CSC 462: Machine Learning

8.1 Imbalanced Datasets

8.2 Combining Models

8.3 Training Neural Networks

8.6 Handling Multiple Outputs

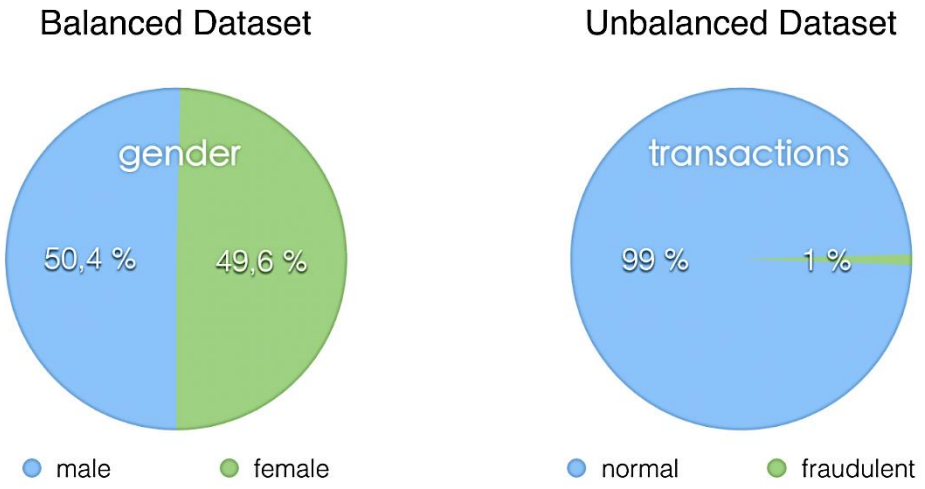
8.7 Transfer Learning

Dr. Sultan Alfarhood

8.1 Handling Imbalanced Datasets

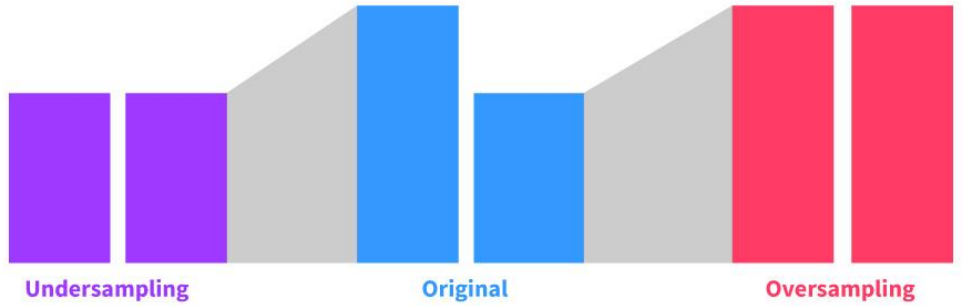
Handling Imbalanced Datasets

- Imbalanced data refers to those types of datasets where the target class has an uneven distribution of observations
  - i.e., one class label has a very high number of observations, and the other has a very low number of observations.



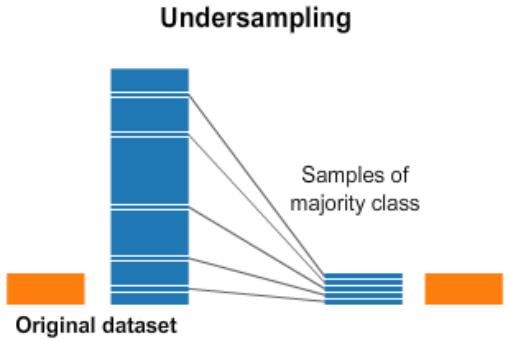
Handling Imbalanced Datasets

- Why this is a problem?
  - Provide a biased predictions
  - Misleading accuracy
- How to deal with it?
  - Undersampling
  - Oversampling
  - SMOTE / ADYSN

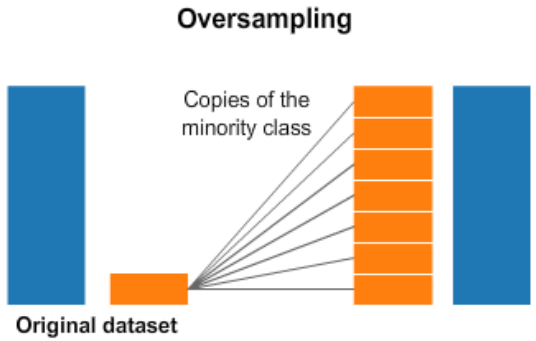


Undersampling and Oversampling

- Undersampling
  - Randomly remove majority class observations
  - May lead to bias
  - Removed observations could have important information

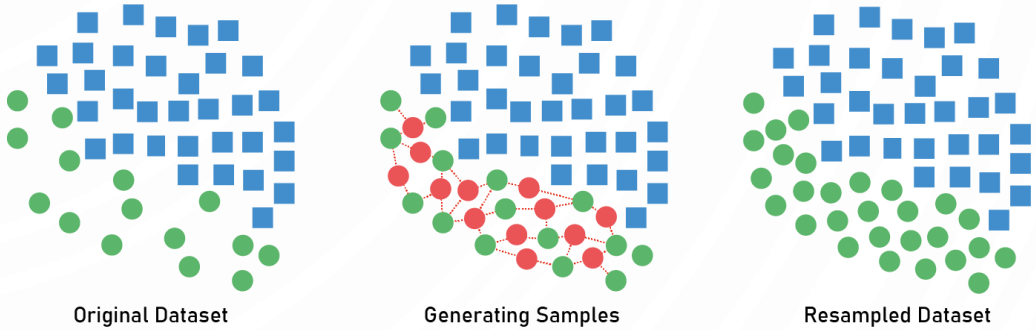


- Oversampling
  - Randomly add more minority observations
  - Prone to overfitting due to copying same information



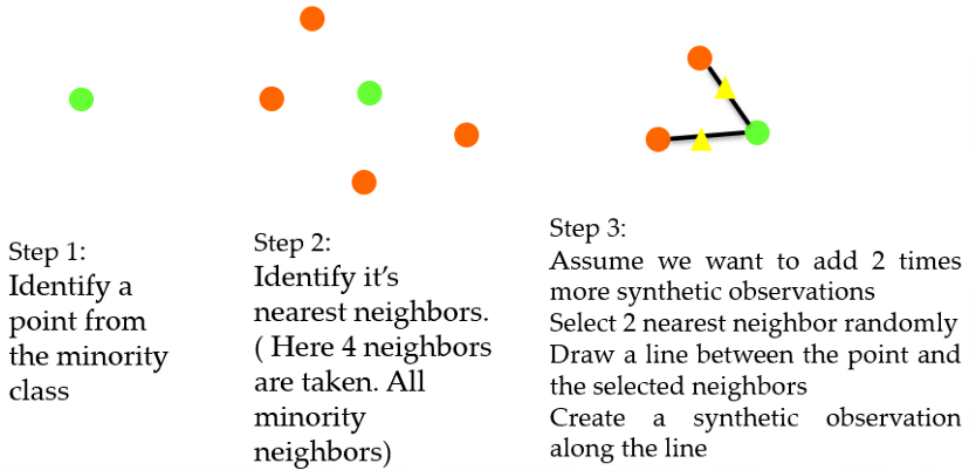
SMOTE and ADASYN

- Two popular algorithms that oversample the minority class by creating synthetic examples:
  - Synthetic minority oversampling technique (SMOTE)
  - Adaptive synthetic sampling method (ADASYN)



How SMOTE Works?

- SMOTE works by utilizing a k-nearest neighbor algorithm to create synthetic data.
  1. SMOTE first starts by **choosing random data** from the minority class
  2. Select **k-nearest** neighbors from the data
  3. Synthetic data would then be made between the random data and the randomly selected k-nearest neighbor

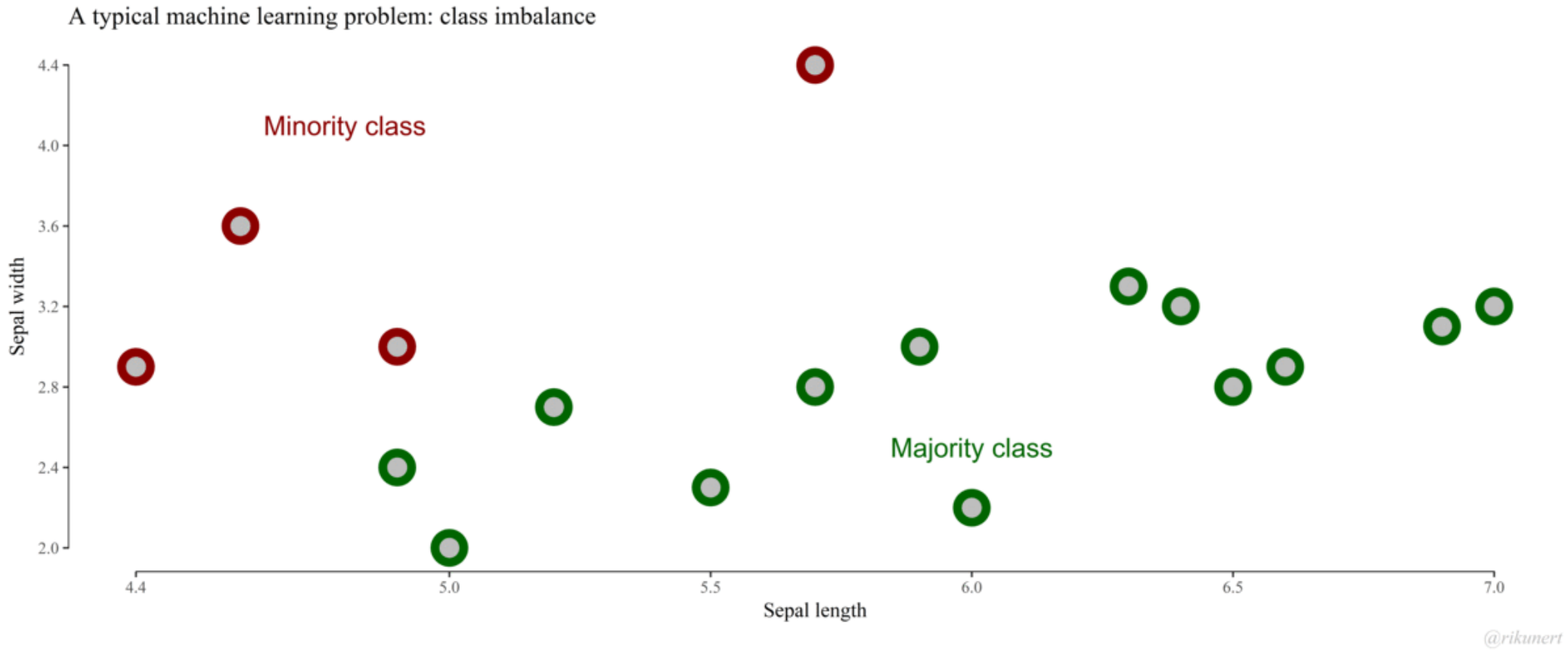


Step 1:
Identify a point from the minority class

Step 2:
Identify it's nearest neighbors. (Here 4 neighbors are taken. All minority neighbors)

Step 3:
Assume we want to add 2 times more synthetic observations
Select 2 nearest neighbor randomly
Draw a line between the point and the selected neighbors
Create a synthetic observation along the line

SMOTE Example (Animated)



SMOTE in Python

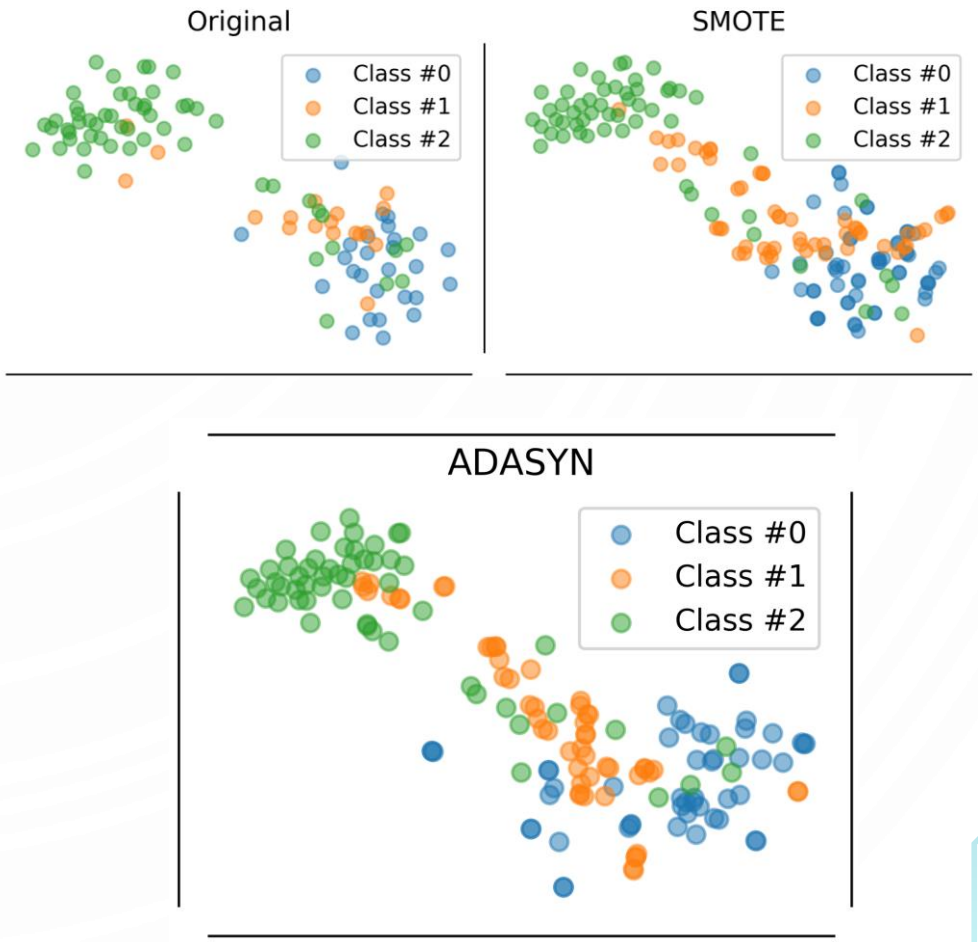
1:
```
$sudo pip install imbalanced-learn
```

2:
```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)
```

How ADASYN Works?

- ADASYN implements a methodology that detects those samples of the minority class **found in spaces dominated by the majority class**
  - This in order to generate samples in the lower density areas of the minority class.



ADASYN in Python

1:
```
$sudo pip install imbalanced-learn
```

2:
```
from imblearn.over_sampling import ADASYN

ada = ADASYN(random_state= 42)
X_res, y_res = ada.fit_resample(X, y)
```

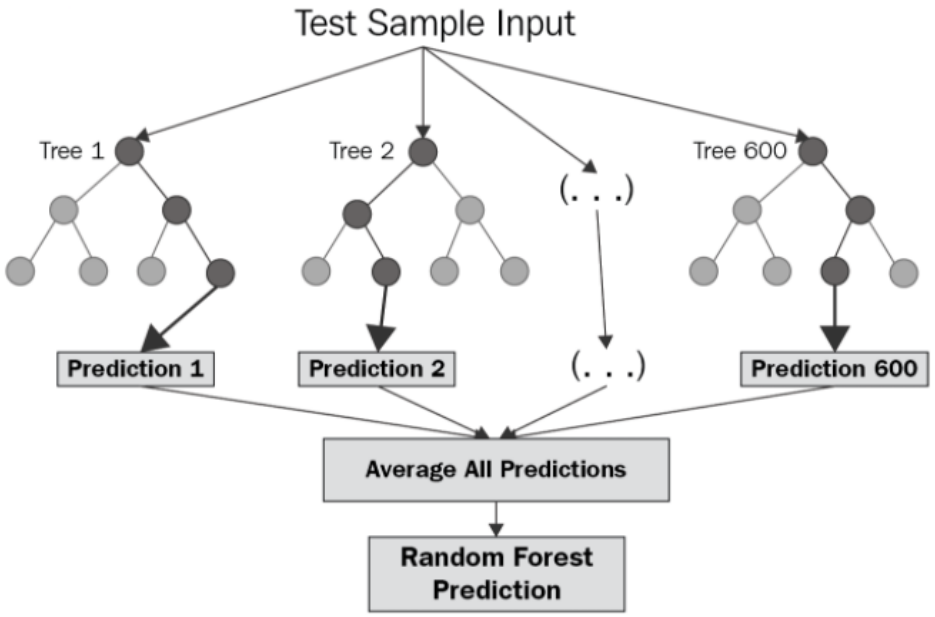
8.2 Combining Models

Combining Models

- Ensemble algorithms, like Random Forest, typically combine models of the same nature.
  - They boost performance by combining hundreds of weak models.
- There are three typical ways to combine models:
  1. Averaging
  2. Majority vote
  3. Stacking

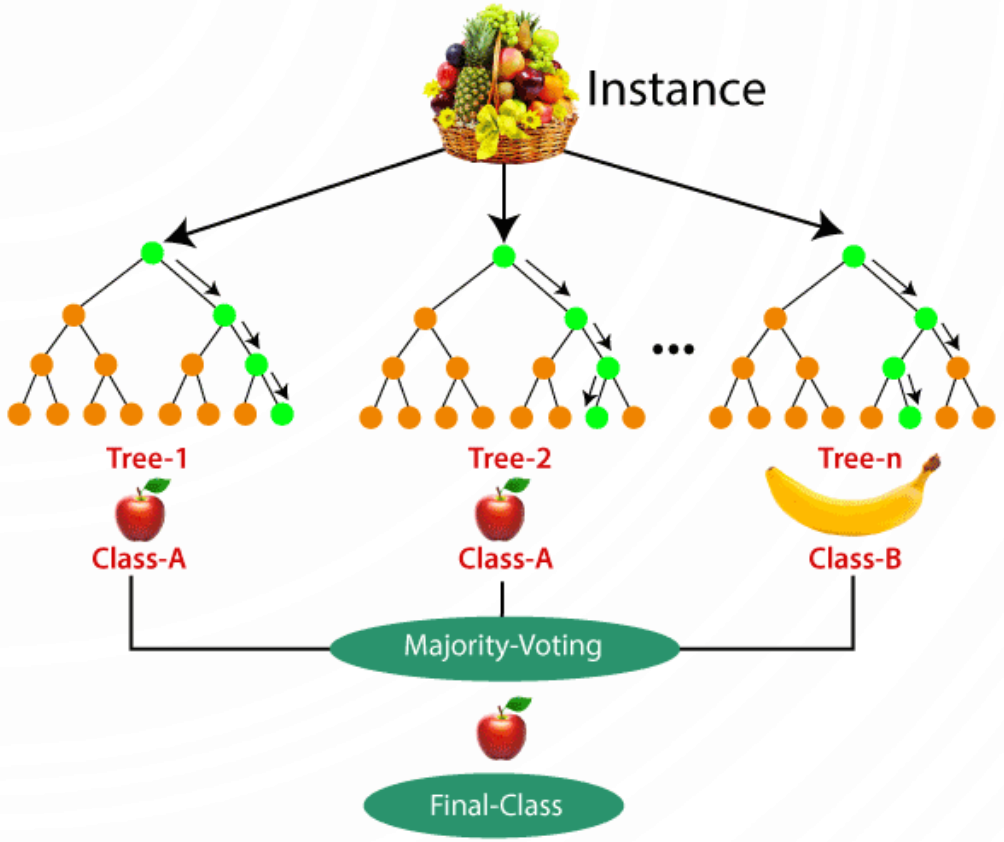
Averaging

- Averaging works for regression as well as those classification models that return classification scores.
- You simply apply all your models, let's call them base models, to the input x and then average the predictions.



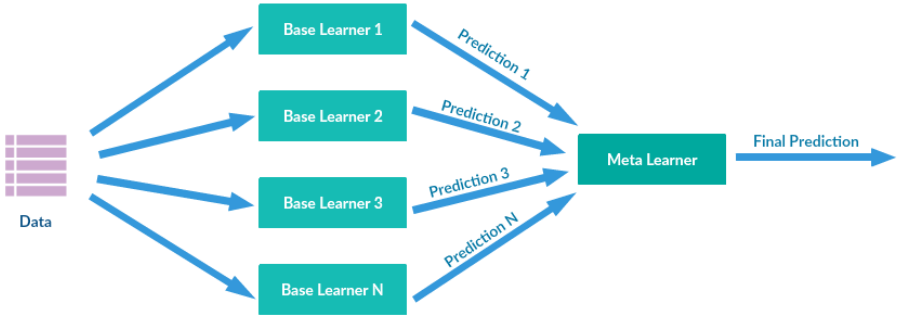
Majority Vote

- Majority vote works for classification models.
- You apply all your base models to the input x and then return the majority class among all predictions.
- In the case of a tie, you either randomly pick one of the classes, or you return an error message



Stacking

- Stacking consists of building a meta-model that takes the output of your base models as input.
- The way how it works is by :
    1. Split training data into k folds
    2. Each fold split into train and test set
    3. We will use the training data to train the different learners (base learners)
    4. And the test data we will use them to make predictions
    5. The predictions will be fed into the meta learner

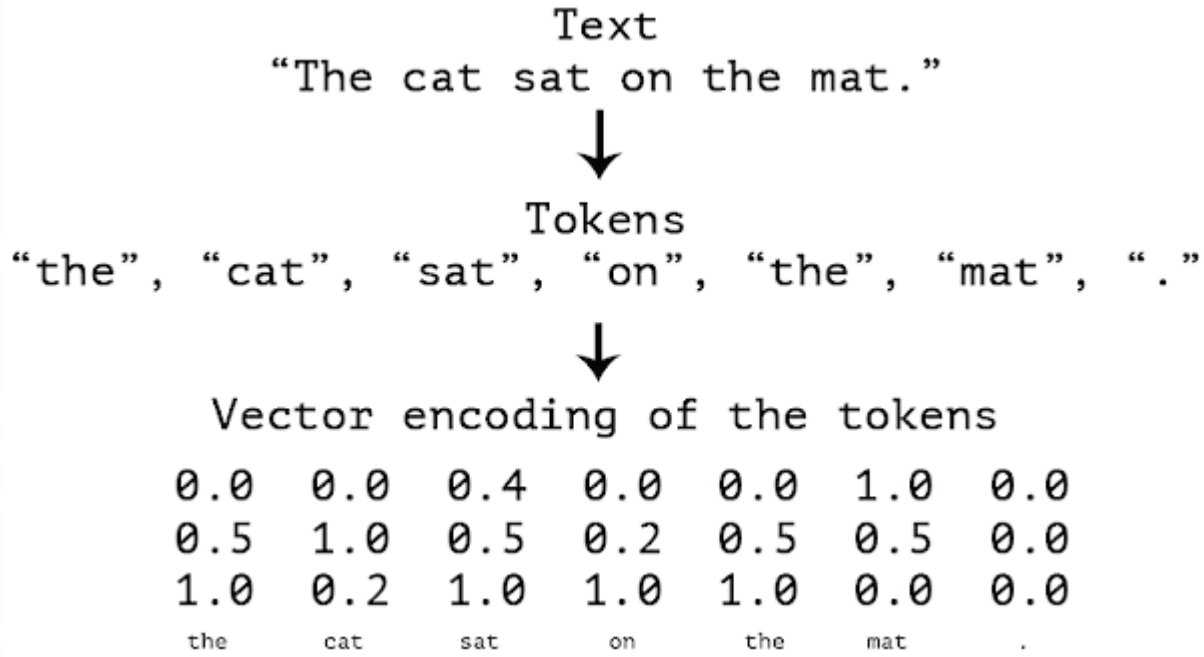


8.3 Training Neural Networks

Training Neural Networks

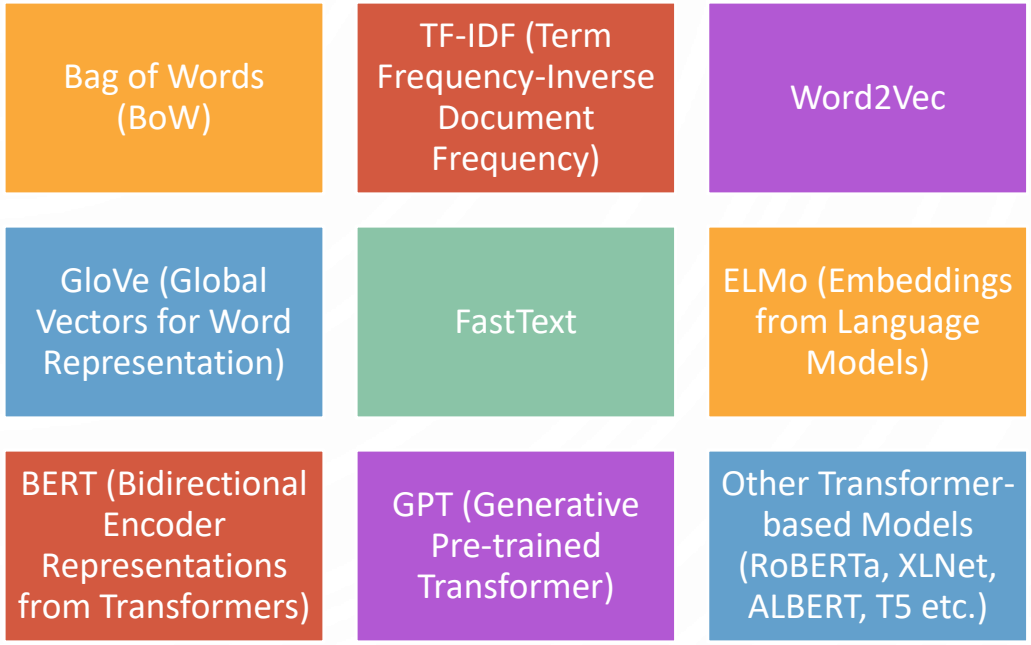
- In neural network training, one challenging aspect is how to convert your data into the input the network can work with.
- If your input is images, first of all, you have to resize all images so that they have the same dimensions.
  - After that, pixels are usually first standardized and then normalized to the range [0, 1].
- Texts have to be tokenized.

Text Tokening (Vector encoding)



Word Embedding

- In natural language processing, a word embedding is a representation of a word.
- Typically, the representation is a real-valued vector that encodes the meaning of the word in such a way that the words that are closer in the vector space are expected to be similar in meaning.



Bag of Words (BoW)

Document D1	*The child makes the dog happy* the: 2, dog: 1, makes: 1, child: 1, happy: 1
Document D2	*The dog makes the child happy* the: 2, child: 1, makes: 1, dog: 1, happy: 1

	child	dog	happy	makes	the	BoW Vector representations
D1	1	1	1	1	2	[1,1,1,1,2]
D2	1	1	1	1	2	[1,1,1,1,2]

TF-IDF (Term Frequency-Inverse Document Frequency)

- TF-IDF can quantify the importance or relevance of string representations (words, phrases, lemmas, etc) in a document amongst a collection of documents (also known as a corpus).

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

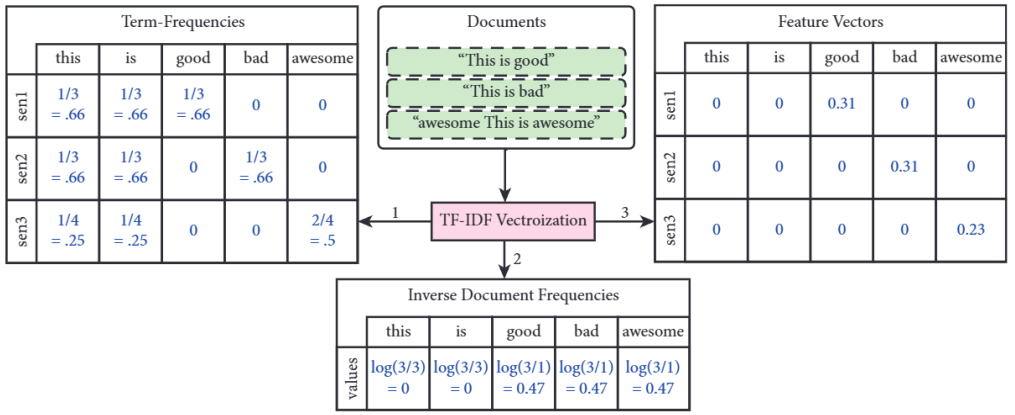
TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents



TF-IDF (Term Frequency-Inverse Document Frequency)

$\texttt{tf}(t, d)$

	blue	bright	can	see	shining	sky	sun	today
1	1/2	0	0	0	0	1/2	0	0
2	0	1/3	0	0	0	0	1/3	1/3
3	0	1/3	0	0	0	1/3	1/3	0
4	0	1/6	1/6	1/6	1/6	0	1/3	0

x

$\texttt{idf}(t, D)$

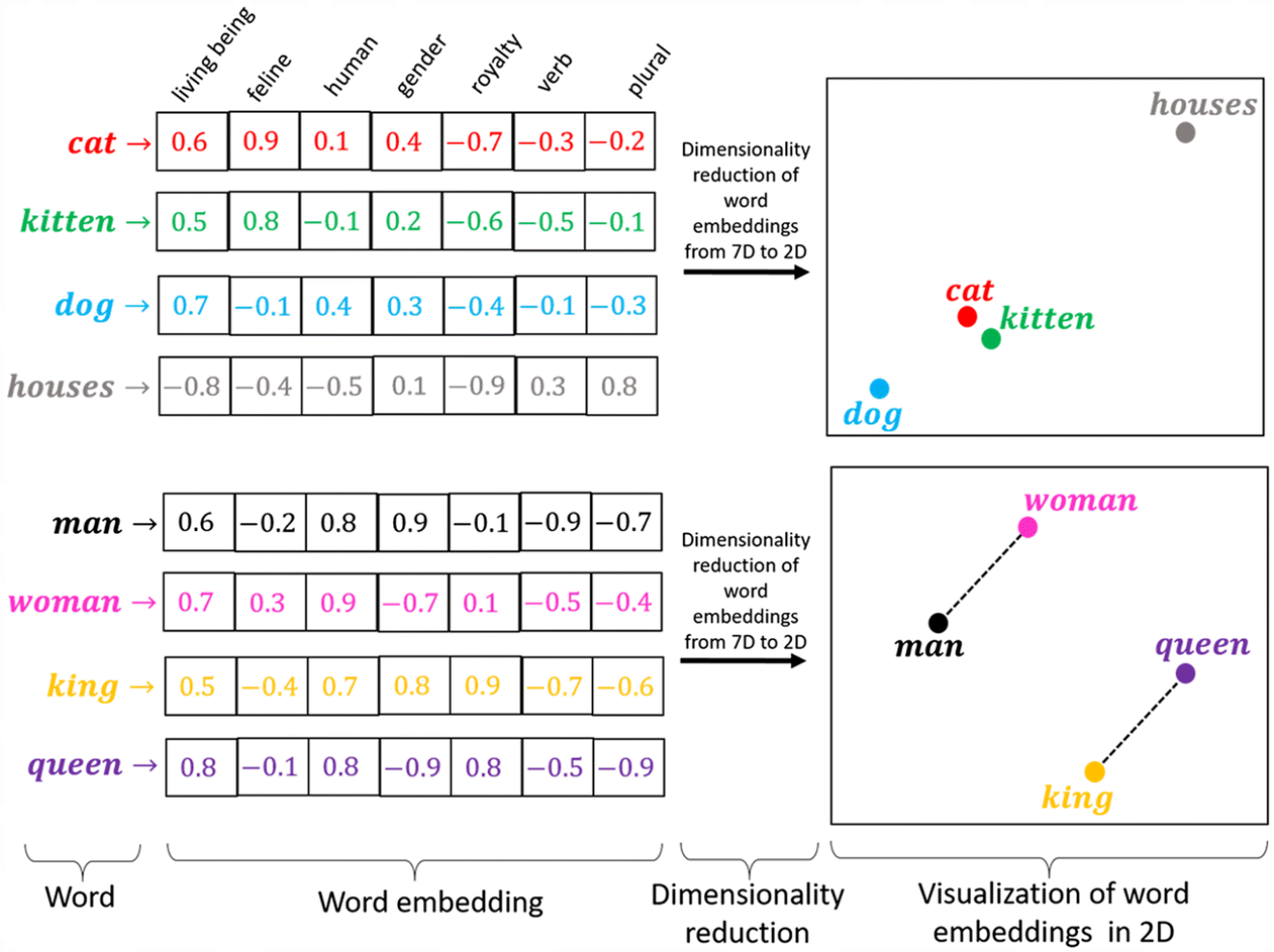
	blue	bright	can	see	shining	sky	sun	today
	0.602	0.125	0.602	0.602	0.602	0.301	0.125	0.602

- TF-IDF: Multiply TF and IDF scores, use to rank importance of words within documents
  - Most important word for each document is highlighted

$\texttt{tfidf}(t, d, D) = \texttt{tf}(t, d) \cdot \texttt{idf}(t, D)$

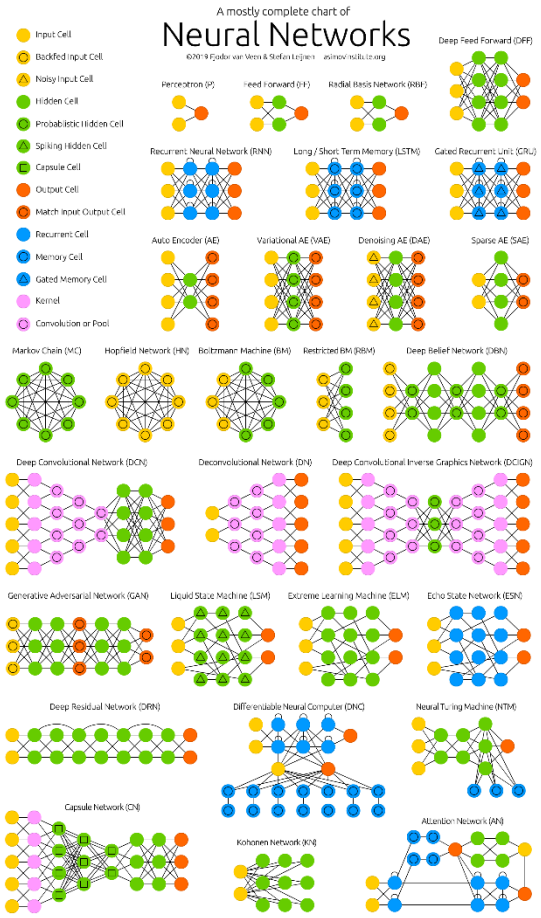
	blue	bright	can	see	shining	sky	sun	today
1	**0.301**	0	0	0	0	0.151	0	0
2	0	0.0417	0	0	0	0	0.0417	**0.201**
3	0	0.0417	0	0	0	**0.100**	0.0417	0
4	0	0.0209	**0.100**	**0.100**	**0.100**	0	0.0417	0

Word Embeddings Visualization (Word2Vec)

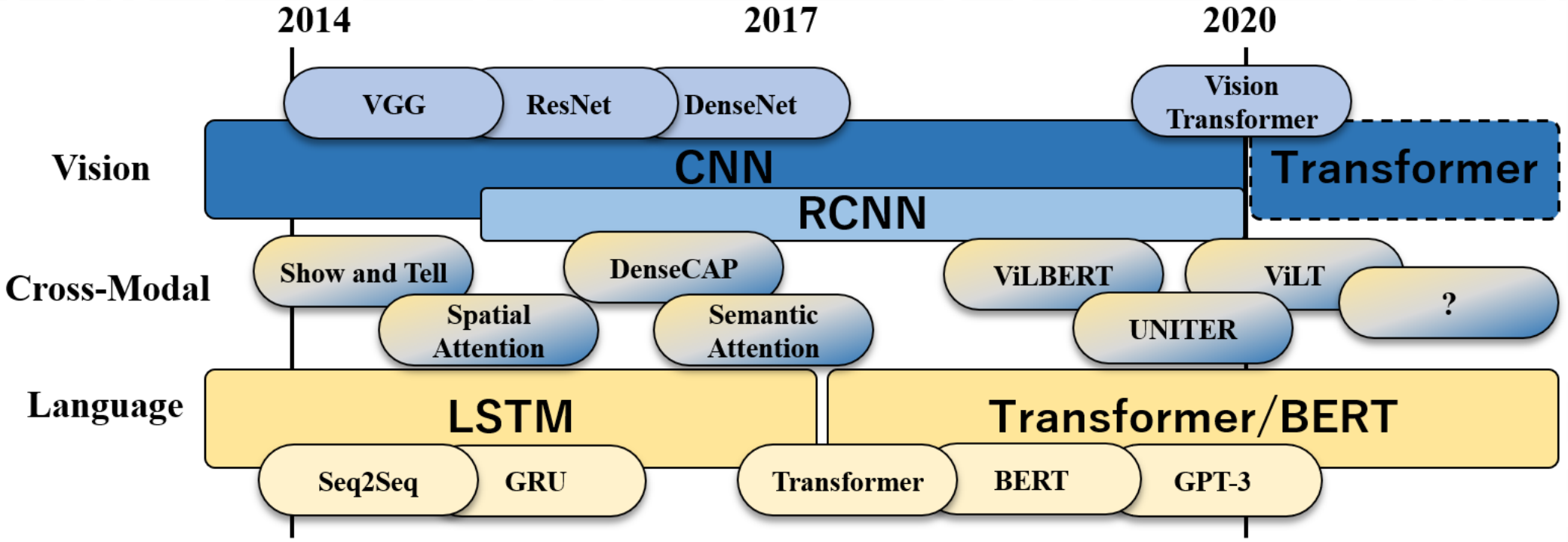


Neural Network Architectures

- The choice of specific neural network architecture is a difficult one.
- For any problem, there is a variety of architectures, and new ones are proposed almost every year.

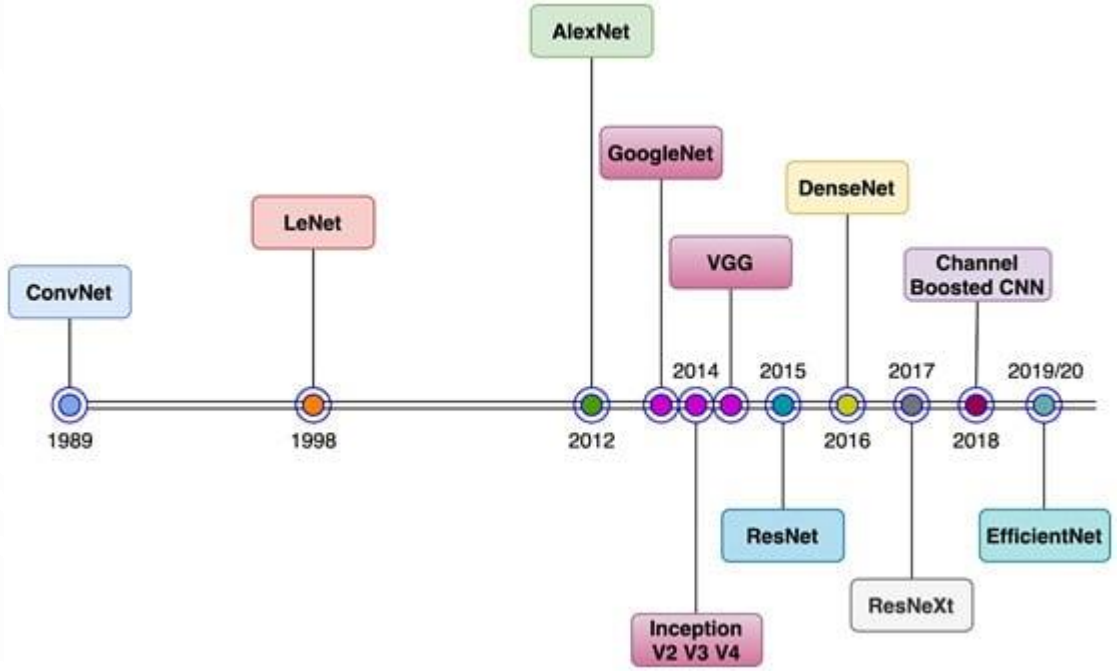


Popular NN architectures Timeline



A brief timeline showing important models in vision (upper) and language (lower) domains respectively, and how they have affected approaches in cross-modal domains (middle).

CNN Architectures Over a Timeline (1989-2019)



ImageNet Classification Comparison

Architecture	Number of Parameters	Top 5 Accuracy	Top 1 Accuracy
AlexNet	62,378,344	84.60%	63.30%
VGG16	138,357,544	91.90%	74.40%
GoogLeNet	23,000,000	92.2%	74.80%
ResNet-152	25,000,000	94.29%	78.57%
DenseNet	8,062,504	93.34%	76.39%
Xception	22,910,480	94.50%	79.00%

Top 5 accuracy, top 1 accuracy, and the number of parameters of AlexNet, VGG, Inception, and ResNet in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) challenge.

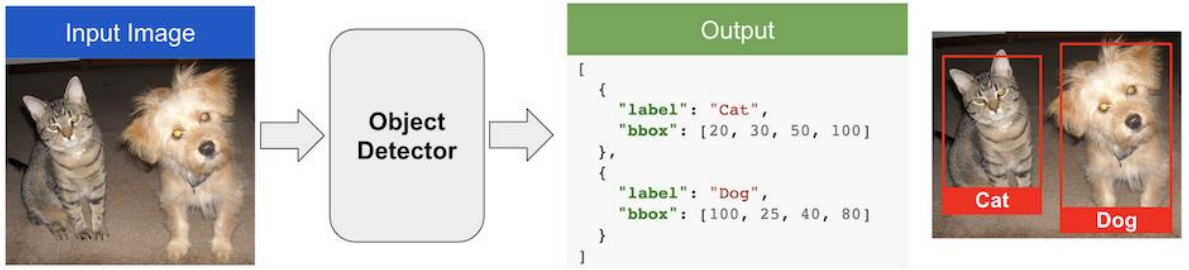
Top-1 accuracy is the conventional accuracy: the model answer (the one with highest probability) must be exactly the expected answer. **Top-5** accuracy means that any of your model 5 highest probability answers must match the expected answer.

Imagenet Benchmark Updated Results: https://paperswithcode.com/sota/image-classification-on-imagenet

8.6 Handling Multiple Outputs

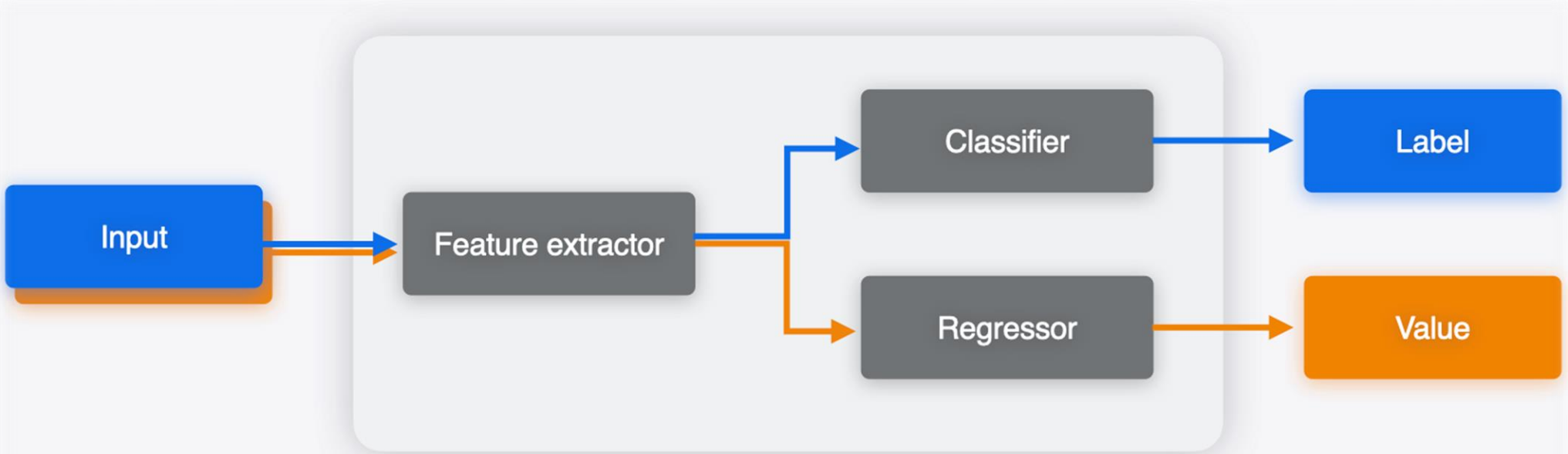
Handling Multiple Outputs

- In some problems, you would like to predict multiple outputs for one input.
  - Example: you want to build a model that detects an object on an image and returns its coordinates.

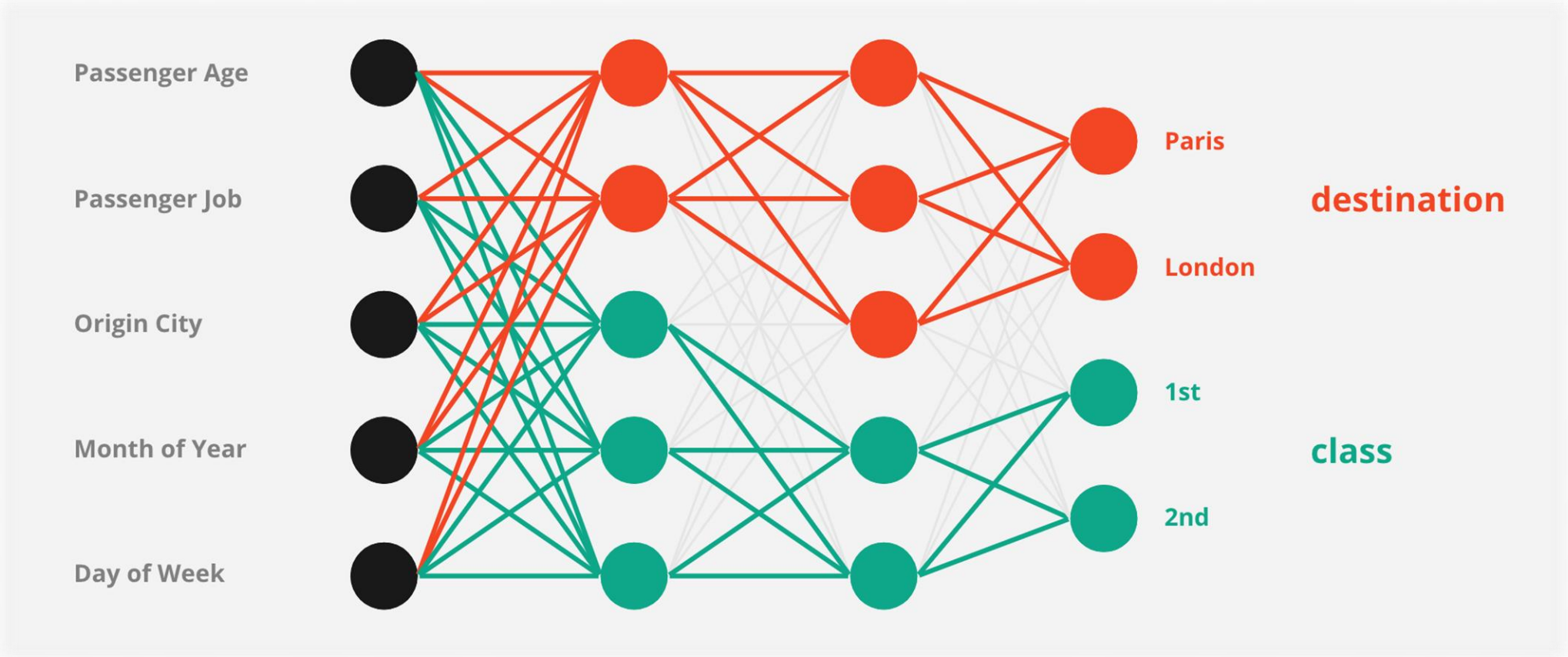


- Some problems with multiple outputs can be effectively converted into a multi-label classification problem.

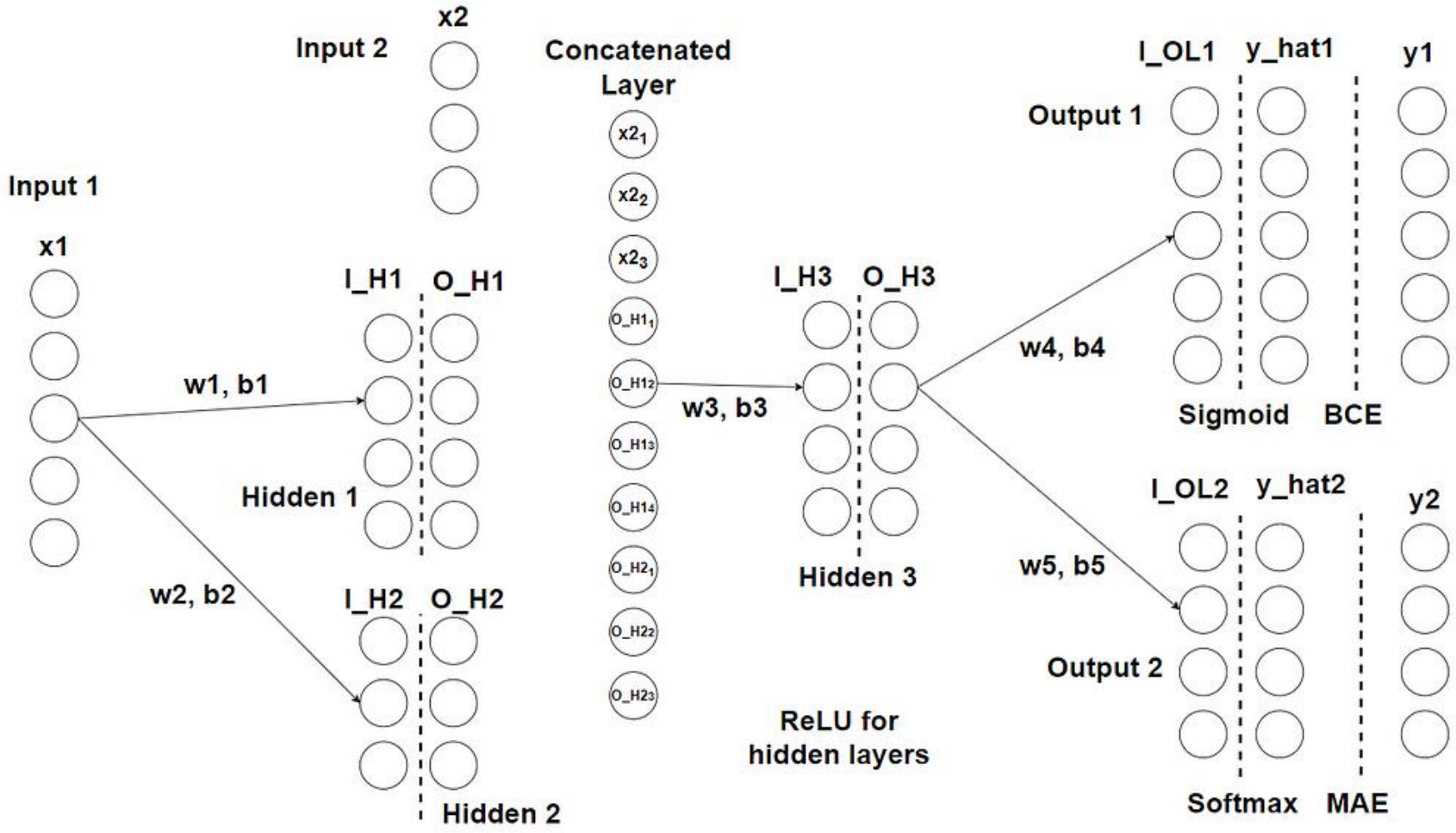
Multiple Outputs Example



Neural Network Multiple Outputs Example



Neural Network Multiple Outputs Example 2

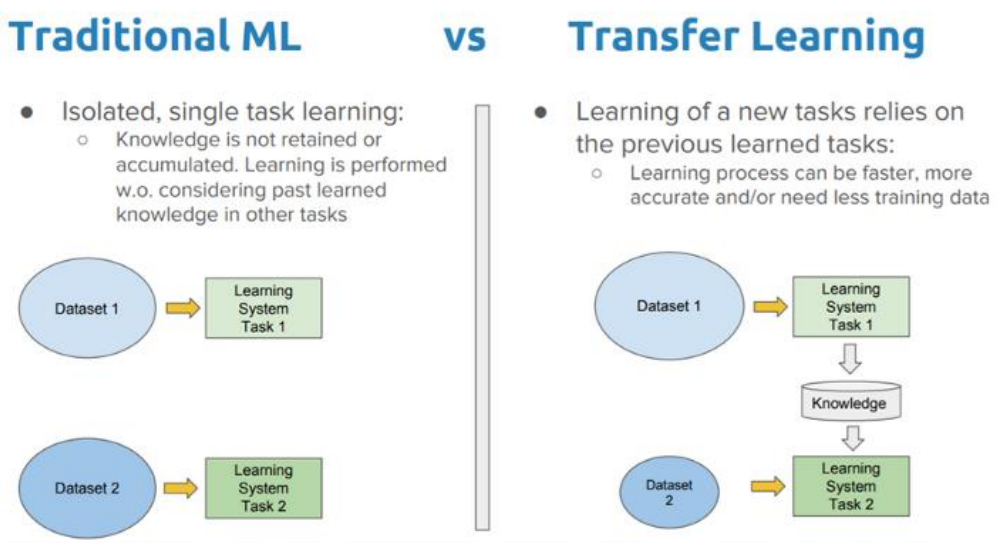


Source: https://neuralthreads.medium.com/multiple-inputs-multiple-outputs-in-a-neural-network-7860c5278a30

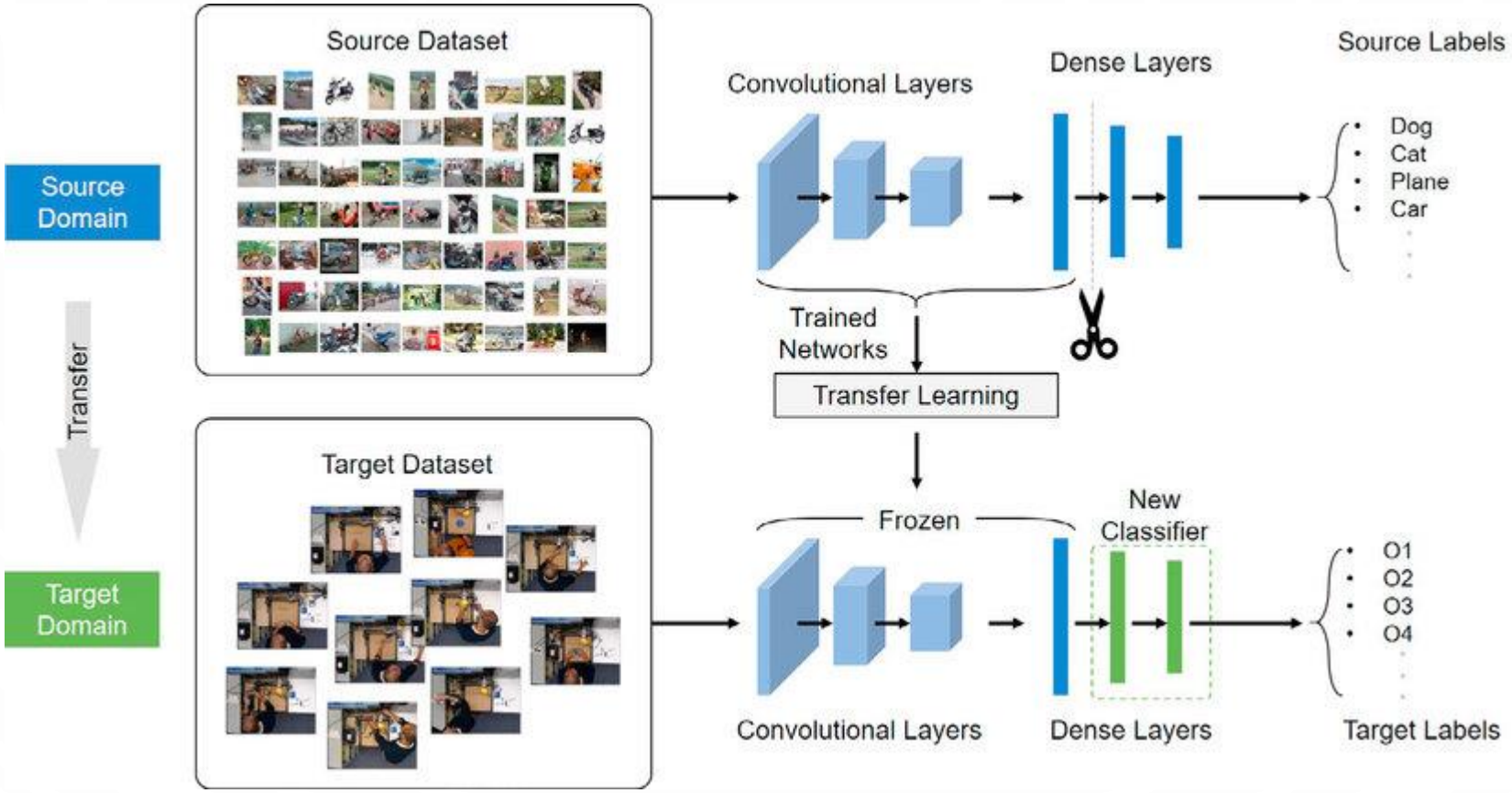
8.7 Transfer Learning

Transfer Learning

- **Transfer Learning:** you pick an existing model trained on some dataset, and you adapt this model to predict examples from another dataset
- **How is it done?**
  1. Build a deep model on the original big dataset.
  2. Compile a much smaller labeled dataset for your second model.
  3. Remove the last one or several layers from the first model.
  4. Replace the removed layers with new layers adapted for your new problem.
  5. “Freeze” the parameters of the layers remaining from the first model.
  6. Use your smaller labeled dataset and gradient descent to train the parameters of only the new layers.



Transfer Learning (Freezing Some Layers)



Transfer Learning with TensorFlow Hub Example

- TensorFlow Hub is a repository of pre-trained TensorFlow models.
- Example
  - https://colab.research.google.com/drive/1S4FIM65Dej0yWFLVcwrKB81ZDtEe4cqg?usp=sharing