Anna Matilda Nilsson

Dr. Steph Paladini

Essex University

April 14, 2025

Individual Presentation: Neural Network Models for Object Recognition

Hello, my name is Matilda Nilsson and in this presentation I'll be walking you through a project I completed in my machine learning module. The topic for this assignment was neural network models for object recognition using the CIFAR-10 dataset. This assignment was designed to assess both the technical and communication aspects of machine learning applications. I'm going to start by explaining the problem we're trying to solve, the dataset I worked with, the process of building a neural network using Python and TensorFlow, and lastly, I will go over my reflections on what I learned along the way.

So, I want to begin by discussing the broader significance of object recognition. Object recognition is a critical function in many of the modern AI systems. The biggest example, in my mind, is autonomous vehicles that need to detect and recognize patterns like pedestrians, road signs, and other cars to navigate the roads safely. These have become more and more popular, especially in the last 10 years, and we will definitely see more of them in the future. Another thing they use is, after recognition, our smartphone facial recognition secure access systems. Think of the facial ID feature that exists on all the more recent iPhone models. Security cameras also analyze footage to detect suspicious activity, and manufacturing robots check product quality using visual inspection systems in a lot of agricultural settings. This assignment stimulates one of these real-world challenges on a much smaller scale. We were set out to train an AI to recognize objects based on images.

The project uses CIFAR-10, which is a widely used dataset for benchmarking image classification models. So, for our dataset overview of the CIFAR-10, this dataset contains 60,000 32x32 pixel color images, which were categorized into 10 classes. As you can see on the right, these classes are airplane, automobile, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Each class had 6,000 images, and this balanced dataset was very ideal for exploring how well a neural network can distinguish between visual categories. It's divided into 50,000 images for training and 10,000 images for testing. For my project, I also created a validation set by setting aside 5,000 images from the training set, which I'll explain why shortly.

As I mentioned, I split the original 50,000 images into 45,000 for training and 5,000 for validation. The validation set helped monitor how well the model generalized unseen data during training. Each image is an RGB image with a 32x32x3, which gives us 3,072 features per image, which is a high dimensional input for such small visuals. Handling this complexity is part of why convolutional neural networks are essential. They're also known as CNNs. So, why did I not just use the test set to evaluate the model? I created a validation set to serve a few purposes. Firstly, it provides feedback during the model development, and by using it, I can tune for hyperparameters like the learning rate or dropout rate and detect overfitting. If we tune our model based on the test set, you can run the risk of over-optimizing it and getting misleading information metrics. So, the validation set allows for iterative improvement, while the test gives us a final unbiased evaluation.

Here are a few examples from the dataset. At 32x32 pixels, the images are fairly low resolution, which makes object recognition harder than it would be with higher resolution images. Us humans can usually recognize them at a glance, and we want to train a machine to do the same. These visualizations help me appreciate the diversity within each category. For example, I look at the trucks or the automobiles, and a lot of times they could be front-facing or side-facing or in different lightings, and that variation made the dataset much more realistic for a project testing perspective. For the CNN model architecture, I designed to work with image data by automatically learning the hierarchies of features. The architecture I used include a convolutional layer with 32 filters, then a max pooling layer to reduce the dimensions, then a second convolutional layer with 64 filters, another pooling layer. On the right, you can see I have arrows pointing to each section. In the second image, you can see the flattening steps to convert to 1D, a dense hidden layer with 64 units, a dropout layer with a rate of 0.5, and finally the dense output layer with a softmax activation to reduce the class probabilities. Some key model components for this project I used was for the activation functions. I chose ReLU for the hidden layers because it is very efficient and it avoids the vanishing gradient problem. For the output, I used softmax because we are dealing with a multi-class classification. The last function is categorical cross-entropy, which measures how far out predicted class distribution is from the true labels. Lastly, I used the atom optimizer, which adjusts the learning rate during training for better performance.

During my training parameters, I used 10 epochs with a batch size of 32, and I found that this setup provided the optimal balance between speed and model performance. Each epoch iterates over the entire training set, which ends up updating the model performance. I was creating this project using Google Colab, which provided free GPU support, which made it significantly faster than using my local environment. Going over some accuracy and loss data, as you can see, the training and validation accuracy both improved steadily across the epochs and throughout my exploration and learning in this project, leveling off near the end of the project. On the last graph shows the two plots, one for training and validating accuracy, and one for training and validating loss, over 50 epochs. You can also see that in both training and validating accuracy, improved steadily slightly. On the last graph shows the two plots, one for training and validating accuracy, and one for training and validating loss, over 50 epochs. You can also see that in both training and validating accuracy, improved steadily throughout the training, and validation accuracy consistently follows the training curve, this suggesting that the model is generalizing well and not overfitting significantly. On the plot loss, you can see a clear downward trend in both the training and validation loss. It has a minor fluctuation in the validation curve, but I expected this and it reflects sort of the natural variation due to the batch updates. And by the end of the training, you can see both the curves plateau, which indicates convergence. Overall, the plots demonstrate a well-tuned training process with good generalization. They support the conclusion that the model has learned meaningful patterns from the data while maintaining stability and consistency between training and validating, which could indicate a mild overfitting, but I expected this and find it acceptable in models of this size and scope.

Another thing I want to point out is you can see in my very initial testing and learning and playing around with the dataset, I have the two above progression shots of my first graph of my epochs, versus as I got more comfortable with the program and my testing process. After training, I evaluated the model on the untouched test set and my final accuracy was around 68%, which is reasonable for a small basic CNN without any data augmentation or advanced tuning techniques. My test accuracy was almost at 84%, which is well above the baseline and shows the model generalizes pretty effectively. The precision sat at 87%, which means that when the model predicts a class, it's usually correct. The recall sat at a solid 81%, which tells us how well the model is capturing all instances of a given class. The F1 score sat at a 95, which is a pretty harmonic mean for the precision and recall, which was pretty impressive. It shows that the model is very balanced and not biased towards just one of the metrics. The test loss sat at 67%, which is a lower value, which indicates that the model's predictions are aligning well with the true values. These metrics gave me confidence that the model is not just memorizing the training data, but it's actually learning and generalizing patterns. And for this sort of final project to introductionary machine learning, I thought these were very solid results.

Looking at the confusion matrix and classification report, I got a much deeper understanding for how the model performs across specific categories. My top performing classes were frogs, ships, and automobiles, all of them scoring above a 0.9. I think it's because these classes have very distinct visual features, which makes them easier for the model to identify. While the more challenging classes for the model to identify were more generic animals, like cats, birds, and dogs, that had a much lower F1 score of about 0.7, 0.8. I think a lot of those images and animals shared textures and backgrounds, which contributed a lot to the miscalculations. Which aligns with what I saw visually when looking at each image in these classes. I think cats and dogs being almost the most frequently confused together makes a lot of sense because they even just share a lot of facial features that, even for the human eye, observing the data set would take another half second to make sure they're identifying correctly. But the overall macro F1 score was 0.84, which I think suggests consistency in the performance across the classes. But the weighted average also sat around 0.84, which means that the model handled the class imbalance fairly well. I think these results confirm that the model is pretty strong. There is obviously room to improve, especially in the categories where the overlap is pretty high. Like I said, the cats and dogs.

For my design decisions and strategies, I kept my model architecture very simple. Firstly, to reflect the introductory understanding of convolutional networks. My goal was more to build a prototype with enough depth to extract features, but still be interpretable. I experimented a lot. implemented, more with the dropout and increased filter counts to manage overfitting. I think if this was a production system, I'd also consider automated hyperparameter tuning and regularization techniques. I think the biggest encounter and challenge was balancing the model complexity and training time. Deeper networks often performed better, but it took a lot longer to train. And as you can see from my visual here, in a final training set of 50 epochs, each set took about 5 minutes, and with 50 training sets, the model training process took a little bit over 4 hours to run, which then could have multiple setbacks if I encountered any errors. I also had to adjust my batch size and learning rate when I saw the training becoming unstable during my initial steps of this project. But some of the lesser challenges I encountered was overfitting, which I mitigated with dropout and monitoring the validation loss. And I think my least challenging aspect was interpreting the noisy predictions, which I also learned was more common in visual datasets anyway. I think a model like this, with enhancements, could be used in a lot of the basic visual detection systems. Facial recognition is used more commonly now in medical diagnostics, agricultural monitoring, airport security, and more. I know I've started to encounter the facial recognition for airport security traveling around North America. And I think by learning how CNNs work on small tasks, we are setting the stage for tackling bigger problems with much higher stakes, such as road safety or airport security.

I really think this project helped me bridge theoretical learning with the practical implementation, and it gave me a really good chance to play around and understand how model components interact and how important it is to validate and evaluate correctly. It also showed me the value of experimentation and iteration. And I think my biggest takeaway has been that machine learning is not about the one perfect model. It's about the continuous refinement. The initial process was much messier than the final product, and now I feel much more confident in building models and interpreting results responsibly and more correctly.

For my references in this project, I obviously referenced the original CIFAR-10 dataset paper by Krzyzewski, Nair, and Hinton, which served as the foundation for the image data. And then I also used the official documentation from TensorFlow and Keras to implement and evaluate the neural network. And then to support my theoretical understanding and design strategy, I referred to gradient-based learning and applied-to-documentation recognition by Lakin and others. And I also referred to deep learning by Goodfellow from MIT Press. I think these sources specifically provided really good key insights into configurational networks, backpropagation, and regularization techniques. But I want to thank you for your time and attention. I hope this presentation has given you a clear understanding of how I interpreted neural networks and how they can be applied to object recognition tasks. Thank you, and goodbye.