nikhil-goel-srpbc

Self-Attention: How It Works

- Focuses on relationships between all positions in input.
- For each token/image patch:
  a. Compute Query, Key, and Value vectors.
  b. Attention = $\mathrm{softmax}(QK^T / \sqrt{d_k}) \times V$





What are K, Q, and V?

1. **K (Key)**: This is the data used to compare and find relationships.

2. **Q (Query)**: This is the sequence or word you are interested in finding relationships with.

3. **V (Value)**: This is the actual data (information) you will use after calculating the relationships (from Q and K).

Simple Example to Explain

Let’s say we have three words: **"cat"**, **"dog"**, and **"animal"**, and we want the system to understand the relationship between these words, such as how "cat" and "dog" relate (they are both pets or animals).

Steps of calculation:

1. **Q (Query)**: We’re interested in the word "cat", which will be our **Query**.

2. **K (Key)**: The set of words we're comparing against, so **K** will be: ["cat", "dog", "animal"].

3. **V (Value)**: The values are the associated information of these words. For example, **V** could be something like ["cat-info", "dog-info", "animal-info"], which represents the data related to these words.

Simple Example in Action:

Let’s say we have:

- **Q** = "cat" (the word we're interested in)
- **K** = ["cat", "dog", "animal"] (the set of words we’re comparing to)
- **V** = ["cat-info", "dog-info", "animal-info"] (the data associated with these words)

The calculation steps will help the system understand that "cat" is more related to "animal" than to "dog", allowing the model to give more importance to the word "animal".

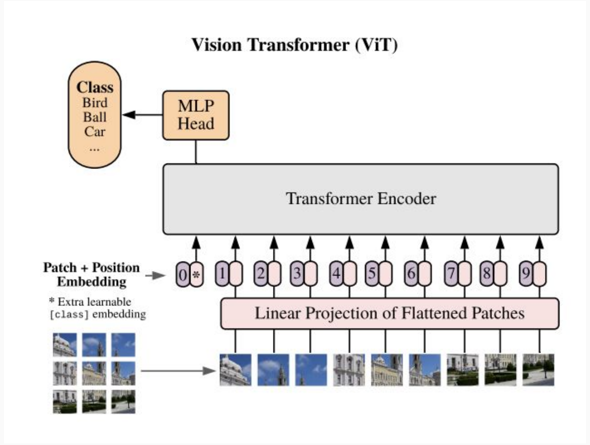
📦 Transformers in Vision (ViT)

- Vision Transformer splits an image into patches.
- Each patch is treated like a word/token.
- Position info is added (Positional Encoding).
- All patches attend to each other using Self-Attention.

Architecture:

1. Patch Embedding
2. Positional Encoding
3. Transformer Encoder Layers
4. Classification Head

Useful for large datasets with global patterns!


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0 *
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches

https://arxiv.org/pdf/2010.11929

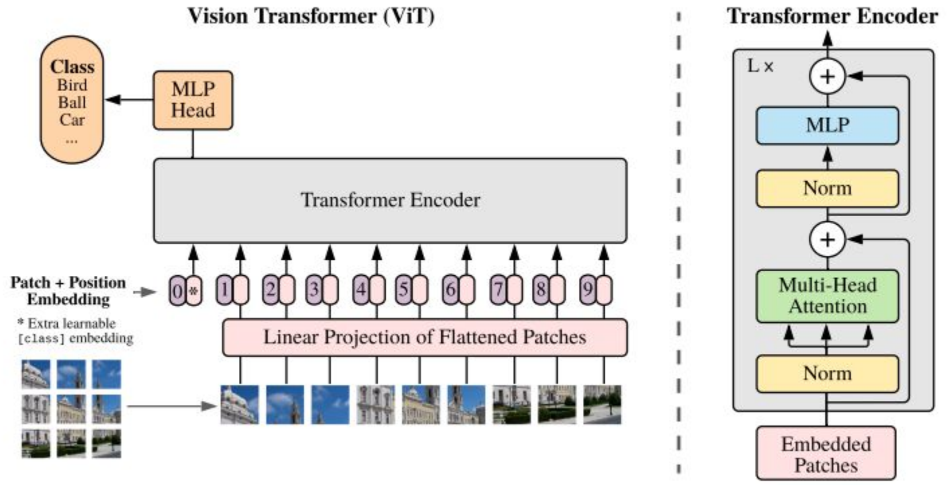


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification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

(Image) Simple Example in Action:

Let’s say we have an image that is divided into 4 patches:

- **Q** = Query patch (a specific patch you want to focus on)
- **K** = All patches (including the one in Q)
- **V** = Feature representations of all patches

🧪 Let's Do the Lab!

🎯 **Project Goal**: Classify faces of 10 Thai Prime Ministers using deep learning!

- You’ll start from scratch: load and explore data
- Build custom CNNs and try ViT
- Use transforms, data loaders, and metrics
- Experiment with tricks to boost accuracy
- Save and evaluate your best model!

🏁 **Output**: A trained model + confusion matrix + sample predictions

💡 Tip: You’ve already learned all the theory. Now it’s time to apply it!

🖥️ Open your Colab, and let’s build your first real-world face recognition system!

PRIME MINISTERS OF THAILAND
Manopakorn Nitithada
Phahon Phonphayuhasena
Plaek Phibunsongkhram
Khuang Aphaiwong
Thawi Bunyaket
Seni Pramoj
Khuang Aphaiwong
Pridi Banomyong
Thawan Thamrongnawasawat
Khuang Aphaiwong
Plaek Phibunsongkhram
Pote Sarasin
Thanom Kittikachorn
Sarit Thanarat
Thanom Kittikachorn
Sanya Dharmasakti
Seni Pramoj
Kukrit Pramoj
Seni Pramoj
Thanin Kraivichien
Kriangsak Chamanan
Prem Tinsulanonda
Chatichai Choonhavan
Anand Panyarachun
Suchinda Kraprayoon
Anand Panyarachun
Chuan Leekpai
Banharn Silpa-archa
Chavalit Yongchaiyudh
Chuan Leekpai
Thaksin Shinawatra
Surayud Chulanont
Samak Sundaravej
Somchai Wongsawat
Abhisit Vejjajiva
Yingluck Shinawatra
Prayut Chan-o-cha
Srettha Thavisin
Paetongtarn Shinawatra