Neural Network Models for Object Recognition

by Ben Zapka

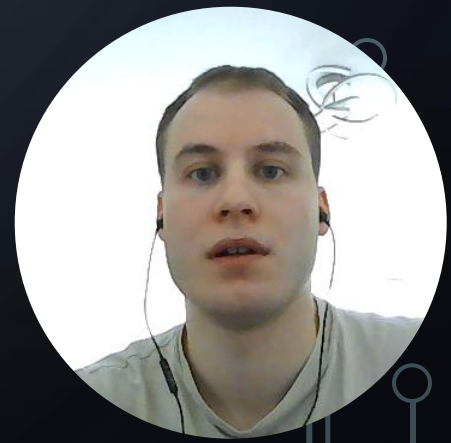
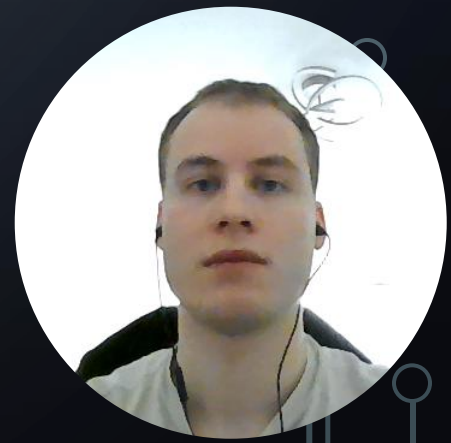


Table of Contents



1. Introduction

- Deep learning models, particularly Convolutional Neural Networks (CNNs), have significantly improved object detection accuracy and efficiency (Xue, 2024).
- Applications include autonomous driving, video surveillance, and medical imaging (Wu et al., 2020).
- CNNs have revolutionized object detection by automatically extracting hierarchical features from images, enabling complex pattern recognition without manual feature engineering (Xue, 2024).

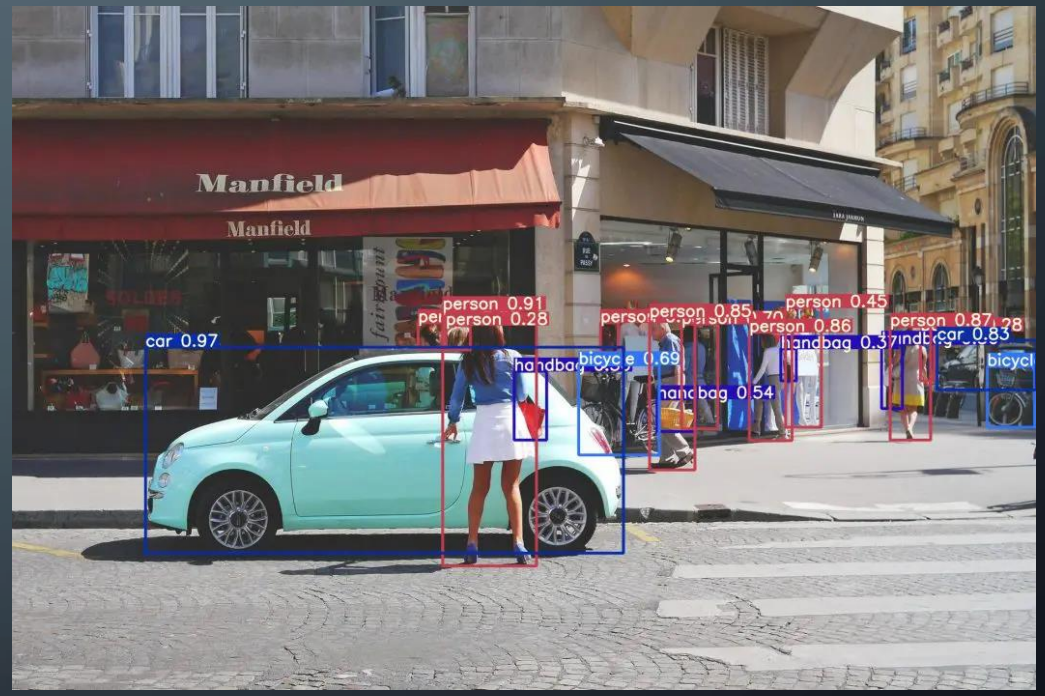


Image 1: Object Recognition in Practice (Boesch, 2024)

2. Dataset Description

- CIFAR-10 includes 50,000 training images and 10,000 test images with balanced distribution across 10 classes (Krizhevsky, 2009).
- The dataset is frequently used to test Neural Networks‘ ability to classify images.
- Challenging: Rather low resolution and small image size (32x32)

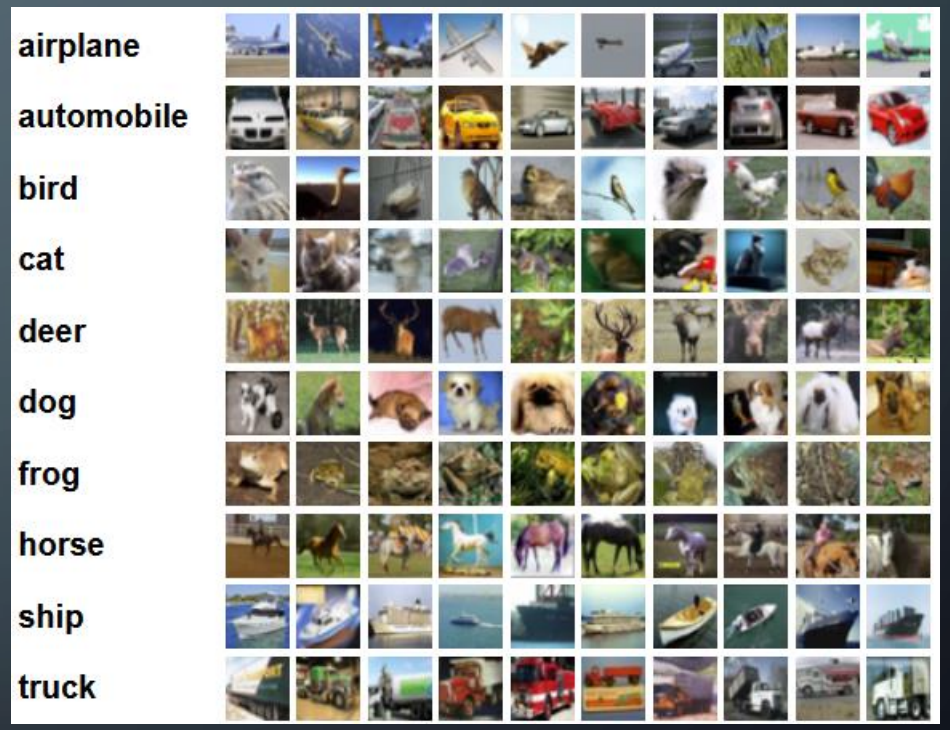


Image 2: The CIFAR-10 dataset (Krizhevsky, 2009)

3. Data Preparation

- I normalise the pixel values of the images from the range [0,255] to the range [0,1].
- This improves the efficiency of activation functions and accelerates convergence (Ioffe & Szegedy, 2015).

```
# Normalize images to scale pixel values between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

- I split the original training dataset into two subsets: training data (80%) and validation data (20%).
- This is supposed to prevent overfitting (Goodfellow et al., 2016) and optimise parameter tuning (Bengio, 2012).

```
# Split training data to create a validation set (80-20 split)
train_images, val_images, train_labels, val_labels = train_test_split(
    train_images, train_labels, test_size=0.2, random_state=42)
```

4. Neural Network Architecture (1/4)

- I initialise a sequential model, where layers are stacked linearly.

```
model = models.Sequential()
```

- For the first convolutional layer, I add a convolutional layer, batch normalization and MaxPooling.
- Convolutional layer extracts spatial features from the image, using the ReLu activation function (LeCun et al., 2015).
- Batch Normalisation normalises the activations to stabilise training and accelerate convergence (Ioffe & Szegedy, 2015).
- MaxPooling reduces computational complexity and captures dominant features (Zafar et al., 2022).

```
model.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32, 3),
                        kernel_regularizer=regularizers.l2(0.0001)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
```

4. Neural Network Architecture (2/4)

- A second and third convolutional layer are used to extract more complex features.
- For that it also increases the depth to 128 filters (Krizhevsky et al., 2012).

```
model.add(layers.Conv2D(128, (3, 3), activation='relu',
                        kernel_regularizer=regularizers.l2(0.0001)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu',
                        kernel_regularizer=regularizers.l2(0.0001)))
model.add(layers.BatchNormalization())
```

- I now introduce a dropout to increase the generalisation of the model.
- Dropout randomly drops units (here 40%) during training, which prevents units from co-adapting too much (Srivastava et al., 2014).

```
model.add(layers.Dropout(0.4))
```

4. Neural Network Architecture (3/4)

- The introduced flatten layer converts the 2D feature map to a 1D vector, which is necessary to input it into a dense layer afterwards (Goodfellow et al., 2016).
- The dense layer is then used to capture complex relationships between features extracted by convolutional layers (Simonyan & Zisserman, 2014).

```
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.3))
```

- Final dense layer with 10 neurons (one for each CIFAR-10 class) and softmax activation to output class probabilities.
- Softmax ensures outputs are interpreted as probabilities for multi-class classification (Bishop, 2006).

```
model.add(layers.Dense(10, activation='softmax'))
```

4. Neural Network Architecture (4/4)

ReLU Activation Function:

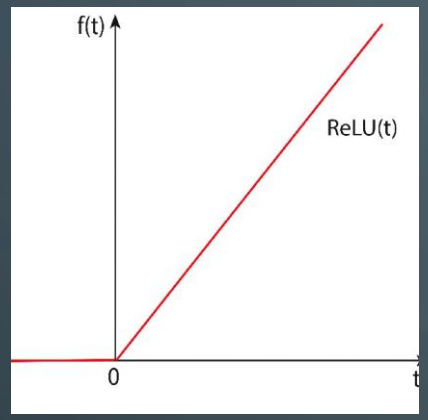


Figure 1: ReLU Activation Function (Olanipekun et al., 2022)

- The ReLU activation function introduces non-linearity while being computationally efficient and easy to optimize (Nair & Hinton, 2010).
- It allows the network to model complex relationships by activating only specific neurons, improving sparsity in the model.

Softmax Activation Function:

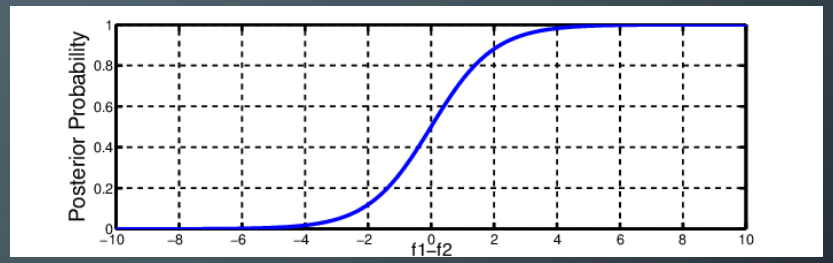


Figure 2: Softmax Activation Function (Chen & Deng, 2017)

- The Softmax activation function is applied in the output layer to convert raw model outputs into probabilities, ensuring they sum to 1 across all classes (Bishop, 2006).

5. Training the model (1/4)

- I now configure the model for training by specifying the optimiser, loss function, and evaluation metric.
- Adam optimizer is used to update the model's weights during training based on the gradients computed from the loss function (Kingma & Ba, 2014).
- I use the sparse categorical cross-entropy loss function which calculates the error between the predicted class and the true class labels, which are provided as integers (Bishop, 2006).
- As evaluation metric, I choose accuracy, which provides immediate feedback on how well the model is learning during training and on unseen test data.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

5. Training the model (2/4):

- Data augmentation artificially increases the size of the training dataset by generating transformed versions of the original images.
- It is used to prevent overfitting and increase the generalisation of the classifier for real-world scenarios (Shorten & Khoshgoftaar, 2019).

```
datagen = ImageDataGenerator(
    rotation_range=5,
    width_shift_range=0.05,
    height_shift_range=0.05,
    horizontal_flip=True
)
```

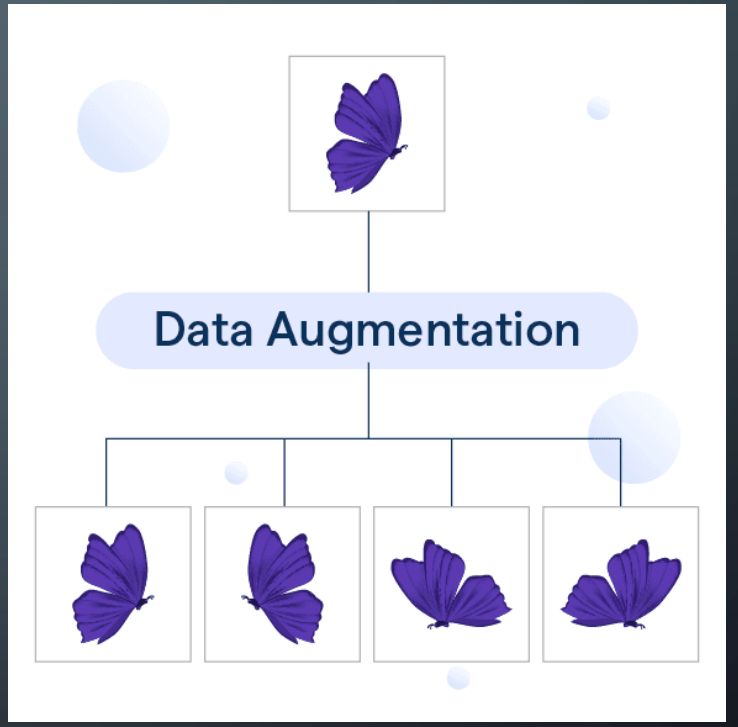


Image 3: Visualisation of Data Augmentation (Bot Penguin, 2024)

5. Training the model (3/4)

- Early stopping stops the training process if the validation loss does not decrease for three consecutive epochs.
- Reducing the learning rate by a factor of 0.5 whenever the validation loss does not improve for 2 consecutive epochs additionally helps the model converge (Smith, 2017).
- Using datagen.flow in training ensures a more robust model by dynamically generating batches of augmented images during training (Shorten & Khoshgoftaar, 2019).
- I train the model for 30 epochs, validating its performance on a separate validation data set.

```
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, min_lr=1e-5)

history = model.fit(datagen.flow(train_images, train_labels, batch_size=128),
                    epochs=30,
                    validation_data=(val_images, val_labels),
                    callbacks=[early_stopping, reduce_lr])
```

5. Training the model (4/4)

```
Epoch 1/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 37s 94ms/step - accuracy: 0.3645 - loss: 1.9650 - val_accuracy: 0.1441 - val_loss: 3.3022 - learning_rate: 0.0010
Epoch 2/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 25s 77ms/step - accuracy: 0.5700 - loss: 1.2733 - val_accuracy: 0.5426 - val_loss: 1.3693 - learning_rate: 0.0010
Epoch 3/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 24s 76ms/step - accuracy: 0.6339 - loss: 1.0982 - val_accuracy: 0.5998 - val_loss: 1.2299 - learning_rate: 0.0010
Epoch 4/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 25s 79ms/step - accuracy: 0.6775 - loss: 0.9932 - val_accuracy: 0.5560 - val_loss: 1.3994 - learning_rate: 0.0010
Epoch 5/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 79ms/step - accuracy: 0.6904 - loss: 0.9393 - val_accuracy: 0.6143 - val_loss: 1.2471 - learning_rate: 0.0010
Epoch 6/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 25s 77ms/step - accuracy: 0.7274 - loss: 0.8527 - val_accuracy: 0.7203 - val_loss: 0.8664 - learning_rate: 5.0000e-04
Epoch 7/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 25s 79ms/step - accuracy: 0.7420 - loss: 0.8111 - val_accuracy: 0.7275 - val_loss: 0.8541 - learning_rate: 5.0000e-04
Epoch 8/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 78ms/step - accuracy: 0.7593 - loss: 0.7743 - val_accuracy: 0.7113 - val_loss: 0.9068 - learning_rate: 5.0000e-04
Epoch 9/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 77ms/step - accuracy: 0.7605 - loss: 0.7631 - val_accuracy: 0.7397 - val_loss: 0.8264 - learning_rate: 5.0000e-04
Epoch 10/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 25s 78ms/step - accuracy: 0.7705 - loss: 0.7382 - val_accuracy: 0.7449 - val_loss: 0.8243 - learning_rate: 5.0000e-04
Epoch 11/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 25s 79ms/step - accuracy: 0.7718 - loss: 0.7219 - val_accuracy: 0.7058 - val_loss: 0.9382 - learning_rate: 5.0000e-04
Epoch 12/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 26s 82ms/step - accuracy: 0.7805 - loss: 0.7083 - val_accuracy: 0.7352 - val_loss: 0.8571 - learning_rate: 5.0000e-04
Epoch 13/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 25s 80ms/step - accuracy: 0.8004 - loss: 0.6502 - val_accuracy: 0.7710 - val_loss: 0.7432 - learning_rate: 2.5000e-04
Epoch 14/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 25s 79ms/step - accuracy: 0.8018 - loss: 0.6370 - val_accuracy: 0.7887 - val_loss: 0.6830 - learning_rate: 2.5000e-04
Epoch 15/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 26s 79ms/step - accuracy: 0.8090 - loss: 0.6256 - val_accuracy: 0.7797 - val_loss: 0.7315 - learning_rate: 2.5000e-04
Epoch 16/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 25s 77ms/step - accuracy: 0.8110 - loss: 0.6215 - val_accuracy: 0.7714 - val_loss: 0.7614 - learning_rate: 2.5000e-04
Epoch 17/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 26s 80ms/step - accuracy: 0.8225 - loss: 0.5820 - val_accuracy: 0.8068 - val_loss: 0.6472 - learning_rate: 1.2500e-04
Epoch 18/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 26s 82ms/step - accuracy: 0.8275 - loss: 0.5752 - val_accuracy: 0.7963 - val_loss: 0.6761 - learning_rate: 1.2500e-04
Epoch 19/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 39s 77ms/step - accuracy: 0.8299 - loss: 0.5667 - val_accuracy: 0.8178 - val_loss: 0.6101 - learning_rate: 1.2500e-04
Epoch 20/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 43s 82ms/step - accuracy: 0.8273 - loss: 0.5677 - val_accuracy: 0.8107 - val_loss: 0.6264 - learning_rate: 1.2500e-04
Epoch 21/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 81ms/step - accuracy: 0.8290 - loss: 0.5599 - val_accuracy: 0.8163 - val_loss: 0.6138 - learning_rate: 1.2500e-04
Epoch 22/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 26s 81ms/step - accuracy: 0.8322 - loss: 0.5494 - val_accuracy: 0.8180 - val_loss: 0.6048 - learning_rate: 6.2500e-05
Epoch 23/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 26s 83ms/step - accuracy: 0.8347 - loss: 0.5438 - val_accuracy: 0.8183 - val_loss: 0.6034 - learning_rate: 6.2500e-05
Epoch 24/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 82ms/step - accuracy: 0.8383 - loss: 0.5378 - val_accuracy: 0.8230 - val_loss: 0.5997 - learning_rate: 6.2500e-05
Epoch 25/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 26s 83ms/step - accuracy: 0.8385 - loss: 0.5353 - val_accuracy: 0.8148 - val_loss: 0.6241 - learning_rate: 6.2500e-05
Epoch 26/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 26s 80ms/step - accuracy: 0.8419 - loss: 0.5265 - val_accuracy: 0.8068 - val_loss: 0.6431 - learning_rate: 6.2500e-05
Epoch 27/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 81ms/step - accuracy: 0.8374 - loss: 0.5311 - val_accuracy: 0.8202 - val_loss: 0.6006 - learning_rate: 3.1250e-05
```

Table 1: Training results throughout the epochs

6. Evaluation and Testing (1/2)

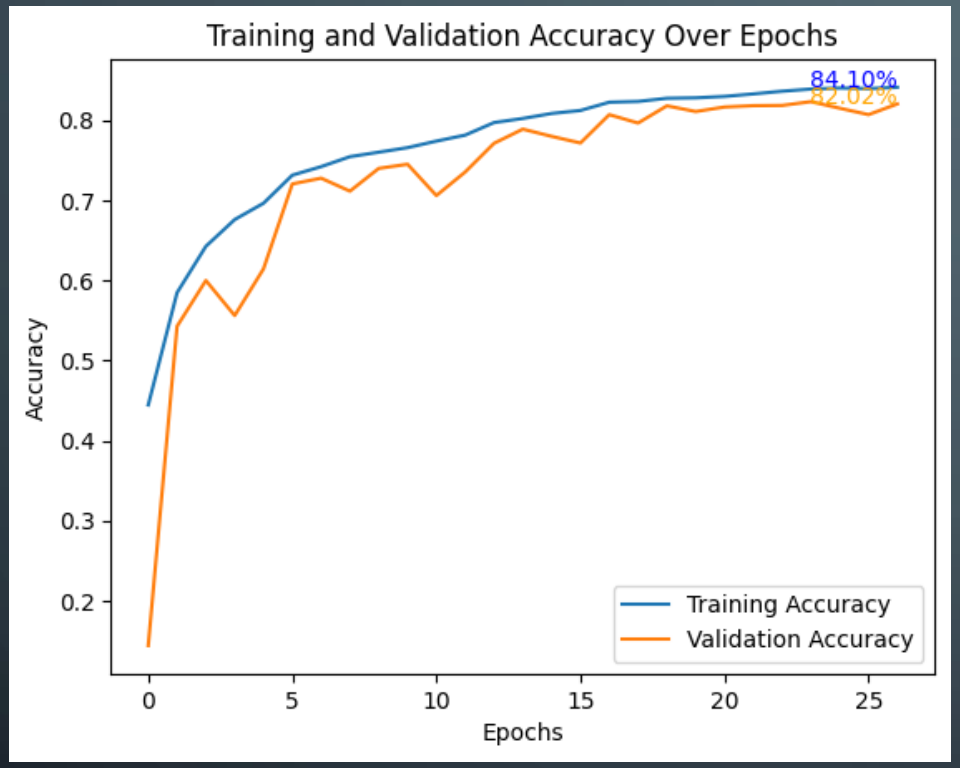


Figure 3: Development of training and validation accuracy over the epochs

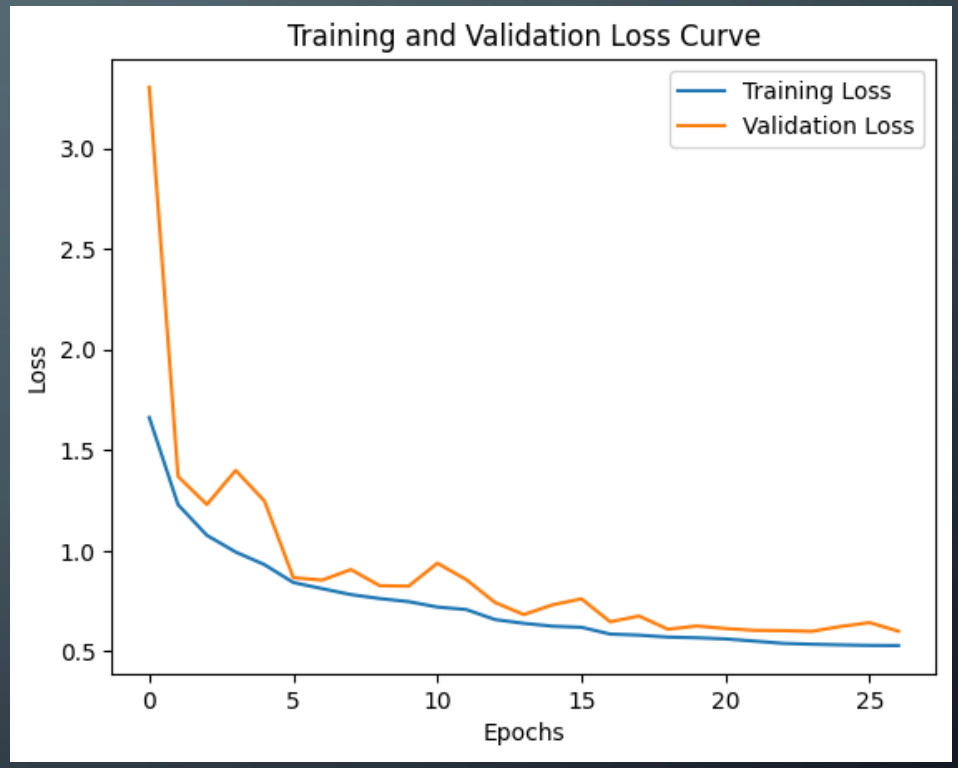


Figure 4: Development of training and validation loss over the epochs

Relatively close to training and validation accuracy, test accuracy is at 81.42%.

6. Evaluation and Testing (2/2)

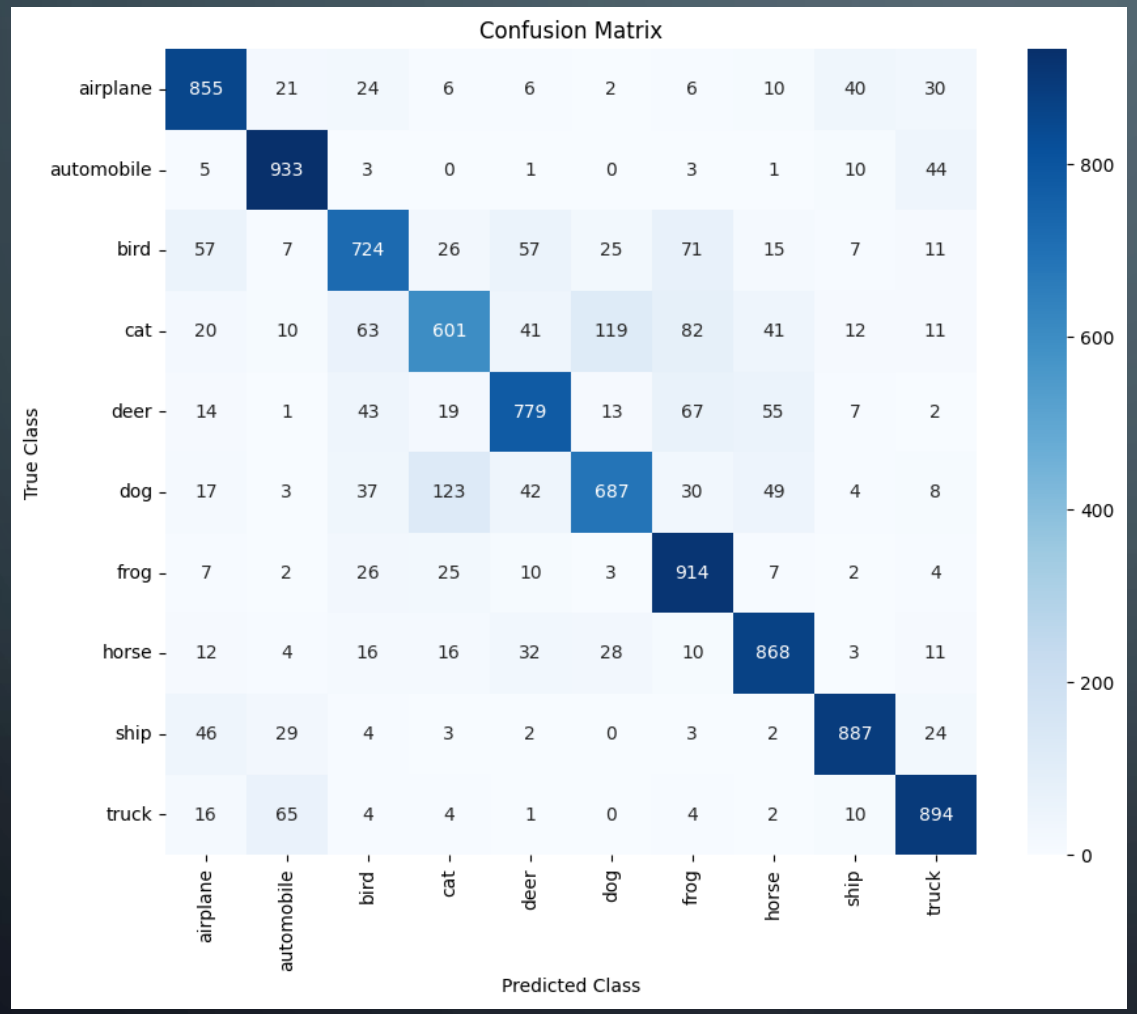


Figure 5: Confusion Matrix

	precision	recall	f1-score	support
airplane	0.82	0.85	0.83	1000
automobile	0.87	0.93	0.90	1000
bird	0.77	0.72	0.74	1000
cat	0.73	0.60	0.66	1000
deer	0.80	0.78	0.79	1000
dog	0.78	0.69	0.73	1000
frog	0.77	0.91	0.83	1000
horse	0.83	0.87	0.85	1000
ship	0.90	0.89	0.90	1000
truck	0.86	0.89	0.88	1000
accuracy			0.81	10000
macro avg	0.81	0.81	0.81	10000
weighted avg	0.81	0.81	0.81	10000

Table 2: Precision and recall for each class

7. Conclusion

- With a rather simple model and only 13 minutes of training time, I can create a reasonably good classifier for the CIFAR-10 dataset, which reaches a test accuracy of 81.42%.

- The model works especially good for automobiles and trucks, but not that good for cats and dogs.

- Main challenges during the creation of this classifier include:
  - Weighing out underfitting and overfitting
  - Creating a model which is complex enough to present a good classification but at the same time light-weight enough to enable a sufficiently fast training process
  - Relatively large similarities between the features of different classes

- Possible improvements for the neural network include:
  - Additional hyperparameter tuning
  - Additional regularisation techniques like DropBlock (Ghiasi et al., 2018)
  - Optionally also use a pre-trained model like EfficientNet and fine-tune it for this data set (Tan & Le 2019)

- Industry applications of this kind of convolutional neural network include:
  - E-Commerce: Products can be classified by extracting different features in uploaded images.
  - Autonomous Vehicles: Detection of objects that are in the way informs the automatic steering of the vehicle.
  - Mobile Devices: Usage of CNNs for facial recognition to unlock devices.

8. List of References (1/2)

- Bengio, Y. (2012): "Practical Recommendations for Gradient-Based Training of Deep Architectures." Neural Networks: Tricks of the Trade: 437 - 478.
- Bishop, C. M. (2006). Pattern Recognition and Machine Learning. New York: Springer.
- Boesch, G. (2024) Object Detection: The Definitive 2025 Guide. viso.ai. Available from: https://viso.ai/deep-learning/object-detection/ [Accessed 17.01.2025]
- Bot Penguin (2024) Data Augmentation. Available from: https://botpenguin.com/glossary/data-augmentation [Accessed 17.01.2025]
- Chen, B., Deng, W. (2017) Noisy Softmax: Improving the Generalization Ability of DCNN via Postponing the Early Softmax Saturation. IEEE Conference on Computer Vision and Pattern Recognition 2017. DOI: https://ieeexplore.ieee.org/document/8099911
- Ghiasi, G., Lin, T.-Y., & Le, Q. V. (2018) DropBlock: A regularization method for convolutional networks. NIPS ’18: Proceedings of the 32nd International Conference on Neural Information Processing Systems: 10750 - 10760. DOI: https://dl.acm.org/doi/10.5555/3327546.3327732
- Goodfellow, I., Bengio, Y., & Courville, A. (2016) Deep Learning. Genetic Programming and Evolvable Machines 19: 305 - 307.
- Ioffe, S., & Szegedy, C. (2015) Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of the 32nd International Conference on Machine Learning. DOI: http://proceedings.mlr.press/v37/ioffe15.pdf
- Kingma, D. P., & Ba, J. (2014) Adam: A Method for Stochastic Optimization. International Conference on Learning Representations (ICLR).
- Krizhevsky, A. (2009) CIFAR Dataset Documentation. Available from: https://www.cs.toronto.edu/~kriz/cifar.html [Accessed 17.01.2025]
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012) ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM 60 (6): 84 - 90. DOI: https://doi.org/10.1145/3065386

8. List of References (2/2)

- LeCun, Y., Bengio, Y., & Hinton, G. (2015): "Deep learning." Nature 521: 436 - 444. DOI: https://doi.org/10.1038/nature14539
- Nair, V., & Hinton, G. E. (2010) Rectified Linear Units Improve Restricted Boltzmann Machines. ICML ’10: Proceedings of the 27th International Conference on Machine Learning: 807 - 814. DOI: https://dl.acm.org/doi/10.5555/3104322.3104425
- Olanipekun, A. T., Mashinini, P. M., Owojaiye, O. A. & Maledi, N. (2022) Applying a Neural Network-Based Machine Learning to Laser-Welded Spark Plasma Sintered Steel: Predicting Vickers Micro-Hardness. Journal of Manufacturing and Materials Processing 6 (5) DOI: https://www.mdpi.com/2504-4494/6/5/91
- Shorten, C., & Khoshgoftaar, T. M. (2019) A survey on image data augmentation for deep learning. Journal of Big Data 6. DOI: https://doi.org/10.1186/s40537-019-0197-0
- Smith, L. N. (2017) Cyclical Learning Rates for Training Neural Networks. IEEE Winter Conference on Applications of Computer Vision (WACV): 464 – 472. DOI: https://ieeexplore.ieee.org/abstract/document/7926641
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting. The Journal of Machine Learning Research 15 (1): 1929 - 1958. DOI: https://dl.acm.org/doi/abs/10.5555/2627435.2670313
- Tan, M., & Le, Q. (2019). "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." Proceedings of the 36th International Conference on Machine Learning. DOI: https://proceedings.mlr.press/v97/tan19a.html
- Wu, X., Sahoo, D. & Hoi, S. C. H. (2020) Recent advances in deep learning for object recognition. Neurocomputing 396: 39 - 64. DOI: https://www.sciencedirect.com/science/article/abs/pii/S0925231220301430?via%3Dihub
- Xue, Q. (2024) Advancements in object detection: From machine learning to deep learning paradigms. Applied and Computational Engineering 75 (1): 154 - 159. DOI: https://www.ewadirect.com/proceedings/ace/article/view/13717
- Zafar, A. et al. (2022) A Comparison of Pooling Methods for Convolutional Neural Networks. Applied Sciences 12 (17): DOI: https://doi.org/10.3390/app12178643