**CODE EXPLANATION**

**Q1**. Input Image Shape/Size: Taken as 28x28: using LeNet architecture:

LeNet Architecture:

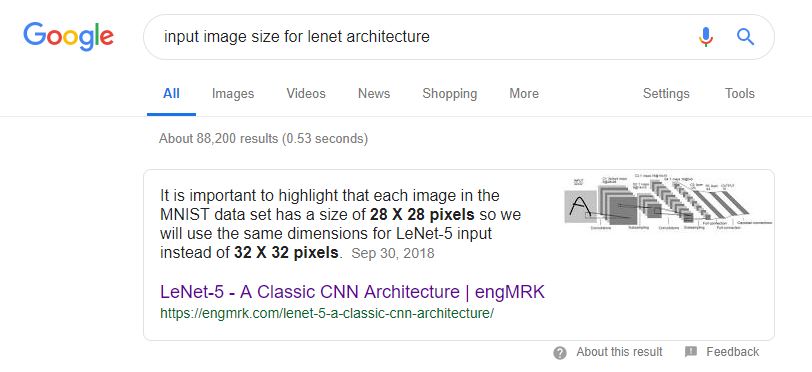
WHAT?

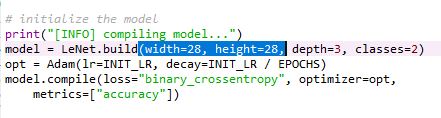
The LeNet architecture was first introduced by LeCun et al. in their 1998 paper, Gradient-Based Learning Applied to Document Recognition. As the name of the paper suggests, the authors’ implementation of LeNet was used primarily for OCR and character recognition in documents.

The LeNet architecture consists of two sets of convolutional, activation, and pooling layers, followed by a fully-connected layer, activation, another fully-connected, and finally a softmax classifier. We’ll be implementing this network architecture using Keras and Python.

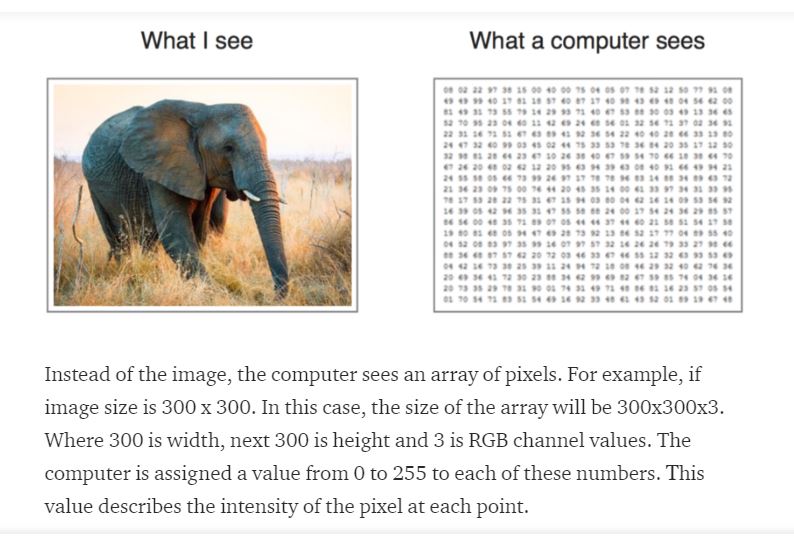
WHY?

The LeNet architecture is straightforward and small, (in terms of memory footprint), making it perfect for teaching the basics of CNNs — it can even run on the CPU (if your system does not have a suitable GPU)





CONCLUSION: 28x28 is the dimensions of the 2D array i.e. storage of the image, for our project we can take a different input size Since we are classifying b/w images of LP’s and NONLP’s.



**Source:**

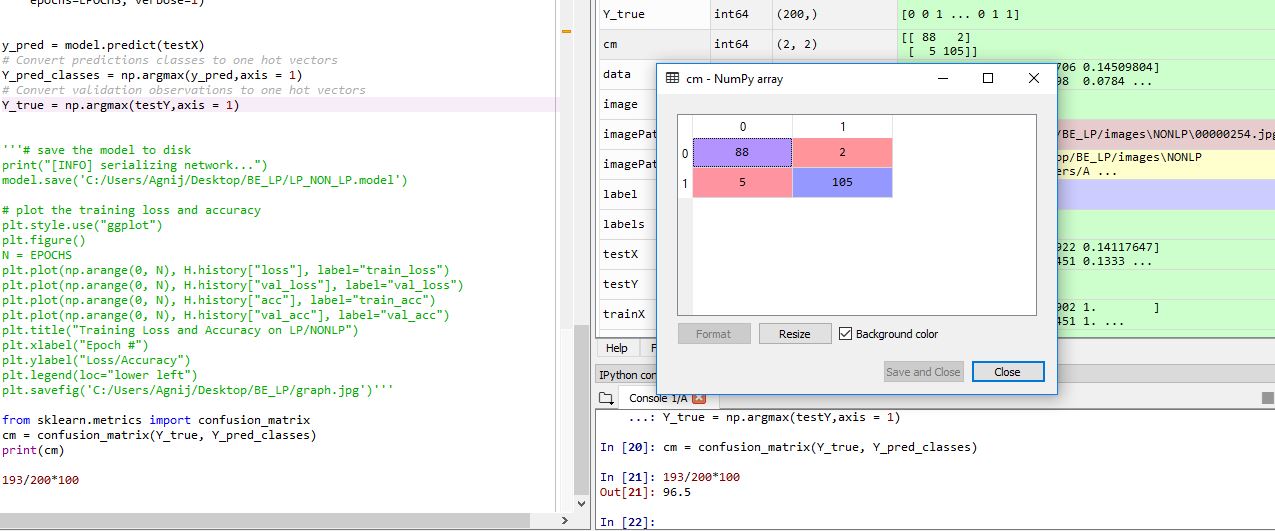
**https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8**

**\*\*Using Input Size as 28x28** **, for the first set of CONV => RELU => POOL layers we have selected --20 --feature maps of filter size --5x5-- and for the second set of CONV => RELU => POOL layers we have selected—50-- feature maps of filter size --5x5-- , for the first fully connected layer we have selected ----500-- i.e. output of the dimensionality space.**



Training accuracy is 95% and val\_acc i.e. test accuracy is 96.5%

Using the Confusion matrix we get a test accuracy again the same as above i.e. 96.5%



TP: 105 images are truly predicted as LP’s, TN: 88 images are truly predicted as NONLP’s,

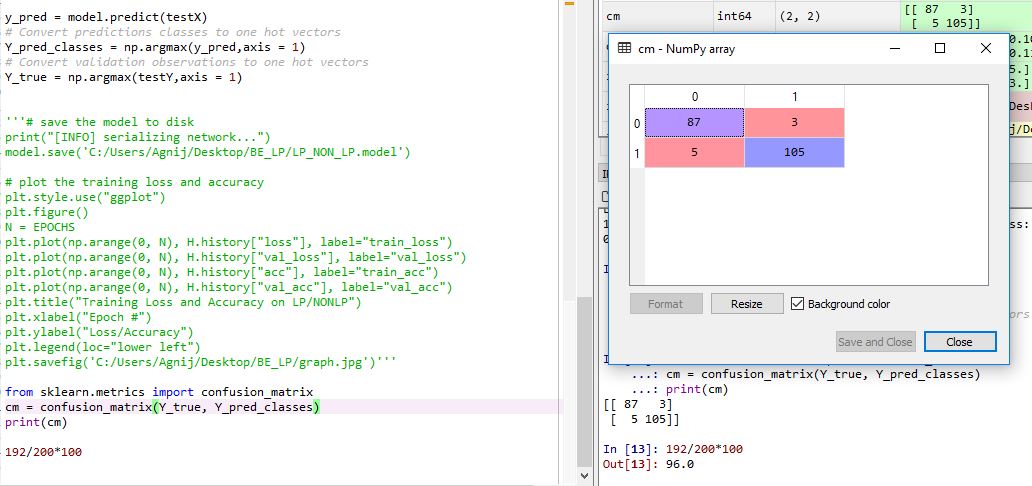
FP: 2 images are falsely predicted as LP’S, FN: 5 images are falsely predicted as NONLP’s.

**\*\*Using Input Size as 32x32** **, for the first set of CONV => RELU => POOL layers we have selected --16 --feature maps of filter size --3x3-- and for the second set of CONV => RELU => POOL layers we have selected --32-- feature maps of filter size-- 3x3-- and for the first fully connected layer we have selected --128-- i.e. output of the dimensionality space.**

****

Training accuracy is 95.89% and val\_acc i.e. test accuracy is 96%

Using the Confusion matrix we get a test accuracy again the same as above i.e. 96%



TP: 105 images are truly predicted as LP’s, TN: 87 images are truly predicted as NONLP’s,

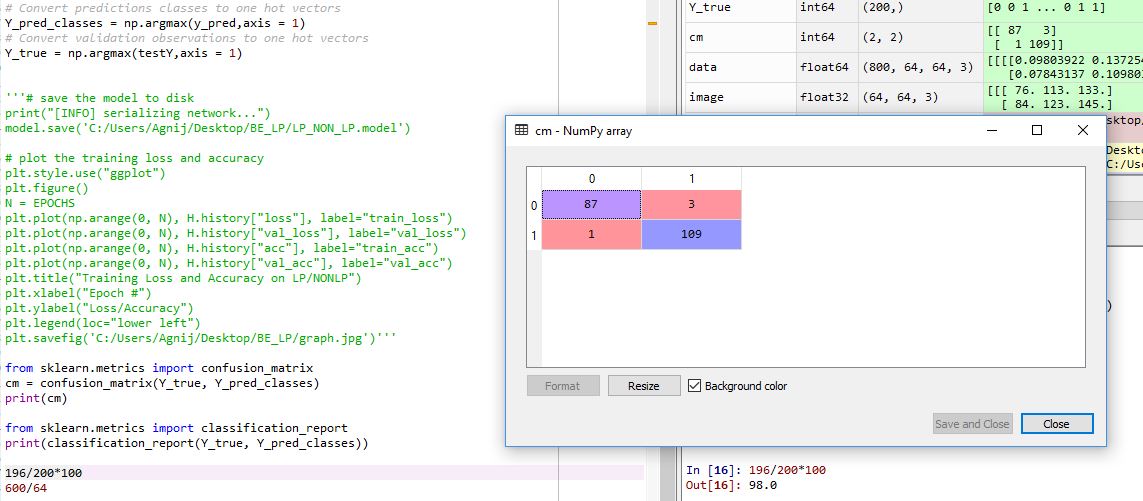
FP: 3 images are falsely predicted as LP’s, FN: 5 images are falsely predicted as NONLP’s.

**\*\*Using Input Size as 64x64** **, for the first set of CONV => RELU => POOL layers we have selected --32-- feature maps of filter size --3x3-- and for the second set of CONV => RELU => POOL layers we have selected --64-- feature maps of filter size --3x3-- and for the first fully connected layer we have selected--128-- i.e. output of the dimensionality space.**



Training accuracy is 97.51% and val\_acc i.e. test accuracy is 98%

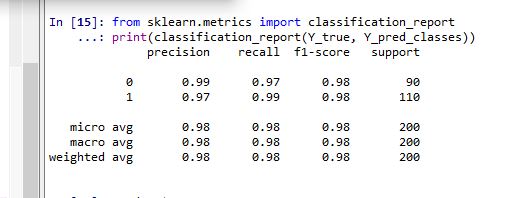
Using the Confusion matrix we get a test accuracy again the same as above i.e. 98%



TP: 109 images are truly predicted as LP’s, TN: 87 images are truly predicted as NONLP’s,

FP: 3 images are falsely predicted as LP’s, FN: 1 image is falsely predicted as NONLP’s.

**CONCLUSION**

****

**Here the precision = TP/(TP+FP) for LP’s is 97% and a f1 score of 0.98 i.e. the highest.**

**Therefore we can go ahead with this model (i.e. input size 64x64) in comparison to the previous two models (Input Size as 28x28 AND Input Size as 32x32).**

**\*\*LEARNING RATE (Taken AS 1e - 3)**

**This parameter scales the magnitude of our weight updates in order to minimize the network's loss function.**

**If your learning rate is set too low, training will progress very slowly as you are making very tiny updates to the weights in your network. However, if your learning rate is set too high, it can cause undesirable divergent behavior in your loss function.**



**Source: https://www.jeremyjordan.me/nn-learning-rate/**

**Q2. EPHOCHS (Taken AS 25), BATCH SIZE (Taken AS 32), and IMAGE AUGMENTATION.**

**\*\*The above values were all the same for the previously shown three models.**

The entire dataset consists of 800 images i.e. 400 LP’s and 400 NONLP’s. Further we have divided this dataset into a training set containing 600 images and a test set containing 200 images.

Now considering the input size as 64x64 we went ahead and experimented with the Batch Size.

**BATCH SIZE**

The batch size is a hyper parameter that defines the number of samples to work through before updating the internal model parameters.

Think of a batch as a for-loop iterating over one or more samples and making predictions. At the end of the batch, the predictions are compared to the expected output variables and an error is calculated. From this error, the update algorithm is used to improve the model, e.g. move down along the error gradient.

A training dataset can be divided into one or more batches.

* **Batch Gradient Descent**. Batch Size = Size of Training Set
* **Stochastic Gradient Descent**. Batch Size = 1
* **Mini-Batch Gradient Descent**. 1 < Batch Size < Size of Training Set

In the case of mini-batch gradient descent, popular batch sizes include 32, 64, and 128 samples.

Mini-batch gradient descent seeks to find a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent. It is the most common implementation of gradient descent used in the field of deep learning.

**Upsides**

* The model update frequency is higher than batch gradient descent which allows for a more robust convergence, avoiding local minima.
* The batched updates provide a computationally more efficient process than stochastic gradient descent.
* The batching allows both the efficiency of not having all training data in memory and algorithm implementations.

**Downsides**

* Mini-batch requires the configuration of an additional “mini-batch size” hyper parameter for the learning algorithm.
* Error information must be accumulated across mini-batches of training examples like batch gradient descent.

Mini-batch sizes, commonly called “batch sizes” for brevity, are often tuned to an aspect of the computational architecture on which the implementation is being executed. Such as a power of two that fits the memory requirements of the GPU or CPU hardware like 32, 64, 128, 256, and so on.

Batch size is a slider on the learning process.

* Small values give a learning process that converges quickly at the cost of noise in the training process.

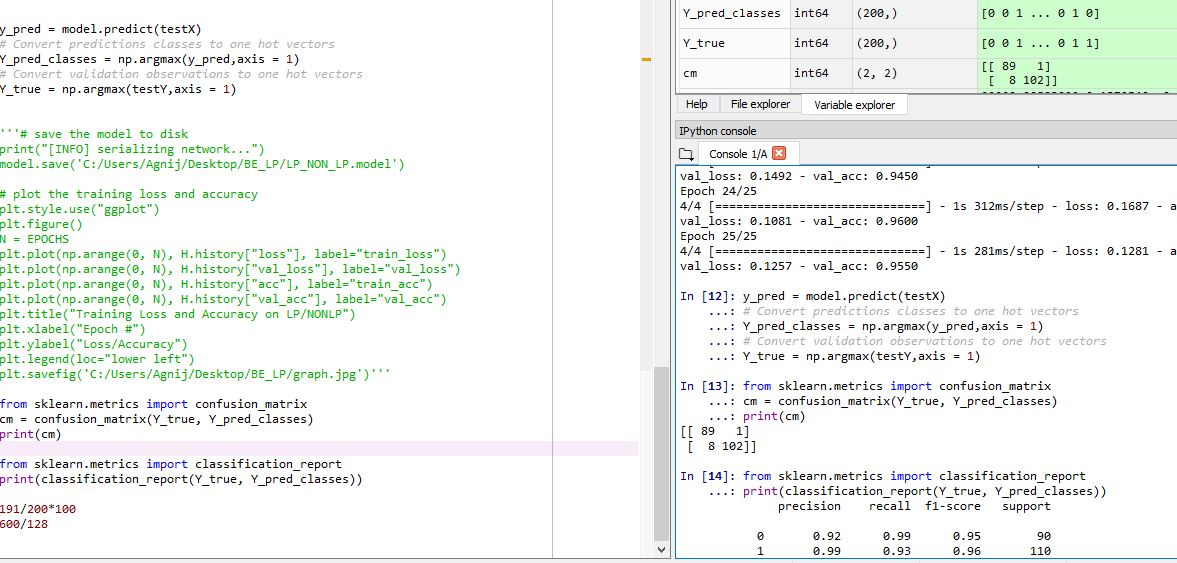
Large values give a learning process that converges slowly with accurate estimates of the error gradient.

**Source: 1.** [**https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/**](https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/)

**2. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/**

**\*\*using a Batch size of 128**

**So a single batch will have 600/128 = 4.68 i.e. 4 batches of 128 samples each.**

****

**We get a total of 9 incorrect predictions.**

**Here although precision is 99% f1 score is only 0.96.**

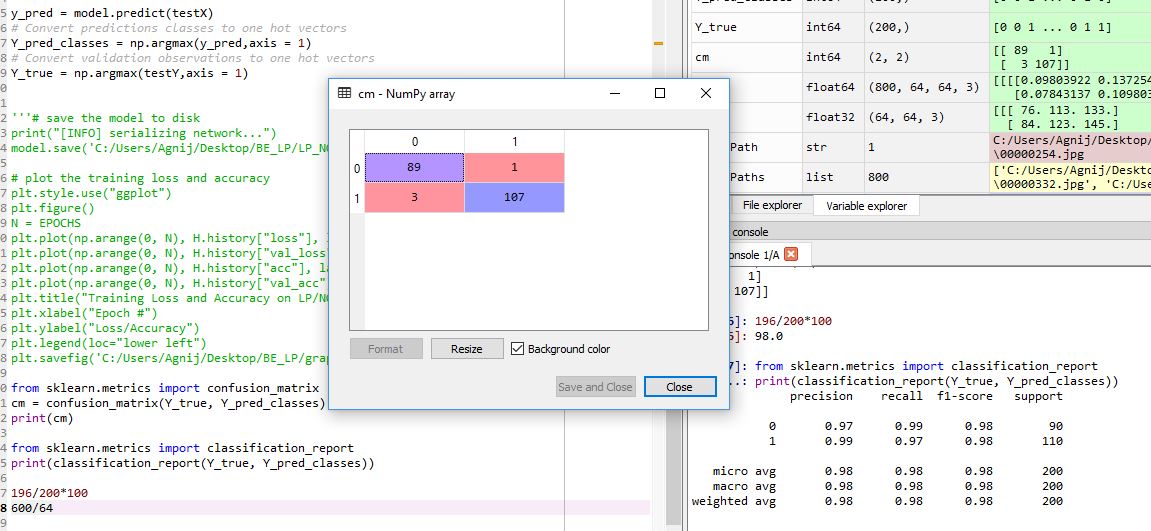
**\*\*using a Batch size of 64**

**So a single batch will have 600/64 = 9.375 i.e. 9 batches of 64 samples each.**

****

Training accuracy is 95.98% and val\_acc i.e. test accuracy is 98%

Using the Confusion matrix we get a test accuracy again the same as above i.e. 98%



TP: 107 images are truly predicted as LP’s, TN: 89 images are truly predicted as NONLP’s,

FP: 1 images are falsely predicted as LP’s, FN: 3 images are falsely predicted as NONLP’s.

**CONCLUSION**

**Selecting a Batch size of 64 gives us a better result as shown we get a precision of 99% which means only a single LP image was predicted falsely as a NONLP and also a f1 score of 0.98, therefore we go ahead and prefer a Batch Size of 64 for our project.**

**EPOCHS**

The number of epochs is the number of complete passes through the training dataset.

The number of epochs can be set to an integer value between one and infinity. You can run the algorithm for as long as you like and even stop it using other criteria besides a fixed number of epochs, such as a change (or lack of change) in model error over time.

It is an integer value and a hyper parameter for the learning algorithm, e.g. parameters for the learning process, not internal model parameters found by the learning process.

**[Source:** https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/**]**

**SO,**

We have a dataset with 600 samples (images) and chose a batch size of 64 and 25 epochs.

This means that the dataset will be divided into 9 batches, each with 64 samples. The model weights will be updated after each batch of 64 samples.

This also means that one epoch will involve 9 batches or 9 updates to the model.

With 25 epochs, the model will be exposed to or pass through the whole dataset 25 times. That is a total of 225 batches during the entire training process.

**Using data augmentation basically means that the number of images will remain the same i.e. 600/64 🡪 9 batches containing 64 images each, the only difference is that at each epoch a different version of the image will be applied to the CNN.**

**FINAL MODEL PARAMETERS:**

**\*\*Using Input Size as 64x64** **, for the first set of CONV => RELU => POOL layers we have selected --32-- feature maps of filter size --3x3-- and for the second set of CONV => RELU => POOL layers we have selected --64-- feature maps of filter size --3x3-- and for the first fully connected layer we have selected--128-- i.e. output of the dimensionality space.** [**BATCH SIZE = 64, EPOCHS = 25.]**