University of Essex Online

MSc Artificial Intelligence

Transcript

Individual Presentation

Presented by: Elias Medig

Submission date: 14.04.2025

Contents

Transcript

Title Slide

Welcome, everyone. My name is Elias, and today I will be presenting my individual assignment on Neural Network Models for Object Recognition. This presentation will cover the key aspects of my project from an introduction of the case over to the solution deployment and then finally to the conclusion but let’s start with the agenda.

Agenda

In this presentation, I will cover the following topics:

First, an introduction to the project.

Then, I will discuss the dataset used.

Next, I will explain the neural network architecture.

Following that, I will talk about the activation function and loss function.

Then, I will cover the training process, including epochs and neural network strategy.

Finally, I will present the results and conclude with key takeaways.

1.1 Introduction

Neural Networks are widely used in object recognition. They play a crucial role in applications such as self-driving cars, facial recognition, and security systems. The objective of this project is to develop a neural network that can recognize objects using the CIFAR-10 dataset.

1.2 Introduction

Convolutional Neural Networks, or CNNs, are highly effective for image recognition tasks. CNNs extract features from images, allowing them to classify objects with high accuracy. They have become the standard for computer vision tasks (Wu, 2017). You can see how a CNN looks like on the image on this slide. The process begins with an image and concludes with a prediction that provides a probability for each class. In this scenario you can see that the model is 70% sure the input image shows Tweety while it has a 20% probability for Donald and 10% for Goofy.

2.1 Dataset

But let’s start from scratch. We will first talk about the dataset. For this project I used the CIFAR-10 dataset which consists of 60,000 images, each measuring 32x32 pixels, distributed across 10 classes, with 6,000 images per class. You can find some examples in the image on this slide.

2.2 Dataset

Now before starting to import the data set, we need to define how we want to split it. The CIFAR-10 dataset comes with a training and testing dataset, but we want to split it further by having an additional validation dataset. Validation datasets are important as they will allow us to fine tune model parameters at a later stage (Xu and Goodacre, 2018). We aim at distributing accordingly by having a training dataset of 40’000 images, and a test as well as a validation dataset with 10’000 images each.

2.3 Dataset

We started by doing some exploration data analysis where we took a closer look at the labels and the shapes of the test and train dataset offer by the CIFAR10 dataset. What we can observe are the 10 classes, from airplane to truck as well as the current distribution of 10’000 images in the test and 50’000 in the train dataset. Note that we can also see that the images are all 32 times 32 pixels large and have 3 color channels which represent the RGB color palette.

2.4 Dataset

We continue our exploration by looking at individual images. On the first image you can see how we are looking at the first image of the dataset. We can use the show data button to get an overview of the raw pixel values. Each pixel has three values ranging from 0 to 255 which basically represent the RGB color code of each pixel. Since deep learning models perform better with normalized inputs, we divide all pixel values by 255, converting them into a range between 0 and 1. This can be seen on image 3 where we not only process the train but also the test dataset. Last but not least image 4 shows the new normalized arrays that now only include values between 0 and 1.

2.5 Dataset

Similar to how we did pixel normalization, we also preprocess the class labels. The dataset consists of 10 different categories, including airplanes, automobiles, birds, cats, and more. Instead of using raw numerical class labels, we convert them into one-hot encoded arrays, which are required for categorical classification in neural

networks. This ensures compatibility with the softmax classifier used in the final layer. We will learn more about this later. What you can see here on this slid eon the left side is the array before and on the right side is the array after doing the pre-processing task.

2.6 Dataset

As a final step we will now distribute the dataset according to our initial plan by creating a validation dataset consisting of 10’000 images. The code on the slide shows how this is done and ends with providing us the shape of each dataset where we can see that we finally have a 40’000:10’000:10’000 split and are ready to move on.

3.1 Architecture

Chapter 3 is where things are getting more interesting. We are now looking at the neural network architecture we chose. Remember how I showed the architecture of a convolutional neural network on the introduction slide. This slide is now showing the architecture we are going to implement. The neural network consists of multiple layers. We start with a convolutional layer that uses the ReLU activation function that detects patterns in the image such as edges and texture. Note that I will talk more about the chosen activation function later. The convolutional layer is followed by a pooling layer that reduces the dimensionality of the feature maps. Those two layers are duplicated so wo go through them twice before they are followed by a flattening layer that aims at converting the two dimensional feature maps into a one dimensional vector. This step is required for the next one for the dense layer. The dense layer is the fully connected layer that processes the extracted features. Last

but not least we use a softmax classifier that outputs the results in probabilities, for example telling us that an image is showing a truck with 87% certainty.

3.2 Architecture

Now, let's link these layers to the actual code. On this slide you can see which parts of the code are responsible for which part of the neural network. I will not dive into depth here but will give you one example by looking closer at the first line. You can see that we are building the convolutional layer with 32 filters, a kernel size of 4,4 and we also provide the input shape of the images which are 32,32,3 so again the pixels and the color channels. Last but not least we use ReLU as our activation function. Similar to this you can also read the other lines responsible for the other layers. Which I will now not get into for this presentation.

3.3 Architecture

The model summary visually confirms our architecture. It details layer shapes, parameter counts, and transformations. Note that the parameter counts represent the trainable values. They are calculated based on input dimensions, filter sizes, and neuron connections.

4.1 Activation Function

This slide highlights the use of ReLU and Softmax. ReLU is applied in the convolutional layers to introduce non-linearity, allowing the model to learn complex features. Softmax is used in the final classifier to produce probability distributions for each class (Goodfellow, 2016).

4.2 Activation Function

The ReLU function graph illustrates its behavior. It outputs zero for negative values and passes positive values unchanged. It is known for its computational efficiency and sparsity (Sharma, 2017).

5.1 Loss Function

Looking at the loss function we use categorical crossentropy which is a loss function that measures the difference between the predicted probability distribution and the true class distribution. This helps the model to improve accuracy by minimizing this distance (Chollet, 2021). So, the distance between the predicted probability distribution and true class distribution is minimized to get better results.

6.1 Epochs

To prevent overfitting the model, we employ early stopping, which stops training when validation loss ceases to improve. Initially we will set the patience to two, which means that after the validation loss does not improve for two consecutive epochs the training will stop. Note that on the second code strip you can see how we train the model using this early stop configuration.

6.2 Epochs

This screenshot shows how the results turned out. We can read the number of epochs where we can see that it started with one of 25 indicating that we run a maximum of 25 epochs. Now on the right side you can see the validation loss. Notice how it decreased and became better until epoch 5 where it got worse followed by

epoch 6 where it again worsened and thus the process reached its early stop condition and the model training stopped. Also notice that in about the center of the screenshot you can see model accuracy. For this model that ended after epoch 6 we reached an accuracy of 0.7313. Which is equal to 73.13%.

7.1 Neural Network Strategy

Now after the first model training let's look at how we want to proceed. What is our actual neural network strategy? I researched about the importance of various parameters and came up with this list of six configuration items which will be adjusted between runs. Note that I built dozens of models but will focus on three for this presentation. The parameters changed were the number of layers, the filter per layer, the kernel sizes, the number of neurons in the hidden dense layer, the early stop patience and the dropout. Dropout was an option I came across when researching about CNNs. It helps to prevent the model from overfitting and will be implemented in the dense hidden layer. You can see the code required on the right side. We import Dropout from tensorflow keras layers and then use it by simply saying model dot add and then the dropout rate.

8.1.1 Results

Let's look at the most important part. Let's look at the results. Our first run resulted in a 64% accuracy after 6 epochs. You can see the specific parameters for each run on the slides. For this run we used two layers, 32 filters on both of them, a kernel size of 4,4 a total of 256 neurons and an early stop patience of 2. We did not use the dropout feature for this first run. Note that this run was my baseline and I aimed at improving accuracy from here by adjusting parameters for future runs.

8.1.2 Results

Looking at the training versus validation loss as well as at the training versus validation accuracy we can see that there is a clear gap. The validation became worse in terms of increased loss and decreased accuracy. Both could indicate overfitting as the model got better with the training data and worse with the validation data.

8.2.1 Results

This second run I not only increased the layers by adding an additional layer to the CNN but also increased the Neurons by doubling them while also decreasing the early stop policy doubling the number of consecutive validation loss decrease for early stop. The accuracy got worse with 63% while the epochs increased to 14.

8.2.2 Results

Still overall the charts show that the overfitting issue got better. The validation and training sets are less far apart as indicated by the two arrows on this slide.

8.3.1 Results

The last and best run I achieved was by keeping three layers but changing the filters by doubling them with every layer. You can see this on the second line where I had 32,64 and then 128 filters. I also decreased the kernel sizes from 4,4 to 3,3 and additionally used dropout with a value of 0.5 which helped to reduce the risk of overfitting of the model. This training resulted in a model that had 70% accuracy after 10 epochs making it the best performing model during my project work.

8.3.2 Results

For this model the validation versus training data loss and accuracy also got better making overfitting less of a problem. You can see that by looking at the blue and yellow curves. The margin between the lines got smaller compared to the other runs. There is still potential to further reduce the difference making the model even more accurate.

8.4 Results

But the main question is how the model performed in the end. That's why I want to give some examples. Testing the final model against images shows that it was able to detect the ship and airplane, but it failed for the automobile as it labeled it as a truck. You can see this by looking at the result marked green below every image. What we are looking at is a view of the image before I run a prediction against it using the model we built. I then compare the actual label of the image with the prediction.

So step one was to check the image, step two to run a prediction and step three to compare the prediction with the actual label of the image.

I think it was an interesting observation to see that the model failed on a closely aligned object. It got me thinking about the actual quality of the model. Having a model that wrongly interpreted an automobile as a truck could mean that it simply requires more data to clearly learn how to distinguish those two closely aligned objects. I also think it would be interesting to understand if there is a way to change how accuracy is measured for this example. I would like to see a way where I could segment objects into high-level categories as from my perspective having a model

that wrongly predicts an automobile from a truck is better than wrongly predicting a plane for a frog because I would put automobiles and trucks into the same high-level category whereas plane and frog would be in different ones. This would allow me to make the result more granular instead of simply being right or wrong. It's just like you don't want an activation function to only output 0 or 1. Also, this information could help to turn the model in the right direction. I will continue my research in this area as I want to achieve higher accuracy and a better understanding of the reasons behind the wrongly labeled data.

9.1 Conclusion

My key takeaways are that the CIFAR-10 dataset is a great dataset for learning. It only requires minimal adjustments before training, offers a large number of images in the right quality and is freely available. I also think my results showed that a well-planned model architecture is key to a successful model implementation. It's better to take some additional time planning before jumping into the building part as it may make it easier to understand what finetuning may be required to increase the prediction accuracy. This brings me to my last takeaway which is fine-tuning. Fine-tuning parameters is essential to get a model's accuracy up and it requires a deep understanding of the model's components. I don't think it's realistic to set up a well working machine learning model without having to finetune it based on testing and result measurements.

Overall, this project was a great opportunity to learn more about Convolutional Neural Networks and how they are implemented. It also showed me how easy it is to implement a machine learning algorithm but how hard it is to achieve high quality

results. I am looking forward to learn more about this and will continue my journey in the area of machine learning.

Thank you

This brins me to my last slide which concludes my presentation. Thank you for your time. You can find all references and the code in the transcript of this presentation.

References

- CIFAR-10 dataset. Available at: https://www.cs.toronto.edu/~kriz/cifar.html (Accessed: 7 April 2025)
- Lukmaan IAS. (2024, December 18). Convolutional Neural Networks (CNN): An in-depth exploration [Blog post]. Retrieved from https://blog.lukmaanias.com/2024/12/18/convolutional-neural-networks-cnn-an-in-depth-exploration/
- Wu, J., 2017. Introduction to convolutional neural networks. National Key Lab for Novel Software Technology. Nanjing University. China, 5(23), p.495.
- Patel, A., 2023. The Power of Deep Learning in Image Recognition. Artificial Intelligence in Plain English, 15 February. Available at: https://ai.plainenglish.io/the-power-of-deep-learning-in-image-recognition-65e3485541d0 (Accessed: 7 April 2025).
- Sharma, A.V. (2017) Understanding Activation Functions in Neural Networks. The Theory Of Everything. Available at: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0 (Accessed: 7 April 2025).
- Chollet, F. and Chollet, F., 2021. Deep learning with Python. simon and schuster.
- Goodfellow, I., 2016. Deep learning.
- Xu, Y. and Goodacre, R., 2018. On splitting training and validation set: a comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning. *Journal of analysis and testing*, *2*(3), pp.249-262.

- University of Essex Online (2025) ‘Artificial Neural Network (ANN)’ [Online learncast]. In: Machine Learning. University of Essex. Available at: https://www.my-course.co.uk/mod/scorm/player.php?a=17531¤torg=articulate_rise&scoid=35136&sesskey=qbh3fNljxo&display=popup&mode=normal (Accessed: 7 April 2025).
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), pp.1929-1958.

Code Appendix

Code copied from Jupyter notebook that I used for this assessment. The Jupyter Notebook will be handed in together with the Reflection and Presentation:

```
from numpy.random import seed
#fixing numpy seed for reproducability
seed(123)
import tensorflow
#fixing tensorflow seed for reproducability
tensorflow.random.set_seed(321)
#importing all libraries that we will need
import os
import numpy as np
import itertools
import tensorflow as tf
import keras
from keras.datasets import cifar10 # importing the cifar-10 dataset
from keras.models import Sequential      #this part is required to define model
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten
from sklearn.metrics import confusion_matrix
#for image exploration we need this
from IPython.display import display
from keras.preprocessing.image import array_to_img
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
#This part is about loading the dataset as a tuple the dataset comes with 50'000 training and 10'000 test images
(x_train_all, y_train_all), (x_test, y_test) = cifar10.load_data()
#seting up a constant for the label names
LABEL_NAMES = ['airplane', 'automobile','bird','cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
x_test.shape
x_train_all.shape
x_train_all[0]
#preprocessing the data
x_train_all =x_train_all / 255.0
```

```
x_test =  x_test / 255.0
#look at an image in the data after getting it into shape (between 0-1)
x_train_all[0]
#look at the classes
y_test
#getting clases into one-hot encoding so they become a 10 element vector which is required for other process steps in
categorization
y_cat_train_all = to_categorical(y_train_all,10)
y_cat_test = to_categorical(y_test,10)
#take a look at the classes again after converting
y_cat_test
#defining validation size variable as 10'000
VALIDATION_SIZE = 10000
#creating valiadtion set out of training set
x_val = x_train_all[:VALIDATION_SIZE]
y_val_cat = y_cat_train_all[:VALIDATION_SIZE]
x_val.shape
x_train = x_train_all[VALIDATION_SIZE:]
y_cat_train= y_cat_train_all[VALIDATION_SIZE:]
#printing out the dataset sizes to check
print("Training set shape:", x_train.shape)
print("Validation set shape:", x_val.shape)
print("Test set shape:", x_test.shape)
from tensorflow.keras.layers import Dropout
model = Sequential()
# CONVOLUTIONAL LAYER
model.add(Conv2D(filters=32, kernel_size=(3,3),input_shape=(32, 32, 3), activation='relu',))
# POOLING LAYER
model.add(MaxPool2D(pool_size=(2, 2)))
# CONVOLUTIONAL LAYER
model.add(Conv2D(filters=64, kernel_size=(3,3),input_shape=(32, 32, 3), activation='relu',))
# POOLING LAYER
model.add(MaxPool2D(pool_size=(2, 2)))
# CONVOLUTIONAL LAYER
model.add(Conv2D(filters=128, kernel_size=(3,3),input_shape=(32, 32, 3), activation='relu',))
# POOLING LAYER
model.add(MaxPool2D(pool_size=(2, 2)))
# FLATTEN IMAGES FROM 32 x 32 x 3 =3072 BEFORE FINAL LAYER
```

```
model.add(Flatten())
# DENSE HIDDEN LAYER
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
# CLASSIFIER
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy',
optimizer='adam',
metrics=['accuracy'])
model.summary()
#implementing early stop
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='val_loss',patience=2)
#this part actually trains the model using the x_train and y_cat_train data for a maximum of 25 epochs.
#the model is evaluated on the x_val and y_val_cat data at the end of each epoch.
history = model.fit(x_train,y_cat_train,epochs=25,validation_data=(x_val,y_val_cat),callbacks=[early_stop])
model.history.history.keys()
#let's take a look at the metrics
metrics = pd.DataFrame(model.history.history)
metrics
metrics[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
metrics[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
model.evaluate(x_test,y_cat_test)
from sklearn.metrics import classification_report, confusion_matrix
#predictions = model.predict_classes(x_test)
predictions = np.argmax(model.predict(x_test), axis=-1)
print(classification_report(y_test,predictions))
#taking a look at an image
plt.imshow(x_test[3])
#running the prediction on the image my_image which is an image out of x_test
my_image = x_test[3]
predicted_class = model.predict(my_image.reshape(1,32,32,3)).argmax()  # Get predicted class index
actual_class = y_test[3][0]
#printing the prediction stored in predicted_class as well as the actual class stored in actual_class
```

```
print("Predicted:", LABEL_NAMES[predicted_class])
print("Actual:", LABEL_NAMES[actual_class])
```