Convolutional vs Fully connected:

How and when should we use them?

Summer Xia

(1) Abstract.

This project experiments with different DNN and CNN architectures and compares their performances. Through the process it reaches a deeper understanding of the functions of hidden layers and regularization methods in neural networks.

(2) Introduction.

Neural networks are critical for businesses due to their ability to process large volumes of complex data, identify patterns, and make predictions or decisions. Computer vision is one of the most popular AI applications as it enables automation for various visual tasks, and CNN is one of the major neural network architectures used for computer vision.

A Convolutional Neural Network (CNN) is a type of deep learning algorithm specifically designed for processing structured grid data such as images. It uses convolutional layers with learnable filters to automatically detect and extract hierarchical features, followed by pooling layers to reduce spatial dimensions. CNNs are widely used in computer vision tasks, such as image classification, object detection, and image segmentation.

How is a CNN different from a fully connected network? CNNs differ from fully connected neural networks in how they connect neurons and process data. CNNs use convolutional layers with localized connections and parameter sharing, which makes them more efficient in terms of the number of parameters and reduces the risk of overfitting. This design also allows CNNs to maintain spatial hierarchies in data, such as images, making them ideal for computer vision tasks. Fully connected networks, on the other hand, connect every neuron to every other neuron, treating inputs as flat vectors, which is less efficient for spatial data.

The goal of this project is to compare CNN and DNN architectures with real implementation. By understanding the strengths and weaknesses of various neural network architectures, businesses can select the most suitable model for their specific needs, ensuring optimal performance, efficiency, and business outcomes.

(3) Literature review.

Since DNN and CNN are both very popular neural network architectures, there are many researches and articles that compare their effectiveness in different settings. For example, “Fully Connected vs Convolutional Neural Networks” trained both CNN and DNN with MNIST data and compared their performance. The article concluded that although fully connected networks make no assumptions about the input they tend to perform less and aren’t good for feature extraction. Plus they have a higher number of weights to train that results in high training time while on the other hand CNNs are trained to identify and extract the best features from the images for the problem at hand with relatively fewer parameters to train.

(4) Methods.

The first four experiments consist of two DNNs and two CNNs of different numbers of layers, aiming to compare between CNN and DNN, as well as between single layer and multi-hidden layer networks. To compare the impact of layers, the experiments control the number of nodes, activation function, optimization method, number of epochs, batch size, and loss function. All models use Adam as the optimization method and ReLu as the activation function, have 20 epochs and a batch size of 64, and use sparse categorical cross entropy as its loss function because the labels are integers. The first layer of both DNN models has 256 nodes,

and the second layer in the two layered DNN has 128 nodes. However, for CNNs, the first convolutional layer has 128 nodes and the number doubles for the next convolutional layer, and the fully connected layer has 128 nodes. This design is because early layers in a CNN are responsible for capturing basic features like edges, textures, and simple shapes from the input data. As the data progresses through the network, the features become more abstract, representing higher-level concepts or patterns. Because the later layers are dealing with more abstracted and condensed information, they require fewer nodes to capture the necessary features.

A Convolutional Neural Network (CNN) typically begins with an input layer that accepts structured data, like images, followed by convolutional layers that apply filters to extract features such as edges and textures. After convolution, activation functions like ReLU introduce non-linearity, while pooling layers, such as max-pooling, reduce spatial dimensions to control complexity. Regularization techniques, like dropout, are sometimes applied to prevent overfitting. The final stage involves fully connected layers, where the feature maps are flattened into a one-dimensional vector, culminating in an output layer that provides the model's prediction, often using softmax for classification tasks.

The next part of the experiment uses the same architectures as the first four models but adds regularization after every hidden layer. L1 and L2 regularization, along with dropout, are key techniques used to prevent overfitting in neural networks. L1 regularization adds a penalty proportional to the absolute value of the weights, promoting sparsity by encouraging some weights to be zero. L2 regularization imposes a penalty proportional to the square of the weights, leading to smoother weight distributions. Dropout, on the other hand, randomly deactivates a proportion of neurons during training, forcing the network to learn redundant

representations and improving its ability to generalize. Together, these techniques help balance model complexity and robustness, enhancing neural network performance on unseen data.

To compare the performance, all the parameters of regularizers are the same. For L1 and L2, I chose 0.0001 which is much smaller than the default value 0.01 to prevent underfitting for the DNN models. All the dropout parameters are set to be 0.3, meaning that 30% of the neurons in the layer are randomly deactivated.

All the experiments were intended to be run on Google Colab TPU for optimal speed and efficiency. However, after the first 10 experiments, I reached the limit for free GPU, so I had to selectively choose the CNN model with two convolutional layers and dropout layers to run on CPU.

The models train on CIFAR-10 dataset which consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. After importing the data, I further split the training data into 10% validation set and 90% training set. Then we normalize the data by dividing them by 255 since the images are 28x28 NumPy arrays, with pixel values ranging from 0 to 255.

(5) Results.

The results of the first four experiments show that, without regularizations, CNNs outperforms DNNs in terms of accuracy. However, the CNN models seem to have a problem of

overfitting as the training accuracies are much higher than the validation accuracies. The output of the max pooling layers also exhibits the mechanism of CNN: the output of the second layer is more abstract and generalized than the first. Moreover, the results of DNNs show that despite the increased training time, the extra layer does not seem to improve the accuracy of the model significantly.

The results of DNNs with regularizations tell a very interesting story since their accuracies are overall worse than the ones without regularizations. This is very likely because DNN models did not exhibit serious overfitting issues based on the previous experiments, thus the regularizations lead to underfitting models. The dropout regularizer causes a more significant drop in accuracy because it randomly disables neurons during training, which can eliminate critical information and causes the network to miss key features or patterns.

On the other hand, CNNs benefit from regularizations as their accuracy witnesses some visible improvements. Nevertheless, the very small parameter I chose for the L1 and L2 is proven to be insufficient to solve the overfitting problem.

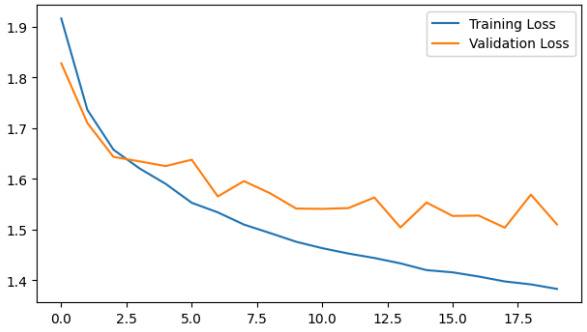
(6) Conclusions.

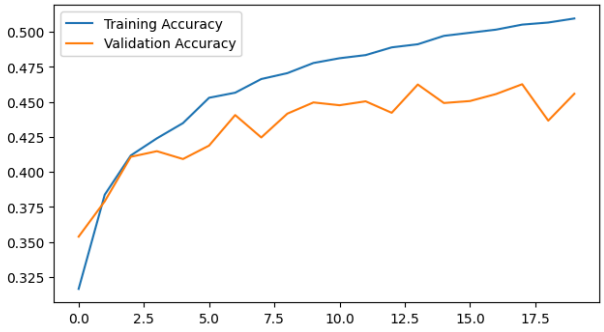
This research compares different neural network architectures and regularizations. The performance of DNNs and CNNs show that increasing the number of layers does not always lead to increased model performance, and the regularizations only work when our model has an overfitting problem. This proves the importance of experimenting with different neural networks and understanding their mechanisms for achieving the best results.

Appendix:

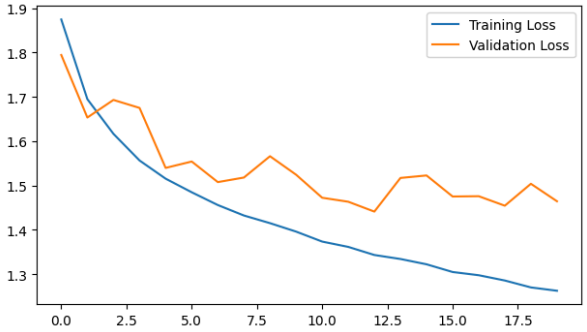
Loss and Accuracy plot Model 1 through 4:

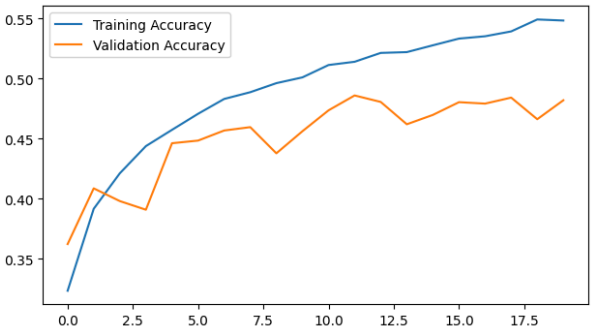
1.



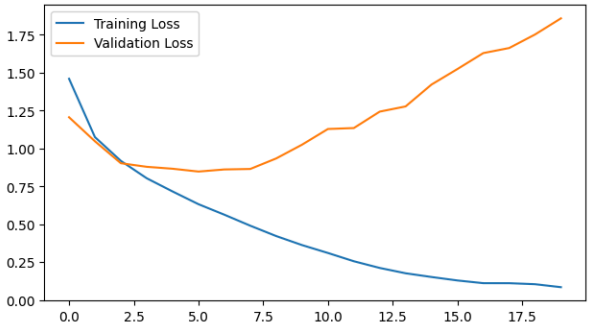


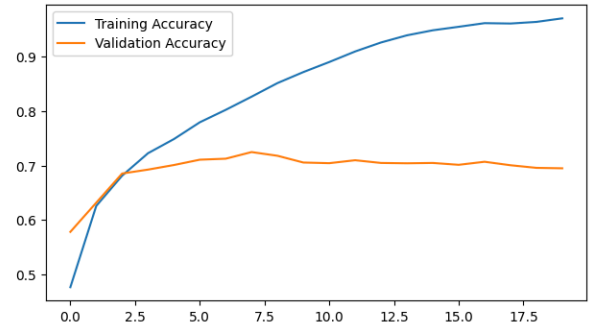
2.



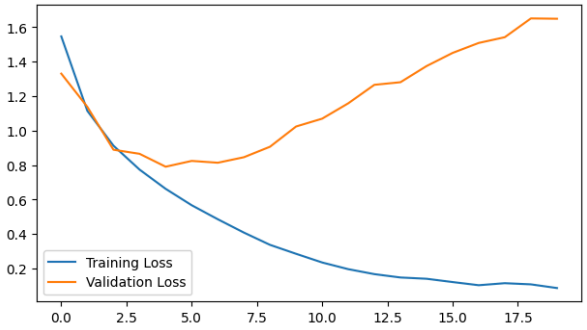


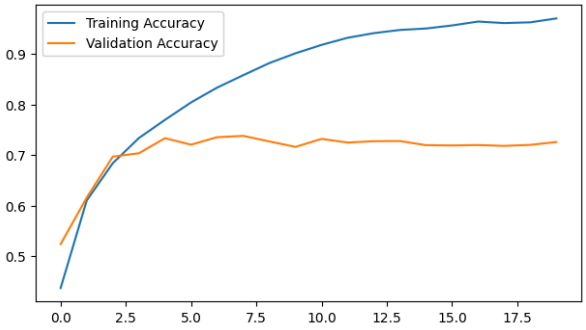
3.





4.

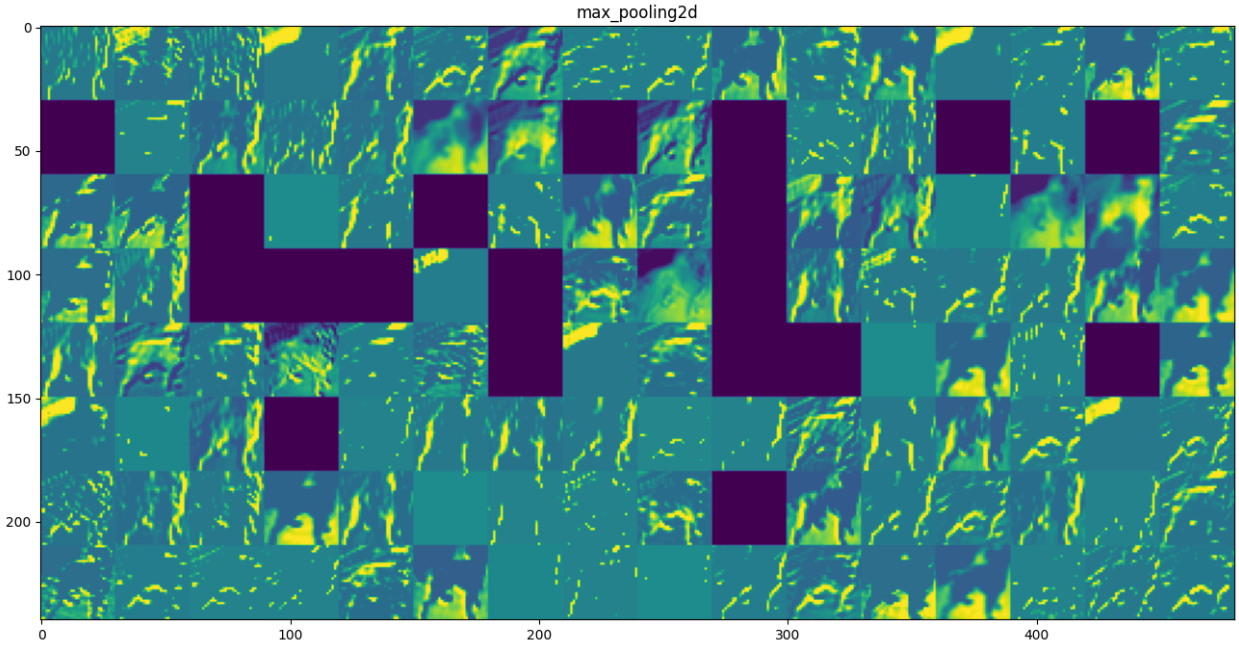




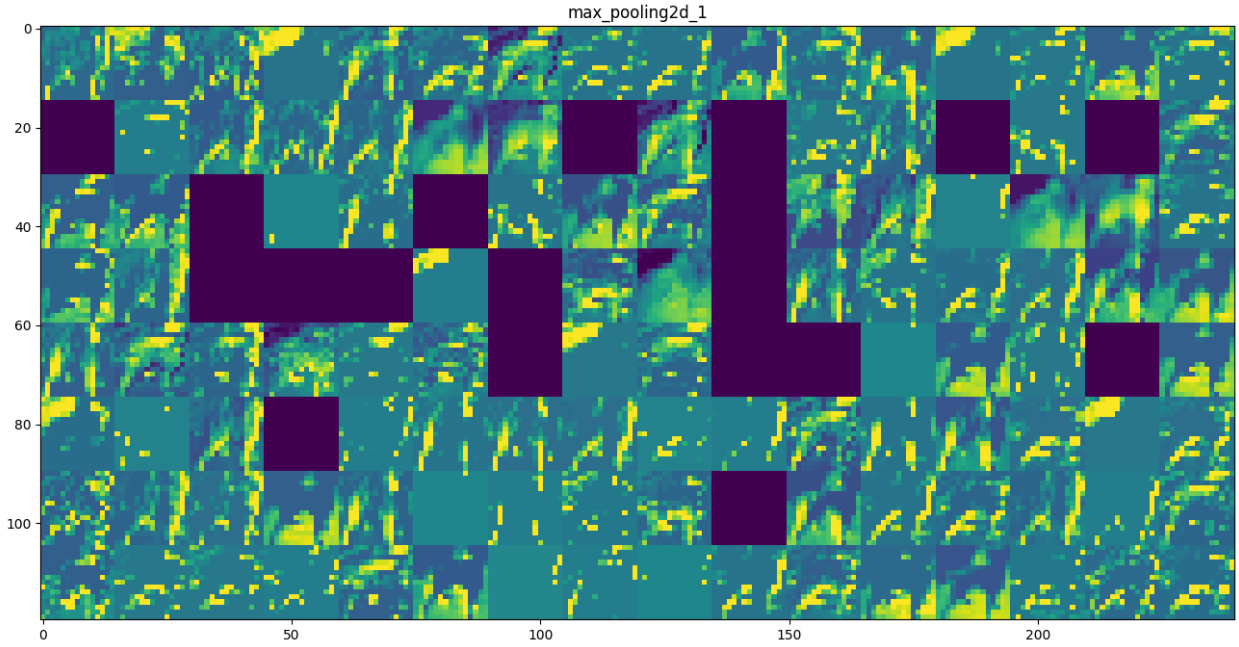
Max Pooling layers result:

Input:

1st layer:



2nd layer:



Result Tables:

A	B	C	D	E	F	G	H
	training_accuracy	training_loss	val_accuracy	val_loss	testing_loss	testing_accuracy	training_time
experiment1	0.541155577	1.293520689	0.465999991	1.527248144	1.515076995	0.471100003	53.40663457
experiment2	0.545422196	1.280854821	0.460399985	1.514373302	1.480298877	0.482199997	84.20704794
experiment3	0.959666669	0.116188034	0.691200018	1.790645361	1.864528298	0.690199971	144.4478927
experiment4	0.969799995	0.085461318	0.71359998	1.826800823	1.879687548	0.709800005	204.7096825

A	B	C	D	E	F	G	H
	training_accuracy	training_loss	val_accuracy	val_loss	testing_loss	testing_accuracy	raining_time
experiment5	0.450288892	1.672268033	0.426400006	1.725060344	1.6974901	0.442699999	84.19591
experiment6	0.489444435	1.486009359	0.453799993	1.573916674	1.54140949	0.467299998	84.61823
experiment7	0.347288877	1.780998468	0.388200015	1.713435531	1.70337927	0.390799999	84.08618
experiment8	0.482533336	1.594371319	0.457399994	1.656319499	1.63587391	0.465799987	84.20187
experiment9	0.534844458	1.36698103	0.468800008	1.5441643	1.51392329	0.480300009	84.72123
experiment10	0.355644435	1.786620498	0.392800003	1.7294662	1.7122941	0.410899997	56.74066