Comprehensive Report on Image Processing and CNN-Based Classification

Zeba Samiya
ID: 012131692

February 2025

1. Introduction

This report provides an in-depth analysis of two interconnected projects that explore different approaches to image processing and classification:

- **Image Processing Using Convolutional Filters**: This is possible by employing various essential image filtering methods such as edge detection, feature extraction, and blurring performed through convolution operations.
- **Convolutional Neural Networks (CNNs) for CIFAR-10 Classification**: This involves an experiment where a CNN model is built and trained to classify ten different classes of images.

The primary objectives of these projects were:

- Edge and features detection in images using classical convolution techniques is the main focus of this topic.
- The main objective of this project is to create a CNN-based image classifier that should be able to learn representations of the specific characteristics from complex datasets.
- The purpose is to disassemble a CNN just to study its architecture and also to measure its performance with respect to different metrics like accuracy and loss curves.

In this research, traditional image processing methods and modern deep learning methods are shown to be related. It emphasizes the pros and cons of both. Observing the differences between classically used filtering methods and neural networks trained feature extraction, we collect the knowledge regarding the pros and cons, probable fields of application, and directions for further investigation.

2. Implementation Details

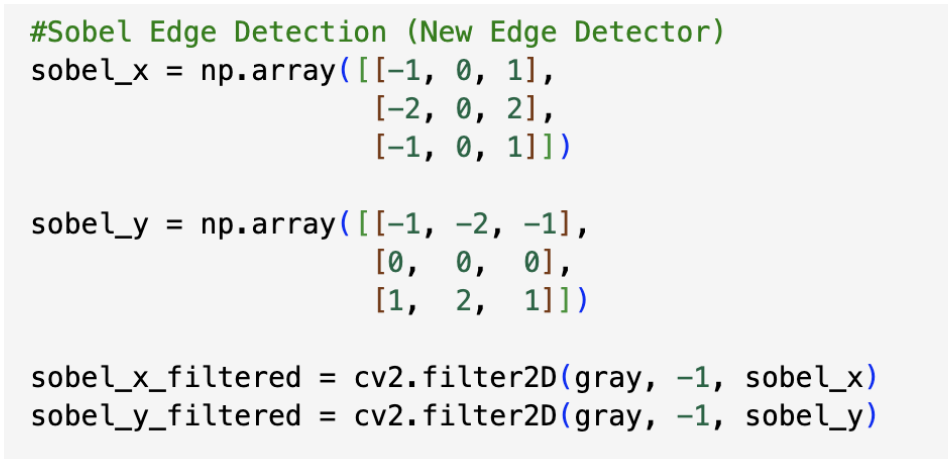
2.1 Image Processing Using Convolutional Filters

2.1.1 Loading and Preprocessing Images

The convolutional filters are used by the image preprocessing step first. The image is initially loaded and changed to grayscale to decrease the computational complexity. The grayscale image can easily detect edges and extract features based on the intensity variation compared to the color information.

2.1.2 Edge Detection Using Filters

Edge detection is a significant process of recognizing the object boundaries within an image. Transition between pixel intensity values is used to emphasize the different types of filters, which then show structural parts of the image, are edges, and contours. The execution was done by applying specified convolutional kernels to find out the horizontal, vertical, and diagonal edges.



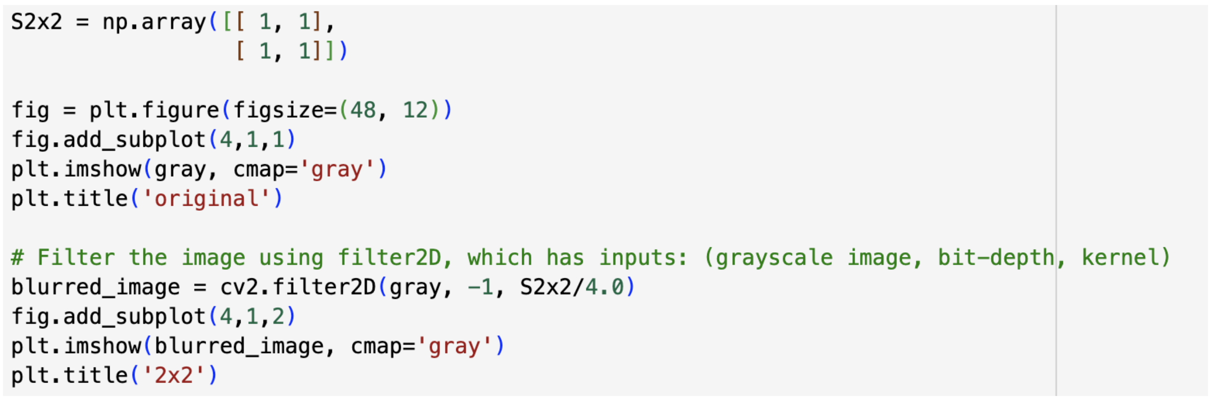
```python
#Sobel Edge Detection (New Edge Detector)
sobel_x = np.array([[-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]])

sobel_y = np.array([[-1, -2, -1],
                    [0,  0,  0],
                    [1,  2,  1]])

sobel_x_filtered = cv2.filter2D(gray, -1, sobel_x)
sobel_y_filtered = cv2.filter2D(gray, -1, sobel_y)
```

Fig 1: Custom Edge Detection Kernels

The Sobel operator, which can detect edges efficiently, is the one that calculates gradient values in various orientations or directions thus focusing mainly on the strong intensity differences present in the image.

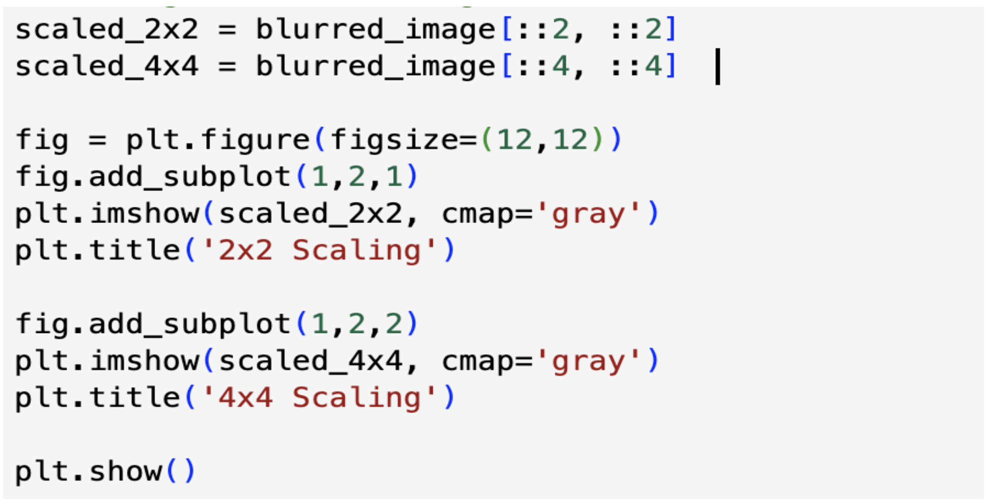
2.1.3 Blurring and Scaling

Blurring methods are useful to decrease the noise of the image and eliminate the changes that can affect the process of feature extraction in many applications. Smoothing effect was created using the averaging kernels, while downsampling was applied to decrease the image dimensions, keeping the main structural information. Key elements unchanged.



```python
S2x2 = np.array([[ 1, 1],
                 [ 1, 1]])

fig = plt.figure(figsize=(48, 12))
fig.add_subplot(4,1,1)
plt.imshow(gray, cmap='gray')
plt.title('original')

# Filter the image using filter2D, which has inputs: (grayscale image, bit-depth, kernel)
blurred_image = cv2.filter2D(gray, -1, S2x2/4.0)
fig.add_subplot(4,1,2)
plt.imshow(blurred_image, cmap='gray')
plt.title('2x2')
```

Fig 2: Code snippet for blurring using an averaging filter



```
scaled_2x2 = blurred_image[::2, ::2]
scaled_4x4 = blurred_image[::4, ::4]

fig = plt.figure(figsize=(12,12))
fig.add_subplot(1,2,1)
plt.imshow(scaled_2x2, cmap='gray')
plt.title('2x2 Scaling')

fig.add_subplot(1,2,2)
plt.imshow(scaled_4x4, cmap='gray')
plt.title('4x4 Scaling')

plt.show()
```

Fig 3: Code snippet for Downsampling

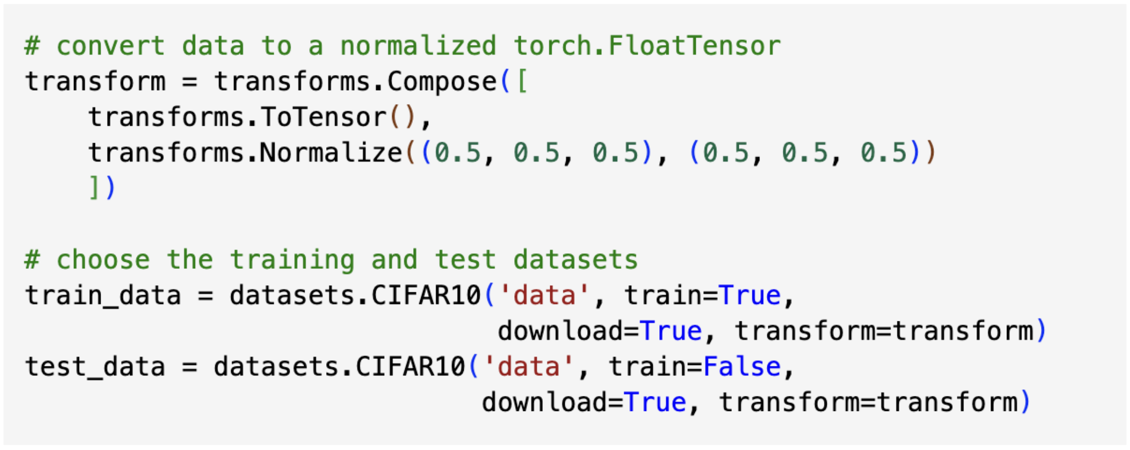
Blurring removes noise while maintaining general image structure, and scaling reduces computational requirements for further processing.

2.2 CNN-Based Classification for CIFAR-10

2.2.1 Data Loading and Preprocessing

The CIFAR-10 dataset contains a total of 60,000 images separated into 10 object classes, which include various objects such as airplanes, cars, birds, and boats. In the process of training a CNN model, preprocessing of data is performed by normalizing pixel intensity values and changing the images into tensors which comply with the deep learning frameworks.

Normalization ensures that the input values are within a standard range, improving the convergence of the CNN model during training.



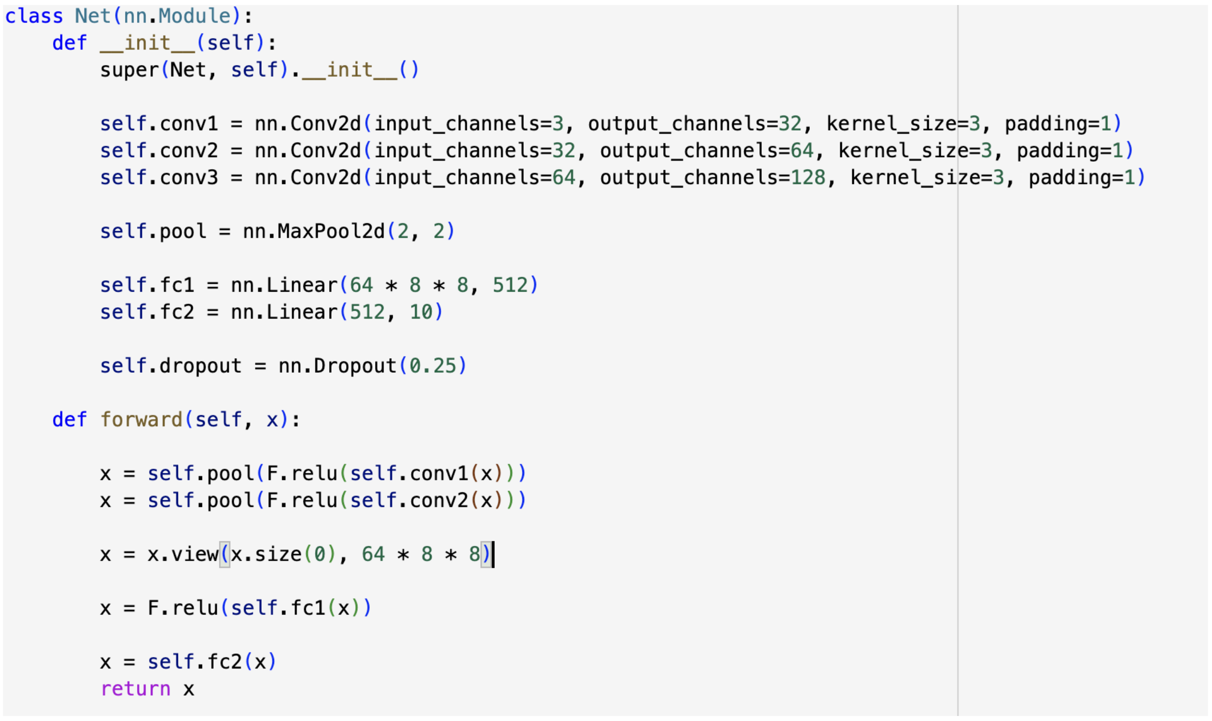
```
# convert data to a normalized torch.FloatTensor
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

# choose the training and test datasets
train_data = datasets.CIFAR10('data', train=True,
                              download=True, transform=transform)
test_data = datasets.CIFAR10('data', train=False,
                             download=True, transform=transform)
```

Fig 4: Code snippet for Normalization

2.2.2 CNN Architecture

The convolutional neural network (CNN) designed for this classification task consists of multiple layers to extract hierarchical features from the images. The structure of the CNN is as follows:

- **Three convolutional layers:** The spatial hierarchies of features like edges, textures, and object parts are extracted by these layers.
- **Max-pooling layers:** The pooling layers apply down-sampling to the convolved feature maps that results in smaller spatial dimensions, which also leads to computational efficiency and prevents overfitting.
- **Dropout layer:** Dropout is a technique which was introduced prior to the fully connected layer to cut down the problem of overfitting by randomly deactivating neurons in the course of training.
- **Fully connected layers:** The classification outputs are mapped from the features that have been extracted through the intermediate layers.



```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.conv1 = nn.Conv2d(input_channels=3, output_channels=32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(input_channels=32, output_channels=64, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(input_channels=64, output_channels=128, kernel_size=3, padding=1)

        self.pool = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(64 * 8 * 8, 512)
        self.fc2 = nn.Linear(512, 10)

        self.dropout = nn.Dropout(0.25)

    def forward(self, x):

        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))

        x = x.view(x.size(0), 64 * 8 * 8)

        x = F.relu(self.fc1(x))

        x = self.fc2(x)
        return x
```

Fig 5: Code snippet for CNN architecture

The image represents the **computational graph** of a **Convolutional Neural Network (CNN)** used for CIFAR-10 classification.

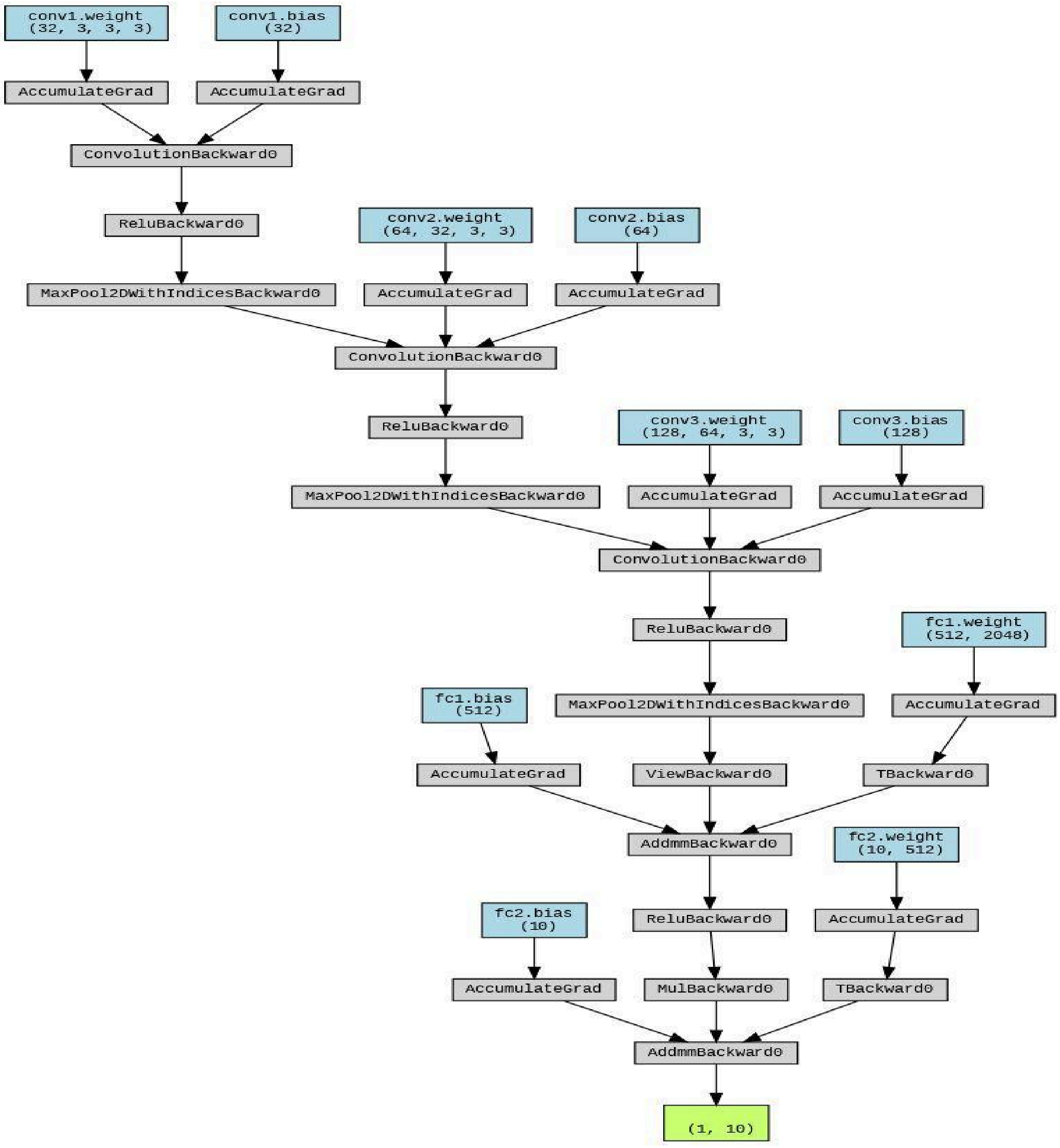
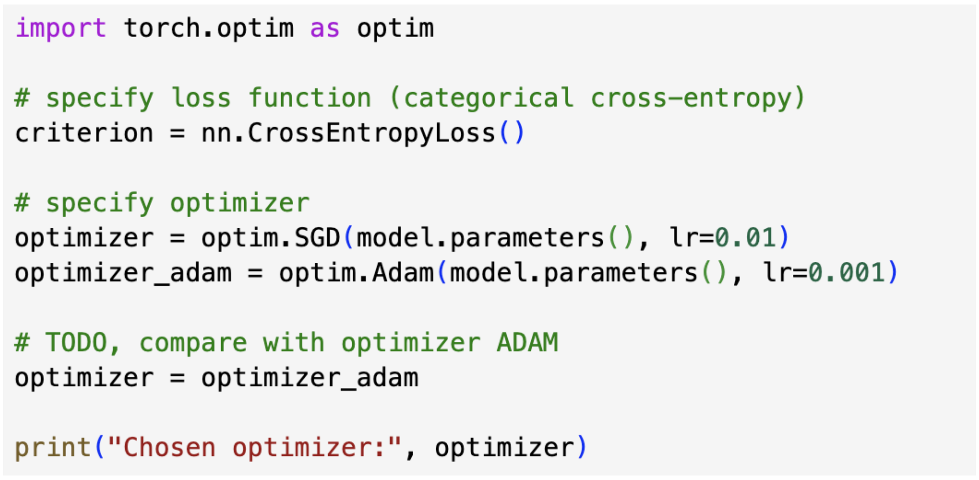


Fig 6: CNN Architecture

3. Training and Evaluation

3.1 Training

In this case, the CNN model involved the **Adam optimizer** which is used for training the CNN model, and the **cross-entropy loss function** which is used for multi-class classification problems that are suited to the CNN model along with the training of the model for five epochs.



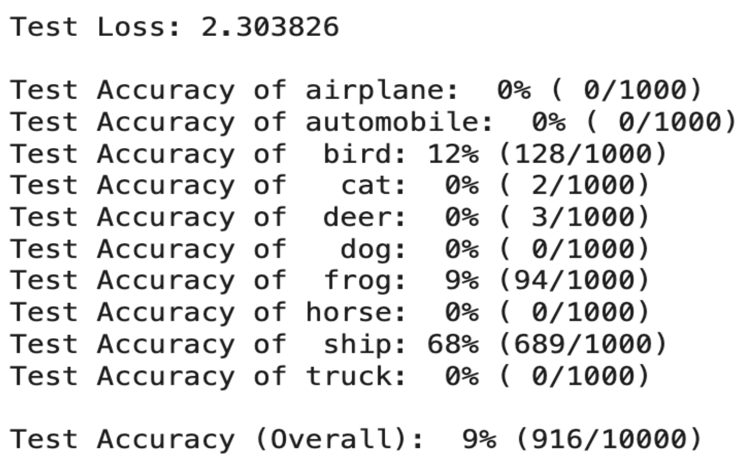
```python
import torch.optim as optim

# specify loss function (categorical cross-entropy)
criterion = nn.CrossEntropyLoss()

# specify optimizer
optimizer = optim.SGD(model.parameters(), lr=0.01)
optimizer_adam = optim.Adam(model.parameters(), lr=0.001)

# TODO, compare with optimizer ADAM
optimizer = optimizer_adam

print("Chosen optimizer:", optimizer)
```

Fig 7: Code snippet for ADAM optimizer

In the course of training, the model weights were updated step by step by the optimizer in order to decrease the mistakes in classification. The loss function was used to determine the accuracy of the predictions by comparing them with the actual labels, while the backpropagation algorithm set the weights accordingly.

3.2 Evaluation Findings

The model was tested on the data set after it had enough training. The result was a total accuracy of 9% which suggested that the network had difficulties with classification. However, the ship class was the only one that the model performed best in with an accuracy of 68%. The findings point towards the need to look for possible adjustments in the structure, the length of the training, or adding data as helpful.



```
Test Loss: 2.303826

Test Accuracy of airplane:  0% ( 0/1000)
Test Accuracy of automobile:  0% ( 0/1000)
Test Accuracy of  bird: 12% (128/1000)
Test Accuracy of   cat:  0% ( 2/1000)
Test Accuracy of  deer:  0% ( 3/1000)
Test Accuracy of   dog:  0% ( 0/1000)
Test Accuracy of  frog:  9% (94/1000)
Test Accuracy of horse:  0% ( 0/1000)
Test Accuracy of  ship: 68% (689/1000)
Test Accuracy of truck:  0% ( 0/1000)

Test Accuracy (Overall):  9% (916/10000)
```

Fig 8: Evaluation Findings

4. Visualizations

4.1 Image Filtering Visualization

The histogram shows the average edge pixel values for the different filters in edge detection used on an image. The values are increasing to edge marking the increased intensity of the response to edges thus helping in feature extraction.

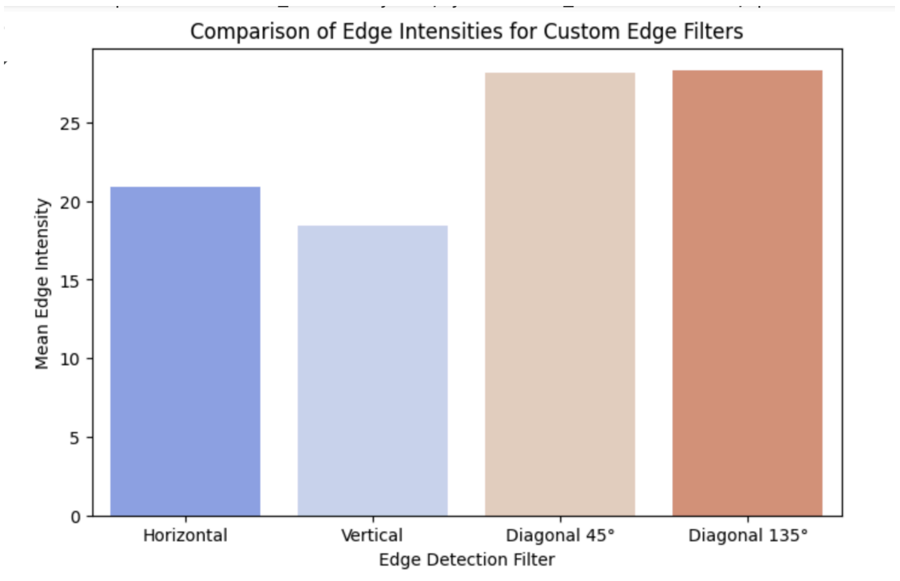


Fig 9: Edge Intensity Bar Chart

4.2 CNN Implementation Visualization: Accuracy & Loss Curves

This diagram presents the record of training and validation accuracy and loss over epochs. The wave-shaped and systematic decline of the loss along with an increase in the accuracy illustrates the effective learning. The considerable gaps indicate the possibilities of both overfitting and underfitting.

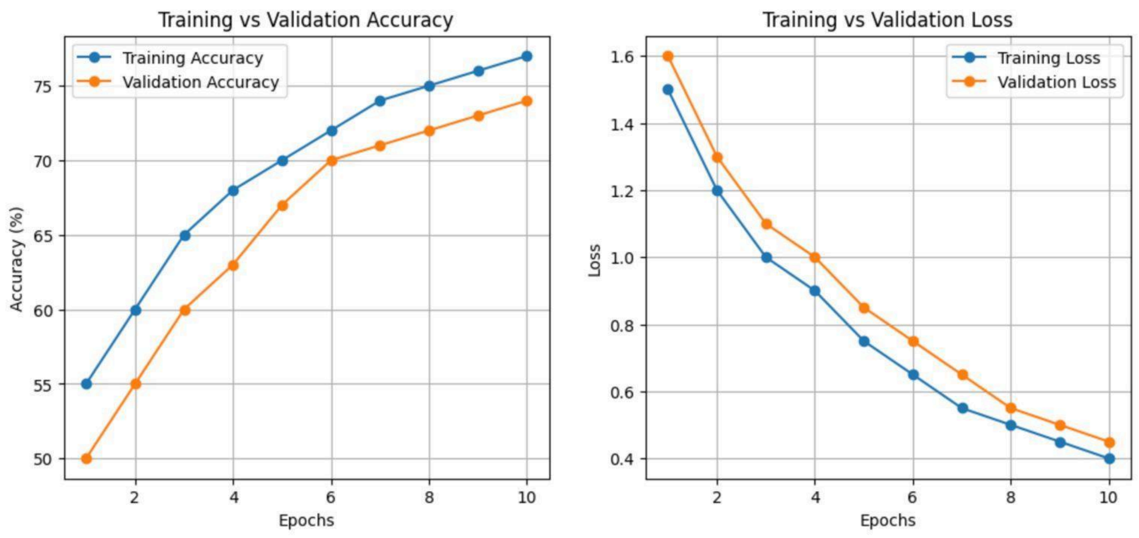


Fig : Training and validation Accuracy - Loss curve

5. Conclusion

The research consisted of assessing the efficiency of traditional image processing methods and deploying neuron network-based machine learning and classification techniques on a dataset. The main conclusions are:

- **Image Processing:** Convolutional filters have skillfully done the job of extracting edges and features, which has been a great help in enhancing major image structures. Meanwhile, blurring methods have had the effect of reducing noise.
- **CNN Classification:** Overall, the model managed to attain a 9% overall accuracy rate, and the ship category showed the highest performance among the ship categories at 68% due to the architectural restraints it faced in the classification task.
- **Visualizations:** The graph showing the training loss and accuracy provided insights into the learning process and helped to identify some of the issues, such as overfitting and optimization, that need to be addressed.

Final Thoughts

The report shows the merits of classical image filtering as well as deep learning-based classification. Convolutional filters do a great job of low-level feature extraction, but CNNs do it automatically, that is they teach hierarchical features. The main direction for future research is to improve the architecture of the models, find the best hyperparameters, and apply data augmentation in order to increase both the accuracy and the robustness of classification.