# RESEARCH PRESENTATION

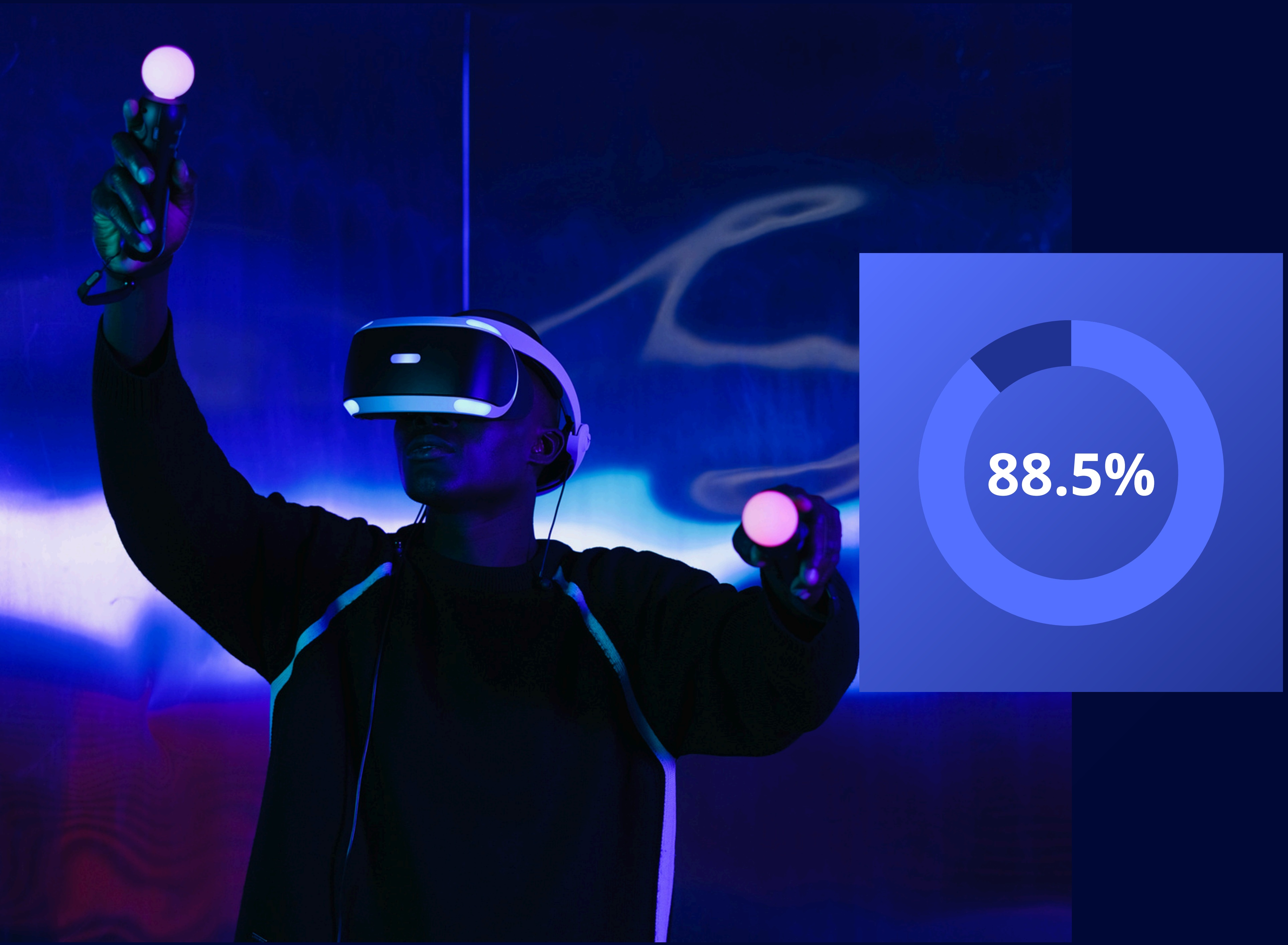
## An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale - ViT

FAST NUCES

Presented By: Nerdy Freaks

# WARM UP !!

How many of you have worked with CNNs before?
What’s one limitation you’ve encountered with them




# OBJECTIVE

Vision Transformer (ViT), which shows that a transformer-based architecture can outperform state-of-the-art CNN-based models (such as ResNet) on image classification tasks when trained on sufficient data.

“Can you guess what accuracy ViT achieved on the ImageNet dataset?
Hint: it beat ResNet.”

# RESULTS

01

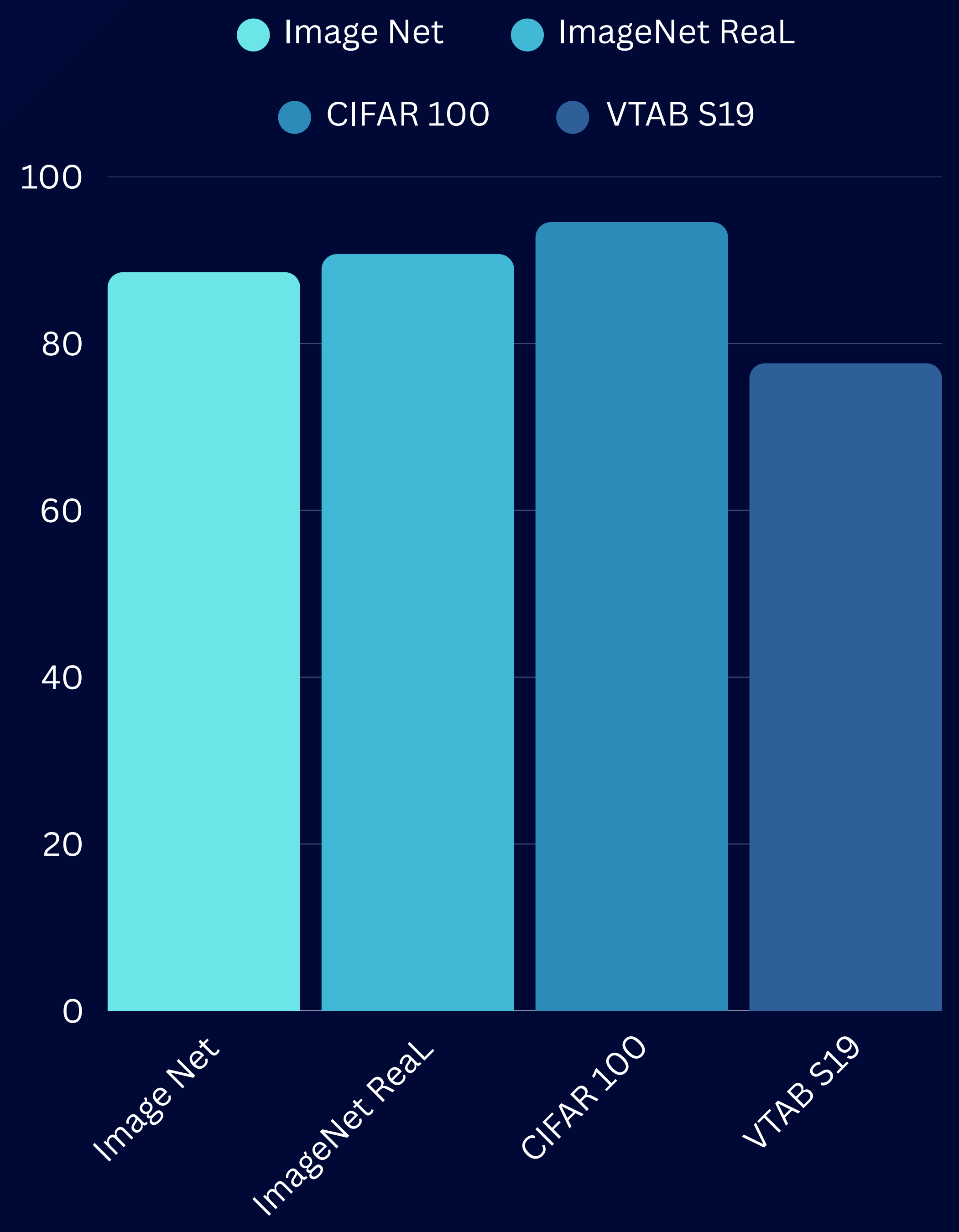
The ViT achieved state-of-the-art performance on the ImageNet benchmark, surpassing the performance of convolutional neural networks (CNNs) on image classification tasks

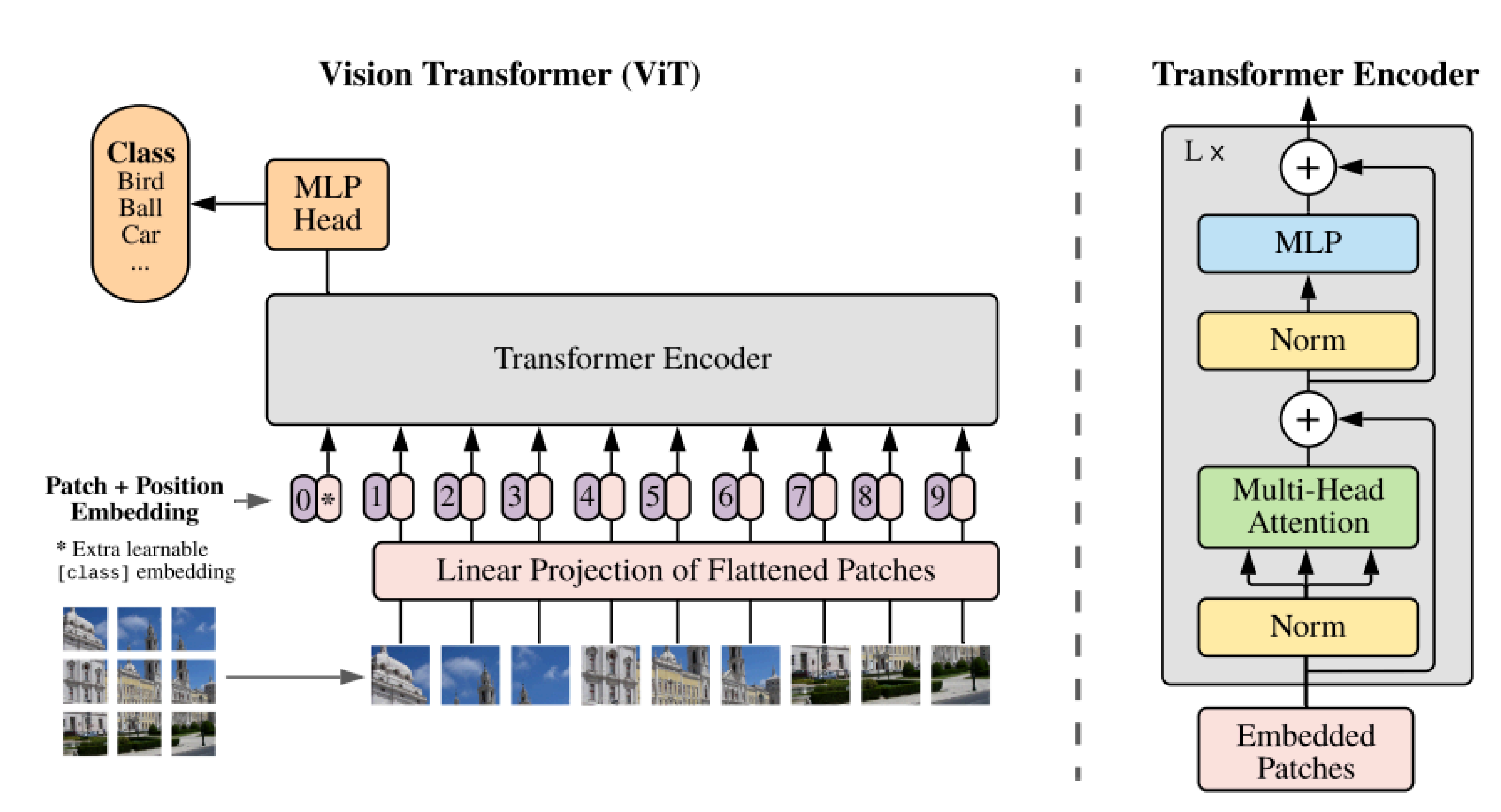
02

It demonstrates that transformers can scale well for computer vision tasks, especially when trained on large datasets.

03

By training ViT on large datasets like JFT-300M (a dataset with 300 million labeled images), it achieved an impressive accuracy of 88.5% on ImageNet, outperforming traditional architectures like ResNet.






# ARCHITECURE

01

**Patch Embeddings:**
ViT splits an image into non-overlapping patches and flattens them into a sequence of tokens, similar to how words are treated in NLP with transformers. Each patch is embedded into a flat vector (like a word embedding).

02

**Transformer Encoder**
The main component of ViT is the transformer encoder, which processes the sequence of patch embeddings. The attention mechanism in transformers allows the model to capture long-range dependencies in the image.

03

**Positional Encoding**
Since transformers don't have an inherent understanding of the order of the patches (like CNNs have with local receptive fields), positional encodings are added to preserve the spatial information of the patches.

04

**MLP Head**
After processing the patch tokens through the transformer layers, a multi-layer perceptron (MLP) is used for the classification task.

Which optimizer have you found most effective in your own projects—SGD, Adam, or something else?”

# DEATILS FOR IMPROVEMENT

**01**

**Large-scale Pretraining**
ViT achieves its best results when trained on massive datasets (such as JFT-300M), showing that a large amount of data is crucial for transformers to outperform CNNs.

**02**

**Patch Size**
The choice of patch size (16x16 pixels in the ViT paper) is essential, as it determines the granularity of information the model will work with. Larger patches may lose finer details, while smaller patches may lead to inefficient training.

**03**

**Regularization:**
ViT utilizes techniques like dropout to avoid overfitting when trained on large datasets.

**04**

**Data Augmentation:**
Data augmentation is critical to prevent overfitting, especially on smaller datasets like ImageNet.

# TRAINING OF MODEL

- The model is typically trained using cross-entropy loss for image classification tasks.
- Optimizers like Adam or AdamW are commonly used.
- Learning rate schedulers are employed to adjust the learning rate throughout training for better convergence.

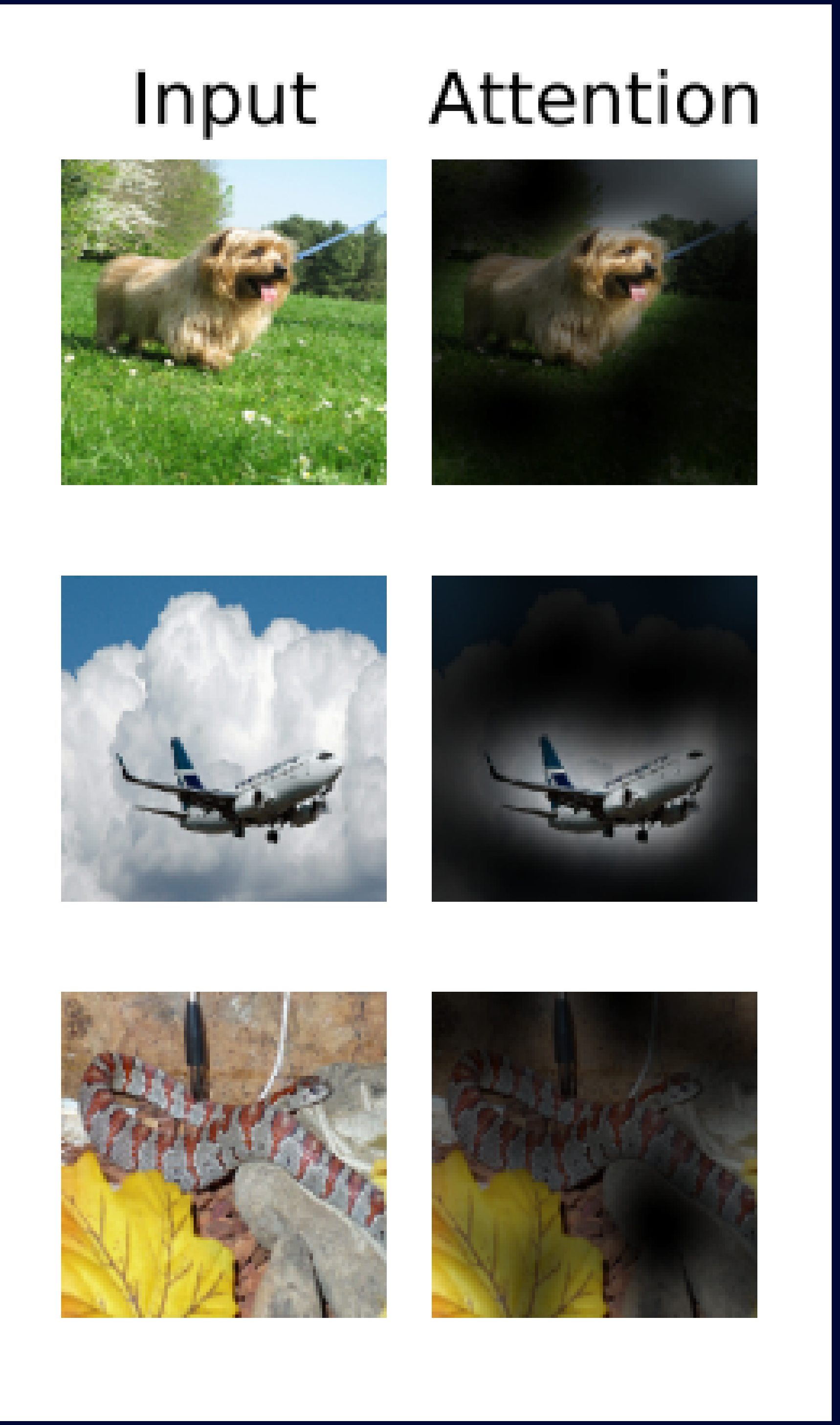
**Orginial Model Hyperparameters:**

- Steps - 1 Millions
- Learning Rate - 2 x 10^-4
- Warmup of 10k steps and cosine learning rate decay.
-

| name | Epochs | ImageNet | ImageNet ReaL | CIFAR-10 | CIFAR-100 | Pets | Flowers | exaFLOPs |
|---|---|---|---|---|---|---|---|---|
| ViT-B/32 | 7 | 80.73 | 86.27 | 98.61 | 90.49 | 93.40 | 99.27 | 55 |
| ViT-B/16 | 7 | 84.15 | 88.85 | 99.00 | 91.87 | 95.80 | 99.56 | 224 |
| ViT-L/32 | 7 | 84.37 | 88.28 | 99.19 | 92.52 | 95.83 | 99.45 | 196 |
| ViT-L/16 | 7 | 86.30 | 89.43 | 99.38 | 93.46 | 96.81 | 99.66 | 783 |
| ViT-L/16 | 14 | 87.12 | 89.99 | 99.38 | 94.04 | 97.11 | 99.56 | 1567 |
| ViT-H/14 | 14 | 88.08 | 90.36 | 99.50 | 94.71 | 97.11 | 99.71 | 4262 |
| ResNet50x1 | 7 | 77.54 | 84.56 | 97.67 | 86.07 | 91.11 | 94.26 | 50 |
| ResNet50x2 | 7 | 82.12 | 87.94 | 98.29 | 89.20 | 93.43 | 97.02 | 199 |
| ResNet101x1 | 7 | 80.67 | 87.07 | 98.48 | 89.17 | 94.08 | 95.95 | 96 |
| ResNet152x1 | 7 | 81.88 | 87.96 | 98.82 | 90.22 | 94.17 | 96.94 | 141 |
| ResNet152x2 | 7 | 84.97 | 89.69 | 99.06 | 92.05 | 95.37 | 98.62 | 563 |
| ResNet152x2 | 14 | 85.56 | 89.89 | 99.24 | 91.92 | 95.75 | 98.75 | 1126 |
| ResNet200x3 | 14 | 87.22 | 90.15 | 99.34 | 93.53 | 96.32 | 99.04 | 3306 |
| R50x1+ViT-B/32 | 7 | 84.90 | 89.15 | 99.01 | 92.24 | 95.75 | 99.46 | 106 |
| R50x1+ViT-B/16 | 7 | 85.58 | 89.65 | 99.14 | 92.63 | 96.65 | 99.40 | 274 |
| R50x1+ViT-L/32 | 7 | 85.68 | 89.04 | 99.24 | 92.93 | 96.97 | 99.43 | 246 |
| R50x1+ViT-L/16 | 7 | 86.60 | 89.72 | 99.18 | 93.64 | 97.03 | 99.40 | 859 |
| R50x1+ViT-L/16 | 14 | 87.12 | 89.76 | 99.31 | 93.89 | 97.36 | 99.11 | 1668 |

# STEPWISE PROCESS

- Load the dataset and apply preprocessing (resizing, augmentation).
- Convert the image into patches and embed them. Add the positional encoding.
- Pass the embedded patches through the transformer layers for encoding and multiheaded self attention.
- Use the MLP head for classification/detection.
- Train the model using a loss function and backpropagate errors.



# CODE BRIEFING

```
class ViT(nn.Module):
    def __init__(self, ch=3, img_size=144, patch_size=4, emb_dim=32,
                n_layers=6, out_dim=37, dropout=0.1, heads=2):
        super(ViT, self).__init__()

        # Attributes
        self.channels = ch
        self.height = img_size
        self.width = img_size
        self.patch_size = patch_size
        self.n_layers = n_layers

        # Patching
        self.patch_embedding = PatchEmbedding(in_channels=ch,
                                              patch_size=patch_size,
                                              emb_size=emb_dim)
        # Learnable params
        num_patches = (img_size // patch_size) ** 2
        self.pos_embedding = nn.Parameter(
            torch.randn(1, num_patches + 1, emb_dim))
        self.cls_token = nn.Parameter(torch.rand(1, 1, emb_dim))

        # Transformer Encoder
        self.layers = nn.ModuleList([])
        for _ in range(n_layers):
            transformer_block = nn.Sequential(
                ResidualAdd(PreNorm(emb_dim, Attention(emb_dim, n_heads = heads, dropout = dropout))),
                ResidualAdd(PreNorm(emb_dim, FeedForward(emb_dim, emb_dim, dropout = dropout))))
            self.layers.append(transformer_block)

        # Classification head
        self.head = nn.Sequential(nn.LayerNorm(emb_dim), nn.Linear(emb_dim, out_dim))
```

# WHY VTS

**Minimal Inductive Bias:**
Unlike CNNs, ViT relies less on assumptions about the data, allowing it to learn more flexible features.

**Better Scalability:**
Unlike CNNs which often saturate in performance, ViTs continue to improve when trained on massive datasets (e.g., JFT-300M, ImageNet-21k).

**Efficient Architecture**
The use of transformers facilitates modeling long-range dependencies and global context effectively.

“Why do you think CNNs perform better on small datasets compared to ViTs? Any ideas?”

# Questions

?