IML tutorial 8

Joanna Ficek

8.04.2020

Outline

1. **Technical considerations on data preprocessing (Project 2)**
    a. High-level approach
    b. Missing data handling

2. **Convolutional Neural Networks**
    a. Recap
    b. Example exam questions
    c. Transfer learning (demo)
    d. Other applications

Project 2 data excerpt

	A	B	C	D	E	F	G	H	I	J
1	pid	Time	Age	EtCO2	PTT	BUN	Lactate	Temp	Hgb	HCO3
2	1	3	34	nan	nan	12	nan	36	8.7	24
3	1	4	34	nan	nan	nan	nan	36	nan	nan
4	1	5	34	nan	nan	nan	nan	36	nan	nan
5	1	6	34	nan	nan	nan	nan	37	nan	nan
6	1	7	34	nan	nan	nan	nan	nan	nan	nan
7	1	8	34	nan	nan	nan	nan	37	nan	nan
8	1	9	34	nan	nan	nan	nan	37	nan	nan
9	1	10	34	nan	nan	nan	nan	37	nan	nan
10	1	11	34	nan	nan	12	nan	nan	8.5	26
11	1	12	34	nan	nan	12	nan	38	8.5	26
12	1	13	34	nan	nan	nan	nan	nan	nan	nan
13	1	14	34	nan	nan	nan	nan	nan	nan	nan
14	100	2	68	nan	nan	nan	nan	35	nan	nan
15	100	3	68	nan	20.9	21	nan	nan	12.5	27
16	100	4	68	nan	nan	21	nan	36	12.5	27

Project 2: High-level approach

Data loading

Data can be first loaded for all patients using <pd.read_csv(“train_features.csv”)>, but should be processed patient-by-patient for imputation and feature generation, and then assembled in training matrices passed into Machine Learning Models.

Imputation

For a given patient, if some of your methods require fully-observed data (like a dense 12-hour time-series), precompute imputed time series or partially imputed time series (in case they can accept some missing values).

Feature generation

For a given patient, load the imputed data or real data (or a mixture thereof) and compute the feature of your choice, that can capture the most relevant information across patient’s measurements.

Code snippet: data loading

```
# load data and split patients into train and validation sets
X_features = pd.read_csv("train_features.csv")
all_pids = extract_unique_pids(X_features)
train_pids, val_pids = partition(all_pids)
# extract population statistics (if used for imputation)
train_stats = population_statistics(get_patient_data(X_features,train_pids))
val_stats = population_statistics(get_patient_data(X_features,val_pids))
X_all=[]
```

Code snippet: data preprocessing

```
for pid in train_pids:
    # Get training data for a patient (Dimension 12 x d)
    X_pid = get_patient_data(X_features, pid)

    # Impute variable from (possibly other) variables using imputation strategy of choice,
    # the imputed time series needs to contain at least one non-missing value.
    imputed_time_series={}
    for var in VARIABLES:
        imputed_time_series[var,imp_strategy]=impute_variable(X_pid,var,imp_strategy,trains_stats)

    # Construct features of your choice, using one variable or several.
    feat_row=[]
    for feat_desc in FEAT_DESCS:
        needed_inputs,function = feat_desc
        args=[]
        for var,needs_imputed,strategy in needed_inputs:
            if needs_imputed:
                input=imputed_time_series[var,strategy]
            else:
                input=get_raw_time_series(X_pid,var)
            # Input could be multivariate, like 12 time_steps, needs
            # to contain at least one non-missing value.
            args.append(input)

        feature=function(args)
        feat_row.append(feature)

    X_all.append(feat_row)

X_train = np.concatenate(X_all,axis=0)
```

Code snippet: further steps

```
for pid in val_pids:
    ... # Construct the features for the validation set -> X_val

# Compute the same features on the test set... -> X_test

# Load labels from the train/val sets, train a Machine Learning model of your choice, and
# create predictions on the test set.
```

Missing values: mechanisms of missingness

- Missing Completely At Random (MCAR)
- Missing At Random (MAR)
- Not Missing At Random (NMAR)

Missing values: mechanisms of missingness

- Missing Completely At Random (MCAR)
  - Mechanism of missing data is unrelated to the both the observed and the unobserved data.
  - Example: A person drops from a study on reading speed due to moving to another city.

- Missing At Random (MAR)
  - Mechanism of missingness depends only on the observed data.
  - Example: A person drops from a study on reading speed due to lack of time to practice and the number of practice hours is also recorded.

- Not Missing At Random (NMAR)
  - Mechanism of missingness is related to the observed and unobserved values.
  - Example: A participant in a study on blood pressure skips a regular check-up because he is feeling bad (possibly due to a too high blood pressure).

Missing values: how to proceed?

- Discard observations with any missing values (complete case analysis).
- Rely on the learning algorithm to deal with missing values in its training phase.
- Impute all missing values before training.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2001

Missing values: how to proceed?

- Discard observations with any missing values (complete case analysis).
  - Useful only if small fraction of missing values, otherwise information loss.
  - Can bias feature space.
- Rely on the learning algorithm to deal with missing values in its training phase.
  - Some algorithms allow missing observations, e.g. CART constructs “surrogate splits”.
  - In most cases not feasible.
- Impute all missing values before training.
  - Avoids information loss.
  - Increases uncertainty in estimates and predictions.
  - Introduces bias if MCAR mechanism not in place.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2001

Imputation: a different perspective

- Traffic jam data from São Paulo. link to data download
- Used in the context of air pollution.
- Available columns:
  - passage (str) - Name of the passage
  - direction (str)
  - type (str) - Indicates if the passage is an expressway (E)
  - region (str) - São Paulo region
  - timestamp (datetime) - When the traffic jam was measured (UTC-4)
  - jam_size (int) - Traffic jam in meters
  - segment (str) - Where the passage is located

Source: https://en.wikipedia.org/wiki/Transport_in_S%C3%A3o_Paulo

Imputation: a different perspective

	timestamp	type	jam_size
0	2011-09-02 19:00:00	E	3850
1	2015-07-21 11:30:00	E	3860
2	2015-07-21 12:00:00	E	3860
3	2017-03-15 07:30:00	A	3900
4	2017-03-16 07:00:00	A	3900
5	2017-03-16 07:30:00	A	3900
6	2017-03-21 09:00:00	A	3900
7	2017-03-21 09:30:00	A	3900
8	2017-08-07 10:30:00	A	3900
9	2017-08-07 11:00:00	A	3900
10	2017-08-07 11:30:00	A	3900
11	2017-08-07 12:00:00	A	3900
12	2017-08-09 13:30:00	A	3900
13	2017-08-09 14:00:00	A	3900
14	2018-05-25 09:30:00	A	3810
15	2018-05-25 10:00:00	A	3810
16	2018-06-14 12:00:00	E	3858
17	2018-06-14 12:30:00	E	3858
18	2018-06-20 10:00:00	E	NaN

Data: Traffic jam data from a passage segment in São Paulo.

Task: Impute *jam_size* (to use it for prediction of air pollution).

Can we reformulate the task in the context of prediction?

Task: Predict the *jam_size* for the next timestamp *2018-06-20*.

Imputation: a different perspective

	timestamp	type	jam_size
0	2011-09-02 19:00:00	E	3850
1	2015-07-21 11:30:00	E	3860
2	2015-07-21 12:00:00	E	3860
3	2017-03-15 07:30:00	A	3900
4	2017-03-16 07:00:00	A	3900
5	2017-03-16 07:30:00	A	3900
6	2017-03-21 09:00:00	A	3900
7	2017-03-21 09:30:00	A	3900
8	2017-08-07 10:30:00	A	3900
9	2017-08-07 11:00:00	A	NaN
10	NaN	A	3900
11	2017-08-07 12:00:00	NaN	3900
12	2017-08-09 13:30:00	A	3900
13	2017-08-09 14:00:00	A	3900
14	2018-05-25 09:30:00	A	3810
15	2018-05-25 10:00:00	A	3810
16	2018-06-14 12:00:00	E	3858
17	2018-06-14 12:30:00	E	3858
18	2018-06-20 10:00:00	E	NaN

Data: Traffic jam data from a passage segment in São Paulo.

Task: Impute traffic jam data (predictors' values).

Can we reformulate the task in the context of prediction?

Task: Predict all the missing values.

Imputation: using observed values of the variable

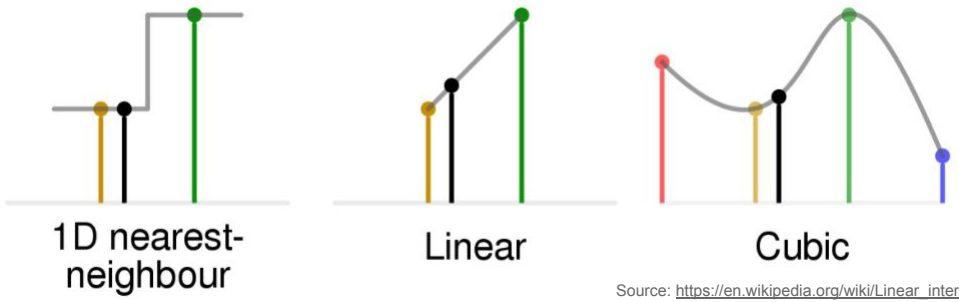
	timestamp	type	jam_size
8	2017-08-07 10:30:00	A	3900
9	2017-08-07 11:00:00	A	NaN
10	NaN	A	3900
11	2017-08-07 12:00:00	NaN	3900
12	2017-08-09 13:30:00	A	3900
13	2017-08-09 14:00:00	A	3900

Dataframe with predictors' values.

- Using **summary statistics** (mean/median/mode)

```
median_jam_size = df['jam_size'].median()
df_imputed  =  df.fillna(value = {'jam_size': median_jam_size})
```

- **Substitute** with another observed value (neighboring or chosen at random)
- **Interpolation** (linear, splines etc.)



Source: https://en.wikipedia.org/wiki/Linear_interpolation

Imputation: based on other variables

	timestamp	type	jam_size
8	2017-08-07 10:30:00	A	3900
9	2017-08-07 11:00:00	A	NaN
10	NaN	A	3900
11	2017-08-07 12:00:00	NaN	3900
12	2017-08-09 13:30:00	A	3900
13	2017-08-09 14:00:00	A	3900

Dataframe with predictors' values.

jam_size ~ timestamp + … + type

- **Regression** imputation
  - Build a predictive model with the variable including missing values as a dependent variable and use remaining variables as predictors
  - **Stochastic regression**: add random noise
  - **Bayesian regression**: EM + draw parameters from posterior distribution to generate missing values
  - Useful when dependency between features.

- **Hot-deck** imputation/matching imputation
  - Find cases with a highly similar profile in other variables and use their value/summary of values to replace the missing value.
  - How to find similar cases? E.g. using kNN

Imputation: very sparse matrix case

	timestamp	type	jam_size
0	2011-09-02 19:00:00	E	3850
1	2015-07-21 11:30:00	NaN	3860
2	2015-07-21 12:00:00	E	NaN
3	2017-03-15 07:30:00	A	NaN
4	2017-03-16 07:00:00	A	NaN
5	2017-03-16 07:30:00	NaN	3900
6	2017-03-21 09:00:00	A	NaN
7	2017-03-21 09:30:00	NaN	3900
8	2017-08-07 10:30:00	NaN	NaN
9	2017-08-07 11:00:00	A	NaN
10	NaN	A	3900
11	2017-08-07 12:00:00	NaN	3900
12	2017-08-09 13:30:00	NaN	NaN
13	2017-08-09 14:00:00	A	NaN
14	2018-05-25 09:30:00	A	3810
15	2018-05-25 10:00:00	A	NaN
16	2018-06-14 12:00:00	NaN	3858
17	2018-06-14 12:30:00	E	3858
18	2018-06-20 10:00:00	E	NaN

Data: Traffic jam data from a passage segment in São Paulo.

Task: Impute traffic jam data (predictors' values).

Can we reformat the matrix to **extract crucial information**?

Yes, keeping all the data in a "raw format" for a downstream analysis is not always necessary.

What would be the best way to summarize the sparse information provided by the given variable?
Which part of the data can be summarized in another feature and which should be kept in a "raw format"?

Imputation: practical considerations

- Imputation allows for using all the data at hand (not only complete cases), but may bias the results.
- Imputation increases uncertainty in estimates and predictions.
  => Multiple Imputation (repeat analysis for each imputed dataset)
- When imputing keep in mind bounds of the variable values.
- The choice of missing data handling method is context-specific and it's important to keep in mind the tradeoffs of different solutions.

Note:

- Several advanced methods exist, but are beyond the scope of this lecture.
- Some considerations regarding missing data in the context of generative modeling will be presented later in the lecture (not needed for Project 2!).

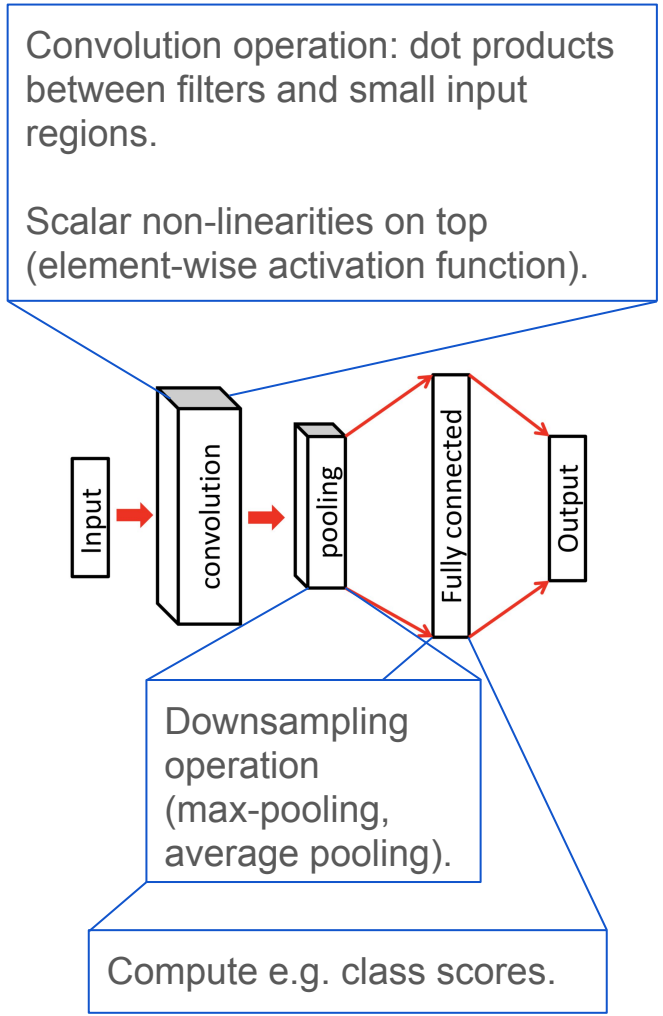
Other questions regarding Project 2?

=> please post on Piazza!

CNN: recap

- ANNs designed for specialized high-dimensional data (images, time-series)

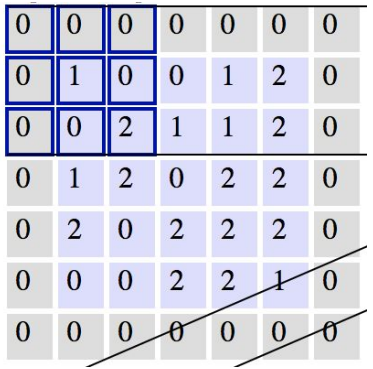
- Key ideas:
  - Robustness against small transformations of the input (exploitation of invariances)
  - Reduction of the number of parameters by introducing weight sharing
    => more scalable
    => less prone to overfitting



CNN: output dimensions

The output dimensions of each convolutional layer
are defined by the input size (N) and the hyperparameters:

- M: Number of filters used
- F: Size of filters used
- S: Stride (how many positions we move the filter at a time)
- P: Padding (pad input with zeros, often to preserve the input's spatial size)



From A. Karpathy example showed in the lecture (link)

$$L = \left\lfloor \frac{N - F + 2P}{S} + 1 \right\rfloor$$

For this to happen, with S=1, P=?

$$P = \frac{F - 1}{2}$$

Output dim: L×L×M

Source: http://cs231n.github.io/convolutional-networks/

CNN: dimensions example

$$L = \frac{N - F + 2P}{S} + 1$$

Consider a convolutional layer. The input consists of 6 feature maps of size 21×21. The output consists of 8 feature maps, and the filters are of size 5×5. The convolution is done with a stride of 2 and zero-padding, so the output feature maps are of size…?

What would be the size of the feature maps after applying a max-pooling layer on top with F=2, S=2?

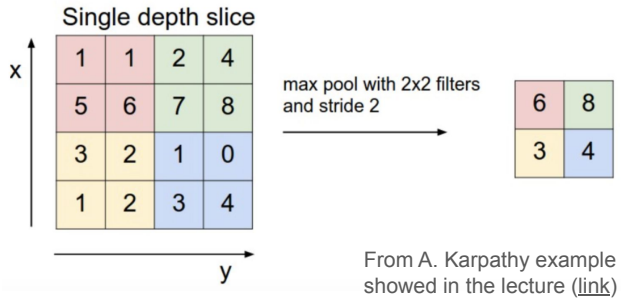
CNN: dimensions example (solution)

$$L = \frac{N - F + 2P}{S} + 1$$

Consider a convolutional layer. The input consists of 6 feature maps of size 21×21. The output consists of 8 feature maps, and the filters are of size 5×5. The convolution is done with a stride of 2 and zero-padding, so the output feature maps are of size…?

10×10 (the total output dimension is 10×10×8)

What would be the size of the feature maps after top with F=2, S=2?



on

5×5 (the total output dimension is 5×5×8)

From A. Karpathy example showed in the lecture (link)

CNN: #parameters example

Consider a convolutional layer. The input consists of 6 feature maps of size 21×21. The output consists of 8 feature maps, and the filters are of size 5×5. The convolution is done with a stride of 2 and zero padding, so the output feature maps are of size 10×10.

Number weights in convolutional layer:

Number of weights if fully connected layer (input/output dimensions unchanged):

CNN: #parameters example (solution)

Consider a convolutional layer. The input consists of 6 feature maps of size 21×21. The output consists of 8 feature maps, and the filters are of size 5×5. The convolution is done with a stride of 2 and zero padding, so the output feature maps are of size 10×10.

Number weights in convolutional layer:
There's one filter for each pair of an input and output feature map, and the filters are each 5×5. Therefore, the number of weights is 5×5×6×8 = 1200.

Number of weights if fully connected layer (input/output dimensions unchanged):
There are 21×21×6 units in the input layer and 10×10×8 units in the output layer, so the number of weights is 21×21×6×10×10×8 = 2,116,800.

CNN: characteristics example

Alice and Bob implemented two neural networks for recognizing handwritten digits from 16×16 grayscale images. Each network has a single hidden layer, and makes predictions using a softmax output layer with 10 units, one for each digit class.

- Alice's network is a convolutional net. The hidden layer consists of three 16×16 convolutional feature maps, each with filters of size 5×5, and uses the logistic nonlinearity. All of the hidden units are connected to all of the output units.
- Bob's network is a fully connected network with no weight sharing. The hidden layer consists of 768 logistic units (the same number of units as in Alice's convolutional layer).

Explain one advantage of Alice's approach and one advantage of Bob's approach.

Source: University of Toronto Computer Science

CNN: characteristics example (solution)

The inputs to the convolution layer are a linear function of the images. In Bob's network, the hidden units can compute any linear function of the images; by contrast, Alice's convolutional layer is more restricted because of weight sharing and local connectivity.

The advantage of Bob's network is that it is more powerful, i.e. it can compute any function Alice's network can compute, plus some additional functions.
Advantages of Alice's network include:

(a) It has fewer parameters, so it is less likely to overfit.
(b) It has fewer connections, so it requires fewer arithmetic operations to compute the activations or the weight gradients.

Source: University of Toronto Computer Science

Transfer learning: idea

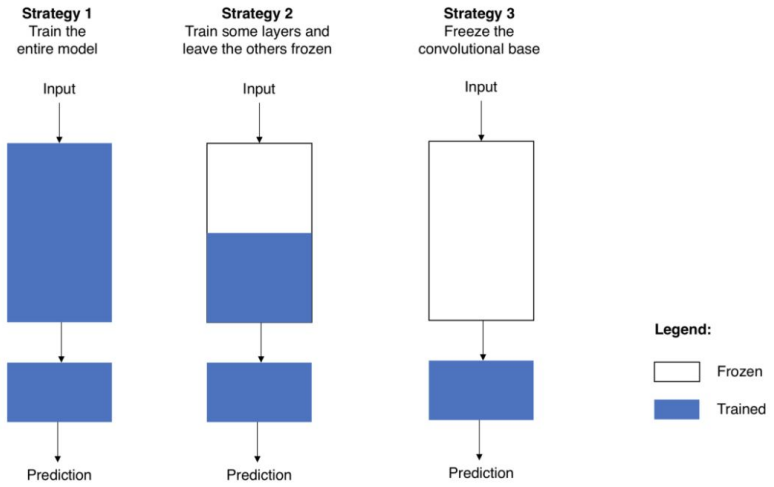
“In transfer learning, we first train a base network on a base dataset and task, and then we repurpose the learned features, or transfer them, to a second target network to be trained on a target dataset and task.”

Yosinki et al., 2014

- Motivation: Large amount of training data required, but not always available
- Idea: pre-train the network on some larger dataset from a related domain
- Caveat: training networks with a lot of parameters on a large dataset can be computationally very expensive
- **Solution: Use a hidden layer from a large pretrained model and fine-tune** the network weights on the dataset at hand (backpropagate errors)

Transfer learning: scenarios with CNNs

- (Train available architecture from scratch*)
- **Fine-tune** the CNN
  - Fine-tune all weights or (more commonly) keep initial layers fixed and fine-tune only later ones
  - Motivation: Features extracted earlier tend to be more generic and later ones more task-specific
- CNN as fixed **feature extractor**
  - Remove the last Fully Connected layer
  - Treat the extracted features as fixed and add a classifier on top

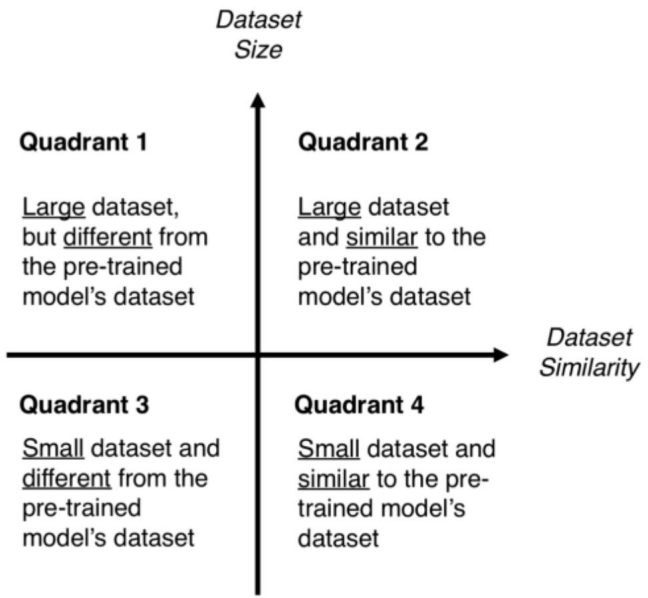


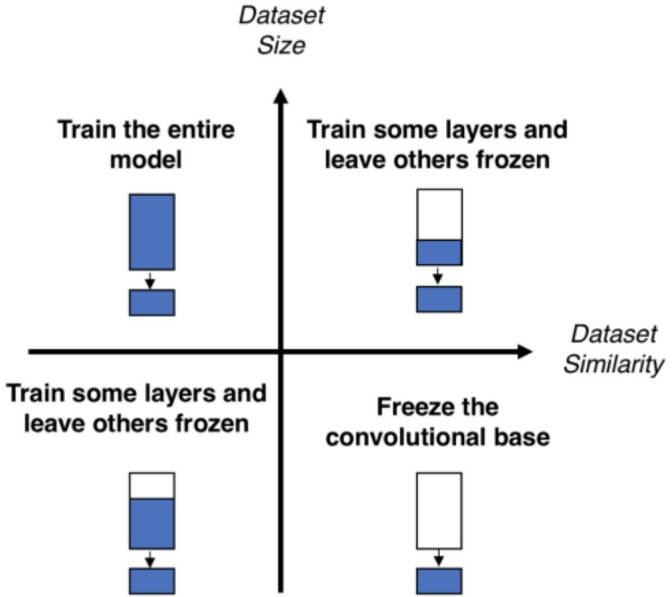
Source: towardsdatascience.com

*often the weights from a pre-trained model are still used for initialization

Transfer learning: guidelines

- Decisions should be based on **dataset size**, number of parameters and **similarity to the original dataset**





Source: towardsdatascience.com

Transfer learning: guidelines (2)

- Constraints from pretrained models
  - Less freedom in architectural choices.

- Learning rates
  - When fine-tuning use smaller learning rates.

    Why? We expect the weights to be “good”, so we don’t want to “distort them too quickly”.

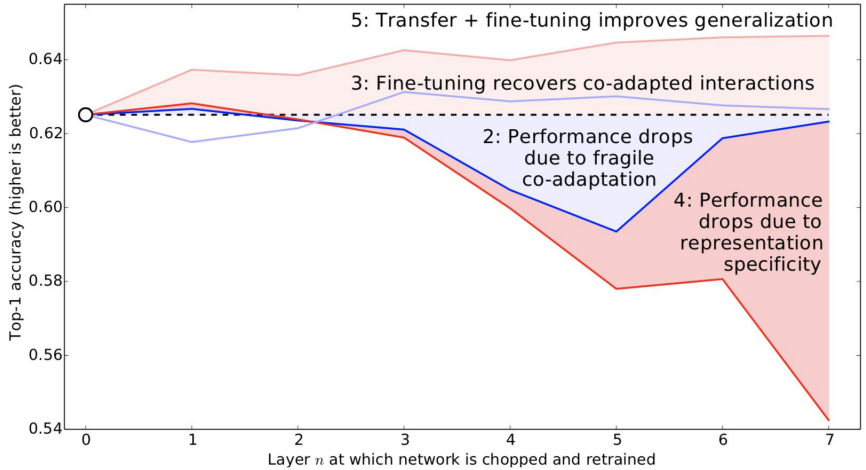
- Fine-tune after training the layers added on top (source)
  - First train the network without unfreezing the layers
  - Then fine-tune the weights

    Why? Otherwise too large gradient updates may cause the model to “forget” the knowledge.

Transfer learning: transferable features?

How transferable are features in deep neural networks?

Jason Yosinski,[1] **Jeff Clune,**[2] **Yoshua Bengio,**[3] **and Hod Lipson**[4]
[1] Dept. Computer Science, Cornell University
[2] Dept. Computer Science, University of Wyoming
[3] Dept. Computer Science & Operations Research, University of Montreal
[4] Dept. Mechanical & Aerospace Engineering, Cornell University



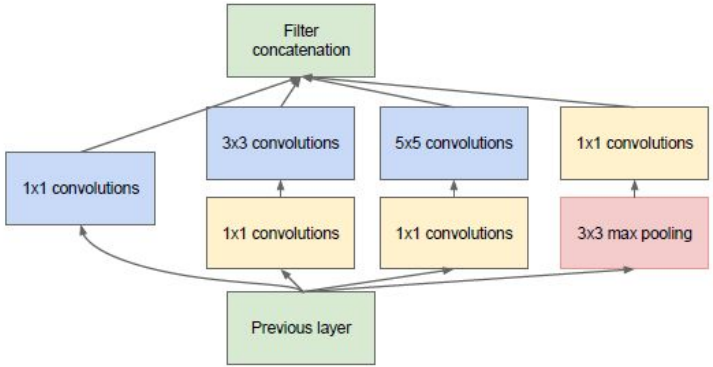
- Initializing a network with transferred features boosts generalization
- The effect persists even after substantial fine-tuning and holds for tasks with different training objectives
- If no fine-tuning, there can be some performance degradation (specificity of extracted features, optimization difficulties)

Leveraging pretrained models: demo

- Setting:
  - We train a dogs vs. cats classifier (on images)
  - We have only 3000 examples at hand (1500 per class)
  - We want to use a powerful Inception network to increase performance of our classificator pre-trained on ImageNet dataset (1.4 mln images with 1000 categories)

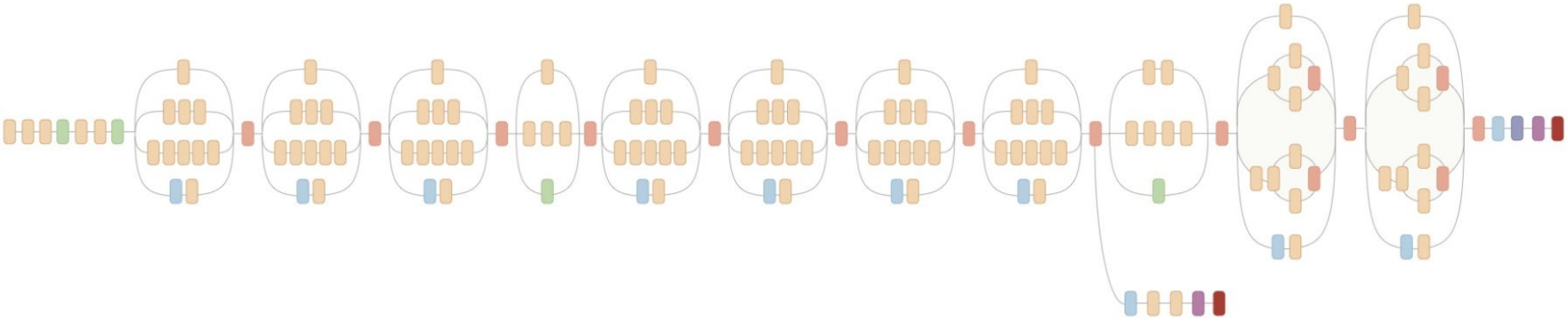
- Demo: https://developers.google.com/machine-learning/practica/image-classification/exercise-3

Inception network



Source: Szegedy et al., 2014

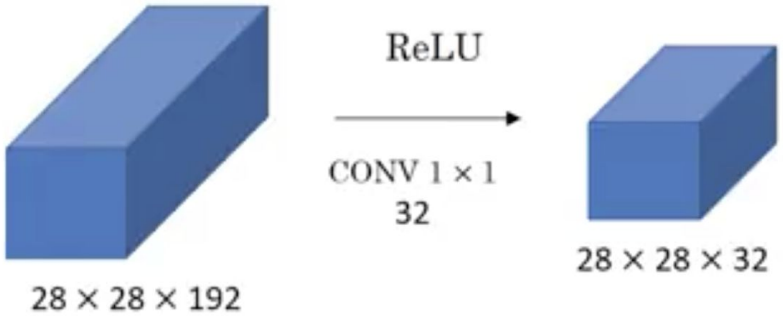
- Very deep network
- Concatenation of different filters
- 1x1 convolution used for dim. reduction and application of non-linearity



Convolution
AvgPool
MaxPool
Concat
Dropout
Fully connected
Softmax

Source: Szegedy et al., 2015 (github)

1×1 convolutions



- Dimensionality reduction
- Application of non-linearity
- Proposed in Network in Network

Source: tutorial by Andrew Ng

Pre-trained models

Documentation for individual models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

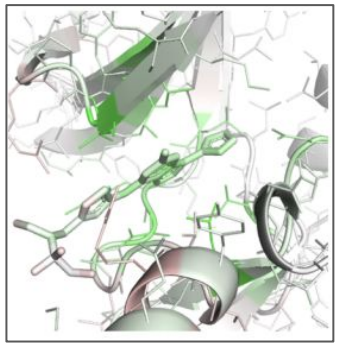
The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Great resource: https://keras.io/applications/

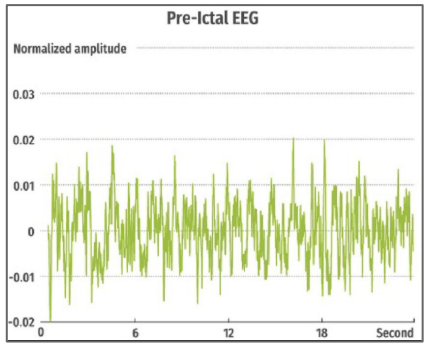
- Description of pre-trained models (image classification task, trained on ImageNet dataset)
- Easily accessible through Keras
- Examples how to use these models (and tweak the options) are available on the website

CNN: various applications

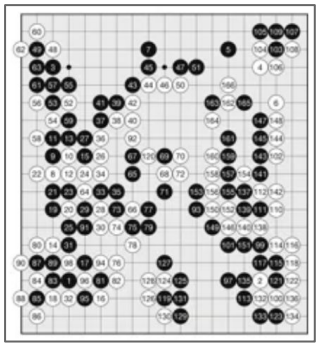
- Image recognition
- Natural language processing
- Anomaly detection
- Drug discovery (e.g. protein-ligand scoring)
- Feature extraction and classification of time-series data
- Computer games (used by AlphaGo)



Source: Ragoza et al., 2017



Source: Acharya et al., 2017



Source: Silver and Huang et al., 2016

References

- Data formatting and missing data handling
  - Pandas tutorial (esp. Section 9 on reshaping pandas data frames)
  - Little and Rubin (2002) link
  - Gelman A., Hill J. (2017) link
  - Best N., Manson A. (2012) slides on Bayesian framework for imputation
- Convolutional Neural Networks
  - Stanford tutorial
  - Goodfellow et al. (2016) (Deep Learning book)
- Transfer learning
  - Stanford tutorial
  - Pan and Yang (2009)
  - Yosinki et al. (2014)
  - Google ML Practicum
  - Blogpost

Thank you for your attention!