convolutional neural network using Keras for Mnist dataset with user interface Gui

Implementation file contains two python notebooks: main.py and interface.py and conv2D_classifier for saving the best model and use it for predict user input.

First you should download The MNIST database of handwritten digits from "http://yann.lecun.com/exdb/mnist/" that contains four files:

train-images.idx3-ubyte: training set images

train-labels.idx1-ubyte: training set labels

t10k-images.idx3-ubyte: test set images

t10k-labels.idx1-ubyte: test set labels

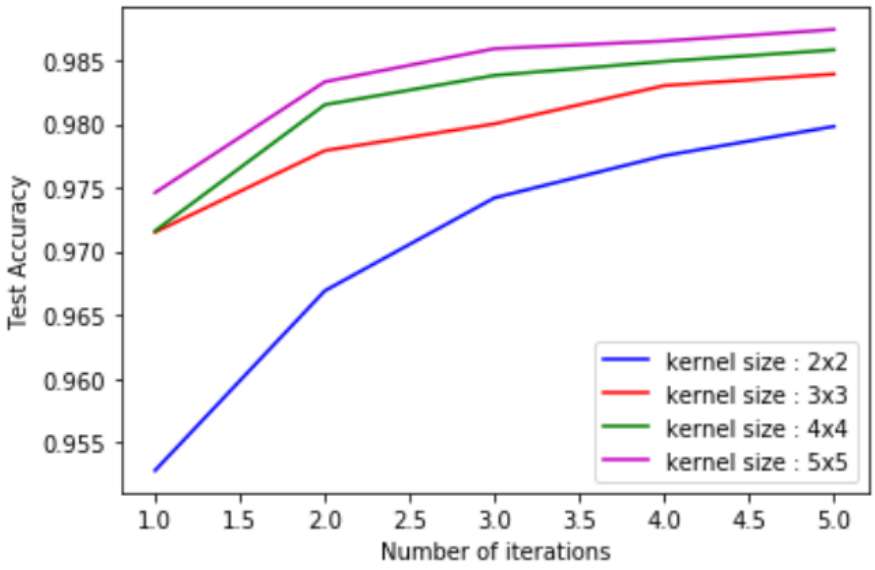
and extract it in python notebook files folder,

main.py contains:

- image_t class which contains : 1- output_terminal function to print digits to terminal , 2- normalize function to normalize digits with dividing each entry by 255 , 3- onehot_label function to create one-hot vector for each label (for example if the class is 2 , then the vector should be like [0,1,0,0,0,0,0,0,0,0])
- read_words function which is used for reading in the metadata of the file at the header (read a single word from a file and return the decimal representation)
- read_bytes and get_images function for visualization
- load_data function load test and train images and labels and correlate labels with images
- load_normalize_vectorize function which use load_data function , normalize and vectorize the data
- main function which build a CNN model contains convolutional layers, pooling layers and fully connected layers, fit the model on train data and train the model using gradient descent and at last evaluate the model on test data at each epoch , you can change these parameters in main function : kernel_size, kernel_numbers, learning_rate, activation_function
- save_model function will save the trained model with name "conv2D_classifier.h5" in python notebook folder

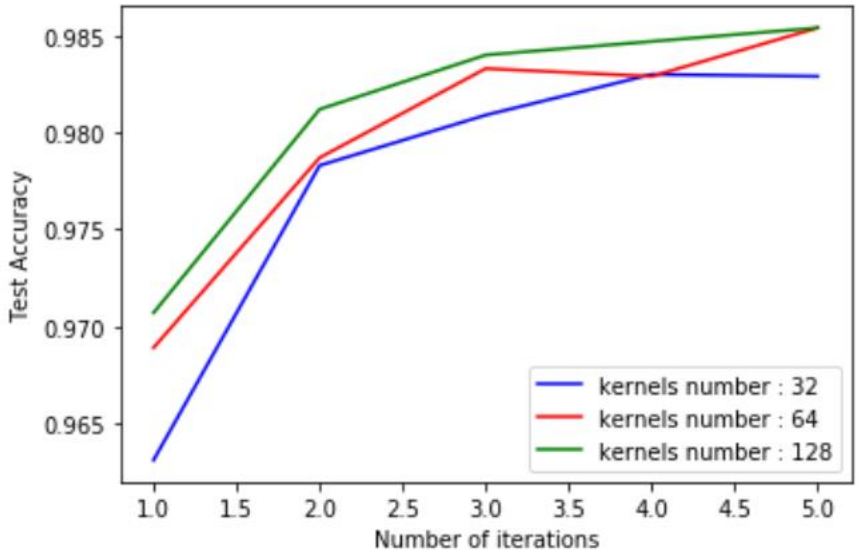
First loading the data and test the effect of different parameters of model for data:

- Test for different kernel sizes:



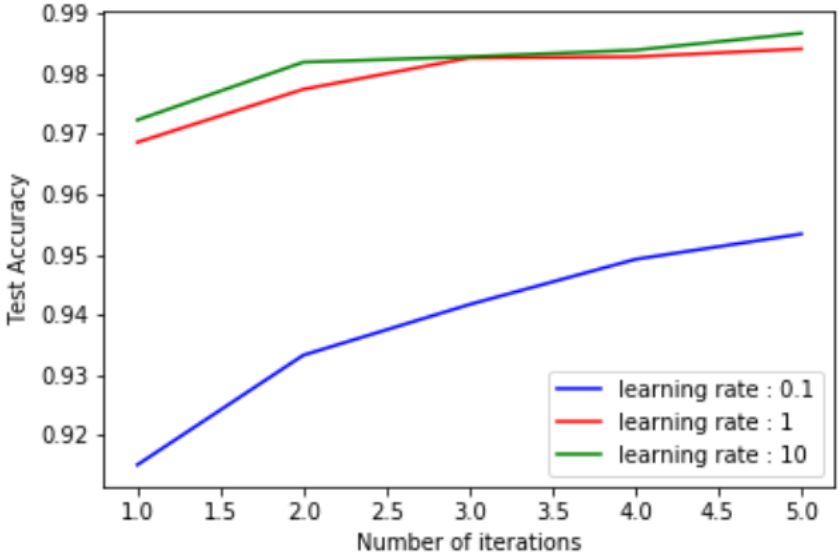
In this model accuracy increases with increasing the kernel size, but in general there is no direct relation between the kernel size and the accuracy. If you start using larger kernel you may start loosing details in some smaller features (where 3x3 would detect them better) and in other cases, where your dataset has larger features the 5x5 may start detect features that 3x3 misses.

- Test for different numbers of kernels:



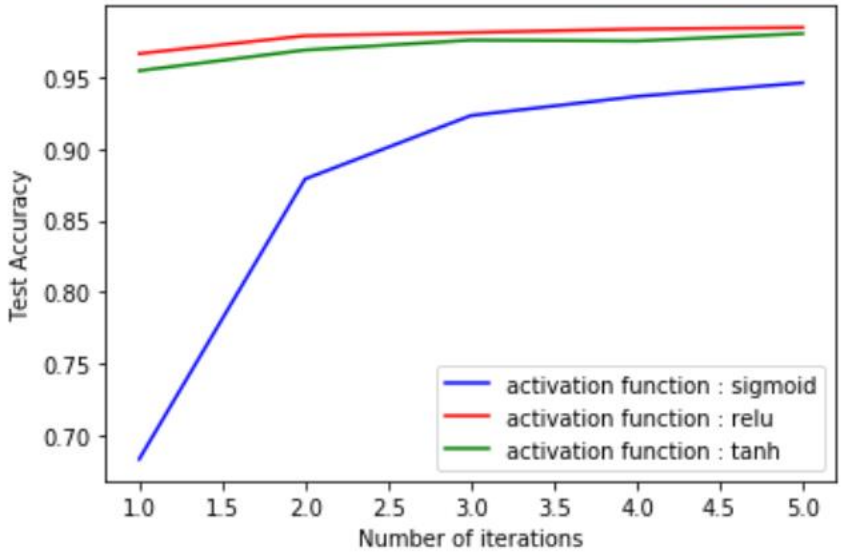
In this model accuracy increases with increasing the number of kernels, although it depends on the problem domain again, the significance # of feature detectors intuitively is the number of features (like edges, lines, object parts etc...) that the network can potentially learn. Also note that each filter generates a feature map. Feature maps allow you to learn the explanatory factors within the image, so the more # filters means the more the network learns (not necessarily good all the time - saturation and convergence matter the most)

- Test for different Learning rate values:



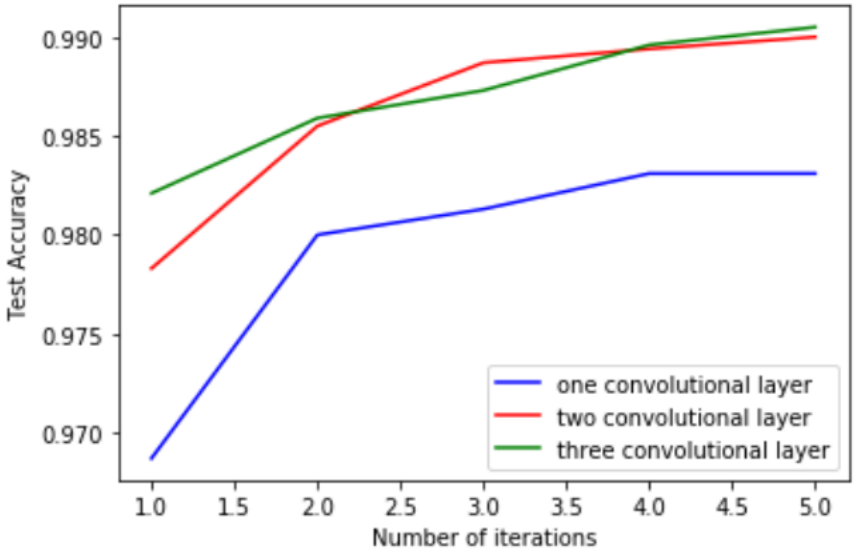
In this model accuracy increases with increasing the learning rate value, generally, a large learning rate allows the model to learn faster, at the cost of arriving on a sub-optimal final set of weights. A smaller learning rate may allow the model to learn a more optimal or even globally optimal set of weights but may take significantly longer to train.

- Test for different activation functions:



In this model , " relu " function has the best accuracy , then " tanh " function and then " sigmoid " function

- Test for different numbers of layers:



In this model increasing the number of layers increases the accuracy, but not always is like this, adding layers increases the number of weights in the network, also the model complexity. Without a large training set, an increasingly large network is likely to overfit and in turn reduce accuracy on the test data. Adding more layers will help you to extract more

features. But we can do that up to a certain extent. There is a limit. After that, instead of extracting features, we tend to 'overfit' the data. Overfitting can lead to errors in some or the other form like false positives.

interface.py creates a user interface to draw digits and predict it using the saved model from main.py.

interface.py contains:

image_t class , predict_worker class for predict the drawing digit on user interface window , drawing_path class to get the drawing digit as an image, window class to create user interface window and main function to do the prediction .

some examples of predictions: