OULU UNIVERSITY OF
APPLIED SCIENCES

Hung Dao

IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

Hung Dao
Bachelor's Thesis
Spring 2020
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Internet Services

Author: Hung Dao
Title of the bachelor's thesis: Image Classification Using Convolutional Neural Networks
Supervisor: Jukka Jauhiainen
Term and year of completion: Spring 2020 Number of pages: 31

The objective of this thesis was to study the application of deep learning in image classification using convolutional neural networks.

The Python programming language with the TensorFlow framework and Google Colaboratory hardware were used for the thesis. Models were chosen from available ones online and adjusted by the author.

After the research, an accurate and performant model was developed and there is still room for further optimization.

Keywords: image classification, convolutional neural network, deep learning

CONTENTS

1 INTRODUCTION

Artificial intelligence (AI) has been a growing trend lately. One of the tasks which can be achieved by AI is computer vision, which is the ability for computers to process and analyse images, aiming to mimic human vision. One of the main tasks of computer vision is image classification, which is the process of labelling images into “classes”. For example, if there are images of multiple objects, and those images need to be categorized into “classes”, for instance “car”, “plane”, “ship”, or “house”, that is image classification.

One common way to execute image classification is through convolutional neural networks, a technique implementing deep learning, which is a subset of machine learning, which is in turn a subset of AI.

The dataset used in this thesis is CINIC-10, from the University of Edinburgh. CINIC-10 is approximately the combination of the two famous datasets for image classification: CIFAR-10 and ImageNet, without their shortcomings. In short, CINIC-10 is not too large, not too small, therefore not too difficult and not too easy, since it is intended to use as a benchmarking dataset. (1.)

The images in CINIC-10 are already labeled into ten categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The dataset is also split into three equal subsets: train, validation, and test, each of which contains 90,000 images. (1.) The aim of this project is to develop a convolutional neural network to automatically classify them.

2 ARTIFICIAL NEURAL NETWORKS

2.1 Human neural network

Because artificial neural networks are inspired by the human neural network, it is good to know how it works (Figure 1).

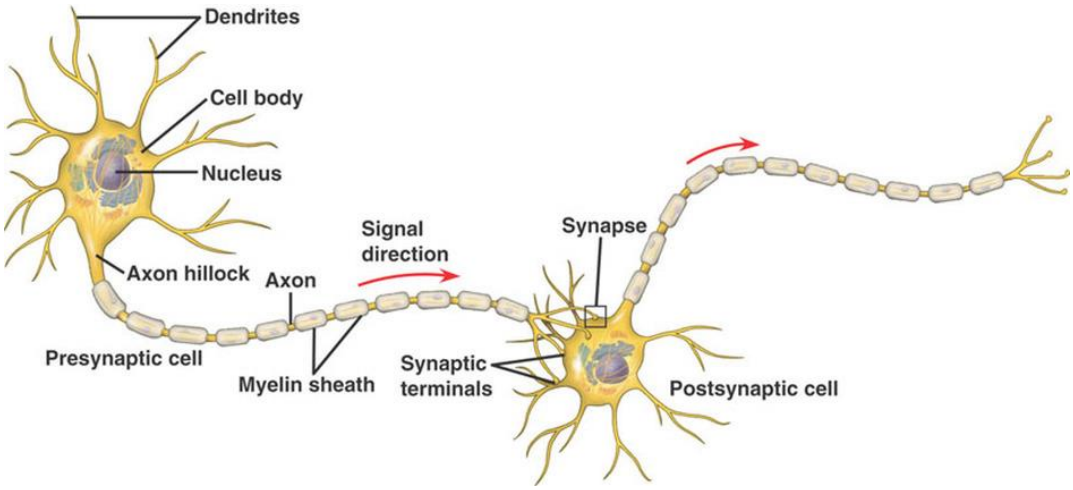


FIGURE 1. A diagram of the neuron highlighting the chain structure between the axon and dendrite. (2.)

When a neuron fires, usually when triggered by a stimulus (3), signals are sent down the axon of that neuron to another neuron's dendrites through a synapse. After that, the new neuron might fire and cause another neuron to fire, repeating the process in the whole system. (2; 4.)

2.2 Artificial neural networks

An artificial neural network (ANN) is a set of layers of neurons (in this context they are called units or nodes). In the case of a fully connected ANN, each unit in a layer is connected to each unit in the next layer (Figure 2).

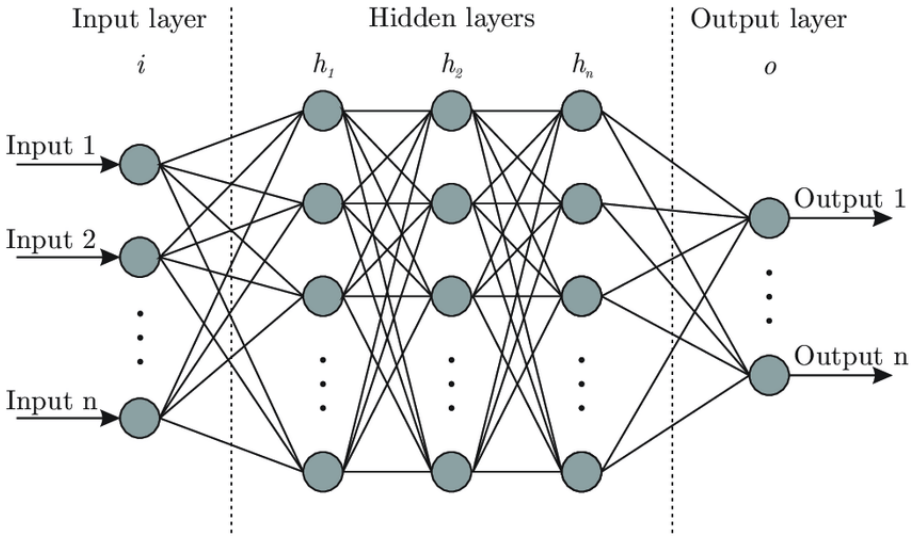


FIGURE 2. The artificial neural network architecture (ANN i-h 1-h 2-h n-o). (5.)

There is an input layer, where the network takes all the information needed, in this case the images to classify. Between the input layer and the output layer are hidden layers. Each hidden layer is used to detect a different set of features in an image, from less to more detailed. For example, the first hidden layer detects edges and lines, the second layer detects shapes, the third layer detects certain image elements, for example a face or a wheel. (6.)

The output layer is where the network makes predictions. The predicted image categories are compared to the labels provided by humans. If they are incorrect, the network uses a technique called backpropagation (to be discussed in chapter 2.5) to correct its learning, so it can make guesses more correctly in the next iteration. (7.)

After enough learning, a network can make classifications automatically without human help. (6.)

2.3 Artificial neuron

An artificial neuron is a connection point (unit or node) in an artificial neural network and has the capacity to process input signals and produce an output signal. (8.)

2.4 Weights, biases, and activation functions

2.4.1 Weights and structure of a neuron

The connections between the units in a neural network are weighted, meaning that the weight indicates how much influence the input from a previous unit has on the output of the next unit. (9.) To mathematically compute an artificial neuron, all the products of all the inputs (x_1 to x_n) and their corresponding weights (w_1 to w_n) are added, then a bias (b) is added to that sum, then the resulting value is fed into an activation function (f) to form the output (Figure 3).

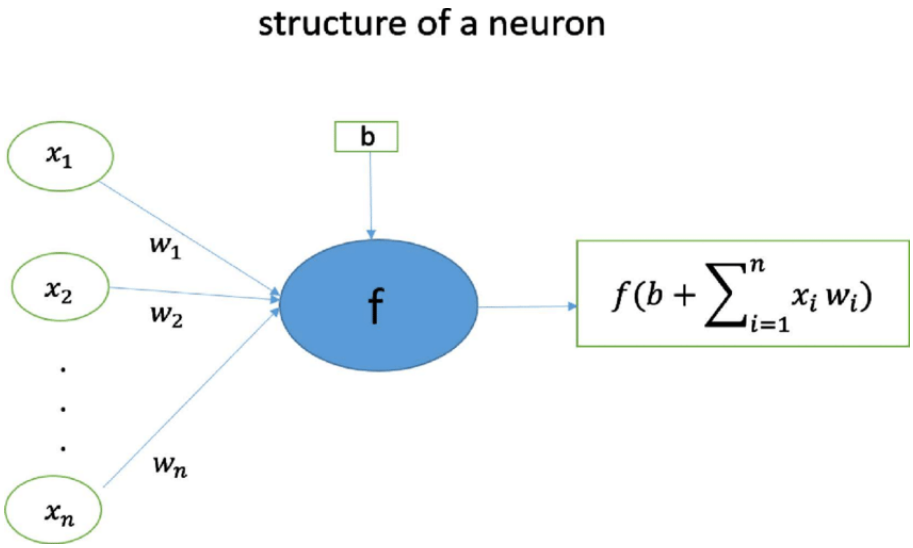


FIGURE 3. A diagram to show the work of a neuron: input x, weights w, bias b, activation function f. (10.)

2.4.2 Biases

A bias (b) is an extra input to a neuron (Figure 3) and it is technically the number 1 multiplied by a weight. (9.) The bias makes it possible to move the activation function curve left or right on the coordinate graph, enabling the neuron to create the required output value (Figure 4). (11.)

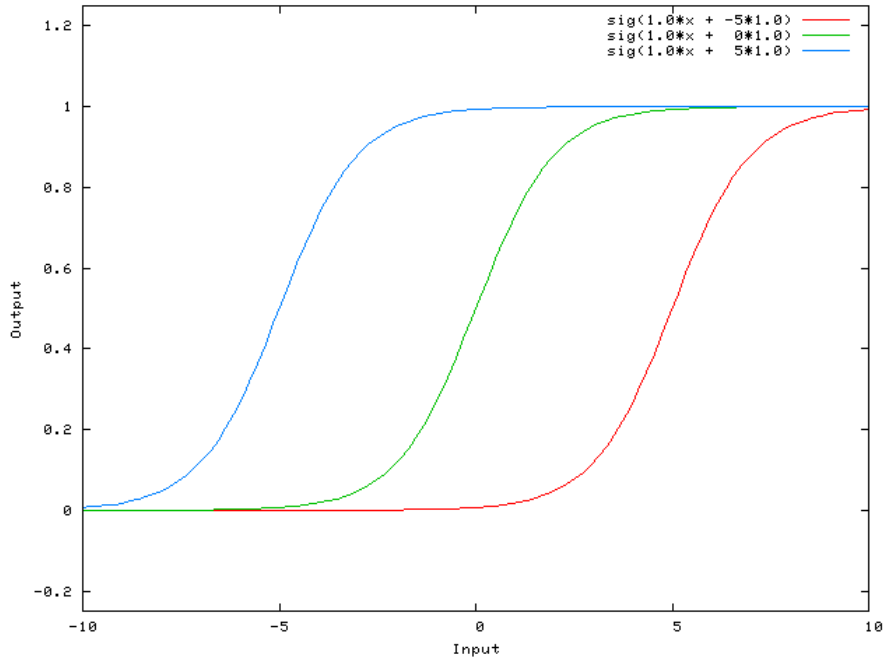


FIGURE 4: A bias value allows the activation function to shift to the left or right. (11.)

To explain the Figure 4, a bias value of -5 enables the Sigmoid activation function to output 0 when the input (x) is 2.

2.4.3 Activation functions and the ReLu

By definition, an activation function decides if a neuron should be activated ("fired") or not. It introduces non-linearity to the output of a neuron. A neural network without activation functions is just a linear regression model. (12.)

The most common activation function for CNNs (13) and also the one used in this thesis is the ReLu: $A(x) = \max(0, x)$. (14.) It outputs x when x is positive and outputs 0 otherwise (Figure 5).

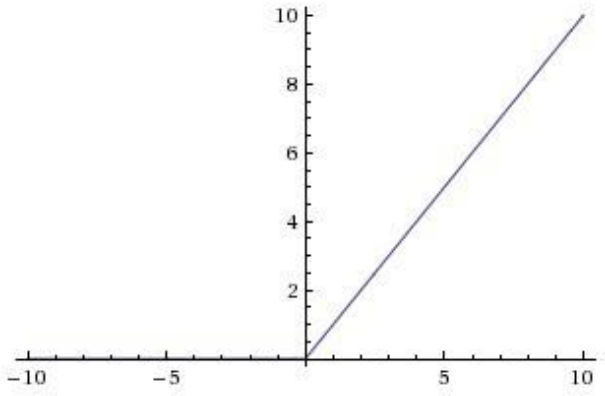


FIGURE 5. The ReLu function. (14.)

ReLu is less computationally expensive than some other common activation functions like tanh and Sigmoid because the mathematical operation is simpler and the activation is sparser. Since the function outputs 0 when $x \leq 0$, there is a considerable chance that a given unit does not activate at all. (13; 14.). Sparsity also means more concise models with more predictive power and less noise or overfitting. In a sparse network, neurons are more likely to process meaningful information. For example, a neuron which can identify human faces should not be activated if the image is actually about a building. (13.)

One more advantage, which the ReLu possesses over the others, is that it converges faster. Linearity (when $x \geq 0$) means that the slope of the line does not plateau when x increases. Therefore ReLu does not have the vanishing gradient problem suffered by some other activation functions, such as Sigmoid or tanh. (13.) Convergence and gradient descent will be discussed in section 2.5.2.

Another common activation function used in CNNs is the Softmax function. It is often used in the output layer, where a multiclass classification happens. The mathematical computation of this function, however, is out of the scope of this thesis. The result (or output) of this function will be discussed in section 3.2.4.

2.5 Backpropagation

Backpropagation is an algorithm which helps neural networks to learn their parameters (15), mostly from errors in predictions. This chapter is going to explain backpropagation using a gradient descent.

2.5.1 Loss function

A loss function is an error metric, a way to calculate the inaccuracy of predictions. The aim of deep learning models is to minimize this loss function value, and the process of minimizing the loss function value is called optimization. (7.)

2.5.2 Gradient descent

A gradient descent is an optimization algorithm which modifies the internal weights of the neural network to minimize the loss function value.

After each iteration, the gradient descent algorithm attempts to decrease the loss function value by tweaking weights, until the point where further tweaks produce little or no change to the loss function value, also called convergence (Figure 6). (16.)

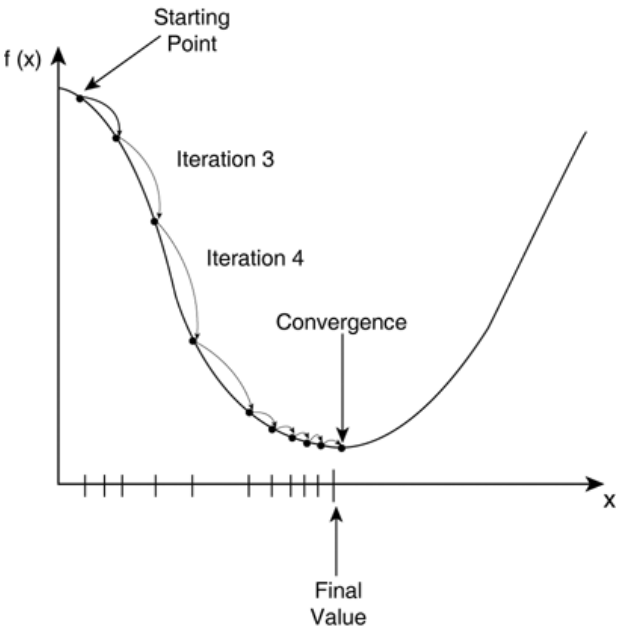


FIGURE 6. The gradient descent (16.)

2.5.3 Learning rate and introduction to the Adam optimizer

A learning rate is the step size of each iteration in the gradient descent or other optimazation algorithms. If the learning rate is too small, convergence will take a long time to happen, but if the learning rate is too large, there might be no convergence at all. (17, p. 247.)

There are many optimization algorithms (optimizer), one of which is the Adam. It computes individual learning rates for different parameters. (18, p. 1.) Adam is the optimization algorithm used in this thesis.

3 CONVOLUTIONAL NEURAL NETWORKS

3.1 How computers see images

An image is comprised of pixels. In the RGB model, each pixel has three color elements, red, green and blue. Most commonly each element can range from 0 (no color) to 255 (full saturation) in value. (19.)

In digital image processing, a colored image is a three-layered matrix of pixels, where each layer is a two-dimensional matrix representing red, green or blue pixel values (Figure 7).

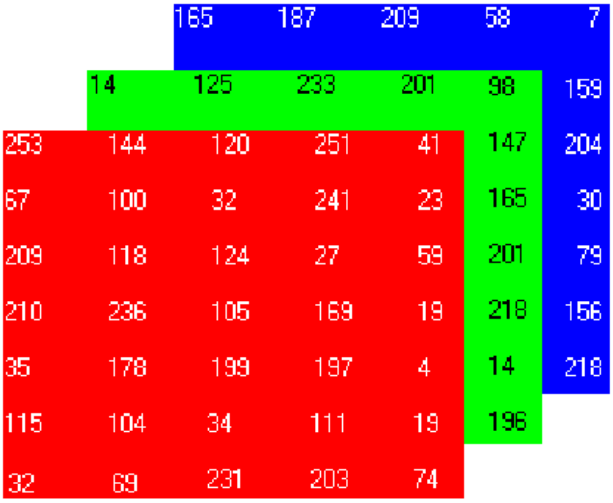


FIGURE 7. A three-dimensional RGB matrix. Each layer of the matrix is a two-dimensional matrix of red, green or blue pixel values. (20.)

3.2 Convolutional Neural Networks

Convolutional neural networks (CNN) are a class of artificial neural networks and one of the most common deep learning architectures for image recognition tasks. (21.)

3.2.1 Architecture

All CNN models share the same architecture (Figure 8). (21.)

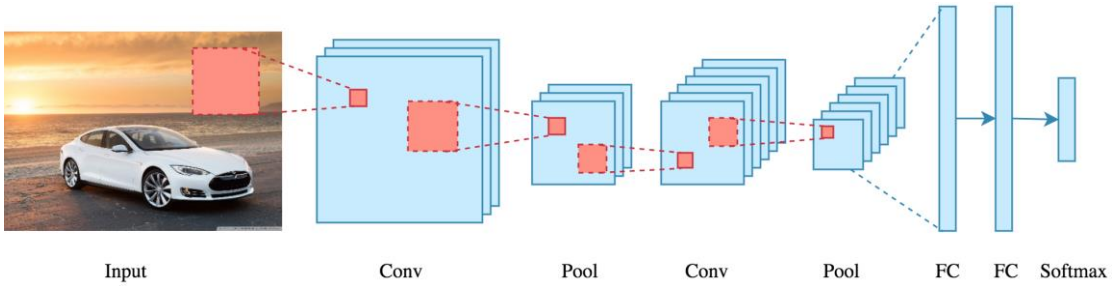


FIGURE 8. The CNN architecture. (21.)

The model takes an input image, performs a series of convolutional and pooling layers, followed by fully connected layers to produce the output. The following chapters will describe each of the architecture components.

3.2.2 Convolutional layers

The term “convolution” refers to the mathematical combination of two functions to form a third function. When that happens, two sets of information are merged.

In the context of CNNs, a convolutional layer (called filter or kernel) is applied to the input data to then produce a feature map (Figure 9). (22.)

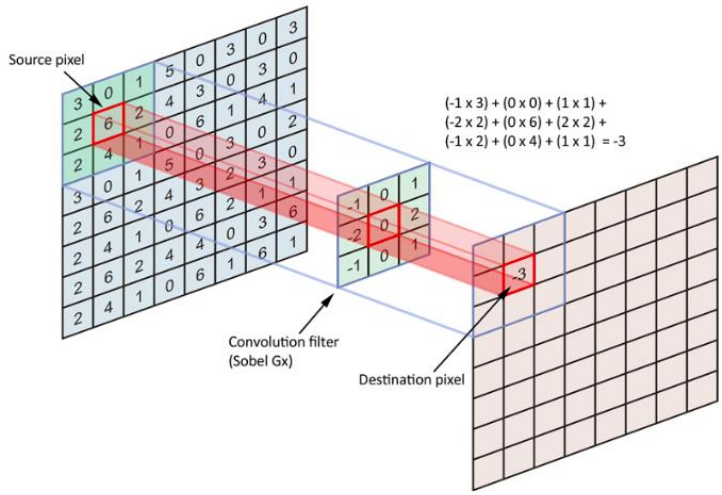


FIGURE 9. The filter slides over the input and performs its output on the new layer. (22.)

In Figure 9, a dot product multiplication is made between a 3x3 sized filter matrix and a 3x3 sized area of the input image’s matrix. The elements of the resulting matrix are summed and the sum is the output value (“destination

pixel”) on the feature map. The filter then slides over the input matrix, repeats the dot product multiplication with every remaining combination of 3x3 sized areas and completes the feature map.

Multiple filters are used for one input and the resulting feature maps are joined together for the final output of one convolutional layer. (22.)

There are two other important concepts in convolutional layers: strides and padding. Strides are the number of pixel a kernel or a filter slides over the input matrix. Padding is what is used when the filter does not fit the input matrix. (23.) There are two types of padding: valid padding, when the border pixels of the input matrix are discarded; and zero or “same” padding, when zeros are added to the borders so that the filter fits the input matrix. (23; 24.)

3.2.3 Pooling layers

Pooling layers are responsible for reducing the dimentionality of feature maps, specifically the height and width, preserving the depth. (21.) Doing so is beneficial because it decreases the required computational power to process the data, while extracting the dominant features in feature maps. (25.)

There are two types of pooling layers: max pooling and average pooling (Figure 10).

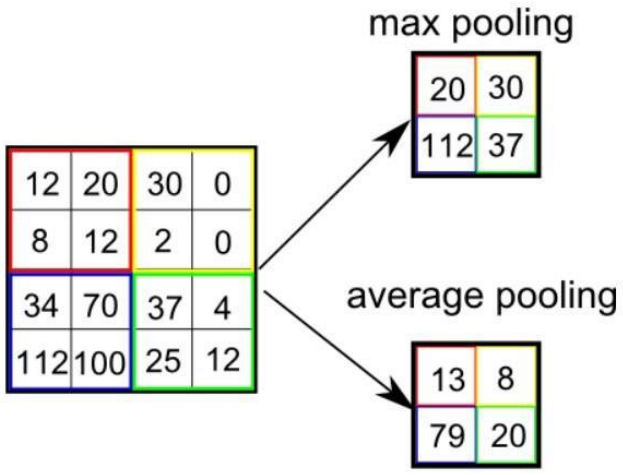


FIGURE 10. Types of pooling. (25.)

Max pooling outputs the maximum value of the elements in the portion of the image covered by the filter, while average pooling returns the average value. Max pooling is better at extracting dominant features and therefore considered more perfomant. (25.)

3.2.4 Fully connected layers

Fully connected layers are where classification actually happens. The input matrix is flattened into a column vector and is fed into a set of fully connected layers which are the same as the fully connected ANN architecture that was previously discussed in chapter 2.2. (21.)

Each fully connected layer (called Dense layer) is passed through an activation function (e.g. tanh or ReLu), but the output Dense layer is passed through Softmax. In the Softmax multiclass classification, the loss function used is Cross Entropy (*categorical_crossentropy* in Keras). (26.)

The output of the Softmax function is an N-dimensional vector, where N is the number of classes the CNN has to choose from. Each number in this N-dimensional vector *represents* the probability that the image belongs to each certain class. For example, if the output vector is [0 .1 .1 .75 0 0 0 0 0 .05], then there is a 10% probability that this image belongs to the class 2, 10% probability that it belongs to the class 3, 75% probability that this image belongs to the class 4, and 5% probability that it belongs to the class 10. (27.)

4 MODEL TRAINING AND RESULTS

4.1 Hardware and software used

The thesis uses free GPUs from Google Colab (Colaboratory). The deep learning framework used is TensorFlow with Keras API.

4.2 Some definitions

4.2.1 Train, validation and test

The train dataset is used to train the model with. In the case of neural networks, the model learns its weights and biases.

The validation dataset is what the model uses for evaluation after every set of predictions. It helps the model tune its hyperparameters.

The test dataset is used to evaluate the model after it has been completely trained. (28.)

4.2.2 Overfitting and underfitting

Overfitting occurs when the model captures the noise of the data. Intuitively, it fits the data too well, or in other words it is too dependent on the data used for training.

Conversely, underfitting occurs when the model cannot capture the underlying trend of the data, or intuitively it does not fit the data well enough.

Overfitting and underfitting both result in poor predictions in new datasets. (29.)

4.2.3 Batch size

Most of the time the whole dataset cannot be fed into the neural network at once, so it has to be divided into parts, or batches. The batch size indicates the number of training samples in a single batch. (30.)

4.2.4 Epoch

One epoch is when the entire dataset (i.e. every training sample) is fed forward and backward through the neural network only once. (30.)

4.2.5 Dropout

Dropout is a technique used to reduce overfitting. The term dropout refers to randomly dropping out units and their connections during training. (31, abstract.)

4.2.6 Batch normalization

Batch normalization is also a method for reducing overfitting. It normalizes the input layer by adjusting and scaling the activations. (32.) The mathematics behind batch normalization is out of the scope of this thesis.

4.3 Models and results

4.3.1 First model

This model is based on the Convolutional Neural Network (CNN) tutorial of TensorFlow (33), with adjustments. There are three convolutional layers, after each is a max pooling layer, and two dropout layers with the dropout rate of 0.3 added to prevent overfitting. After that, there are two dense layers, with 256 and 10 units respectively (10 is the number of classes for classification). Between the two dense layers there is a dropout layer with the dropout rate of 0.2.

The batch size is 32 and the number of epochs is 32. The optimizer is Adam with the learning rate of 0.0001.

Here are the code and the summary of the model:

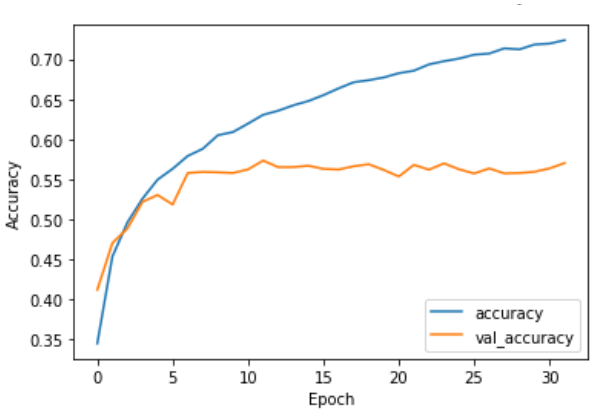
In [0]:
```
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))

model.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))
```

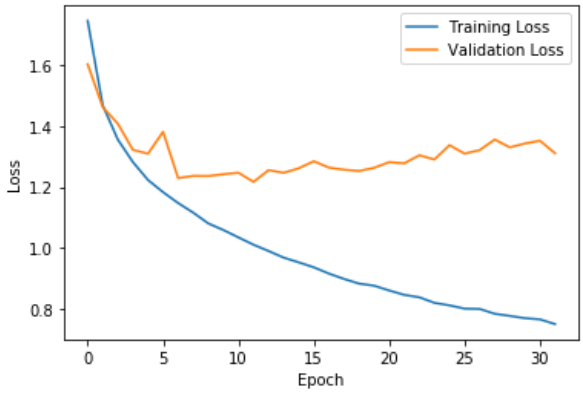
In [0]:
```
model.add(layers.Flatten())

model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.2))

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

In [229]:
```
model.summary()
```

```
Model: "sequential_47"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_155 (Conv2D)          (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d_111 (MaxPoolin (None, 16, 16, 32)        0
_________________________________________________________________
dropout_140 (Dropout)        (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_156 (Conv2D)          (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_112 (MaxPoolin (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_157 (Conv2D)          (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_113 (MaxPoolin (None, 4, 4, 128)         0
_________________________________________________________________
dropout_141 (Dropout)        (None, 4, 4, 128)         0
_________________________________________________________________
flatten_46 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_111 (Dense)            (None, 256)               524544
_________________________________________________________________
dropout_142 (Dropout)        (None, 256)               0
_________________________________________________________________
dense_112 (Dense)            (None, 10)                2570
=================================================================
Total params: 620,362
Trainable params: 620,362
Non-trainable params: 0
```

Here are the results, regarding accuracies and losses:





Test loss: 1.25/ Test accuracy: 0.58

The training accuracy keeps rising but the validation accuracy plateaus very early. Hence, the model is very overfitting, despite many Dropout layers.

4.3.2 Second model

This model is based on the CIFAR-10 dataset training of the official Keras' Github account (34), with adjustments. There are four convolutional layers, after every two of which there is a max pooling layer and a dropout layer with the dropout rate of 0.25. After that, there is a dense layer of 256 units and a dense layer of 10 units. Between the two dense layers there is a dropout layer with the dropout rate of 0.5.

The batch size is still 32 and the number of epochs is 25. The optimizer is Adam with the learning rate of 0.0001.

Here are the model code and summary:

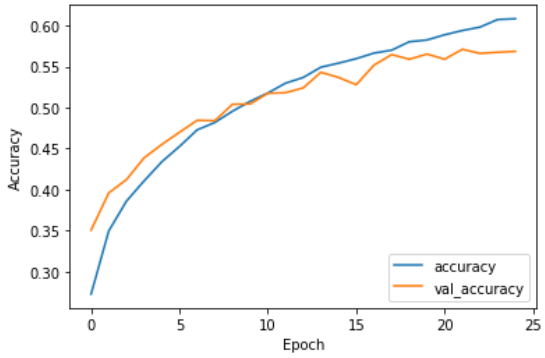
In [0]:
```
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))

model.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))
```

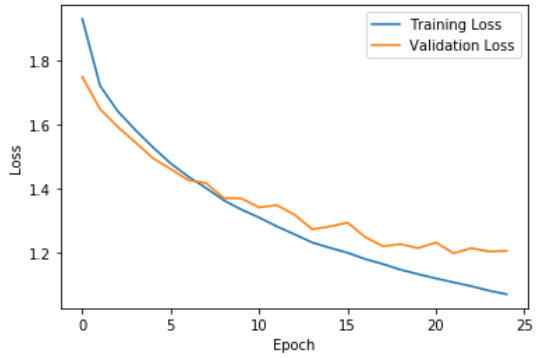
In [0]:
```
model.add(layers.Flatten())

model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.5))

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

In [0]:
```
model.summary()
```

```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_65 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_66 (Conv2D)           (None, 30, 30, 32)        9248
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 15, 15, 32)        0
_________________________________________________________________
dropout_45 (Dropout)         (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_67 (Conv2D)           (None, 15, 15, 64)        18496
_________________________________________________________________
conv2d_68 (Conv2D)           (None, 13, 13, 64)        36928
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
dropout_46 (Dropout)         (None, 6, 6, 64)          0
_________________________________________________________________
flatten_13 (Flatten)         (None, 2304)              0
_________________________________________________________________
dense_25 (Dense)             (None, 256)               590080
_________________________________________________________________
dropout_47 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_26 (Dense)             (None, 10)                2570
=================================================================
Total params: 658,218
Trainable params: 658,218
Non-trainable params: 0
_________________________________________________________________
```

Here are the results:





Test loss: 1.16 / Test accuracy: 0.58

The model is not as overfitting, but the accuracy is not high enough (just over 60%).

4.3.3 Third model

The third model has still the same structure as the second model, but with batch normalization added after each convolutional layer.

The batch size is increased to 128 to save learning time and the number of epochs is 27. The optimizer is also Adam but this time the learning rate is increased to 0.001 also to save learning time.

Here are the code and the model summary:

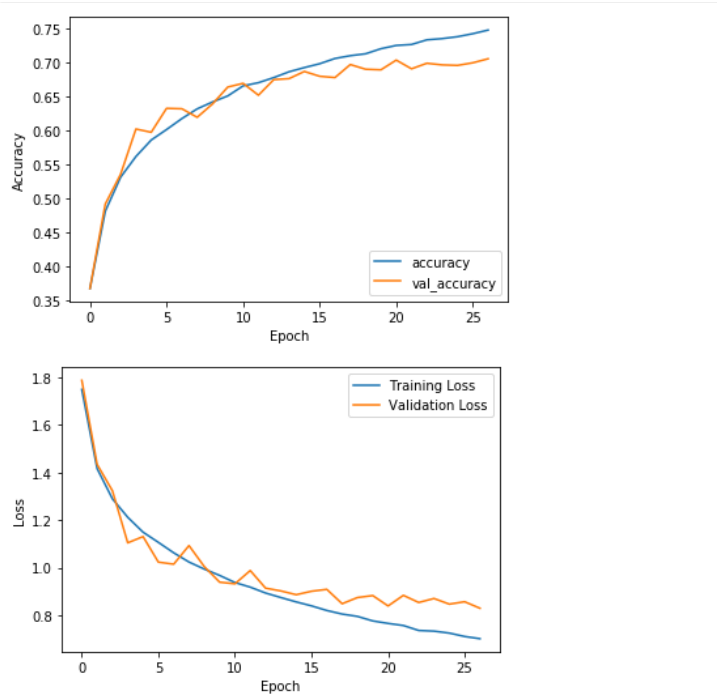
In [0]:
```
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(32, 32, 3)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(rate=0.35))

model.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(rate=0.35))
```

In [0]:
```
model.add(layers.Flatten())

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(rate=0.5))

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

In [83]:
```
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_41 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
batch_normalization_41 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_42 (Conv2D)           (None, 30, 30, 32)        9248
_________________________________________________________________
batch_normalization_42 (Batc (None, 30, 30, 32)        128
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 15, 15, 32)        0
_________________________________________________________________
dropout_29 (Dropout)         (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_43 (Conv2D)           (None, 15, 15, 64)        18496
_________________________________________________________________
batch_normalization_43 (Batc (None, 15, 15, 64)        256
_________________________________________________________________
conv2d_44 (Conv2D)           (None, 13, 13, 64)        36928
_________________________________________________________________
batch_normalization_44 (Batc (None, 13, 13, 64)        256
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
dropout_30 (Dropout)         (None, 6, 6, 64)          0
_________________________________________________________________
flatten_9 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_17 (Dense)             (None, 512)               1180160
_________________________________________________________________
dropout_31 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_18 (Dense)             (None, 10)                5130
=================================================================
Total params: 1,251,626
Trainable params: 1,251,242
Non-trainable params: 384
```

Here are the results:



Test accuracy: 0.71 / Test loss: 0.83

Now the model is performing very well. It is still not overfitting; training, validation and test accuracies are relatively high: 75%, 71% and 71% respectively. Since time and hardware resources were limited, no further research was done and this is the final model.

5 CONCLUSION

The models with four convolutional layers (the second and third model) perform significantly better than the model with three convolutional layers (the first model), with notably less overfitting. This, however, is expected because the second and third model are based on the CIFAR-10 example of the official Keras' Github account, which is still underfitting after 50 epochs (34); while the first model is based on the CNN tutorial of TensorFlow, which already shows signs of overfitting after the 8th epoch. (33.)

The third model performs significantly better than the second model in terms of accuracies. The reasons for this are: a more complicated dense layer (512 units compared to 256 units), larger dropout rates after the convolutional layers (0.35 compared to 0.25), but perhaps most importantly the introduction of batch normalization. The combination of dropouts and batch normalization truly decreases overfitting and enhances the performance of the model.

The increase in the batch size (32 to 128) and learning rate (0.0001 to 0.001) from the second model to the third model suprisingly does not degrade the quality of the model.

If I had had more time and better hardware resources, I would have done more research on how decreasing the batch size, increasing the number of epochs and keeping the learning rate at 0.0001 would affect the accuracies and how overfitting the model is.

REFERENCES

1. Darlow, L. et al. 2018. CINIC-10 Is Not ImageNet or CIFAR-10, University of Edinburgh. Cited 26.11.2019. Available: https://doi.org/10.7488/ds/2448

2. Radice, M. 2019. Basic Structure of the Human Nervous System. Date of retrieval 5.12.2019. Available: https://mikerbio.weebly.com/structure--function.html

3. Khan Academy 2019. Neuron action potentials: The creation of a brain signal. Date of retrieval 7.12.2019. Available: https://www.khanacademy.org/test-prep/mcat/organ-systems/neuron-membrane-potentials/a/neuron-action-potentials-the-creation-of-a-brain-signal

4. Richárd, N. 2018. The differences between Artificial and Biological Neural Networks. Date of retrieval 5.12.2019. Available: https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7

5. Bre, F., Giminez, J. 2017. Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks. Cited 6.12.2019. Available: https://www.researchgate.net/publication/321259051_Prediction_of_wind_pressure_coefficients_on_building_surfaces_using_Artificial_Neural_Networks

6. Dormehl, L. 2019. What is an artificial neural network? Here's everything you need to know. Date of retrieval 6.12.2019. Available: https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/

7. Moawad, A. 2018. Neural networks and back-propagation explained in a simple way. Date of retrieval 13.12.2019. Available: https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e

8. Brownlee, J. 2016. Crash Course On Multi-Layer Perceptron Neural Networks. Date of retrieval 8.12.2019. Available: https://machinelearningmastery.com/neural-networks-crash-course/

9. Ahirwar, K. 2017. Everything you need to know about Neural Networks. Date of retrieval 16.12.2019. Available: https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491

10. Hu, W. 2018. Towards a Real Quantum Neuron. Cited 8.12.2019. Available: https://www.researchgate.net/publication/323775654_Towards_a_Real_Quantum_Neuron

11. Kohl, N. 2010. Answer to Role of Bias in Neural Networks on Stackoverflow. Date of retrieval 16.12.2019. Available: https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks

12. Tiwari, S. 2019. Activation functions in Neural Networks. Date of retrieval 17.12.2019. Available: https://www.geeksforgeeks.org/activation-functions-neural-networks/

13. Liu, D. 2017. A Practical Guide to ReLU. Date of retrieval 17.12.2019. Available: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

14. Sharma V, A. 2017. Understanding Activation Functions in Neural Networks. Date of retrieval 17.12.2019. Available:

https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

15. Ng, A. 2011. Backpropagation Algorithm. Date of retrieval 18.12.2019. Available: https://www.coursera.org/lecture/machine-learning/backpropagation-algorithm-1z9WW

16. McDonald, C. 2017. Machine learning fundamentals (I): Cost functions and gradient descent. Date of retrieval 18.12.2019. Available: https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220

17. Murphy, K. 2012. Machine Learning: A Probabilistic Perspective. Cited 18.12.2019. Available: https://books.google.fi/books?id=NZP6AQAAQBAJ&pg=PA247&redir_esc=y#v=onepage&q&f=false

18. Kingma, D., Ba, J. 2014. Adam: A Method for Stochastic Optimization. Cited 18.12.2019. Available: https://arxiv.org/abs/1412.6980

19. Christensson, P. 2019. RGB Definition. Date of retrieval 10.12.2019. Available: https://techterms.com/definition/rgb

20. Courtney, J. 2001. Application of Digital Image Processing to Marker-free Analysis of Human Gait. Cited 10.12.2019. Available: https://www.researchgate.net/publication/267210444_Application_of_Digital_Image_Processing_to_Marker-free_Analysis_of_Human_Gait

21. Dertat, A. 2017. Applied Deep Learning - Part 4: Convolutional Neural Networks. Date of retrieval 11.12.2019. Available:

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

22. Cornelisse, D. 2018. An intuitive guide to Convolutional Neural Networks. Date of retrieval 11.12.2019. Available: https://medium.com/free-code-camp/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

23. Prabhu 2018. Understanding of Convolutional Neural Network (CNN) — Deep Learning. Date of retrieval 12.12.2019. Available: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

24. Corvil.com 2019. What is the difference between 'SAME' and 'VALID' padding in tf.nn.max_pool of tensorflow? Date of retrieval 12.12.2019. Available: https://www.corvil.com/kb/what-is-the-difference-between-same-and-valid-padding-in-tf-nn-max-pool-of-tensorflow

25. Saha, S. 2018. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. Date of retrieval 16.12.2019. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

26. Dertat, A. 2017. Applied Deep Learning - Part 1: Artificial Neural Networks. Date of retrieval 16.12.2019. Available: https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6?#860e

27. Deshpande, A. 2016. A Beginner's Guide To Understanding Convolutional Neural Networks. Date of retrieval 16.12.2019. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

28. Shah, T. 2017. About Train, Validation and Test Sets in Machine Learning. Date of retrieval 6.1.2020. Available: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

29. Cai, E. 2014. Machine Learning Lesson of the Day – Overfitting and Underfitting. Date of retrieval 6.1.2020. Available: https://chemicalstatistician.wordpress.com/2014/03/19/machine-learning-lesson-of-the-day-overfitting-and-underfitting/

30. Sharma, S. 2017. Epoch vs Batch Size vs Iterations. Date of retrieval 6.1.2020. Available: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

31. Srivastava, N. et al. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Cited 6.1.2020. Available: http://jmlr.org/papers/v15/srivastava14a.html

32. D., F. 2017. Batch normalization in Neural Networks. Cited 6.1.2020. Available: https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c

33. Convolutional Neural Network (CNN). Date of retrieval 6.1.2020. Available: https://www.tensorflow.org/tutorials/images/cnn

34. Keras team, 2019. Train a simple deep CNN on the CIFAR10 small images dataset. Date of retrieval 6.1.2020. Available: https://github.com/keras-team/keras/blob/master/examples/cifar10_cnn.py