Comparison of Activation Functions in CNNs

Github : KarthikChikki/Comparison-of-Activation-Functions-in-CNNs

1. Introduction

Convolutional neural networks (CNNs) depend critically on activation functions as they provide non-linearity and let the network learn intricate characteristics from input. CNNs would operate like linear models without activation functions, therefore restricting their capacity to detect complex patterns in images. These processes control neural firing and impact the training process, therefore influencing gradient flow, accuracy, and convergence speed. With often utilised choices like Sigmoid, Tanh, and ReLU, the choice of activation function dramatically affects network performance. Optimising CNNs depends on an awareness of their strengths and constraints, which guarantees their efficient learning and generalisation throughout many deep-learning applications.

2. Understanding the Activation Functions

Sigmoid Activation Function:

One of the first activation functions used in neural networks is the sigmoid activation. In mathematical terms the sigmoid activation can be defined as:

$$S(x) = \frac{1}{1 + e^{-x}}$$

This formula results the output in the range of (0, 1). For binary classification problems where probabilities are required, this function is very helpful. The Sigmoid function's primary benefit is its short-range mapping of any input, therefore avoiding extreme values. Still, it has major disadvantages mostly related to the vanishing gradients. The function saturates in response to excessively big or too tiny inputs, producing almost zero gradients. Backpropagation's difficulty to efficiently update weights slows down training. Moreover, Sigmoid outputs are not centred on zero, which results in ineffective weight updates and slowing down of deep network convergence. These restrictions mean that deep CNNs nowadays seldom ever utilise it.

Tanh Activation Function:

The Tanh (hyperbolic tangent) activation function is another commonly used activation function in the CNN models. In mathematical terms the tanh activation can be defined as:

$$f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$

Where the value of 'x' represents the input value, and the value of 'e' represents the base of the natural logarithm.

Tanh is zero-cantered and helps balance gradient updates as it generates outputs in the range (-1,1) unlike Sigmoid. Tanh provides faster training results from this characteristic than from Sigmoid. Tanh also offers better gradients for values around zero, hence enhancing learning effectiveness. For really big or small input values, tanh still suffers from the vanishing gradient issue, however. Training in deeper networks is ineffective when inputs reach the saturation areas of the function because the gradient approaches zero. Though it performs better than Sigmoid in most circumstances, its computational cost and gradient restrictions make it not the recommended option for contemporary CNN designs.

ReLu Activation Function:

Particularly in CNNs, the ReLU (Rectified Linear Unit) activation function has grown to be the most popular activation function. In mathematical terms the ReLu activation can be defined as:

$$f(x) = max\ (0, x)$$

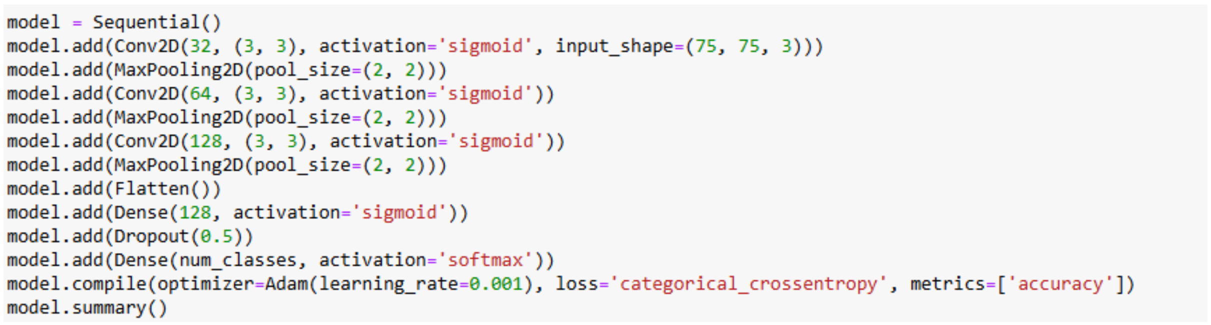
The ReLu outputs zero for negative inputs and the input itself for positive values. Among the many advantages that that this simple function provides are computing efficiency and resolving of the vanishing gradient issue for positive inputs. ReLU accelerates training and lets deeper networks converge quicker as it does not call for complex exponentiation. ReLU may also suffer from the dying ReLU issue, in which case neurones cease learning and produce zero for all inputs. This occurs when weights get negative and zero gradients cause them not to recover. By tolerating tiny gradients for negative inputs, variants like Leaky ReLU and Parametric ReLU have been created to handle this problem. ReLU's efficiency and great performance help it to be the most often used method for CNNs despite this restriction.

3. Implementing a CNN to Compare Activation Functions

Three models were developed utilising Sigmoid, Tanh, and ReLU activation functions in order to evaluate the efficacy of many activation functions in Convolutional Neural Networks (CNNs). The Satellite Image Classification dataset acquired from Kaggle makes up the basis for this comparison. It comprises pictures arranged into four categories: water, cloud, desert, green area. Comprising 5,131 pre-processed photos, the dataset was separated into training and testing sets for performance assessment.

Data Preprocessing: Different preprocessing steps were performed before CNN training. Corresponding class labels were applied when the paths of image files were gathered. Then LabelEncoder converted these labels into numerical numbers. Each image was then scaled to pixel value between 0 and 1 and resized to 75x75 pixels. To guarantee efficient assessment of model performance, the dataset was then divided in 80% training and 20% testing.

CNN Model with Sigmoid Activation:



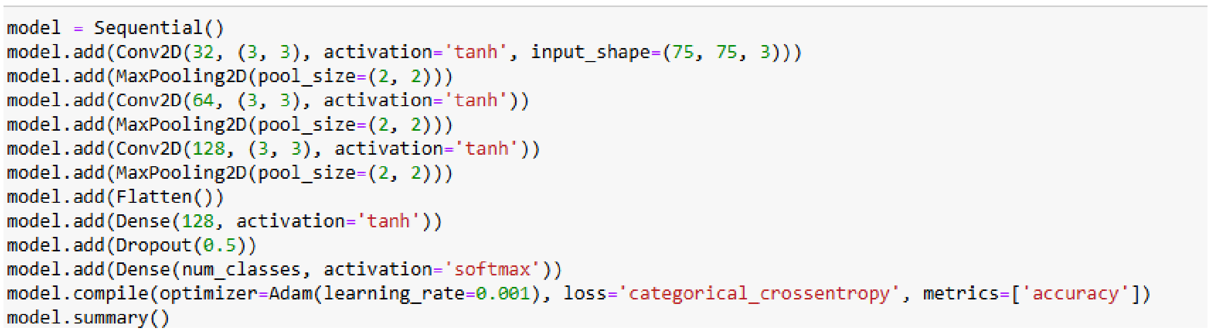
```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='sigmoid', input_shape=(75, 75, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='sigmoid'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='sigmoid'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='sigmoid'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

The initial CNN model built all layers using a sigmoid activation function. Multiple convolutional layers for feature extraction made up the architecture; Max Pooling layers then helped to reduce spatial dimensions. The architecture comprised:

- Conv2D layer with 32 filters and a (3,3) kernel size, followed by MaxPooling (2,2); Activation = Sigmoid

- Conv2D layer with 64 filters and a (3,3) kernel size, followed by MaxPooling (2,2); Activation = Sigmoid
- Conv2D layer with 128 filters and a (3,3) kernel size, followed by MaxPooling (2,2); Activation = Sigmoid
- Flatten layer to convert the feature maps into a one-dimensional vector
- Dense layer with 128 neurons and Sigmoid activation
- Dropout layer (0.5) to reduce overfitting
- Dense output layer with four neurons and SoftMax activation for multi-class classification

CNN Model with Tanh Activation:

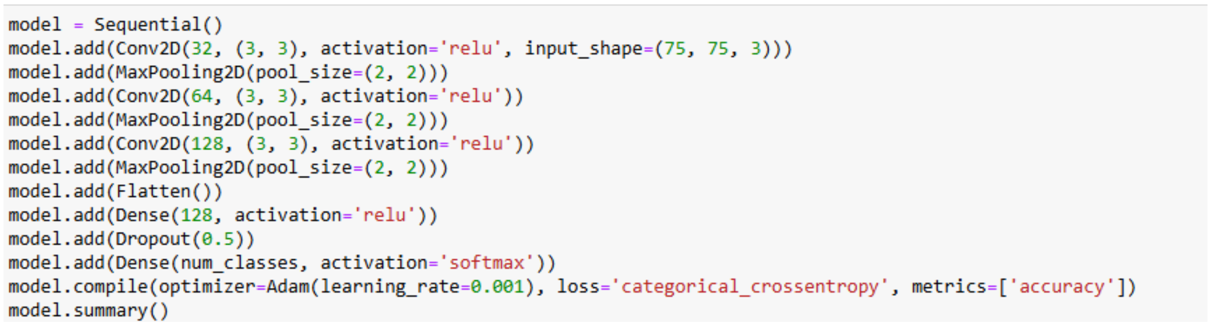


```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='tanh', input_shape=(75, 75, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='tanh'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='tanh'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='tanh'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

The next CNN model built all conv layers using a Tanh activation function. Multiple convolutional layers for feature extraction made up the architecture; Max Pooling layers then helped to reduce spatial dimensions. The architecture comprised:

- Conv2D layer with 32 filters and a (3,3) kernel size, followed by MaxPooling (2,2); Activation = Tanh
- Conv2D layer with 64 filters and a (3,3) kernel size, followed by MaxPooling (2,2); Activation = Tanh
- Conv2D layer with 128 filters and a (3,3) kernel size, followed by MaxPooling (2,2); Activation = Tanh
- Flatten layer to convert feature maps into a one-dimensional vector
- Dense layer with 128 neurons and Tanh activation
- Dropout layer (0.5) to prevent overfitting
- Dense output layer with four neurons and SoftMax activation for multi-class classification

CNN Model with ReLu Activation:



```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(75, 75, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

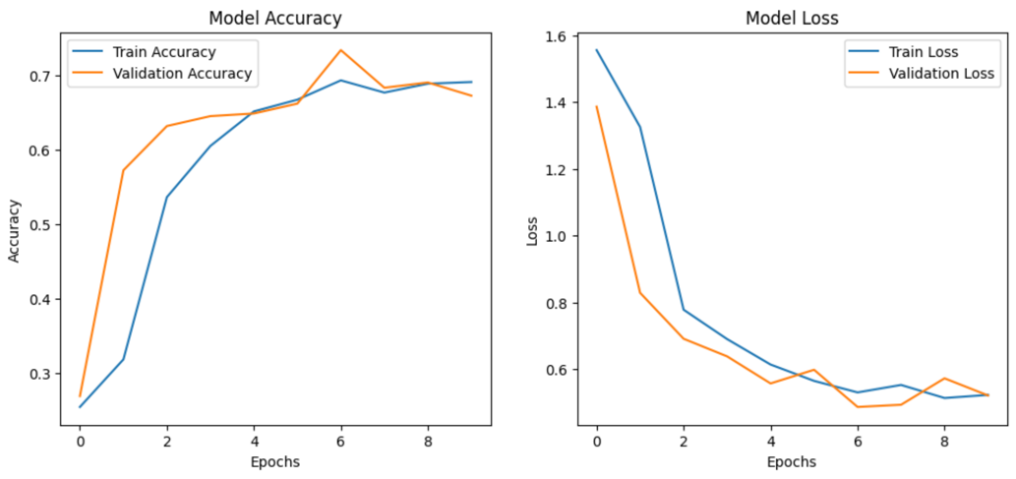
The next CNN model built all conv layers using a ReLu activation function. Multiple convolutional layers for feature extraction made up the architecture; Max Pooling layers then helped to reduce spatial dimensions. The architecture comprised:

- Conv2D layer with 32 filters and a (3,3) kernel size, followed by MaxPooling (2,2); Activation = ReLu
- Conv2D layer with 64 filters and a (3,3) kernel size, followed by MaxPooling (2,2); Activation = ReLu
- Conv2D layer with 128 filters and a (3,3) kernel size, followed by MaxPooling (2,2); Activation = ReLu
- Flatten layer to convert feature maps into a one-dimensional vector
- Dense layer with 128 neurons and Tanh activation
- Dropout layer (0.5) to prevent overfitting
- Dense output layer with four neurons and SoftMax activation for multi-class classification

4. Comparison of Results

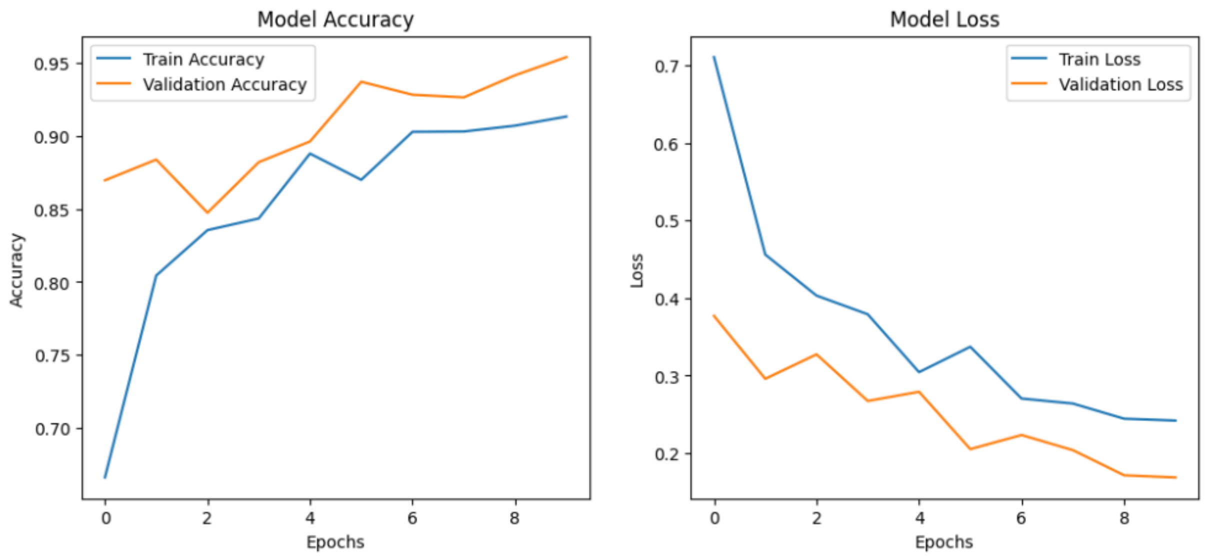
CNN using Sigmoid Activation:

CNN performance under Sigmoid activation showed restrictions in learning accuracy and efficiency. In the first epoch the model began with a poor training accuracy of 25.59% and validation accuracy of 26.89%. After 10 epochs, the model progressed rapidly and attained final training accuracy of 70.04% and validation accuracy of 67.26%. Starting at 1.7680 and working down to 0.5233 for training loss, the loss values similarly show steady convergence; validation loss varied somewhat and ended at 0.5216. Particularly in deeper layers, which limited effective weight updates, the sluggish training and reduced overall accuracy point to Sigmoid activation producing vanishing gradient problems. The model showed variations in validation accuracy throughout epochs, suggesting possible problems effectively capturing complicated characteristics.



Furthermore, the plateau in the validation accuracy in the latter epochs indicated that the model suffered with generalising. Sigmoid activation most certainly produced problems in spreading gradients over many layers, therefore affecting learning by compressing data to a limited range between 0 and 1.

CNN using Tanh Activation:

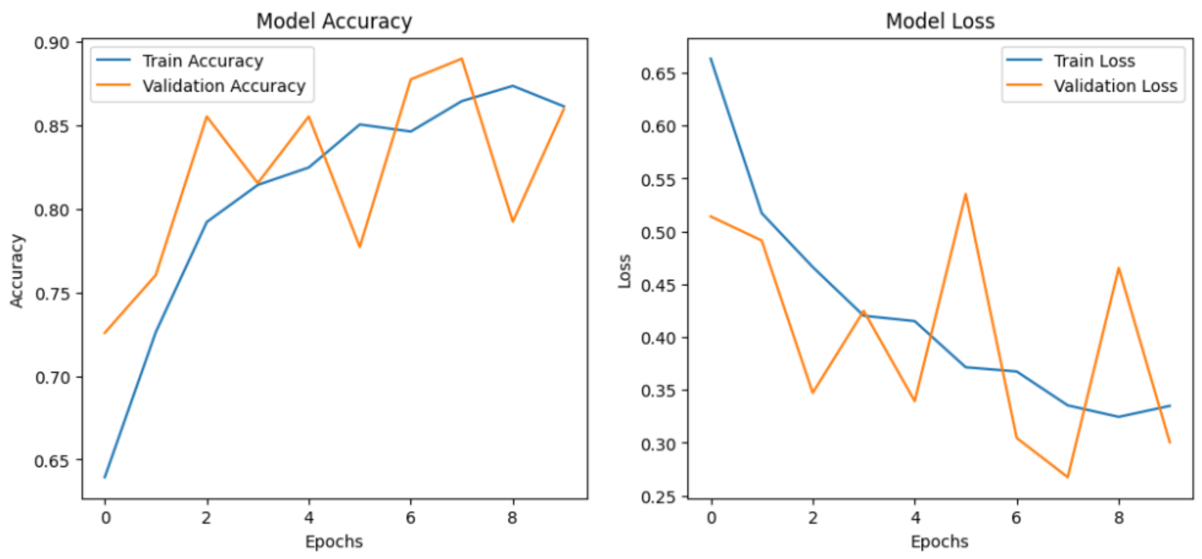


Comparatively to the Sigmoid-based model, the model using Tanh activation showed a far superior learning path. Starting at 57.61%, the training accuracy indicated that the network more rapidly caught significant patterns than the beginning accuracy of the Sigmoid model.

Reflecting great generalisation, the ultimate training accuracy came at 91.03% while validation accuracy peaked at 95.39%. Furthermore, more successful in this approach was loss minimisation. While validation loss reduced consistently to 0.1689, training loss dropped from 0.9334 in the first epoch to 0.2547, hence indicating effective convergence. The always high validation accuracy implies that Tanh activation better than Sigmoid handled both positive and negative data, hence reducing vanishing gradient problems.

One major benefit of Tanh is its capacity to centre activations around zero, hence accelerating convergence. This quality probably helped it to have better stability and speed of learning than Sigmoid. With little variation, the model maintained great validation accuracy throughout many epochs, therefore proving dependability in generalisation.

CNN using ReLU Activation:



With an initial training accuracy of 55.20%, which was higher than the Sigmoid model but somewhat lower than the Tanh model, the CNN using ReLU activation exhibited interesting performance. Although validation accuracy settled at 85.98%, lower than the Tanh-based model but higher than the Sigmoid-based model, the ultimate training accuracy came out to 85.16%. Strong learning performance was also indicated by training loss declining from 0.8367 to 0.3500 and validation loss lowering to 0.3004. Some variations in validation accuracy, however, point to sporadic overfitting or instability. The model was sensitive to training modifications as the validation accuracy peaked at 89.00% in the ninth epoch and declined to 85.98% in the last epoch.

Deeper networks benefit especially from ReLU activation as it maintains non-zero gradients for positive values, hence avoiding vanishing gradient problems. This characteristic probably helped to explain quicker convergence and better accuracy than Sigmoid. ReLU's sensitivity to dying neurones, where certain neurones output zero for all inputs, hence lowering model capacity, may have contributed to several generalisation inconsistencies shown by the model.

Comparison of Results:

Tanh turned out the best among the three activation functions, with the lowest validation loss (0.1689) and the greatest validation accuracy (95.39%). It was the most dependable option for this categorisation work as it matched generalisation with learning speed.

Though it exhibited obvious variations in validation accuracy, the ReLU model outperformed the Sigmoid-based model. Although it performed really well, it lacked the constancy of the Tanh-based model. The slowest convergence and lowest accuracy of the Sigmoid model indicated that the vanishing gradient issue hampered efficient feature extraction. This implies that in image classification activities, Sigmoid activation is less appropriate for deep networks. In this particular CNN architecture, Tanh exceeded ReLU and Sigmoid overall to show the best trade-off between convergence speed, accuracy, and stability.

5. Conclusion

Evaluation of different activation functions in CNNs reveals that in terms of accuracy, stability, and convergence Tanh beats ReLU and Sigmoid. To show great generalisation, Tanh attained the lowest loss and the best validation accuracy. ReLU exhibited variations but performed well, most likely because of dying neurones, therefore influencing consistency. With delayed learning and reduced accuracy hampered by vanishing gradient problems, Sigmoid suffered most.