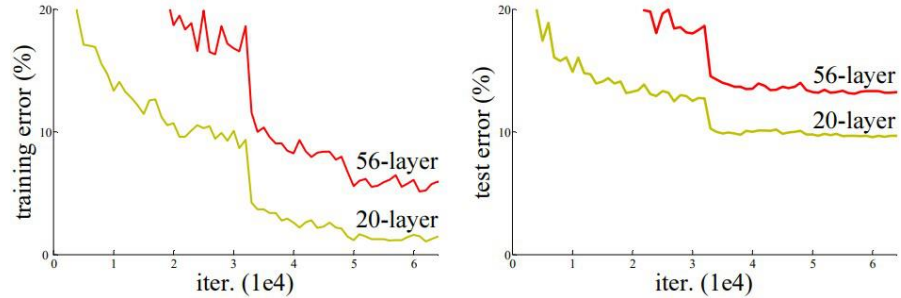


Name :- Ramishka Madhushan

Assignment No :- 01

MODEL	Pro	Cons	High level structure of the algorithm																																								
RESNET (Residual Neural Network)	<ul style="list-style-type: none">• It introduced the concept of residual learning, which helps alleviate the vanishing gradient problem in deep neural networks.• Achieves state-of-the-art accuracy on various computer vision tasks. (Object detection, Image classification)• Allows training of much deeper networks without degradation in performance.	<ul style="list-style-type: none">• It may be prone to overfitting when dealing with small datasets.• It required more computational resources and memory due to its increased depth compared to other models.	<ul style="list-style-type: none">• It utilizes skip connections, or "identity shortcuts," to allow the flow of information from earlier layers directly to later layers.• It consists of residual blocks containing multiple convolutional layers followed by batch normalization and ReLU activation.• The input is passed through multiple residual blocks, and the output is finally fed into fully connected layers for classification. <div><p>The figure consists of two side-by-side line graphs. Both graphs have 'iter. (1e4)' on the x-axis (ranging from 0 to 6) and error percentage on the y-axis (ranging from 0 to 20). The left graph is labeled 'training error (%)' and the right graph is labeled 'test error (%)'. Each graph contains two lines: a yellow line for the '20-layer' model and a red line for the '56-layer' model. In the training error graph, the 20-layer model's error decreases steadily from about 18% to 2%, while the 56-layer model's error starts at 18%, fluctuates, and then drops sharply after 3e4 iterations to about 5%. In the test error graph, the 20-layer model's error decreases from about 18% to 10%, while the 56-layer model's error starts at 18%, fluctuates, and then drops sharply after 3e4 iterations to about 12%.</p><table><caption>Approximate data points from Figure 01</caption><tr><th>Iteration (1e4)</th><th>20-layer Training Error (%)</th><th>56-layer Training Error (%)</th><th>20-layer Test Error (%)</th><th>56-layer Test Error (%)</th></tr><tr><td>0</td><td>18</td><td>18</td><td>18</td><td>18</td></tr><tr><td>1</td><td>15</td><td>18</td><td>15</td><td>18</td></tr><tr><td>2</td><td>12</td><td>18</td><td>12</td><td>18</td></tr><tr><td>3</td><td>10</td><td>18</td><td>10</td><td>18</td></tr><tr><td>4</td><td>8</td><td>10</td><td>8</td><td>12</td></tr><tr><td>5</td><td>5</td><td>5</td><td>5</td><td>12</td></tr><tr><td>6</td><td>2</td><td>5</td><td>2</td><td>12</td></tr></table><p>Figure 01 :- Comparison of 20-layer vs 56-layer architecture</p></div>	Iteration (1e4)	20-layer Training Error (%)	56-layer Training Error (%)	20-layer Test Error (%)	56-layer Test Error (%)	0	18	18	18	18	1	15	18	15	18	2	12	18	12	18	3	10	18	10	18	4	8	10	8	12	5	5	5	5	12	6	2	5	2	12
Iteration (1e4)	20-layer Training Error (%)	56-layer Training Error (%)	20-layer Test Error (%)	56-layer Test Error (%)																																							
0	18	18	18	18																																							
1	15	18	15	18																																							
2	12	18	12	18																																							
3	10	18	10	18																																							
4	8	10	8	12																																							
5	5	5	5	12																																							
6	2	5	2	12																																							

<p>DenseNet (Densely Connected Convolutional Network)</p>	<ul style="list-style-type: none"> • It promotes feature reuse and strengthens feature propagation by connecting each layer to every other layer in a dense manner. • It reduces the number of parameters compared to other architectures, improving parameter efficiency. • It achieves competitive performance with fewer layers, making it computationally efficient. 	<ul style="list-style-type: none"> • DenseNet may suffer from increased memory requirements due to the dense connections, especially in deep networks. • DenseNet can be more susceptible to overfitting than other models, mainly when dealing with small datasets. 	<ul style="list-style-type: none"> • DenseNet is composed of dense blocks, where each block contains multiple convolutional layers. • The input of each layer is the concatenation of the feature maps from all preceding layers. • Transition layers, consisting of convolutional and pooling layers, are used to downsample feature maps and control the growth of dimensions. • The output of the last dense block is fed into global average pooling, followed by fully connected layers for classification. <div data-bbox="1196 464 2119 756"> <p>The diagram illustrates the DenseNet architecture. It starts with an input image, followed by a 'Convolution' layer. This is followed by three 'Dense Block's. Each dense block contains multiple 'Convolution' layers. The input of each block is the concatenation of feature maps from all preceding layers. The diagram shows that pooling reduces feature map sizes, and feature map sizes match within each block. The final output is produced by a 'Pooling' layer followed by a 'Linear' layer.</p> </div> <p>Figure 02 :- DenseNet Architecture</p>
---	---	--	---

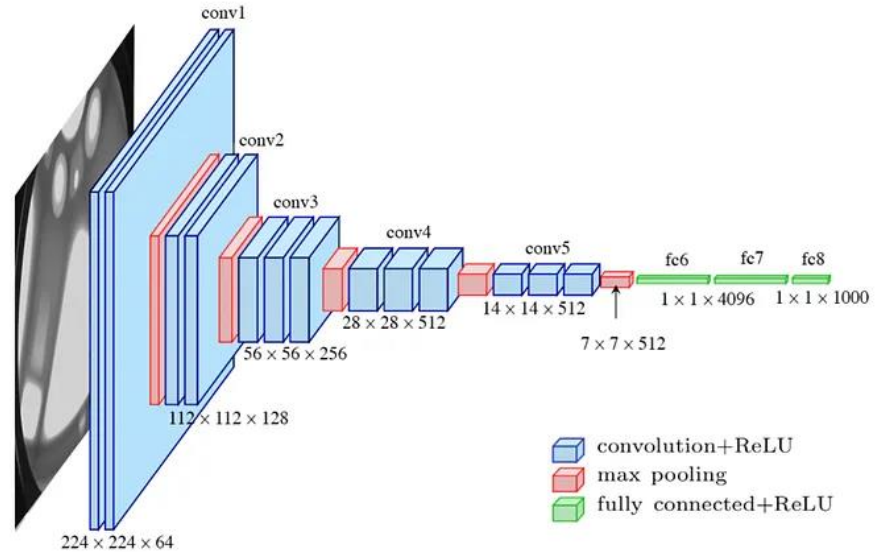
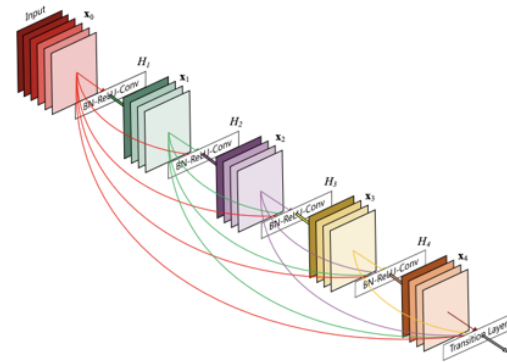
MODEL	Pro	Cons	High level structure of the algorithm
VGG (Visual Geometry Group)	<ul style="list-style-type: none"> VGG has a simple and uniform architecture, making it easy to understand and implement. It has achieved outstanding performance on various image recognition tasks and serves as a strong baseline for CNN models. VGG's design principles have influenced subsequent architectures and research in deep learning. 	<ul style="list-style-type: none"> VGG has many parameters, which leads to increased memory requirements and slower training and inference times. It may struggle with overfitting when dealing with small datasets due to its high capacity. 	<ul style="list-style-type: none"> VGG consists of multiple convolutional layers stacked on top of each other, with smaller 3x3 filters used extensively. Max pooling layers are employed to down sample feature maps. The network ends with fully connected layers for classification. VGG is characterized by its depth, with variations like VGG16 and VGG19 having 16 and 19 layers, respectively.  <p>The diagram illustrates the VGG-Net architecture. It starts with an input image of size $224 \times 224 \times 64$. This is followed by a series of convolutional layers (conv1 to conv5) and max pooling layers. The dimensions of the feature maps are shown at each stage: $112 \times 112 \times 128$ after conv1, $56 \times 56 \times 256$ after conv2, $28 \times 28 \times 512$ after conv3, $14 \times 14 \times 512$ after conv4, and $7 \times 7 \times 512$ after conv5. The final layers are fully connected layers (fc6, fc7, fc8) with dimensions $1 \times 1 \times 4096$, $1 \times 1 \times 4096$, and $1 \times 1 \times 1000$ respectively. A legend indicates that blue blocks represent convolution+ReLU, red blocks represent max pooling, and green blocks represent fully connected+ReLU.</p>

Figure 03 :- VGG-Net Architecture

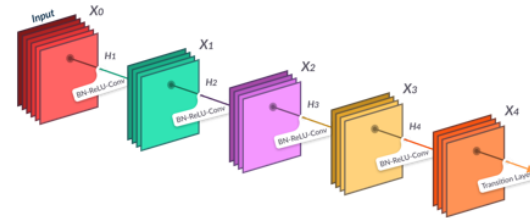
<p>Inception (GoogLeNet / Inception-v3)</p>	<ul style="list-style-type: none"> • Inception incorporates the use of "Inception modules," which employ multiple filter sizes in parallel, capturing different scales of information. • It achieves excellent performance with fewer parameters compared to other architectures. • Inception models are known for their computational efficiency and are often used in resource-constrained environments. 	<ul style="list-style-type: none"> • Inception networks can be challenging to train due to the complexity of their architecture. • They require careful tuning of hyperparameters to achieve optimal performance. 	<ul style="list-style-type: none"> • Inception models consist of a chain of Inception modules, where each module has parallel convolutional branches with different filter sizes. • These branches are combined by concatenation of their output feature maps. • Inception networks also incorporate auxiliary classifiers at intermediate layers to provide additional regularization during training. • The final output is obtained through global average pooling and fully connected layers. <div data-bbox="1254 526 2038 813"> <p>The diagram illustrates the VGG-Net architecture. It starts with a 'Stack of 64 feature maps' with dimensions 32x32x64. This stack is fed into three parallel processing branches: <ul style="list-style-type: none"> Convolution 1x1x16: Produces a 32x32x16 feature map (orange). Convolution 3x3x32: Produces a 32x32x32 feature map (yellow). Max pooling 3x3, padding = same, stride = 1: Produces a 32x32x64 feature map (grey). The outputs of these three branches are concatenated to form a single 32x32x112 feature map (112 = 16+32+64). This concatenated map is then processed by a final convolution layer to produce the final 32x32x64 output feature map. </p> </div>
--	---	---	--

Figure 04 :- VGG-Net Architecture



DenseNet Structure

$$a^{[l]} = g([a^{[0]}, a^{[1]}, a^{[2]}, \dots, a^{[l-1]}])$$



ResNet Structure

$$a^{[l]} = g(z^{[l+1]} + a^{[l]})$$

Figure 05 :- DenseNet vs ResNet Structure