Artificial Neural Network

Md. Hasnain Ali
Student Id: 2010976153

May 2025

1 Q1: Describing the effect of different types of activation function of your chosen 10 CNNs.

Answer: Activation functions play a critical role in deep learning models by introducing non-linearity into the network, allowing it to learn complex mappings between inputs and outputs. In Convolutional Neural Networks (CNNs), the choice of activation function significantly affects performance, training stability, convergence speed, and generalization capability.

This document analyzes the effects of various activation functions on ten CNN architectures, including Xception, ResNet50V2, VGG16, VGG19, DenseNet121, MobileNet, MobileNetV2, InceptionV3, and ResNet101V2. The evaluation is based on model accuracy, loss, and training time, offering insight into which functions are most effective in different architectural contexts.

1.1 Softmax

Softmax, while essential in the final output layer of classifiers, performs poorly when used as a hidden layer activation. All models—including Xception and ResNet50V2—exhibited extremely low accuracy, averaging around 5%, with evaluation losses nearing 3.0.

This is because softmax normalizes activations across a layer, turning outputs into a probability distribution. In hidden layers, this disrupts gradient propagation and hinders learning. Thus, using softmax outside of output layers should be strictly avoided.

1.2 Sigmoid

The sigmoid activation function maps inputs to the interval $(0, 1)$ and introduces smooth non-linearity. It performed moderately well across several models, with the highest accuracies observed in VGG19 and DenseNet121 (both at 78.5%). However, its performance dropped significantly in deeper models like InceptionV3, where accuracy fell to 57.4%.

One key drawback of sigmoid is the vanishing gradient problem. As the depth of the network increases, gradients may become too small to contribute meaningful updates, slowing or even halting training. Additionally, sigmoid-based models required longer training times; for example, Xception took 0.191 seconds with sigmoid compared to 0.114 seconds with GELU.

1.3 ReLU (Rectified Linear Unit)

ReLU remains one of the most commonly used activation functions in CNNs due to its simplicity and effectiveness. It sets all negative values to zero, thus introducing sparsity and avoiding the vanishing gradient problem.

Its performance was reliable, though not always the best. MobileNet and VGG16 achieved strong results with ReLU, scoring 72.3% and 76.05% respectively. However, ResNet50V2, a deeper model, saw better accuracy with sigmoid (70.15% vs. ReLU's 70.0%).

One issue with ReLU is the "dying ReLU" phenomenon, where neurons output zero for all inputs and cease to learn. Despite this, ReLU remains a solid baseline for most architectures.

1.4 Tanh

Tanh maps inputs to the range $[-1, 1]$ and produces outputs centered around zero, which can benefit gradient flow early in training. However, it generally underperformed compared to ReLU and sigmoid.

In practice, the Xception model achieved 63.15% accuracy with tanh, while MobileNetV2 dropped as low as 41.35%. Tanh suffers from vanishing gradients similar to sigmoid but with a more centered output. Although it can stabilize training in shallow networks, it is rarely the best choice in modern CNNs.

1.5 Linear

Linear activations do not introduce any non-linearity, making them unsuitable for deep learning. In our experiments, models using linear activations performed only marginally better than softmax, with ResNet50V2 achieving 63.35% accuracy but suffering a very high evaluation loss of 30.83.

Since learning in CNNs depends on modeling complex non-linear relationships, the linear activation fails to enhance model expressiveness, severely limiting its utility in hidden layers.

1.6 GELU (Gaussian Error Linear Unit)

GELU is a smooth approximation of the ReLU function and combines the advantages of both sigmoid and ReLU. It adapts the activation based on the input value's magnitude, resulting in improved gradient flow and optimization.

This function achieved top performance across several models. DenseNet121 reached 81.35% accuracy with GELU, the highest among all tested combinations. Xception also performed well, attaining 71.15%. GELU also provided faster convergence, with Xception training in just 0.114 seconds—faster than its sigmoid and tanh counterparts.

1.7 CELU and ELU

Both CELU (Continuously Differentiable ELU) and ELU (Exponential Linear Unit) aim to address the drawbacks of ReLU by allowing negative outputs and maintaining smooth gradients. These functions demonstrated mixed results.

For instance, ELU helped VGG19 achieve 72.9% accuracy and MobileNet 67.85%. CELU showed decent results in MobileNetV2 (66.55%) and ResNet101V2 (67.4%), but not as consistently strong as GELU or ReLU.

Although these functions can mitigate the dead neuron problem, their higher computational cost and inconsistent advantages make them a secondary choice unless specific benefits are observed in a given architecture.

1.8 SILU (Sigmoid Linear Unit / Swish)

SILU, also known as Swish, combines sigmoid's smoothness with ReLU's strengths. It performed competitively across many models, with MobileNetV2 reaching 72.2% and Xception 71.7%.

This function enhances gradient propagation, particularly in deeper networks, making it suitable for advanced architectures. Its consistent high performance makes it a strong alternative to GELU in modern CNN designs.

1.9 Conclusion

The performance of activation functions varies notably across CNN architectures. GELU and SILU emerge as top choices, especially for deep and dense models, due to their superior gradient handling and optimization dynamics. ReLU remains a solid default, especially when computational efficiency is critical.

Sigmoid and tanh are acceptable in shallower networks but are less effective in deep CNNs due to gradient issues. Softmax and linear activations should be strictly avoided in hidden layers, as they fail to introduce the non-linearity required for deep learning.

Ultimately, selecting the right activation function involves balancing model depth,

Code: https://shorturl.at/8C5mi

Table 1: Xception - Model Performance with different activation functions.

Model Name	Activation Function	Model Size (MB)	Training Time (Sec)	Evaluation Time (Sec)	Evaluation Loss	Evaluation Accuracy
Xception	softmax	431.92	0.15	0.04	3.00	0.05
Xception	sigmoid	431.92	0.19	0.06	1.00	0.68
Xception	relu	431.92	0.18	0.06	0.98	0.71
Xception	tanh	431.92	0.19	0.06	1.26	0.63
Xception	linear	431.92	0.18	0.06	16.23	0.65
Xception	gelu	431.92	0.11	0.03	0.96	0.71
Xception	celu	431.92	0.11	0.03	1.01	0.70
Xception	elu	431.92	0.11	0.03	1.31	0.62
Xception	silu	431.92	0.11	0.03	1.03	0.72

Table 2: VGG16 - Model Performance with different activation functions.

Model Name	Activation Function	Model Size (MB)	Training Time (Sec)	Evaluation Time (Sec)	Evaluation Loss	Evaluation Accuracy
VGG16	softmax	152.48	0.07	0.02	3.00	0.09
VGG16	sigmoid	152.48	0.06	0.02	0.72	0.77
VGG16	relu	152.48	0.07	0.02	0.90	0.76
VGG16	tanh	152.48	0.06	0.02	1.07	0.70
VGG16	linear	152.48	0.06	0.02	23.67	0.67
VGG16	gelu	152.48	0.06	0.01	0.88	0.74
VGG16	celu	152.48	0.06	0.01	1.03	0.71
VGG16	elu	152.48	0.06	0.01	0.98	0.73
VGG16	silu	152.48	0.06	0.01	0.84	0.76

Table 3: VGG19 - Model Performance with different activation functions.

Model Name	Activation Function	Model Size (MB)	Training Time (Sec)	Evaluation Time (Sec)	Evaluation Loss	Evaluation Accuracy
VGG19	softmax	172.73	0.07	0.02	3.00	0.05
VGG19	sigmoid	172.73	0.07	0.02	0.70	0.79
VGG19	relu	172.73	0.07	0.02	0.89	0.76
VGG19	tanh	172.73	0.07	0.02	1.02	0.70
VGG19	linear	172.73	0.07	0.02	21.42	0.69
VGG19	gelu	172.73	0.07	0.01	0.86	0.75
VGG19	celu	172.73	0.06	0.01	0.95	0.76
VGG19	elu	172.73	0.06	0.01	1.07	0.73
VGG19	silu	172.73	0.07	0.01	0.91	0.75

Table 4: ResNet50V2 - Model Performance with different activation functions.

Model Name	Activation Function	Model Size (MB)	Training Time (Sec)	Evaluation Time (Sec)	Evaluation Loss	Evaluation Accuracy
ResNet50V2	softmax	442.24	0.17	0.05	3.00	0.05
ResNet50V2	sigmoid	442.24	0.21	0.06	0.89	0.73
ResNet50V2	relu	442.24	0.20	0.06	1.07	0.70
ResNet50V2	tanh	442.24	0.21	0.06	1.12	0.64
ResNet50V2	linear	442.24	0.20	0.06	30.83	0.63
ResNet50V2	gelu	442.24	0.13	0.03	1.08	0.68
ResNet50V2	celu	442.24	0.13	0.03	1.14	0.69
ResNet50V2	elu	442.24	0.13	0.03	1.08	0.69
ResNet50V2	silu	442.24	0.13	0.03	1.04	0.69

Table 5: ResNet101V2 - Model Performance with different activation functions.

Model Name	Activation Function	Model Size (MB)	Training Time (Sec)	Evaluation Time (Sec)	Evaluation Loss	Evaluation Accuracy
ResNet101V2	softmax	514.95	0.24	0.06	3.00	0.05
ResNet101V2	sigmoid	514.95	0.28	0.08	0.92	0.71
ResNet101V2	relu	514.95	0.27	0.07	1.10	0.69
ResNet101V2	tanh	514.95	0.28	0.08	1.08	0.69
ResNet101V2	linear	514.95	0.26	0.07	23.95	0.62
ResNet101V2	gelu	514.95	0.20	0.04	1.22	0.66
ResNet101V2	celu	514.95	0.21	0.04	1.15	0.67
ResNet101V2	elu	514.95	0.20	0.04	1.27	0.65
ResNet101V2	silu	514.95	0.20	0.04	1.03	0.71

Table 6: MobileNet - Model Performance with different activation functions.

Model Name	Activation Function	Model Size (MB)	Training Time (Sec)	Evaluation Time (Sec)	Evaluation Loss	Evaluation Accuracy
MobileNet	softmax	140.66	0.11	0.03	3.00	0.06
MobileNet	sigmoid	140.66	0.16	0.06	0.99	0.70
MobileNet	relu	140.66	0.15	0.05	0.91	0.72
MobileNet	tanh	140.66	0.16	0.05	1.01	0.70
MobileNet	linear	140.66	0.14	0.05	8.41	0.60
MobileNet	gelu	140.66	0.08	0.02	0.90	0.72
MobileNet	celu	140.66	0.07	0.02	1.06	0.68
MobileNet	elu	140.66	0.08	0.02	1.18	0.68
MobileNet	silu	140.66	0.07	0.02	0.92	0.72

Table 7: MobileNetV2 - Model Performance with different activation functions.

Model Name	Activation Function	Model Size (MB)	Training Time (Sec)	Evaluation Time (Sec)	Evaluation Loss	Evaluation Accuracy
MobileNetV2	softmax	252.96	0.13	0.04	3.00	0.05
MobileNetV2	sigmoid	252.96	0.12	0.03	1.04	0.68
MobileNetV2	relu	252.96	0.12	0.03	0.97	0.72
MobileNetV2	tanh	252.96	0.12	0.03	1.93	0.41
MobileNetV2	linear	252.96	0.11	0.03	19.69	0.68
MobileNetV2	gelu	252.96	0.11	0.03	0.95	0.70
MobileNetV2	celu	252.96	0.11	0.03	1.07	0.67
MobileNetV2	elu	252.96	0.11	0.03	1.04	0.69
MobileNetV2	silu	252.96	0.12	0.03	0.92	0.72

Table 8: InceptionV3 - Model Performance with different activation functions.

Model Name	Activation Function	Model Size (MB)	Training Time (Sec)	Evaluation Time (Sec)	Evaluation Loss	Evaluation Accuracy
InceptionV3	softmax	179.51	0.21	0.07	3.00	0.05
InceptionV3	sigmoid	179.51	0.17	0.05	1.35	0.57
InceptionV3	relu	179.51	0.17	0.05	1.24	0.62
InceptionV3	tanh	179.51	0.17	0.05	1.55	0.56
InceptionV3	linear	179.51	0.17	0.05	4.20	0.50
InceptionV3	gelu	179.51	0.17	0.04	1.26	0.61
InceptionV3	celu	179.51	0.17	0.05	1.52	0.57
InceptionV3	elu	179.51	0.17	0.04	1.54	0.54
InceptionV3	silu	179.51	0.17	0.04	1.27	0.62

Table 9: InceptionResNetV2 - Model Performance with different activation functions.

Model Name	Activation Function	Model Size (MB)	Training Time (Sec)	Evaluation Time (Sec)	Evaluation Loss	Evaluation Accuracy
InceptionResNetV2	softmax	295.62	0.39	0.11	3.00	0.05
InceptionResNetV2	sigmoid	295.62	0.33	0.08	1.10	0.65
InceptionResNetV2	relu	295.62	0.32	0.08	0.95	0.70
InceptionResNetV2	tanh	295.62	0.34	0.08	1.09	0.66
InceptionResNetV2	linear	295.62	0.32	0.08	5.95	0.61
InceptionResNetV2	gelu	295.62	0.32	0.07	0.98	0.69
InceptionResNetV2	celu	295.62	0.32	0.07	1.05	0.66
InceptionResNetV2	elu	295.62	0.31	0.07	1.09	0.67
InceptionResNetV2	silu	295.62	0.34	0.07	1.06	0.68

Table 10: DenseNet121 - Model Performance with different activation functions.

Model Name	Activation Function	Model Size (MB)	Training Time (Sec)	Evaluation Time (Sec)	Evaluation Loss	Evaluation Accuracy
DenseNet121	softmax	155.19	0.35	0.13	3.00	0.05
DenseNet121	sigmoid	155.19	0.31	0.09	0.70	0.79
DenseNet121	relu	155.19	0.31	0.09	0.76	0.78
DenseNet121	tanh	155.19	0.31	0.09	0.89	0.73
DenseNet121	linear	155.19	0.32	0.09	15.38	0.73
DenseNet121	gelu	155.19	0.31	0.09	0.65	0.81
DenseNet121	celu	155.19	0.31	0.09	0.79	0.76
DenseNet121	elu	155.19	0.30	0.09	0.86	0.76
DenseNet121	silu	155.19	0.31	0.09	0.70	0.79

2 Q2: Mention at least three CNNs which use regular kernel, deformable kernel, dialated kernel, depthwise separable kernel, modified depthwise-separable kernel, pointwise kernel.

Answer: In convolutional neural networks (CNNs), kernels are crucial components used to extract features from input data. Different types of kernels are designed to optimize learning capacity, computational efficiency, and adaptability to diverse data patterns. Below, we describe several important types of kernels used in modern deep learning systems, and Table 11 summarizes the architectures that use these kernels.

Regular Kernel: A regular kernel is the standard filter in CNNs, typically of size 3×3 or 5×5, which slides over the input feature map performing dot products. It captures local spatial patterns such as edges, corners, and textures. Regular kernels are fundamental to early CNN architectures like AlexNet and VGG, providing strong feature extraction capabilities at low computational complexity.

Deformable Kernel: A deformable kernel enhances standard convolution by introducing learnable offsets to the sampling locations, allowing the receptive field to adapt dynamically to object shapes and spatial deformations. This flexibility helps model geometric variations in data, making it especially effective in tasks like object detection and instance segmentation where shape and pose vary significantly.

Dilated Kernel: A dilated (or atrous) kernel inserts fixed gaps between the kernel weights, expanding the receptive field without increasing the number of parameters. This allows the network to aggregate multi-scale contextual information while maintaining the same resolution, which is particularly valuable in semantic segmentation models like DeepLab that require large context awareness.

Depthwise Separable Kernel: A depthwise separable kernel decomposes a standard convolution into two operations: a depthwise convolution, which applies a single filter per input channel, and a pointwise (1×1) convolution, which combines the outputs across channels. This greatly reduces the computational cost and number of parameters, making it ideal for lightweight models like MobileNet used in mobile and embedded applications.

Modified Depthwise-Separable Kernel: A modified depthwise separable kernel builds on the standard version by incorporating enhancements like grouped convolutions, skip connections, or attention mechanisms to increase representational power. These improvements offer a better trade-off between accuracy and efficiency, and are used in advanced models such as MobileNetV2 and EfficientNet, especially in resource-constrained environments.

Pointwise Kernel: A pointwise kernel is a 1×1 convolution filter used to perform channel-wise transformations without altering spatial dimensions. It enables dimensionality reduction or expansion and facilitates interactions between channels. Pointwise convolutions are critical in network components like bottleneck layers and are widely employed alongside depthwise convolutions in efficient CNN architectures.

Table 11: CNN Models and Associated Kernel Types

Kernel Type	**Example CNN Architectures**
Regular Kernel	**VGG16**, **VGG19**, **ResNet50V2**
Deformable Kernel	**Deformable ConvNet**, **DCN v2**, **YOLACT++**
Dilated (Atrous) Kernel	**DeepLabV3**, **DRN (Dilated ResNet)**, **InceptionResNetV2**
Depthwise Separable Kernel	**MobileNet**, **MobileNetV2**, **Xception**
Modified Depthwise-Separable Kernel	**EfficientNet**, **MixNet**, **MobileNetV3**
Pointwise Kernel	**InceptionV3**, **ResNet101V2**, **Xception**

3 Q3: Describe different layers' feature map of your chosen CNN.

We have chosen MobileNet. The MobileNet architecture leverages depthwise separable convolutions to reduce computation and model size. Below is a detailed explanation of each layer and the corresponding feature map dimensions as shown in the architecture diagram (Figure 1).

- **Input Layer:** The model takes an input image of size $224 \times 224 \times 3$, representing height, width, and RGB channels.
- **Initial Convolution (3x3 Conv):** A 3×3 standard convolution with stride 2 reduces the spatial resolution to 112×112 and increases the depth to 32, resulting in a feature map of size $112 \times 112 \times 32$.
- **Depthwise Separable Convolution Block 1 ($\times$2):** This block applies two depthwise separable convolutions (each comprising a 3×3 depthwise convolution followed by a 1×1 pointwise convolution). The first convolution reduces the resolution to 56×56 and increases the channel depth to 64. Thus, output size becomes $56 \times 56 \times 64$.
- **Depthwise Separable Convolution Block 2 ($\times$2):** Another pair of depthwise separable convolutions is applied. The feature map size is halved again to 28×28, and the depth increases to 128, yielding $28 \times 28 \times 128$.
- **Depthwise Separable Convolution Block 3 ($\times$6):** Six consecutive depthwise separable convolutions are used. The resolution is reduced to 14×14, and the channel depth is expanded to 256, resulting in $14 \times 14 \times 256$.

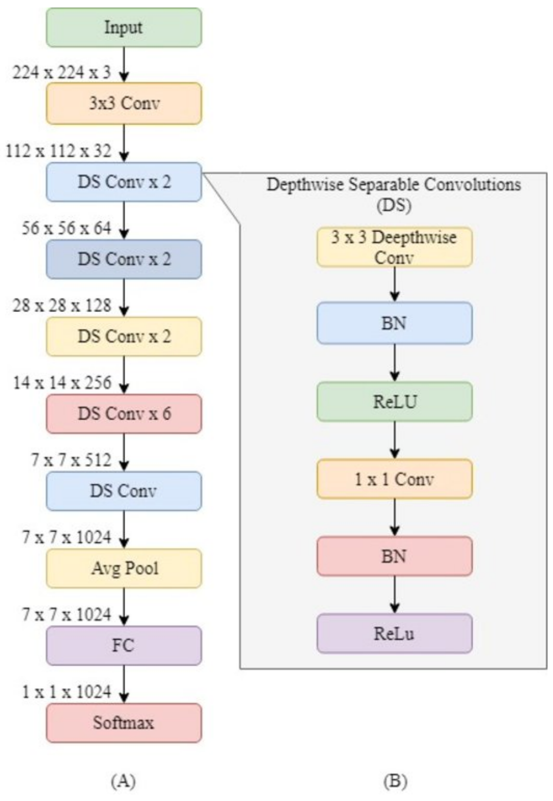

Figure 1: Architecture of MobileNet model.

- **Depthwise Separable Convolution Block 4:** A single depthwise separable convolution reduces the resolution to 7×7 and increases the depth to 512, producing $7 \times 7 \times 512$ feature maps.
- **Final Convolution Block:** The last depthwise separable convolution increases the depth to 1024 while maintaining the spatial dimensions, giving $7 \times 7 \times 1024$.
- **Average Pooling:** A global average pooling layer reduces the spatial size from 7×7 to 1×1, resulting in a feature map of size $1 \times 1 \times 1024$ by computing the average of each channel.
- **Fully Connected Layer (FC):** This layer processes the $1 \times 1 \times 1024$ feature vector for classification, keeping the dimensions the same.
- **Softmax Layer:** Finally, the output passes through a softmax layer to generate class probabilities from the $1 \times 1 \times 1024$ logits, typically mapped to the number of classes (e.g., 1000 for ImageNet).

Each depthwise separable convolution block (as shown in part B of the diagram) consists of a 3×3 depthwise convolution followed by batch normalization and ReLU, then a 1×1 pointwise convolution, again followed by batch normalization and ReLU. This factorization helps drastically reduce computation while retaining accuracy, making MobileNet suitable for mobile and embedded systems.