A survey on Transfer Learning techniques for Traffic Sign Classification

Manuel Scurti & Mingjie Ye

Traffic Sign Recognition Task

The german traffic sign benchmark (GTSRB) is a multi-class classification challenge. The benchmark has the following properties:

- Multi-class classification problem
- More than 40 classes
- More than 50000 images in total

Previously provided solutions

- Our previous work focused more on the network architecture than in the training process itself.
- Training process is a big problem for this task. because the amount of data is not enough to feed a very deep neural network and train it generalize well for new examples of traffic signs.
- ImageNet contains more than 14 million images while our dataset contains only around 50000 traffic signs images. That is a huge difference

Net structure	Accuracy	Time	Key feature of the net
Fully Connected	0.8781	0.053	Better preprocessing
CNN	0.9584	0.1735	Convolutional Layers

How to improve?

Transfer Learning solution (1 / 2)

- When dealing with few examples to learn from, Transfer Learning makes possible to re-use a pretrained neural network, like one of them trained on ImageNet, in order to use it with significantly different problem with only minor changes
- Why is it working? Deep learning models are by nature highly repurposable because they learn the underlying structure of our data. They learn to see only the relevant features in our data

Transfer Learning solution (2 / 2)

- We based our experiments on 5 pretrained, well known, architectures:
  - DenseNet201
  - VGG19
  - InceptionV3
  - InceptionResNetV2
  - NASNetMobile
- What we expect from the experiments:
  - Fast network (since we are using a pretrained network)
  - Awesome accuracy

Experiments - Attempt #1

The general process followed in the first attempt is:

1. Take a pretrained model without top fully connected layers
2. Attach the top of the model a different fully connected network structure of our choice, with the 43 neurons output layer
3. Set ONLY the fully connected layers to be trainable/set at most the first convolutional part just behind the top layers
4. Train the whole model and see the results

Result: Training the model took 9 - 20 seconds per epoch and accuracy was stucked in different attempts between 0.06 and 0.70. FAIL

Experiments - Attempt #2

1. Take a pretrained model without top fully connected layers
2. Attach the output layer of our classification task
3. Train the model

Result: accuracy 0.9722, time 2.2628 using InceptionV3. Nice attempt but still too slow and thus still a FAIL

Experiments - Attempt #3

1. Take a pretrained model without fully connected layers (this time VGG19)
2. Use it as a bottleneck features extractor. i.e. transform our dataset into a newer one containing features found at the output of the last conv layer
3. Train a fully connected model from scratch using this dataset
4. Merge the trained fully connected model and the baseline model, freezing all conv layers except the last one, and train it using very small weight updates (i.e. Fine-tune)

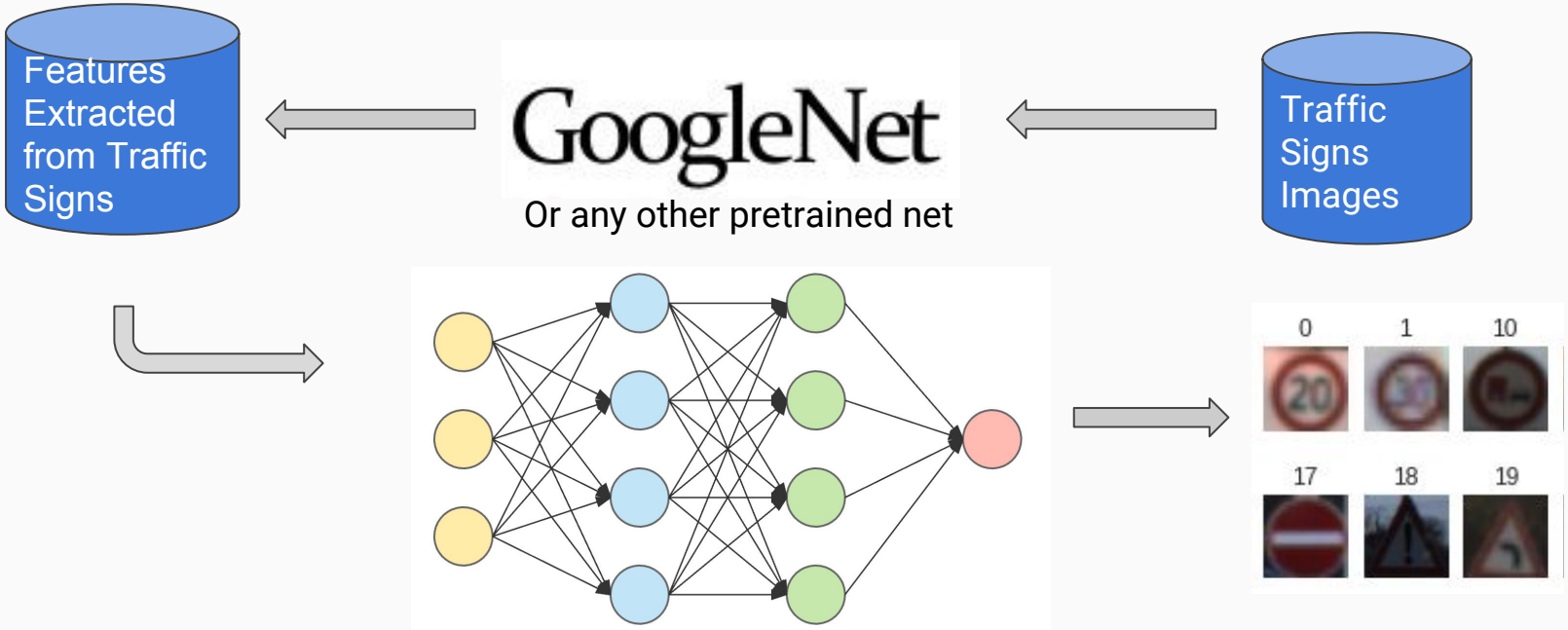
Result: accuracy 0.85 time 0.40 seconds. Still not what we expect FAIL

Note on the experiments

- All the training processes were conducted with data augmentation for big models (conv layers + fully connected). and classic training for fully connected models
- What emerges from the results is that the best accuracy we got was using only the pretrained model, while the best performance obtained was when we transformed our dataset using a feature extractor

Idea: let’s cache the results of the feature extractor and use them as input for completely new fully connected model

Final solution proposal (1 / 2)



Final solution proposal (2 / 2)

1. Take a pretrained model without fully connected layers
2. Train the pretrained model using traffic signs images for few epochs
3. Use it as a bottleneck features extractor
4. Train for few epochs a fully connected model from scratch using this dataset

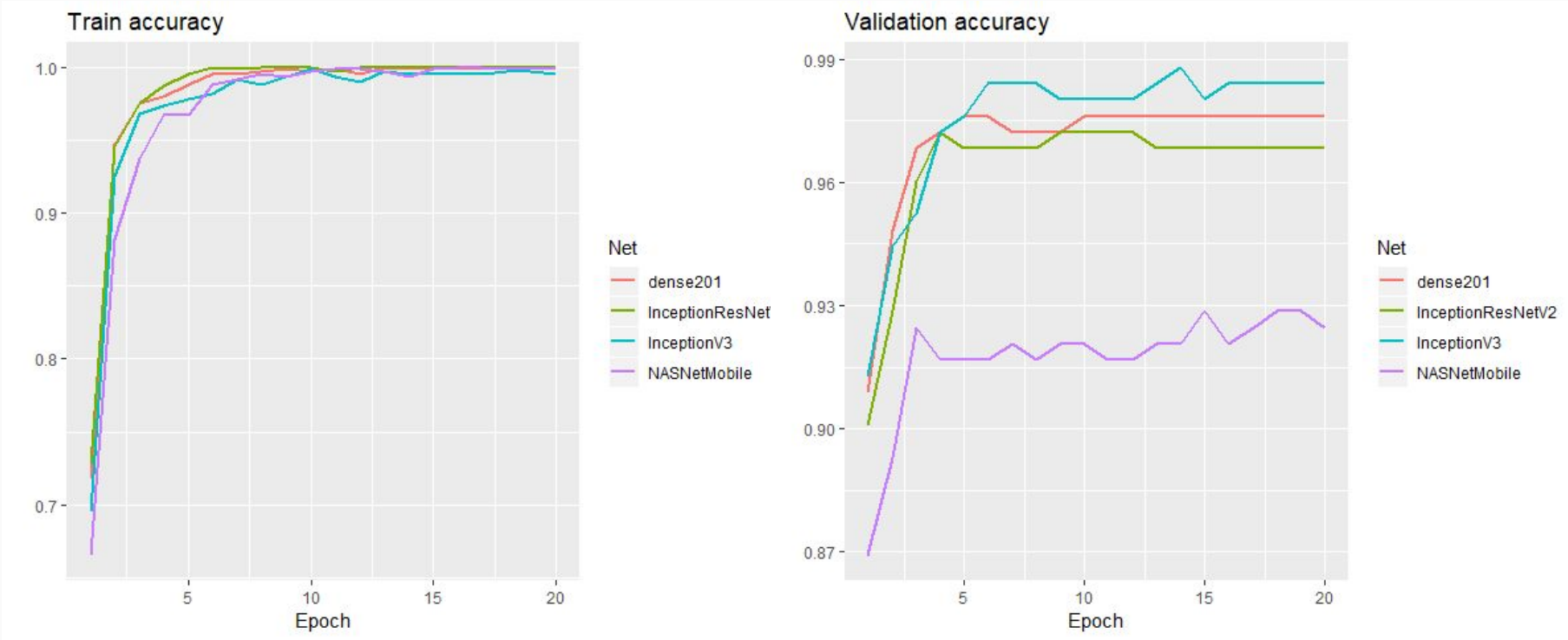
Final results (1 / 3)

- Faster network performance than first attempts (+13% faster)
- Best accuracy so far (+4% better)

Bottleneck feature extractor	Training parameters of the extractor	Training parameters for fnn	Accuracy	Time
InceptionV3	Data aug, epochs 60	epochs 20	0.9833	0.2174
Dense201	Data aug, epochs 60	epochs 20	0.9889	0.4528
InceptionResNetV2	Data aug, epochs 40	epochs 20	0.9861	0.2952
NASNetMobile	Data aug, epochs 40	epochs 20	0.9667	0.3020
VGG19	Data aug, epochs 60	epochs 20	0.0747	0.1560

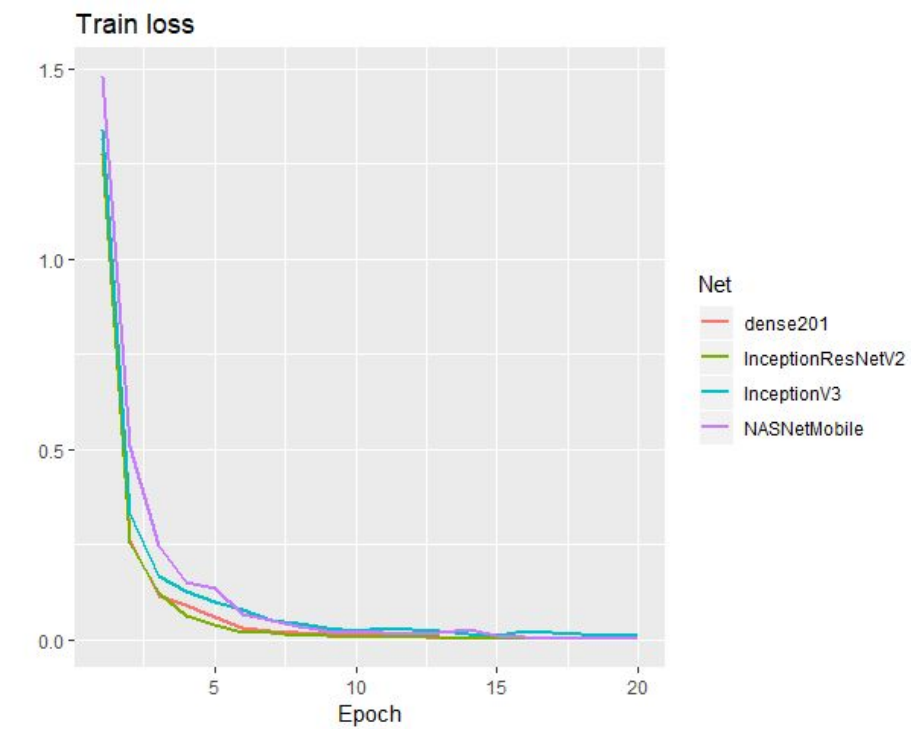
Final results (2 / 3)

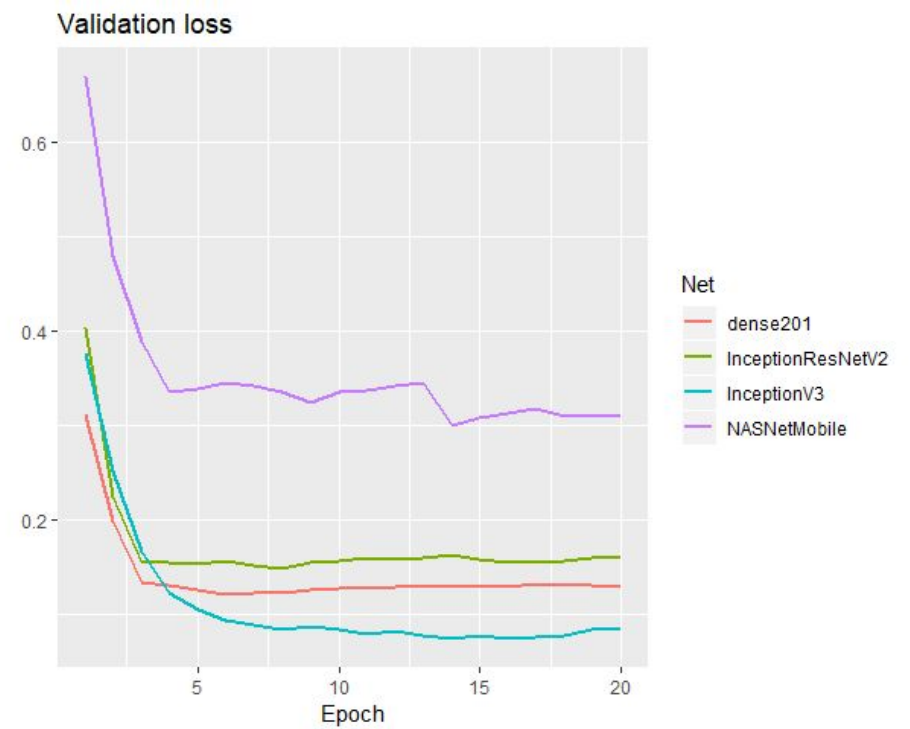
Note: VGG19 is excluded from comparisons



Final results (3 / 3)

Note: VGG19 is excluded from comparisons





Bibliography

- VGG19 - Very Deep Convolutional Networks for Large-Scale Image Recognition
- InceptionV3 - Rethinking the Inception Architecture for Computer Vision
- InceptionResNetV2 - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning
- DenseNet - Densely Connected Convolutional Networks
- NasNetMobile - Learning Transferable Architectures for Scalable Image Recognition
- Keras pretrained networks - https://keras.io/applications/

Thank you!

Questions?