Convolutional Neural Network (CNN)

NGUYEN Huy Hoang
Department of information and communication
University of science and technology of Hanoi (USTH)
Hanoi, Vietnam
hoangnh2440040@usth.edu.vn

***Abstract*—Today, the term deep learning is trending because of its application in various fields. And one of the deep learning models became famous is the convolutional neural network, which plays a role as a basis in many famous state-of-the-art models. Through the past years, although this model has becomes traditional, but it is still very strong and believed by researchers and engineers in applications. In this project, we will see the background about this famous model and implement it in pure Python without the support of advanced libraries. Then we will see the result and discuss the obstacles during the implementation and have an insight into when to improve this model in the future.**

***Index Terms*—deep learning, convolutional neural network, classification**

I. Introduction

In recent years of development in computer science, artificial intelligence (AI) has emerged as the new trending research field with a variety of applications in many aspects of life. The development of a machine learning model is an area of interest in artificial intelligence. Specifically, deep learning direction is where a model can be learned with high accuracy result with a very deep architecture, as the name of the subfield. Deep learning can solve many kinds of problems with complex unstructured datasets such as images, audio, video, and text. Also, many hard tasks such as image classification, object detection, and semantic segmentation can be solved by using deep learning. In [3], he shows the advantages of different face recognition models, one of the most famous application of CNN in the computer vision.

CNN is a kind of model in the deep learning area, specifically resolve the problems which is related to the image. It considered more powerful than the traditional neural networks because it applies advanced techniques that are used in image processing, convolution techniques.

In this report, we are going to do research about different aspect of CNN. This report is structured as follows:

- Theory: Firstly, we will study about the theory behind the CNN so that we can comprehend why CNN is famous for the image data.
- Implementation: we will see the detailed developed CNN by using vanilla Python.
- Next section will talk about the results of the developed CNN and discuss it.
- Conclusion and future work will follow to the end of the report.

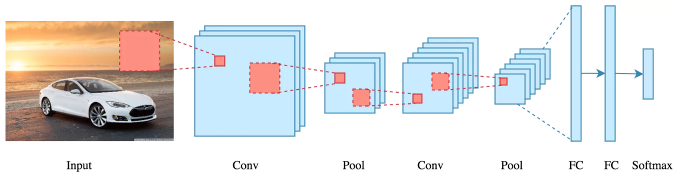


Fig. 1. Typical CNN architecturee

II. Convolutional neural network layer

From Figure 1, we can see the structure of a typical CNN architecture. A CNN mainly contains of three different kinds of layers.

- Convolutional layers: the core building blocks of a CNN. This layer applies a set of learnable filters (kernels) to the input image to extract spatial features such as edges, textures, and patterns.
- Pooling layer: used to reduce the spatial dimensions, it will be used to suppress noise and ignore unnecessary elements, but still retain the most important information in a particular region of interest.
- (Fully connected layers) dense layer: the same as standard neural networks, where current layers are fully connected to all activations in the previous one.

A. Forward-propagation in CNN

1) Convolutional Layer: From [2], assuming that we have 5×5 an input image with 3×3 a filter, for the simplification, we will consider that the stride is 1 and only 1 channel:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \end{bmatrix} \qquad \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

We will try performing padding equal to 1, and now we have a result like the following matrix.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & 0 \\ 0 & x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & 0 \\ 0 & x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & 0 \\ 0 & x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & 0 \\ 0 & x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Now, we have the image matrix $\mathbf{X}$ and the filter matrix $\mathbf{W}$ as the following matrix.

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & 0 \\ 0 & x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & 0 \\ 0 & x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & 0 \\ 0 & x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & 0 \\ 0 & x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

Now, we apply the filter matrix $\mathbf{W}$ from the outermost top-left, and then we get the result matrix like the following:

$$\begin{bmatrix} 0 \times w_{11} & 0 \times w_{12} & 0 \times w_{13} & 0 & 0 & 0 & 0 \\ 0 \times w_{21} & x_{11} \times w_{22} & x_{12} \times w_{23} & x_{13} & x_{14} & x_{15} & 0 \\ 0 \times w_{31} & x_{21} \times w_{22} & x_{22} \times w_{33} & x_{23} & x_{24} & x_{25} & 0 \\ 0 & x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & 0 \\ 0 & x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & 0 \\ 0 & x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We got the result of y_{11}:

$$\begin{aligned} y_{11} = {} & (0 \times w_{11}) + (0 \times w_{12}) + (0 \times w_{13}) + \\ & (0 \times w_{21}) + (x_{11} \times w_{22}) + (x_{12} \times w_{23}) + \\ & (0 \times w_{31}) + (x_{21} \times w_{32}) + (x_{22} \times w_{33}) \end{aligned}$$

We continue to move the filter matrix from left to right, top to bottom, and then we get the following matrix:

$$\begin{bmatrix} 0 & 0 \times w_{11} & 0 \times w_{12} & 0 \times w_{13} & 0 & 0 & 0 \\ 0 & x_{11} \times w_{21} & x_{12} \times w_{22} & x_{13} \times w_{23} & x_{14} & x_{15} & 0 \\ 0 & x_{21} \times w_{31} & x_{22} \times w_{22} & x_{23} \times w_{33} & x_{24} & x_{25} & 0 \\ 0 & x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & 0 \\ 0 & x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & 0 \\ 0 & x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{aligned} y_{12} = {} & (0 \times w_{11}) + (0 \times w_{12}) + (0 \times w_{13}) + \\ & (x_{11} \times w_{21}) + (x_{12} \times w_{22}) + (x_{13} \times w_{23}) + \\ & (x_{21} \times w_{31}) + (x_{22} \times w_{32}) + (x_{23} \times w_{33}) \end{aligned}$$

We can see that result after performing the convolution operation. The size of the result matrix is reduced with the size following the formula:

$$H_{out} = \left\lfloor 1 + \frac{(H + 2 \cdot \text{pad} - HH)}{\text{stride}} \right\rfloor$$

$$W_{out} = \left\lfloor 1 + \frac{(W + 2 \cdot \text{pad} - WW)}{\text{stride}} \right\rfloor$$

After that, we got the result as follows:

$$\begin{bmatrix} y_{11} & y_{12} & y_{13} & y_{14} & y_{15} \\ y_{21} & y_{22} & y_{23} & y_{24} & y_{25} \\ y_{31} & y_{32} & y_{33} & y_{34} & y_{35} \\ y_{41} & y_{42} & y_{43} & y_{44} & y_{45} \\ y_{51} & y_{52} & y_{53} & y_{54} & y_{55} \end{bmatrix}$$

Applying the formulae to calculate the size of the output matrix, we will have the following results:

$$H_{out} = \left\lfloor 1 + \frac{(5 + 2 \cdot 1 - 3)}{1} \right\rfloor = 5$$

$$W_{out} = \left\lfloor 1 + \frac{(5 + 2 \cdot 1 - 3)}{1} \right\rfloor = 5$$

2) Pooling layer: There are two kinds of famous pooling layers, which are often used in CNN: max pooling and average pooling. Max pooling layer is often used in CNN because it highlights the transition from pixels in one region to another region. For instance, when we consider the transition from black color to white color, the max pooling can remove the redundant pixel value and only retain the highest in that region, which could be the white color, and another region could be the gray color as the highest value.

We consider the example from [1] as follows:

$$\begin{bmatrix} \begin{array}{|cc|cc|} \hline x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ \hline x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \\ \hline \end{array} \end{bmatrix}$$

With the stride of 2, we choose the biggest max value in each region. After that, we have a new matrix with the value

$$y_{11} = \max(x_{11}, x_{12}, x_{21}, x_{22})$$

$$y_{12} = \max(x_{13}, x_{14}, x_{23}, x_{24})$$

$$y_{21} = \max(x_{31}, x_{32}, x_{41}, x_{42})$$

$$y_{22} = \max(x_{33}, x_{34}, x_{43}, x_{44})$$

$$H_{out} = \left\lfloor 1 + \frac{H - \text{pool\_height}}{\text{stride}} \right\rfloor$$

$$W_{out} = \left\lfloor 1 + \frac{W - \text{pool\_width}}{\text{stride}} \right\rfloor$$

Applying into the above example, we can get the result of the output shape is

[

$$H_{out} = \left\lfloor 1 + \frac{(4 - 2)}{2} \right\rfloor = 2$$

[

$$W_{out} = \left\lfloor 1 + \frac{(4 - 2)}{2} \right\rfloor = 2$$

And the result of the above example can become like the below matrix

$$\begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix}$$

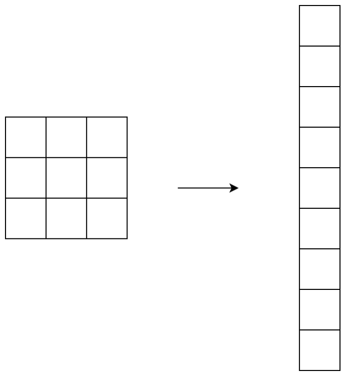


Fig. 2. Feature map is flattened into 1 dimension array

3) Dense layer: The output (feature map) of the max pooling layer now is flattened into a one-dimensional array, and the data into the dense layer. Dense layer, or fully connected layers, have each neuron in the current layer connect to all the neurons from the previous layer. the activation in the current layer is calculated with the matrix multiplication between the weights the number of feature from the previous layers. This is the same as a normal neural network.

There are many activation function used in the dense layer. But the famous activations we often see are the ReLU function or the softmax function for multi-classification.

$$\text{ReLU}(x) = \max(0, x)$$

$$\text{Softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

B. Backpropagation

1) Convolutional layer: Continue with the example from [2]. In the backpropagation process, we need to compute derivatives of the output $\mathbf{Y}$ with respect to input $\mathbf{X}$, filter $\mathbf{W}$ and bias $\mathbf{b}$.

Moving into the example of calculation of $\frac{\partial Y}{\partial X}$. First, let see the calculation of $\frac{\partial Y}{\partial x_{11}}$. We see that the change of x_{11} can affect the y_{11}, y_{12}, y_{21} and y_{22}. So we have the formula:

$$\frac{\partial Y}{\partial x_{11}} = \frac{\partial y_{11}}{\partial x_{11}} + \frac{\partial y_{12}}{\partial x_{11}} + \frac{\partial y_{21}}{\partial x_{11}} + \frac{\partial y_{22}}{\partial x_{11}}$$

We can see the matrix representation of how changing x_{11} can affect Y.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \boxed{x_{11}} & x_{12} & x_{13} & x_{14} & x_{15} & 0 \\ 0 & x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & 0 \\ 0 & x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & 0 \\ 0 & x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & 0 \\ 0 & x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \boxed{x_{11}} & x_{12} & x_{13} & x_{14} & x_{15} & 0 \\ 0 & x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & 0 \\ 0 & x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & 0 \\ 0 & x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & 0 \\ 0 & x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \boxed{x_{11}} & x_{12} & x_{13} & x_{14} & x_{15} & 0 \\ 0 & x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & 0 \\ 0 & x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & 0 \\ 0 & x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & 0 \\ 0 & x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \boxed{x_{11}} & x_{12} & x_{13} & x_{14} & x_{15} & 0 \\ 0 & x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & 0 \\ 0 & x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & 0 \\ 0 & x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & 0 \\ 0 & x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Furthermore, when doing the backpropagation, we also have the incoming gradient from the following layer. So in this case, we would have the incoming gradient of Y with respect to the loss L, which is $\frac{\partial L}{\partial Y}$ with the following:

$$\begin{bmatrix} dy_{11} & dy_{12} & dy_{13} & dy_{14} & dy_{15} \\ dy_{21} & dy_{22} & dy_{23} & dy_{24} & dy_{25} \\ dy_{31} & dy_{32} & dy_{33} & dy_{34} & dy_{35} \\ dy_{41} & dy_{42} & dy_{43} & dy_{44} & dy_{45} \\ dy_{51} & dy_{52} & dy_{53} & dy_{54} & dy_{55} \end{bmatrix}$$

So by performing the chain rule when calculate the derivative $\frac{\partial Y}{\partial X}$, we modify the computation into

$$\frac{\partial Y}{\partial x_{11}} = \frac{\partial L}{\partial y_{11}}\frac{\partial y_{11}}{\partial x_{11}} + \frac{\partial L}{\partial y_{12}}\frac{\partial y_{12}}{\partial x_{11}} + \frac{\partial L}{\partial y_{21}}\frac{\partial y_{21}}{\partial x_{11}} + \frac{\partial L}{\partial y_{22}}\frac{\partial y_{22}}{\partial x_{11}}$$

Similarly, we also need to update the new weight of the filter. Take the example of w_{11}, the change of w_{11} can also affect the output Y where w_{11} do the convolution.

$$\frac{\partial Y}{\partial w_{11}} = \frac{\partial L}{\partial y_{11}}\frac{\partial y_{11}}{\partial w_{11}} + \frac{\partial L}{\partial y_{12}}\frac{\partial y_{12}}{\partial w_{11}} + \cdots + \frac{\partial L}{\partial y_{45}}\frac{\partial y_{45}}{\partial w_{11}} + \frac{\partial L}{\partial y_{55}}\frac{\partial y_{55}}{\partial w_{11}}$$

2) Pooling layer: Because there are no parameters in the max pooling layer. We only have the max value that can affect the output so we only see the max value in each regions. For instance [1], we have

$$\begin{bmatrix} dy_{11} & dy_{12} \\ dy_{21} & dy_{22} \end{bmatrix}$$

and we also have the corresponding x value with respect to the y value as the follow formula

$$y_{11} = \max(x_{11}, x_{12}, x_{21}, x_{22}) = x_{12}$$

$$y_{12} = \max(x_{13}, x_{14}, x_{23}, x_{24}) = x_{23}$$

$$y_{21} = \max(x_{31}, x_{32}, x_{41}, x_{42}) = x_{31}$$

$$y_{22} = \max(x_{33}, x_{34}, x_{43}, x_{44}) = x_{34}$$

Applying the chain rule, now we will the matrix as follows:

$$\begin{bmatrix} \begin{array}{|cc|cc|} \hline 0 & dy_{11} & 0 & 0 \\ 0 & 0 & dy_{12} & 0 \\ \hline dy_{21} & 0 & 0 & dy_{22} \\ 0 & 0 & 0 & 0 \\ \hline \end{array} \end{bmatrix}$$

3) Dense layer: The purpose of backpropagation is to reduce the loss; MSE is often used in regression, and cross-entropy loss is the common loss for classification. The next is that of calculating the gradients of loss with respect to each of the weights in each layer by using the chain rule. After that, we will use gradient descent to update the weights so that the loss is minimized.

III. Implementation

A. Overall Architecture

In this project, we will see the implementation of a simple CNN implemented by using purely in python programming language. The architecture of this CNN is illustrated in Figure 3.

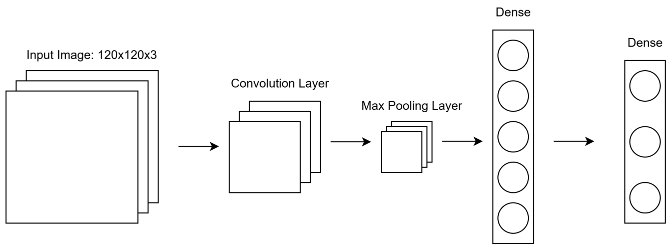


Fig. 3. Developed simple CNN architecture

We can see this simple configuration from the config file 'file_config.txt':

- `input_shape`: h=120, w=120, c=3
- `conv`: filter_size=3, num_filters=4
- `maxpool`: pooling_size=2
- `dense1`: output_size=12
- `dense2`: output_size=2
- `num_classes`: num_classes=3

Other than that, there are some default configurations that have been implemented that we can see in the code. In general, we can see the summary of the model as the following:

CNN Model Summary:

- **Input Shape:** $(120, 120, 3)$
- **Conv Layer:** 4 filters, size 3×3
- **MaxPool Layer:** 2×2
- **Dense Layer 1:** $14400 \rightarrow 12$ (ReLU)
- **Dense Layer 2:** $12 \rightarrow 3$ (Softmax)
- **Output Classes:** 3

B. Dataset

In this project, it takes some pictures from the dataset 'Animal Species Classification - V3' in the Kaggle website. The data contains of 9 different images, each labels will contains three different images for that label, where the label 1 is the cat, label 2 is the cow, and label 3 is the dog.

C. Developed environment

This project is implemented in pure Python 3 with the support of some basic built-in libraries, including

- math
- typing
- PIL
- matplotlib

D. Developed CNN

We can see the organization of the implementation is structured as follows.

```
final_project
├── data
├── utils
│   ├── conv_layer.py
│   ├── dense_layer.py
│   ├── image_util.py
│   ├── my_random.py
│   ├── pooling_layer.py
│   └── shape_util.py
├── cnn.py
└── file_config.txt
```

where the files `conv_layer.py`, `pooling_layer.py`, and `dense_layer.py` contain the corresponding implementations of the `ConvLayer`, `MaxPoolingLayer`, and `DenseLayer` classes, respectively. Each of these classes includes the implementation of both `forward` and `backward` methods as similar as possible with the theory.

These class is implement to deal with images three channels In the `ConvLayer` class, it is still not implemented the padding, stride function due to the obstacles to dealing with python's list data structure. There are also some struggle with limited features in the other layer class. In the `image_util.py`, there is a method `convert_to_list`, which is used for converting the PIL image into the Python's list data structure In the `my_random.py`, we have the `MyPseudoRandom`, which is used for pseudo random initializingg the weights. In the `cnn.py`, it contains the `MyCNN` class, which is used for create the MyCNN architecture, it also contains the main method to run the MyCNN as the architeture of figuge 3.

IV. Result and discussion

In this section, we will see the result of our implementation. We can see the training result from the figure 4

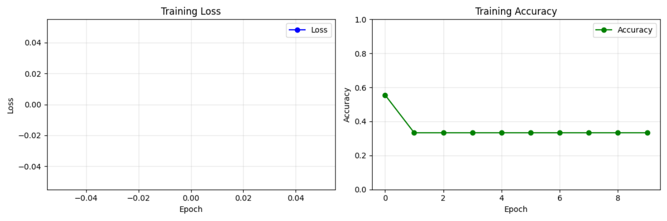
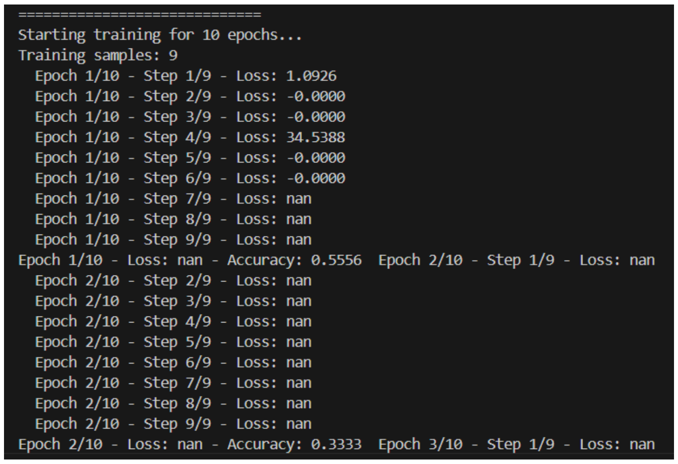


Fig. 4. Training result of MyCNN

From the figure 4, we can see that there are some problems with the calculation of the training loss where we can not calculate the loss in each epoch. On the other hand, the training accuracy can not improve better, it almost 60% in epoch 0 but but reduce below 40% in the next epoch.



```
============================
Starting training for 10 epochs...
Training samples: 9
  Epoch 1/10 - Step 1/9 - Loss: 1.0926
  Epoch 1/10 - Step 2/9 - Loss: -0.0000
  Epoch 1/10 - Step 3/9 - Loss: -0.0000
  Epoch 1/10 - Step 4/9 - Loss: 34.5388
  Epoch 1/10 - Step 5/9 - Loss: -0.0000
  Epoch 1/10 - Step 6/9 - Loss: -0.0000
  Epoch 1/10 - Step 7/9 - Loss: nan
  Epoch 1/10 - Step 8/9 - Loss: nan
  Epoch 1/10 - Step 9/9 - Loss: nan
Epoch 1/10 - Loss: nan - Accuracy: 0.5556  Epoch 2/10 - Step 1/9 - Loss: nan
  Epoch 2/10 - Step 2/9 - Loss: nan
  Epoch 2/10 - Step 3/9 - Loss: nan
  Epoch 2/10 - Step 4/9 - Loss: nan
  Epoch 2/10 - Step 5/9 - Loss: nan
  Epoch 2/10 - Step 6/9 - Loss: nan
  Epoch 2/10 - Step 7/9 - Loss: nan
  Epoch 2/10 - Step 8/9 - Loss: nan
  Epoch 2/10 - Step 9/9 - Loss: nan
Epoch 2/10 - Loss: nan - Accuracy: 0.3333  Epoch 3/10 - Step 1/9 - Loss: nan
```

Fig. 5. The loss over each epoch

From Figure 5, we can see that in the first epoch, the loss is only 1.0926 and reduce to zero, but at the step 4/9, the loss became higher with the result of 34.5388. The reason is that the training dataset introduces a new data with a new output label, which causes the error to becomes higher. but it reduces again in the next step. In the next step, the loss surprisingly became nan. Probably the reason behind it is because of the Python float type can not store the very small zero value such as 0.000000000001. Instead of that, it consider as the zero value, which may cause the derivative of some $\frac{\partial L}{\partial w}$ actually became zero: $\frac{\partial L}{0}$ and the value of the loss now become `nan`. Another reason can be the initialized value is not actually optimized.

V. Conclusion and Future Work

Implement CNN in pure python is really a big challenges because there are a lack of optimized, specific-purpose libraries. Another challenges is that we have to create many inner loops to go through a big Python list, which may cause the time of training to is very long. It is not good as the parallel element-wise operations libraries. Also, the variable type in python may not store a very small zero value and consider it as the actual 0. In the future, this work needs to improved the way of pseudorandom actually can be more random. Dealing with the very small value of zero so that it can not be divided by the actual zero. Also, the convolution can develop with more advanced feature such as adding the padding and stride features with the bigger value.

Acknowledgment

This work was done as the final project in the course deep learning, which was taught by the lecturer Assoc. Prof. Dr. Tran Giang Son at the university of science and technology of Hanoi (USTH).

References

[1] A. Kravets, "Forward and backward propagation of pooling layers in convolutional neural networks," *Towards Data Science*, Feb. 21, 2022. [Online]. Available: https://towardsdatascience.com/forward-and-backward-propagation-of-pooling-layers-in-convolutional-neural-network

[2] A. Kravets, "Forward and backward propagation in convolutional neural networks," *Towards Data Science*, Feb. 18, 2022. [Online]. Available: https://towardsdatascience.com/forward-and-backward-propagation-in-convolutional-neural-networks-64365925fdfa/.

[3] J. Yan, "Application of CNN in computer vision," *Applied and Computational Engineering*, vol. 30, pp. 104–110, Jan. 2024, doi: 10.54254/2755-2721/30/20230081.