FAKULTÄT FÜR PHYSIK
Institut für Experimentelle Kernphysik

# Eventklassifikation am Castor-Kalorimeter mithilfe von CNNs (Classification of Events with CNNs at the Castor Calorimeter)

Bachelorarbeit
von

Isabel Haide

am Institut für Experimentelle Kernphysik

Reviewer: Prof. Dr. F. Bernlochner
Second Reviewer: Dr. R. Ulrich

Bearbeitungszeit: 15.03.2018 – 01.07.2018

# Erklärung zur Selbstständigkeit

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der gültigen Fassung vom 17.05.2010 beachtet habe.

Karlsruhe, den 01.07.2018, _____

Isabel Haide

Als Ansichtsexemplar genehmigt von

Karlsruhe, den 01.07.2018, _____

Prof. Dr. F. Bernlochner

# Contents



# List of Figures

# List of Tables

# 1. The detector

## 1.1 The CMS Experiment

The large hadron collider (LHC) at Geneva, Switzerland, is build to accelerate protons to very high velocities. In two adjacent beamlines, proton bunches travel in opposite directions around the ring. Eventually, those bunches collide at four crossing points, which are surrounded by seven detectors. One of those detectors is the CMS experiment (see fig. 1.1). CMS stands for Compact Muon Solenoid, a general-purpose detector built in several layers. In the centre of the detector is the interaction point where the proton-proton collisions occur.

Around this interaction point the different kinds of detectors are build in layers around the beamline. Those track and identify the particles produced by the proton-proton collision. These layers consist of the tracker to identify the momentum of the particle and the electromagnetic followed by the hadronic calorimeter to measure the energy of various particles. The name giving solenoid magnet outside of the calorimeters produces a magnetic field of 3.8 T to curve the paths of the particles inside the tracker. To detect muons, which penetrate the iron of the calorimeters, muon chambers are installed outside of the CMS magnet. [4]

To give the angle of a particle relative to the beam axis, the pseudorapidity $\eta \equiv -\ln\left[\tan\left(\frac{\theta}{2}\right)\right]$ is used. Here $\theta$ gives the angle between the particle impuls $\vec{p}$ and the positive direction of the beam axis. Most of the time particles with a high pseudorapidity are not measured, as they escape the detector alongside the beam.

## 1.2 The CASTOR Calorimeter

The CASTOR calorimeter is part of the CMS detector. CASTOR stands for „Centauro And Strange Objects Research“. It is a very forward detector, covering the pseudorapidity range of -6.6 $< \eta <$ -5.2 . CASTOR is a electromagnetic and hadronic calorimeter which utilizes the Cherenkov effect to detect and classify particles. The calorimeter is divided into 16 sectors azimuthally and has 14 segments of reading units or modules along the longitudinal axis. The total number of channels is therefore 224. The first two modules of every tower are electromagnetic reading units (EM modules) with a thickness of 7 mm each, which combine to 20.12 $X_0$. The remaining 12 modules form the hadronic calorimeter (HAD modules) with a thickness of 14 mm each, therefore with a length of 9.504 $\lambda_I$. Its active material are quartz plates (2 mm thickness in the EM modules, 4 mm in the HAD modules) with tungsten

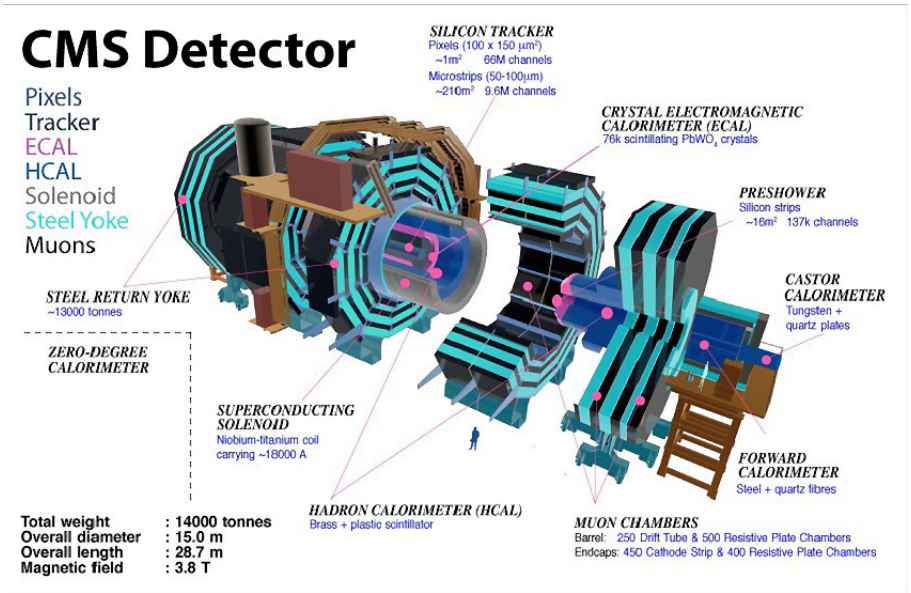


**Figure 1.1:** The CMS detector including its layers. The beam line is surrounded by the tracker and various calorimeters, as well as the superconducting solenoid and the muon chambers (source: [1]).

absorbers inbetween (5 mm in EM, 10 mm in HAD modules). The entire detector has a length of 10.30 $\lambda_I$, an inner radius of 3.7 cm and an outer radius of 14 cm. Outside of that the readout and infrastructure components are located, restricted to an outer radius of 30 cm given by the radiation shielding of CMS. Since CASTOR is therefore between the beam line and the radiation shield, it had to be built to withstand high radiation levels as well as high magnetic fields from the CMS magnet. During high luminosity runs with p-p collisions CASTOR has to be removed from the beam line to avoid back scattering of particles into CMS. For runs with low luminosity or Pb-Pb collisions it has to be easily reinstalled. For this reason the detector can be split into two halves, each containing eight towers, to be removed from the beam line and then lifted up by a crane. [5]

If a particle produced by the initial collision at the interaction point reaches the calorimeter, it initiates a shower of secondary particles. These produce Cherenkov light, which can be detected by CASTOR. The quartz plates and tungsten absorbers have a 45° inclination in respect to the beam axis to maximize the production of Cherenkov light. The photons produced by the Cherenkov effect are then transmitted to photomultiplier tubes (PMTs) with the aid of aircore lightguides. The PMTs transform the light into an electric signal. Each PMT corresponds to one of the 224 modules. As the detector does not compensate the effect that electromagnetic particles produce more Cherenkov photons and therefore a higher energy response than hadronic particles, it is called a non-compensating calorimeter.

As the detector is positioned very near the interaction point and because of its geometry it is impossible to evaluate pileup events. For that reason and to reduce the amount of radiation CASTOR is only installed during runs with low luminosity, where the number of collision per bunch crossing is $\approx 1$.

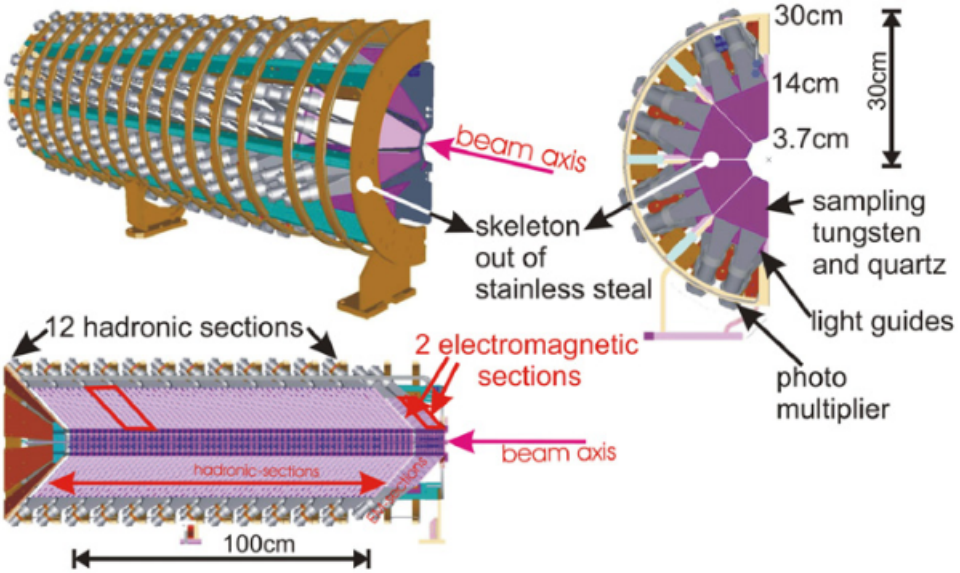


**Figure 1.2:** The CASTOR calorimeter. It is built in two parts around the beam line, with eight towers respectively, each tower containing two electromagnetic and twelve hadronic reading units. The reading units have a 45° inclination respective to the beam axis to maximize light production (lower left). (source: [2]).

CASTOR can provide information on a number of topics. One of these is the hypothesis of strangelets, a hypothetical particle consisting of up, down and strange quarks. Particles with a small number of strange quarks are unstable as the strange quark decays into the lighter quarks via the weak interaction. The „strange matter hypothesis“ states that particles with a higher number of strange quarks and roughly the same number of up and down quarks may be stable. As such particles have a very high mass, they could mostly be detected in the very forward region of the CMS detector. Thus, CASTOR may be able to observe such events.

# 2. Neural networks

Artificial neural networks were inspired by the working principle of the brain. The idea was to program networks which were able to learn how to perform certain tasks simply by getting examples. Neural networks generally don't have any task-specific rules implemented, they learn the important features by themselves.

The programming idea was to model the neurons in a biological brain. Such artificial neurons (also called connected units or nodes) are linked to each other and transmit signals. The neuron receives the signal and combines it with a certain weight and bias before sending it to the next connected unit. In some cases several input signals are merged with a certain function before reaching the neurons. The weight can be adjusted during the learning process to activate or deactivate certain parameters. It can also be combined with a threshold to only sent a signal if the combined input is over a certain number.

Usually, nodes are combined into layers. The input has to go through several layers connected to each other in different ways, transforming the input with different functions. Different layers normally identify different features of the input to get the required information at the end.

Neural networks are mostly used for two different kinds of analysis. The first one is the prediction of a continuous variable, which is called a regression problem. The second, which is specified here, is the problem of identifying an input as part of a class, called a classification model. The goal of an artificial neural network for classification is specified most of the time. It ranges from image recognition to translating to game play and medical diagnosis.

## 2.1 Image recognition

Image recognition has always played a huge part in neural networks. Current state-of-the-art networks in image recognition have now reached human or even superhuman performance. Image recognition with neural networks is normally placed within the „deep learning“ category. Such architectures include successive layers with nonlinear processing units. Each layer is connected to its predecessor and uses the output of the latter as input. This enables the network to learn different features, higher layers then correspond to higher levels of abstraction.

For image recognition convolutional neural networks (CNNs) are the currently the best performing networks. The problem with conventional multilayer perceptrons

are the computing power needed. As they are fully connected, meaning that every node in one layer is connected to every node in the layer before and after, they scale very badly to higher resolution images. As the number of weights per node in the first hidden layer with an image of size a*b*c (a stands for the width, b for the height, c being the number of colors, 1 for black/white, 3 for RGB) is a*b*c, the number of weights drastically increase for higher resolution pictures. Fully connected layers also waste computing power, as pixels that lie close together should be more connected than pixels far apart to recognize certain features. During the propagation through the network spatially local input features should be highlighted in contrast to nonlocal connections.

Convolutional neural networks are dominated by three different features which make them ideal for image recognition. The first is the 3D volume of neurons, where they are classified in three dimensions, width, height and depth of colour. This ideally represents a picture. The second is the application of spatial locality. One neuron of a following layer is only connected to a small part of neurons in its predecessor. Therefore the network learns to recognize local patterns. With many such layers stacked the patterns learned get increasingly more global, the first one only recognizing lines, while the last one identifies complete features such as „cat“, „dog“ or „mouse“. At last a convolutional neural network takes translational invariance into account. To classify an image correctly, the position of the object in the image is not important. Therefore all neurons in one convolutional layer share the same weights and bias, that means, the same filter is applied while forwarding the signal. Thus one layer always recognizes the same feature, regardless of its position in the image.

## 2.2 Convolutional neural networks

A convolutional neural network contains several convolutional layers. In a convolutional layer, a window of a certain width and height moves across the input space (as seen in fig. ). One neuron in the layer corresponds to only this part of the input image in width and height but gets the whole depth of the input volume. While the information is being forwarded to the neuron one filter is being used. The dot product between the entries of the filter and the input is computed and sent to the neuron. While the window is moving across the input picture, the same filter is used to compute the input for the corresponding neuron. Therefore filters are activated if they detect a certain feature at any position in the picture. The map of neurons for one filter is called the activation map. One convolutional layer uses many different filters which are stacked along the depth dimension and thus form the complete output volume of the layer.

The hyperparameters which have to be manually tuned are firstly the size of the convolutional window. If a size of one in width and height is chosen, the window only takes one pixel into account. A bigger size means more information is being observed. The bigger the window the less localized the features detected are. Normally a kernel field size is (2,2) or (3,3). At times it may be advantageous to choose a non-symmetrical kernel size.

The second hyperparameter is the depth of the output volume. The more filters are used the higher is the depth. Every filter corresponds to a different kind of feature,

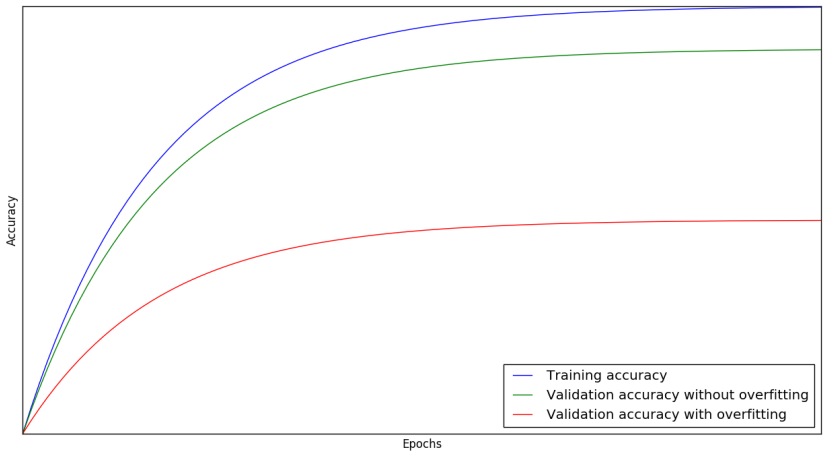


**Figure 2.1:** Training and validation accuracy. The validation accuracy follows the training accuracy well if there is little or no overfitting. In the case of strong overfitting the validation accuracy increases very little or even decreases during the training.

for example edges or localized color spaces. A higher number of filters is typically better but can also be a problem if overfitting occurs.

If the kernel size is chosen, the stride must be finetuned as well. If the stride is one, then the window only moves the filters one pixel in one direction. A high overlap of kernel windows occurs which also produces a large output volume. This can significantly increase the computing time. A normally chosen stride is 2, so the fields overlapp less and the output volume is smaller.

When the size of the input volume is not a multiple of the kernel size window, then the output dimensions differ from those of the input. Most of the time it is helpful to pad the input volume with zeros at the edges to preserve the spatial size of the input while going through layers and to minimize edge effects.

## 2.3 Overfitting

Overfitting is a problem of deep-learning networks. By choosing layers with a high number of filters the number of free parameters is very high compared to the actual number of features learnable. Therefore a network is prone to overfit the training data, performing very well during the training but failing while validating on other data (fig. 2.1).

One method to address the problem of overfitting is max pooling. Max pooling also reduces the spatial size, thus limiting the amount of computational power needed. The general idea is that only the rough location in comparison to others of the feature detected is needed, so the input is downsampled with the help of a non-linear function. First a kernel size is chosen, usually two pixels to two pixels. The kernel window moves over the input without overlapping. In every region only the maximum of this region is chosen and forwarded to the next layer. The spatial size is thus reduced by

75 %, while the depth dimension stays the same. Other pooling methods are average pooling, where the average of every kernel is chosen, or L2-norm pooling. Nowadays max pooling is being used most of the time.

Another method to reduce overfitting is dropout. With dropout a node is dropped out of the network and later reintroduced with its original weight with a possibility $p$. This forces the network to learn more robust features to better generalize the data analysis. Dropout also cuts down on computational time.

## 2.4 Activation functions

Deep neural networks need two different kinds of activation functions. The first one is to propagate the signal through the network. After the product between the input signal and the weight of one neuron is computed, the outcome is put into the activation function. Only if a nonlinear activation function is used, non trivial problems can be solved with only few nodes. The activation function is applied after every layer, meaning that every output signal is put into relation with the help of the function. The current best working activation function is the ReLu function. ReLu stands for rectified linear unit. It is calculated by

$$f(x) = \max(0, x) \tag{2.1}$$

with x as the input signal. Negative signals are therefore always set to zero while propagating through the network. Earlier activation functions were for example the sigmoid function or the tangens hyperbolicus. Unlike the ReLu function, those functions saturate when dealing with very high input signals and therefore perform worse.

The second activation function is needed at the output layer, the last layer in the network. Most deep neural networks are multi-class networks. Multi-class means that the output is not simply signal or background but identifies the input as belonging to one class of several. That means that the output activation function calculates the probability of belonging in each class in correlation to every other probability. A usual activation function in this case is the softmax function. The softmax function calculates a n-dimensional vector of random real values to a n-dimensional vector, where every number lies between 0 and 1 and the sum of all entries is 1. It is calculated by

$$\sigma(\vec{z})_j = \frac{\mathrm{e}^{z_j}}{\sum_{k=1}^{n} \mathrm{e}^{z_k}} \quad . \tag{2.2}$$

Every entry stands for the probability of the input belonging to the corresponding class. The softmax function cannot be applied if the network is not only multi-class, but also multi-label. The categories in a multi-label network are not mutually exlusive. One input can therefore be classified into several different categories, their probability to fit is not correlated to each other. In this case the beforementioned sigmoid activation function can be used. This function also squashes a vector of real numbers to a vector of real numbers between 0 and 1. Here the sum of all numbers does not have to be 1, but every entry is still the probability of belonging to this class. The function looks like

$$f(x) = \frac{1}{1 + \mathrm{e}^{-x}} \quad . \tag{2.3}$$

Choosing a class for the input signal is normally done by selecting those entries which have a number higher than 0.5, meaning a probability higher than 50% of belonging to that class.

## 2.5 Loss functions

Loss functions are a way to track the progress while training a neural network. A loss function assigns a real number to the difference between real and estimated class labels of the input, which should be minimized. During training the network changes the weights of the neurons according to the increase or decrease of the loss function. For a specific problem it is important to use a suitable loss function. For classification problems such as image recognition with more than two classes, either categorical or binary crossentropy can be used. In machine learning cross entropy is the same as logistic loss. It is calculated by

$$f(x) = y_{\text{true}} \cdot \ln(y_{\text{pred}}) + (1 - y_{\text{true}}) \cdot \ln(1 - y_{\text{pred}}) \tag{2.4}$$

where $y_{\text{true}}$ is the true label vector of the input and $y_{\text{pred}}$ the label vector of the prediction of the network.

The difference between categorical and binary crossentropy is only the way in which it is applied. With categorical crossentropy, the whole vector is compared. If only one entry differs, it is considered falsely labeled. This is useful in multi-class problems, where the labels are mutually exclusive. With multi-label problems, binary crossentropy works better. Binary crossentropy compares every label separately, therefore not penalizing one incorrect label so much.

## 2.6 Optimization algorithms

To minimize the loss function the neural network has to update its weights and biases during training. This updating is done through an optimization algorithm. To find a minimum of the loss function it is possible to evaluate the gradient of the function. The updating of the network is then proportional to the negative of the gradient. This is an iterative algorithm which finds the steepest descent to a local minimum. The size of the steps taken in the direction of the descent is also called the learning rate. The learning rate is a hyperparameter which has to be optimized to get best training results.

To compute the gradient for the whole training data takes very long. To shorten the time and computing power needed stochastic gradient descent (SDG) is implemented. SDG uses only one stochastically chosen example to compute the gradient and update the parameters. This can of course result in very varying parameter updates. To prevent this the gradient is computed over a minibatch of examples. Current convolutional networks use multiples of two, from 32 to 256 examples in one minibatch. This works because the samples in the dataset are correlated. They all depict the same or similar things, so an update computed for one batch is a good approximation for the complete set. It also can be done much more often, therefore achieving faster convergence for the loss function.

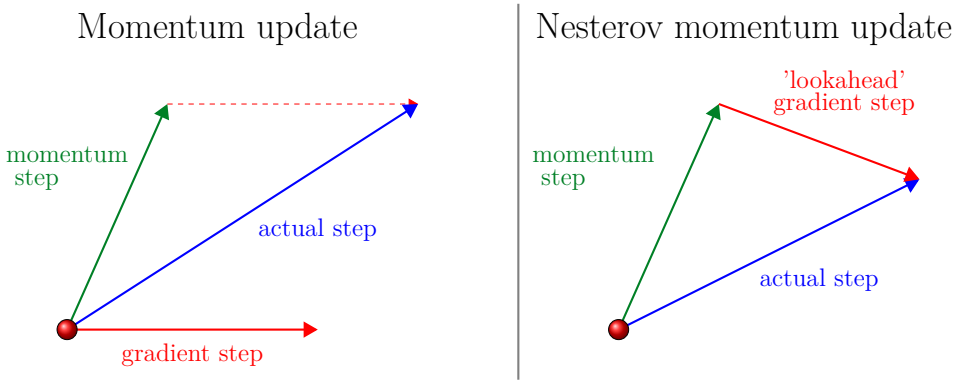


**Figure 2.2:** Classical momentum compared to Nesterov momentum. The network anticipates its future position through momentum and calculates the gradient based on that. It ends up on a slightly different position, thus converging faster. [3]

Most optimization algorithms also use momentum. Momentum can be understood by making an analogy to classical physics. If a ball at the top of a hill rolls down, it gains velocity if the direction downhill always stays the same. The hill here represents our loss function. If the gradient of a step points in the same „direction“ as the step before, the „speed“ or step size increases. If the direction changes, the step size gets smaller. If a minimum is found and the imaginary ball oversteps it, the gradient points in the other direction and momentum decreases. The minimum can so be found iteratively.

To further prevent the network from overstepping the minimum which is still a possibility even with momentum, the Nesterov Accelerated Gradient can be implemented. Instead of evaluating the gradient, changing the loss function and then applying momentum, the network anticipates what is going the happen. At first the momentum is applied to the current position. This is an approximation of the future position of the function. At this position the gradient is computed. The network there „sees“ where its going to end up in the next step and can therefore move in the right direction (see fig. 2.2). This minimizes the possibility of overstepping and decreases the convergence time. The strength of momentum is also a hyperparameter which needs to be tuned.

During the training it is helpful to decrease the learning rate over time. As the loss gets smaller, the gradient also decreases. Too high learning rates behave chaotically and cannot settle into the minimum of the loss function. To manually decrease the learning rate the common types of decay are step decay, which reduces the learning rate by a factor every few epochs, exponential decay or 1/t decay, where t is the number of epochs.

Such decays act globally and are difficult to tune. Setting the decay rate too high means severely slowing down the network, setting it too low risks erratic jumps of the loss function. Several more complicated algorithms have been developed to be used for optimizing functions. One of the current best working algorithms is the Adam algorithm ([6]). Adam is derived from adaptive moment estimation. The learning rate is adjusted for each parameter separately and based on first and second moments

of the gradient. It is bounded by a set step size and naturally annealed during the training. Adam performs well on sparse and noisy gradients and needs very little memory, as only first order gradients have to be calculated.

# 3. Data analysis

# Bibliography


[1] "Desy previous activities: Physics in the very forward region." http://cms.desy.de/sites/site_cms/content/e53612/e90916/e277689/e115497/e261362/e261366/cms_with_castor_drawing.png. Accessed: 2018-06-05.

[2] P. Göttlicher, "Design and test beam studies for the castor calorimeter of the cms experiment," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 623, no. 1, pp. 225 – 227, 2010. 1st International Conference on Technology and Instrumentation in Particle Physics.

[3] "Cs231n: Convolutional neural networks for visual recognition." http://cs231n.github.io/assets/nn3/nesterov.jpeg. Accessed: 2018-06-27.

[4] "The cms experiment at cern." https://cms.cern/detector. Accessed: 2018-06-03.

[5] "Castor detector." http://cmsdoc.cern.ch/cms/castor/html/. Accessed: 2018-06-07.

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[7] "Cs231n: Convolutional neural networks for visual recognition." http://cs231n.github.io/assets/nn3/accuracies.jpeg. Accessed: 2018-06-27.

[8] F. Chollet *et al.*, "Keras." https://keras.io, 2015.