OnionNet: Sharing Features in Cascaded Deep Classifiers

Martin Simonovsky
martin.simonovsky@enpc.fr
Nikos Komodakis
nikos.komodakis@enpc.fr

Imagine Lab
Université Paris Est / École des Ponts
Paris, France

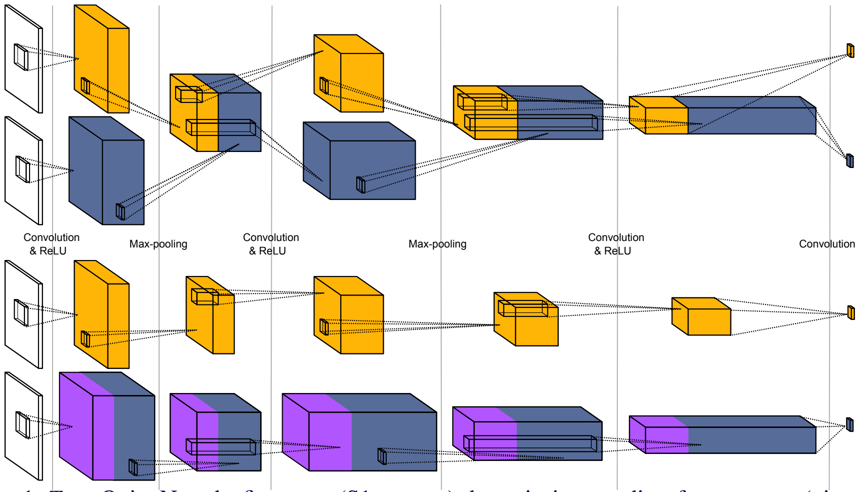

Figure 1: *Top* - OnionNet: the first stage (S1, orange) shares its intermediate feature maps (visualized as cubes) with the second stage (S2, blue). *Bottom* - A traditional cascade: stages are independent and S2 has to be evaluated fully, recomputing certain features (purple).

The focus of our work is speeding up evaluation of deep neural networks in retrieval scenarios. A popular approach to reduce time spent on negative examples is to set up a cascade of classifiers of increasing strength, called stages. As these are trained for the same or a similar objective, the question is how much their features (should) have in common. Without any sharing, a representation presumably at least as powerful as in the previous stage has to be rebuilt in the following one.

We address this by proposing OnionNet, a novel feature-sharing cascaded architecture where the next stage extends the feature map set of the previous stage, preventing repeated computation. Crucially, the architecture is flexible: the next stage may add both new layers as well as new feature channels. We construct our cascades by gradually increasing the width, possibly in addition to depth. This is beneficial as the lowest layers tend to be the most expensive ones to compute while producing weak classifiers on their own.

Figure 1 illustrates a two-stage OnionNet cascade consisting of two branches with the same layer organization. Each takes the same input and is terminated by its own output layer. The core idea is that the branches are linked before every convolutional layer. The feature maps of the first stage (S1) are used as additional input to the following convolutional layer in the second stage (S2) but not the other way round, creating a one-way dependence. The model is trained end-to-end under a joint loss, which makes the cascade learn the proper allocation of features between the stages.

OnionNet is applied to three important tasks: patch matching, proposal-based object detection, and image retrieval. We demonstrate good speed-ups due to cascades and show that OnionNet sharing can bring further gain atop of it, with only a marginal decrease in precision. Specifically, we achieve 2.8x, 2.9x, and 1.7x running time reduction in each respective application. Furthermore, we provide a systematic study in theory and on a synthetic benchmark that sheds further light into the time cost behavior of cascaded architectures.