CS574 - Computer Vision using Machine Learning

Assignment - 1 Report
Group-9

160101022 - Hari Boddu
160101029 - Ajay Ram Gudala
160101045 - Lakshmi Sai Durga
160101056 - Sachin Chouhan
160101064 - Pradeepa Seelam

Digit Recognition from MNIST dataset by
training various classifiers using
Logistic Regression,
Convolutional Neural Networks,
Deep Neural Networks
and Multi Layer Perceptron methods.

CONTENT

- Results and analysis of the four methods on varying different parameters over a range of epochs,showed overfitting problem for higher epochs.
- Explained Overfitting and Underfitting Problem, along with a regularization technique (Dropout) to overcome Overfitting Problem.
- Codes for the aforementioned four methods.

Logistic Regression

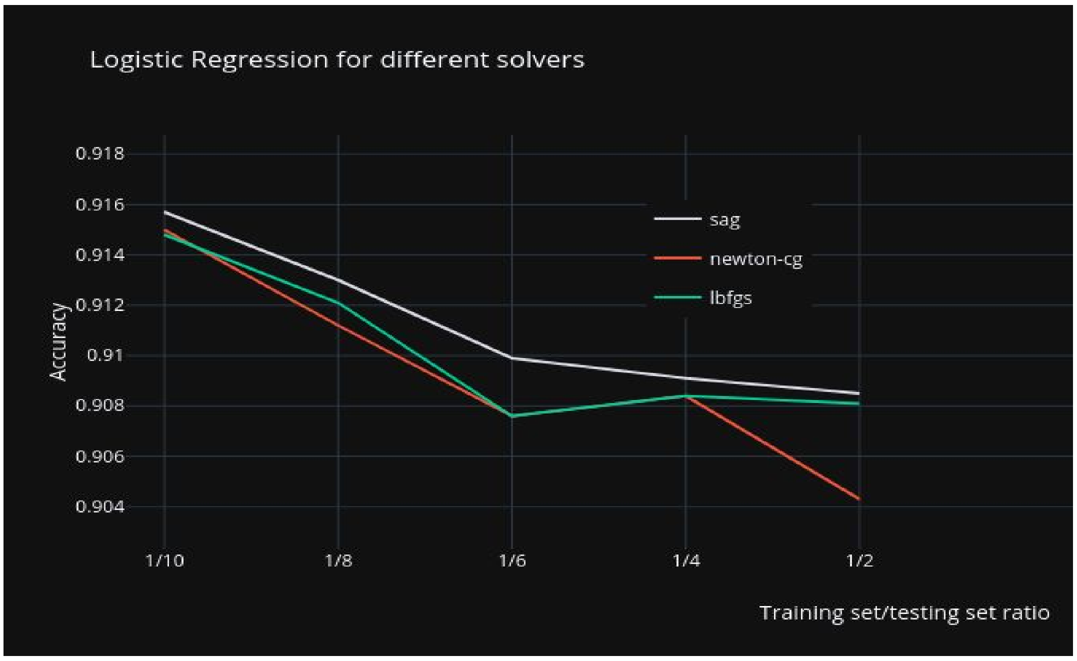
Regression analysis is a set of methods to estimate the relationship among different variables. Like all regression types, logistic regression is a predictive analysis, used for solving binary classification (two-class classification) problems. It describes the relation between a binary variable and other independent variables. Logistic regression is a statistical method for predicting binary classes. The outcome or target variable has only two possible classes

.

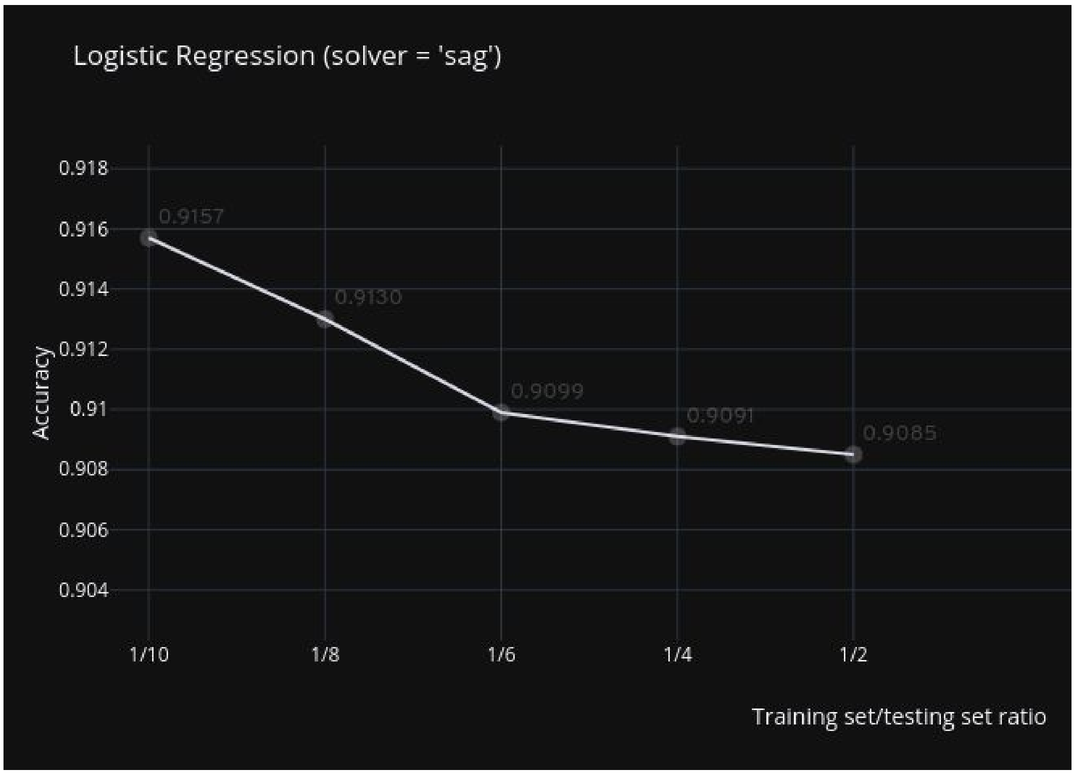
Solvers: These are the algorithms that can be used in optimization problem.The solvers implemented in the class LogisticRegression are "liblinear", "newton-cg", "lbfgs", "sag" and "saga".

- 'sag' is faster than other solvers for large datasets, when both the number of samples and the number of features is large.
- 'saga' is the solver of choice for sparse multinomial logistic regression.
- 'lbfgs' solver is recommended for use for small data-sets but for larger datasets its performance suffers.
- 'newton-cg' is an iterative equation solver and its computationally expensive.

Results after using the following solvers(Digit recognition from MNIST dataset with train:test=1:7) :



Results showing the change of accuracy with increasing training set size for solver 'sag' :



Multi layer perceptron

A perceptron is a linear classifier, that is, it is an algorithm that classifies input by separating two categories with a straight line. Multilayer perceptron (MLP) is a deep, artificial neural network, consisting of more than one perceptron. They are composed of an input layer to receive the signal, an output layer that makes a decision or prediction about the input.

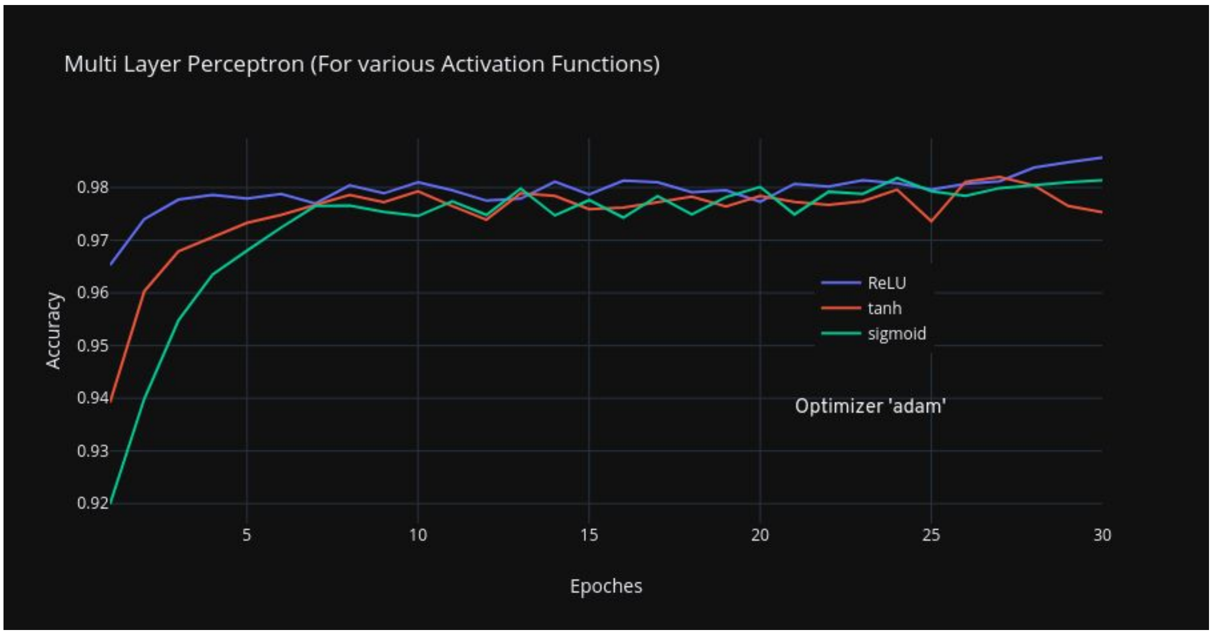
Activation Functions: Activation functions are really important for Multi Layer Perceptron to learn and make sense of something really complicated and Non-linear complex functional mappings between the inputs and response variable.They introduce non-linear properties to network.Their main purpose is to convert an input signal of a node in a MLP to an output signal. That output signal now is used as an input in the next layer in the stack.

Some of the important Activation functions are:

1. *Sigmoid or Logistic*
2. *Tanh — Hyperbolic tangent*
3. *ReLu -Rectified linear units*

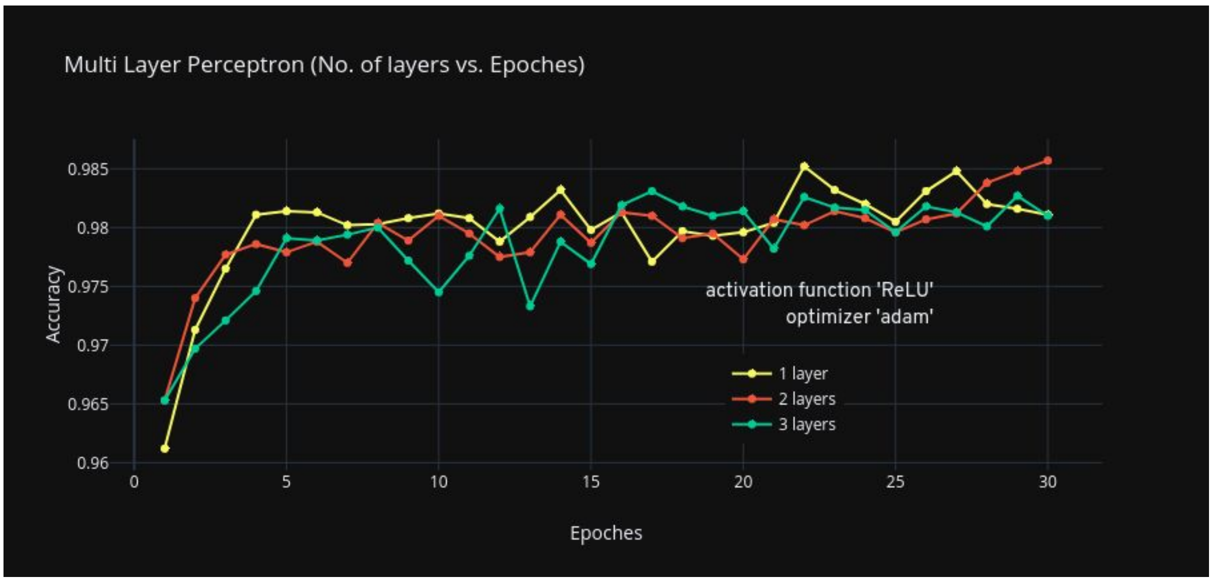
Accuracy is observed to be more for ReLU, tanh compared to sigmoid for the same dataset(considering the overfitting after #epochs=15).

Results showing the MLP for various activation functions(Digit recognition from MNIST dataset) :



Layers: The MLP consists of three or more layers (an input and an output layer with one or more *hidden layers*) of nonlinearly-activating nodes. Since MLPs are fully connected, each node in one layer connects with a certain weight to every node in the following layer.As the number of layers increase more will be the accuracy of prediction. But as the number of epochs increase beyond a limit model gets over trained(here it is seen after #epochs = 20) and hence a decrease in accuracy is observed.

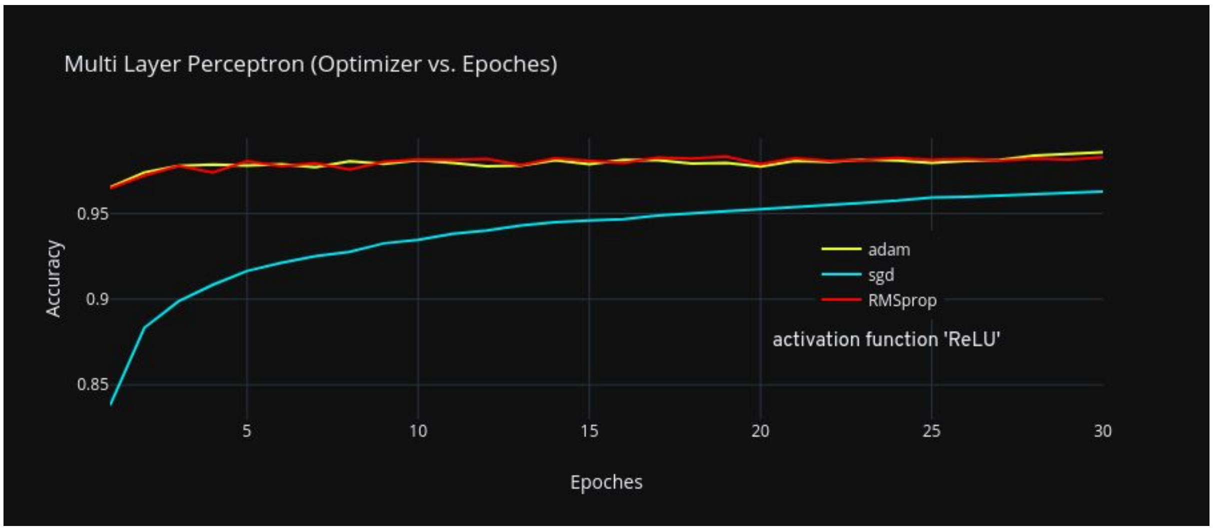
Results showing variation in accuracy with hidden layers :



Optimizers: Accuracy of prediction varies even on changing the optimizer being used in training the model. Optimizers tie together the loss function and model parameters by updating the model in response to the output of the loss function. In simpler terms, optimizers shape and mold your model into its most accurate possible form by futzing with the weights.

‘Adam’ and ‘RMSprop’ gave better results compared to ‘sgd’.

Results showing effect of optimizers on accuracy :



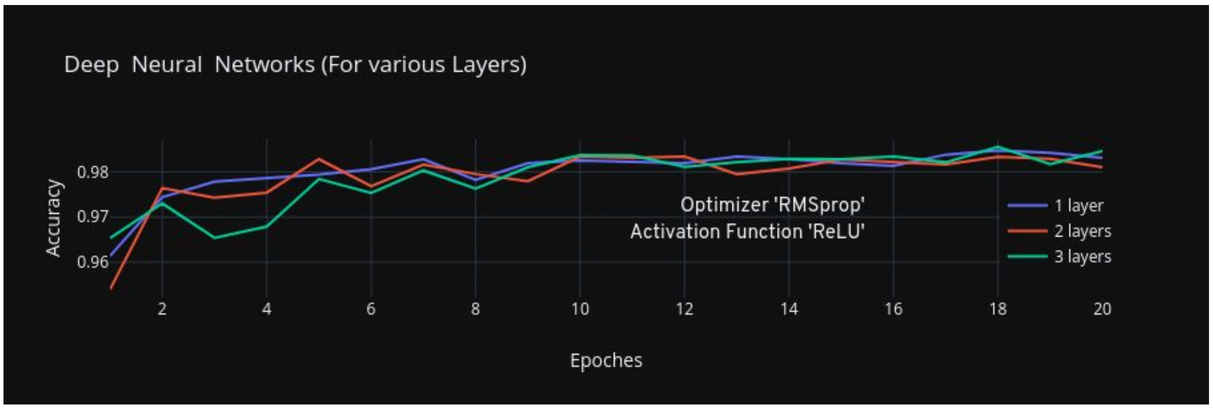
Deep Neural Network

A neural network, in general, is a technology built to simulate the activity of the human brain – specifically, pattern recognition and the passage of input through various layers of simulated neural connections. Neural networks help us cluster and classify. Deep neural network is a neural network with a certain level of complexity, a neural network with more than two layers. They help us group unlabelled data according to similarities among the example inputs and they classify data when they have a labelled dataset to train on.

Layers: Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'.

Adding more layers will help you to extract more features. But we can do that upto a certain extent. Increasing the number of hidden layers much more than the sufficient number of layers will cause accuracy in the test set to decrease(overfitting problem, here observed after #epochs = 10).

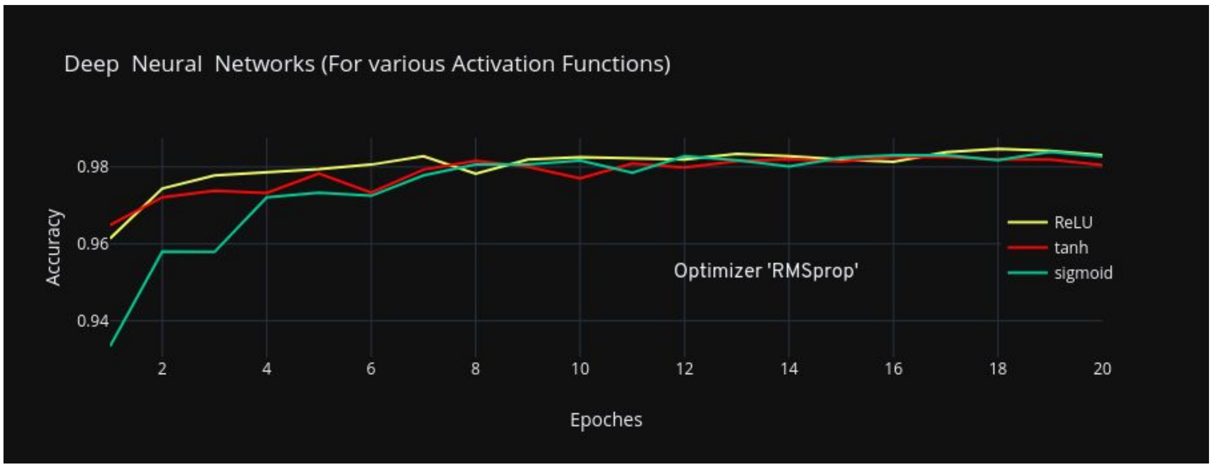
Results of variation in accuracy with increase in layers :



Activation Functions: Their main purpose is to convert an input signal of a node in a A-NN to an output signal. That output signal now is used as an input in the next layer in the stack.

Sigmoid Function is easy to understand and apply but it has major reasons which have made it fall out of popularity - it has slow convergence, it saturates and kills gradients.Optimization is easier in tanh method hence in practice it is always preferred over Sigmoid function.ReLU is very simple and efficient. Almost all deep learning Models use ReLu nowadays. (overfitting problem after #epochs = 15)

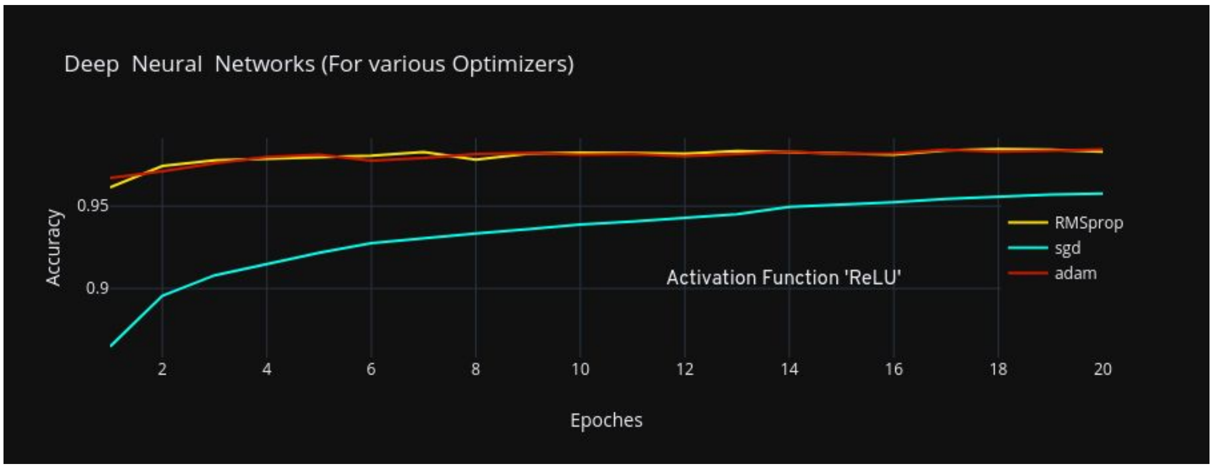
Results showing variation in accuracy for different activation functions :



Optimizers: Optimizers update the weight parameters to minimize the loss function. Loss function acts as guides to the terrain telling optimizer if it is moving in the right direction to reach the bottom of the valley, the global minimum.

‘Adam’ and ‘RMSprop’ proved to be better optimizers over ‘sgd’ for the given dataset.

Results showing variation in accuracy for different optimizers :



Deep Convolutional Neural Networks

Deep Convolutional Neural Network, is a class of Neural networks, commonly applied to analysis of visual imagery. Convolutional neural networks apply a filter to an input to create a feature map that summarizes the presence of detected features in the input. In summary, we have an input, such as an image of pixel values, and we have a filter, which is a set of weights, and the filter is systematically applied to the input data to create a feature map.

Pooling: A pooling layer is another building block of a CNN. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer operates on each feature map independently. The most common approach used in pooling is MaxPooling.

Max pooling is the application of a moving window across a 2D input space, where the maximum value within that window is the output

```
keras.layers.MaxPooling2D(pool_size=(2, 2))
```

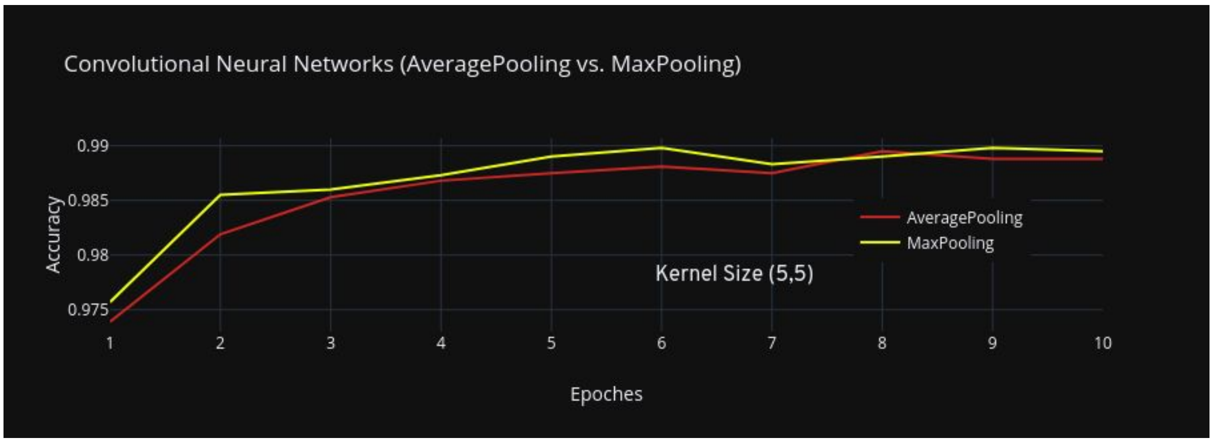
pool_size is an integer, indicating the size of the max pooling windows

Another approach - AveragePooling
Average pooling layer outputs the average values of rectangular regions of its input.

```
keras.layers.AveragePooling2D(pool_size=(2, 2))
```

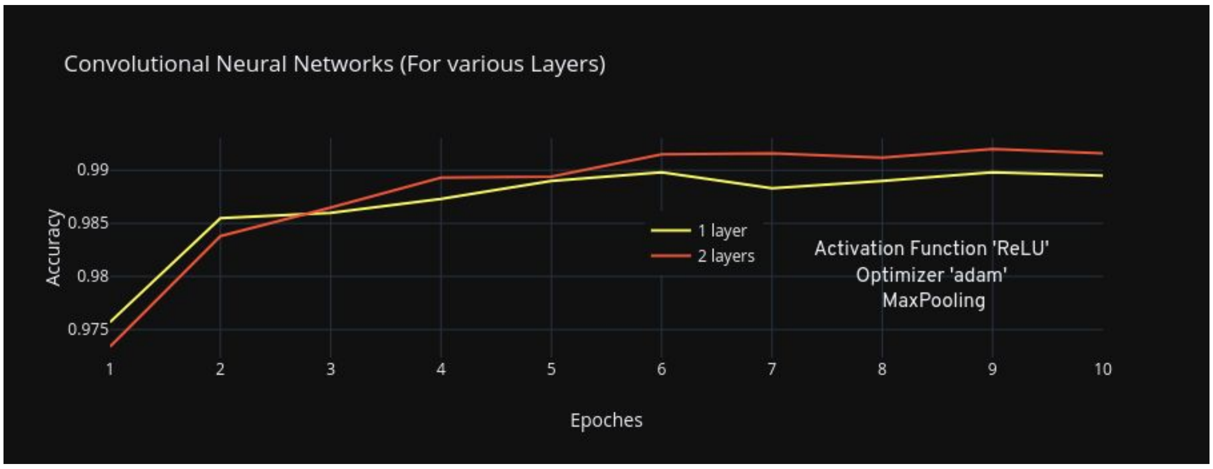
Max pooling extracts the most important features like edges whereas, average pooling extracts features so smoothly.

Results for different pooling approaches :



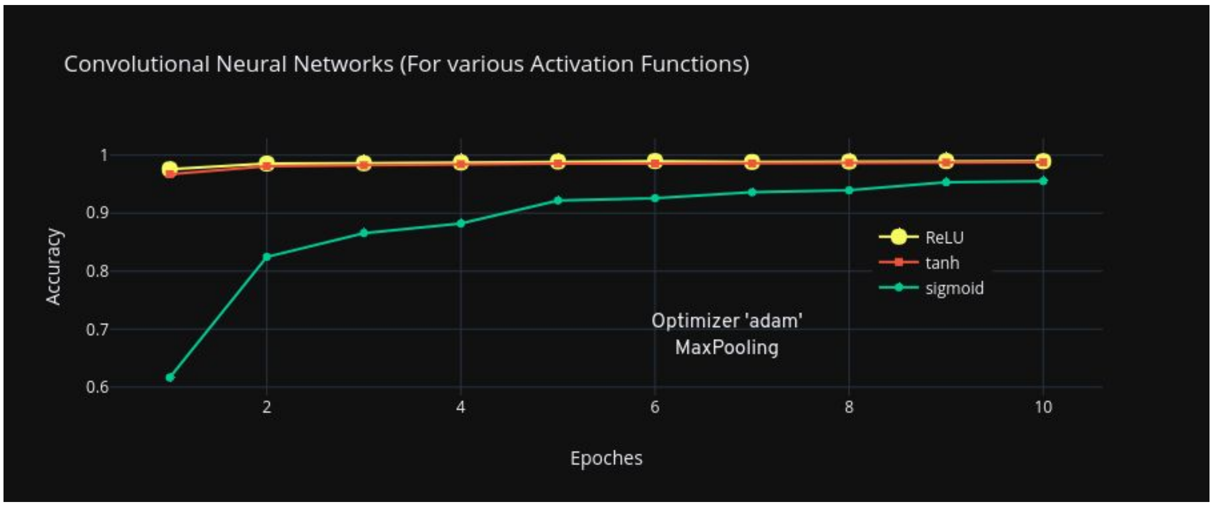
Layers: Dense is a name for a fully connected / linear layer in keras.Dense (fully connected) layers, which perform classification on the features extracted by the convolutional layers and downsampled by the pooling layers. In a dense layer, every node in the layer is connected to every node in the preceding layer.
Accuracy increases with increase in number of layers.

Results showing variation in accuracy for different no.of layers :



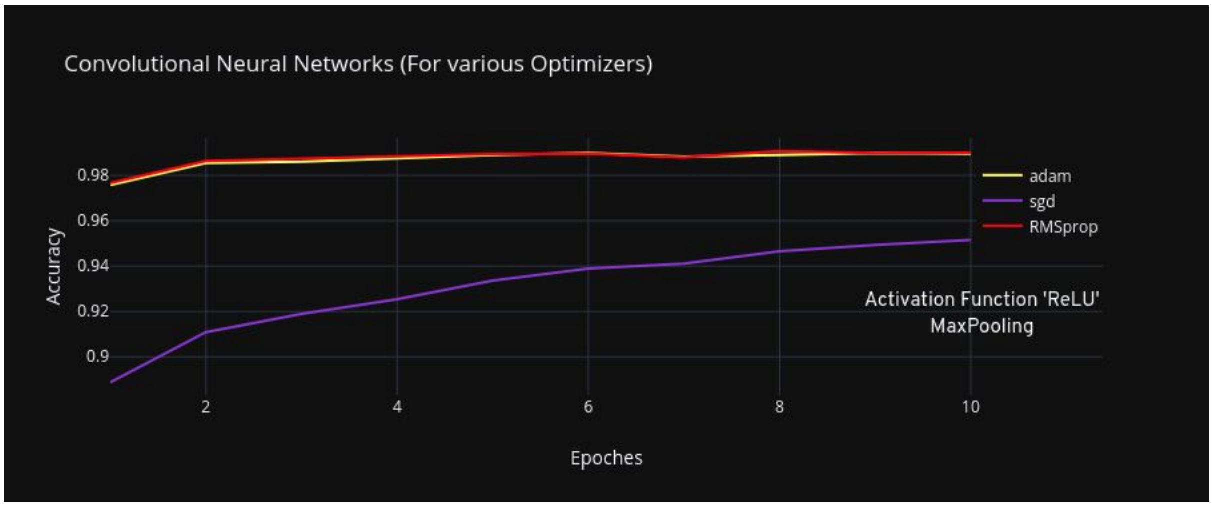
Activation Functions: The activation functions ‘ReLU’ and ‘tanh’ gave better results over ‘sigmoid’ for the given dataset.

Results showing variation in accuracy for different activation functions :



Optimizers: Optimizers ‘adam’ and ‘RMSprop’ gave better results over ‘sgd’ for the given dataset.

Results showing variation in accuracy for different optimizers :



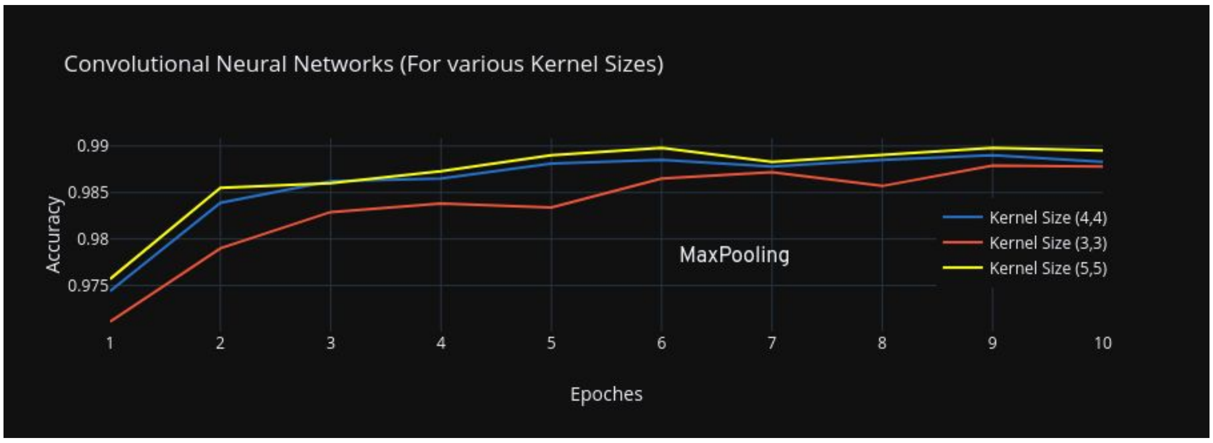
Convolution: Convolution is a function derived from two given functions by integration which expresses how the shape of one is modified by the other. *filters* is an integer, indicating the dimensionality of the output space (i.e. the number of output filters in the convolution).*kernel_size* is an integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window.

```
keras.layers.Conv2D(filters, kernel_size, input_shape)
```

In general smaller kernel sizes are popular choice over larger sizes.

.

Results for various kernel sizes :



OverFitting Problem

Overfitting refers to a model that models the training data too well.Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize.To avoid this, we can use regularization (**dropout**) or resampling(this helps us know the model's accuracy).

Dropout

The term dropout in context of CNN refers to the dropping out units - both hidden and visible during the training phase of certain set of neurons which is chosen at random.It is a technique where randomly selected neurons are ignored during training. More technically, at each training stage, individual nodes are either dropped out of the net with probability *1-p* or kept with probability *p*, so that a reduced network is left, incoming and outgoing edges to a dropped-out node are also removed.

UnderFitting Problem

Underfitting refers to a model that can neither model the training data nor generalize to new data. An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data. The remedy is to move on and try other machine learning algorithms.

Codes

Logistic Regression

```
%matplotlib inline
from sklearn.datasets import fetch_openml

mnist = fetch_openml('MNIST_784')

from sklearn.model_selection import train_test_split
train_img, test_img, train_lbl, test_lbl =
train_test_split(mnist.data, mnist.target, test_size=1/7.0,
random_state=0)

import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
# all parameters not specified are set to their defaults
# default solver is incredibly slow that's why we change it
logisticRegr = LogisticRegression(solver = 'lbfgs')
logisticRegr.fit(train_img, train_lbl)
# Returns a NumPy Array
# Predict for One Observation (image)
logisticRegr.predict(test_img[0].reshape(1,-1))
# Predict for Multiple Observations (images) at Once
logisticRegr.predict(test_img[0:10])
# Make predictions on entire test data
predictions = logisticRegr.predict(test_img)
score = logisticRegr.score(test_img, test_lbl)
print(score)
```

Multi Layer Perceptron

```
# Plot ad hoc mnist instances
from keras.datasets import mnist
import matplotlib.pyplot as plt

(X_train, y_train), (X_test, y_test) = mnist.load_data()

import numpy
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.utils import np_utils
```

```
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# flatten 28*28 images to a 784 vector for each image
num_pixels = X_train.shape[1] * X_train.shape[2]
X_train = X_train.reshape(X_train.shape[0],
num_pixels).astype('float32')
X_test = X_test.reshape(X_test.shape[0],
num_pixels).astype('float32')
# normalize inputs from 0-255 to 0-1
X_train = X_train / 255
X_test = X_test / 255
# one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
# define baseline model
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(num_pixels, input_dim=num_pixels,
kernel_initializer='normal', activation='relu'))
#     model.add(Dense(num_classes, kernel_initializer='normal',
activation='relu'))
#
model.add(Dense(512,kernel_initializer='normal',activation='softma
x'))
    model.add(Dense(100))
    model.add(Dense(100))
    model.add(Dense(num_classes,kernel_initializer='normal',
activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
    return model
# build the model
model = baseline_model()
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=30, batch_size=200, verbose=2)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
# print("Baseline Error: %.2f%%" % (100-scores[1]*100))
print(scores[1])
```

Deep Neural Networks

```
from __future__ import print_function

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop

batch_size = 128
num_classes = 10
epochs = 20
# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
# print(x_train.shape[0], 'train samples')
# print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',optimizer='RMSprop',
metrics=['accuracy'])

history = model.fit(x_train,
y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_d
ata=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
# print('Test loss:', score[0])
print(score[1])
```

Convolutional Neural Networks

```
import numpy
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.convolutional import AveragePooling2D
from keras.utils import np_utils
from keras import backend as K
```

```
K.set_image_dim_ordering('th')
seed = 7
numpy.random.seed(seed)
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# reshape to be [samples][pixels][width][height]
X_train = X_train.reshape(X_train.shape[0], 1, 28,
28).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 1, 28,
28).astype('float32')
# normalize inputs from 0-255 to 0-1
X_train = X_train / 255
X_test = X_test / 255
# one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]


def baseline_model():
      # create model
      model = Sequential()
      model.add(Conv2D(32, (5, 5), input_shape=(1, 28, 28),
activation='relu'))
      model.add(MaxPooling2D(pool_size=(2, 2)))
      model.add(Dropout(0.2))
      model.add(Flatten())
      model.add(Dense(128, activation='relu'))
      model.add(Dense(num_classes, activation='softmax'))
      # Compile model
      model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
      return model

# build the model
model = baseline_model()
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs= 10, batch_size=200, verbose=2)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
# print("CNN Error: %.2f%%" % (100-scores[1]*100))

print(scores[1])
```