Report Title: **VGG and ResNet**

Author Name: **王小慧**

Student ID: **21307110415**

Institute: **信息科学与工程学院**

Report Date: **2024/12/1**

Contents

1 Introduction

This experiment explores the application of deep learning architectures, VGG-16 and ResNet-18, in image classification using the CIFAR-10 dataset. The CIFAR-10 dataset is a standard benchmark in computer vision tasks, consisting of 60,000 color images across 10 categories, with each image having a resolution of 32*32 pixels. Its RGB channels and relatively small size make it challenging for precise object recognition, necessitating advanced neural network architectures.

1.1 VGG

The VGG model[1], developed by the Visual Geometry Group at the University of Oxford, is a deep convolutional neural network that gained recognition through its success in the 2014 ImageNet Large Scale Visual Recognition Challenge. The architecture emphasizes simplicity by employing stacked 3×3 convolutional layers interspersed with max-pooling layers, progressively reducing the spatial dimensions while increasing feature depth.

1.2 ResNet

ResNet[2], or Residual Network, introduced a breakthrough concept to address the vanishing gradient problem common in very deep networks. By incorporating residual blocks, ResNet facilitates identity mapping via shortcut connections, which skip one or more layers. The ResNet-18 model comprises stacked residual blocks, where each block adds the shortcut input directly to the block's output. The mathematical principle behind the residual block is:

$$y = F(x, W) + x \tag{1}$$

where F(x,W) is the transformation function learned by the convolutional layers, and x represents the input directly added through the shortcut, as shown in Figure 1.

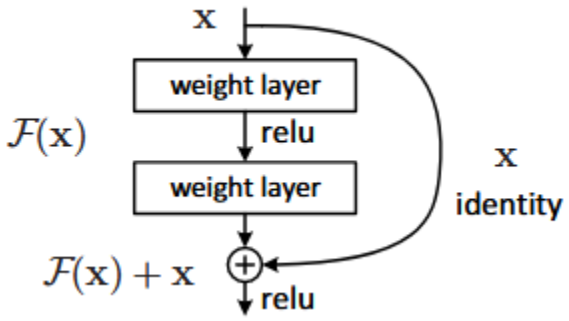

Figure 1 Residual Learning

This mechanism enables efficient training of deeper networks without degradation in performance, as it allows the network to learn adjustments instead of full transformations.

1.3 DenseNet

DenseNet[3], short for Dense Convolutional Network, it addresses some key challenges in training deep neural networks, such as vanishing gradients, overfitting, and parameter inefficiency, by ensuring maximum information flow between layers through dense connectivity.

Architecture:

- **Dense Block:** A sequence of layers where each layer is connected to every other layer in a feed-forward fashion. Inside a dense block, feature maps are concatenated rather than added. Figure 2 illustrates a 5-layer dense block with a growth rate of k = 4. Each layer takes all preceding feature-maps as input.
- **Transition Layer:** Added between dense blocks to reduce feature map dimensions and control overfitting. Comprises a BatchNorm layer, a 1×1 convolution, and a 2×2 average pooling layer.
- **Final Layers:** After the last dense block, global average pooling (GAP) is applied, followed by a fully connected layer for classification.

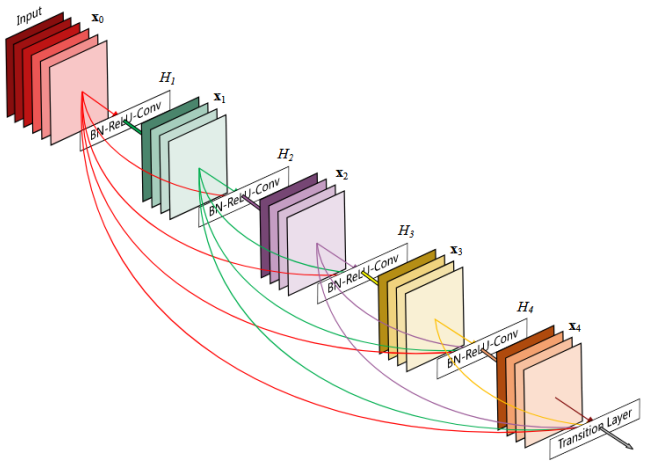

Figure 2 A 5-layer Dense Block

1.4 Data Augmentation

Data augmentation is an essential technique to improve the generalization capability of models like VGG-16 and ResNet-18. Here are some commonly used data augmentation techniques:

- **Horizontal Flipping:** Randomly flips the images along the vertical axis. Since the object class in CIFAR-10 is invariant to horizontal orientation, this technique enriches the dataset.
- **Random Cropping:** Extracts random patches of the original image, ensuring that the model learns to recognize objects irrespective of their exact placement within the image.
- **Color Jitter:** Randomly changes image properties like brightness, contrast, saturation, and hue. This helps the model generalize across varying lighting conditions.
- **Random Rotation:** Rotates the image by a small random angle, making the model less sensitive to the orientation of objects.

2 Experimental Setup

In this section, we explore three different deep learning architectures—VGG-16, ResNet-18, and DenseNet—on the CIFAR-10 dataset. Before the actual experiment, each model is designed and modified to suit the specific characteristics of the CIFAR-10 dataset, which consists of 60,000 32×32 color images belonging to 10 object classes.

2.1 VGG-16

VGG-16 includes 13 convolutional layers and 3 fully connected layers. The original VGG-16 architecture was designed for large-scale image classification on the ImageNet dataset, where input images are of size 224×224 pixels. For CIFAR-10, we adapt the VGG-16 architecture to accommodate the smaller input size of 32×32 pixels. The number of fully connected layers and the spatial dimensions after the convolutional layers are adjusted accordingly.

```
class VGG16(nn.Module):
    def __init__(self, num_classes=1000):
        super(VGG16, self).__init__()
        self.features = nn.Sequential(
            # 5 blocks
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
```

```
            nn.Conv2d(128, 128, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(256, 512, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )

        self.classifier = nn.Sequential(
            nn.Linear(512 * 1 * 1, 1024),  # Adjusting to match the output from the
                last layer
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(1024, 1024),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(1024, num_classes),  # Output for CIFAR-10 classes
        )

    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

2.2 ResNet-18

ResNet-18 consists of 18 learnable layers—16 convolutional layers and 2 fully connected layers. ResNet-18 uses shortcut connections to skip one or more layers, facilitating identity mappings and improving gradient flow during backpropagation. For CIFAR-10, the initial convolutional layer and the number of filters in each block are adjusted to work efficiently with 32X32 images, and the final fully connected layer is altered to output 10 classes.

```
class BasicBlock(nn.Module):
    def __init__(self, in_planes, out_planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=stride,
             padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_planes)
        self.conv2 = nn.Conv2d(out_planes, out_planes, kernel_size=3, stride=1,
            padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != out_planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, out_planes, kernel_size=1, stride=stride, bias
                    =False),
                nn.BatchNorm2d(out_planes)
            )

    def forward(self, x):
        out = nn.ReLU()(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        return nn.ReLU()(out)

class ResNet18(nn.Module):
    def __init__(self):
        super(ResNet18, self).__init__()
        # 1 initial convolution layer
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=
            False)
        self.bn1 = nn.BatchNorm2d(64)

        # 4 residual blocks
        self.layer1 = self._make_layer(64, 64, 2, stride=1)
        self.layer2 = self._make_layer(64, 128, 2, stride=2)
        self.layer3 = self._make_layer(128, 256, 2, stride=2)
        self.layer4 = self._make_layer(256, 512, 2, stride=2)
        self.linear = nn.Linear(512, 10)

```

```
    def _make_layer(self, in_planes, out_planes, blocks, stride):
        strides = [stride] + [1]*(blocks-1)
        layers = []
        for stride in strides:
            layers.append(BasicBlock(in_planes, out_planes, stride))
            in_planes = out_planes
        return nn.Sequential(*layers)

    def forward(self, x):
        x = nn.ReLU()(self.bn1(self.conv1(x)))
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = nn.AvgPool2d(4)(x)  # Global average pooling
        x = x.view(x.size(0), -1)
        x = self.linear(x)
        return x
```

2.3 DenseNet

DenseNet is designed to reduce the number of parameters while maintaining high performance by promoting efficient feature utilization. DenseNet for CIFAR-10 typically uses a lower growth rate to ensure the network remains computationally efficient. The growth rate is the number of output feature maps each layer contributes to the next layer. The final layer is adjusted to output 10 classes for CIFAR-10 instead of the 1000 classes in ImageNet.

```
class DenseLayer(nn.Module):
    def __init__(self, in_channels, growth_rate):
        super(DenseLayer, self).__init__()
        self.bn = nn.BatchNorm2d(in_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv = nn.Conv2d(in_channels, growth_rate, kernel_size=3, stride=1,
            padding=1, bias=False)

    def forward(self, x):
        out = self.conv(self.relu(self.bn(x)))
        return torch.cat([x, out], 1)

class DenseBlock(nn.Module):
    def __init__(self, num_layers, in_channels, growth_rate):
        super(DenseBlock, self).__init__()
        layers = []
        for i in range(num_layers):
            layers.append(DenseLayer(in_channels + i * growth_rate, growth_rate))
        self.block = nn.Sequential(*layers)
```

```
19      def forward(self, x):
20          return self.block(x)
21
22  class TransitionLayer(nn.Module):
23      def __init__(self, in_channels, out_channels):
24          super(TransitionLayer, self).__init__()
25          self.bn = nn.BatchNorm2d(in_channels)
26          self.relu = nn.ReLU(inplace=True)
27          self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=1,
                bias=False)
28          self.pool = nn.AvgPool2d(kernel_size=2, stride=2)
29      def forward(self, x):
30          x = self.conv(self.relu(self.bn(x)))
31          return self.pool(x)
32
33  class DenseNet(nn.Module):
34      def __init__(self, growth_rate=16, block_layers=[6, 12, 24, 16], num_classes
            =10):  # growth_rate is also a hyperparameter
35          super(DenseNet, self).__init__()
36          self.growth_rate = growth_rate
37          num_channels = 2 * growth_rate
38
39          self.conv1 = nn.Conv2d(3, num_channels, kernel_size=3, stride=1, padding=1,
                bias=False)
40
41          # DenseBlocks and Transition Layers
42          layers = []
43          for i, num_layers in enumerate(block_layers):
44              layers.append(DenseBlock(num_layers, num_channels, growth_rate))
45              num_channels += num_layers * growth_rate
46              if i != len(block_layers) - 1:
47                  layers.append(TransitionLayer(num_channels, num_channels // 2))
48                  num_channels = num_channels // 2
49
50          self.features = nn.Sequential(*layers)
51
52          self.bn = nn.BatchNorm2d(num_channels)
53          self.relu = nn.ReLU(inplace=True)
54          self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
55          self.fc = nn.Linear(num_channels, num_classes)
56
57      def forward(self, x):
58          x = self.conv1(x)
59          x = self.features(x)
60          x = self.relu(self.bn(x))
61          x = self.avg_pool(x)
62          x = torch.flatten(x, 1)
63          x = self.fc(x)
64          return x
```

3 Experimental Procedure

3.1 Data Augmentation

Data augmentation is a critical step to improve model generalization, especially for RGB datasets like CIFAR-10. It artificially enlarges the training dataset by applying random transformations to the images, which helps the model learn more robust features and prevents overfitting.

```
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomRotation(10),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    # per-channel mean and standard deviation of the CIFAR-10 dataset
    transforms.Normalize((0.4914, 0.4822, 0.4465),
                         (0.2023, 0.1994, 0.2010)),
])
```

3.2 Model Selection

For this experiment, we evaluate three different convolutional neural network (CNN) architectures: **VGG-16, ResNet-18, DenseNet**. These three models showed a huge difference in performance and the ability of generalization. Code implementation of these three models are mentioned in Section 2.

3.3 Parameter Configuration

All three models are trained using the same base configuration to ensure a fair comparison. Below are the hyperparameters used in the experiment:

- **Optimizer and Learning Rate Decay:** Adam and CosineAnnealingLR.
- **Loss Function:** Cross Entropy Loss
- **Other Hyperparameters:** Batch Size = 64, Epochs = 25

3.3.1 ResNet-18

- **Learing Rate:** We experiment with a learning rate range of 0.0001 - 0.01.

3.3.2 VGG-16

- **Learing Rate:** We experiment with a learning rate range of 0.00001 - 0.0001.

3.3.3 DenseNet

- **Learing Rate:** We experiment with a learning rate range of 0.0001 - 0.01.
- **Growth Rate:** 16

4 Results and Analysis

4.1 Results

Each of these models offers different advantages:

- **VGG-16** is straightforward and easy to implement, but it can suffer from high computational costs and overfitting.
- **ResNet-18** is designed to train very deep networks efficiently by using residual connections, making it well-suited for complex datasets like CIFAR-10.
- **DenseNet** is known for being highly parameter-efficient, and its dense connections help improve performance with fewer parameters compared to traditional CNNs.

4.1.1 ResNet-18

ResNet-18 achieved **91.3%** accuracy on CIFAR-10, as shown in Figure 3. We tested a range of learning rates to assess the model's sensitivity and its ability to generalize across different configurations. Notably, ResNet-18 demonstrated robustness across a wide range of learning rates, which reflects its inherent capacity to prevent overfitting through its residual learning method.

The experimental results listed in Table 1 show that ResNet-18's performance is stable across several learning rates, with only slight variations in accuracy.

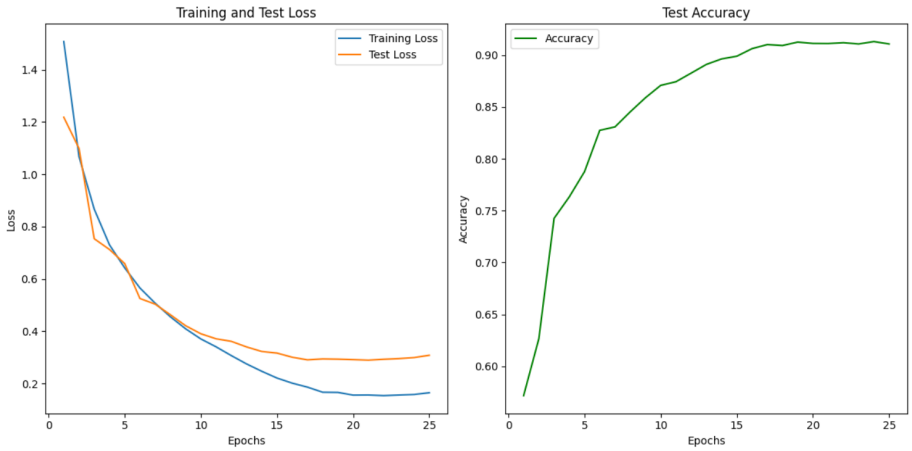

Figure 3 Traing and Testing Losses for ResNet-18

Table 1 ResNet-18 Performances with Different Learning Rate

Learning Rate	Test Accuracy	Tips
1e-4	90.1%	normal
1e-3	91.3%	normal
3e-3	91.1%	normal
5e-3	91.3%	slightly over-fitting
1e-2	90.9%	slightly over-fitting

4.1.2 VGG-16

Training VGG-16 on the CIFAR-10 dataset presented some challenges, particularly in terms of adjusting the learning rate for optimal convergence. Unlike ResNet-18, VGG-16 has a simpler architecture but lacks specific mechanisms to prevent overfitting, such as batch normalization or residual connections. As a result, the model often struggles to learn effectively from the dataset.

During training, VGG-16 showed a delayed learning curve, with the model only starting to show significant improvement around the 6th epoch. This could be attributed to the lack of regularization mechanisms, which may have caused the model to get stuck in local minima early in the training process.

Additionally, many of the learning rates tested caused the model to fail to converge or led to poor performance on the test set, with accuracy results being much worse than those achieved by ResNet-18.The performance at different learning rates is summarized in Table Table 2.

Table 2 VGG-18 Performances with Different Learning Rate

Learning Rate	Test Accuracy	Tips
1e-4	77.8%	normal
3e-4	82.5%	normal
5e-4	82.2%	slightly over-fitting
7e-4	10%	fail to converge

4.1.3 DenseNet

DenseNet achieved **90.5%** accuracy on CIFAR-10, as shown in Figure 4. Its performance is slightly lower than ResNet's, but it exhibits extraordinary generalization and stability, similar to ResNet. DenseNet' s key strength lies in its dense connectivity, where

each layer receives input from all previous layers, promoting feature reuse and improving gradient flow. This leads to better efficiency, requiring fewer parameters for comparable performance.

While DenseNet's test accuracy was not the highest, it showed excellent generalization, and its stability across a wide range of learning rates indicates that the model is less prone to overfitting compared to VGG-16. The performance at different learning rates is summarized in Table Table 3.

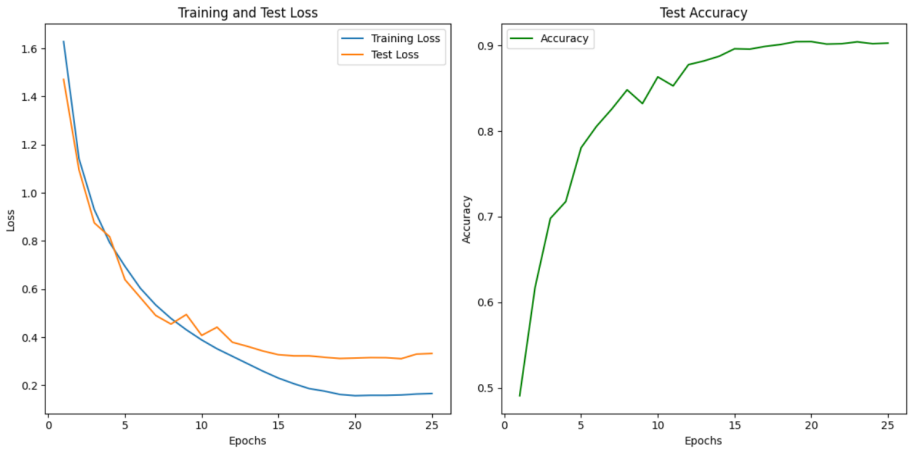

Figure 4 Traing and Testing Losses for DenseNet

Table 3 DenseNet Performances with Different Learning Rate

Learning Rate	Test Accuracy	Tips
1e-4	88.0%	normal
5e-4	90.4%	normal
1e-3	90.5%	normal
3e-3	89.8%	normal
5e-3	90.4%	normal
1e-2	90.2%	slightly over-fitting

4.2 Impact of Data Augmentation

Based on the results obtained from the previous experiments, we chose ResNet-18 as the model to test the impact of data augmentation. Our first experiment was conducted without any data augmentation, using a learning rate of 0.001. Without augmentation, the model showed obvious over-fitting phenomenon, as shown in Figure 5. Furthermore, we attempted to adjust the learning rate in an effort to mitigate this issue. Despite our best efforts, the overfitting persisted, and the model' s performance remained suboptimal.

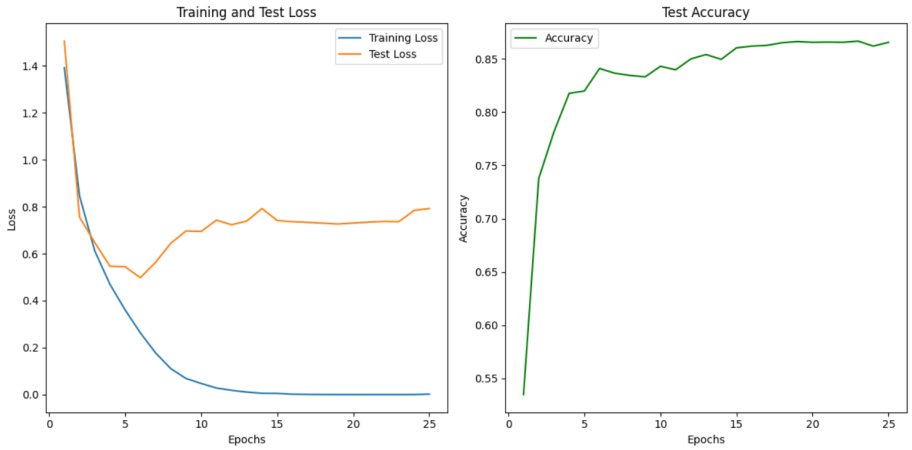

Figure 5 Traing and Testing Losses for ResNet without Augmentation

Data augmentation is critical, especially when working with **RGB** datasets like CIFAR-10, which consists of natural images with variations in color, shape, and orientation. The original CIFAR-10 images are relatively small and might not contain enough information to train a complex model like ResNet-18 without sufficient diversity in the training data.

In the case of CIFAR-10, where the objects are often small and there are many subtle distinctions between categories, data augmentation helps the model not only avoid over-fitting but also achieve higher robustness and accuracy. Without augmentation, the model may struggle to differentiate between these subtle features and fail to generalize to new images.

5 Conclusion

In this experiment, we evaluated three different deep learning models: ResNet-18, VGG-16, and DenseNet, on the CIFAR-10 dataset. ResNet-18 achieved the highest accuracy on CIFAR-10, which is **91.3%**. The model demonstrated excellent robustness across a range of learning rates, highlighting the strength of residual connections in enabling efficient training. VGG-16, on the other hand, struggled with convergence due to its lack of regularization mechanisms like batch normalization or residual connections. DenseNet, while achieving slightly lower accuracy than ResNet-18, demonstrated excellent generalization and stability. It also has fewer parameters and lower computational cost.

In addition to the models, we applied several data augmentation techniques to further improve the performance and generalization capabilities of the models. These augmentations allowed the models to learn more robust features, reducing overfitting and making them more resilient to variations in the data.

References

[1] Karen Simonyan. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[3] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.