

# of a Pre-Trained Model

## EXP No: 13:- Understanding the Architecture

### Aim:-

To study and analyze the architecture and Components of pre-trained CNN models such as VGG16, ResNet 50, and Inception V3.

### Objectives:-

1. To understand transfer learning using pre-trained deep learning models.
2. To explore layer structure, feature extraction, and fine-tuning techniques.
3. To visualize and analyze learned representations from pre-trained models.
4. To load and summarize model architecture in Tensorflow / keras.

### Algorithms:-

1. Import a pre-trained model with SwiggleNet weights.
2. Load the model without top layers.
3. Display the model summary to understand architecture.
4. Analyze convolutional, pooling, and fully connected layers.
5. Use a sample image to visualize model predictions or features.

## pseudo code:-

Import pre-trained model

Load model without top layer.

Display model.summary()

visualize layers & parameters.

Run a sample image through model for prediction

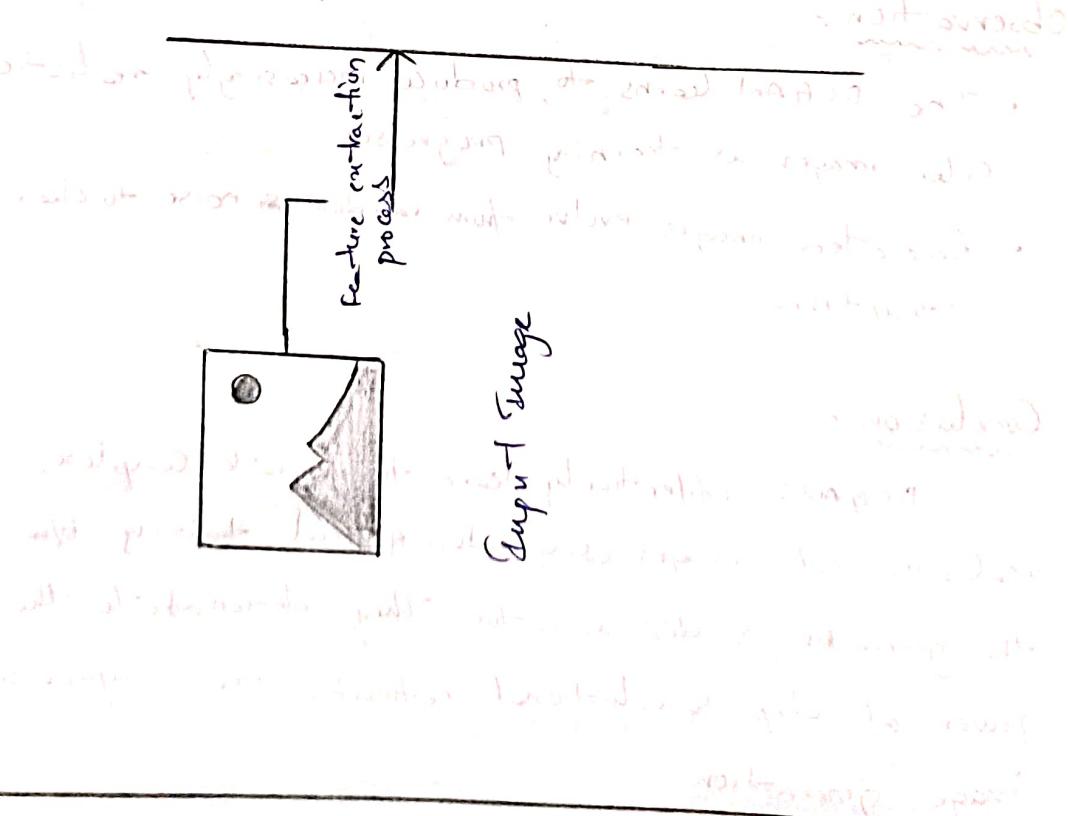
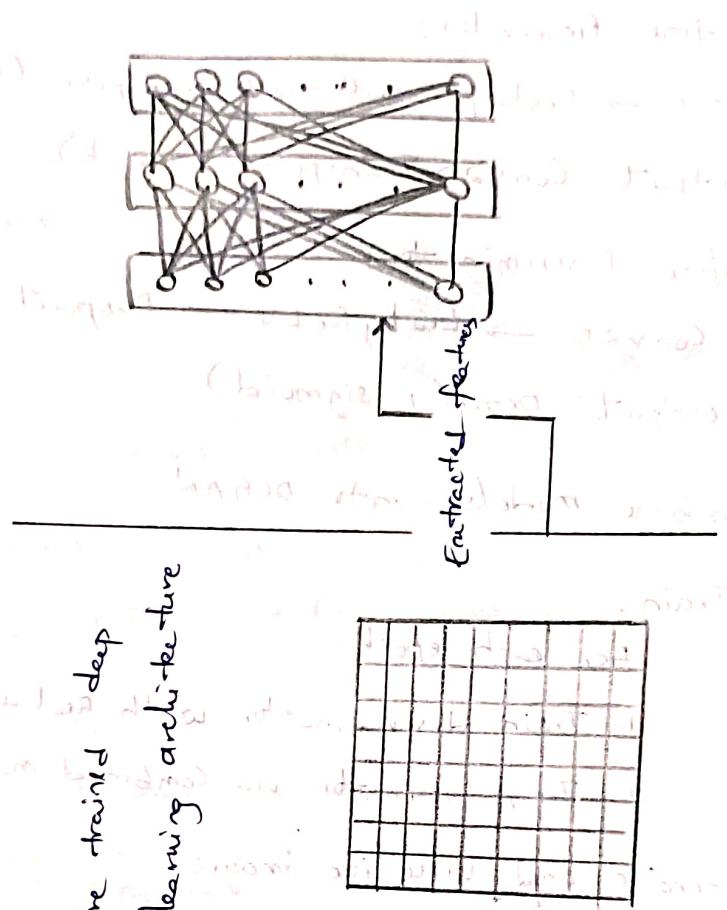
## Observation:-

- VGG16 :- Deep sequential network with 13 convolutional + 3 dense layers.
- ResNet 50 :- Uses skip connections to avoid vanishing gradients.
- Inception V3 :- Employs parallel convolution paths for multi-scale feature extraction
- Each pre-trained model captures hierarchical image features efficiently.

## Conclusion:-

pre-trained models like VGG16, ResNet 50, and Inception V3 provide powerful feature extractors trained on large datasets. Understanding their architectures help apply transfer learning, improving accuracy & training efficiency in new tasks.

## USAGE OF PRE-TRAINED ARCHITECTURE



## Output:

Total parameters: 138357544

Trainable parameters: 138357544

Non Trainable parameters: 0

## Layers names:

conv0: Conv2D

bn1: Batch Normal 2D

relu: ReLU

maxpool: Maxpool 2D

layer1: sequential

layer2: sequential

layer3: sequential

layer4: sequential

avgpool: Avg pool 2D

final

introduction of layers and their functions

conv0: Convolutional layer

bn1: Batch Normalization layer

relu: Activation function

CO dltlab13.ipynb ⚡

File Edit View Insert Runtime Tools Help

Commands + Code + Text | ▶ Run all ▾

[ ]

```
# lab13_pretrained_model_architecture.py
import torch
import torchvision.models as models

# Load a pretrained model (VGG16)
model = models.vgg16(pretrained=True)

# Print model architecture
print("===== VGG16 Pre-trained Model Architecture =====")
print(model)

# Print only feature extractor part
print("\n===== Feature Extractor Layers =====")
print(model.features)

# Print classifier part
print("\n===== Classifier Layers =====")
print(model.classifier)

# Print total trainable parameters
total_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f"\nTotal Trainable Parameters: {total_params}")
```

```
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or 'None' for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to using 'None'.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/hub/checkpoints/vgg16-397923af.pth
100%|██████████| 52MB/52MB [00:01<00:00, 14MB/s]
===== VGG16 Pre-trained Model Architecture =====
VGG(
  features: Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

```
,  
  
===== Feature Extractor Layers =====  
Sequential(  
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (1): ReLU(inplace=True)  
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (3): ReLU(inplace=True)  
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (6): ReLU(inplace=True)  
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (8): ReLU(inplace=True)  
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (11): ReLU(inplace=True)  
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (13): ReLU(inplace=True)  
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (15): ReLU(inplace=True)  
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (18): ReLU(inplace=True)  
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (20): ReLU(inplace=True)  
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (22): ReLU(inplace=True)  
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (25): ReLU(inplace=True)  
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (27): ReLU(inplace=True)  
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (29): ReLU(inplace=True)  
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
)  
  
===== Classifier Layers =====  
Sequential(  
    (0): Linear(in_features=25088, out_features=4096, bias=True)  
    (1): ReLU(inplace=True)  
    (2): Dropout(p=0.5, inplace=False)  
    (3): Linear(in_features=4096, out_features=4096, bias=True)  
    (4): ReLU(inplace=True)  
    (5): Dropout(p=0.5, inplace=False)  
    (6): Linear(in_features=4096, out_features=1000, bias=True)  
)  
  
Total Trainable Parameters: 138357544
```