Project I: Image Classification

Daniele Murgolo
Chalmers University of Technology
murgolo@chalmers.se

Abstract

The goal of this project is to develop a convolutional neural network (CNN) that accurately classifies CIFAR-10 images. The CIFAR-10 dataset consists of 32x32 color images with 10 distinct classes. To accomplish the objective, a basic CNN architecture is employed and progressively improved using pre-processing, early stopping, batch normalization, and hyperparameter tuning techniques.

During the development process, I evaluated the models based on their test accuracy and confusion matrices. My iterative refinement approach resulted in a final CNN model that achieves a test accuracy of 66%. These results demonstrate the effectiveness of employing optimization methods to enhance CNN performance in image classification tasks.

Overall, this study underscores the importance of exploring various approaches to improve the accuracy of CNN models.

1. Introduction

In the computer vision and image analysis field, image classification is the task of associating one or more labels with a given image. The automation of multi-class classification using computers is an increasingly important capability, driven by the large volume of image data currently available, that presents significant challenges for manual analysis. The issue originates from the lack of human experience, inadequate image quality, and length of time involved. We need an approach in the present market that can classify images with small effort and productively. *"Supervised Learning"* and *"Unsupervised Learning"* are the two main classification methods. In this project, we are going to focus only on the former.

In image classification, supervised learning is a method that involves the use of labeled data to recognize and classify other images. The process begins with selecting a training dataset that is representative of the features of interest. Then, we use this data to train a model that can accurately classify unseen images, the test dataset. During training, the model is usually adjusted based on feedback from a subset of the training dataset called the validation dataset (Cunningham *et al*., 2008 [1]).

The CIFAR-10 dataset, Krizhevsky *et al*. (2009) [5], consists of 60000 32x32 color images in 10 classes with 6000 images per class. The classes are *airplane*, *automobile* (but not truck or pickup truck), *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, and *truck* (but not pickup truck). The dataset is divided into five training batches and one test batch each containing 10000 images. The test batch contains 1000 randomly-selected images from each class. The remaining images are randomly distributed to the training batches. This means that a training batch might contain more images from one class than another.

Convolutional neural networks (CNNs) have significantly improved the state-of-the-art in image classification tasks, according to LeCun *et al*. (1998) [7] surpassing traditional methods such as handcrafted feature extraction techniques. CNNs use multiple layers to learn hierarchical features from images. Additionally, they are inherently translation invariant, meaning they can recognize objects in images regardless of their position or orientation.

The overarching goal of this project is to develop a model to classify CIFAR-10 images using a convolutional neural network and achieve at least a 62% accuracy on the test set.

In the remaining report, Section 2 explains the methodology of my work. Section 3 shows experiments carried out on the final network. Section 4 answers to the theoretical questions. Finally, Section 5 summarizes the work done and provides some conclusions.

2. Methodology

This section provides a comprehensive description of the classifier's architecture, as well as the incorporated improvements. In addition, it explains how these improvements will be beneficial.

2.1. Classifier Architecture

The Convolutional Neural Network (CNN) designed for the task is relatively simple. The architecture follows a common pattern in CNNs where a series of convolutional layers and max pooling layers are used to extract features from the

input image, followed by fully connected layers that perform classification. The architecture can be visualized in Figure 1

The input to the network is a 32x32 RGB image. The input is pre-processed by normalizing the pixel values to have zero mean and unit variance. This ensures that each input parameter has a similar data distribution. This makes convergence faster while training. Moreover, it reduces the range of the input values over which the backpropagation algorithm has to work.

The first layer is a 2-D convolutional layer with 32 filters with a width and height of 5 pixels and depth of 3 (corresponding to the 3 color channels). The layer applies the filters to the input with stride 1 (the number of pixels shifts over the input matrix) and padding of 2 (amount of pixels added to an image when it is being processed by the kernel of a CNN). The output of the layer is passed through a rectified linear unit (ReLu) activation function, which introduces non-linearity to the network(Krizhevsky *et al*., 2012) [6].

Next, a max pooling layer with a filter size of 3x3 and stride [2,2] is applied to the output of the first convolutional layer. The purpose of this layer is to downsample the feature maps, reducing their spatial resolution and the number of parameters in the network.

Subsequently, the output of the first max pooling layer is fed through a second 2-D convolutional layer. This layer has the same parameters as the aforementioned first convolutional unit. The number of kernels (32) and their size (5x5) remain unchanged. This layer allows us to extract higher-level features that are more abstract and invariant to small spatial shifts in the input (Hinton & Krizhevsky, 2012 [2]).

The output of the second convolutional layer is again passed through a ReLu activation function and another max pooling layer with the same parameters as the previous one.

The third and final convolutional layer has 64 kernels of size 5x5 and a padding of 2. The resulting output is then passed through a ReLu activation function and a max pooling layer, which is similar to the ones used earlier. This results in a set of 64 feature maps (Zeiler *et al*., 2014 [11]).

The feature maps are then passed through a fully connected layer with 64 units, followed by another ReLu activation function.

The final fully connected layer has 10 neurons, corresponding to the number of output classes in the classification problem. Using a softmax activation function the network converts the output values into a probability distribution over the 10 classes.

The network is trained using the cross-entropy loss function, which measures the difference between the predicted label probabilities and the true labels of the training data. The weights of the network are optimized using backpropagation and stochastic gradient descent.

2.2. Improvements

The initial convolutional neural network was trained using a straightforward set of hyperparameters, which included a learning rate of 0.001, a momentum value of 0.9, a batch size of 128, and 10 epochs. In contrast, the updated version of the network incorporates several enhancements to improve its performance. These enhancements include a modified initial learning rate of 0.0002, the implementation of a piece-wise scheduler with a designated learning rate drop factor and drop period, l2 regularization to mitigate overfitting, a longer training period of 20 epochs, batch normalization to stabilize the internal covariance shift, and the inclusion of a dropout layer to reduce overfitting. Additionally, the updated network includes an output that identifies the best validation loss, thereby enabling further optimization of the model's hyperparameters.

2.2.1 Piece-Wise Scheduler

A piecewise scheduler is a learning rate scheduling technique that involves the learning rate at specific intervals or milestones during the training process. The learning rate is reduced by a factor after each milestone, enabling the model to make fine adjustments to its weights as it converges to an optimal solution (Izmailov *et al*., 2018 [4]). In a CNN, implementing a piecewise scheduler can help improve the model's performance by enabling it to converge more efficiently toward an optimal solution. By reducing the learning rate at specific intervals, the model is less likely to overshoot the optimal solution, and it can make smaller and more precise updates to its weights as it approaches the optimal solution. This can result in a more stable and accurate model with improved generalization performance.

2.2.2 L2 Regularization

Implementing L2 regularization in a CNN can be beneficial for several reasons. Firstly, it can help prevent overfitting to the training data by adding a penalty term to the loss function that discourages the model from assigning too much importance to any individual feature. This encourages the model to generalize better to new and unseen data. Secondly, it can help improve the stability of the model during training by reducing the sensitivity of the model to small changes in the input data or the model parameters. This can lead to more robust and consistent results. Lastly, it can improve the interpretability of the model by reducing the magnitude of the model parameters, which can make it easier to understand the relationship between the input features and the model's prediction. Therefore, implementing L2 regularization can improve the generalization stability, and interpretability of a CNN, leading to improved performance and insights(Tikhonov and Arsenin, 1977 [10]).

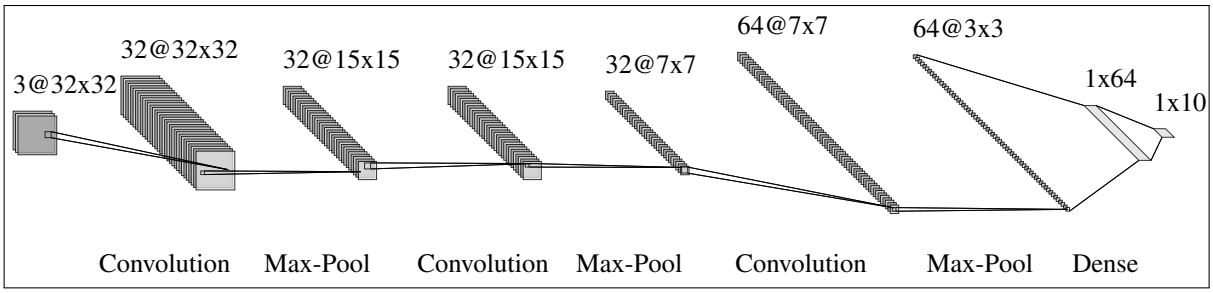


Figure 1: Architecture of the CNN

2.2.3 Batch Normalization

Firstly, implementing batch normalization in a CNN can help address the internal covariate shift problem by normalizing the activations of each layer to have zero mean and unit variance. This helps to stabilize the distribution of the activations and improve the speed and stability of training. Moreover, it can help reduce the overfitting of the data by adding some noise to the activations during training. This acts as a form of regularization, which can improve the generalization of the model to new data. Lastly, batch normalization can reduce the dependence of the model on the initialization of the weights, which can make it easier to train deeper networks. Finally, implementing batch normalization can improve the stability, generalization, efficiency, and depth of a CNN (Ioffe and Szegedy, 2015 [3]).

2.2.4 Dropout Layer

Implementing a dropout layer in a CNN helps reduce overfitting the training data by dropping out some of the neurons in the layer during training. This forces the remaining neurons to learn more robust features that are useful for classification, rather than relying on the co-adaptation of a small set of neurons to fit the training data. Secondly, dropout can act as a form of regularization, which can improve the generalization of the model to new, unseen data. Thirdly, dropout can help reduce the sensitivity of the model to small changes in the input, which can improve the stability and robustness of the model. Lastly, dropout can help reduce the dependence of the model on specific features or patterns in the input, which can make it more flexible and adaptable to a wide range of inputs. Therefore, implementing dropout can improve the robustness, generalization, stability, and flexibility of a CNN, leading to improved performance (Srivastava *et al.*, 2014 [9]).

2.3. OOD Detection

In order to detect out-of-distribution (OOD) samples, I searched for useful features in the deep neural network. Feature extraction is performed by obtaining the activations of a certain layer in the network. The function *"activations"* is used with the OOD detection data and the layer of interest, resulting in a matrix of features where each row corresponds to a data sample.

To determine useful features for OOD detection, a histogram is generated to visualize the distribution of the values of the mean activation across all features. Subsequently, a threshold is identified and chosen based on the distribution of the mean activation values.

To detect OOD samples, the mean activations of the data are compared to the threshold value. If the mean activation value is less than the threshold, the sample is classified as *"ood"*, otherwise, it is classified as *"id"*. The true labels of the OOD data are also collected to evaluate the performance of the detection method.

3. Experimental Evaluation

This section details the experiments I conducted to evaluate my approach and the parameters used in the model. Finally, I present visualizations of the impact of the improvements on the training process and the images, and I analyze the results of the ablation study and confusion matrices. I also include a visualization of a subset of test images to examine which ones were classified correctly and which were not.

3.1. Training Parameters and Improvements

The training of the final model was carried out using additional parameters with respect to the basic network. I trained the model using the Stochastic Gradient Descent with Momentum (SGDM) optimizer with a momentum of 0.9, and an initial learning rate of 0.0002. The learning rate was reduced by a factor of 10. Using a smaller initial learning rate can be advantageous for several reasons. Firstly, it can prevent overshooting of the optimal solution during training, which can occur with a larger value. Secondly, a smaller initial learning rate can improve the stability of the model during training and reduce the risk of overfitting the training data. Thirdly, it can enable the model to make

smaller and more precise weight updates, leading to more efficient convergence toward an optimal solution. Lastly, it can help prevent numerical issues that may arise at a larger rate, such as NaN values. Therefore, using a smaller initial learning rate can improve the precision, stability, and convergence of a CNN during training, ultimately leading to improved performance and generalization. Additionally, I used an L2 regularization of 0.004, a piecewise learning rate schedule with a drop factor of 0.1, and a drop period of 16 epochs. The model was trained for a maximum of 20 epochs with a mini-batch size of 128. I employed early stopping using the best-validation-loss strategy and validated the model using a validation set consisting of a fraction of the training data. Additionally, I used GPU acceleration to speed up the training process. To improve the performance of the models I also added two additional layers to the network. Namely, a dropout layer with a dropout factor of 0.2 and a batch normalization layer.

3.2. Impact of Improvements

During the training procedure, the model's progress was closely monitored, and its training and validation losses were visualized using a training-progress plot. In Figure 2a, the training process of the basic training shows a peak in the training loss, which can lead to the issues mentioned earlier. After implementing the improvements detailed in Section 2, the same peak is absent in Figure 2b. On the other hand, when the batch normalization layer is removed, Figure 2d shows that the same peak in training loss occurs, but in a smaller magnitude. This occurs because batch normalization reduces the network's dependence on initialized weights, which has an impact on the initial training loss.

Furthermore, in Figure 2c, the training starts slowly and then accelerates midway through the first epoch. This indicates the effect of the learning rate on network training. This phenomenon is possibly due to a learning rate that is too high, slowing down the convergence to an optimal solution.

Additionally, upon removing the dropout layer, it can be observed in Figure 2f that the training accuracy surpasses the validation accuracy at a faster rate compared to the other training configurations. This trend highlights the beneficial effect of the dropout layer in improving the model's ability to generalize to new, unseen data.

From the analysis of Figure 3, it can be observed that there is a consistent pattern across all the confusion matrices. Specifically, classes such as *"airplane"* and *"truck"* exhibit better performance compared to other classes. This trend could be attributed to the distinctive and easily recognizable shapes of objects belonging to these classes. Conversely, classes such as *"cat"*, *"dog"*, and *"deer"* exhibit the lowest accuracy among the 10 classes. This could be due to the model's inability to differentiate between animals or more specifically, its inability to accurately classify which animal is present in the image. This inference is supported by the fact that animals are mostly misclassified as other animals. For instance, cats are frequently mislabeled as dogs and vice versa. To further examine the network's behavior, a subset of incorrectly classified images was selected and presented in Figure 4. Similarly, Figure 5 displays correctly classified images. In order to analyze the reason behind misclassifications, the gradCAM visualization technique was employed. This technique uses gradients of the classification score in relation to the final convolutional feature map to identify the areas in the image that are most influential in determining the network score for a particular class. As shown in Figure 6, the network struggles to identify relevant features in certain misclassified images, which may explain the misclassifications. In contrast, Figure 7 illustrates that the network focuses on the legs of the deer, which may shed light on why animals are often misclassified, as the network may overly rely on certain features and struggle to differentiate between similar classes.

Table 1: Test accuracies

	Accuracy (%) ↑
Base	60.92
Improved	66.60
No Scheduler	62.10
No L2 Regularization	61.55
No BatchNorm	60.65
No DropOut	62.48

Table 2: Accuracy of the OOD detection method using different layers in the bignet neural network.

	Accuracy (%) ↑
relu_9	99.3
relu_10	100.0

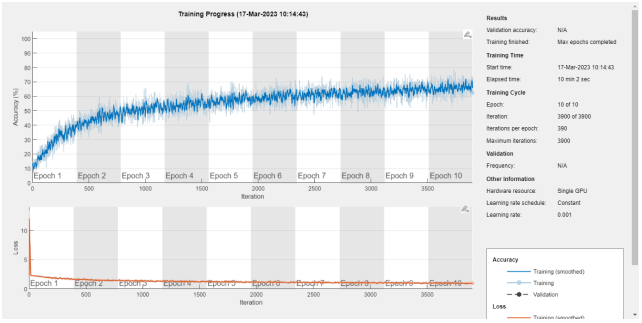
The accuracies achieved during training are shown in Table 1.

3.3. OOD Detection Results

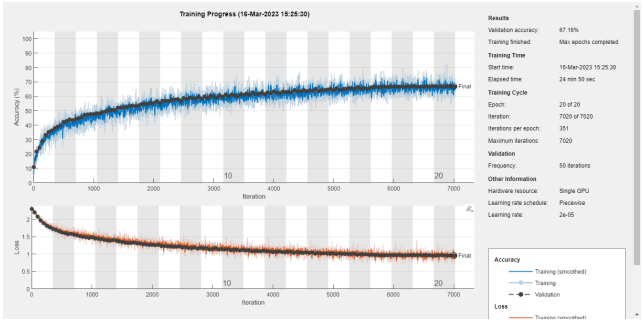
To evaluate the effectiveness of the OOD detection method using the *"bignet"* network features, I tested several different layers on the OOD detection dataset. Only two of the layers achieved high accuracies, these are shown in Table 2.

As shown in Table 2, the best accuracy of 100% was achieved with the *"relu_10"* layer. This indicates that the activations of the *"relu_10"* layer contain the most useful information for detecting OOD samples.

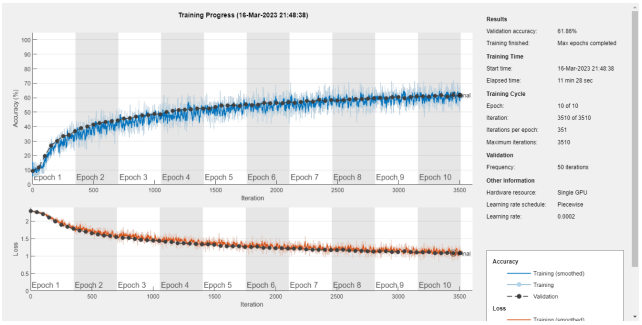
Figure 8 shows the histogram of the mean activation val-



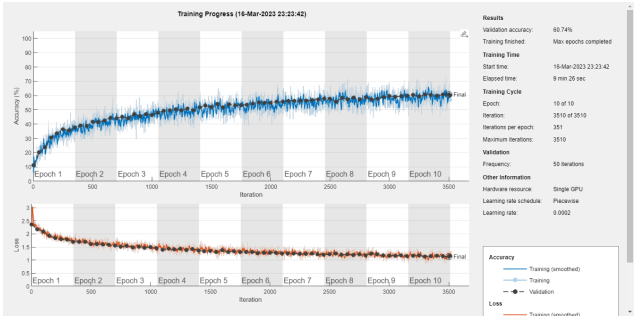
(a) Basic training of CNN



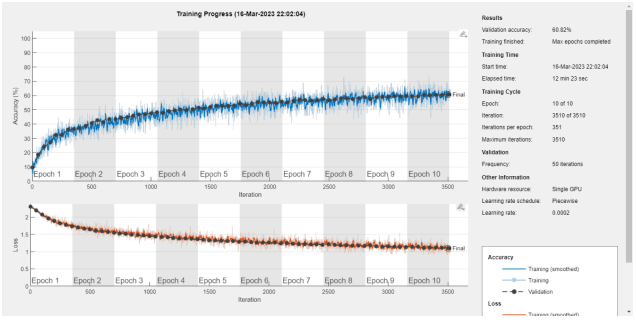
(b) Improved training of CNN



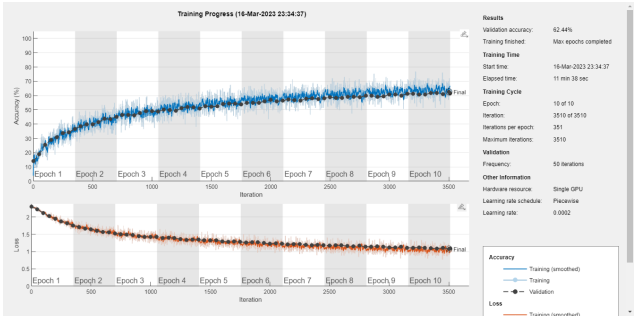
(c) Training without piecewise scheduler



(d) Training without batch normalization



(e) Training without L2 regularization



(f) Training without dropout layer

ues for the *"relu_10"* feature, where the x-axis represents the values of the mean activation and the y-axis represents the frequency of the activation values. The red line indicates the threshold value of 0.32 which was chosen based on the distribution of the values of the mean activation.

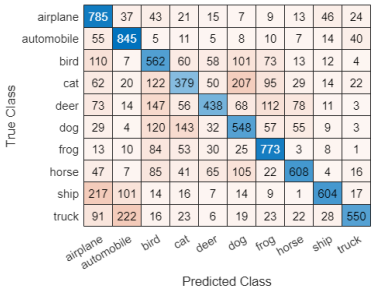
Some sample images detected as *"id"* and *"ood"* are shown in Figure 9. The left column shows OOD samples that were correctly classified, while the right column shows ID samples that were correctly classified as ID using the *"relu_10"* layer. It can be seen that the OOD sample is significantly different from the ID sample.

4. Theoretical Questions

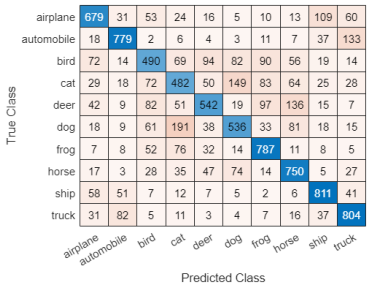
4.1. a

Fully Convolutional Neural Networks (FCNs) and Neural Networks with Fully Connected Layers (FNNs) are both types of neural networks commonly used in image-processing tasks. While they share some similarities, they differ in their architecture and purpose.

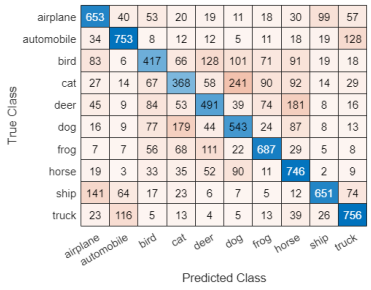
We specifically designed FCNs for image segmentation tasks, where the objective is to classify each pixel in an image according to the object or background it belongs to. Unlike FNNs, FCNs use only convolutional layers, which allows them to process images of varying sizes. This is because convolutional layers can capture local spatial information from an image and maintain it through the network. This makes them more suited for tasks that require pixel-



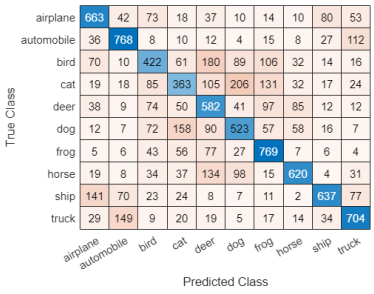
(a) Basic training confusion matrix



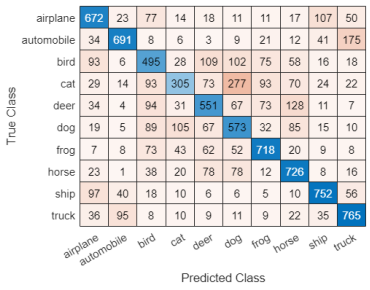
(b) Final training confusion matrix



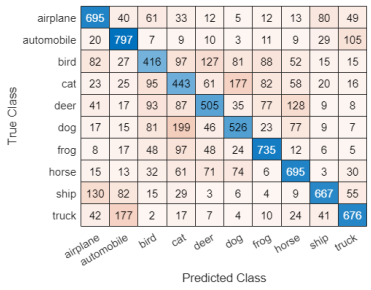
(c) No batch norm confusion matrix



(d) No scheduler confusion matrix



(e) No dropout confusion matrix



(f) No L2Reg confusion matrix

Figure 4: Wrongly predicted images by the final improved model

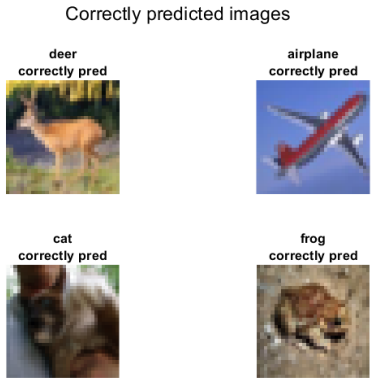


Figure 5: Correctly predicted images by the final improved model

level accuracy.

In contrast, FNNs consist of one or more fully connected layers, which connect every neuron in one layer to every neuron in the next layer. We typically use these layers for classification tasks, where the goal is to assign an input image to one of several predefined categories. FNNs can handle input images of any size, but they do not preserve spatial information as effectively as FCNs.

Fully connected layers are often utilized in Convolutional Neural Networks to perform the final classification or regression task. These layers take the output of the convolutional layers and flatten it into a one-dimensional vector, which is then fed into one or more fully connected layers. These layers can learn high-level representations of the input features, which are then used to make predictions.

FCNs are preferred for tasks where pixel-level accuracy is critical, such as in medical imaging, semantic segmentation, or object detection. FNNs are more suitable for classification tasks, such as image recognition, sentiment analysis, or natural language processing.

While adaptive pooling is a technique used in convolutional neural networks to reduce the spatial dimensionality

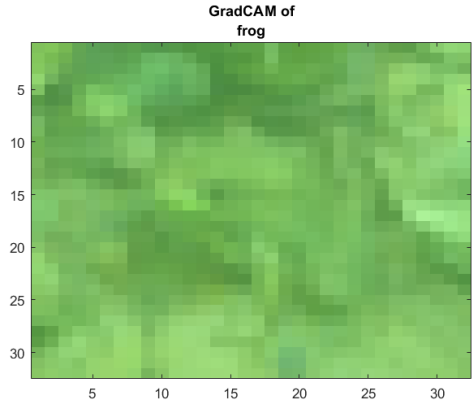


Figure 6: gradCAM of wrongly classified image

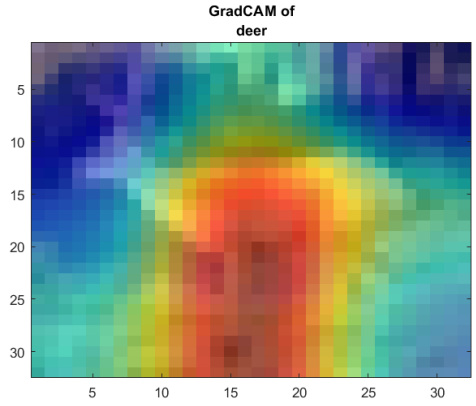


Figure 7: gradCAM of correctly classified image

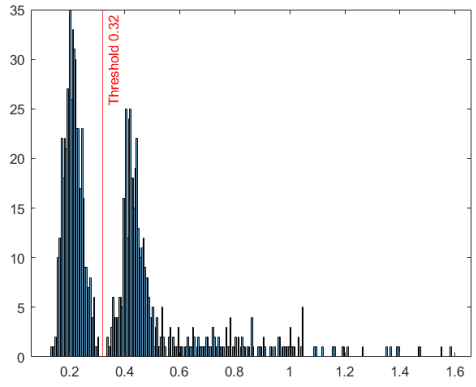


Figure 8: Histogram of the mean activation values of the *"relu_10"*. The red line indicates the threshold.

of the feature maps while preserving important information, it cannot always replace FCNs. This is because adaptive pooling may result in a loss of spatial resolution, which is

Figure 9: Sample images detected as ID and OOD using *"relu_10"* layer

crucial in certain tasks. Thus, FCNs remain the preferred option for tasks that require pixel-level accuracy (Long *et al.*, 2015 [8]).

4.2. b

Over-fitting is a well-known problem in machine learning that occurs when a model becomes too complex and starts to fit the noise in the training data rather than the underlying patterns and generalizations. This results in poor performance on new, unseen data, as the model has memorized the training data instead of learning to generalize from it. Various factors can cause Over-fitting, including a lack of training data, a complex model architecture, and the use of numerous features.

If the model does not receive enough examples to learn generalizable patterns, the issue of a lack of training data arises. In such cases, the model may learn to fit the noise in the data, leading to over-fitting. A complex model architecture can also contribute to over-fitting, as a model with many parameters can easily fit to the noise in the data. Similarly, using too many features can lead to over-fitting, as the model may fit the noise in those features rather than the underlying patterns.

We can use two common methods to combat over-fitting. Regularization is a technique that involves adding a penalty term to the loss function of a model, which discourages the model from fitting the noise in the data and encourages it to learn generalizable patterns. There are two main types of regularization: L1 and L2 regularization. L1 regularization adds a penalty term proportional to the absolute value of the model weights, while L2 regularization adds a penalty term proportional to the square of the model weights.

Dropout is a technique where a certain percentage of neurons in a layer are randomly ignored during training. This forces the model to learn redundant representations of

the input data, preventing it from relying too heavily on any one feature or neuron. Dropout can be applied to the input layer, hidden layers, or both.

In summary, over-fitting is a common problem in machine learning that various factors can cause. Regularization and dropout are two popular techniques used to combat over-fitting by encouraging the model to learn generalizable patterns and preventing it from relying too heavily on any one feature or neuron (Srivastava *et al*., 2014 [9]).

4.3. c

The Scale-Invariant Feature Transform (SIFT) descriptor is a popular hand-crafted feature representation for describing image patches that is robust to scale, rotation, and illumination changes. However, the design and tuning of a SIFT feature descriptor can be a time-consuming process and may not always result in optimal performance. In contrast, Convolutional Neural Networks (CNNs) offer an alternative approach that can learn feature representations automatically from the data.

To design and train a CNN for the same purpose as SIFT, one can use a convolutional architecture consisting of multiple convolutional layers followed by a set of fully connected layers. During training, a dataset of image patches labeled with the object or feature of interest can be used to optimize the network to learn a set of filters that are robust to scale, rotation, and illumination changes. These learned filters can be used to extract feature representations from new image patches for tasks such as object recognition or image retrieval.

One advantage of using a CNN over a hand-crafted feature descriptor like SIFT is that CNNs can learn feature representations that are optimized for the specific task at hand, whereas SIFT relies on a fixed set of hand-crafted features that may not be optimal for all scenarios. Additionally, CNNs can learn features that are robust to variations in scale, rotation, and illumination without the need for additional preprocessing steps.

However, one disadvantage of using a CNN is that they typically require a large amount of training data to learn optimal feature representations. Additionally, CNNs can be computationally expensive to train and evaluate, especially when compared to simpler hand-crafted feature descriptors like SIFT. Finally, CNNs may also suffer from the problem of overfitting, where the model memorizes the training data instead of learning to generalize to new, unseen data.

In summary, while SIFT is a widely used and effective hand-crafted feature descriptor for describing image patches, CNNs offer an alternative approach that can learn optimal feature representations automatically from the data. The advantages of using a CNN include their ability to learn features optimized for the specific task at hand and their robustness to variations in scale, rotation, and illumination. However, CNNs may require a large amount of training data, be computationally expensive, and suffer from overfitting (Hinton & Krizhevsky, 2012 [2]).

5. Conclusions

In this report, I described my work on the design and evaluation of a Convolutional Neural Network. Overall, the model achieved 66% accuracy on the test set.

References

[1] P. Cunningham, M. Cord, and S. J. Delany. Supervised learning. *Machine learning techniques for multimedia: case studies on organization and retrieval*, pages 21–49, 2008. 1

[2] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 2, 8

[3] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015. 3

[4] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018. 2

[5] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009. 1

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. 2

[7] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[8] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 7

[9] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 3, 8

[10] A. N. Tikhonov, V. J. Arsenin, V. I. Arsenin, V. Y. Arsenin, et al. *Solutions of ill-posed problems*. Vh Winston, 1977. 2

[11] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer, 2014. 2