ASSIGNMENT- 2 DEEP LEARNING- a1894603

CNN – Image Classification

**Abstract**

*In recent years, the surge of Artificial Neural Networks (ANNs) and their integration into deep learning has significantly transformed the landscape of machine learning, imbuing it with higher artificial intelligence capabilities. Deep learning, particularly through the employment of Convolutional Neural Networks (CNNs), has become pivotal in various domains such as surveillance, healthcare, sports, robotics, and more due to its wide-ranging applications. CNNS, a blend of ANNs and modern deep learning techniques, have facilitated remarkable progress in tasks like pattern recognition, speech and face recognition, text categorization, and handwritten digit identification. This paper aims to investigate how the accuracy of CNNs in classifying handwritten digits varies concerning the number of hidden layers and epochs used. To conduct this assessment, we carried out experiments using the Modified National Institute of Standards and Technology (MNIST) dataset as a performance benchmark for the CNN model.*

*Keywords- Handwritten digit recognition, Convolutional Neural Network (CNN), Deep Learning, MNIST dataset, Epochs.*

**Introduction**

Recognition involves identifying or distinguishing an object or individual based on prior learning or experiences. In the context of digit recognition, it refers to the process of identifying numeric characters in various documents. Handwritten digit recognition involves enabling machines to interpret manually written digits found in messages, bank cheques, papers, images, etc., across different platforms such as web-based handwriting recognition on tablets, reading vehicle number plates, processing bank cheques, and recognizing digits entered in forms.

Machine Learning offers methods to reduce human effort in recognizing handwritten digits. Deep Learning, a machine learning approach, teaches computers to learn through examples, significantly reducing human involvement in tasks such as perception, learning, and recognition across multiple domains. Deep Learning methods enable computers to conduct classification tasks using images or text from diverse documents, achieving accuracy levels surpassing human capabilities. These models utilize extensive datasets to recognize digits from various sources.

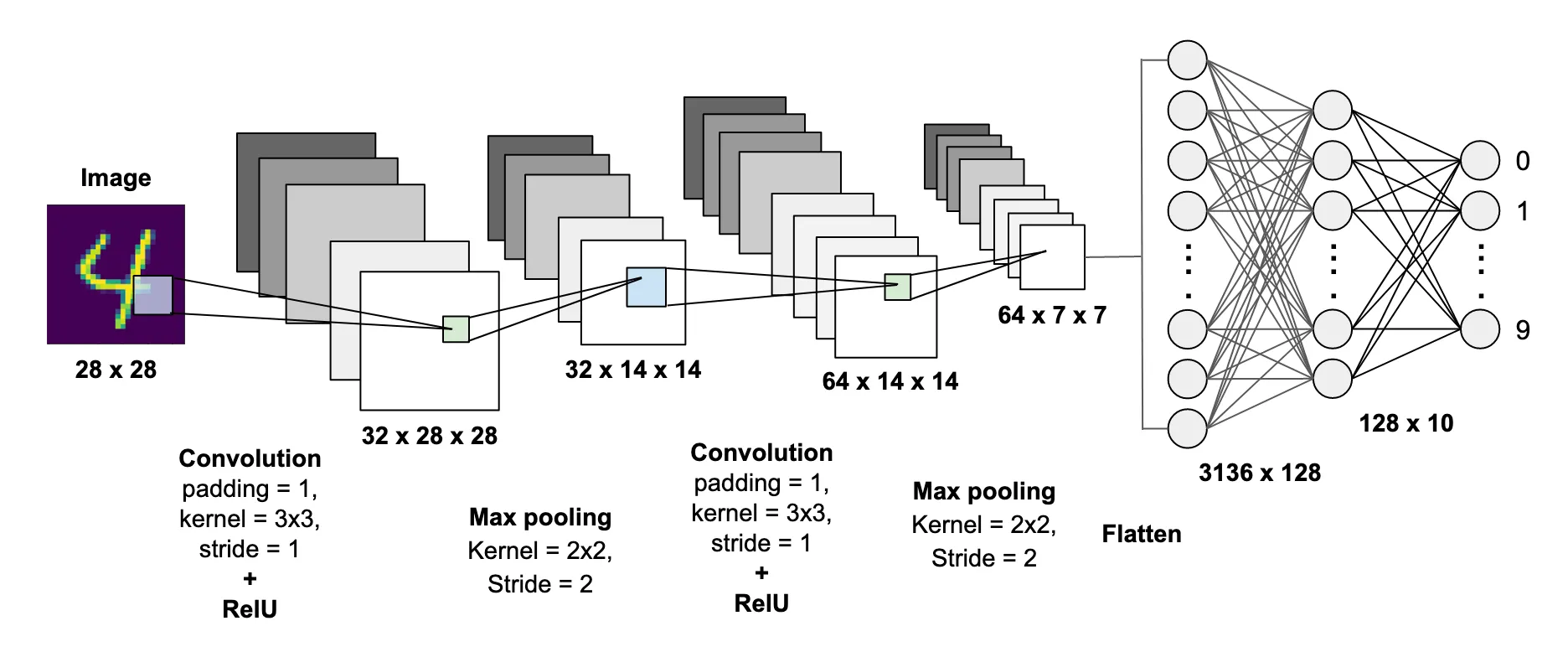
Handwriting recognition, dating back to the 1980s, holds substantial importance, particularly in tasks like online digit recognition on tablets, reading zip codes on mail, processing bank check amounts, and interpreting handwritten numeric inputs in forms like tax documents. Challenges arrives due to variations in size, thickness, orientation, and position of handwritten digits concerning the margins. The primary goal is to implement a pattern recognition method to identify handwritten digits within the MNIST dataset.

**CNN To Classify Handwritten Digits**

The CNN model is defined using the Sequential model from keras. It consists of several layers:

* Convolutional Layer with 32 filters and a ReLU activation function.
* Max Pooling Layer.
* Convolutional Layer with 64 filters and a ReLU activation function.
* Max Pooling Layer.
* Flatten Layer to convert the 2D feature maps into a 1D feature vector.
* Two Dense Layers with ReLU activation functions and Dropout layers to prevent overfitting.
* The output layer with 10 neurons (for digit classes) and a softmax activation function.

To identify handwritten digits, a seven-layered convolutional neural network has been devised, as depicted in Figure 1. This network comprises an input layer, five hidden layers, and an output layer.

****

**Fig.1.**

The input layer is composed of 28 by 28 pixel images, resulting in 784 neurons within the network as the input data. These pixels are grayscale, represented by a value of 0 for a white pixel and 1 for a black pixel. Among the five hidden layers, the initial hidden layer is the first convolution layer, designed to extract features from the input data.

This specific layer operates by executing the convolution operation on localized areas using a filter, enabling the extraction of features from the preceding layer. It comprises multiple feature maps with learnable kernels and rectified linear units (ReLU). The kernel size defines the filter’s scope, while ReLU functions as the activation function in the convolution and fully connected layers, enhancing the model’s performance. The subsequent layer is the first pooling layer, tasked with reducing output information from the convolution layer, diminishing parameters and computational complexity. Among various pooling types like max, min, average, and L2 pooling, max pooling is utilized to downsample each feature map.

Similarly, the Convolution Layer 2 and Pooling Layer 2 function akin to their counterparts but differ in feature maps and kernel size. Following these layers, a Flatten layer converts the 2D feature map matrix into a 1D feature vector, facilitating handling by the fully connected layers. The fully connected layer, also known as the dense layer, interconnects all neurons between the preceding and subsequent layers. To curb overfitting, dropout regularization is applied, randomly deactivating neurons during training, enhancing the network’s robustness and generalization.

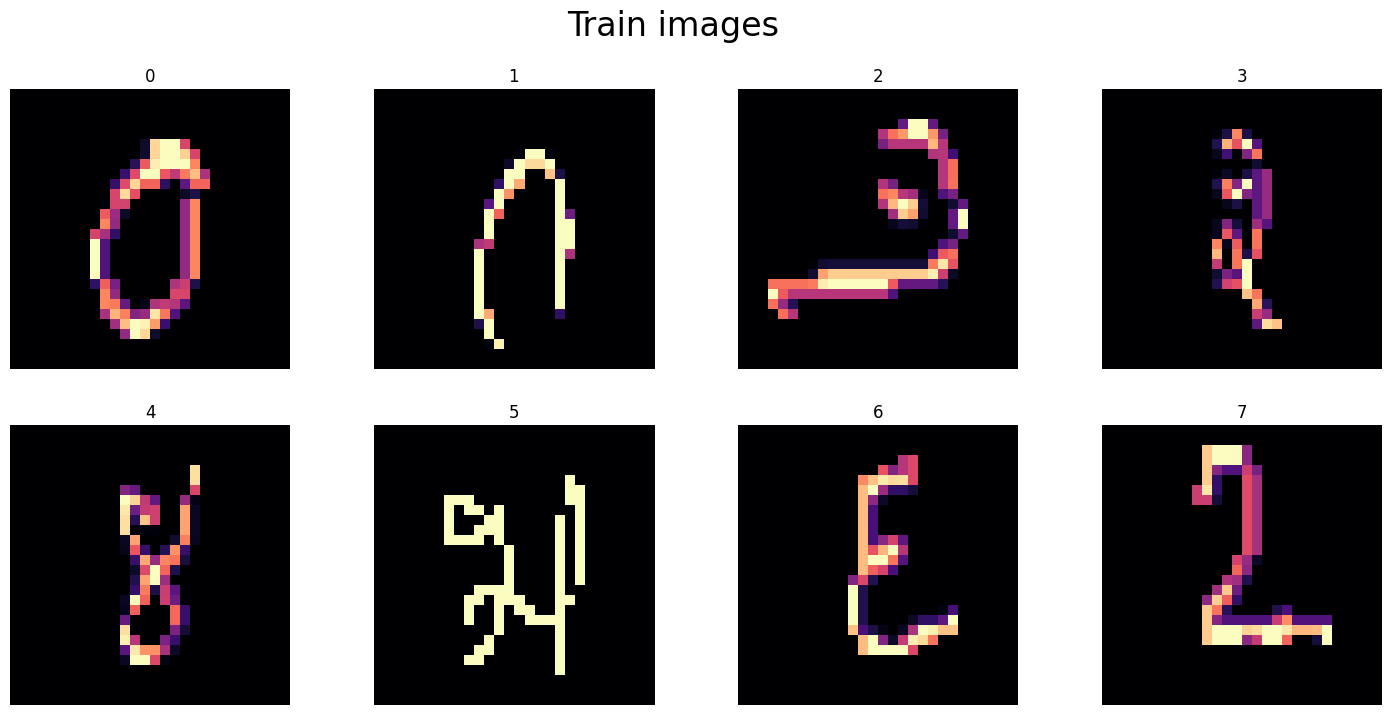
The output layer consists of ten neurons, classifying digits form 0 to 9 via an activation function like softmax. The MNIST databases, comprising 70,000 scanned handwritten digit images, is employed in the experiment. These grayscale images are all sized at 28x28 pixels. Training inputs, denoted as ‘x’, are 784-dimensional vectors representing the 28x28 pixel images. The desired output, ‘y(x)’, is a 10-dimensional vector. The network seeks to determine weights and biases so that its output approximates ‘y(x)’ for all training inputs, relying on these weight and bias values.

**RESULTS AND DISCUSSION**

Before diving into the main task, the code imports several Python libraries. These libraries are essential for various tasks involved in data processing, visualization, and model development like ‘numpy’, ‘pandas’, ‘matplotlib.pyplot’, ‘seaborn’, ‘matplotlib.image’, ‘train\_test\_split’, ‘confusion\_matrix’, ‘tensorflow.keras’, and more.

The code loads and examines the MNIST dataset. It reads the training and test datasets from CSV files. The training dataset contains images of handwritten digits along with their labels, while the test dataset contains only images without labels.

* **Train.head()** : This command displays the first few rows of the training dataset, allowing you to inspect the structure of the data.
* **Test.head():** Similarly, this command shows the first few rows of the test dataset.
* The code also checks for missing values in both datasets using **‘.isna().sum()**’ . Missing values are important to identify and handle, but in this case, the datasets appear to be clean with no missing values.
* The code further explores the distribution of labels in the training dataset using ‘ **train[‘label’].value\_coounts().sort\_index()**’. This provides a count of each digit(0-9) in the dataset, ensuring a balanced distribution.
* The code visually presents the first eight training images with their respective labels using Matplolib. It uitilizes a 2x4 grid of subplots to display these images.

****

* The test dataset is explored as well, even though it does not contain labels. The code plots the first eight test images to visualize the data.

**A screenshot of a test image

Description automatically generated**

**Preprocessing**

Before building and training a deep learning model, data preprocessing is essential. This section focuses on the steps involved in preparing the data for model training.

1. **Splitting the Data:**

* The training dataset is split into two components: input data(X) and lavels(y).
* The ‘**train**’ dataset is split into three components: ‘X’, ‘y’, and ‘test’. Each of these is converted into NumPy arrays.
* Normalization is performed on the input data by dividing it by 255.0, which scales pixel values to the range [0, 1].

1. **Reshaping:**

* The input data and test data are reshaped to represent images with a shape of 28x28x1. The ‘-1’ in ‘**reshape**’ is used to automatically infer the number of samples.
* The new shapes are confirmed to ensure proper reshaping.

1. **One-Hot Encoding:**

* The target labels(‘y’) are one-hot encoded using ‘**to\_categorical**’ from Keras. This transformation converts the singe integer labels into a binary matrix representation.

1. **Train-Validation Split:**

* The code splits the training data into training and validation sets using ‘**train\_test\_split’**. A random seed is set for resproducibility.
* The shapes of the resulting datasets are printed to confirm the split.

A pixelated image of a stethoscope

Description automatically generated A colorful pixelated image of a number

Description automatically generated

**Convolutional Neural Network (CNN)**

**Model Parameters**

Several parameters are set for the model:

* ‘**INPUT\_SHAPE’:** The shape of the input images (28x28x1, where 1 represents grayscale).
* ‘**OUTPUT\_SHAPE’:** The number of output classes (10 for digits 0-9).
* ‘**BATCH\_SIZE’:** The batch size used during training.
* ‘**EPOCHS’:** The number of training epochs.
* ‘**VERBOSE’:** The verbosity level for training output.

**COMPILE MODEL**

The model is compiled using the Adam optimizer, categorical cross-entropy loss (appropriate for multi-class classification), and accuracy as the evaluation metric.

**MODEL SUMMARY**

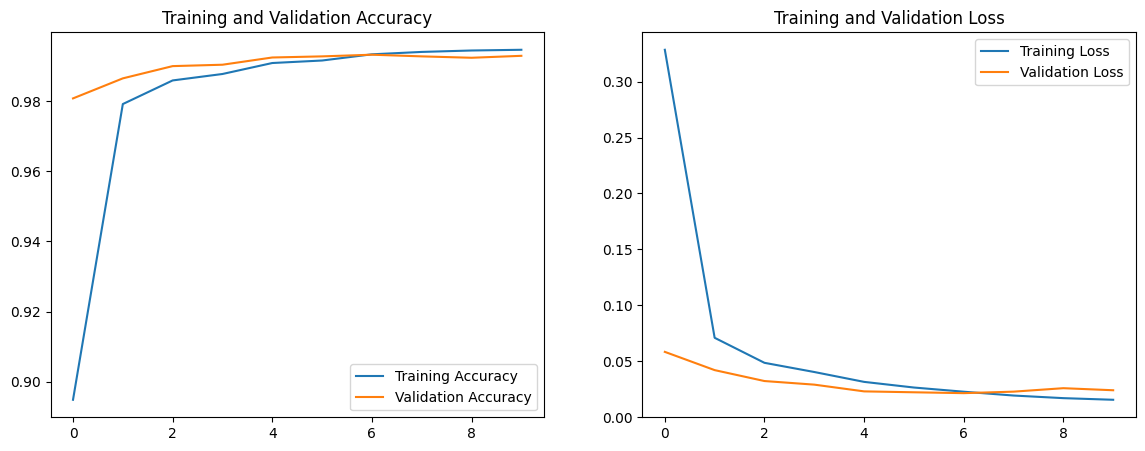
The code displays a summary of the model, providing details about the architecture, layers, and the number of parameters in each layer.

**MODEL FITTING**

The model is trained using the training data (‘X\_train’ and ‘y\_train\_enc’) with the specified number of epochs and batch size. Training progress is monitored and displayed as accuracy and loss curves.

**ACCURACY AND LOSS PLOTS**

The code generates plots displaying the training and validation accuracy as well as training and validation loss over the epochs. These plots help assess the model’s performance and identify overfitting or underfitting.



**Evaluating on Validation Dataset**

The code evaluates the model on the validation dataset using the ‘evaluate’ method. It provides metrics such as loss and accuracy on the validation set. Additionally, it generates a classification report, providing detailed statistics on precision, recall, F1-score, and support for each class.

A confusion matrix is also created and visualized using a heatmap, showing how well the model’s predictions align with the actual labels.

A graph with blue squares

Description automatically generated

**Predicting on the Test Dataset**

1. **Making Predictions:**

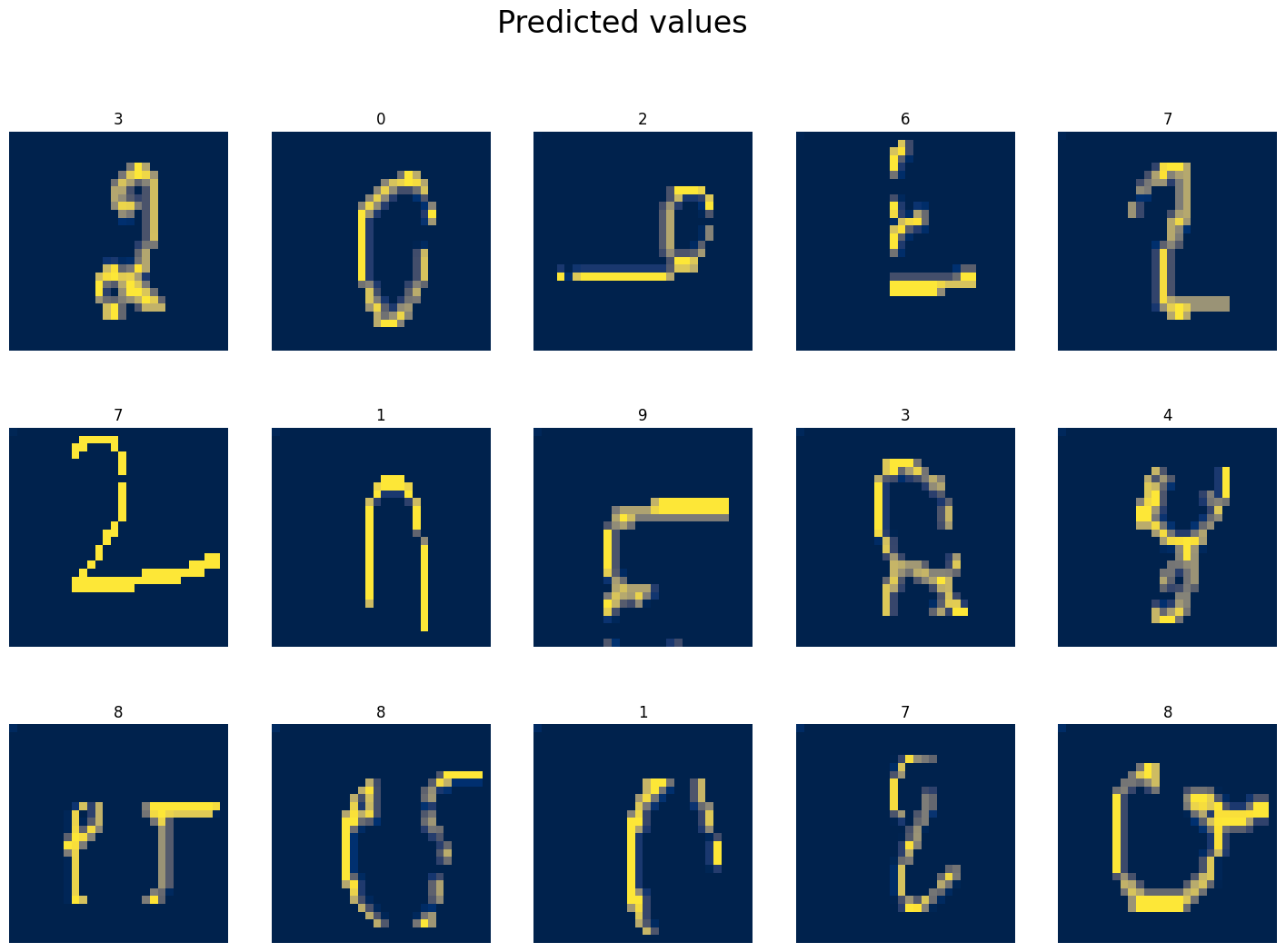
* The model’s ‘predict’ function is used on the test data (‘test’).
* The resulting predictions are stored in ‘**y\_pred\_enc’**, representing the predicted values as encoded arrays.

1. **Decoding Predicted Values:**

* These encoded predictions are decoded to obtain the actual digit labels using ‘**np.argmax’.**
* The decoded predictions are stored in ‘**y\_pred’,** representing the predicted label for the test images.

1. **Visualization of Predictions:**

* A visual display of the predicted digit labels for the first 15 test images is created using Matplotlib. Each image is shown along with its predicted label.



**CONCLUSION:**

* **Model Performance:**

The implemented Convolutional Neural Network (CNN) model exhibits promising performance in digit recognition. The model’s accuracy on the validation set and the training-validation loss convergence suggest that it's effectively learning and generalizing from the training data to unseen samples.

* **Validation and Evaluation:**

The evaluation metrics, including the classification report and confusion matrix, provide a comprehensive understanding of the model’s performance across different digit classes.

* **Data Preprocessing:**

The data was effectively preprocessed, including splitting into training and validation sets, normalization, reshaping images and encoding laves for model training.

* **Model Complexity:**

The CNN architecture includes convolutional layers, pooling layers, dense layers, and droupout regularization.

* **Prediction Visualization:**

The visual display of predicted labels for the test images allows for a quick qualitative assessment of the model’s predictions.

The provide code exemplifies an end- to- end process for developing a CNN model for digit recognition using the MNIST dataset. It covers data preprocessing, model building, training, evaluation, and prediction labels for unseen test images.

**REFERENCES**

LeCun, Y., 1998. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*

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

Siddique, F., Sakib, S. and Siddique, M.A.B., 2019. Handwritten digit recognition using convolutional neural network in Python with tensorflow and observe the variation of accuracies for various hidden layers.