Public: 0.9665

Private: 0.9867

Case 3

IMAGE CLASSIFICATION OF AUTOMOBILES

“ZhmemPoSto” team

QR to our github

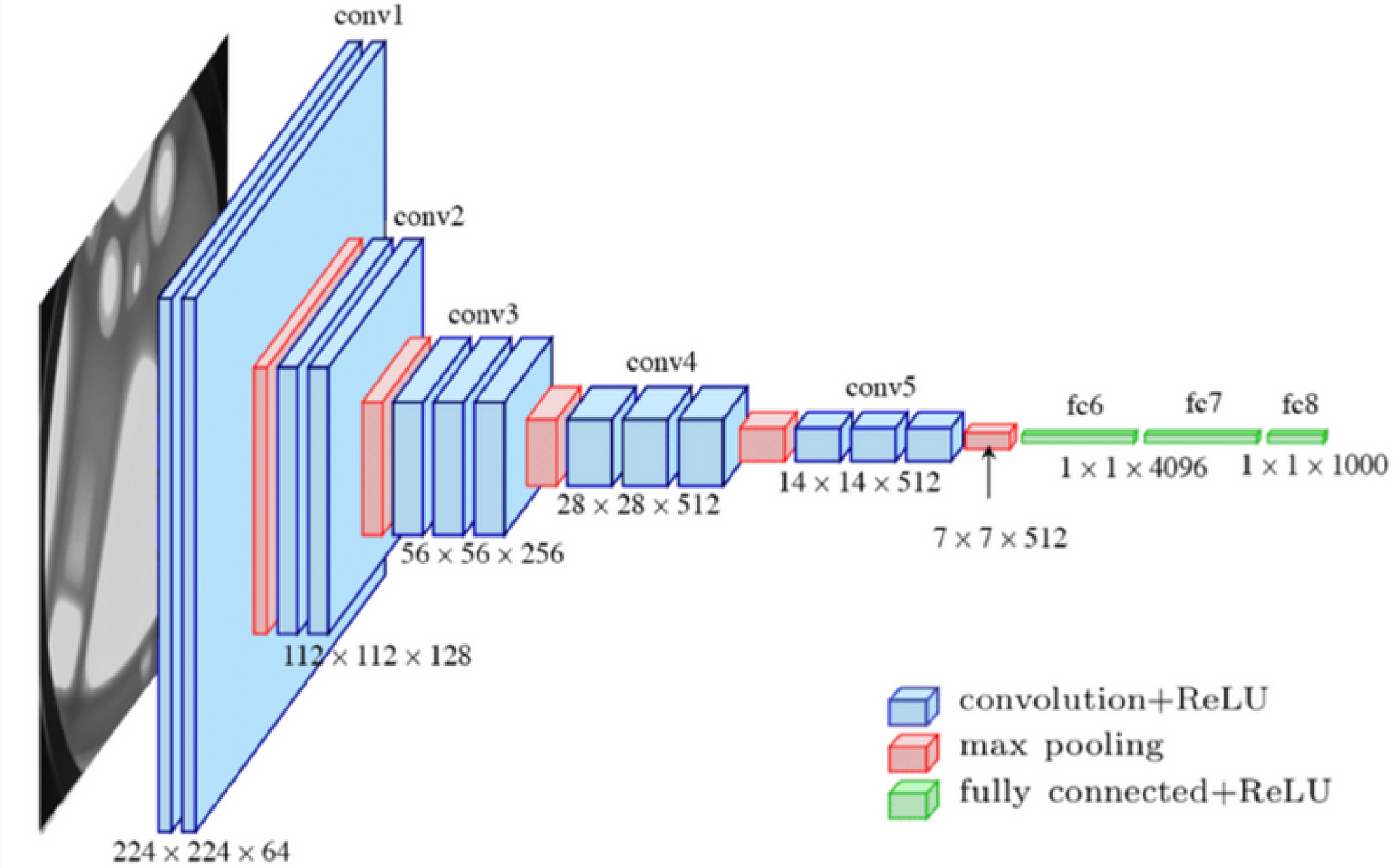
Pathway

Model Training
Evaluation

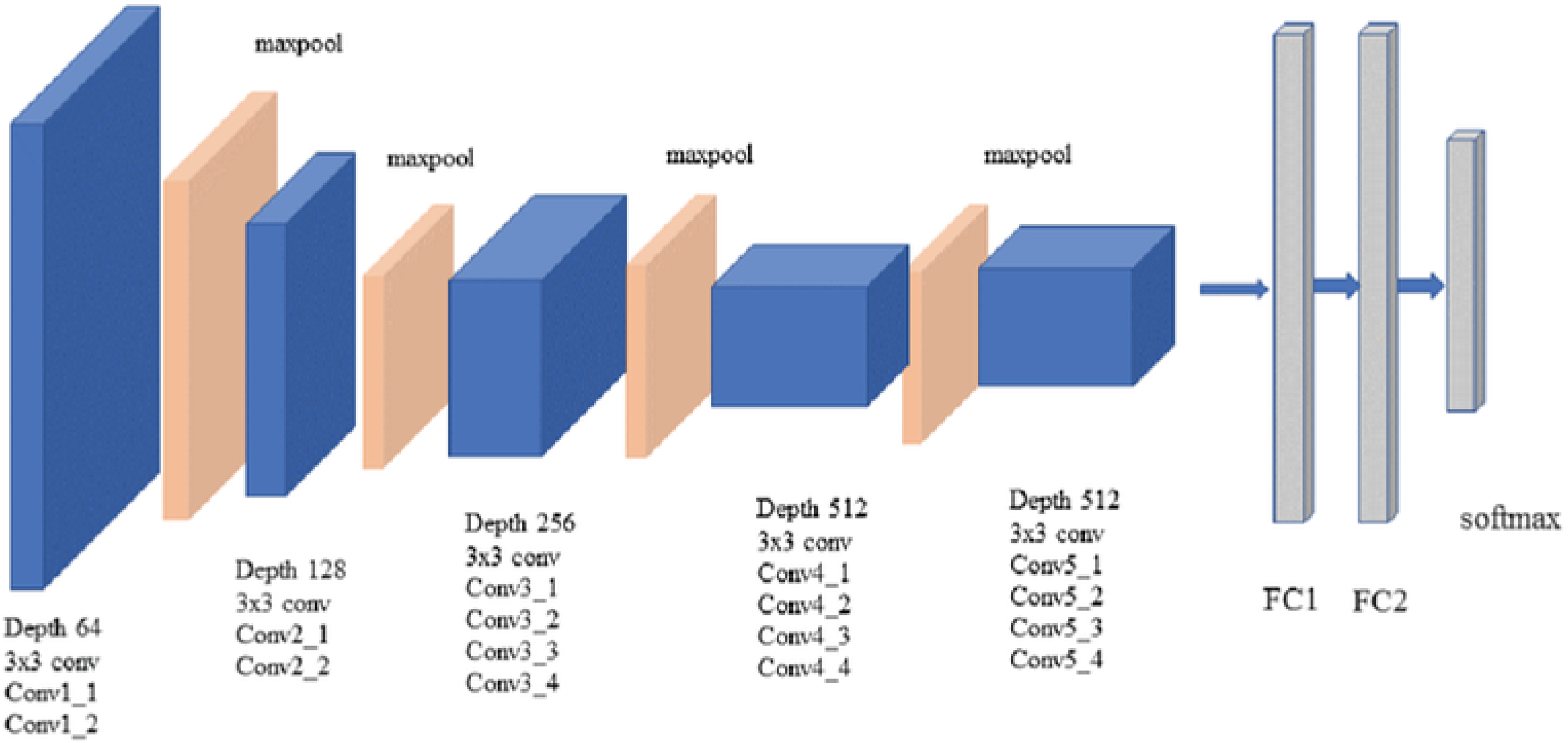
Model building
VGG-16

Data Prepocessing

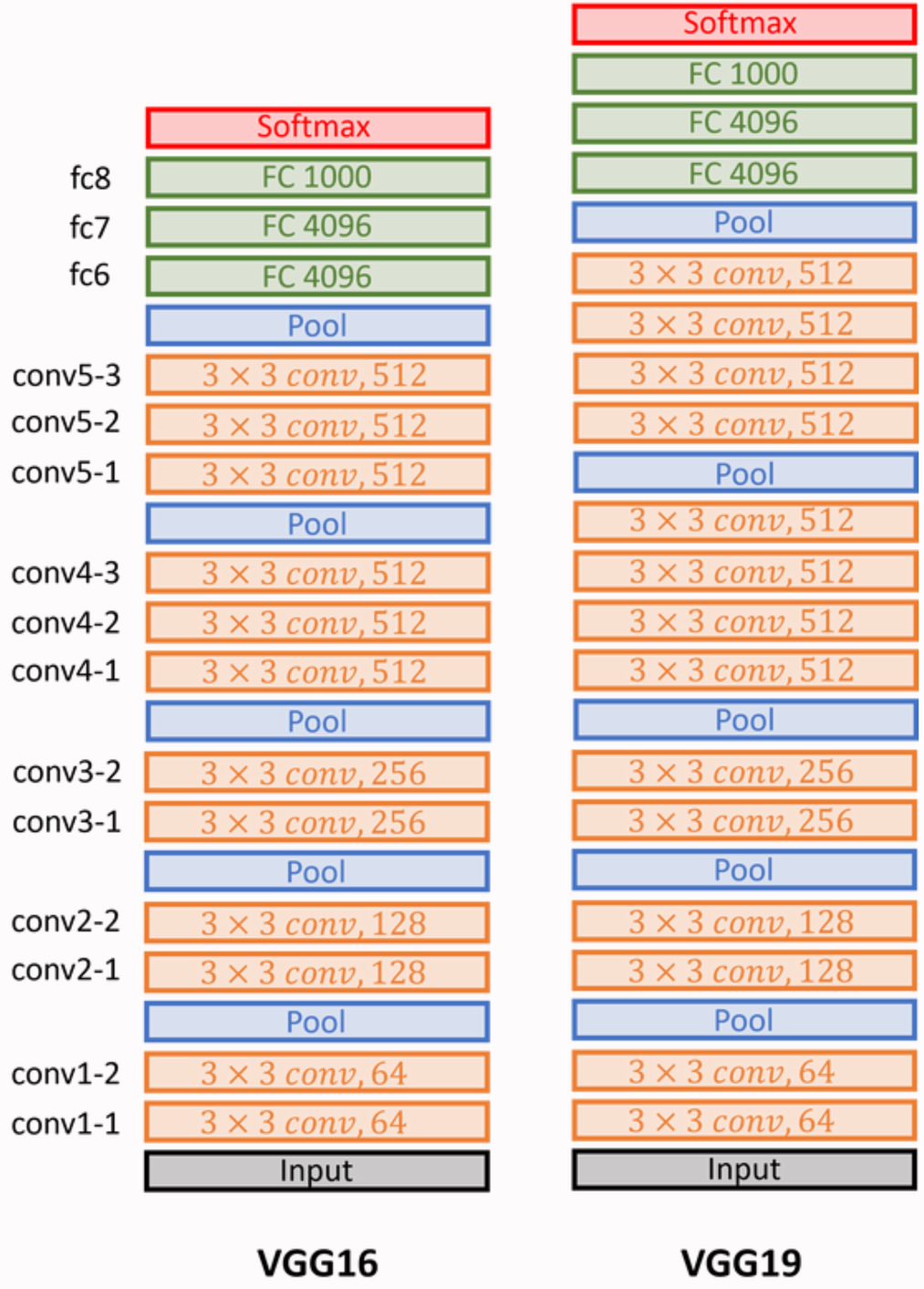
Data Exploration
Hypothesis analysis,
Feature Selection


conv1
conv2
conv3
conv4
conv5
fc6
fc7
fc8
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU

VGG-16


maxpool
maxpool
maxpool
maxpool
Depth 64
3x3 conv
Conv1_1
Conv1_2
Depth 128
3x3 conv
Conv2_1
Conv2_2
Depth 256
3x3 conv
Conv3_1
Conv3_2
Conv3_3
Conv3_4
Depth 512
3x3 conv
Conv4_1
Conv4_2
Conv4_3
Conv4_4
Depth 512
3x3 conv
Conv5_1
Conv5_2
Conv5_3
Conv5_4
FC1
FC2
softmax

VGG-19


Softmax
FC 1000
FC 4096
FC 4096
Pool
3 × 3 conv, 512
3 × 3 conv, 512
3 × 3 conv, 512
3 × 3 conv, 512
Pool
3 × 3 conv, 512
3 × 3 conv, 512
3 × 3 conv, 512
3 × 3 conv, 512
Pool
3 × 3 conv, 256
3 × 3 conv, 256
Pool
3 × 3 conv, 128
3 × 3 conv, 128
Pool
3 × 3 conv, 64
3 × 3 conv, 64
Input
VGG19
Softmax
FC 1000
FC 4096
fc8
FC 4096
fc7
Pool
fc6
3 × 3 conv, 512
conv5-3
3 × 3 conv, 512
conv5-2
3 × 3 conv, 512
conv5-1
Pool
3 × 3 conv, 512
conv4-3
3 × 3 conv, 512
conv4-2
3 × 3 conv, 512
conv4-1
Pool
3 × 3 conv, 256
conv3-2
3 × 3 conv, 256
conv3-1
Pool
3 × 3 conv, 128
conv2-2
3 × 3 conv, 128
conv2-1
Pool
3 × 3 conv, 64
conv1-2
3 × 3 conv, 64
conv1-1
Input
VGG16

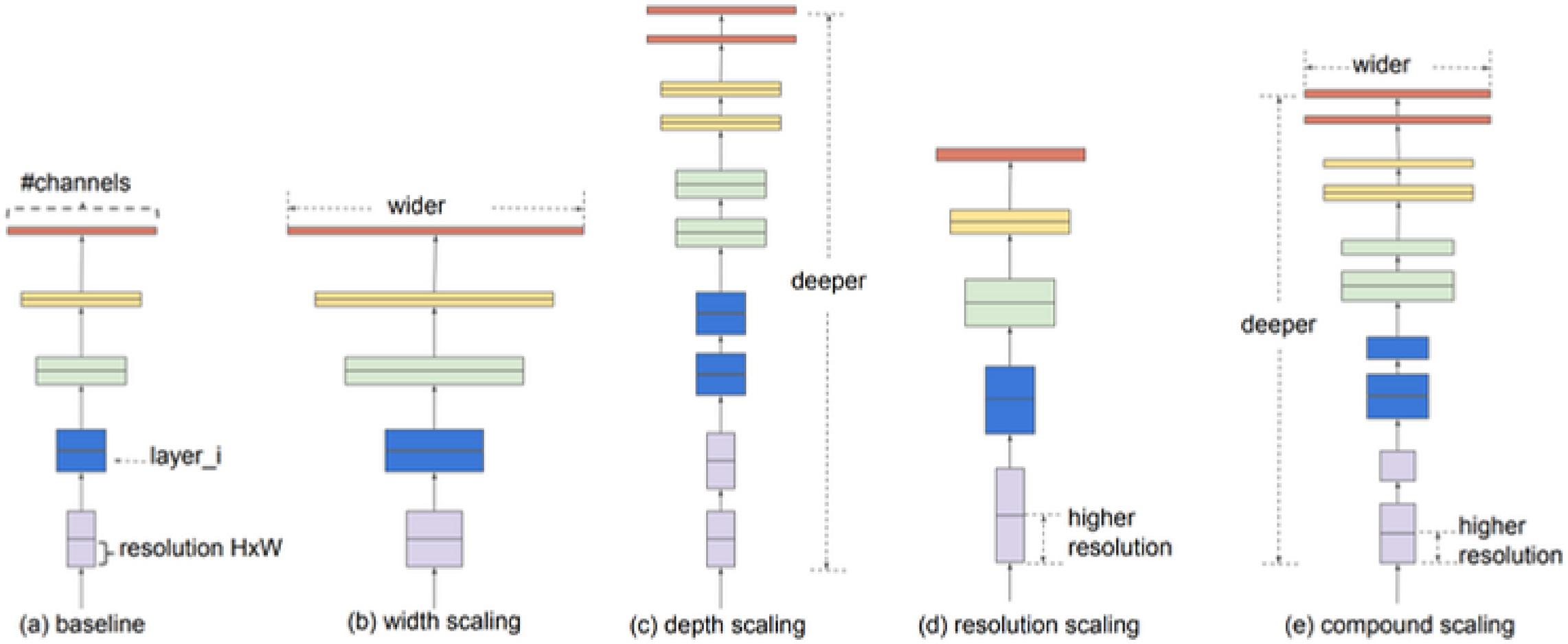
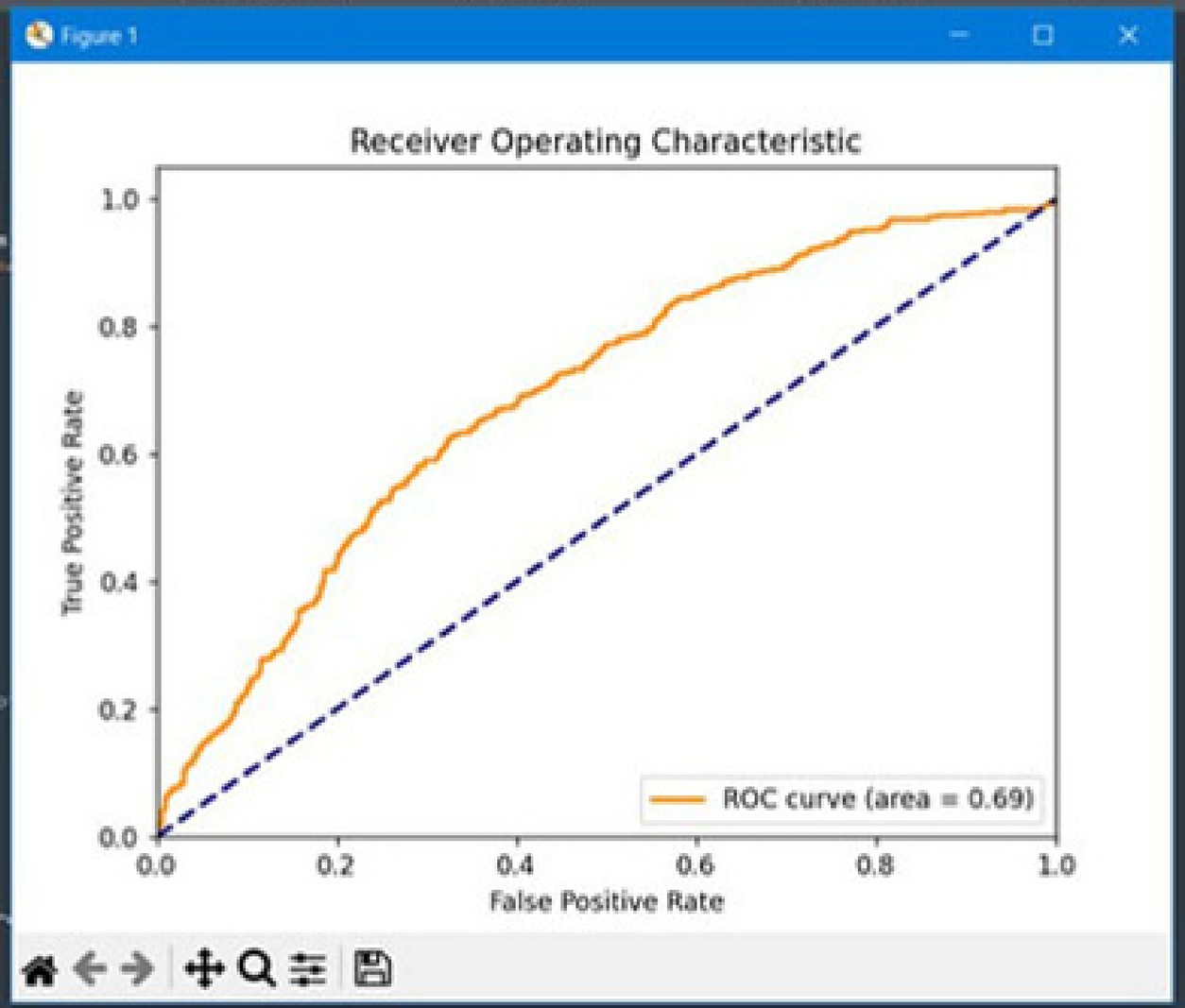


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

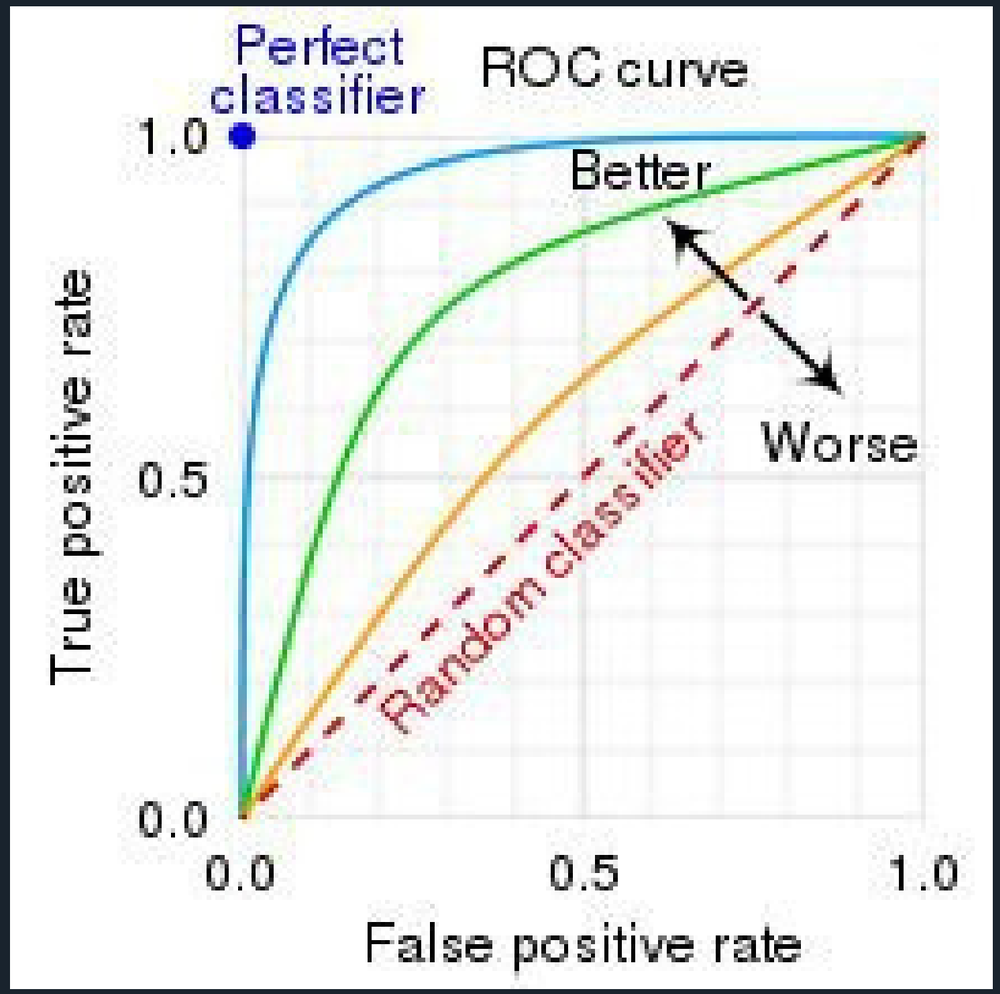
Efficient Net

1.py — Users\...\Telegram Desktop | dasdas.py | 2.py | roc | 1.py — python1 | load3.py | untitled4.py

```python
    # Define data transforms
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])

    train_dataset = torchvision.datasets.ImageFolder(root="test", transform=transform
    data_loader = DataLoader(train_dataset, batch_size=16, shuffle=True, num_workers=
# Initialize lists to store true labels and model's score
    true_labels = []
    model_scores = []

    # Evaluate the model
    with torch.no_grad():
        for data in data_loader:
            images, labels = data
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)
            sm = torch.nn.Softmax(dim=1)  # Create a Softmax
            probabilities = sm(outputs)  # Get the probability of each class

            true_labels.extend(labels.cpu().numpy())
            model_scores.extend(probabilities[:, 1].cpu().numpy())  # Get the score o

    # Compute ROC curve
    fpr, tpr, _ = roc_curve(true_labels, model_scores)
    roc_auc = auc(fpr, tpr)

    # Plotting ROC Curve
    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % r
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc="lower right")
    plt.show()
```



ROC Curves


Perfect classifier
ROC curve
1.0
Better
0.5
Worse
Random classifier
True positive rate
0.0
0.0
0.5
1.0
False positive rate

temp.py | train2.py | untitled1.py | untitled2.py | myvgg16.py | untitled3.py | untitled4.py

```python
import os
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim
from torchvision.models import vgg16
from torch.utils.data import DataLoader, random_split

# Define data transforms
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Define data directory
data_dir = 'train_data'

# Load and prepare the dataset
dataset = torchvision.datasets.ImageFolder(root=data_dir, transform=transform)

# Split the dataset into training and validation sets
val_split = 0.2  # You can adjust the validation split ratio
val_size = int(val_split * len(dataset))
train_size = len(dataset) - val_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

# Create data loaders for training and validation
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=4)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, num_workers=4)
print(torch.cuda.is_available())

num_classes = len(dataset.classes)
# Define the model
model = vgg16(pretrained=True)
model.classifier[6] = nn.Linear(4096, num_classes)  # Modify for your specific classification

# Set the device
```

Help | Variable Explorer | Plots | Files

Console 1/A

```
Class 2: 0.9301
Class 3: 0.8000
Class 4: 0.7368
Accuracy (Fold 1, Epoch 6, sec 101.75319457054138): 0.9400
F1 Scores (Fold 1, Epoch 7, sec 101.76398611068726):
Class 0: 0.9779
Class 1: 0.6780
Class 2: 0.9201
Class 3: 0.7719
Class 4: 0.7047
Accuracy (Fold 1, Epoch 7, sec 101.76398611068726): 0.9355
F1 Scores (Fold 1, Epoch 8, sec 100.71692538261414):
Class 0: 0.9772
Class 1: 0.6857
Class 2: 0.9105
Class 3: 0.7059
Class 4: 0.7320
Accuracy (Fold 1, Epoch 8, sec 100.71692538261414): 0.9317
F1 Scores (Fold 1, Epoch 9, sec 101.63109302520752):
Class 0: 0.9709
Class 1: 0.7164
Class 2: 0.9280
Class 3: 0.7164
Class 4: 0.6981
Accuracy (Fold 1, Epoch 9, sec 101.63109302520752): 0.9279
F1 Scores (Fold 1, Epoch 10, sec 101.544499874114999):
Class 0: 0.9742
Class 1: 0.6667
Class 2: 0.9352
Class 3: 0.7541
Class 4: 0.6965
Accuracy (Fold 1, Epoch 10, sec 101.544499874114999): 0.9339
Fold 2:
```