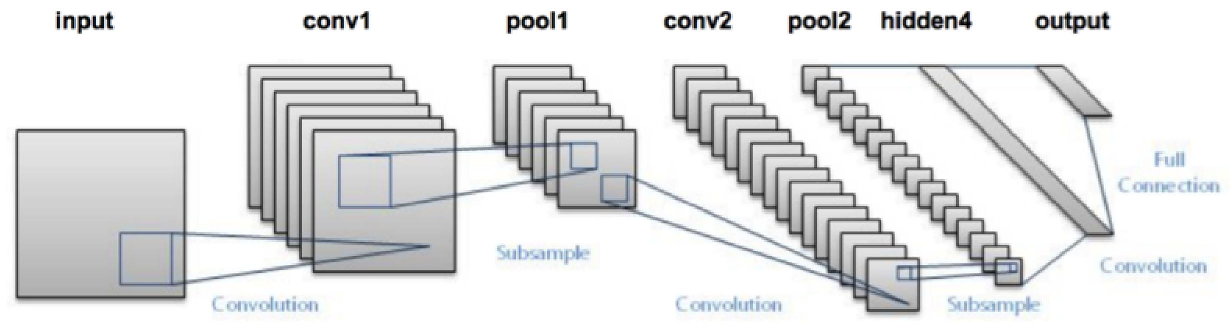
COL341
Assignment 3- Image Classification

Prabhat Kanaujia
2016CS50789

Initial Architecture:

LeNet-5 was the starting point for this assignment. It combines two 2D convolution layers, each of which is followed by a pooling layer. Then there is a dense layer to connect all the extracted features and finally the output layer.



In the initial architecture, I used 20 filters in the first Conv layer and 50 in the second. **Tanh** was used as the activation function in all the layers, except the output layer, where **Softmax** was used.

I used Keras(tensorflow-gpu backend) to build and train this sequential model.
This architecture, without any feature creation and modelling, gave me an accuracy of about 97%.

Improvements:

A. **Using ReLU:** ReLU was used instead of tanh in the hidden layers because it can efficiently avoid the **“vanishing gradients”** problem. Using ReLU bumped up the accuracy to 97.8% on the test data.

   It does have its inherent weaknesses, such as **dead ReLU**, which I planned on fixing by using **leaky-ReLU**, but they didn’t improve the performance and so I continued with the standard ReLU.

B. Some changes in loss function and learning methods:

   a. I used categorical cross entropy loss function

   b. After trying several optimization methods(SGD, RMSprop and Nadam), I settled for **adam** with an initial learning rate of 0.001 and other parameters as per the default suggestions in the paper -> **“ADAM : A METHOD FOR STOCHASTIC OPTIMIZATION”.**

   c. Used the **ReduceLROnPlateau** implementation in Keras to reduce the learning rate by **half** as soon as the **loss on the training** data stopped decreasing for **more than 3** consecutive epochs.

   d. Batch-size of 128 was used.

   e. Normalized the inputs pixels, to bring their values between 0 and 1.

   f. Tried to implement early stopping, but due to the above method to decrease the learning rate, the algorithm kept on improving till almost the end, i.e. 100 epochs. So I removed this function in the final submission.

C. **Additional Layers and Filters:** I tried several modifications to the original LeNet-5 architecture. Increasing the number of layers and number of filters per layer led to significant improvements in accuracy.
   Final architecture I settled for is:

   a. model.add(Conv2D(16, 3, padding="same",activation='relu'))
   b. model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

   c. model.add(Conv2D(32, 3, padding="same",activation='relu'))
   d. model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

   e. model.add(Conv2D(64, 3, padding="same",activation='relu'))
   f. model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

   g. model.add(Flatten())
   h. model.add(Dense(500,activation='relu'))

   i. model.add(Dense(46))
   j. model.add(Activation("softmax"))

This gave an accuracy of 98.6%. Adding extra layers didn't lead to any significant increase in accuracy after this.

D. **Dropout:** A dropout of **0.25** was used after each pooling layer and a dropout of **0.5** was used after the dense hidden layer. It made the model learn more robust features and hence, generalize well to the test data. This led to a significant increase in accuracy to about 99%.

E. **Feature Augmentation:** This part led to a major increase in accuracy and is incorporated in my submission. I used the **ImageDataGenerator** library in Keras with the following parameters:

```
ImageDataGenerator(rotation_range=10,zoom_range=0.1,width_shift_range=0.1,height_shift_range=0.1)
```

It means that instead of training on the entire data divided into certain batch-sizes, I trained my model on small batches of augmented images, obtained by rotating, zooming into, and translating the original images. This led the algorithm to learn better and more robust features.
There was a slight decrease in training accuracy, but the overall test accuracy increased to 99.45%.

REFERENCES:

1. https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5
2. http://slazebni.cs.illinois.edu/spring17/lec01_cnn_architectures.pdf
3. https://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/
4. https://keras.io/callbacks/