Image classificiation and convolutional neural networks

Dawid Bitner, Daniel Broczkowski, Mateusz Kowol, Marcin Krupa
Silesian University of Technology,
Faculty of Applied Mathematics

***Abstract*—Image classification - we define it as a task of extracting information classes from a multiband raster image. Its objective is to identify and portray, as a unique gray or color level, features occurring in an image in terms of the object or type of land cover these features represent on the ground, followed by grouping of all or selected land cover features into summary categories.**

Combined with powerful Convolutional Networks (CNN), which are perfect for classification tasks, like image classification or processing natural language, we can create a great tool for classification purposes, in this case image classification. The main advantage of CNN compared to its predecessors is that it automatically detects the important features without any human supervision, as well as it is computationally efficient compared to other classification algorithms.

This article purpose is to bring the closer look to Convolutional Network and image classification in general.

I. INTRODUCTION

A. Terms related with image analysis

Image analysis is simply a process of extracting meaningful information by computers to recognize attributes within an image. The means used to do so are mostly image processing techniques. Image analysis tasks can be very simple: reading bar coded tags or as sophisticated as identifying a person from their face. There are terms closely related with image analysis, worth explaining:

1) Computer vision: CV, an abbrevation of Computer Vision, is defined as a field of study that works on enabling computers to see, identify and process content of digital images such as photographs and videos in the same way that human vision does, providing appropriate output.

Because computer must interpret what it sees, then perform appropriate analysis, we commonly link it with any type of artificial intelligence, as it is constantly processing and providing useful results based on the observation.

2) Image recognition: Image recognition is a term for computer technologies that can recognize targets such as people, animals, objects etc. through the use of algorithms and machine learning. It can be done in many different ways, but most of them involves the use of convolutional neural networks, which will be described later, for now most important information is that it was specifically set up for image recognition and similar image processing.

Using characteristic techniques for CNN such as max pooling, stride configuration and padding, convolutional neural filters work on images to help machine learning programs get better at identifying the subject of the picture.

3) Image classification: Image classification refers to the task of extracting information classes from a multiband raster image. This process categorizes all pixels in a digital image into one of several land cover classes, commonly refered as ”themes”. This categorized data may then be used to produce thematic maps of the land cover present in an image. There are two types of classification: supervised and unsupervised.

Supervised classification helps us identify examples of the information classes (i.e., land cover type) of interest in the image. These are called ”training sites”. It uses the spectral signatures obtained from training samples to classify an image.

The unsupervised classification examines a large number of unknown pixels and divides into a number of classed based on natural groupings present in the image values. It finds spectral classes (or clusters) in a multiband image without the analyst's intervention.

4) Image processing: Image processing converts an image into digital form to make operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, it can be a video frame or photograph and as an output may be image or characteristics associated with that image. It includes treating images as two dimensional signals while applying already set signal processing methods to them.

B. Convolutional Neural Network

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take image as an input and by a list of layers transform the image volume into an output volume. It can assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. It's strong point is that pre-processing in a ConvNet does not requre that much computing power compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. More about Convolutional Neural Network will be described in math section.

C. Why and where do we use it?

Image classification is enormously popular nowadays. This technology can be used in many different kind of domains like Website databases, city monitoring or intelligent cars such as popular ”Testla”.

If a visual database does not contain metadata about the images, categorizing it is a huge hassle. Classification of images through machine learning is a key solution for it.

With image recognition, companies can easily organize and categorize their database because it allows for automatic classification of images in large quantities. This helps them monetize their visual content without investing countless hours for manual sorting and tagging.

The most recent and complicated application is used in China. Mass surveillance is the complicated network of surveillance used by the Chinese government to supervise the actions of Chinese citizens. In China, mass surveillance mainly comes from the government, though non-publicized corporate surveillance is also a possibility. There are multiple ways in which the Chinese state engages in surveillance, including Internet surveillance, camera surveillance in public, social credit surveillance, and other supporting digital technologies. Chinese mass surveillance has witnessed an increased spending, intensity, and coverage in recent years.

Another example may be very simplistic. Cars fitted with computer vision, that are able to perfectly classify, identify and distinguish objects on and around the road such as traffic lights, pedestrians, traffic signs and so on, to act accordingly. The intelligent device could provide inputs to the driver or even make the car stop if there is a sudden obstacle on the road that is classified as something dangerous.

These are just some of them and it is hard to count them all. We are heading to the moment in the history when Artificial Intelligence is not just a fantasy, but it has a real impact on basically every aspect of our lives.

II. MATHEMATICAL DESCRIPTION

This section will mainly focus on Convolutional Neural Network, as it is a main brain and source of power of anyimage classification systems.

A. What are the ConvNets?

Convolutional Neural Network is an architecture of neural network that is great at detecting and hierarchical learning of features, i.e. which features are worth attention and how to process them without human supervision. Usually we present it as list of layers that transform the image volume into an output volume (e.g. holding the class scores). To some extent they are similar in construction to ordinary neural networks, but the significant difference is that ConvNet networks have been designed to process images and objects that are on them. Thanks to the use of appropriate filters, it is easier for them to capture spatial dependencies on the image. Each layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function, as well as they may have additional hyperparameters or not. The pre-processing required in ConvNet networks is much lower compared to other classification algorithms. The architecture provides a better match to the set of data consisting of images by reducing the number of parameters involved and the reusability of weights, making the forward function more efficient in implementation.

The ConvNet network architecture consists of several layers that transform the original image layer by layer to the values needed to classify, such as:

- input layer
- convolution layers
- activation layers
- pooling layers
- fully-connected layers

B. Input layer

In the ConvNet network, where we give an image to the input, the neurons are arranged in three dimensions: width, height and depth. Neurons in the layer will only be connected to the small area of the previous layer instead of the full connection. The output layer will reduce the full image to a single classification vector, located along the depth dimension. It is important to reduce images to a form that is easier to process, without losing the features that are key to getting a good result. It is important that our architecture should scale to huge data sets.

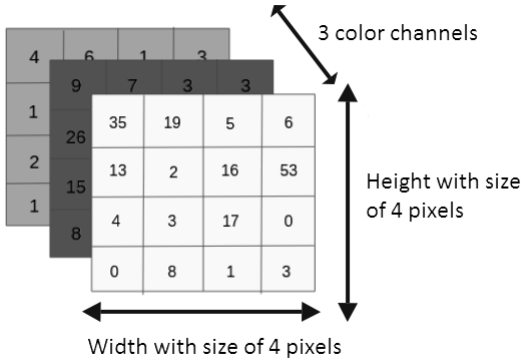

Fig. 1. Example of an input in form of an image in ConvNet

C. Convolutional layers

Main block of the CNN network. They contain a set of filters that are learnable. Convolution is a mathematical operation to merge two functions, creating a new third. In case of a network where an image is input, the convolution is used on the input data by the convolutional filter (kernel) to create a feature map. We perform the convolution operation by sliding this filter over the input. Matrix multiplication is performed and the sum is written on the feature map.

The above example is made in 2D. But in ConvNet networks we performe convolutions on images, represented as a 3D matrix with three dimensions: height, width and depth, where the depth represents color channels. Kernel used over the image must also be 3D, as it covers the entire depth of the input.

Many convolutions are made over the imput, each using a different filter and resulting in a separate feature map. Later, the combination of all these feature maps becomes our final result of the convolutional layer.

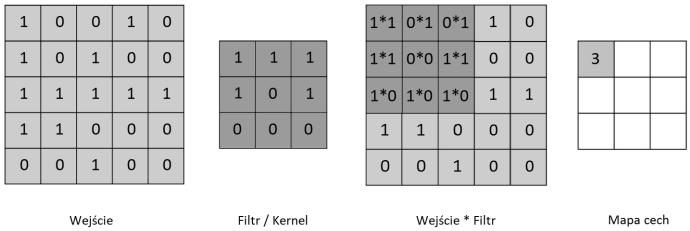

Fig. 2. Example use of 3x3 filter on two dimensional input data. Kernel moves on top of the input, and the sum of the operation goes to the feature map.

Generally, the purpose of the convolution is to extract features such as edges from the input image. ConvNets do not have to be limited to only one layer of convolution. Traditionally, the first ConvLayer is responsible for capturing features such as edges, color, gradient descent etc., and by adding more layers, you can expand and tune the architecture. We can get a network that understands images similar to us.

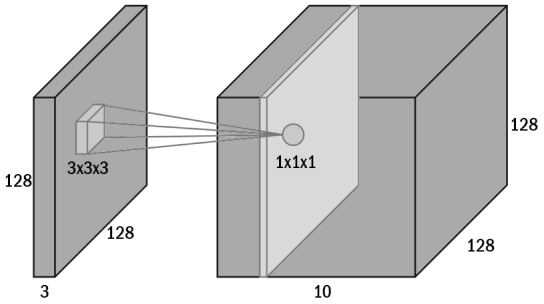

Fig. 3. Example use of 3x3 filter on two dimensional input data. Kernel moves on top of the input, and the sum of the operation goes to the feature map.

Example: above image shows a 128x128x3 image, the kernel has the size of 3x3x3. When the filter is over a specific location and covers a small volume of an input, we perform a convolution operation similar to the 2D example, except that this one takes place in three dimensions. As a result we still get a scalar. By sliding the filter over the input, we calculate the next values, and properly record them on the feature map of size 128x128x1. The size of width and heigh does not change due to the use of padding, as we use this technique to maintain the same dimensionality. In order to do that, we surround the input with zeros or values on the edges. What changes is the depth of an image, it is reduced to just one channel.

If we are using 10 different filters, we get 10 different maps of features with a size of 128x128x1. Putting them along the depth dimension would give us the final result of the convolutional layer: a volume of size 128x128x1. The convolution operation for each filter is done independently, and the resulting feature maps are disjoint.

D. Activation layer

ConvNe is no exception when it comes to using the activation function between layers. There are many functions to pass the result of the convolution operation. Most popular ones are ReLU and sigmoid functions. Using the activation function, we must keep in mind that the values in the final feature maps are not really sums, but the values after applying activation function to them, and that each type of convolution requires an activation function, without them the network will not reach its full potential.

ReLU formula:

$$R(z) = max(0, z)$$

$$z = Wx + b$$

Sigmoid formula:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

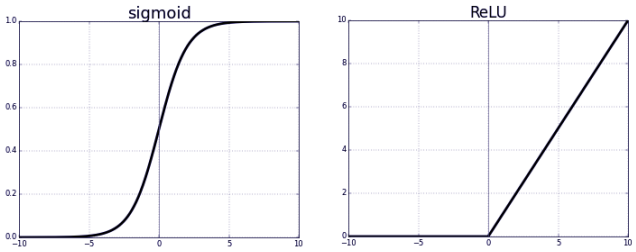

Fig. 4. Comparison of ReLU and sigmoid on the graph.

ReLU is more common for ConvNets for several reasons: it has reduced gradient probability (change in the direction of intensity or color of the image), it is more computationally efficient for calculation than sigmoid functions (ReLU only needs to select $max(0, z)$ and does not perform costly exponential operations).

E. Strides, max-pooling, padding

Stride is a parameter that determines by how many values we are overlapping the convolution filter over an image at each step. We can decide on the value of it - if it is large, the resulting feature map is smaller because we are skipping the potential locations.

Using strides we create a feature map that is smaller than the size of the of the input, because the convolution filter must be contained in the input data. If we want to maintain the same dimensionality, we can use padding to surround the input with zeros or values that are on the edge of an image. It is a common technique to keep the size of the feature maps to prevent shrinkage on each layer. Just like in the two dimensional picture, we use them on convolutional layers that are in 3D.

Next is pooling that reduces ”dimensionality” of an input, which is a convolutional layer, to lower the number of parameters. It shortens the training time and prevent from overfitting. If the network is large, this technique can reduce the number of weights between layers a lot. Height and width of the feature map are halved, but the depth does not change because pooling works independently on each depth slice the input. Commonly

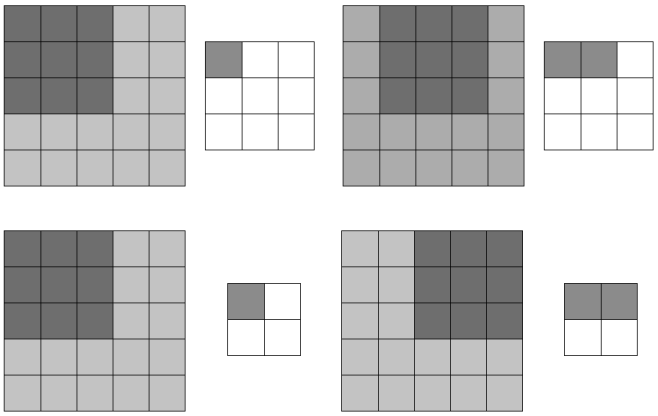

Fig. 5. Comparison of ReLU and sigmoid on the graph.

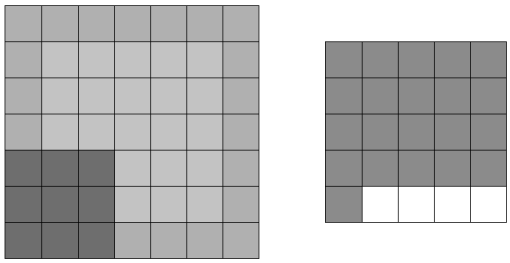

Fig. 6. Example of padding.

we use max-pooling, which gets the maximum value in the "operation window". Max-pooling has no parameters, it is just an process of slidding this operation window over the input and getting the maximum value. The window size and stride are predefined. To generalize pooling layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S
- It produces a volume of size $W_2 \times H_2 \times D_2$, where:
  - $W_2 = \frac{W_1 F}{S} + 1$
  - $H_2 = \frac{H_1 F}{S} + 1$
  - $D_2 = D_1$

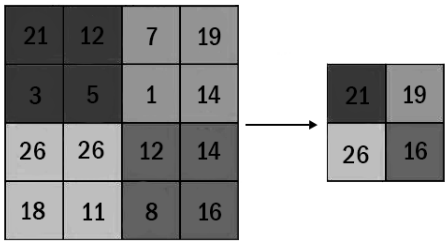

Fig. 7. Max-pooling.

F. Hyperparameters

These are parameters that we should consider more wisely when creating a ConvNet network. They include:

- Filter size - if it is small, more detailed information can be extracted from the image and reused later (it can also act on the contrary: completely useless information can be extracted), slows down decrease in size of an image. Larger filters isolate quite general features in the image, the amount of information extracted is much smaller. Fast image size reduction makes the network quite shallow.
- Number of filters - the most alternating parameter in CNN network, usually a value between 32 and 1024. Using more filters gives a more powerful model, but we risk overfitting due to the increased number of parameters. Usually we start with a small number of filters in the initial layers and gradually increase the number when we go deeper into the network.
- Stride and Padding

G. Fully-connected layer

After convolution and max-pooling, we add several fully-connected layers to complete the CNN architecture. These layers require 1D numbers for input, while convolution and pooling layers have a 3D vector. We must perform *flattening*.

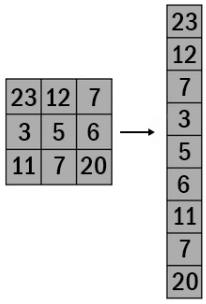

Fig. 8. Flattening the input to a one dimensional vector.

The flattened output signal is fed to the neural network, and as in the classical network, backward propagation applied to each iteration. After a series of eras, the model is able to distinguish between dominant and irrelevant features in images and classify them using the *Softmax* classification technique, that converts numbers into probabilities that sums up to one. Softmax derives a vector that represents the probability distributions of the list of potential results.

Softmax:

$$P\left(y = j|\sigma(z)_i\right) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$\sigma = w_0 x_0 + w_1 x_1 + ... + w_k x_k = \sum_{i=0}^{K} = w^T x$$

To put fully-connected network in more mathematical way:

- Let $x \in R_m$ represent the input to a fully connected layer.

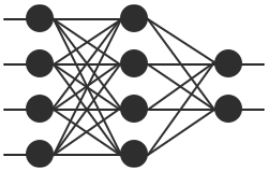

Fig. 9. Fully connected layer schema.

- Let $x_i \in R_m$ be the i-th output from the fully connected layer.

Then $y_i \in R$ is computed as follows:

$$y_i = w_1 x_1 + ::: + w_m x_m)$$

If we use ReLU activation function, and say w and i are learnable parameters in the network, the full output y is as follows:

$$y = \sigma(w_{1,1} x_1 + ... + w_{1,m} x_m)$$

H. Dropouts

Dropout is used to prevent overfitting, that means we exclude neurons intentionally to prevent the network from being too dependent on a small number of neurons and forcing each neuron to act independently. Dropout is used on the input layers and hidden layers, but we do not use them on the output layer. During training in each iteration, the neuron is temporarily "dropped" or disabled with the probability *p*. This means that all inputs and outputs to this neuron will be turned off at the current iteration. Abandoned neurons are re-sampled with the probability *p* at each subsequent training step, so an dropped neuron in one step can be active in the next step. The hyperparameter *p* is called the dropout-rate or a degree of decay and is usually about 0.5, which means that about 50% of neurons are falling out.

I. Loss function, cross-entropy

Loss function is used to measure the inconsistency between predicted value $(\widehat{y})$ and actual label (y). It is a non-negative value, where the robustness of model increases along with the decrease of the value of loss function. Loss functions fall under four major category:

- Regressive loss functions - used in case of regressive problems, that is when the target variable is continuous. Most widely used regressive loss function is Mean Square Error.

$$MSE\left(\widehat{\theta}\right) = E\left(\left(\widehat{\theta} - \theta\right)^2\right)$$

  $\widehat{\theta}$ is an estimator of θ.
- Classification loss functions - as an output in classification is a value f(x), called the score for the input x. The magnitude of the score represents the confidence of our prediction. The target variable y, is a binary variable, 1 for true and -1 for false.
- Embedding loss functions: - deals with problems where we have to measure whether two inputs are similar or dissimilar.

Cross-entropy is most common of all classification loss functions. It loss increases as the predicted probability diverges from the actual label. In information theory, the cross entropy between two probability distributions p and q over the same underlying set of events measures the average number of bits needed to identify an event drawn from the set if a coding scheme used for the set is optimized for an estimated probability distribution q, rather than the true distribution p. The cross entropy for the distributions p and q over a given set is defined as follows:

$$H(p, q) = E_p\left[-\log q\right]$$

If we set up notation $p \in \{y, 1-y\}$ and $q \in \{\widehat{y}, 1-\widehat{y}\}$ we use cross entropy to measure dissimilarity between p and q:

$$H(p, q) = -\sum_i \log q_i = -y \log \widehat{y} - (1-y) \log\left(1-\widehat{y}\right)$$

The typical cost function that one uses in logistic regressionis computed by taking the average of all cross-entropies in the sample. For example, suppose we have N samples with each sample indexed by $n = 1, ..., N$. The loss function is given by:

$$J\left(W\right) = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \widehat{y} + \left(1 - y_n \log\left(1-\widehat{y}\right)\right)\right]$$

where $\widehat{y} = g\left(w\dot{x_n} = \frac{1}{1+e^{-w_n}}\right)$, $g(z)$ is the logistic function as before.

J. Optlimazators

During the network training process, we always adjust and change the parameters (weights) of our model to minimize the loss function and make our predictions as correct as possible. For this purpose, we use optimizers that update the model in response to the loss function result.

The classic example is gradient descent, used to decreased the cost function:

$$w_{n+1} = w_n - \eta \frac{\partial}{\partial w_n} J(w_n)$$

But in ConvNets we use more sophisticated methods. One is Adaptive Moment Estimation (Adam). It computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t, it also keeps an exponentially decaying average of past gradients m_t. We compute the decaying averages of past and past squared gradients m_t and v_t using these formulas:

$$m_t = \beta_1 m_{t-1} + \left(1-\beta_1\right) g_t \rightarrow mean$$

$$v_t = \beta_2 m_{t-1} + \left(1-\beta_2\right) g_t^2 \rightarrow uncentered\, variance$$

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients. As m_t and v_t are initialized as vectors of 0's, they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e. β_1 and β_2 are close to 1).

We counteract these biases by computing bias-corrected first and second moment estimates:

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$$

$$\widehat{v_t} = \frac{v_t}{1-\beta_2^t}$$

They then use these to update the parameters, which yields the Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v_t}} + \epsilon}\widehat{m_t}$$

Proposed default values are: 0.9 for β_1, 0.999 for β_2 and 10^{-8} for ϵ.

III. PROPOSED SYSTEM

System that will be used for classification of 11 animals:

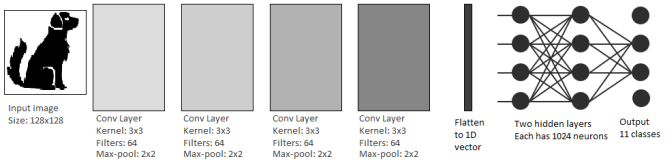

Fig. 10. Classification of animals step by step.

1) This setup is made of 4 convolutional layers in total. Each one has a ReLU activation function. Weights in filters are generated randomly and the have normal distribution with standard deviation of 0.05. Kernels/filters have a 3x3 size. Weight decay 0.001. Each convolution layer has max-pooling next. Last layer, output one, uses Softmax activation function.
2) As an optimizer we use Adam, as it is good at optimizing weights according to the loss function, which is cross-entropy. It's a very classic set.
3) We have a dataset of 43.000 animals, belonging to 11 classes. The data was later splitted: 90% of the dataset is used for training, 10% for validation.
4) Neural network is fed with batches of size containing 128 training images. During training it updates it weights. For validation it does not.
5) Each step the training and validation loss are being printed, and every epoch ends with information how effective is the learning process.
6) Describing it layer-by-layer:
   - First layer: as an input takes 128x128x1 photo, so it has just one color channel, to make sure it is more computing efficient. The testing machine will be using graphic card for learning process. It applies convolution using 64 filters, creating an output of 128x128x64 for max-pooling layer. Later, the output of max-pooling layer is of size 64x64x64.
   - Second layer: input - 64x64x64, after applying 128 filters the output is of size 64x64x128. Max-pooling creates an output of 32x32x128 for next layer.
   - Third layer: input - 32x32x128, uses 128 filters output: 32x32x128, max-pooling: 16x16x128.
   - Fourth layer: input - 16x16x128, uses 256 filters output: 16x16x256, max-pooling: 8x8x256.
   - Flattening: 8x8x256 = 16384 input neurons.
   - Two fully-connected hidden layers, each has 1024 neurons.
   - Output layer, has 11 neurons, uses Softmax activation function.

IV. EXPERIMENTS

As described earlier, the dataset consists of 43000 images of animals, that comes from different distributions. Images were taken in different light, the animals were turned in diffrenet direction (some images had to be flipped), but also the image themselves were of different quality. Because of such a vast imperfections in the dataset, zero-centering was used. It increased the accuracy by $2-3\%$.

WITHOUT ZERO-CENTERING:

```
Training Step: 10815  | total loss: 0.12775 | time: 108.192s
| Adam | epoch: 012 | loss: 0.12675 - acc: 0.9604 -- iter: 41472/41555
Training Step: 10816  | total loss: 0.12665 | time: 115.106s
| Adam | epoch: 012 | loss: 0.12473 - acc: 0.9607 | val_loss: 0.08657 - val_acc: 0.7755 -- iter: 41555/41555
```

WITH ZERO-CENTERING:

```
Training Step: 10815  | total loss: 0.12675 | time: 107.190s
| Adam | epoch: 012 | loss: 0.12675 - acc: 0.9604 -- iter: 41472/41555
Training Step: 10816  | total loss: 0.12565 | time: 117.196s
| Adam | epoch: 012 | loss: 0.12565 - acc: 0.9614 | val_loss: 0.08657 - val_acc: 0.8001 -- iter: 41555/41555
```

Fig. 11. With and without use of zero-centering, same amount of epochs.

Experiments included changing filter number on every convolutional layer. If the number was small, it learned rapidly, but the results were far from good, no matter the number of epochs or batch size. Network with more filters on each of convolutional layers, or with more convolutional layers started overfitting quite fast. This occurs when our model perform unusually well on its training data, but very poorly on new, unseen data. Model described earlier was just a decent balance.

More experiments were made creating a model with Adam as it's optimizer. 80% was a maximum this network could get using Adam, and it is a good practice to change later on Adam to something else, like Stochastic gradient descent (SGD) or RMSprop. Both were tested, both failed.

The reason SGD failed is because it is SGD. Meaning behind it is that SGD usually achieves to find a minimum, but it might take significantly longer than with some of the optimizers, it is much more reliant on a robust initialization and annealing schedule, and may get stuck in saddle points rather than local minima. It also takes much more time to get a decent results. That was the case.

```
Training Step: 22655  | total loss: 0.20861 | time: 157.238s
| SGD | epoch: 020 | loss: 0.20861 - acc: 0.9592 -- iter: 41472/41555
Training Step: 22656  | total loss: 0.21735 | time: 169.188s
| SGD | epoch: 020 | loss: 0.21735 - acc: 0.9563 | val_loss: 0.09538 - val_acc: 0.7891 -- iter: 41555/41555
```

Fig. 12. Use of SGD. Accuracy drops down.

Some of the graphs for a tested model:

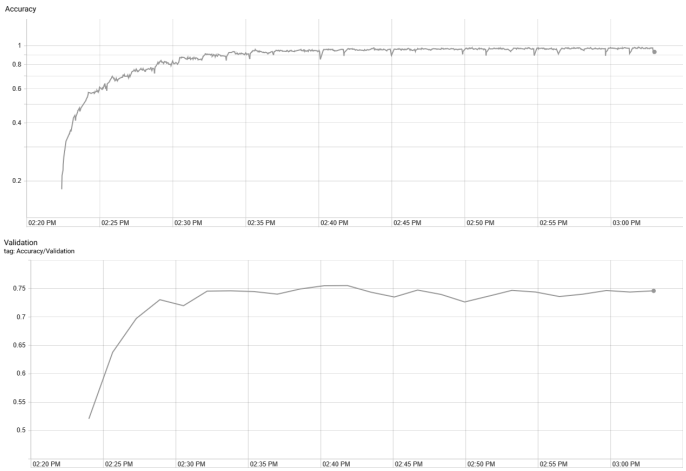

Fig. 13. Graph shows the accuracy for training and validation data.

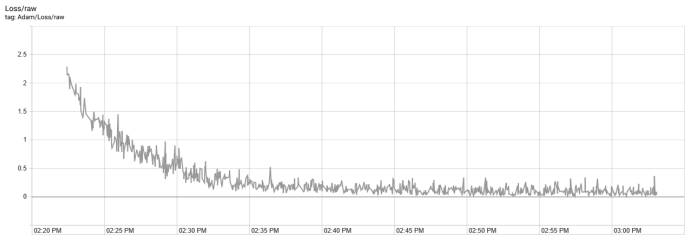

Fig. 14. Graph shows the Adam optimizer loss.

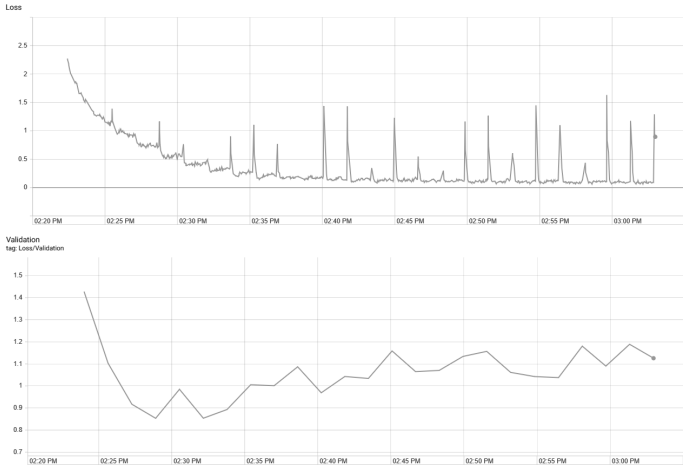

Fig. 15. Graph shows the loss on training and validation data.

V. CONCLUSIONS

Not perfect, but a very decent model created with around $78 - 80\%$ of accuracy. Even with the output size of 8x8x256 at the last convolutional layer after max-pooling, it did great at extracting essential features from an image, proving that convolutional neural network are a powerful tool for image classification.

Dataset consist of 43000 images, almost 4000 image per class. Network recognizes very well images that are similar to stock photos, and do very well with images that have clear contour of animal silhouette. Otherwise, it classifies wrong.

How to improve this network? Firstly, using 3 color channels in the input image instead of just one grayscale. Color at natural image processing is very important, but it takes more computing power that the testing machine did not have. Secondly, improving the database to be more vast at covering differences between the animals. Lastly, just spending more time on tuning the network by adjusting the kernel size, increasing number of layers and number of filters, giving more time to SGD to prove itself etc.

VI. BIBLIOGRAPHY

- Dertat A., *Applied Deep Learning - Part 4: Convolutional Neural Networks*, https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2, 2019-06-06
- Karpathy A., *CS231n Convolutional Neural Networks for Visual Recognition*, http://cs231n.github.io/convolutional-networks/, 2019-06-06
- *Overfitting in Machine Learning: What It Is and How to Prevent It*, https://elitedatascience.com/overfitting-in-machine-learning, 2019-06-06
- Reppel E., *Visualizing parts of Convolutional Neural Networks using Keras and Cats*, https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59, 2019-06-06
- Ruder S., *An overview of gradient descent optimization algorithms*, http://ruder.io/optimizing-gradient-descent/index.html, 2019-06-06
- Saha S., *A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way*, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, 2019-06-06
- Sahoo S., *Deciding optimal kernel size for CNN*, https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363, 2019-06-06
- *Supervised Learning*, https://www.techopedia.com/definition/30389/supervised-learning, 2019-06-06