International Research Journal of Computer Science (IRJCS)
Issue 09, Volume 07 (September 2020)
ISSN: 2393-9842
https://www.irjcs.com/archives

AN ANALYSIS OF CONVOLUTIONAL NEURAL NETWORKS AND THEIR APPLICATIONS

Advait Pravin Savant
Sardar Patel Institute of Technology, Mumbai,India
adisav17@gmail.com
Prof. Varsha Hole
Sardar Patel Institute of Technology, Mumbai,India
varsha_hole@spit.ac.in

Publication History
Manuscript Reference No: IRJCS/RS/Vol.07/Issue09/SPCS10081
Received: 26, August 2020
Accepted: 05, September 2020
Published: 08, September 2020
DOI: https://doi.org/10.26562/irjcs.2020.v0709.002
Citation: Advait Savant, Hole(2020). AnAnalysis of ConvolutionalNeuralNetworks and Their Applications. IRJCS:: International Research Journal of Computer Science, Volume VII, 227-232.
https://doi.org/10.26562/irjcs.2020.v0709.002
Peer-review: Double-blind Peer-reviewed
Editor: Dr.A.Arul Lawrence Selvakumar, Chief Editor, IRJCS, AM Publications, India

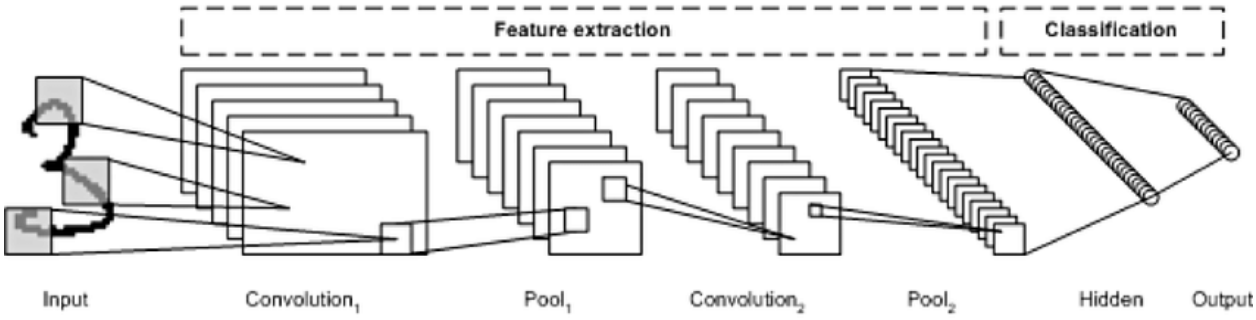
Abstract: The field of machine learning concerns itself with creating models to represent phenomena in the real world based on the data that is obtained. We model the probability distributions over input and output variables in order to estimate functions which capture the relationship between the input and the output. For a statistical parameterized model, we traverse through the parameter space in order to obtain a parameterized hypothesis which best fits the data based on some optimization criterion used in the context of our model. With the increase in computing power and the availability of a large amount of data, Machine Learning has been successfully used in a wide variety of tasks. Deep learning is that subfield of machine learning which uses artificial neural networks related models to create nested representations for the data. An important part of deep learning, convolutional neural networks are computational models similar to standard ANNs but designed for data with a spatial or a grid like topology. They contain the convolution operation which is inspired from digital signal processing. Computer Vision is the field of AI which concerns itself with the task of obtaining a high level understanding from image and video data. Convolutional neural networks have been successfully used in computer vision and they have set the benchmark surpassing traditional computer vision techniques. From image classification to object detection and face recognition, CNNs have proven to be very effective. In this paper, we will understand how CNNs work and we will understand the nature of information representation in CNNs. We also look at various computer vision applications of CNNs.

Keywords: Convolutional neural networks, backpropagation, class activation maps, representation learning, object localization

I. CNN ARCHITECTURE

A convolutional neural network makes use of the inherent structure of the data with a grid like topology. Image data consists of 3 channels (Red, Green and Blue) each of which have a matrix of pixels which represent the intensity of light for that color at that particular position. The input image data has a height H, width W and C channels for the depth where C=3. We have a H*W*C tensor. This tensor is fed to the first convolutional layer. The convolutional layer consists of a set of filters, also called kernels. Let us say there are K such filters. Each filter is a tensor of weights having dimensions F*F*D for the height, width and depth. The value of F is less than the values of H and W. The depth of filters in each layer is the same as the depth of the input coming in from the previous layer. In the first convolutional layer, D will be the same as the number of channels in the input image which is 3. Using each filter, we perform the convolutional operation on the input data with that filter in order to obtain a feature map. This operation is performed across the spatial extent of the image data. The convolution operation is as follows, let us say we start with the top left, we superimpose the filter on the top left of the image such that there is an element by element correspondence between the image pixels and the filter weights. We than multiply the corresponding elements between the filter and the slice of the image under consideration. We all the terms that we obtained from element wise multiplication.

This result goes into the top left position of the feature map produced by that filter. We are essentially taking the dot product in the convolution operation. We then move the filter a unit to the right and repeat the same operation for the next corresponding image slice to obtain the second term on the feature map which is placed to the right of the first term. We keep moving the filter to the right until the filter keeps fitting on the image slices and there is a one to one correspondence. Once we are done with convolution between a filter and the image slices which have the first row of the image data as the top of the slice, we have obtained the first row of the feature map. We then move the filter down left to the second row, start with the left most position in the width and again superimpose the filter with the image slice to establish an element to element correspondence and repeat the convolution operation to obtain the first element of the second row of the feature map. We then move the filter to the right and follow the same procedure to obtain a complete feature map from one filter. The feature map will have dimensions (H-F+1)*(W-F+1). We repeat this procedure for all K filters to obtain K feature maps. We stack these feature maps together to obtain a (H-F+1)*(W-F+1)*K volume which is the output of the convolutional layer. We add a bias term to each feature map and pass this volume through a RELU activation function to obtain non linearity in our mapping. After activation, pooling is performed. The pooling function replaces the output in a locality with a single summary statistic. Max Pooling in a rectangular neighborhood takes the max value among the neighborhood (only for that channel), places it in the corresponding position in the feature map and ignores the rest of the values in the neighborhood. Average pooling takes the average value in the neighborhood and passes it forward in place of all the values in the neighborhood. Pooling helps make the representation approximately invariant to small changes in the input and enables robust learning. This was one round of convolution, activation and pooling which has given us a new volume. We then take this volume and feed it as input to the next convolutional layer which has its own set of filters each of depth K (for the K feature maps from the K filters used in the first convolutional layer). We repeat the convolution, activation and pooling operations across layers in a CNN. As we go deeper into the layers the height and width of the volume decreases and the depth increases. Later on, we flatten the output from an intermediate layer to obtain a one dimensional layer of neurons. We can then add dense layers just as in a standard ANN. The final layer will represent the output. For example, if it is an multi class image classification task, we will use the softmax activation over the last layer outputs to get a probability distribution for various classes. Convolutional neural networks as suited for image data which has a spatial topology as we are able to learn multiple layers of meaningful filters in the training process which enable us to extract features from the image data across the layers. CNNs give us sparse interactions, parameter sharing and translation equivariance.[1] In a standard ANN, each unit in one layer is connected to each unit in the previous layer. In a CNN, each unit in a feature map for one filter is only connected to a small number of units which precede it and are spatially nearby in correspondence. Also, the same filter is used at different positions across the image, that is, the weights are shared. Filters and convolution have been used in computer vision for detecting features such as edges even before the dominance of deep learning.[5] Parameter sharing enables us to detect features wherever they are present in the image. A CNN also exhibits translation equivariance, if we move an object in the input image data, its representation will also move similarly in the layers of the CNN. Convolution is a linear transformation and it can be expressed as a matrix multiplication. Convolution and Pooling can be interpreted as infinitely strong priors which necessitate sharing of the same weights and demand that other weights be zero when we are talking about a particular output unit in a layer(only the locality is considered by the filter concerned). For each output unit in a layer, the preceding units which influence it are called its receptive field. As we go deeper in the network, we see that the receptive field of the unit increases. This enables the units in deeper layers to be indirectly connected to a large number of input image data units even if the connections are sparse. In practice, we zero pad the input volume to make it wider. Zero padding allows us to prevent the rapid shrinking of the spatial extent of the network and even enables us control the kernel width. This gives better expressive power to our network.[1]



II. CNN TRAINING

It is expected that the reader is aware of vectorized backpropagation with gradient descent as used in standard ANNs. The procedure for CNNs is similar. We move backwards in the network and we keep taking the derivative of the loss function with respect to the weights using the chain rule. We know that the gradient vector, which is the vector of the partial derivatives of a multivariate function, is the direction of steepest ascent for that function. To minimize the loss function, we thus take small steps opposite the gradient in each iteration and update our weights in an iterative fashion. In a CNN, for an input image, as we move backwards, initially we have the dense layers, we calculate the derivative of the loss function with respect to the predicted output, the pre activations of the output layer and then the weights between the final output layer and the penultimate layer. We then calculate the derivative of the loss function with respect to the post activations of the penultimate layer and we continue this backwards to find the derivative of the loss function with respect to the layer that was flattened. We know the correspondence between the one dimensional flattened layer and the final volume that was flattened. We thus have the derivatives of the loss function with respect to the units in the volume that was flattened. Now we will explain how to get the derivatives of the loss function with respect to the post activation volume, which precedes the final volume.

Consider a top left slice in the volume, which precedes the final volume. For each filter which exists in that last convolutional layer, we have taken the dot product of the filter with that slice in order to obtain the corresponding unit in the feature map of the final volume. We have seen earlier that we have the derivative of the loss function with respect to that unit. As we use the chain rule, we take that derivative and multiply it with all individual filter weights in order to obtain a tensor with the same dimensions as the filter. Notice that this is the derivative of the loss function with respect to that slice (when we take into consideration one filter and the corresponding unit in the final volume feature map). We now have to consider all other filters, which are convolved with that slice to produce the respective units in the final volume (we have the derivatives of the loss function with respect to these units). We then perform a similar multiplication of the derivative (a scalar) from the unit with each element in the corresponding filter (a tensor) to obtain a tensor just like we did before. We then add up all these tensors as we have produced to get the derivative of the loss function with respect to that top left slice as a tensor. Each element in the tensor gives us the derivative of the loss function with respect to the corresponding element in the slice. We can repeat this procedure for all slices in the penultimate volume. Now that we have the derivative for the post activations, the RELU activation makes it easy to find the derivative for the pre activations in the penultimate volume. Now we discuss the derivative of the loss function with respect to the filter weights for the last convolutional layer. Each filter is multiplied element wise with each penultimate volume slice to produce a unit in a feature map in the final volume. We have the derivatives for the units in the final volume. For each filter, as we traverse through the penultimate volume slice, we realize that as we use the chain rule, we just have to take the derivative for the unit in the feature map(scalar) and multiply it with the slice(tensor) that produced it in an element wise fashion and add these tensors for all slices. In this way, we obtain the derivative of the loss function with respect to the filter weights. We can extend this procedure backwards for all layers. We must now talk about pooling. Consider max pooling. Max pooling is done after convolution. As we move backwards, let us say we used the procedure as we explained above to find the derivatives for the volume after pooling. We must find the derivative of the volume before pooling. For each unit x in the volume after pooling, the unit which was chosen as the max unit in the corresponding feature map is given the same value for the derivative as x. For other units in that rectangular neighbourhood which weren't chosen during pooling, the derivative value is 0. Thus we have explained how to find the derivative of the loss function with respect to the weights and the activations in convolutional and pooling layers. We can then use stochastic gradient descent or any of its improvements like adam[6] to iteratively update the weights and find meaningful filters across the CNN.

III. INFORMATION REPRESENTATION

We now look at how successive convolutional layers transform the input and the nature of the filters learned. Let us say we are using a benchmark CNN like ResNet[4] for an image classification task. As the image passes through a CNN, it shrinks and the number of features increases. Consider a final volume to be of dimensions (7,7,2048). We now do max pooling to obtain a 2048 length vector. We then add a dense layer for the outputs and we do softmax over the outputs. Consider that the network is trained and we are correctly predicting a car wheel in the classification task. We enumerate the weights between the output neuron for a car wheel and the previous layer with 2048 units. We have 2048 weights, one weight between the output neuron for the car wheel and each unit in the previous layer. We also have 2048 (7,7) matrices on which we did max pooling in order to obtain the 2048 units in the penultimate layer. We take the dot product between these weights and the corresponding (7,7) matrices in order to obtain a 2 D image. Let us call this image x. If we scale this image to fit the size of the original car wheel image and represent it as a heat map with a warm to cool color scheme on the top of the car wheel image, we see higher magnitudes of intensity in image x in the region of the car wheel wherever the wheel is present in the original image.

The region where the wheel is present will be red in the heat map and surrounding regions will be green while the rest of the image will have a blue hue. This shows us the representative power of a CNN. Now we must understand the hierarchical nature of information representation in a CNN.[3] Once we have a trained CNN, we keep the weights fixed and train the input to maximize the activations of a feature map from one filter. We use gradient assent and iteratively update the input, we keep the cost to be the mean of the activations in the feature map from one filter and maximize the cost. This enables us to find the input which maximally activates each filter. What we observe is that for the initial convolutional layers, the input which maximally activates the filters consists of simple patterns like lines at an orientation, simple curves and simple dotted figures. As we move deeper in the CNN, the inputs which maximally activate the filters in deeper convolutional layers get more and more complex and contain patterns and figures which are progressively more intricate. This demonstrates a hierarchy of information representation designed to capture the nuances in image data.[1] Another important use of CNNs concerns transfer learning. We use the knowledge that we have gained in performing one task to better perform another related task. State of the art CNN models such as ImageNet which have been trained on a large amount of image data have captured useful patterns and abstractions across its layers. We can use the representations created here to better perform our task. Suppose we want to train a model to classify cats and dogs. We could take the ImageNet, cut off its last layer and use the penultimate layer activations which are generated for each of the cat and dog images in our training data as features in our new training set. We can then train a shallow neural net using these features as input and a sigmoidal unit for cat/dog classification as output. This gives us fast and efficient results.

IV. NEUROSCIENTIFIC PARALLELS

Neuroscientists talk about neuronal representations for external entities and concepts[2]. They consider a population of neurons and their activity to stand in for external entities. Light from the surroundings enters the eye, strikes the retina and images are formed. Here some preliminary processing is done and then the image passes through the optic nerve to the lateral geniculate nucleus. From there, the signal is taken to the primary visual cortex, an area of the brain known for doing significant image processing.[1] This primary visual cortex has many similarities with a convolutional neural network. It is arranged in a two dimensional spatial map like structure just like the 2 D feature maps in a CNN. It contains simple cells which work based on a small receptive field. It contains complex cells which are invariant to small shifts in the position of an entity in the image. This relates to the concept of pooling in CNNs. As we go deeper into regions of the brain, we find neurons which are specialized to respond to a particular concept and are invariant to a large number of transformations in the input with regards to that concept. These are called grandmother neurons. The logic is that a person has a neuron that fires whenever he sees his grandmother. This is regardless of how her face is oriented and where she is present in his visual field. These grandmother neurons are known to exist in the medial temporal lobe. This relates to our concept of a representational hierarchy in a CNN. The CNN can be thought of as a feature detector in an hierarchical fashion and this feature detection also has similarities with Gabor functions known to be present in the primary visual cortex.[1]

V. APPLICATIONS

We can use the CNN for object localization. If we take a CNN and use a final layer which contains a sigmoidal unit which tells us whether or not an object is present in the image, 2 units which tell us the coordinates of the height and width of the center of the object and 2 units which tell us the length and the breath of the bounding box around the object, we have converted object detection into a supervised learning task. The last 4 units will be trained using a squared error loss whereas we will use cross entropy for the sigmoidal unit. We can train a CNN using this approach for closely cropped images containing the object at various orientations and images not containing the object. We can then use this CNN as a sliding window across a new image which contains the concerned object in a limited region. This gives us a good method for object localization. However, the sliding window approach is computationally expensive. We realize that fully connected layers can be implemented as convolutional layers and we can do the computations for the whole image in one forward pass without the need of a sliding window. This serves us better computationally [7]. We can train such a CNN for recognizing faces in an image and producing a bounding box for faces. What we also do is differentiate between faces in order to not only recognize a face but also establish the identity of the face. This is done using Siamese networks. We use a CNN trained on facial data to generate a vector which acts as an encoding for the face. The training is done in such a way that the Euclidean norm of the difference between the encodings of two face images of the same person is minimized and the Euclidean norm of the difference between two encodings the faces of two different people is maximized. For this, we use the triplet loss.[8] Once our training is done we train a logistic model with a sigmoidal unit on the modulus of the difference between two encodings of two people for verification of the identity of a person. The sigmoidal unit helps us to do this binary classification for any two face images to recognize if they are of the same person or not. This can help us establish an identity of a person.

VI. CONCLUSION

In this paper we have seen how the CNN, a connectionist model for spatial data having neurobiological parallels is used successfully in a variety of tasks. We have been through its architecture and understood the concepts of filters, feature maps, convolution and pooling. We have been through the training procedure and have understood how backpropagation is used in a CNN. We saw the nature of information representation in a CNN and the statistical strength it displays during transfer learning. We saw how CNNs can be used for object detection and face recognition and we also saw some neuroscientific parallels with a CNN. CNNs have also been used in generative models for a broad range of exciting tasks.[1] CNNs are an exemplar of the power of deep learning and connectionist frameworks for artificial intelligence. CNNs have transformed computer vision and we will see the AI community progress in applications of deep learning such as these.

REFERENCES

1. Goodfellow, Bengio and Courville, Deep Learning, MIT press,2016.
2. Eliasmith and Anderson, Neural Engineering, MIT press,2003.
3. From Generic to Specific Deep Representations for Visual Recognition. Azizpour, Hossein & Razavian, Ali & Sullivan, Josephine & Maki, Atsuto & Carlsson, Stefan, Royal Institute of Technology, Stockholm 2014.
4. Deep Residual Learning for Image Recognition. Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun. Mircosoft research 2015.
5. N. Kanopoulos, N. Vasanthavada and R. L. Baker, "Design of an image edge detection filter using the Sobel operator," in IEEE Journal of Solid-State Circuits, vol. 23, no. 2, pp. 358-367, April 1988, doi: 10.1109/4.996.
6. Adam: A Method for Stochastic Optimization. Diederik P. Kingma, Jimmy Ba. ICLR 2015.
7. SSD: Single Shot MultiBox Detector. Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. ArXiv Computer Vision and Pattern Recognition 2015.
8. Bukovcikova, Zuzana & Sopiak, Dominik & Oravec, Milos & Pavlovicova, Jarmila. (2017). Face verification using convolutional neural networks with Siamese architecture. 205-208. 10.23919/ELMAR.2017.8124469.

