Satellite Image Classification with Deep Learning

The problem to solve is recognition in high-resolution multi-spectral satellite imagery. To achieve this, we are going to use CNN.

The data base will be IARPA Functional Map of the World (fMoW) that is a data set that gathers information from many telescopes and classify the images in 62 classifications plus false detection. Reference [25] with nominal 0.5 meter per pixel and with more spectral bands as near IR plus the metadata.

The architecture that is going to be used is CNN plus a fully connected network. This doesn’t require any algorithm for feature detection.

Complications:

C.1. One complication is that the satellite images are too big. Normally the images taken are small, otherwise it will take too long to process them. Example for ResNet and DenseNet: 224x224, Inception: 299x299. Normally the dataset (Images) are cropped to fit this size. Satellite images have thousands of pixels and the object to recognize also can have thousands of pixels then it is not useful to crop it.

C.2. Other complication is the orientation of the object, for example in an image of a person the head will be most of the time at the top and the feet at the bottom.

C.3. Another complication are the clouds.

C.4. Datasets.

Solutions:

The dataset that is going to be used (IARPA-fMoW) seems to be large and complete enough to cover C.4.

The metadata comes with the bounding box of the image, we can crop the images to such box instead of all the image, that will help to solve C.1. This bounding box can be a little bit enlarged to provide more context of the environment and it also can be adjusted to make it square.

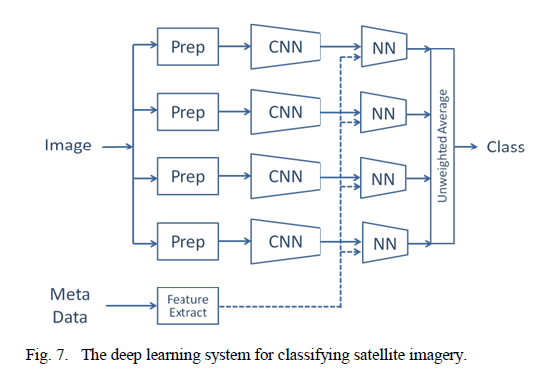
NIR spectrum trespass the clouds that can provide important information in case of C.3. The solution that they also give is to skip all the images with mire that 40% of cloud cover and box sizes smaller that 5 pixels.

For the sequences the same method is applied and an average value is taken with all the outputs.

The images of the dataset were multiplied by 1 flip and 3 rotations the images 90°,180° and 270°, this also helps to solve point C.2. One image is generating 7 more. We are analyzing static objects then image flipping and rotation is not affecting.

Methodology:

Keras is used to create the CNN plus the full connected network, the input is the image and the metadata, four CNNs different are used they are connected to a similar full connected, a mean values of the 63 classifications are taken form the 4 systems and the maximum value determines the classification.



The CNNs used are going to be DenseNet-161 [11], ResNet-152 [10], Inception-v3 [9] and Xception [27].

The input of the NN is the output of the CNN(63 values) plus 27 values from the metadata but they have to be processed and normalized between -1 and 1.

The NNs layers are 90 size input layer, 1024 size hidden layer with a dropout of 0.6 and a 63 size output layer, softmax is the activation function of the output layer.

Training

90% of false detections for train 10% for testing. The other ones 87% for training and 13% for testing.

The CNN were pretrained using models of the community. [29,30], or ImageNet [17-19,31,32].

They are using just one epoch. A second epoch can be executed with a reduced learning rate.

NN were trained for 20 epochs or until the validation loss stopped decreasing. The training can be separated [26].

DataSet

<https://github.com/fMoW/dataset>

DenseNet-161

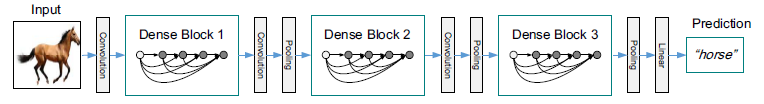
They alleviate the vanishing-gradient problem and reduce the number of parameters.

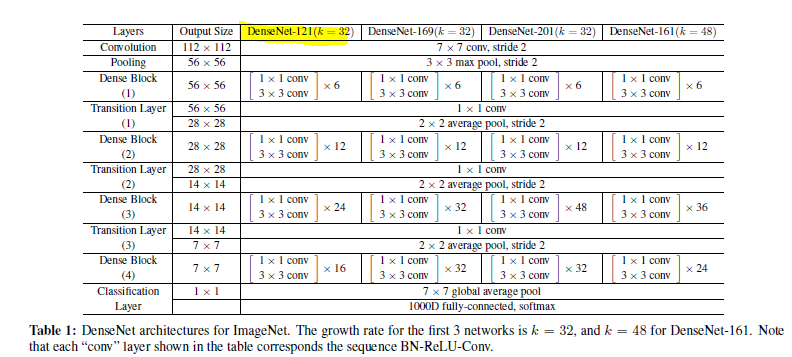
Each layer obtains additional inputs from all preceding layers. ResNets use a similar strategy and have shown that many layers contribute very little and can be randomly dropped without problems.

<https://github.com/liuzhuang13/DenseNet>

Compared to Inception networks, which also concatenate feature form different layer, DenseNets are simpler and more efficient.

Normally CNN the last layers have more smaller but more filters, that is expected. That is not a problem, but as proposed here we are going to concatenate all the previous outputs, logically this is not possible if all the filters are not of the same size. Then we apply the strategy mentioned but in blocks. All the filters in the same block have the same size. Although between block and block we have a batch normalization (BN) a 1x1 CNN of one layer and pooling (2x2 average) to reduce the size.





Where k is the number of filters(growth rate). Note that the input image for ImageNet is 224x224.

The filters at each block are a combination of BN-ReLU-Conv(1×1 X 32)-BN-ReLU-Conv(3×3 X 32) for DenseNet 121.

For all the DB excepting ImageNet the first Convolution is of size k=16 or k=2k. Convolutions 3x3 the padding=same. For ImageNet k=32.

The architecture for all de DB excepting imageNet was : {L = 40, k = 12}, {L = 100, k = 12} and {L = 100, k = 24}.

At the end of the last dense block, a global average pooling is performed and a softmax classifier is attached.

Training is with All the networks are trained using stochastic gradient descent (SGD). For imageNet 90 epochs, Initial lr=0.1, 0.01 after epoch 30 and 0.001 after epoch 60.

Inception-v3

Designed to perform well even under strict constraints on memory and computational budget.

Design Principles

Principles:

1) The representation size should decrease from the inputs to the outputs.

2) Highly dimensional layers are easier to train in tiles.

3) Reduce the dimensions promotes faster learning.

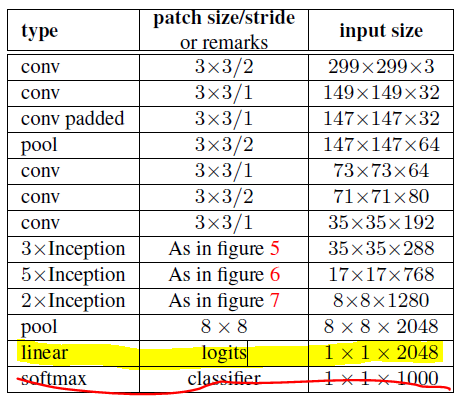
4) The shall be a balance between the filters per layer and the number of layer, in general the best solution is to increase them in parallel, but the computational power has a limit.

There is a principle of factorization with the deep learning, for example we can reduce one layer 5x5 with two layers 3x3. With this we can reduce from 25 parameters to 2(9) = 18 parameters. We can even factorize in asymmetric filters for example one layer filter 3x3 can be factorized in one layer filters 3x1 and after that one layer filter 1x3. In general we can factorize any nxn layer into a 1xn convolutional followed by a convolutional nx1. It seems not to be good in early layer but seem to have good results with inputs between 12 and 20. Very good results with 7x1 and 1x7.

Following the principle 1) the goal is to reduce the parameters that can be achieved by applying convolution and then pooling but the results have to be concatenated.

Inception-v3 is an improvement of the architecture ILSVRC2012.

The architecture is the following.



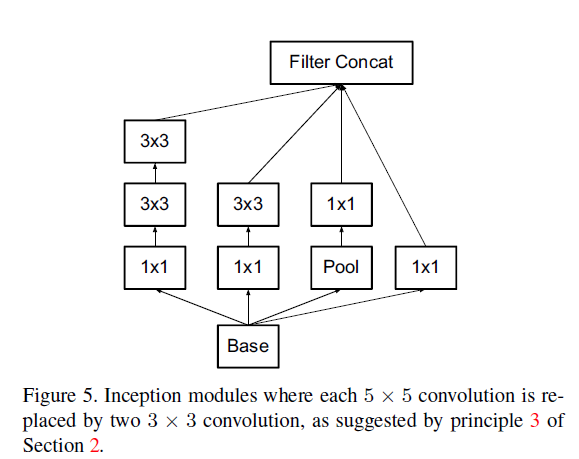
The layer that has padded means that padding=same in keras/Tensorflow. The last layer is for classification and we are going to execute it in a different way. The one highlighted will be executed but using just our 63 lavels that we have and this in how we are going to have our 63 outputs.

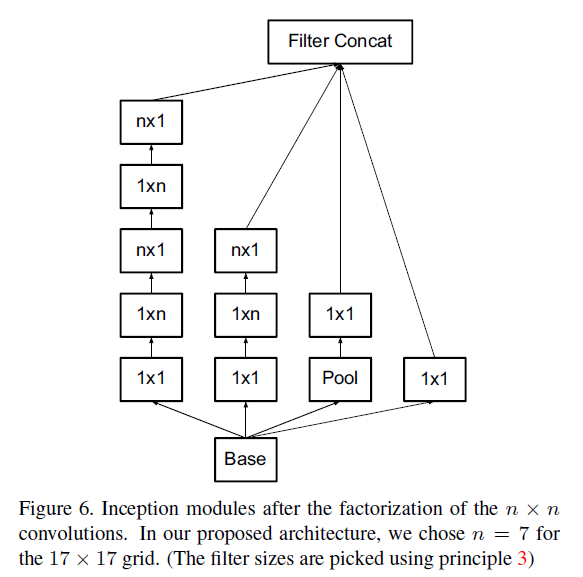
Linear layer will execute the following operation:

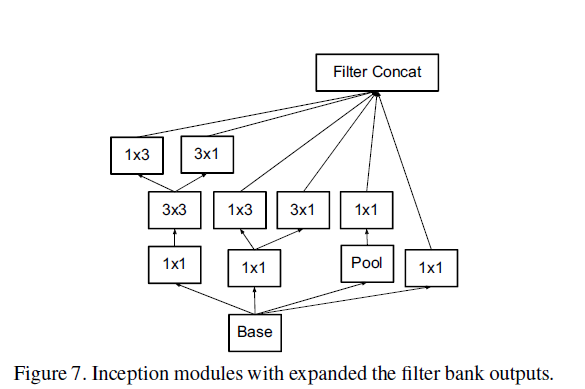


Where k is the number of labels and i the number of inputs.

The inception layers are as the ones showed below:







All the layers are 3x3, 3x1 or 1x3 otherwise the filters will not match with the architecture description and inside the inceptions the stride is 1 an padding same excepting the last layer of every inception were the padding is 0 and stride of 2. The concatenation represents that all the output of every path are going to be placed one after the other.

