Introduction

The project is split into the following three parts: (1) the derivation of the general case for a convolutional neural network (CNN) with an arbitrary input of a tensor of order 4 for usage in our own neural network library implementation, (2) the systematic construction and evaluation of a fully convolutional model for detecting a variable number of faces for real time video, and (3) the improvement of temporal convolutional networks (TCNs) and their application on a complex task which they have never been used for before, namely, automatic image captioning.

Purpose : General Derivation of a CNN

Advanced concepts of CNN training utilizes tensors of order 4. When we first started out in the field of deep learning, we found that the rigorous derivation for this general case was missing in the current literature. Since four spatial dimensions can't be visualized in an effective manner, the literature often only explains the simple case of an input of order 2. However, concepts such as Batch Normalization [**?**], which revolutionized the training of CNNs, requires an input of order 4 to function. Ignoring the more complex order 4 case hinders students from reaching their full potential. Therefore, we present a thorough derivation of a CNN with an arbitrary input of order 4, with varying strides and zero-padding, including both forward and backpropagation.

This derivation does not only serve as an educational resource for anyone learning about CNNs, but also serves as a baseline for our own open-source Python and C++ implementation of convolutional and feed-forward neural networks, using only elementary math operations, including both forward and backpropagation.

Purpose : Dense Face Detection

Conventional face tracking and detection methods are limited to a few or just a single face. The aim of this paper is additionally to construct a dense face detector using a fully convolutional neural network which will be capable of detecting a variable number of faces for real time video. This method should serve as a baseline to be applicable for CCTV or other security systems, or areas such as person tagging on social media and face detection for digital cameras.

There exists multiple moderns techniques and network architectures used for the training of multi-class object detectors. The additional aim of this paper is to systematically and empirically investigate how the most successful and popular methods and techniques, such as Feature Pyramid Networks [**?**], Focal Loss [**?**], Online Hard Example Mining [**?**], and data augmentation affects the performance of a single class object detector. Since object detectors are often specifically used for multi-class classification and detection, the field currently lacks structured data and empirical evaluations on how these methods affect binary region classification problems (e.g. face or no face).

Purpose : Improving TCNs for Image Captioning

Additionally, the purpose of this project is to improve TCN in two areas which we have found the original implementation in [**?**] is lacking. Firstly, the authors PyTorch implementation of TCNs is slightly computationally inefficient due to their zero-padding scheme. We propose a different zero-padding scheme, which final result is mathematically equivalent to the implementation in [**?**], but is slightly more computationally efficient, while still making sure there's no information leakage from the future into the past. Secondly, we propose a new dilation scheme which takes inspiration from how conventional spatial state of the art CNNs function. These changes should increase both the efficiency and accuracy of the models, and we empirically evaluate them on two sequential modeling tasks. Furthermore, we will compare three different residual building blocks to see their effect on the accuracy of the model.
Improving TCNs, which in a recent study [**?**] has been found to perform better than its recurrent counterparts, has an effect on every sequential task. Showing empirical evidence of how TCNs perform better than GRUs and LSTMs, which are traditionally used for sequential tasks, lies in everyone's best interest to further the state of the art in sequential modeling. This can lead to the eventuality of replacing every recurrent network with its temporal convolutional counterpart, while increasing the accuracy on any task. Because of this, this paper investigates how TCNs compare to their recurrent counterpart, a Gated Recurrent Unit (GRU), on a task which TCNs have never been applied to before, namely automatic image captioning.

7. References

General Derivation of a CNN

1 General Derivation of a CNN

1.0.1 Forward Propagation

CNNs learn to apply a set of sequential discrete convolutions on a given signal. The convolution operation is given by $*$. A kernel moves along a set of spatial positions with a stride s, and is multiplied element-wise with every neuron on a subset of the signal (SEE FIGURE).

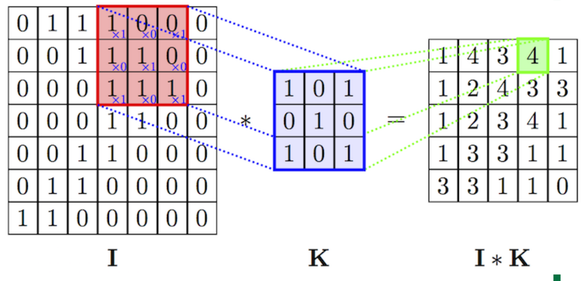

Figure 1: A kernel of size $1 \times 1 \times 3 \times 3$ convolving over activations of size $1 \times 1 \times 7 \times 7$, producing activations of size $1 \times 1 \times 4 \times 4$ in the next layer IMAGE CITATION SOURCE [**?**].

The basic building block of the convolutional neural network is the convolution and the convolutional layer. It uses the learnable parameters $W^{(l)} \in \mathbb{R}^{C' \times C \times k \times k}$, called a kernel, and is the before mentioned filter the network learns to apply to images. C and C' are the number of channels in layer l and $l+1$ respectively. k is a variable called the kernel size of the convolution [**?**].
The activations at layer $l+1$ are derived from the activations at layer l by applying the kernel at every possible spatial location on the activations at layer l (see figure 1). Every neuron is multiplied by the value of the kernel at the same spatial location. The sum of all the products becomes the activation of a single neuron in layer $l+1$. Applying this operation on every region of neurons in the activation is called a convolution. The convolution operator is denoted by $*$ [**?**] [**?**] [**?**].
A feature map in layer $l+1$ is the result of a single kernel of size $1 \times C \times k \times k$ being convolved over the whole activation volume of the previous layer. C' is the number of kernels a layer has and is also the number of feature maps the next layer will have [**?**] [**?**].
The kernels have two additional non-learnable hyperparameters: a stride s and zero-padding p. s is the size of the step the kernel takes when it moves from one spatial location to the next during a convolution. The convolution in figure 1 has a stride of $s = 1$. Zero-padding entails padding the edges of the activation tensor with p zeros (see figure 2). Since a convolution decreases the spatial size of the activations of the next layer, zero-padding is a way to control the size of the activations [**?**] [**?**] [**?**].

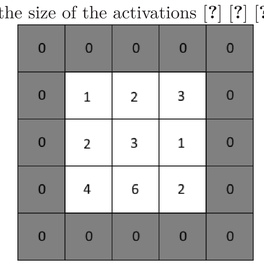

Figure 2: An activation with size $1 \times 3 \times 3$ is zero-padded with $p = 1$ and the resulting tensor is of size $1 \times 5 \times 5$. White symbolizes the original tensor and grey symbolizes the padded zeros.

Miles, 17, of Niceville High School in Let $W^{(l)} \in \mathbb{R}^{C' \times C \times k \times k}$, $X^{(l)} \in \mathbb{R}^{R \times C \times (H+2p) \times (W+2p)}$ and $X^{(l+1)} \in \mathbb{R}^{R \times C' \times H' \times W'}$. The dimensions of layer $l+1$ are defiend by equations (1) and (2) [**?**] [**?**] [**?**]:

$$W' = \frac{W - k + 2p}{s} + 1 \tag{1}$$

$$H' = \frac{H - k + 2p}{s} + 1 \tag{2}$$

The following equation defines the convolutional layer algebraically [**?**] [**?**]:

$$\begin{aligned} \left[X^{(l+1)}\right]_{r,c',h',w'} &= X^{(l)}_{r,c',h',w'} * W^{(l)}_{c'} \\ &= \sum_{c=0}^{C-1} \sum_{j=0}^{k-1} \sum_{i=0}^{k-1} X^{(l)}_{r,c,(1+sh'-s+j),(1+sw'-s+i)} W^{(l)}_{c',c,j,i} \end{aligned} \tag{3}$$

The index of the term which shall be used to convolve specifies which dimensions will be convolved upon and summed over. For instance, $W^{(l)}_{c'}$ implies that the C, H and W dimensions (all channels) should be used in the convolution, while $W^{(l)}_{c',c}$ implies that only the H and W dimension (one channel) should undergo the convolution operation.
Convolutions are in practice implemented with the functions $row2im$ and $im2row$ which enable the convolution to be computed with a single dot product [**?**] [**?**] [**?**]. The underlying math is equivalent with the equations shown in this paper. However, $row2im$ and $im2row$ are outside of the scope of this paper and are left to the reader to research if a more computationally efficient implementation is required.

1.0.2 Backpropagation

At every layer l the delta-error of the proceeding layer $\delta^{(l+1)}$ has to be backpropagated to create the delta-error at the current layer. $\delta^{(l)}$ is then used to compute the partial derivatives of the loss with respect to the weights $W^{(l)}$ to be used in the gradient [**?**] [**?**].
The backpropagation of the recursive delta-error $\delta^{(l+1)}$ is derived by the use of the chain rule. $\delta^{(l+1)} = \frac{\partial L(\theta)}{\partial X^{(l+1)}}$ is broken up into two smaller partial derivatives $\frac{\partial L(\theta)}{\partial X^{(l+1)}}$ and $\frac{\partial X^{(l+1)}}{\partial X^{(l)}}$. Additionally, because more than one single neuron in layer l is responsible for the delta-error at layer $l+1$, all the neurons of layer l have to be summed over, similar to equation (**??**), (**??**), and (**??**). This can be done since the derivative of a sum is equivalent to the sum of the derivatives of each element. $X^{(l+1)}_{r,c',h',w'}$ is then replaced by its definition from equation (3) [**?**] [**?**] [**?**] [**?**]:

$$\begin{aligned} \delta^{(l)}_{r,c,h,w} &= \frac{\partial L(\theta)}{\partial X^{(l)}_{r,c,h,w}} \\ &= \sum_{c'=1}^{C'} \sum_{h'=1}^{H'} \sum_{w'=1}^{W'} \frac{\partial L(\theta)}{\partial X^{(l+1)}_{r,c',h',w'}} \frac{\partial X^{(l+1)}_{r,c',h',w'}}{\partial X^{(l)}_{r,c,h,w}} \\ &= \sum_{c'=1}^{C'} \sum_{h'=1}^{H'} \sum_{w'=1}^{W'} \delta^{(l+1)}_{r,c',h',w'} \frac{\partial \sum_{c=0}^{C-1} \sum_{j=0}^{k-1} \sum_{i=0}^{k-1} X^{(l)}_{r,c,(1+sh'-s+j),(1+sw'-s+i)} W^{(l)}_{c',c,j,i}}{\partial X^{(l)}_{r,c,h,w}} \end{aligned} \tag{4}$$

Dense Face Detection

2 Dense Face Detection

In this section we systematically and empirically investigate how to construct and improve a convolutional-neural-network-based model to detect a variable number of faces in an image. Every factor and suggestion which affects the model performance is evaluated one by one and added to a baseline model, which we name FaceNet.

2.1 Background

2.1.1 WIDERFace

For the task of detecting a variable number of faces in an iamge, the WIDERFace dataset [**?**] was used. It is a dataset consisting of 32 203 images of a total of 393 703 number of faces at different scales, lighting and occlusions. Every training example consists of a number of bounding boxes which describes all the faces in the image. A bounding box consists of four coordinates specifying its location: two for the upper left corner and two for the bottom right corner of the bounding box. The images are labeled by humans and is called the ground truth of the image.

2.1.2 One-shot detectors

One-shot detectors were first introduced by the model YOLO [**?**] and later modified and improved by the models SSD [**?**], YOLO9000 [**?**], DSSD [**?**] and RetinaNet [**?**]. A one-shot detector works by predicting up to tens of thousands of bounding boxes in different spatial positions in an image, that sets out to detect every object in that image. For every set of predetermined spatial positions in the image, the model predicts 4 bounding box coordinates and probability scores that there exists an object inside the specified bounding box.
The model is trained by assigning every bounding box either a negative or a positive label during training, depending on if the bounding box contains a ground truth object or not. A bounding box is said to contain an object if the intersection over union (IoU) [**?**] of the area of the bounding box and the area of the ground truth is bigger or equal to a threshold value (0.55). The intersection over union is used to determine how similar two sets A and B are (see figure 3). It is bounded by the interval $[0, 1]$ and is defined by the following equation:

$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{7}$$

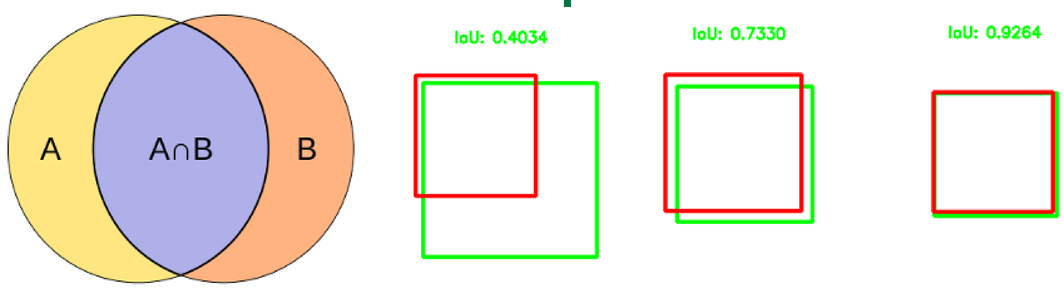

Figure 3: IoU [**?**] is defined as the size of the union divided by the size of the intersection of two sets A and B. A bigger IoU implies that the predicted bounding box is closer to the ground truth.

2.2 Method

2.2.1 FaceNet Baseline Model

The baseline model uses the deep residual convolutional neural network ResNet-18 [**?**] as a backbone network and feature extractor. It uses a total of 18 convolutional layers throughout blocks named *conv1* to *conv5* to extract features from the input image. An additional *conv6* block is created after the last layer to create one additional scale of feature maps. The block consists of one convolutional layer with stride 2, and 2 residual bottleneck layers used by Resnet-50 up to ResNet-152. The feature maps created by the backbone model is then fed into two smaller sub-networks, called the classification head and the regression head. They are constructed out of 4 residual bottleneck layers.
Bounding boxes are created from a number of anchor boxes at every spatial position (width and height) from a set of feature maps. An anchor box is a bounding box of a specific size and scale, used to base the bounding box predictions off. Anchor boxes used by FaceNet are set to be of sizes in the range $[16, 406]$ pixels, each increasing by a factor of $2^{1/3}$ for every new anchor box size. Two ratios of width:height are used at every scale, namely $1 : \frac{2}{3}$ and $1 : 1$, to enable the detection of faces facing sideways and directly into the camera. Having densely packed anchors makes it that every ground truth bounding box (of sufficient size) has an IoU> 0.55 with at least one anchor box. The anchor boxes are spaced out to include 3 scales with 2 aspect ratios across feature maps from the levels *conv2* to *conv5*, to enable the detection of faces of different scales, similar to RetinaNet [**?**].
The regression and classification sub-networks take in the last convolutional output from a single network level (*conv2* to *conv5*) as input. Let W and H be the width and height of the tensor of activations at a single network level, and C be the amount of classes to classify. FaceNet uses $C = 2$, one for a background class (no face) and one for a foreground class (a face). The classification head predicts C probabilities for every anchor A, that there exists a face in the anchor box at every spatial location ($W \times H$), for a total of $KWHA$ values. The regression head predicts 4 coordinate offsets for every point of the anchor box, for it to match the ground truth as closely as possible, for a total of $4WHA$ coordinate offsets. Additionally, similar to RetinaNet [**?**], the last conventional layer which predicts class probabilities uses a bias $b = -\log \frac{C(1-\pi)}{\pi}$, such that the model, when first initialized, predicts a probability of π for the case of there existing a foreground class (a face) in every bounding box. The value $\pi = 0.001$ was used.

2.2.2 Training

If not otherwise stated, each version of FaceNet was trained for 70 000 iterations on an Nvidia GTX 1080ti with an initial learning rate of 0.0001, and lowering the learning rate by a factor of 10 after 50 000 and 60 000 iterations. Due to memory limitations a batch size of 8 is used. The models are trained on square image crops, randomly picked to be of between 0.33 and 0.95 of the short size of the image, and resized to be of size 512^2 pixels. This is done to artificially increase the size of the training data. Additionally, the image is flipped horizontally with a probability of 0.5.
The models are evaluated on the WIDERFace validation set, using the easy, medium and hard splits provided by the WIDERFace evaluation server. The evaluation metric used is mean average precision (mAP). It is calculated by plotting the accuracy of the model as a function of its recall. The mean average precision is given by the integral of the accuracy with respect to the recall. The models will be evaluted on the easy, medium and hard validation splits.

Improving TCNs for Image Captioning

3 Improving TCNs for Image Captioning

Background

In conventional convolutional neural networks, the model learns to transform a given input by learning to apply a number of two-dimensional discrete convolutions on the input data. Instead of convolving along the spatial dimensions, a convolutional neural network can be used to convolve along the time dimension, and can thus be used as a sequence model for time series data, as long as there is no information leakage from the future into the past. Recent work on Temporal Convolutional Networks (TCN) [**?**] shows that the convolutional model outperforms its recurrent neural network (RNN) variants, including Long Short Term Memory (LSTM), and Gated Recurrent Units (GRU), on a variety of tasks.

I suggest the following two improvements regarding zero-padding and dilation: