An Overview of Convolutional Neural Networks with application to the MNIST dataset

Arthur Deng*, Jiankang Geng†, Sarthak Jain‡ and Theo Tollet§

*35997444
†35996412
‡36065118
§35916709

***Abstract*—Convolutional Neural Networks (CNNs) are a class of neural network architectures characterised by their use of a particular class of linear operation, known as convolution, to combine input feature values. Finding particular success in the field of image processing and computer vision, CNNs have become widely adopted within industry and academia alike as a result. This report presents an exploration of the underlying principles justifying this architecture, demonstrating - via this group's own implementation - its ability to identify various handwritten digits from the widely known "MNIST" dataset in the process. This is followed by a comparison of the results against those of various related state of the art models to conclude.**

I. Introduction

The development of convolutional neural networks - hereafter referred to as "CNNs" - was motivated in part by the study of neuroscience's understanding of mammalian visual cognition [1], perhaps motivated in turn by the definition of artificial intelligence as "that which thinks like a human" [2]. This study gave rise to neural network architectures such as that of the "neocognitron" [3] which - following the introduction of back-propagation [4] - led to the formation of modern conventions in CNN architecture today.

Finding initial industry adoption reading handwriting toward the start of the 2000s [5, 6], CNNs - and deep learning more generally - yielded significant interest from the private sector following major success in the ImageNet object recognition challenge of 2012 [7], clearly demonstrating the capacity for CNN applications beyond conventional character recognition. Leveraging key ideas in the improvement of machine learning systems, CNNs are notably some of the first deep learning models - neural networks with more than two non-output layers (a.k.a "hidden layers") [8] - to have achieved good performance so as to be considered feasible for widespread commercial usage, promoting the consideration and research of other deep learning neural network architectures in the process [1].

II. Methods

The key difference between CNNs and DNNs is that CNNs make use of convolutions. In the case of DNNs, each unit in the first layer is connected with each and every pixel of the image. All these connections may not be needed as the image pixels are mostly correlated to each other. Using convolutions, the CNNs take the pixel correlations into account. CNNs make use of kernels which convolve across the input images generating a feature map. These feature maps actually learn the underlying patterns in the input images. The more convolution layers are used in the model, the more complex underlying patterns are learned by the model, with every subsequent convolution layer. Thus, every unit in the first layer of a CNN has as many connections as the size of the kernel. The size of the feature map generated also depends on the size of the kernel. It also depends on how many pixels the kernel window skips when convolving with the image. This is also called the strides taken by the kernel. Size of the output can also be controlled by dilating the kernel which changes the spacing between the pixels affected by the kernel.

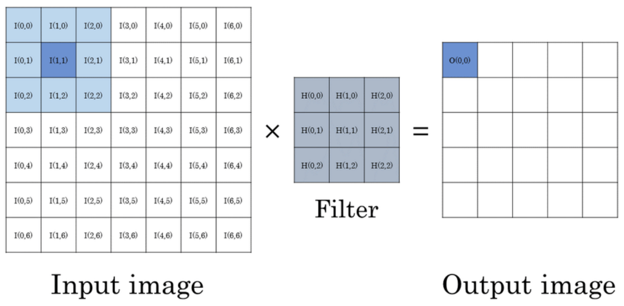


Fig. 1. An example of convolution with Images (source: Researchgate)

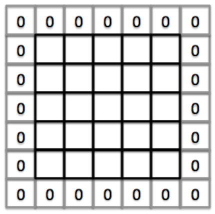


Fig. 2. An example of padding (source: Medium)

If the feature map generated by the covolution is not of the same size as the input, we may lose pixels off the edge of the image in each subsequent convolution layer. This can be avoided by adding a zero padding to the input layer of the CNN. Padding is helpful when the model has multiple layers. The size of the feature map generated can be calculated using eq.(1)

$$Output = \frac{Input - Kernel + 2Padding}{Strides} \tag{1}$$

It may appear that CNNs require fewer parameters to be trained when compared to DNNs, but it can be the other way round as more convolution layers are added to the model. Unlike DNNs, where most of the parameters are concentrated at the beginning of the model, a CNN has fewer parameters associated with the initial few layers of the model while most of the parameters are concentrated at the last layers. This shows that while CNNs have more expressive power and require fewer parameters, reading these more expressive representations requires more parameters at the output side.

Training a model with a lot of parameters can mean a lot of computation overhead. Thus, CNNs make use of pooling operations along with Convolution layers. Predominantly two type of pooling operations are used namely, max pooling and average pooling. This means that the pooling will take the maximum or the average over a window of defined from the input for each location in the output. Pooling can reduce the number of parameters dramatically depending on the size of the pooling window, but there can be small trade-off involved with the model performance.

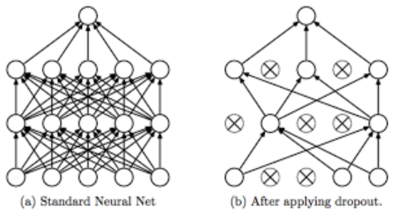


Fig. 3. Dropouts (source: Srivastava, Nitish, et al., JMLR 2014 [9])

Like other models, CNNs can overfit as well. However, this issue can be mitigated using two regularization techniques namely, dropout [9] and batch normalization [10]. When using dropouts, at each learning step, a proportion of nodes are simply ignored in forward as well as backward propagation. As the nodes are selected randomly, different networks inside the model are trained on different parts of data. Thus, if a part of the model becomes too sensitive to noise in the data, other parts of the model can compensate because they have not seen the noise in the data. This also prevents different units in the model from being overly correlated, ensuring that each neuron learns different information. In batch normalization, the output of a particular layer of the CNN is standardized using eq.(2)

$$x‘ = \frac{x - \mu}{\sigma} \tag{2}$$

Batch normalization helps if different batches of input result in very different distributions of output in any given layer of the model.

It is however observed that batch normalization and dropout don't always work very well together and might affect model performance if used together [11]. This is because both affect the learning rate of model differently thus, neutralizing the overall affect. In our model, the input data is normalized as a part of preprocessing and dropouts are introduced in the model.

To make the classifications, before the last layer the output of the model is flattened so that all the units can be connected to the final layer making the last layer fully-connected. The number of units in this layer is kept same as the number of classes. For classification problems, softmax activation function is used at the final layer as it supports multi-class classification.

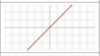
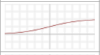
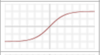
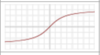
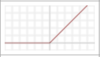
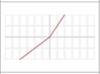
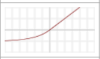
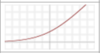
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Fig. 4. Activation Funtions (source: towardsdatascience)

Activation functions are link functions which allow neural networks to take non-linearities into account. Various linear like identity, relu, binary step, leaky-relu, etc. and non-linear activation functions like sigmoid, tanh, arctan exist. Usually, relu is the preferred choice of activation function for training the model because it has two linear pieces and it can account for non-linearity when multiple hidden layers exist in the model.

To help the model make better predictions and adjust the parameters of the model, CNN's also utilize loss functions, which aggregate the errors made in predictions from multiple data points/or images into a single number. For classification problems, categorical crossentropy is a preferred loss function because it works very well with softmax activation function and it can differentiate as well as quantify different probability distributions thus, making it suitable for multi-class classification problems. Mathematical equation for categorical crossentropy is given in eq.(3) where y_i is the i^{th} target value and $\hat{y}_i$ is the corresponding model output value.

$$Loss = -\sum_{i=1}^{outputsize} y_i \log \hat{y}_i \tag{3}$$

TABLE I
CNN Model Hyper-parameters

Hyper-parameter	Value
Neurons per layer	32
Kernel size	3 X 3
Pooling window size	2 X 2
dropout size	0.5
Image size	28 X 28
train data size	60000
test data size	40000
validation split	0.2
maximum epochs	30
total parameters for CNN	55658
total parameters for DNN	42310

TABLE II
Model Evaluation Metrics By Activation Function

Model	Train Accuracy	Train Loss	Valid. Accuracy	Valid. Loss	Test Accuracy	Test Loss	Approx. Time(sec)
relu	0.9902	0.0309	0.99	0.0351	0.9907	0.0291	49
sigmoid	0.9847	0.0479	0.9854	0.0494	0.9872	0.0473	51
tanh	0.9824	0.0552	0.9845	0.0602	0.9857	0.05	48
DNN	0.9813	0.0609	0.969	0.1045	0.9664	0.1639	119
resnet	0.9345	0.2681	0.814	0.6924	0.8569	0.5412	783

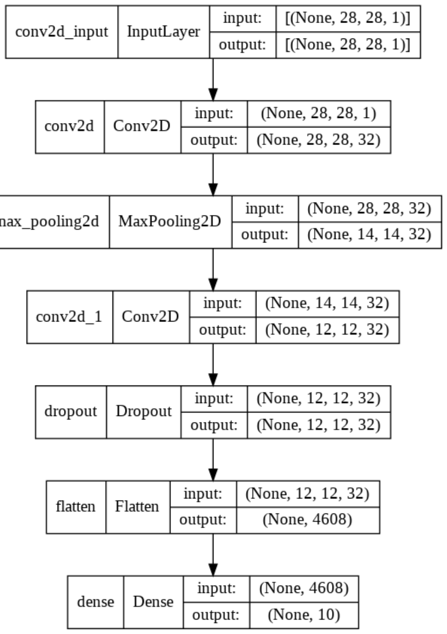
Sparse categorical cross entropy is utilized in this implementation as it works exactly like categorical cross entropy but it saves the trouble of one-hot encoding the categorical data as well as the target labels.

Since, loss function is used, in order to make the model learn the weights so that the loss incurred is the least, optimizer algorithms are used to update these weights time and again. Gradient descent is one of the most popular optimizer algorithms, which updates the weights in accordance with the with the slope of the loss function. In order to keep a check on the updates in the weights, learning rate is used. Adam is another optimizer algorithm which makes use of the gradient descent but it adjusts the learning rate dynamically hence, it is used in the implementation.

III. Experiment/Results

The working of the CNN is showcased by implementing a model, using the Keras wrapper of the tensorflow library, for a classification task. Permutation of different activation functions, namely, relu, sigmoid and tanh are tested to find the best possible model, whose performance is then compared to that of a Deep Neural Network and resnet50 [12], which is a state-of-the-art model used for image processing.

The images of MNIST dataset are colored images with three channels corresponding to the RGB color model. These images are converted into single-channel grayscale images. This is because colours are not that significant in differentiating the numbers from each other and hence, can be ignored. Further, the pixel values are normalized to scale of 0 to 1, to avoid using batchnormalization with dropouts in the model. 40% of the total data is used to testing. 20% of training data is used for validation in order to avoid overfitting. Adam is used as the optimzer for the sparse categorical cross entropy loss function.



conv2d_input | InputLayer | input: | [(None, 28, 28, 1)]
output: | [(None, 28, 28, 1)]
conv2d | Conv2D | input: | (None, 28, 28, 1)
output: | (None, 28, 28, 32)
max_pooling2d | MaxPooling2D | input: | (None, 28, 28, 32)
output: | (None, 14, 14, 32)
conv2d_1 | Conv2D | input: | (None, 14, 14, 32)
output: | (None, 12, 12, 32)
dropout | Dropout | input: | (None, 12, 12, 32)
output: | (None, 12, 12, 32)
flatten | Flatten | input: | (None, 12, 12, 32)
output: | (None, 4608)
dense | Dense | input: | (None, 4608)
output: | (None, 10)

Fig. 5. CNN Architecture

The model architecture is described in fig.5. The dropout layer next to second convolution layer randomly chooses a specified number of the units in the layer to be ignored. Permutations of relu, sigmoid and tanh activation functions (fig.4) across the layers of the CNN are trained and the best model so found is compared against the DNN and resnet.

Each model is trained for a maximum of 30 epochs. A callback technique is applied which saves the weights of the model when the validation loss is minimum. This model is then used against test data for further evaluation. The hyper-parameters used in the CNNs are described in table I while the performance of all the models in training validation and testing phases is illustrated in table II. The CNN architecture and the training and validation progression of all the models through each epoch is described in figures 6 to 11, respectively.

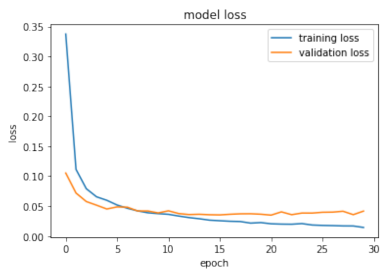


Fig. 6. Loss v/s epoch for CNN with relu activation function

IV. Discussion

Among resnet, DNN and CNN, the CNN by far gives the best result. The time taken to train the CNN despite learning

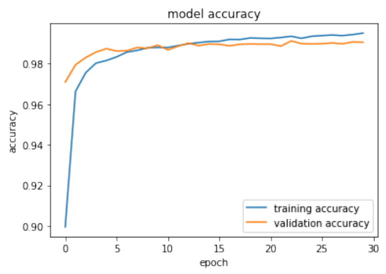


Fig. 7. Accuracy v/s epoch for CNN with relu activation function

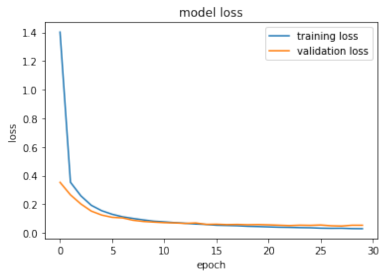


Fig. 8. Loss v/s epoch for CNN with sigmoid activation function

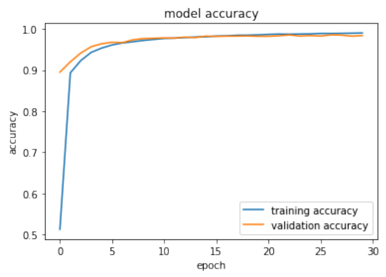


Fig. 9. Accuracy v/s epoch for CNN with sigmoid activation function

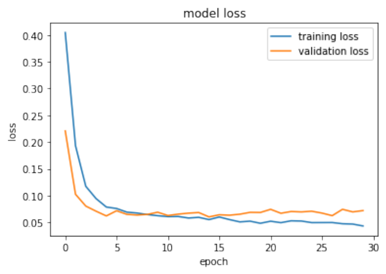


Fig. 10. Loss v/s epoch for CNN with tanh activation function

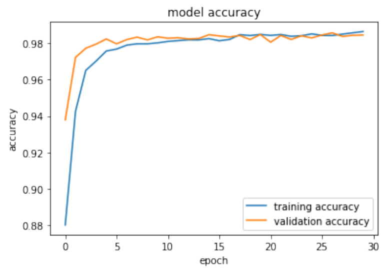


Fig. 11. Accuracy v/s epoch for CNN with tanh activation function

more paramters is lesser than the DNN and resnet. The values for accuracy and loss function are also found to be significantly better. This, indicates that the CNN has more expressive power than the simpler DNN and the architecture thus implemented is better for this use case than the resnet which is a state of the art model.

Interestingly however, the number of epochs required to generate the best result for validation loss is significantly less for the DNN considered. This indicates that for smaller data sets where classification problems do not require learning complex patterns and where time is a major factor, a DNN could be preferred over a CNN.

Across all the activation functions tested, CNNs are observed to perform extremely well at handwritten character classification - the best of these being the relu activation function, demonstrated through its overall performance across each metric considered. As the depth of the corresponding model is not too large, the relu activation function does not yield a model exhibiting the dying neuron problem [13] - where a relu unit becomes inactive, only outputting zero regardless of its input. If the model had been composed of more layers then this could have notably affected the model's performance.

A relatively smoother training/validation graph is generated for the sigmoid activation function, among others. This could be because it forces the neurons to take values between 0 and 1. Thus, unless there is significant change in weights of the model during backpropagation, the values taken by the neurons don't change significantly, causing the predictions to be consistent every epoch. This in turn causes smaller changes in the loss function resulting in the smoother curves. If the predictions are correct the loss function curve is closer to 0, else it is closer to 1 but it remains smooth. This could also be why it takes a longer time to train the model using the sigmoid activation function when compared to the others. In contrast, the tanh activation function trains the model in the least amount of time.

Furthermore, it is found that adding a pooling layer to the CNN decreases the training time significantly, whilst also reducing the number of trained parameters as a result.

In conclusion, CNNs are one of the most powerful tools for image processing with great expressive power. The working of CNNs and their comparison with DNNs and state of the art models is extensively discussed. The implemented CNN is further fine tuned by considering relu, sigmoid and tanh activation functions in turn.

Further CNN performance analysis could be performed by considering various permutations of kernel size, pool window size and the proportion of dropouts per layer. Other optimization algorithms like stochastic gradient descent could also have been detailed, and the performance of the model could have been analyzed over different learning rates to ensure the model generalizes well. Performance of the CNN across different data sets could also have been performed.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[2] S. Russell and P. Norvig, “Artificial intelligence: a modern approach,” 2002.

[3] K. Fukushima, S. Miyake, and T. Ito, “Neocognitron: A neural network model for a mechanism of visual pattern recognition,” *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 826–834, 1983.

[4] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] P. Y. Simard, D. Steinkraus, J. C. Platt *et al.*, “Best practices for convolutional neural networks applied to visual document analysis.” in *Icdar*, vol. 3, no. 2003, 2003.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[8] A. Burkov, *The hundred-page machine learning book*. Andriy Burkov Canada, 2019, vol. 1.

[9] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[10] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[11] X. Li, S. Chen, X. Hu, and J. Yang, “Understanding the disharmony between dropout and batch normalization by variance shift,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2682–2690.

[12] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[13] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, “Dying relu and initialization: Theory and numerical examples,” *arXiv preprint arXiv:1903.06733*, 2019.