Tracing the Convolutions of a CNN

Frieda Wik

December 2023

1 Abstract

The integration of convolutional operations within Neural Networks has contributed to important advancement in image and pattern recognition, paving the way for the success of Convolutional Neural Network (CNN) architectures. In this study we classify the CIFAR-10 image dataset while attempting to give insights in the effect of the convolutional operation by visualising kernels weights and their corresponding feature maps. Our best CNN model outperformed the Feed-Forward Neural Network (FFNN) model in terms of accuracy and misclassification rates. The accuracy score reached 71% for the CNN model, compared to 52% for the FFNN model. The filters of the CNN and their associated feature maps converged to their final pattern as the model approached its peak accuracy. This study explores the significance of convolutional operations in image classification tasks, and takes a first step in the quest for making such methods more interpretable. On the way, we discover the need for more advanced methods, which leads us to the field of explainable AI.

2 Introduction

Convolutional Neural Networks (CNN) have become a fundamental tool for image classification tasks [2]. According to the Oxford Languages dictionary, a convolution is "a thing that is complex and difficult to follow" [8]. In mathematics, the convolution is defined as a binary operation that combines two functions into a third function that represents how the two first functions influence each other. Even though the convolution in mathematical terms has a concrete and well-defined meaning [1], its effect in a Neural Network can become difficult to follow as the images become more complex and the network grows deeper. In this project we explore the effect of various layers of CNNs by visualising the training process of the filters and their corresponding feature maps during classification. We use the classical CIFAR-10 dataset and we apply both a simple LeNet-5 inspired CNN and a fully connected Feed- Forward Neural Network (FFNN) without any convolutional layers. We successfully extract and visualise the various kernels and layers of the CNN model, but we realise that more sophisticated methods are needed to be able to understand the decision making process of the CNN.

This report is structured as follows: In the method section we describe the CNN and its different layers, along with a presentation of the CIFAR-10 dataset. In the results section we present a visualisation study of the convolutional layers together with the relevant performance metrics of the classification result of both the FFNN and CNN models. In addition, an animation tool for visualising the training process of any chosen layer or filter within the CNN can be found at the GitHub repository of the project: https://github.com/friedawik/FYS-STK4155-. Finally, in the discussion and conclusion sections we present an analysis and some conclusions from the various analyses performed in the study.

3 Methods

In this project a fully connected FFNN and a simple CNN was applied to study the CIFAR-10 image dataset. Since the datapoints of an image are structured in a grid where neighbour pixels often are correlated to each other, CNNs are in generally more suitable for image analysis than the FFNN. This is because the CNN can exploit two features important for object classification. First, the *translation invariance* implies that the classification needs to be independent of the position of the object in the image. Secondly, the *locality* refers to the fact that the object should be classified independently of pixels far away [7]. As we will see, the convolutional operation allows for both these features to be accounted for. Since the main motivation for the study was to explore the effect of the convolutional operation, an effort was applied in visualising the different filters, hoping that this would give insight in the learning process of the CNN. In general, less attention was paid to finding the best model. Some overarching parameter search was performed to find the best architecture and optimisation parameters, but to really compete with the best published results one would need to use more modern and deeper networks. Instead, an architecture inspired by the classic LeNet-5 network was used. This CNN was proposed by the LeCun et al. in 1998 [5] and it is by many regarded as the beginning of the development of CNNs for image analysis. This network was designed to classify handwritten digits and is considered too simple the classification of more complex images. In 2009 the image dataset CIFAR-10 was published by Krizhevsky, who later introduced AlexNet which was one of the first deep networks for image classification [4].

3.1 Convolutional layer

The convolutional layer is the core component of the CNN architecture. In a convolutional layer, multiple kernels scan the input data and compute the dot products with the local regions they cover. The purpose of this operation is to extract specific features as the kernels learn weights during training. From the mathematical viewpoint, the convolution can be described as an operation that combines two functions to produce a third function that represents how one function influences or modifies the other. If two functions f and g are continuous and take real valued inputs, the convolution between them is defined as

$$(f * g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)\,dt, \in \mathbb{R}. \tag{1}$$

The convolutional operation is visualised in Figure 1. In this example, the kernel W is of dimension $3x3$ and it moves across the input data x where it performs an elementwise multiplication resulting in the convolutional layer z. Thus, each entry in z is connected to a small region of pixels called its receptive field and the convolution converts this field into a single value. This allows the layer to capture local patterns and spatial relationships. The same set of learnable filters is used across different regions of the input. This concept of weight sharing is crucial in CNNs, as it enables the model to recognize similar patterns in different parts of the input, thus allowing for the earlier discussed translational invariance. The parameter sharing also reduces the number of parameters in the network and improves generalization [7]. The kernel is typically smaller than the input and therefore the filter slides horizontally and vertically across the input. The stride s specifies how many pixels or units the filter shifts at each step. If the stride value is set to $s = 1$, the filter moves one pixel at a time. This means that the filter processes each neighboring element and overlaps the neighboring receptive fields, as illustrated in Figure 1. Using a larger stride can reduce the computational cost and memory requirements of the network as it results in downsampling of the input data. However, it may result in a loss of spatial resolution and possible loss of fine-grained features. The output size resulting from the convolution performed with a squared kernel can be calculated by equation 2.

$$n_{out} = \frac{n_{in} - k + 2p}{s} + 1 \tag{2}$$

Here, n_{out} is the number of output features, n_{in} is the number of input features, k is the kernel size, p is the padding described in section 3.2 and s is the stride. It should be noted that although the kernel operations are called convolutions in this text, most CNN libraries use the related cross-correlation operation instead. Since the only difference is the whether the kernel is flipped or not, the learning process of the two operations are equivalent [7].

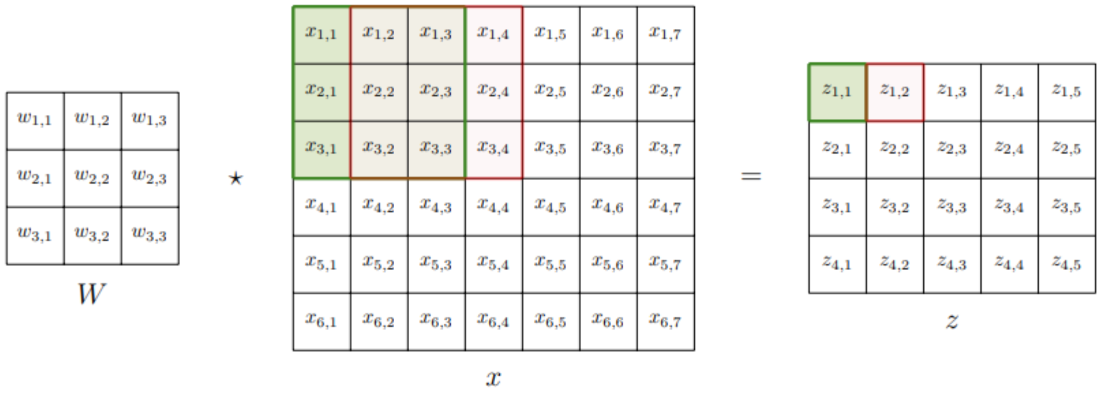


Figure 1: Illustration of a 3×3 kernel W that is convolved with the input x to produce the feature map z. Note that CIFAR-10 images have three channels (RGB) and therefore there would be three separate kernels W that are convolved with x and summed up to produce the first feature map z. Illustration taken from [7].

3.2 Padding

Padding is the process of adding extra empty border pixels around an input image and it is typically applied before the convolutional layers. It helps counteract the loss of information that occurs during convolution and pooling operations. Since the pixels at the border of the image are less represented in the resulting feature map, the filters may start to focus too much on the boundaries. By adding padding, the network can better capture information from the edges and corners of the image and thus avoid some of the boundary effects. As described by equation 2, the padding maintains the size of the feature map. For example, if the filter size is 3x3, adding a padding of one element to each side of the input would maintain the input size. Without padding, the size of the feature map would reduce after each convolution operation. Figure 2 shows an illustration of padding when $p = 1$.

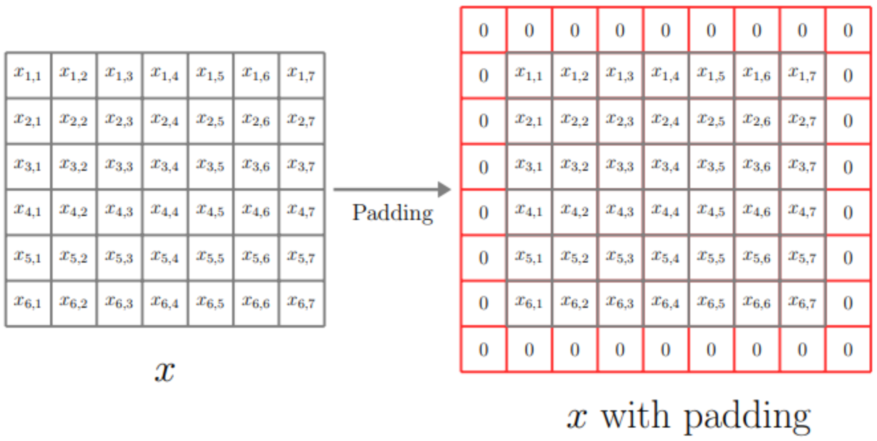


Figure 2: Visualisation of padding of $p = 1$. Illustration taken from [7].

3.3 Dilation

Dilation is the technique of adding space between the elements of the filters. The size of the spacing is given by the dilation factor. For example, a dilation factor of 1 means no spacing, while a rate of 2 means that a zero is added between each element. The larger the dilation rate, the larger the gaps between the filters. By using dilation in a CNN, the receptive field of the filters increases without down-sampling the input feature map. This can be beneficial because it allows the network to capture larger contextual information while maintaining the spatial resolution of the input. It can also reduce the computational cost compared to a

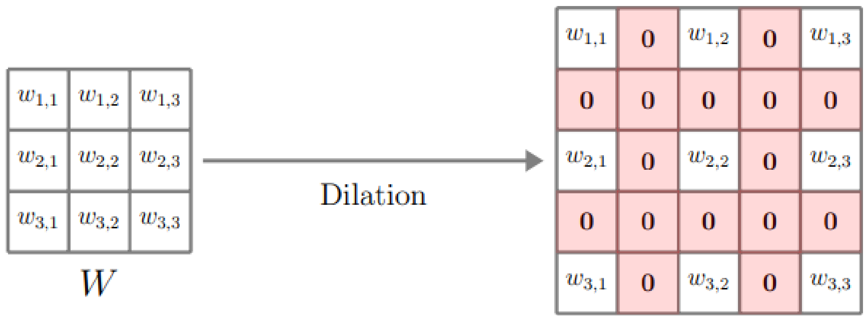


Figure 3: Illustration of dilation with dilation factor 2. Illustration taken from [7].

regular convolution with a larger filter size. In this project no dilation was used.

3.4 Pooling

Pooling is a technique for spatial downsampling. It helps reduce the dimensionality of the feature maps while retaining the most relevant information. Several methods of pooling exist, with average pooling and max pooling being common choices. In this project only max pooling was used since it is considered to perform better in practise [6], but both methods are illustrated in Figure 4. In max pooling, a pooling window slides over the input feature map and selects the maximum value within the window. Since maxpooling only takes the maximum value within a region, it does not take into account the precise position of the feature. As long as the feature is present within the region, the output of the maxpooling operation will be the same. This increases the position tolerance needed for translation invariance [7]. By reducing the spatial dimensions of the feature maps, pooling reduces the number of parameters and computational complexity in subsequent layers, making the model more efficient.

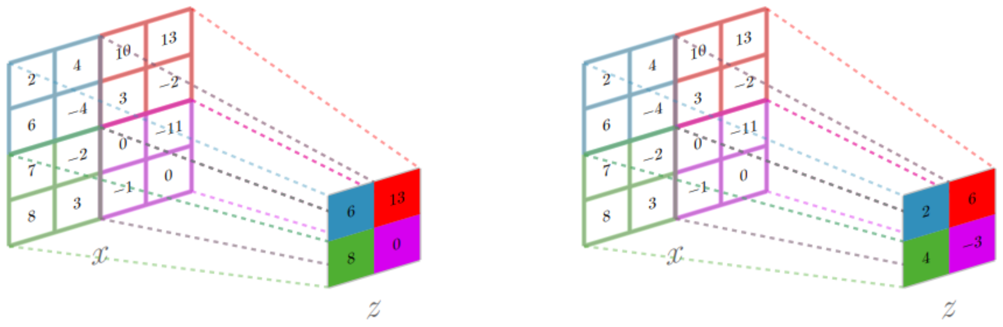


Figure 4: Illustration of max pooling (left) and average pooling (right). Illustration from [7]

3.5 The CIFAR-10 dataset

The CIFAR-10 dataset was downloaded through the PyTorch library. This dataset is a well- balanced image classification dataset with 10 classes, each representing a specific category of objects [3]. Each class contains 6,000 images, split evenly among the training and test sets. The training set consists of 50,000 images, while the test set contains 10,000 images. Each image is of size 32x32 pixels, and each pixel is represented by three color channels (RGB). The images were normalised in the range $[-1, 1]$ using the PyTorch function torchvision.transforms.Normalize().

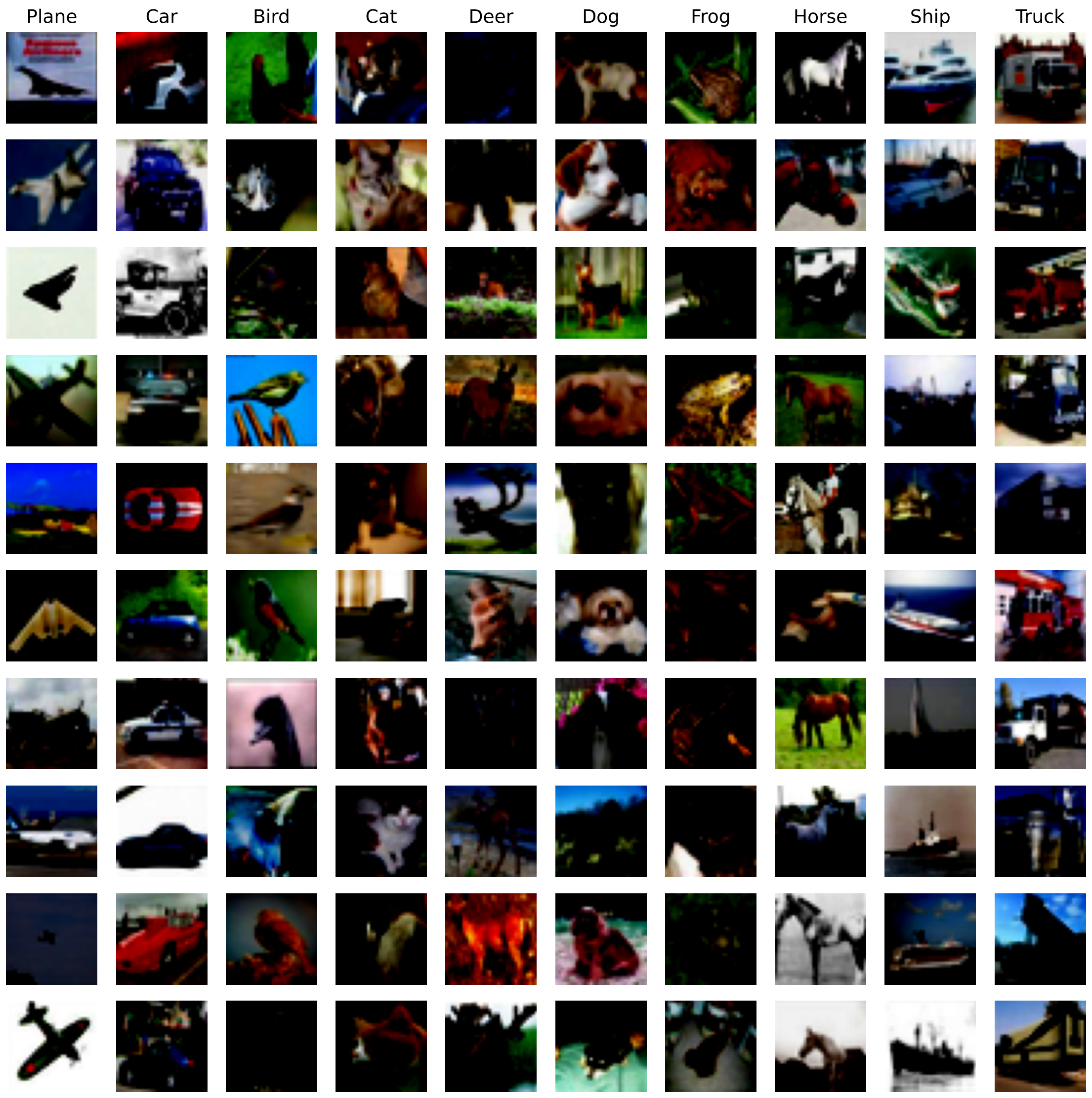


Figure 5: 10 random images from all 10 classes in the CIFAR-10 dataset.

Ten example pictures of every class are shown in Figure 5. The covariance matrix of a subset of flattened images from the dataset can be seen in Figure 6. Each entry in this matrix represents the covariance between two specific pixels in the dataset, where the first 1024 pixels correspond to the red channel, the middle 1024 pixels correspond to the green channel and the last 1024 pixels correspond to the blue channel. The diagonal elements represent the variances of individual pixel values. A higher variance indicates greater variation in pixel values, suggesting more diversity in colors or patterns within the dataset. The off-diagonal elements of the covariance matrix represent the covariances between pairs of pixel values. High covariances suggest that two pixel values tend to increase or decrease together, while zero covariance suggest that there is no relationship between the pixels. As can be seen, pixels that are close to each other are strongly correlated, while pixels far apart are not. This is also true across the three color channels, indicating that a pixel value of one color channel often is accompanied by a related value in the other colors. Also, pixels are typically more similiar to other pixels in the same column or row than to pixels in other columns or rows. These patterns may be attributed to typical choices in photography, where one may strive for symmetry within the image frame [3].

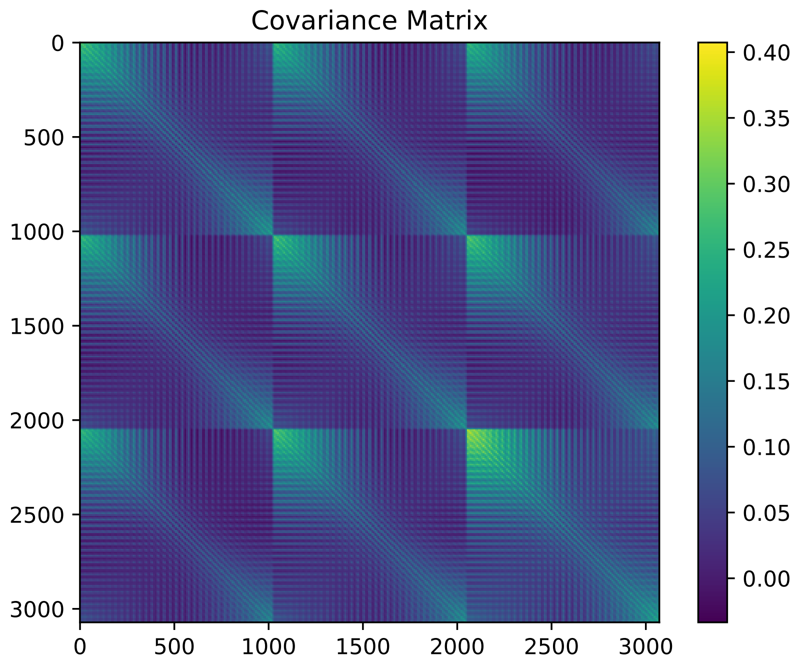


Figure 6: The covariance matrix of a subset of 6000 images of the CIFAR-10 dataset. The images are flattened and the pixels are indexed in row-major order such that the first $32x32 = 1024$ pixels correspond to the red channel, the next 1024 pixels correspond to the green channel and the last 1024 pixel correspond to the blue channel.

3.6 The FFNN and CNN models

For both models, the Softmax function was used as output function and the Cross-entropy function was used as loss function. The Adam method was used for optimization during backpropagation, and a grid search was performed to find the best learning rate and decay rate. The FFNN model was build with one hidden layer of 100 nodes. The architecture of the CNN is given in Table 1. All models were build using the PyTorch library. The weights were initialized using the default method, which was the Kaiming Uniform for the CNN. The weights of the CNN model were stored after each epoch and the resulting layers were plotted. The GPT UiO chat was used throughout the whole project, but perhaps mostly to build and understand the CNN model in PyTorch. An example chat can be found in the GitHub repository of the project.

Table 1: The architecture of the CNN used in this study.

Layer	**Type**	**Hyperparameters**
1	Convolutional	32 filters of size 5×5
2	Relu	N/A
3	Maxpool	Size 2×2
4	Convolutional	64 filters of size 5×5
5	Relu	N/A
6	Maxpool	Size 2×2
7	Fully connected	N/A

4 Results

The CIFAR-10 test data was classified using both the FFNN and CNN models. A grid search was performed to find the best learning rate and decay rate for both models. The search for the best hyperparameter is documented in the GitHub repository of the project. The results of the two best models can be seen in Figure 7, where the test accuracy increased rapidly during the first epoch and continued an upwards trend until epoch 5. After this, the test accuracy stabilised and only the train accuracy continued to increase slowly. The training was continued to epoch 30 to enable visualisation until the pattern of the kernels stabilised. Even though the test accuracy stagnated, it did not start declining, which would be a clear sign of overfitting. However, as we will see later, the second convolutional layer showed signs of overfitting as its weights approached zero.

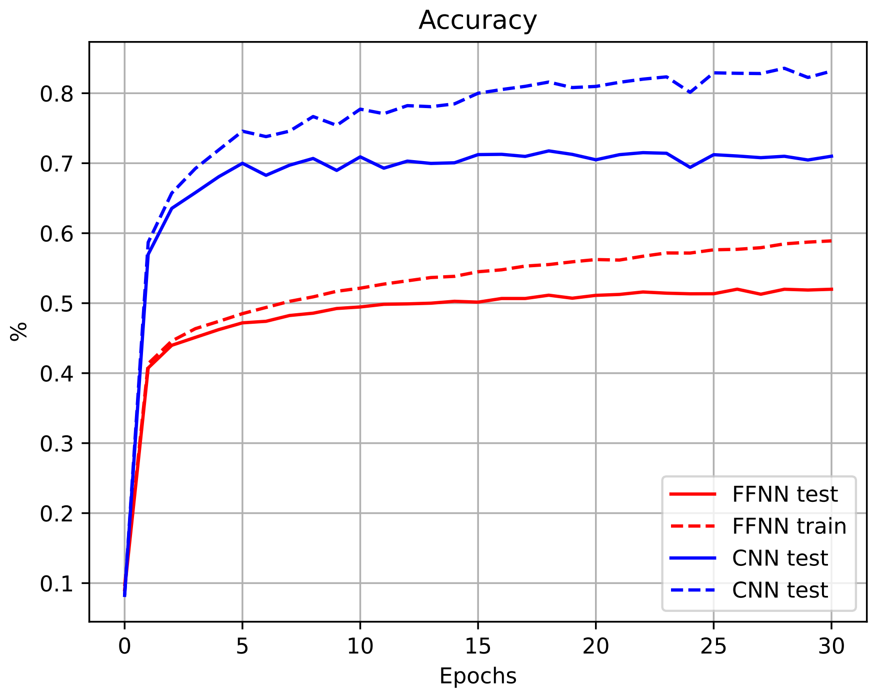


Figure 7: Convergence for the FFNN and CNN models on the CIFAR-10 dataset.

4.1 Visualisation analysis

A picture of a peacock was chosen for the analysis of the CNN layers. The peacock is a large and colorful bird known for its extravagant plumage. In the chosen picture, the peacock keeps its tail closed and the contour of a long neck is seen against the green background (see Figure 8). The various CNN layers were plotted during the 30 epoch long training to see the evolving pattern as the model is learning. The first convolutional layer, ReLU layer and max pooling layer are presented in this report, while the rest of the layers can be found in the GitHub repository. All weights were rescaled between 0 and 1 prior to plotting. During training, the kernels stride along the image and produce feature maps. Figure 9 shows the evolution of one of the 32 kernels used in the first convolutional layer. The top-left image corresponds to the kernel before any training, and the subsequent images correspond to the filter at every second epoch until the end of the simulation. Before training, the filter pattern has a random appearance. Already at epoch two, a pattern has emerged, but it is not until epoch 16 that it becomes indistinguishable from the last epoch. For every kernel, a separate feature map is produced. Figure 10 shows the evolution of the feature maps corresponding to the kernel above. Again, in the top-left corner we can see the feature map prior to any training. Already at epoch 2, most of the prominent features seen at later epoch have emerged. Until epoch 16 some changes in pattern can be observed, but then the pattern converges. Figure 11 shows the evolution of the first ReLU layer. The ReLU function introduces non-linearity and sparsity by eliminating negative values. This can be observed directly in the resulting images, since the layers are more sparse in comparison to the feature maps above. In these layers, the bird's head is the prominent feature, along with some pixels along the lower-left part of the image. Figure 12 shows how the first maxpool layer changes during training. Even though the images are down-sampled, the prominent features of the ReLU layer are still present.

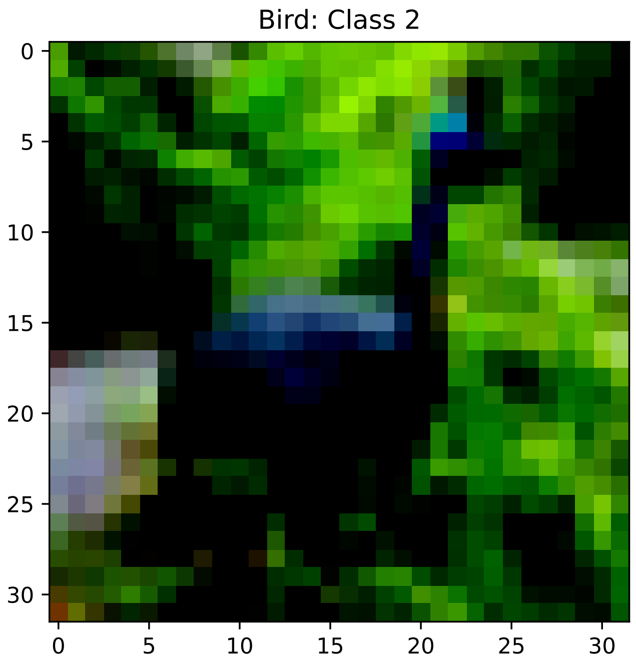
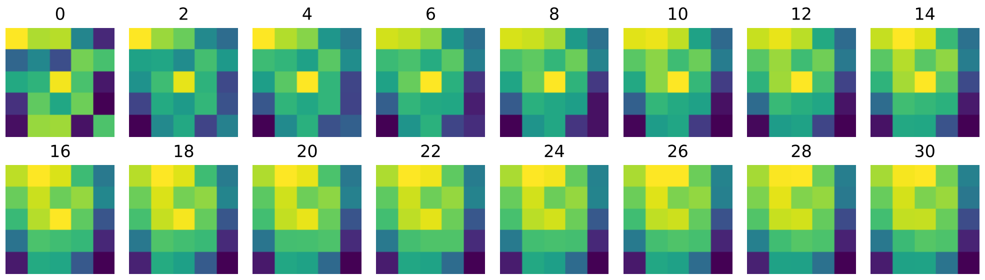
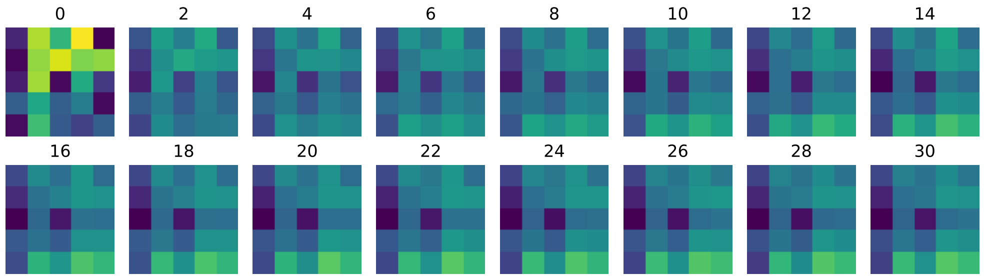


Figure 8: A 32 × 32 pixel image from the CIFAR-10 dataset used for further analysis.

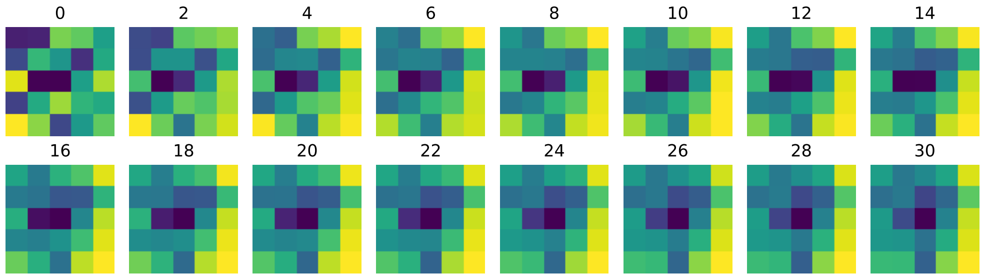
We can only speculate in what feature this specific kernel contributed to in the final decision when classifying the peacock as a bird (which it did correctly). Perhaps it extracted some part of the long bird neck and beak, as this is what can be seen in the the resulting feature maps. However, one must remember that this is only one kernel out of the 32 kernels of the first convolutional layer. The second convolutional layer has 64 kernels, and the model will decide on a class based on the full model. In addition, the same kernels are used to classify all 10 classes of objects, so it is reasonable to believe that this specific kernel will lead to very different patterns in other images. The second convolutional layer, ReLU layer and maxpool layer were plotted in a similar manner and can be found in the GitHub repository. The weights of the second kernel approached zero as training proceeded, and resulted in the last epochs producing feature maps containing entries with NaN. Having too many filters can allow the network to learn overly complex and specific patterns from the training data, which may result in overfitting. More importantly for the visualisation study, it can become challenging to interpret and understand the learned representation in each layer when there is an excessive amount of filters.



(a) The red channel of the kernel.



(b) The green channel of the kernel.



(c) The blue channel of the kernel.

Figure 9: A randomly chosen kernel of the first convolutional layer. The kernels har a depth of tree, where each dimension correspond to a colour channel.

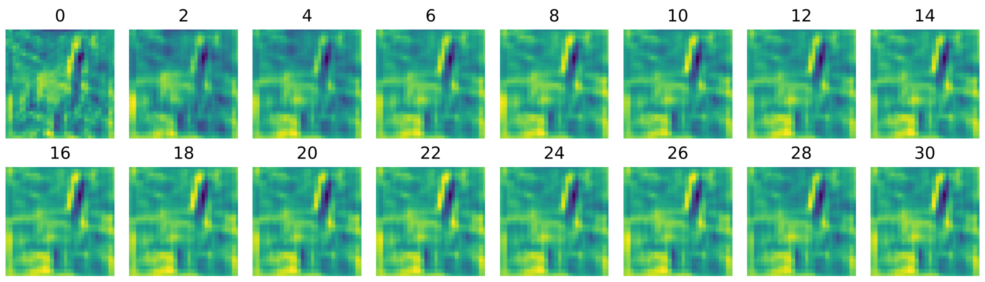


Figure 10: The feature maps corresponding to the kernel above during training.

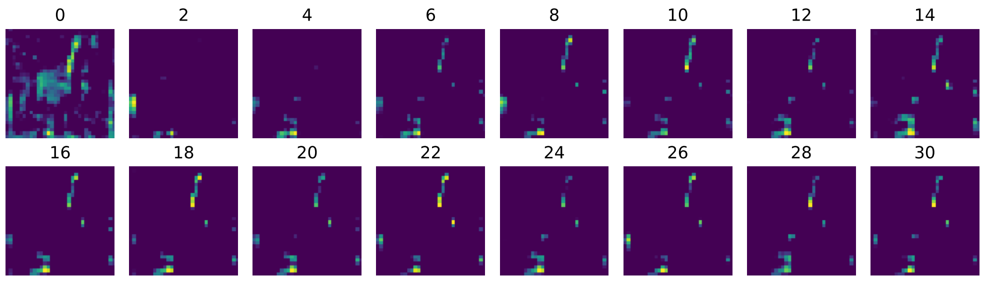


Figure 11: The first Relu layer during training.

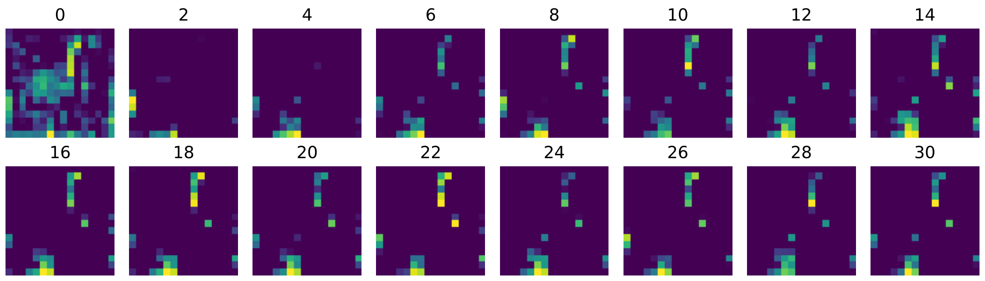
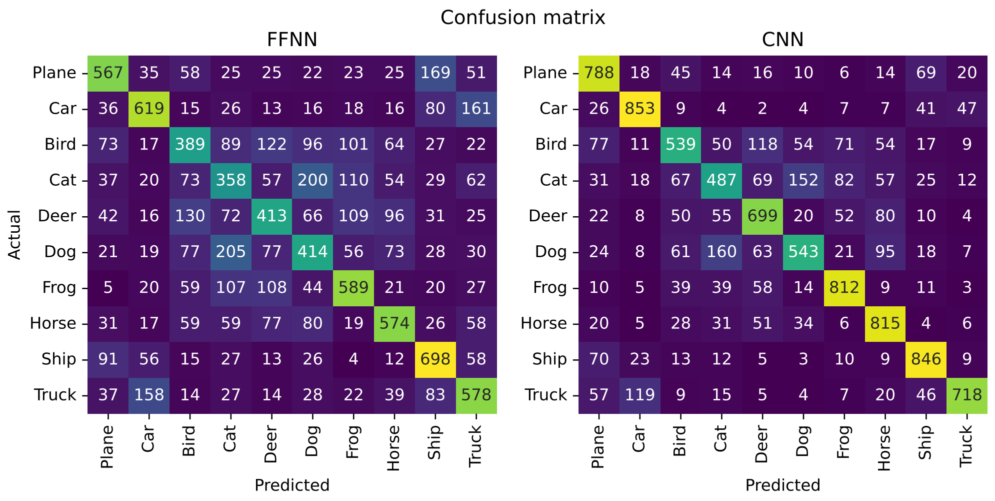


Figure 12: The first maxpool layer during training.

4.2 Classification results

The true and predicted class labels of the FFNN model (left) and the CNN model (right) are summarised in Figure 13. Each row in the confusion matrix represents the true class of the images, while each column represents the predicted class. The entries on the diagonal give the number of correct predictions, while the non-diagonal numbers correspond to the incorrectly predicted images of each class. In general, the CNN model performs better than the FFNN model. As expected, there were more wrongly predicted images when the object have higher resemblance with each other, for example the animal images were more often confused with each other than with the vehicle images. For instance, from the 1000 cat images, the FFNN model incorrectly identified 200 as a dog, while the CNN did the same for 152 images. Amongst the same 1000 cats, only 20 (FFNN model) and 18 (CNN model) were incorrectly classified as cars.

The trade-off between the true positive rate and the false positive rate was plotted in the ROC (Receiver Operating Characteristic) plot in Figure 14. To produce this plot, the multiclass problem was translated into a multiple binary classification tasks using the one-vs-all approach. This means treating each class as the positive class and the rest as the negative class in separate binary classification problems. In general, the CNN model resulted in curves closer to the top-left corner and corresponding higher Area Under Curve score (AUC) than the FFNN model. Both models resulted in similar plots, with the cars being the most easily classified object and the cats needing the most false positives to classify all cats correctly.



FFNN

Actual \ Predicted	Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Plane	567	35	58	25	25	22	23	25	169	51
Car	36	619	15	26	13	16	18	16	80	161
Bird	73	17	389	89	122	96	101	64	27	22
Cat	37	20	73	358	57	200	110	54	29	62
Deer	42	16	130	72	413	66	109	96	31	25
Dog	21	19	77	205	77	414	56	73	28	30
Frog	5	20	59	107	108	44	589	21	20	27
Horse	31	17	59	59	77	80	19	574	26	58
Ship	91	56	15	27	13	26	4	12	698	58
Truck	37	158	14	27	14	28	22	39	83	578

CNN

Actual \ Predicted	Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Plane	788	18	45	14	16	10	6	14	69	20
Car	26	853	9	4	2	4	7	7	41	47
Bird	77	11	539	50	118	54	71	54	17	9
Cat	31	18	67	487	69	152	82	57	25	12
Deer	22	8	50	55	699	20	52	80	10	4
Dog	24	8	61	160	63	543	21	95	18	7
Frog	10	5	39	39	58	14	812	9	11	3
Horse	20	5	28	31	51	34	6	815	4	6
Ship	70	23	13	12	5	3	10	9	846	9
Truck	57	119	9	15	5	4	7	20	46	718

Figure 13: The confusion matrix of the CIFAR-10 classification results.

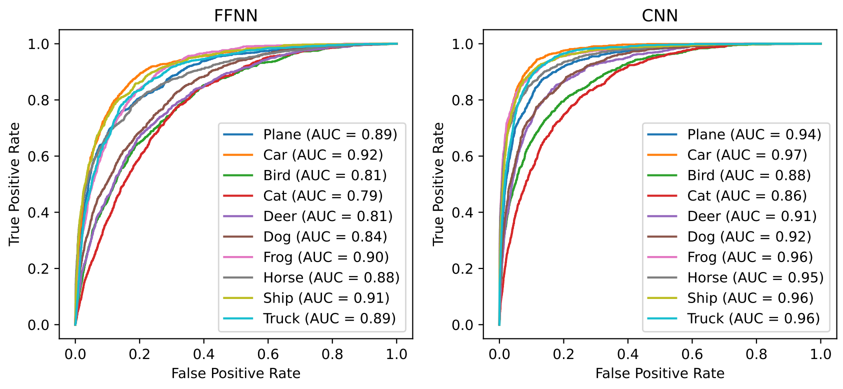
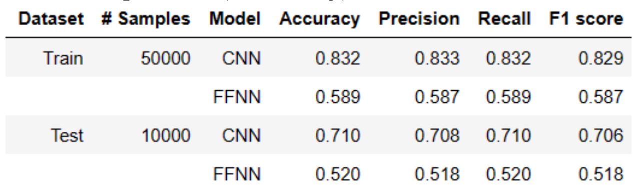


Figure 14: The one-vs-rest ROC curves for all image types.

The performance metrics of the FFNN and CNN are given in Table 2. The CNN model reached an accuracy of 71%, while the FFNN model reached an accuracy of 52%. The CNN model also scored better on precision, recall and F1 score. The f1 score was included even though the CIFAR-10 dataset is balanced with equally many object from each class.

Table 2: The precision, accuracy, recall and F1 score metrics.



Dataset	# Samples	Model	Accuracy	Precision	Recall	F1 score
Train	50000	CNN	0.832	0.833	0.832	0.829
		FFNN	0.589	0.587	0.589	0.587
Test	10000	CNN	0.710	0.708	0.710	0.706
		FFNN	0.520	0.518	0.520	0.518

5 Discussion

The visualisation study was performed in the hope of gaining insights in the decision making process imposed by the convolutional operation. However, the approach of simply looking at the kernels to extract information proved too naive as it was not possible to establish a link between the kernel pattern and the resulting feature maps. Not only would one need to study all kernels of all convolutional layers in the network but also one would need to do so for all object classes in the dataset to ensure that all information is considered. This is obviously not a feasible task, and the challenge of rendering CNNs more comprehensible to the user asks for a more systematic and quantitative approach. The field of explainable AI has emerged for answering the need of understanding the decision making of Neural Networks.

The kernel weights of the second convolutional layer approached zero as the training proceeded. If the weights of a kernel approach zero, it indicates that the kernel is not contributing significantly to the feature extraction process. To find the reason for this, the network architecture should be reevaluated by performing a search over architecture parameters. For example, the number of kernels in the convolutional layers may have been high, as they were significantly higher than a similar set-up in the PyTorch documentation [9].
The accuracy scores achieved in this study are by no means competing with the top performing architectures presented by other authors. However, these networks are often deep and complex, and thus harder to interpret. In this project, the focus was on visualising and interpreting the effect of the convolutional operation on the input image. As such, it could be seen as a preliminary tentative to an explainable AI method, as its goal was to make the decision making more transparent and understandable. There was therefore a trade-off between having a model that was complex enough to learn relevant patterns, but simple enough for the visualisations to be interpretable.

6 Conclusion

The CIFAR-10 image dataset was classified using both a FFNN and a CNN network. Even though the LeNet-5 inspired CNN architecture used in this project is considered too simple to classify the low resolution and complex images in this dataset, the CNN outperformed the FFNN by good margin. The CNN model reached an accuracy score of 71%, while the FFNN reached an accuracy score of 52%. The CNN model converged after the fifth epoch but the kernels in the convolutional layer continued to change slightly as training continued. This was expected as the model was still learning from the data, as indicated by the increasing training accuracy. The aim of this project was to get insight in the inner workings of a CNN by visually studying the kernel weights. While each filter and corresponding feature map could easily be visualised and analysed separately, this approach did not allow for a systematic interpretation of the overall effect of the convolutional operations. To gain such insights, one may need to turn to explainable AI methods.

A natural continuation of this project is to apply explainable AI methods:

Local explanations: Techniques like LIME (Local Interpretable Model-Agnostic Explanations) and SHAP (Shapley Additive Explanations) can provide local explanations by approximating the behavior of the AI model in a interpretable way.
Interactive visualisation: Techniques like saliency maps and attention maps can be used to visualize the internal workings of the CNN.
Feature importance: Methods like feature attribution and sensitivity analysis can provide insights into the contribution of each feature.

References

[1] Julius Berner et al. “The Modern Mathematics of Deep Learning”. In: *Mathematical Aspects of Deep Learning*. Cambridge University Press, Dec. 2022, pp. 1–111. ISBN: 9781316516782. DOI: 10.1017/9781009025096.002. URL: http://dx.doi.org/10.1017/9781009025096.002.

[2] Morten Hjorth-Jensen. *Lecture notes in Applied Data Analysis and Machine Learning*. Dec. 2023. URL: https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html.

[3] Alex Krizhevsky. “Learning Multiple Layers of Features from Tiny Images”. In: (2009), pp. 32–33. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[5] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[6] Fei-Fei Li. *Lecture Notes in CS231n: Deep Learning for Computer Vision*. Dec. 2023. URL: http://cs231n.stanford.edu/.

[7] B. Liquet, S Moka, and Y. Nazarathy. *The Mathematical Engineering of Deep Learning*. https://deeplearningmath.org/. Accessed on 12.12.2023. 2023.

[8] *Oxford English Dictionary*. Accessed: 2023-12-12. Oxford University Press, 2023. URL: https://languages.oup.com/.

[9] PyTorch. *DEEP LEARNING WITH PYTORCH: A 60 MINUTE BLITZ*. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html. 2023.