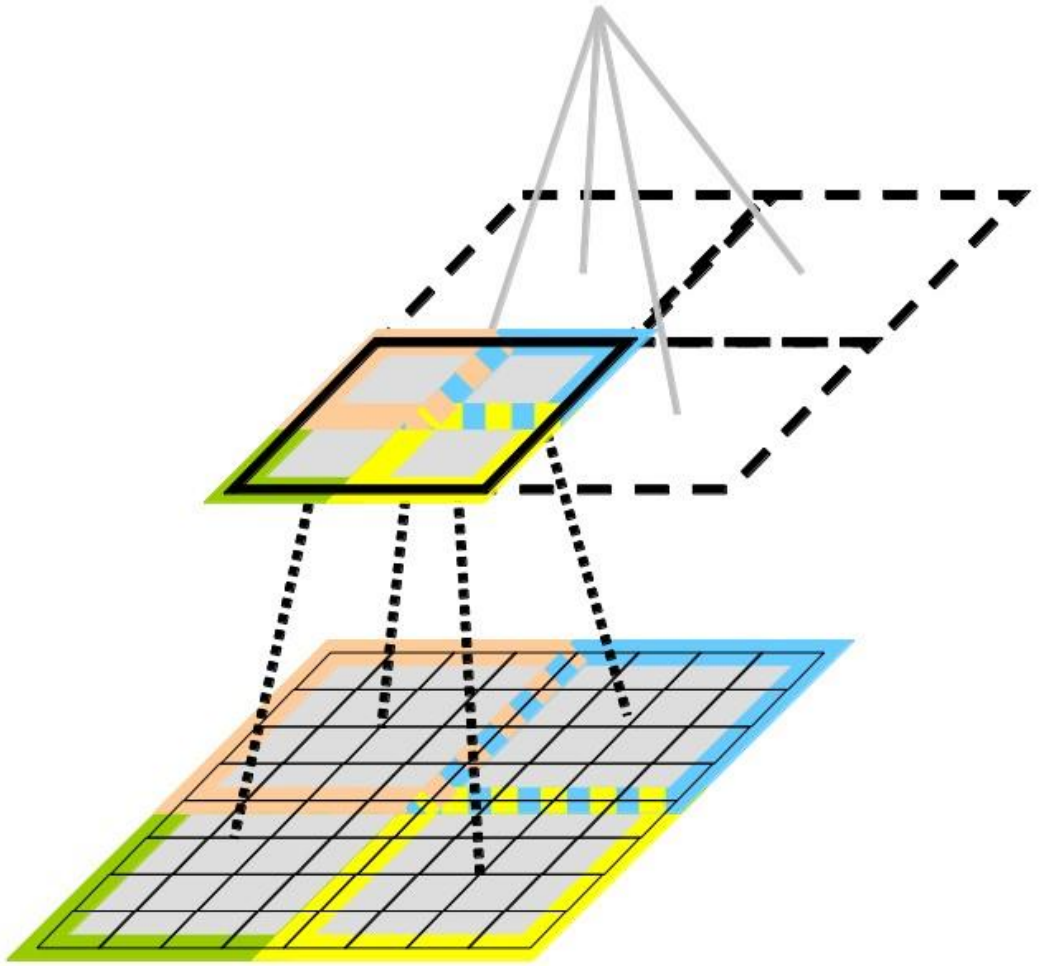
CSC 767 Neural Networks & Deep Learning

Lecture 8

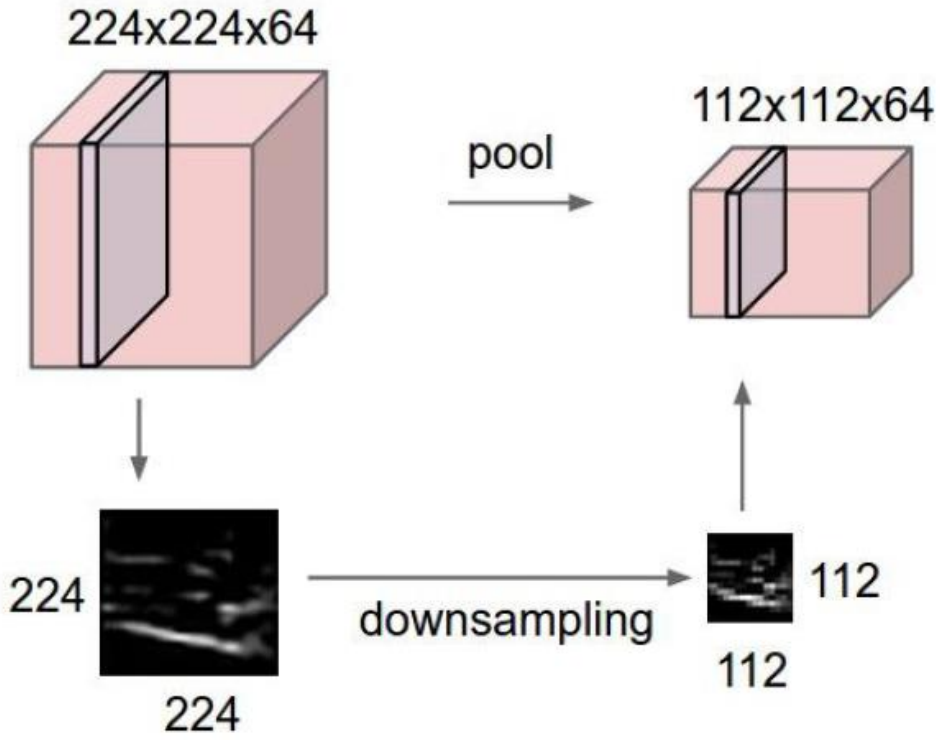
Pooling. Convolution Demo. More on Augmentation. Training Convolutional NN. Imbalanced Datasets. Ensembles

Pooling

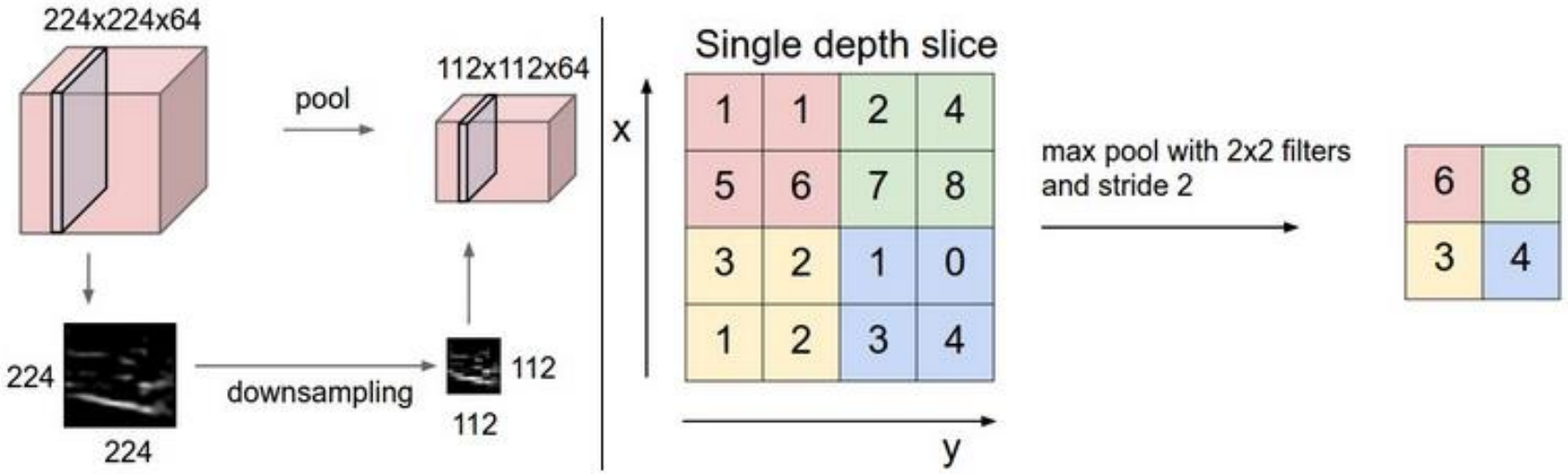


Pooling (1)

- makes the representations smaller and more manageable
- operates over each activation map independently:



Pooling (2)



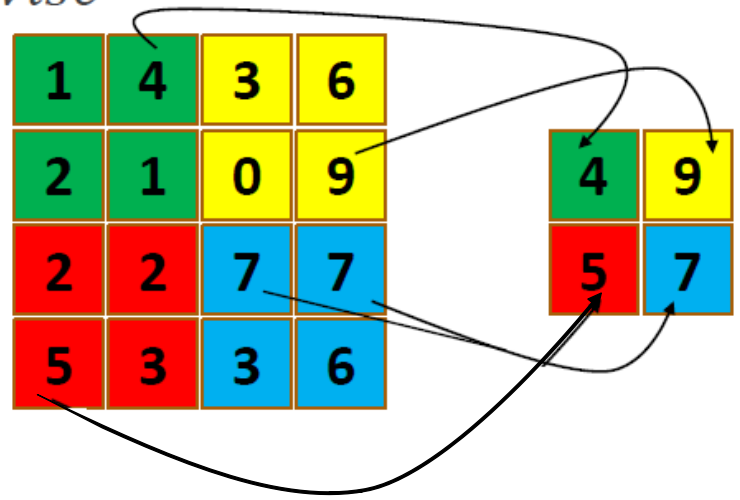
Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

Pooling (3)

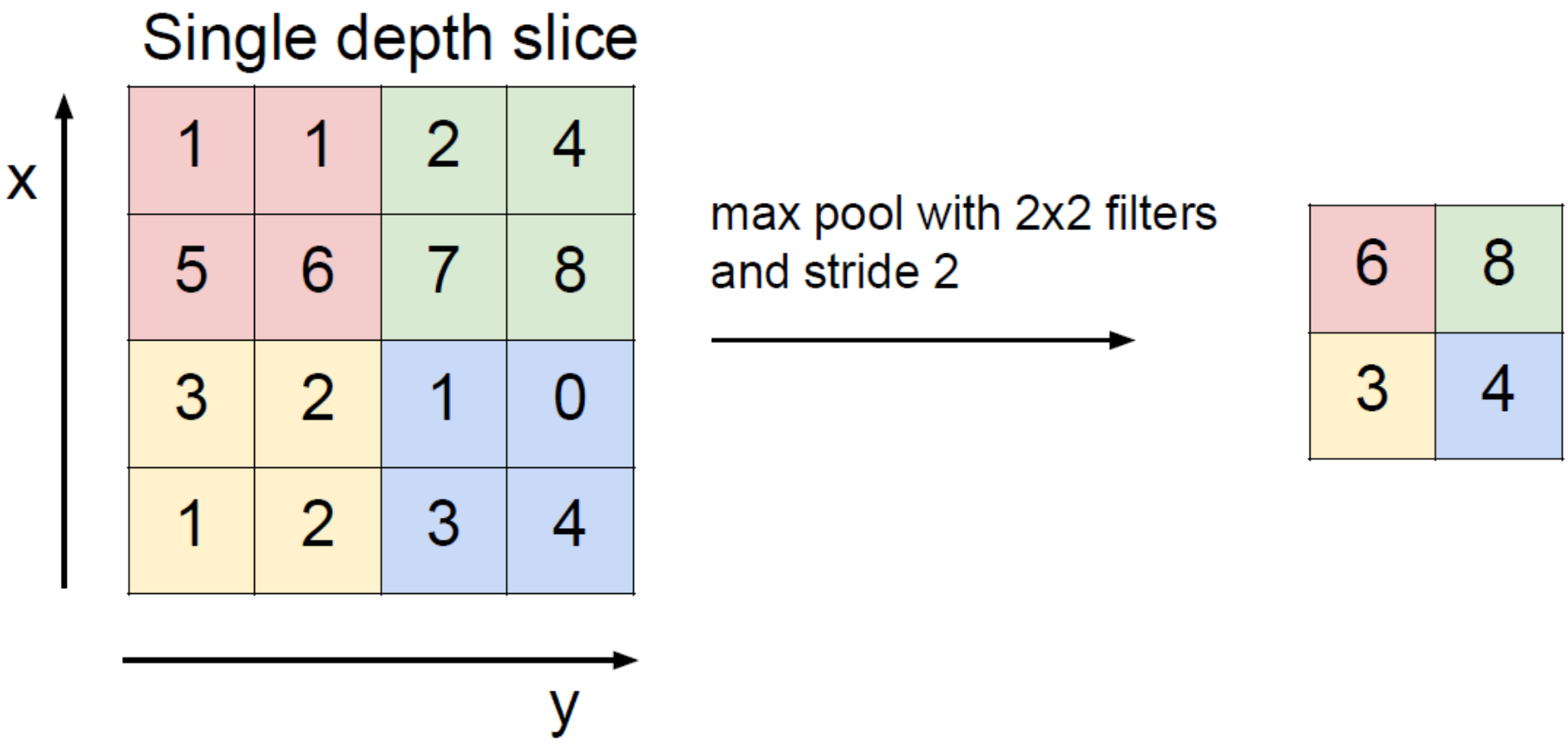
- Aggregate multiple values into a single value
  - Invariance to small transformations
  - Reduces the size of the layer output/input to next layer → Faster computations
  - Keeps most important information for the next layer
- Max pooling
- Average pooling

Max Pooling (New Module!) (1)

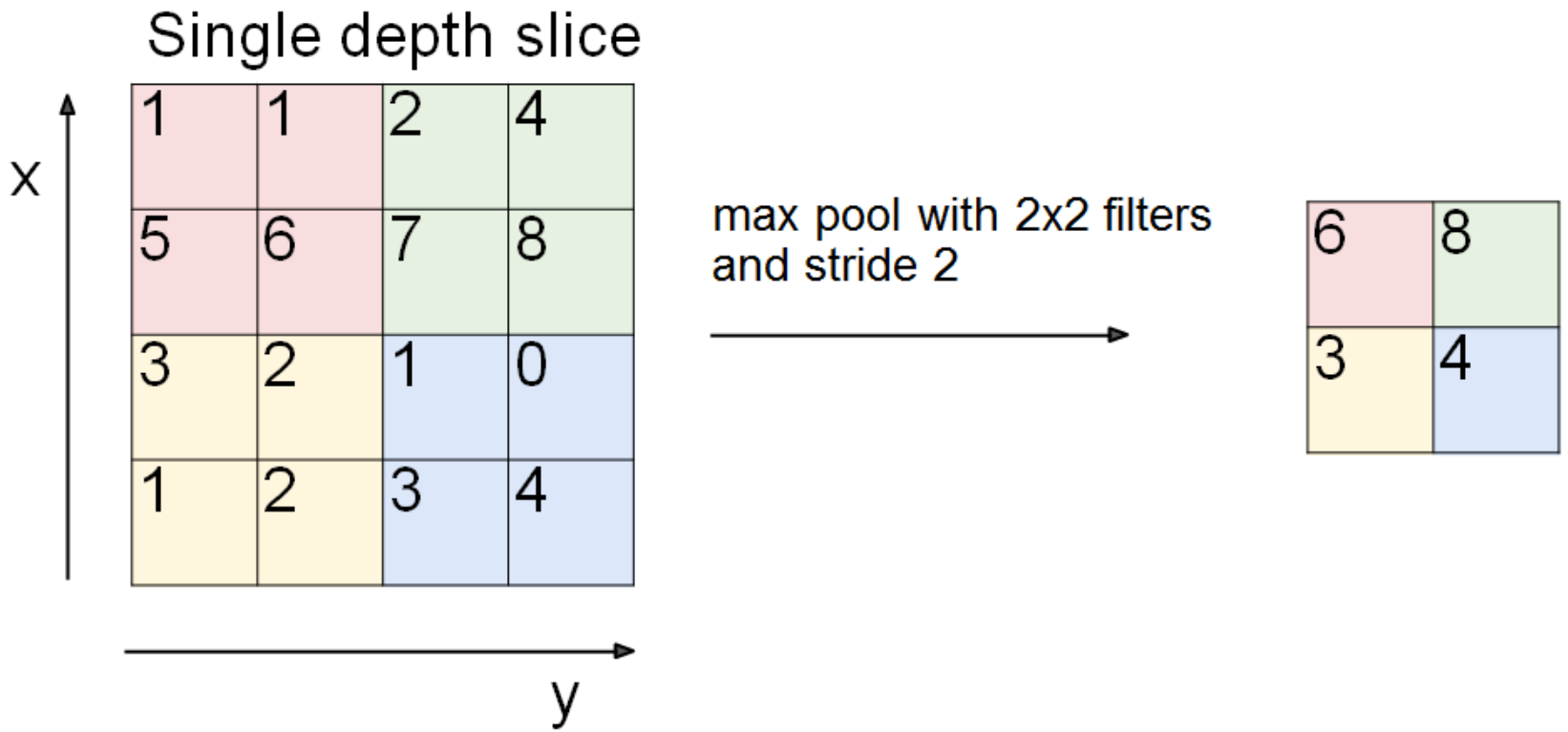
- Run a sliding window of size $[h_f, w_f]$
- At each location keep the maximum value
- Activation function: $\mathrm{i}_{\max}, \mathrm{j}_{\max} = \underset{i,j \in \Omega(r,c)}{\arg\max}\, x_{ij} \rightarrow a_{rc} = x[\mathrm{i}_{\max}, \mathrm{j}_{\max}]$
- Gradient w.r.t. input $\frac{\partial a_{rc}}{\partial x_{ij}} = \begin{cases} 1, & if\ i = \mathrm{i}_{\max}, \mathrm{j} = \mathrm{j}_{\max} \\ 0, & otherwise \end{cases}$
- The preferred choice of pooling



Max Pooling (New Module!) (2)

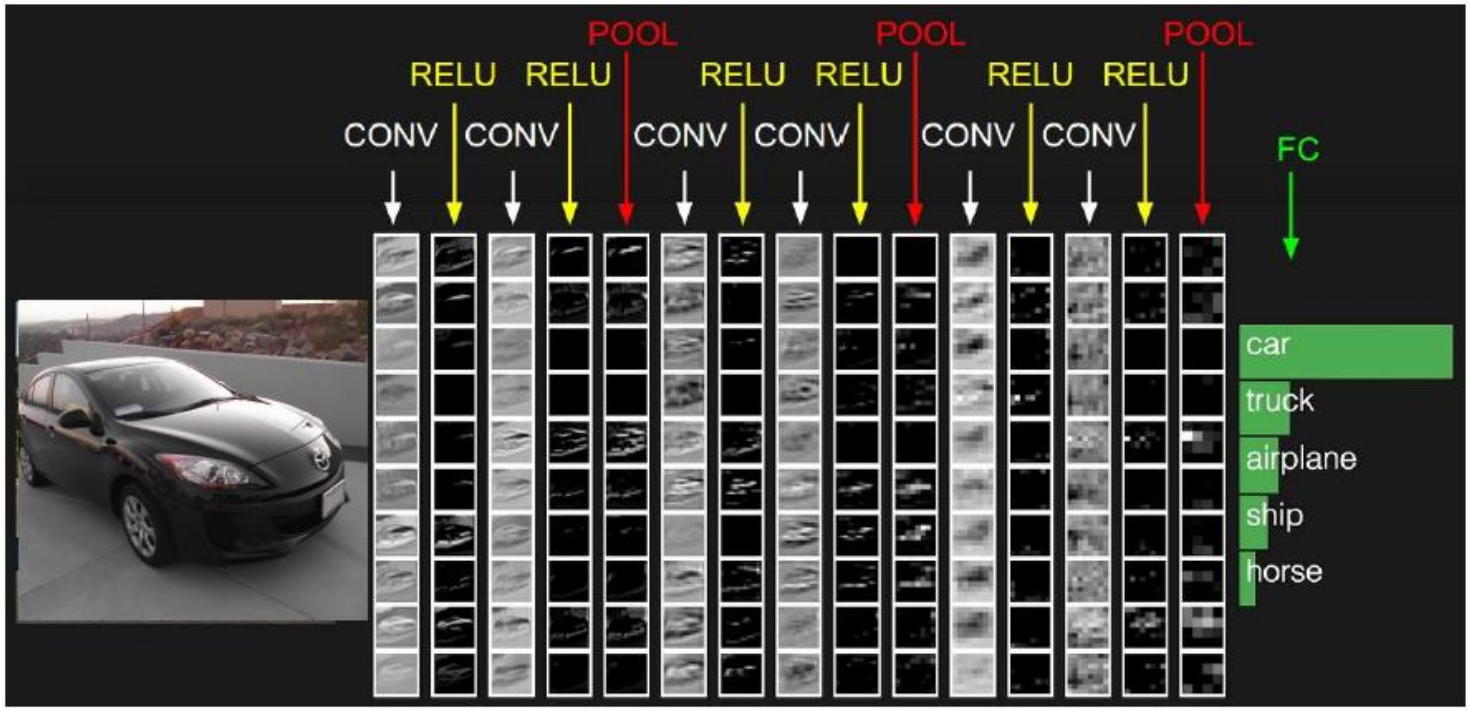


MAX POOLING


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Fully Connected Layer (FC Layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



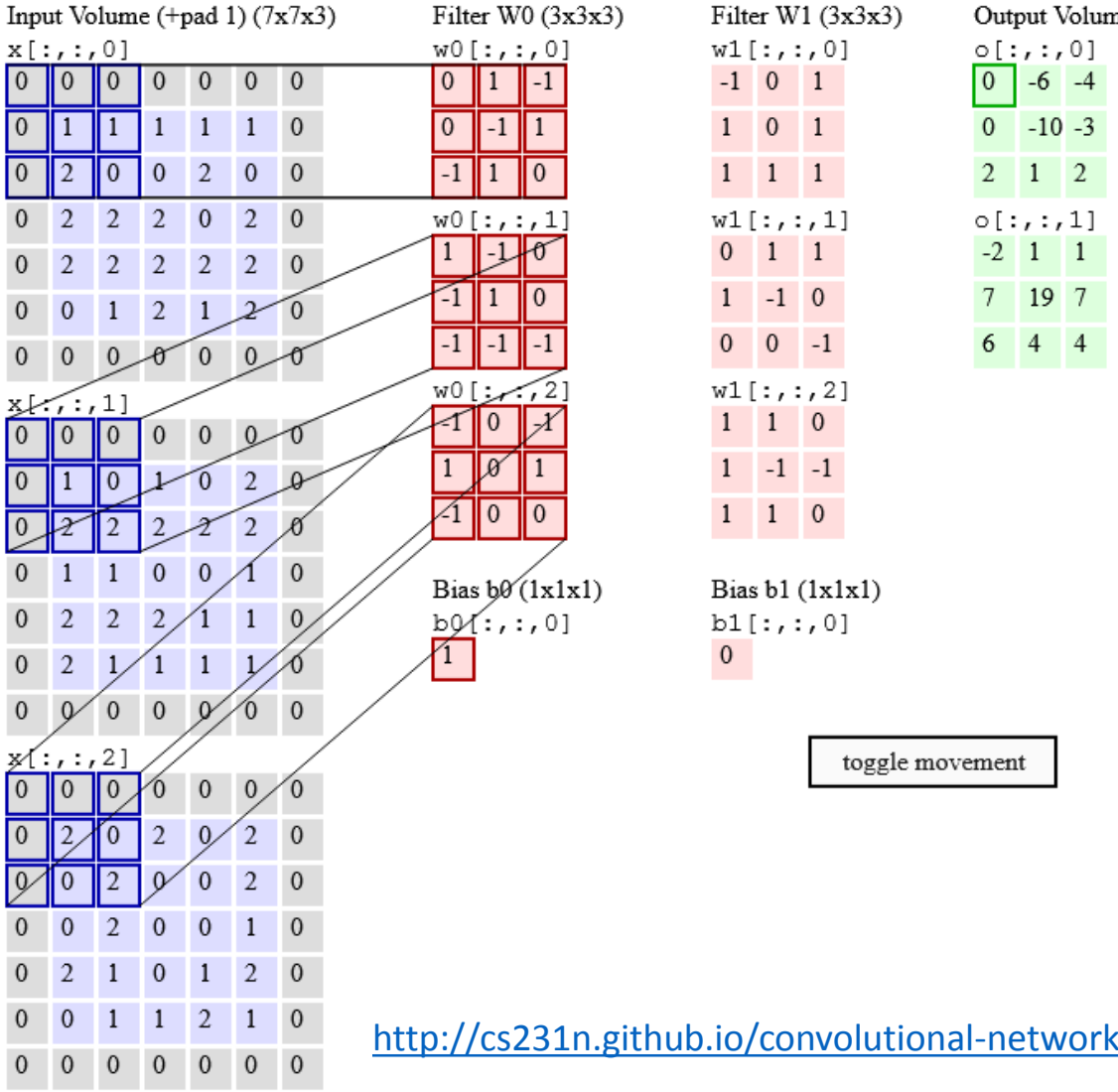
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

F = 2, S = 2

F = 3, S = 2



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

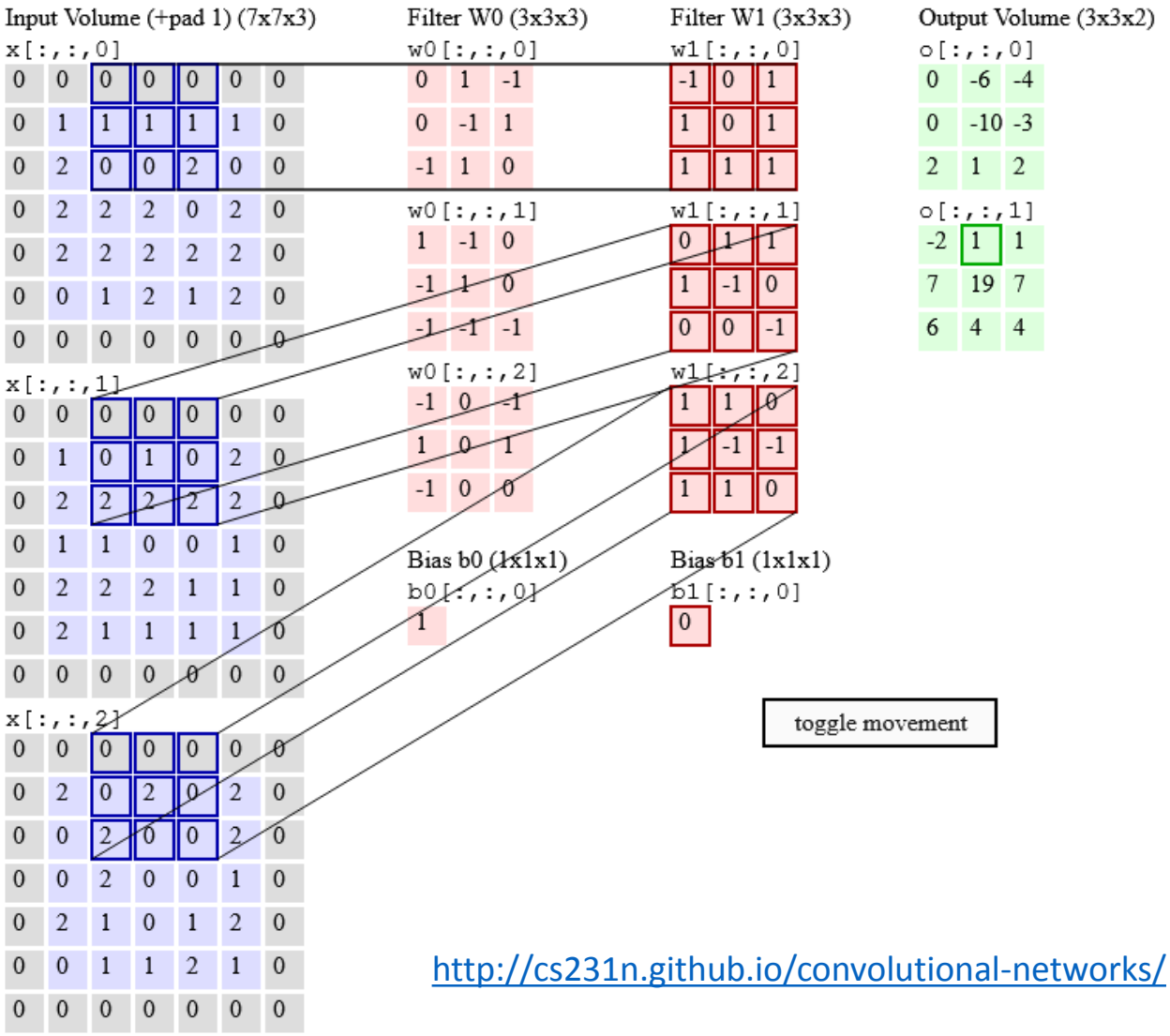
Convolution Demo

•Since 3D volumes are hard to visualize, all the volumes (the input volume (in blue), the weight volumes (in red), and the output volume (in green)) are visualized with each depth slice stacked in rows.

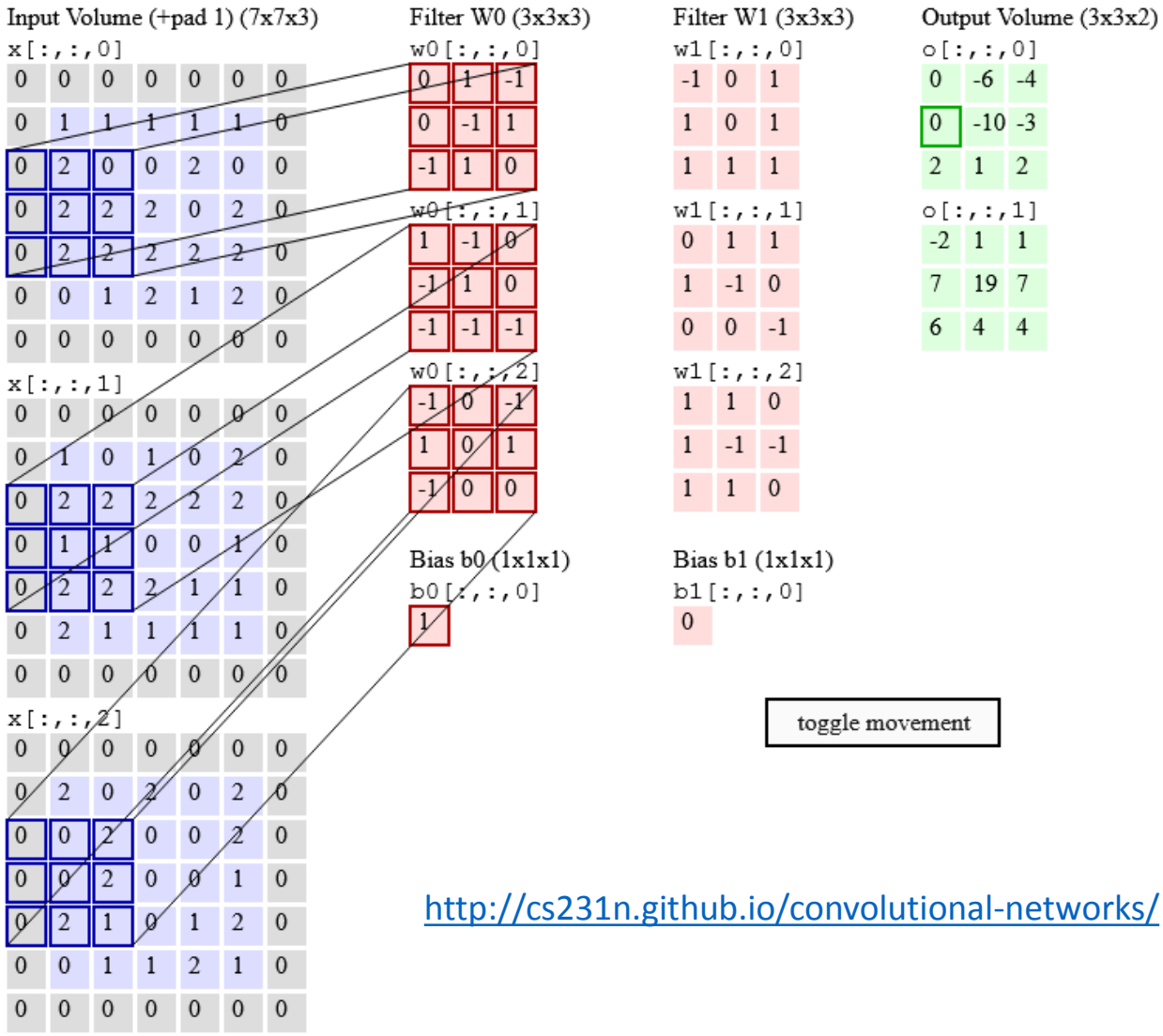
•The input volume is of size W1=5, H1=5, D1=3, and the CONV layer parameters are K=2, F=3, S=2, P=1. That is, we have two filters of size 3×3, and they are applied with a stride of 2.

•Therefore, the output volume size has spatial size (5 - 3 + 2)/2 + 1 = 3. Moreover, notice that a padding of P=1 is applied to the input volume, making the outer border of the input volume zero.

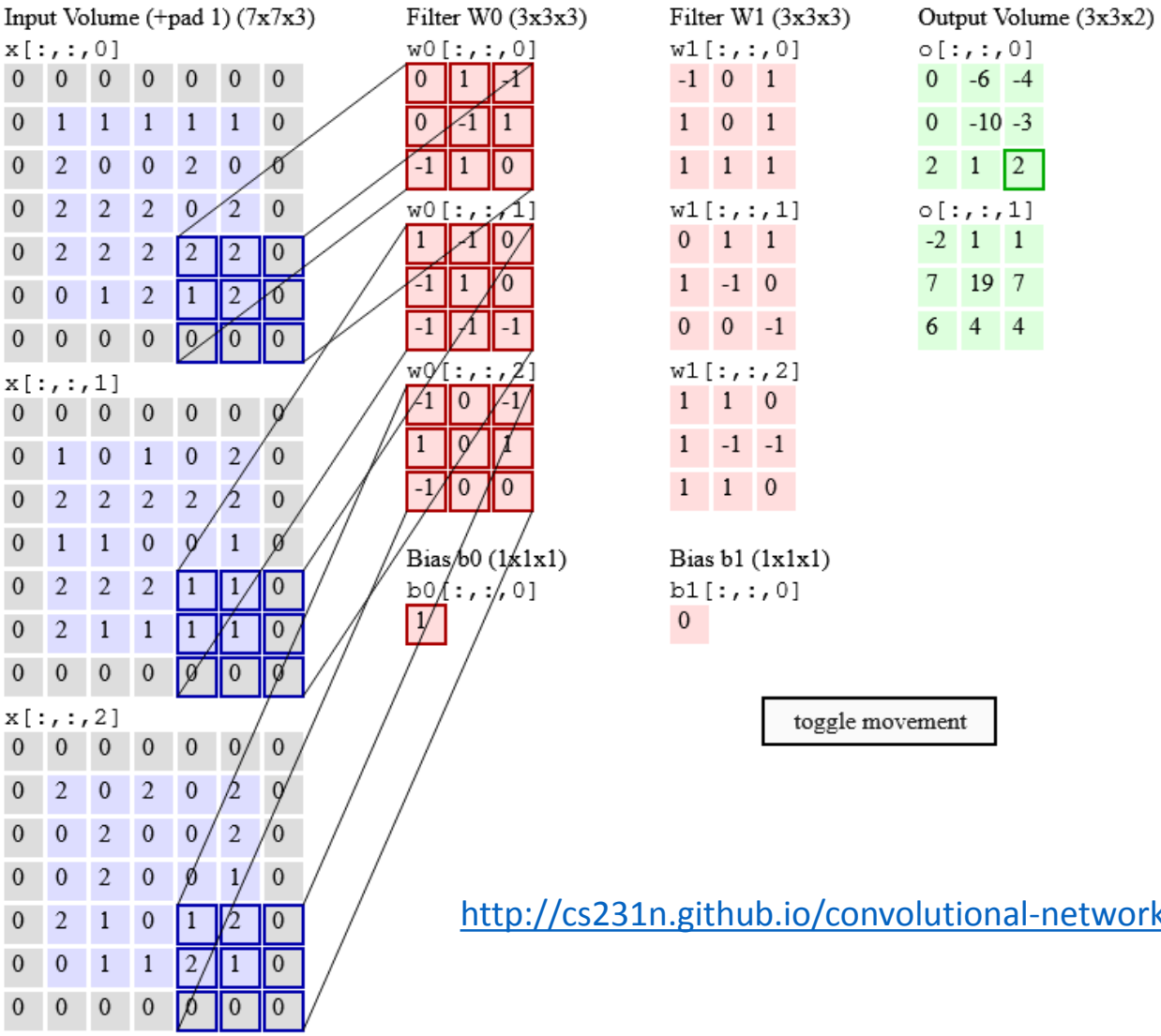
•The visualization below iterates over the output activations (green), and shows that each element is computed by elementwise multiplying the highlighted input (blue) with the filter (red), summing it up, and then offsetting the result by the bias.



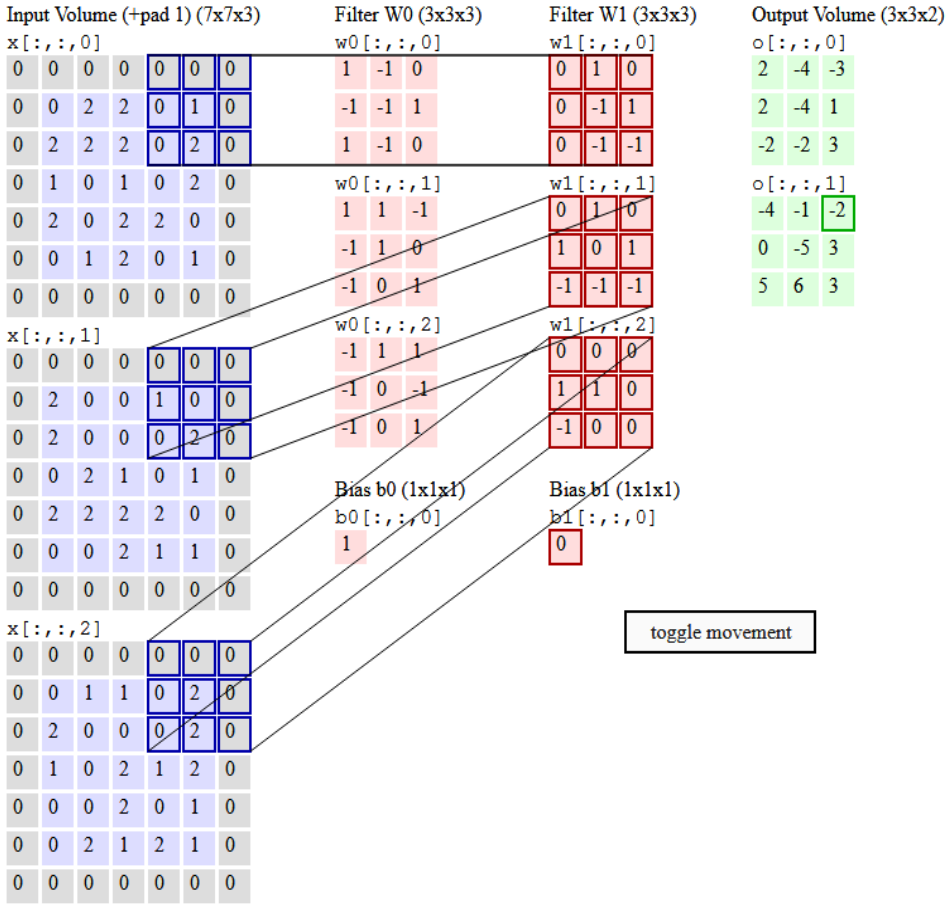
http://cs231n.github.io/convolutional-networks/

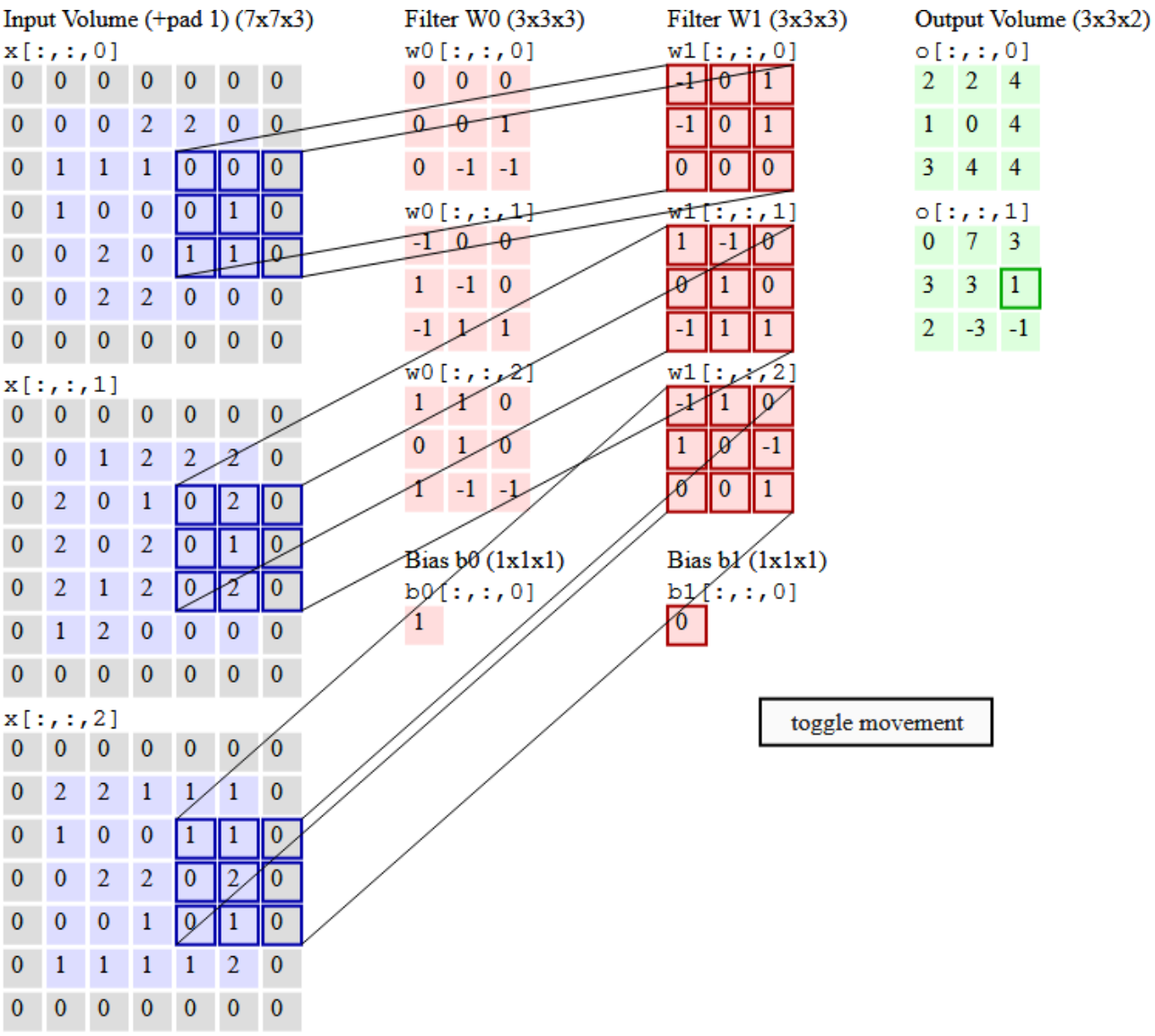


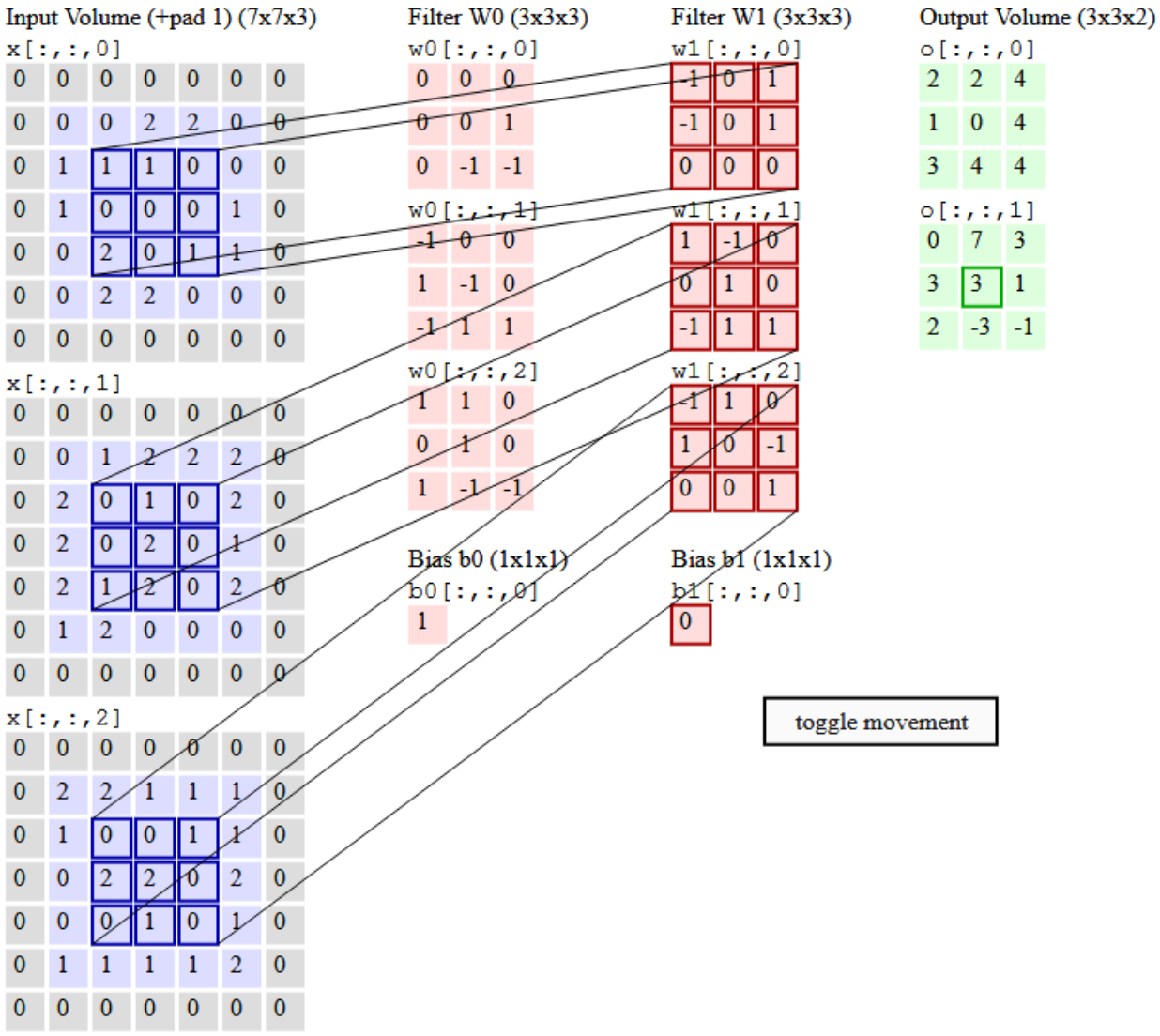
http://cs231n.github.io/convolutional-networks/



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


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 2 2 0 1 0
0 2 2 2 0 2 0
0 1 0 1 0 2 0
0 2 0 2 2 0 0
0 0 1 2 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 0 0 1 0 0
0 2 0 0 0 2 0
0 0 2 1 0 1 0
0 2 2 2 2 0 0
0 0 0 2 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 1 0 2 0
0 2 0 0 0 2 0
0 1 0 2 1 2 0
0 0 0 2 0 1 0
0 0 2 1 2 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
-1 -1 1
1 -1 0
w0[:,:,1]
1 1 -1
-1 1 0
-1 0 1
w0[:,:,2]
-1 1 1
-1 0 -1
-1 0 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
0 1 0
0 -1 1
0 -1 -1
w1[:,:,1]
0 1 0
1 0 1
-1 -1 -1
w1[:,:,2]
0 0 0
1 1 0
-1 0 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 -4 -3
2 -4 1
-2 -2 3
o[:,:,1]
-4 -1 -2
0 -5 3
5 6 3
toggle movement


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 2 2 0 0
0 1 1 1 0 0 0
0 1 0 0 0 1 0
0 0 2 0 1 1 0
0 0 2 2 0 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 1 2 2 2 0
0 2 0 1 0 2 0
0 2 0 2 0 1 0
0 2 1 2 0 2 0
0 1 2 0 0 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 2 1 1 1 0
0 1 0 0 1 1 0
0 0 2 2 0 2 0
0 0 0 1 0 1 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 0 1
0 -1 -1
w0[:,:,1]
-1 0 0
1 -1 0
-1 1 1
w0[:,:,2]
1 1 0
0 1 0
1 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 1
-1 0 1
0 0 0
w1[:,:,1]
1 -1 0
0 1 0
-1 1 1
w1[:,:,2]
-1 1 0
1 0 -1
0 0 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 2 4
1 0 4
3 4 4
o[:,:,1]
0 7 3
3 3 1
2 -3 -1
toggle movement


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 2 2 0 0
0 1 1 1 0 0 0
0 1 0 0 0 1 0
0 0 2 0 1 1 0
0 0 2 2 0 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 1 2 2 2 0
0 2 0 1 0 2 0
0 2 0 2 0 1 0
0 2 1 2 0 2 0
0 1 2 0 0 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 2 1 1 1 0
0 1 0 0 1 1 0
0 0 2 2 0 2 0
0 0 0 1 0 1 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 0 1
0 -1 -1
w0[:,:,1]
-1 0 0
1 -1 0
-1 1 1
w0[:,:,2]
1 1 0
0 1 0
1 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 1
-1 0 1
0 0 0
w1[:,:,1]
1 -1 0
0 1 0
-1 1 1
w1[:,:,2]
-1 1 0
1 0 -1
0 0 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 2 4
1 0 4
3 4 4
o[:,:,1]
0 7 3
3 3 1
2 -3 -1
toggle movement

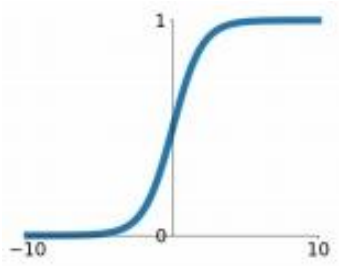
Training Convolutional Neural Networks

- Activation Functions
- Data Preprocessing
- Weight Initialization
- Batch Normalization
- Hyperparameter Optimization

Activation Functions

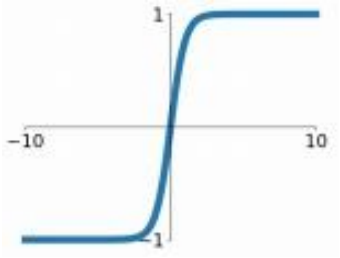
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



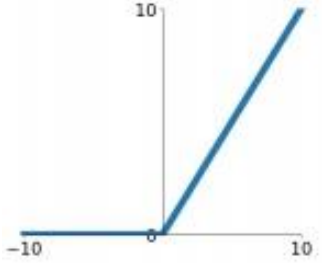
tanh

$$\tanh(x)$$



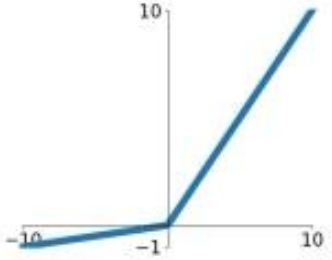
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

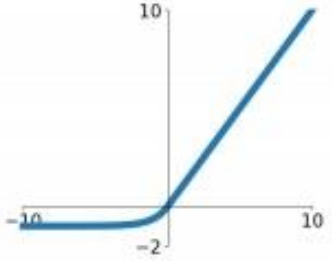


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



In practice:

- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU
- Try out tanh but don’t expect much
- Don’t use sigmoid

ReLU (Rectified Linear Units) Layers

- After each conv layer we apply a nonlinear layer (or **activation layer**) immediately afterward.
- The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (just element wise multiplications and summations).
- **ReLU layers** work far better because the network is able to train a lot faster (because of the computational efficiency) without making a significant difference to the accuracy.
- It also helps to alleviate the vanishing gradient problem, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers.
- The **ReLU layer** applies the function f(x) = max(0, x) to all of the values in the input volume. In basic terms, this layer just changes all the negative activations to 0.This layer increases the nonlinear properties of the model and the overall network without affecting the receptive fields of the convolution layer.

More on Augmentation

Sample augmentations

Why do We Need Image Augmentation?

- Image augmentation is a very powerful technique used to artificially create variations in existing images to expand an existing image data set. This creates new and different images from the existing image data set that represents a comprehensive set of possible images.
- This is done by applying different transformation techniques like zooming, rotating the existing image by a few degrees, shearing or cropping the existing set of images etc.

Why Do we Need Image Augmentation in DL?

- Deep learning Convolutional Neural Networks(CNN) need a huge number of images for the model to be trained effectively. **This helps to increase the performance of the model by generalizing better and thereby reducing overfitting**. Most popular data sets for classification and object detection data sets have a few thousand to millions of images.
- **CNN, due to its in-variance property, can classify objects even when visible in different sizes, orientations or different illumination.** Hence, we can take a small dataset of images and transform the objects to different sizes by zooming in or zooming out, flipping them vertically or horizontally or changing the brightness whatever makes sense for the object. This way we create a rich, diverse data set with variations.
- Image Augmentation creates a rich, diverse set of images from a small set of images for image classification, object detection or image segmentation.

When to Apply Image Augmentation?

Offline or pre-processing augmentation

- Augmentation is applied as a pre-processing step to increase the size of the data set. This is usually done when we have a small training data set that we want to expand.
- Generating augmentation on smaller data set is helpful but we need to consider the disk space when applying on larger data sets.
- In Offline augmentation, augmented image is part of the training set, it can view the augmented image multiple times depending on the number of epochs.

Online or real-time augmentation

- As the name suggests, augmentation is applied in real time. This is usually applied for larger data sets as we do not need to save the augmented images on the disk.
- In this case, we apply transformations in mini-batches and then feed it to the model.
- Online augmentation model will see different images at each epoch. The model generalizes better with online augmentation as it sees more samples during training compared with offline data augmentation.

Basic Data Augmentation Techniques

- **Flipping**: flipping the image vertically or horizontally.
- **Rotation**: rotates the image by a specified degree.
- **Shearing**: shifts one part of the image like a parallelogram.
- **Cropping**: object appears in different positions, or in different proportions in the image.
- **Zoom in, Zoom out.**
- **Changing brightness or contrast**

Displaying the Image After Augmentation (1)

Original Image

Adding Gaussian noise

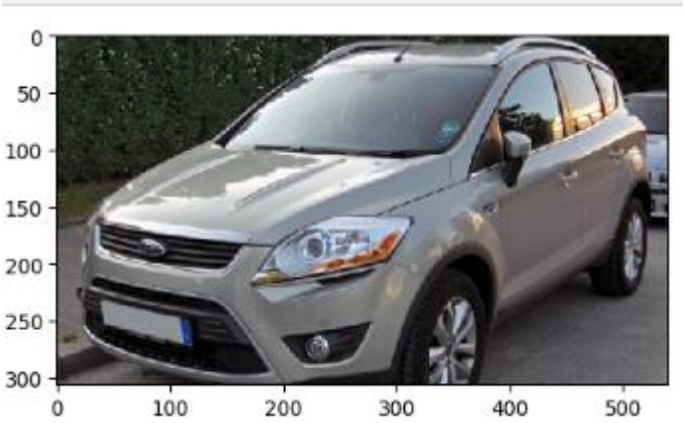
Image can be rotated

Cropped one side of the image
In this case by 30%

Displaying the Image After Augmentation (2)

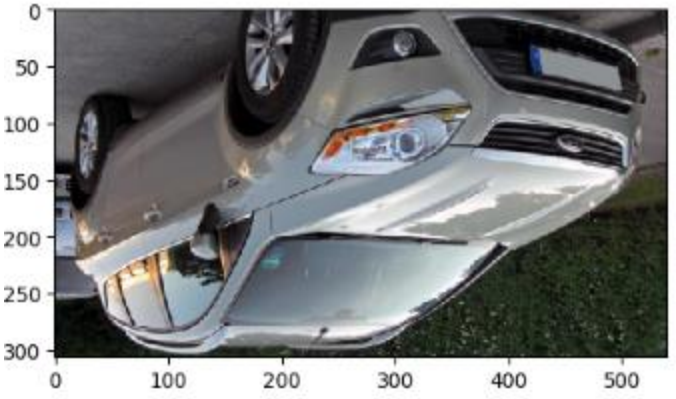
Original Image

Flipping image horizontally



Shearing image

Flipping image vertically



Displaying the Image After Augmentation (3)

Original Image

Scaling image of its height/width

Adding contrast to the image

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Sun 05 June 2016

By Francois Chollet

In Tutorials.

- training a small network from scratch (as a baseline);
- using the bottleneck features of a pre-trained network;
- fine-tuning the top layers of a pre-trained network.

Data Augmentation with Keras (1)

```
from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

These are just a few of the options available (for more, see the documentation)

Data Augmentation with Keras (2)

- *rotation_range* is a value in degrees (0-180), a range within which to randomly rotate pictures;
- *width_shift and height_shift* are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally;
- *rescale* is a value by which we will multiply the data before any other processing. Our original images consist in RGB coefficients in the 0-255, but such values would be too high for our models to process (given a typical learning rate), so we target values between 0 and 1 instead by scaling with a 1/255. factor;
- *shear_range* is for randomly applying shearing transformations

Data Augmentation with Keras (3)

- *zoom_range* is for randomly zooming inside pictures;
- *horizontal_flip* is for randomly flipping half of the images horizontally --relevant when there are no assumptions of horizontal assymetry (e.g. real-world pictures);
- *fill_mode* is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift.

Dogs vs Cats Dataset (1)

- The code below is our first model, a simple stack of 3 convolution layers with a ReLU activation, followed by max-pooling layers. This is very similar to the architectures that Yann LeCun proposed in the 1990s for image classification (with the exception of ReLU).
- The full code for this experiment can be found here.

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(3, 150, 150)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# the model so far outputs 3D feature maps (height, width, features)
```

Dogs vs Cats Dataset (2)

- On top of it we stick two fully-connected layers. We end the model with a single unit and a sigmoid activation, which is perfect for a binary classification.
- To go with it we will also use the *binary_ crossentropy* loss to train our model.

```
model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

Use .flow_from_directory( ) to generate batches of image data (and their labels)

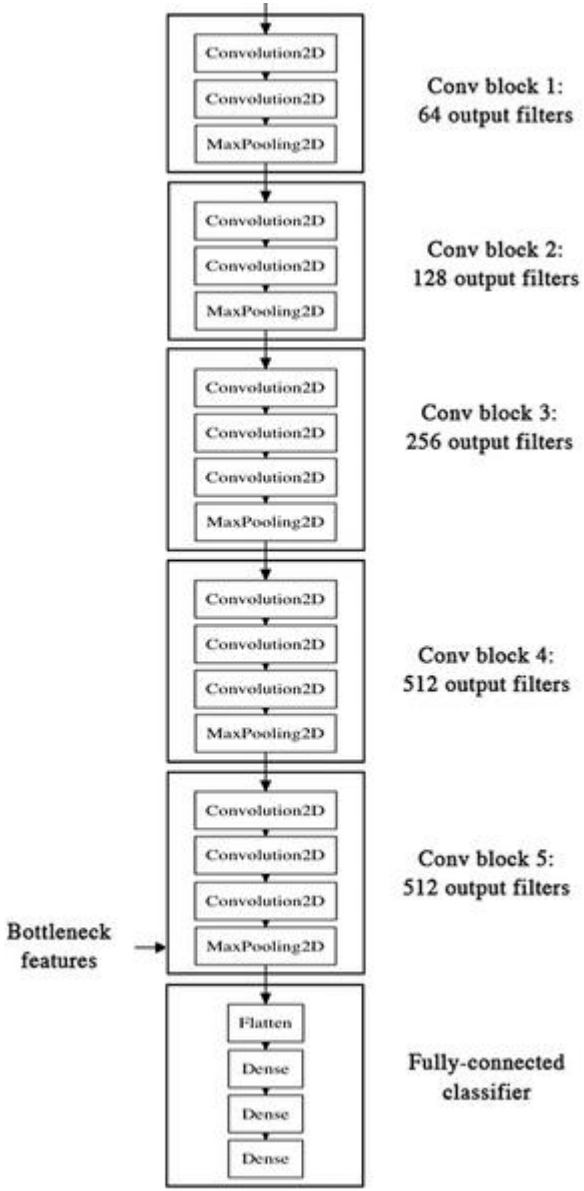
```
batch_size = 16

# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

# this is the augmentation configuration we will use for testing:
# only rescaling
test_datagen = ImageDataGenerator(rescale=1./255)

# this is a generator that will read pictures found in
# subfolers of 'data/train', and indefinitely generate
# batches of augmented image data
train_generator = train_datagen.flow_from_directory(
        'data/train',  # this is the target directory
        target_size=(150, 150),  # all images will be resized to 150x150
        batch_size=batch_size,
        class_mode='binary')  # since we use binary_crossentropy loss, we need binary labels

# this is a similar generator, for validation data
validation_generator = test_datagen.flow_from_directory(
        'data/validation',
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')
```

Dogs vs Cats Dataset (3)

- Now we can train our model. Each epoch takes 20-30s on GPU and 300-400s on CPU.
- This approach gets us to a validation accuracy of 0.79-0.81 after 50 epochs (a number that was picked arbitrarily --because the model is small and uses aggressive dropout, it does not seem to be overfitting too much by that point).

```
model.fit_generator(
        train_generator,
        steps_per_epoch=2000 // batch_size,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=800 // batch_size)
model.save_weights('first_try.h5')  # always save your weights after training or during training
```

Using the Bottleneck Features of a Pre-trained Neural Network



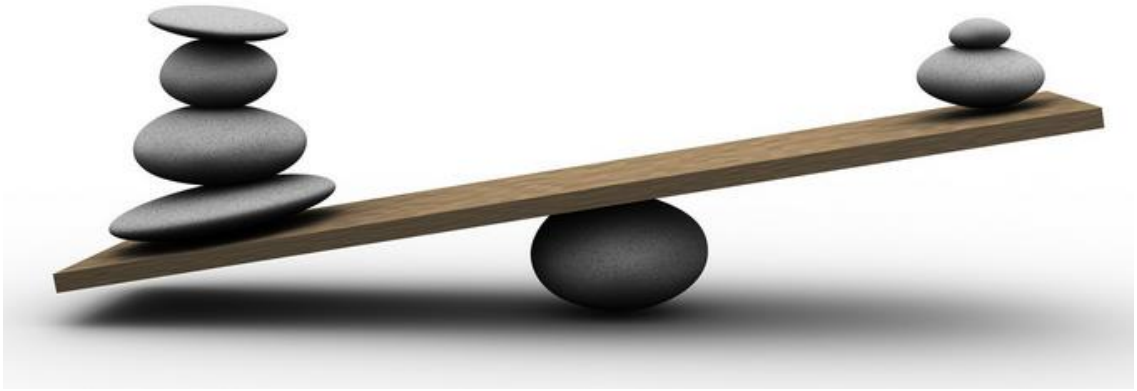
- We will use the VGG16 architecture, pre-trained on the ImageNet dataset --a model previously featured on this blog. Because the ImageNet dataset contains several "cat" classes (persian cat, siamese cat...) and many "dog" classes among its total of 1000 classes, this model will already have learned features that are relevant to our classification problem.
- You can find the full code for this experiment here.

https://gist.github.com/fchollet/7eb39b44eb9e16e59632d25fb3119975

- You can get the weights file from Github.

https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3

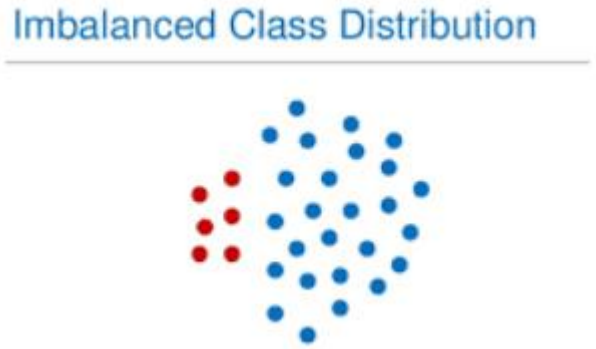
Imbalanced Datasets



What are Balanced and Imbalanced Datasets?

Balanced Dataset: — Let's take a simple example if in our data set we have positive values which are approximately same as negative values. Then we can say our dataset in balance
Consider Orange color as a positive values and Blue color as a Negative value. We can say that the number of positive values and negative values in approximately same.

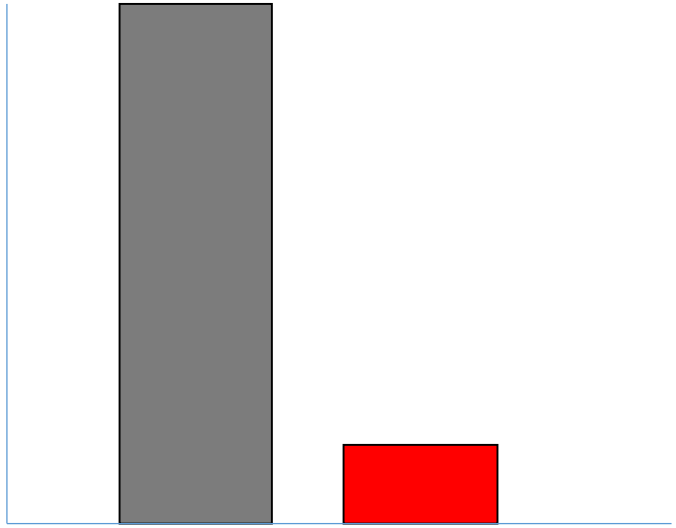
Balance Dataset



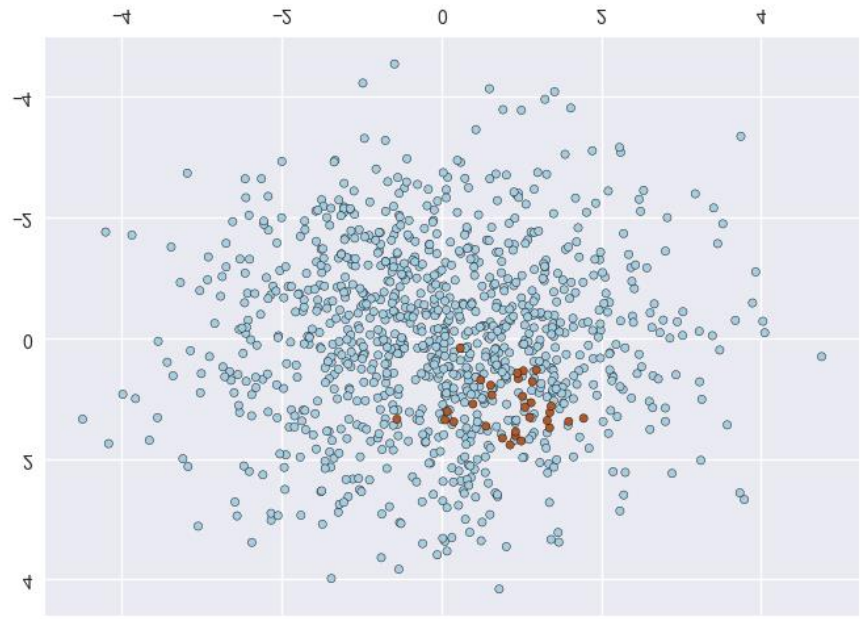
Imbalanced Dataset: — If there is the very high different between the positive values and negative values. Then we can say our dataset in Imbalance Dataset.

What Imbalance Means?

- **Imbalance:** number of examples belonged to a class is significantly greater than those of the others.



Majority (Negative) class Minority (Positive) class



What is Imbalanced Dataset

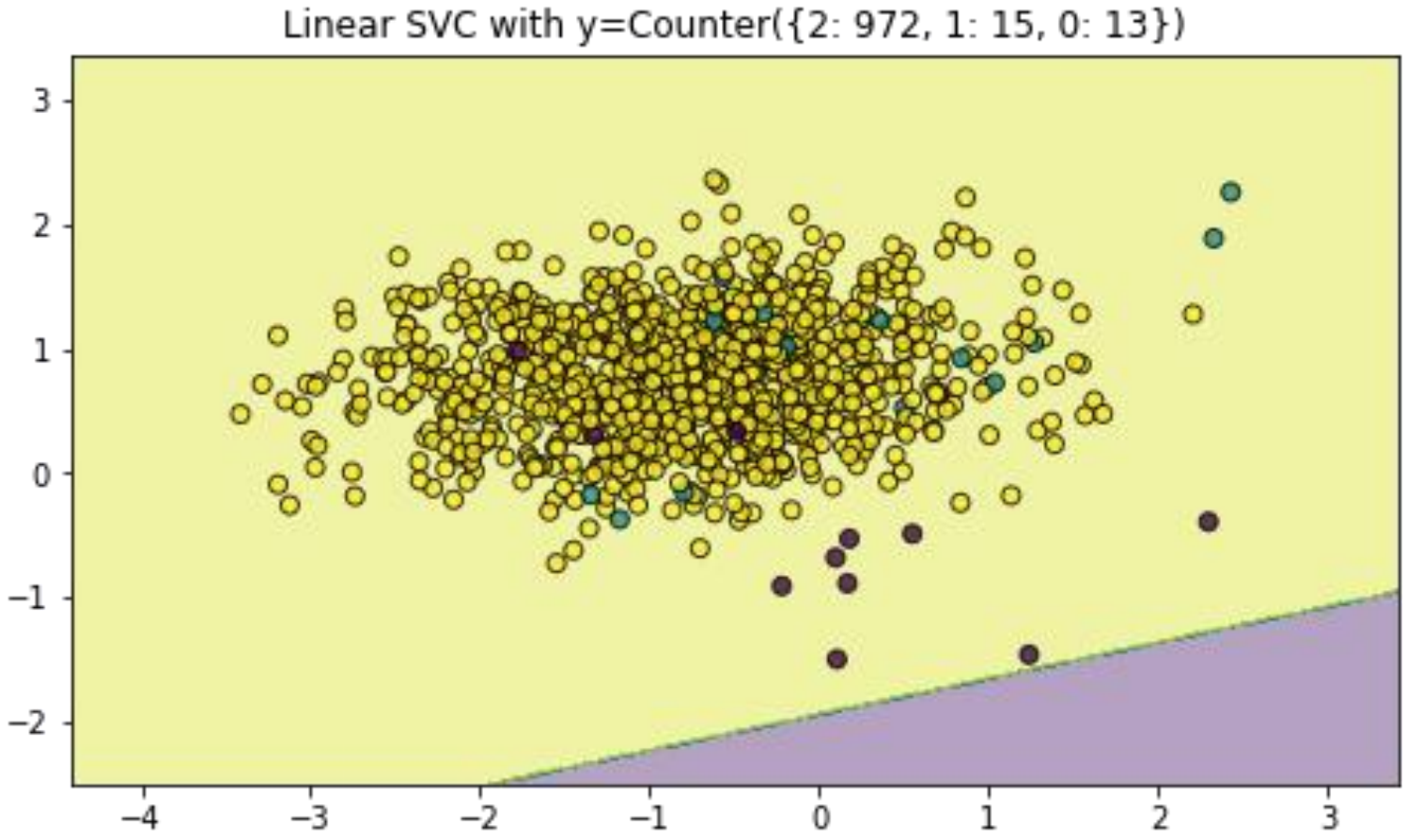
- Classification is one of the most common machine learning problems. The best way to approach any classification problem is to start by analyzing and exploring the dataset in what we call **E**xploratory **D**ata **A**nalysis **(EDA)**.
- The sole purpose of **EDA** is to generate as many insights and information about the data as possible. It is also used to find any problems that might exist in the dataset. One of the common issues found in datasets that are used for classification is **imbalanced classes** issue.
- Data imbalance usually reflects an unequal distribution of classes within a dataset. For example, in a credit card fraud detection dataset, most of the credit card transactions are not fraud and a very few classes are fraud transactions.

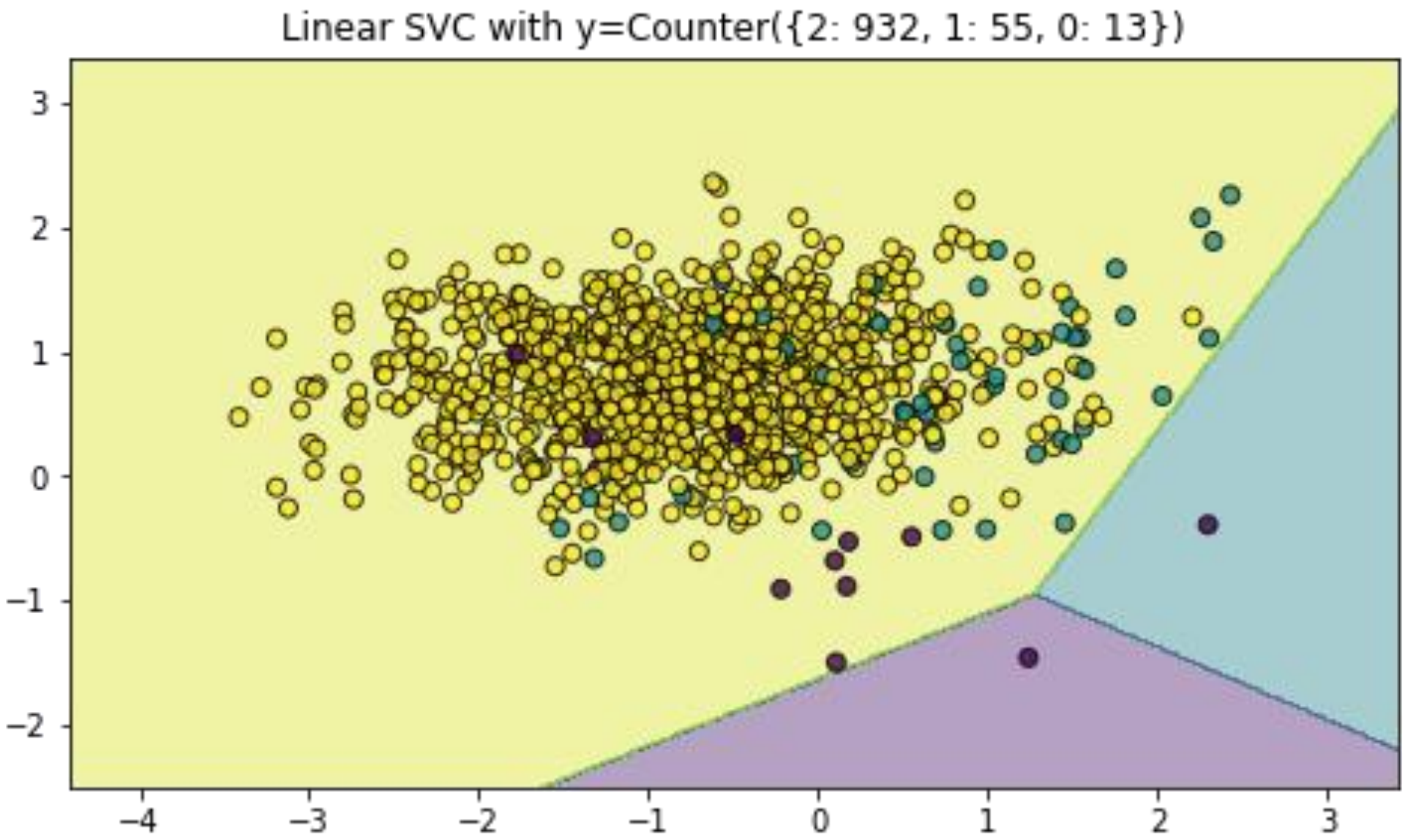
Why Do We Care?

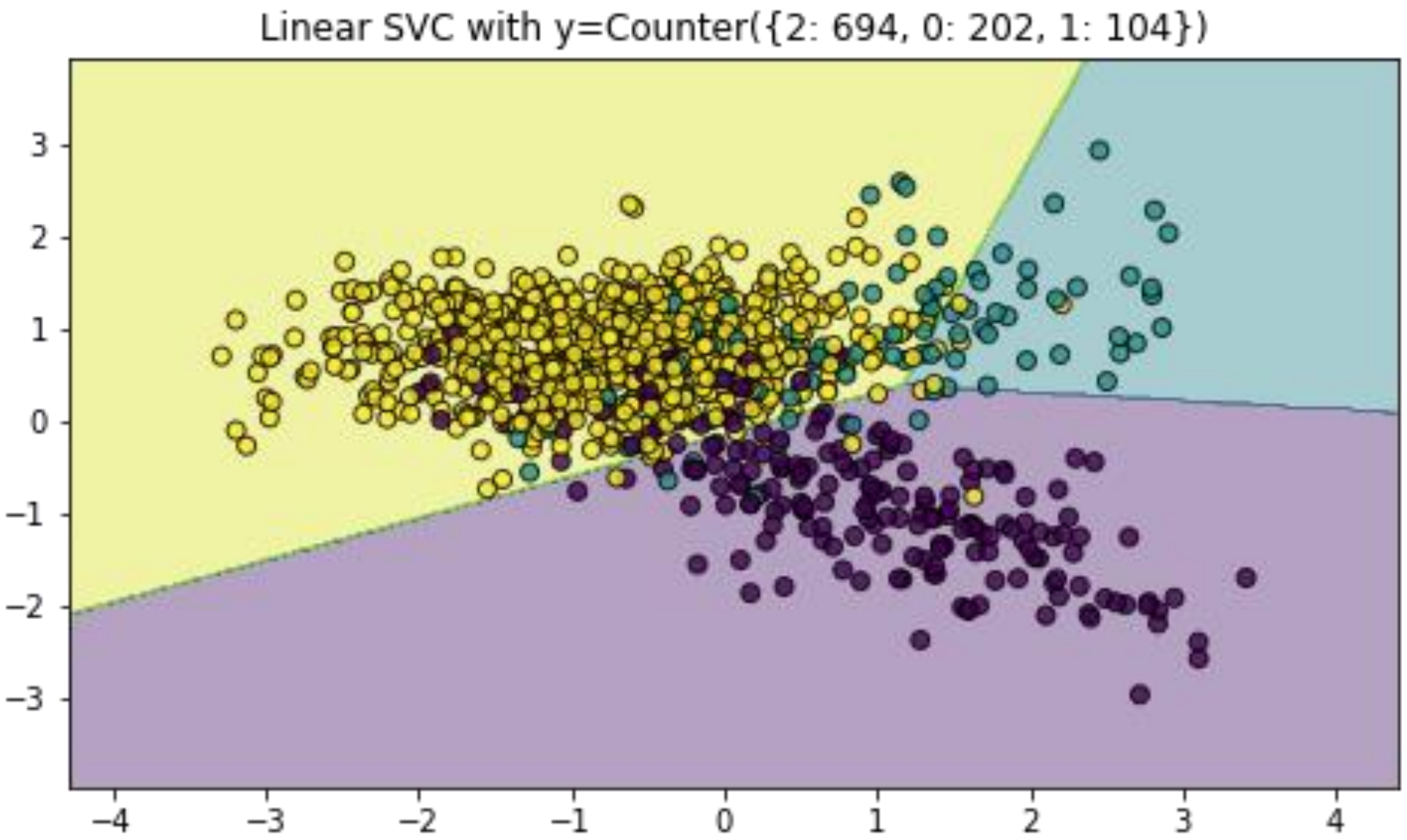
- Standard algorithms have ***poor performance*** on imbalanced data
- Minimizing global error rate without taking data distribution into consideration will cause ***performance bias***
- ***Poor accuracy*** on *minority class* and ***high accuracy*** on *majority class*
- Correctly classification of ***minority*** class examples are more important than those of majority class

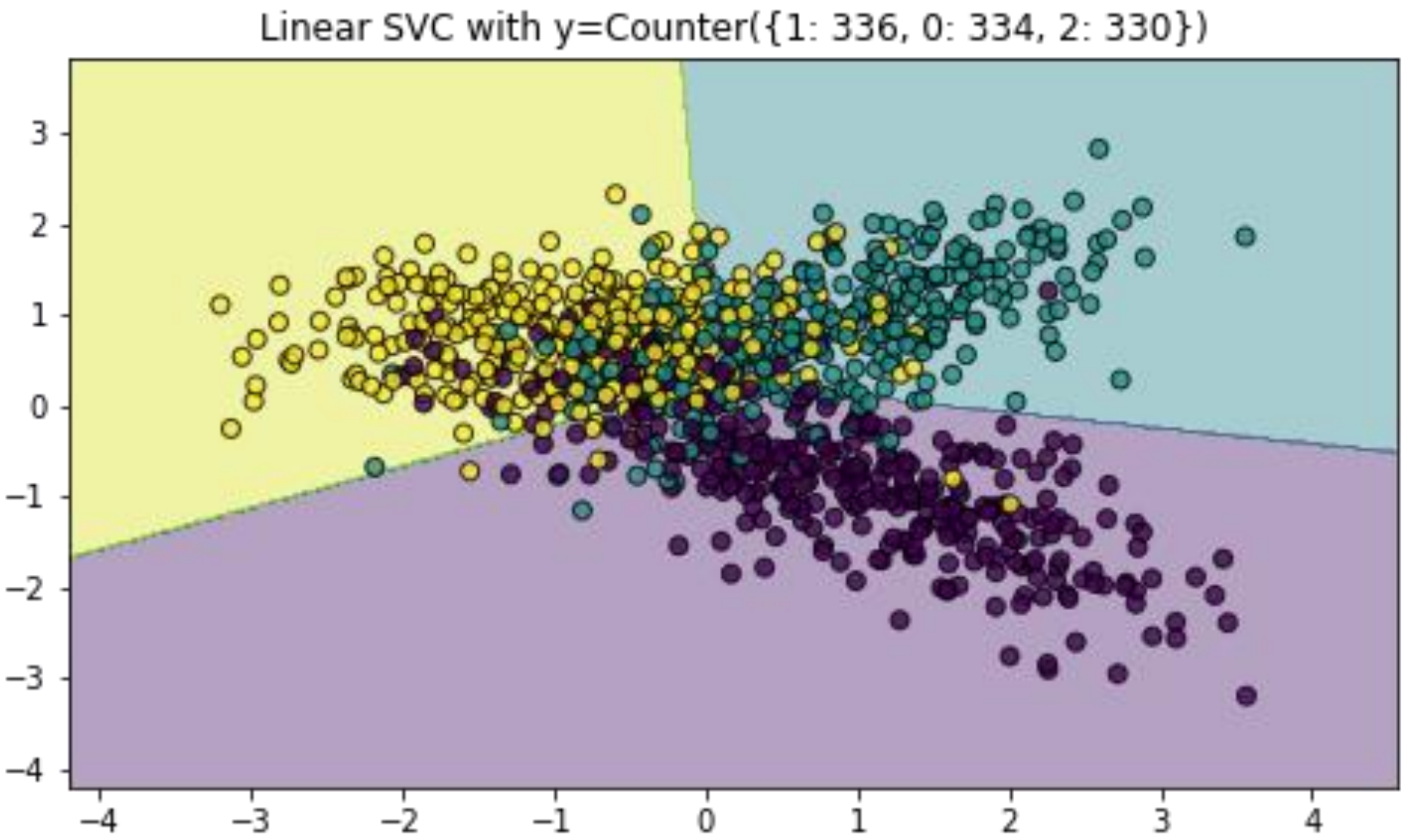
Linear Support Vector Classifier (SVC)

The objective of a Linear SVC (Support Vector Classifier) is to fit to the data we provide, returning a "best fit" hyperplane that divides, or categorizes, our data. From there, after getting the hyperplane, we can then feed some features to the classifier to see what the "predicted" class is.

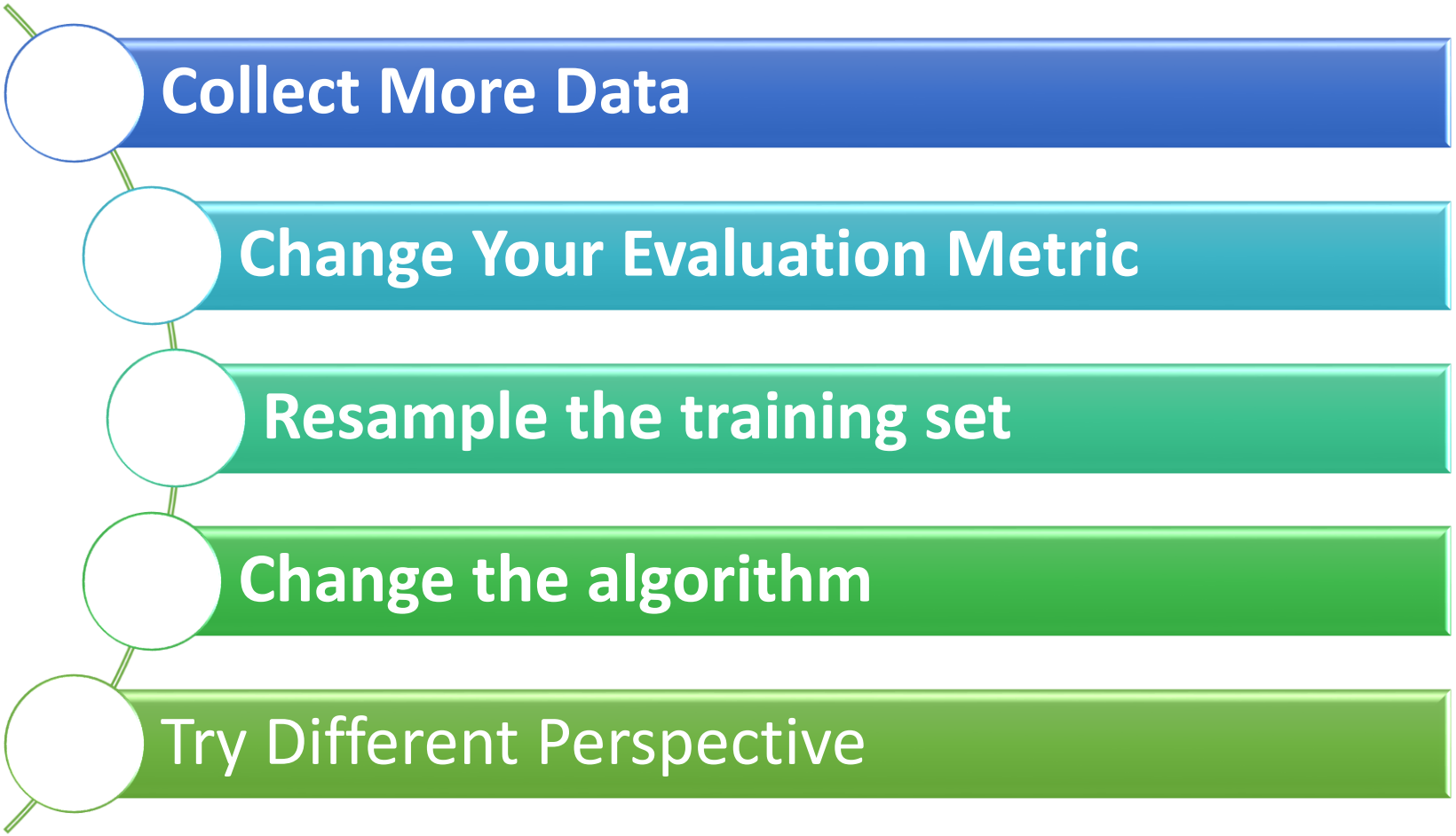



Linear SVC with y=Counter({2: 932, 1: 55, 0: 13})
3
2
1
0
−1
−2
−4
−3
−2
−1
0
1
2
3


Linear SVC with y=Counter({2: 694, 0: 202, 1: 104})
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3
4


Linear SVC with y=Counter({1: 336, 0: 334, 2: 330})
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3
4

How to Address the Issue of Imbalance?

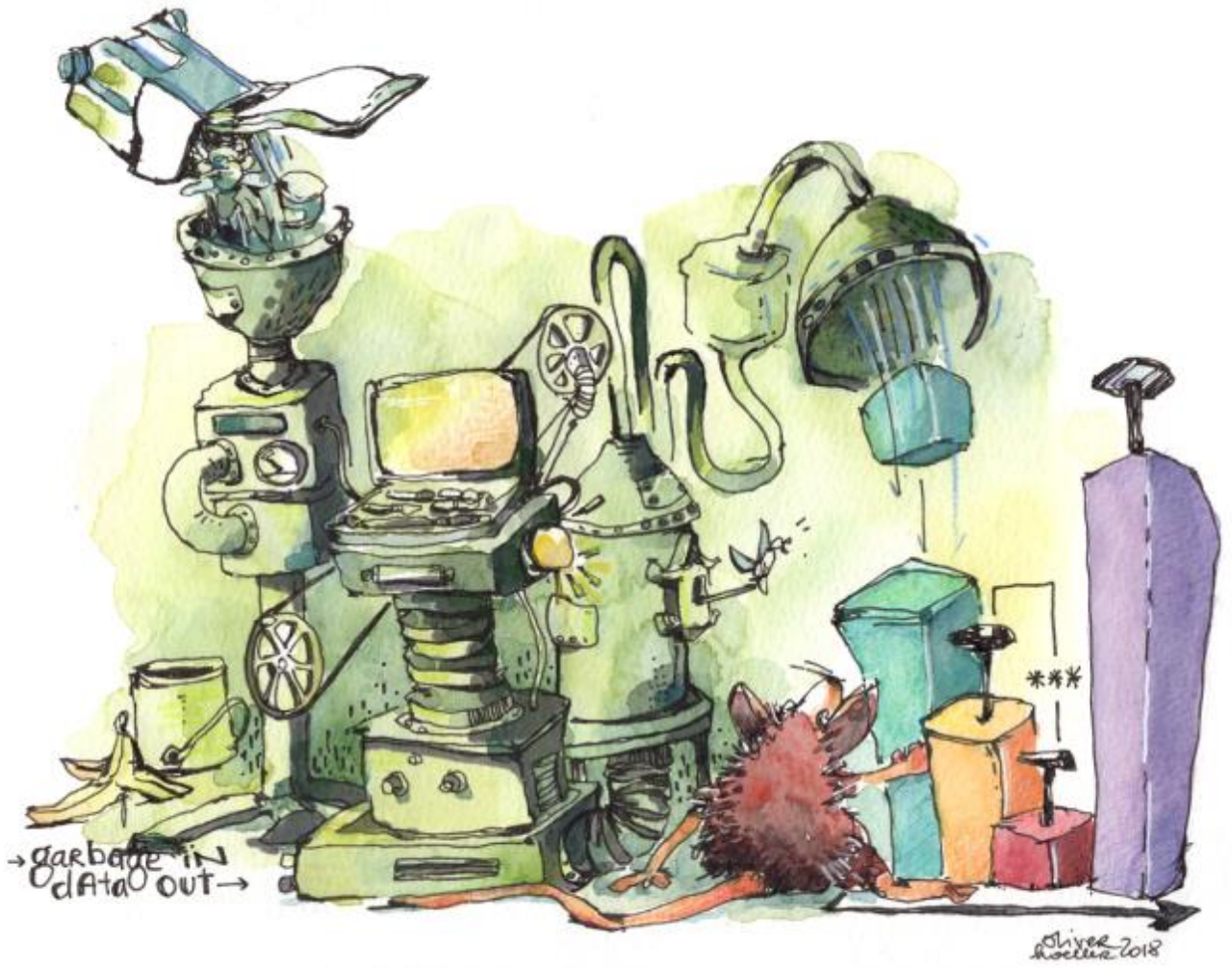


Tactics To Combat Imbalanced Training Data

- Can You Collect **More Data**?
- Look at your **Performance Metric**: *confusion matrix*, *correlation coefficient*, *precision*, *recall*, *F1 Score*, *ROC curve* and *AUC (***Area under the ROC Curve.**
- **Resample** your dataset.
- **Ensemble** different resampled datasets.
- **Resample** with different ratios

Collect More Data

GIGO:
Garbage in =
Garbage out



The quality of information coming out of model cannot be better than the quality of information that went in!!!

Evaluation Metrics

$$Accuracy = \frac{TP + TN}{(\mathrm{TP} + FN + FP + TN)}$$

$$F_{value} = \frac{(1+\beta^2)\,\mathrm{Re}\,call * \mathrm{Pr}\,ecision}{\beta^2\,\mathrm{Re}\,call + \mathrm{Pr}\,ecision}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$FP\ rate = \frac{FP}{TN + FP}$$

		Actual Values: Positive (1)	Actual Values: Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Resample the Training Set

- ***Oversampling***: adding copies of instances from the under represented class.

  Over-sampling increases the number of minority class members in the training set. The advantage of over-sampling is that no information from the original training set is lost, as all observations from the minority and majority classes are kept.

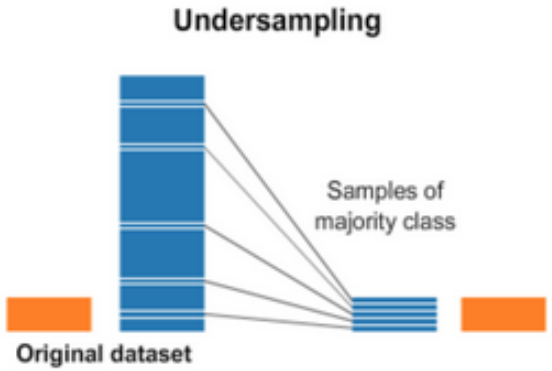
  On the other hand, it is prone to over fitting.

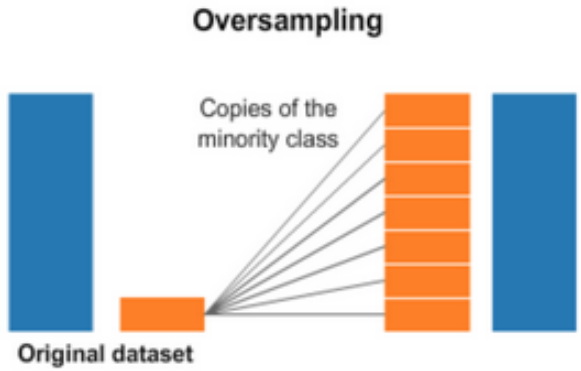
- ***Undersampling***: deleting instances from the over represented class.

  Under-sampling, on contrary to over-sampling, aims to reduce the number of majority samples to balance the class distribution. Since it is removing observations from the original data set, it might discard useful information.

Resampling (Oversampling and Undersampling)

- **Undersampling** is the process where you randomly delete some of the observations from the majority class in order to match the numbers with the minority class.
- **Oversampling** is a process of generating synthetic data that tries to randomly generate a sample of the attributes from observations in the minority class. There are a number of methods used to oversample a dataset for a typical classification problem. The most common technique is called **SMOTE (**Synthetic Minority Over-sampling Technique).





Oversampling - Overview

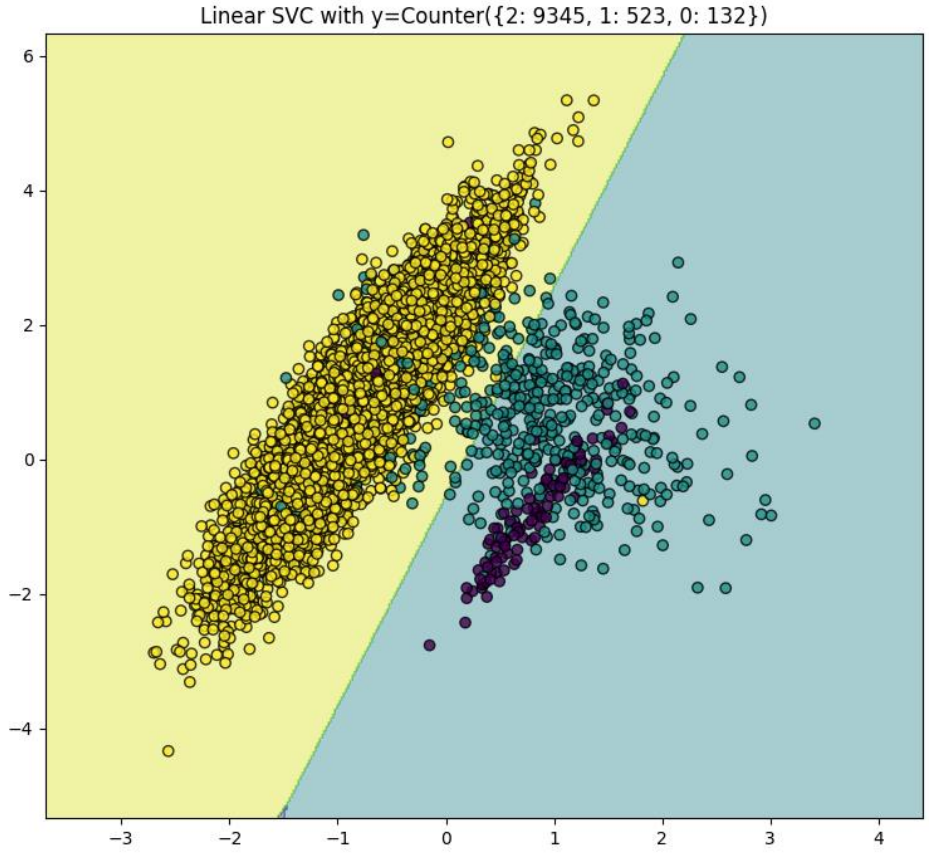
- Random oversampling

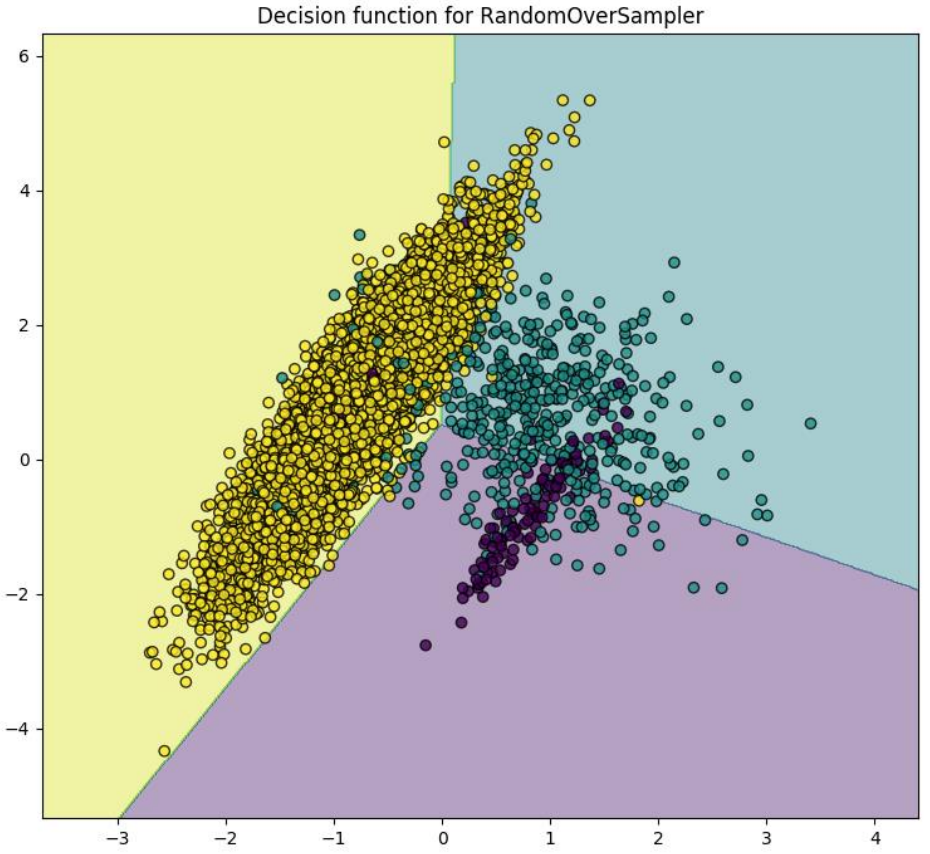
- SMOTE: Synthetic Minority Oversampling Techniques

Random Oversampling

- Simply replicates randomly the minority class examples.
- Advantages:
  - This method leads to no information loss.
- Drawbacks:
  - Increasing the likelihood of overfitting since it replicates the minority class events.
  - Increasing the training time.

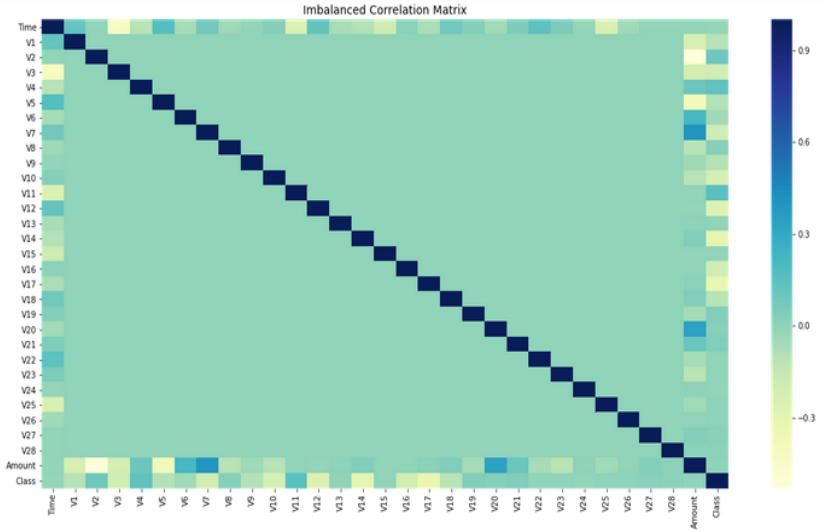
Random Oversampling - Results



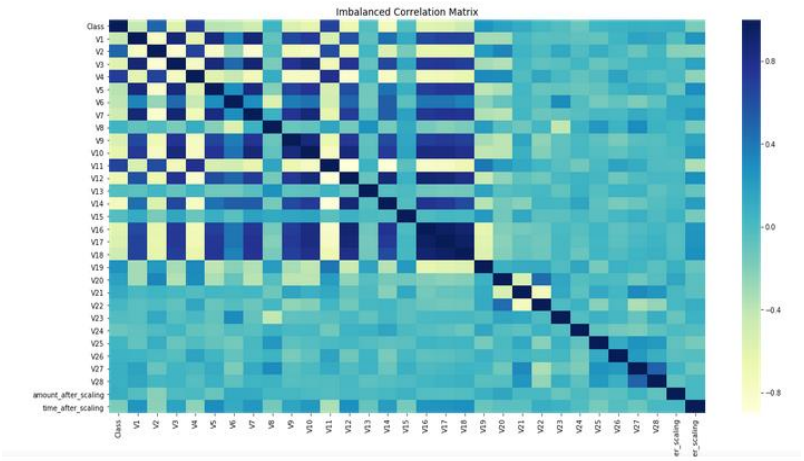


Imbalanced Correlation Matrix and After Resampling

Before fixing the imbalance problem, most of the features did not show any correlation which would definitely have impacted the performance of the model.



After Resampling:

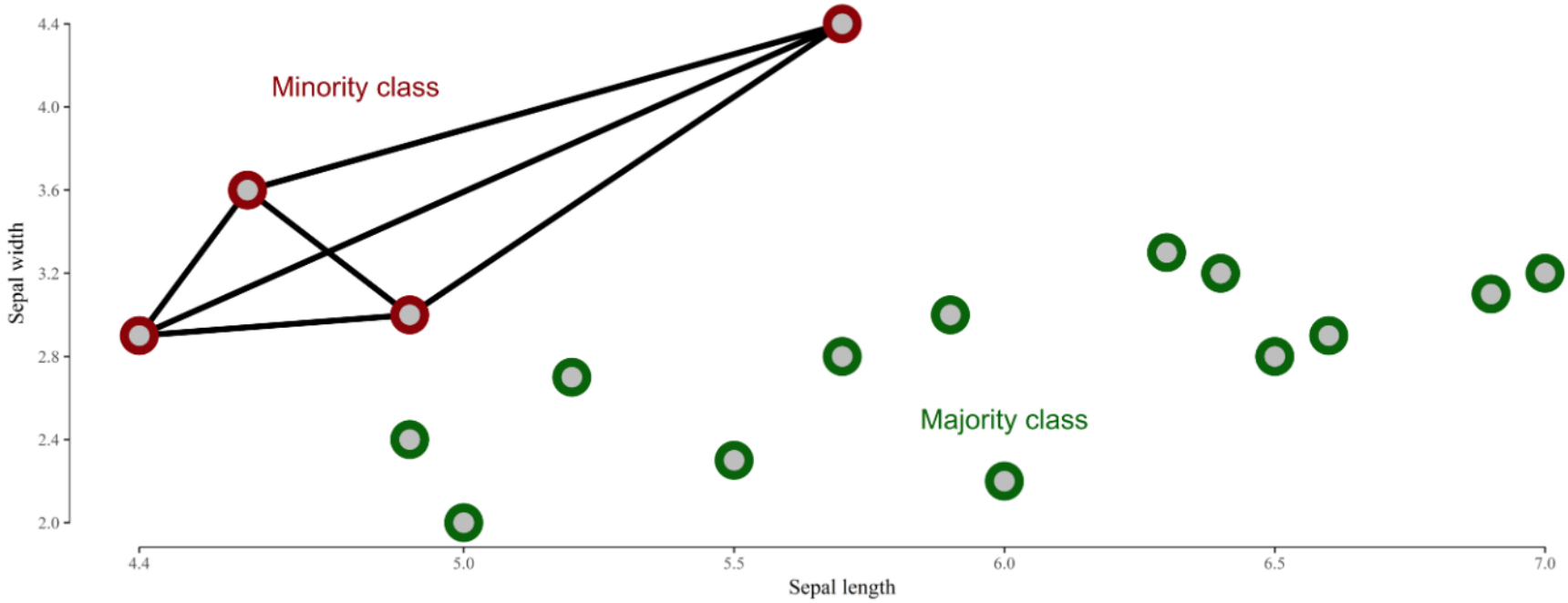


SMOTE - Synthetic Minority Oversampling Techniques

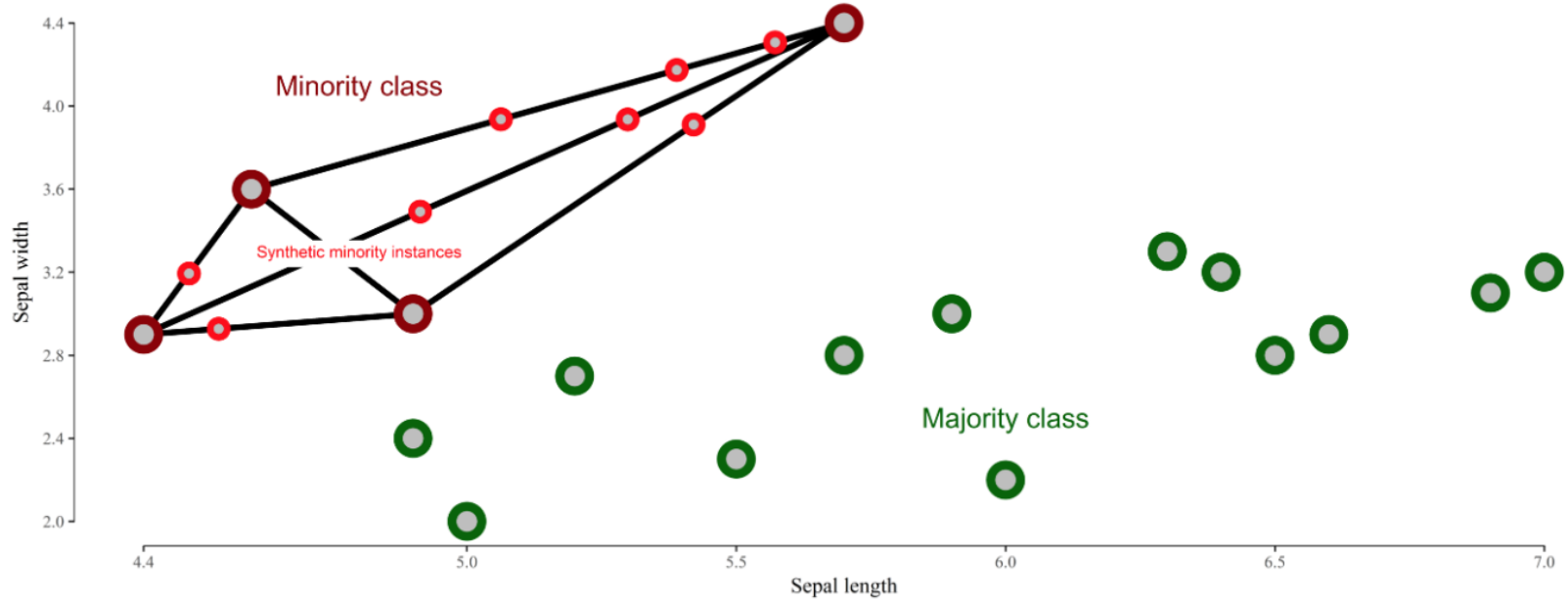
- SMOTE thinks from the perspective of existing minority instances and synthesizes new instances at some distance from them towards one of their neighbors.

- SMOTE loops through the existing, real minority instance. At each loop iteration, one of the K closest minority class neighbors is chosen and a new minority instance is synthesized somewhere between the minority instance and its neighbor.

Addressing class imbalance problems of ML via SMOTE: connecting the dots



Addressing class imbalance problems of ML via SMOTE: synthesising new dots between existing dots

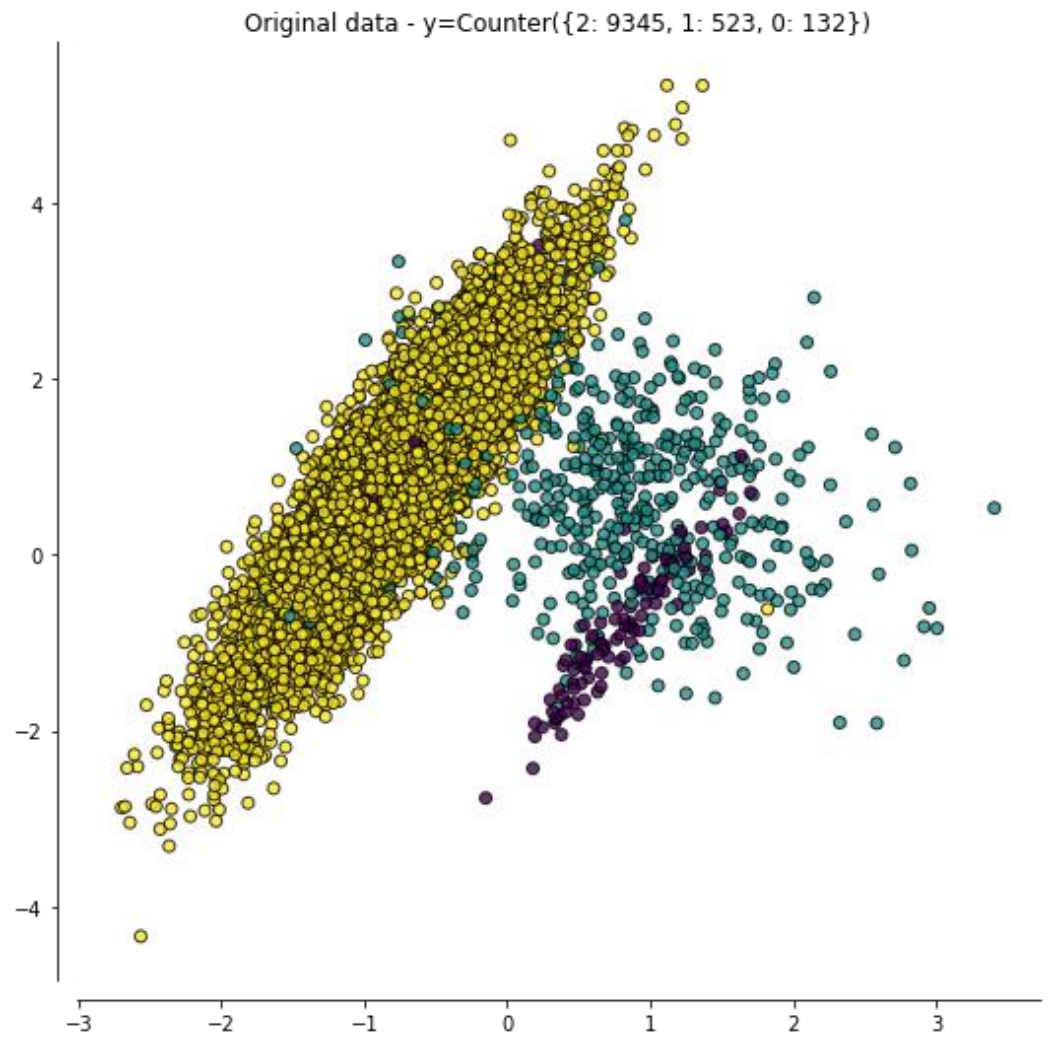


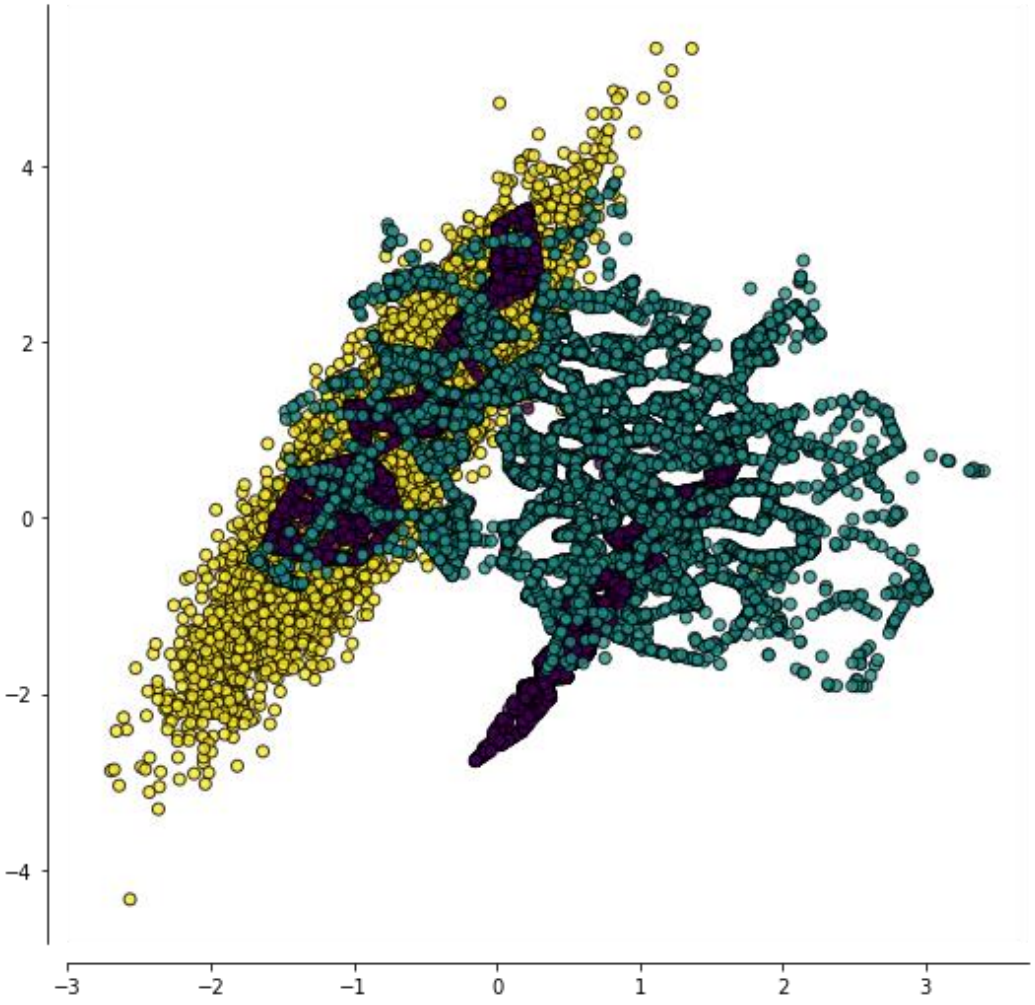
SMOTE - Synthetic Minority Oversampling Techniques

- Advantages:
  - Alleviates overfitting caused by random oversampling as synthetic examples are generated rather than replication of instances.
  - No loss of information.
  - It's simple to implement and interpret.

- Disadvantages:
  - While generating synthetic examples, SMOTE does not take into consideration neighboring examples can be from other classes. This can increase the overlapping of classes and can introduce additional noise.
  - SMOTE is not very practical for very high dimensional data.

SMOTE - Results





Undersampling - Overview

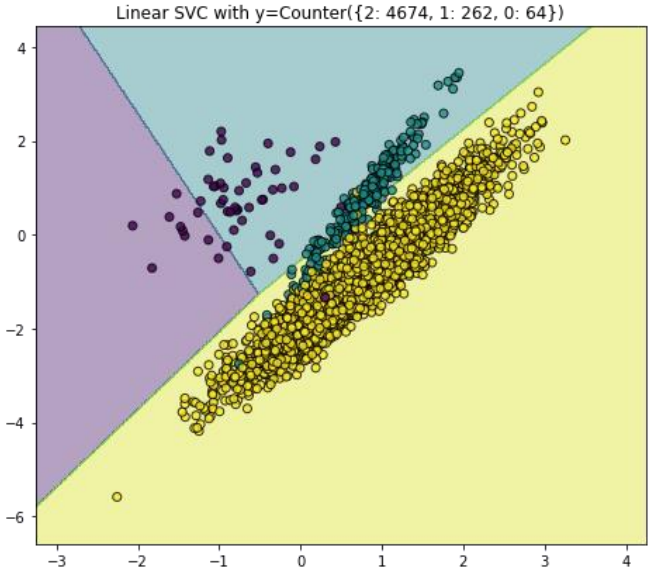
- Random under sampling
- NearMiss: NearMiss-1, NearMiss-2, NearMiss-3

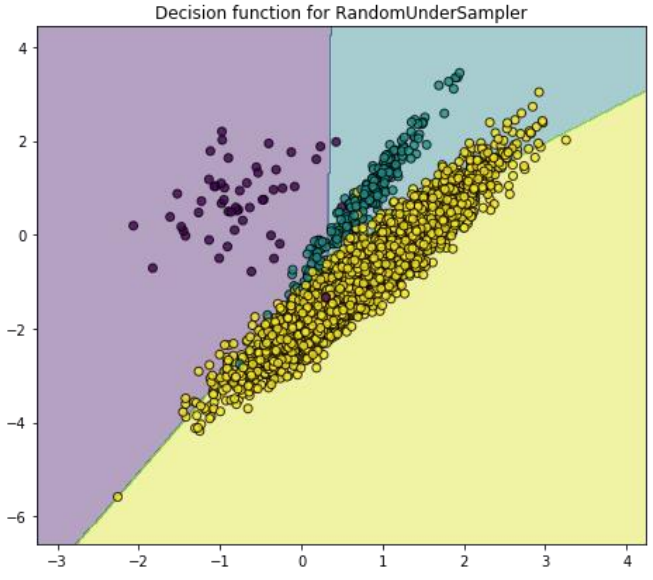
Random Undersampling

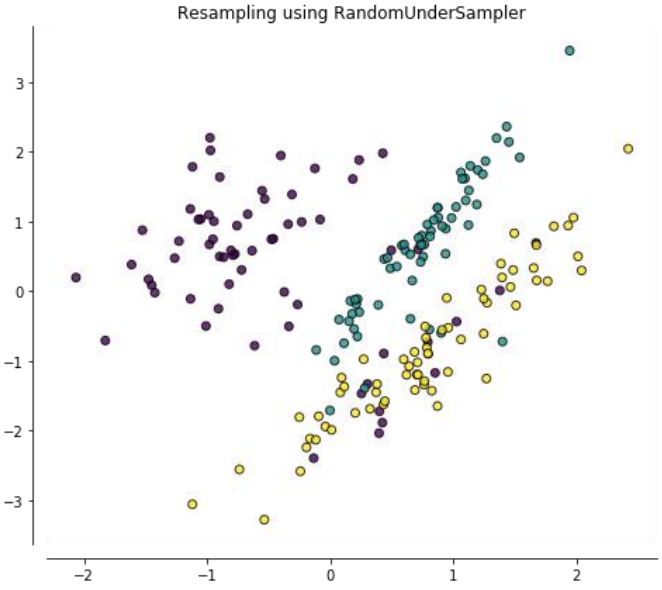
- Random under-sample the majority class by randomly picking samples with or without replacement.

- Disadvantages:
  - It can discard useful information about the data itself.
  - The chosen sample may be a biased and inaccurate representation of the population. Therefore, it can cause the classifier to perform poorly on real unseen data.
- Advantages:
  - Improving the runtime of the model and solving the memory problems by reducing the number of training data samples.

Random Undersampling - Results







NearMiss

- For each point in the majority class, computes its distances to every points in the minority class, and uses the k nearest neighbor algorithm.

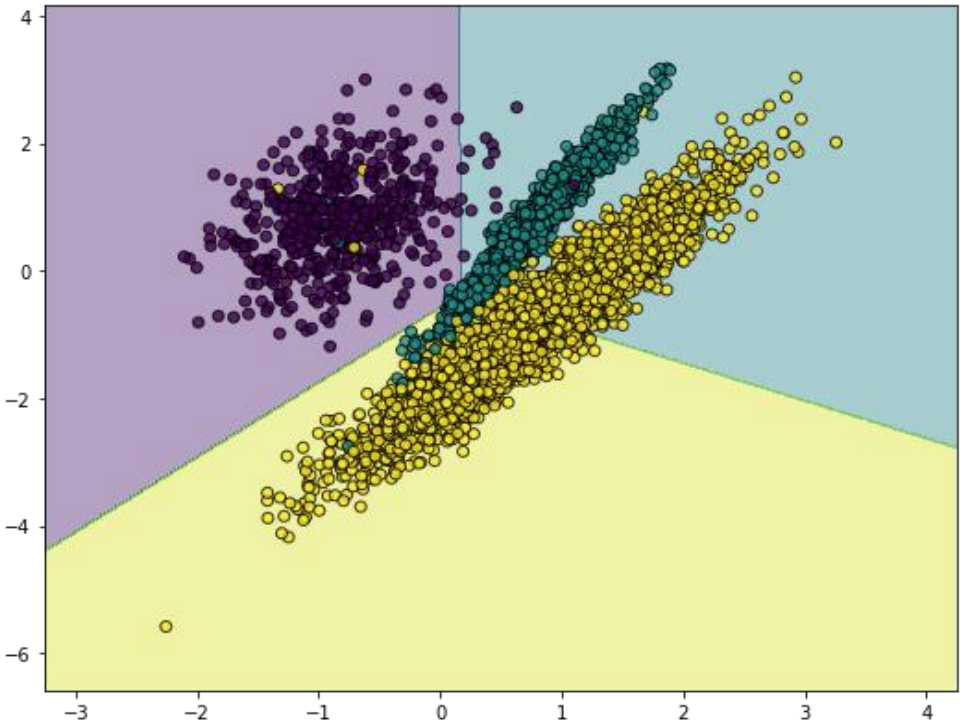
NearMiss – version 1, 2 & 3

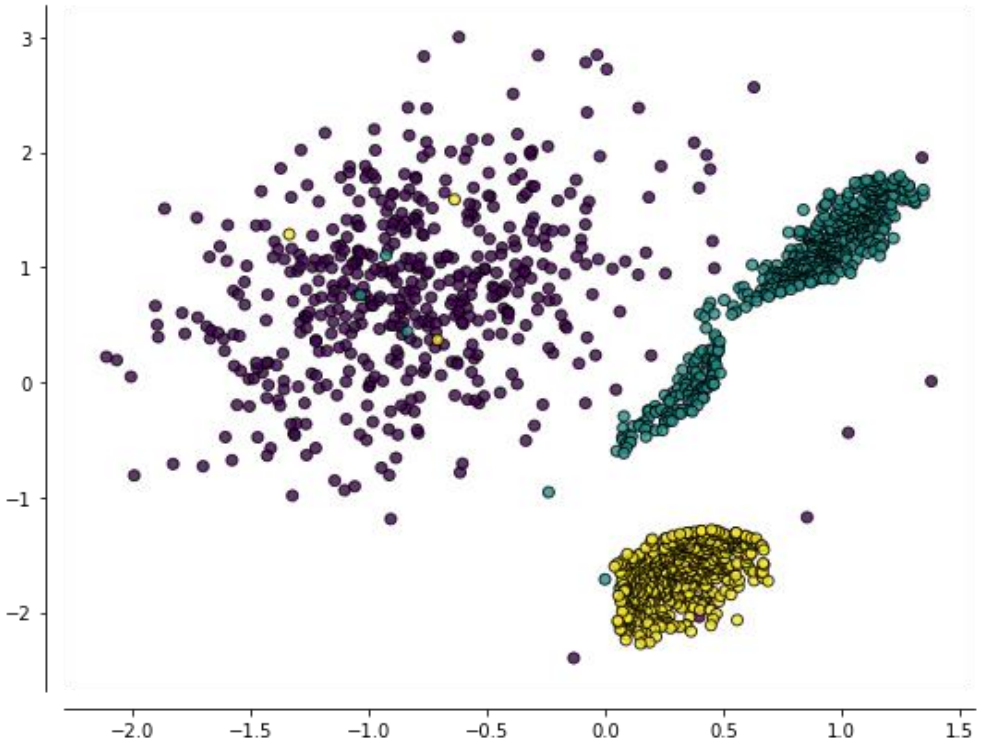
NearMiss – Version 1 : It selects samples of the majority class for which average distances to the k closest instances of the minority class is smallest.

NearMiss – Version 2 : It selects samples of the majority class for which average distances to the k farthest instances of the minority class is smallest.

NearMiss – Version 3 : For each minority class instance, their M nearest-neighbors in majority class will be selected.

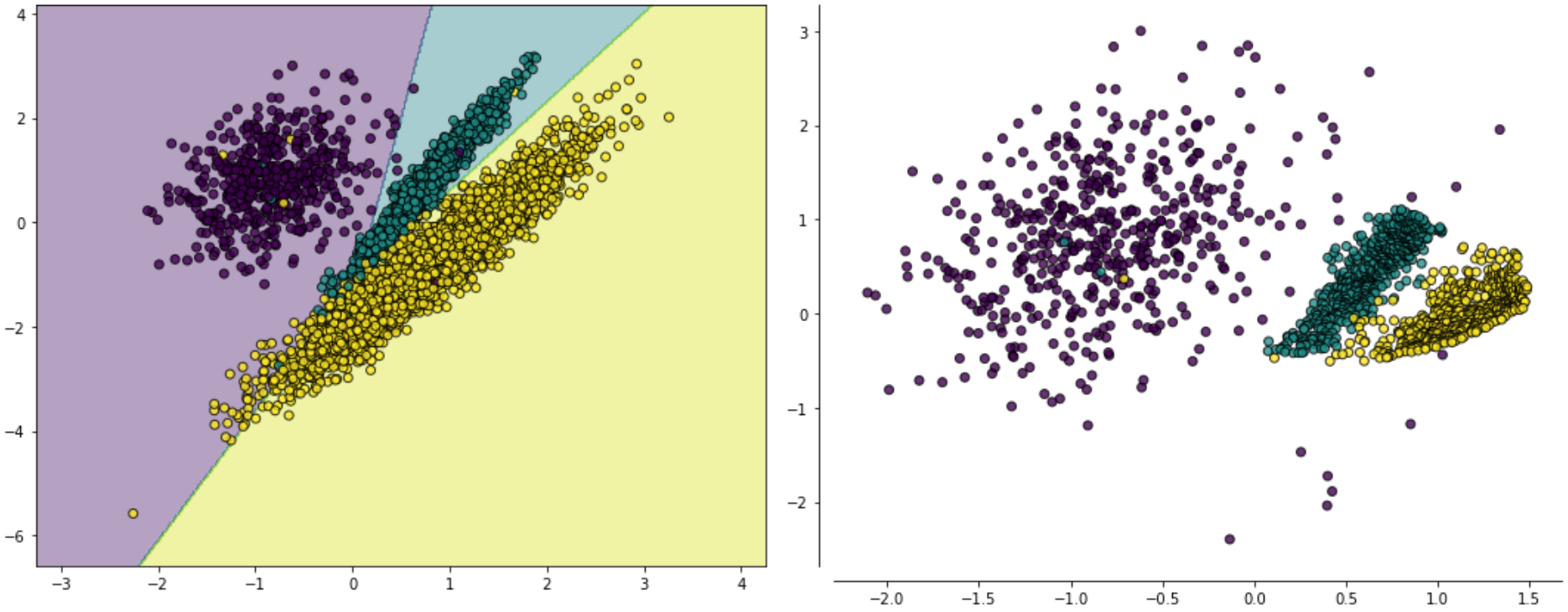
NearMiss 1– Results





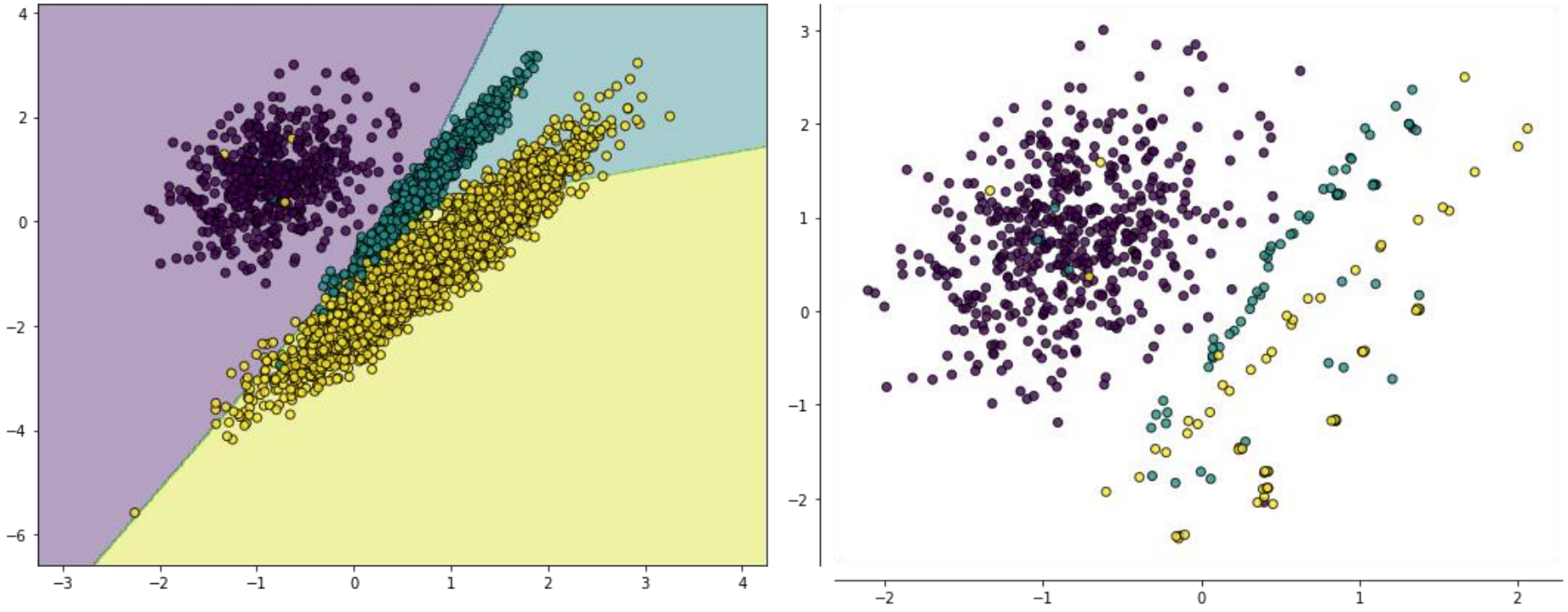
NearMiss – Version 1 : It selects samples of the majority class for which average distances to the k closest instances of the minority class is smallest.

NearMiss 2– Results



NearMiss – Version 2 : It selects samples of the majority class for which average distances to the k farthest instances of the minority class is smallest.

NearMiss 3– Results



NearMiss – Version 3 : For each minority class instance, their M nearest-neighbors in majority class will be selected.

Use Different Algorithms

- Decision tree:
  - Decision trees often perform well on imbalanced datasets.
  - The splitting rules that look at the class variable used in the creation of the trees, can force both classes to be addressed.
  - Examples: C4.5, C5.0, CART, and Random Forest.

Use Penalized Models

- Try to pay more attention to the minority class.
  - Penalized classification imposes an additional cost on the model for making classification mistakes on the minority class during training.

- Penalized version of algorithms
  - penalized-SVM (support vector machine), etc.
- Cost-Sensitive Learning
  - It takes the misclassification costs into consideration.
  - VS Cost-insensitive learning: cost-sensitive learning treats the different misclassifications differently but cost-insensitive learning does not.

Penalized-SVM

- Increasing the penalty for misclassifying minority classes to prevent them from being “overwhelmed” by the majority class.
  - In SVM, C is a hyper-parameter determining the penalty for misclassifying an observation. Penalized-SVM is to weight C by classes, so that $C_j = C * W_j$
    - where C is the penalty for misclassification, W_j is a weight inversely proportional to class j’s frequency, and C_j is the C value for class j.

Cost Sensitive Classifier

- The Cost-Sensitive Learning (CSL) takes the misclassification costs into consideration by minimizing the total cost. The goal of this technique is mainly to pursue a high accuracy of classifying examples into a set of known classes. It is playing as one of the important roles in the machine learning algorithms including the real-world data mining applications.
- Two methods can be used to introduce cost-sensitivity:
  - reweighting training instances according to the total cost assigned to each class.
  - predicting the class with minimum expected misclassification cost (rather than the most likely class).

Cost Sensitive Classifier

		Actual class	
		Positive class	Negative class
Predicted class	Positive class	True positive (TP) $C(+,+)$	False positive (FP) $C(+,-)$
	Negative class	False negative (FN) $C(-,+)$	True negative (TN) $C(-,-)$

Model M	Predicted class		
		+	-
Actual class	+	150	40
	-	60	250

	Predicted class		
		+	-
Actual class	+	-1	100
	-	1	0

Total cost = -1·150 + 100·40 + 1·60 + 0·250 = 3910

Cost Sensitive Classifier Based on Bayes Conditional Risk

THE UNIVERSITY OF RHODE ISLAND

Cost-Sensitive methods

Cost-Sensitive Learning Framework

- Define the cost of misclassifying a majority to a minority as $C(Min, Maj)$
- Typically $C(Maj, Min) > C(Min, Maj)$
- Minimize the overall cost - usually the *Bayes conditional risk* - on the training data set

$$R(i|x) = \sum_j P(j|x)C(i,j)$$

Predicted Class i \ True Class j	1	2	...	k
1	C(1,1)	C(1,2)	...	C(1,k)
2	C(2,1)	...	...	...
...	...	...	...	...
k	C(k,1)	...	...	C(k,k)

Fig. 7. Multiclass cost matrix.

Source: H. He and E. A. Garcia, "Learning from Imbalanced Data," IEEE Trans. Knowledge and Data Engineering, vol. 21, issue 9, pp. 1263-1284, 2009

Try Different Perspective

- **Anomaly detection** is the detection of rare events. This shift in thinking considers the minor class as the outliers class which might help you think of new ways to separate and classify samples.
- **Change detection** is similar to anomaly detection except rather than looking for an anomaly it is looking for a change or difference. This might be a change in behavior of a user as observed by usage patterns or bank transactions.

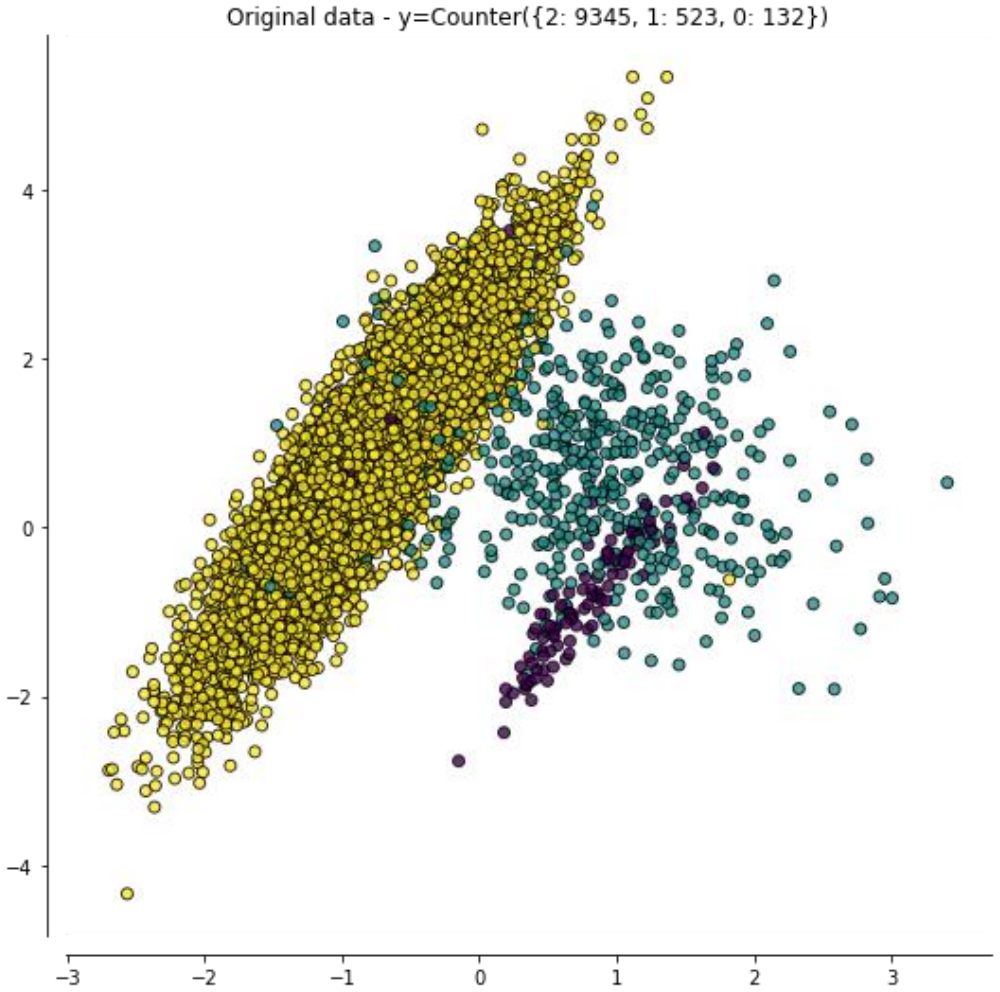
ADASYN Algorithm

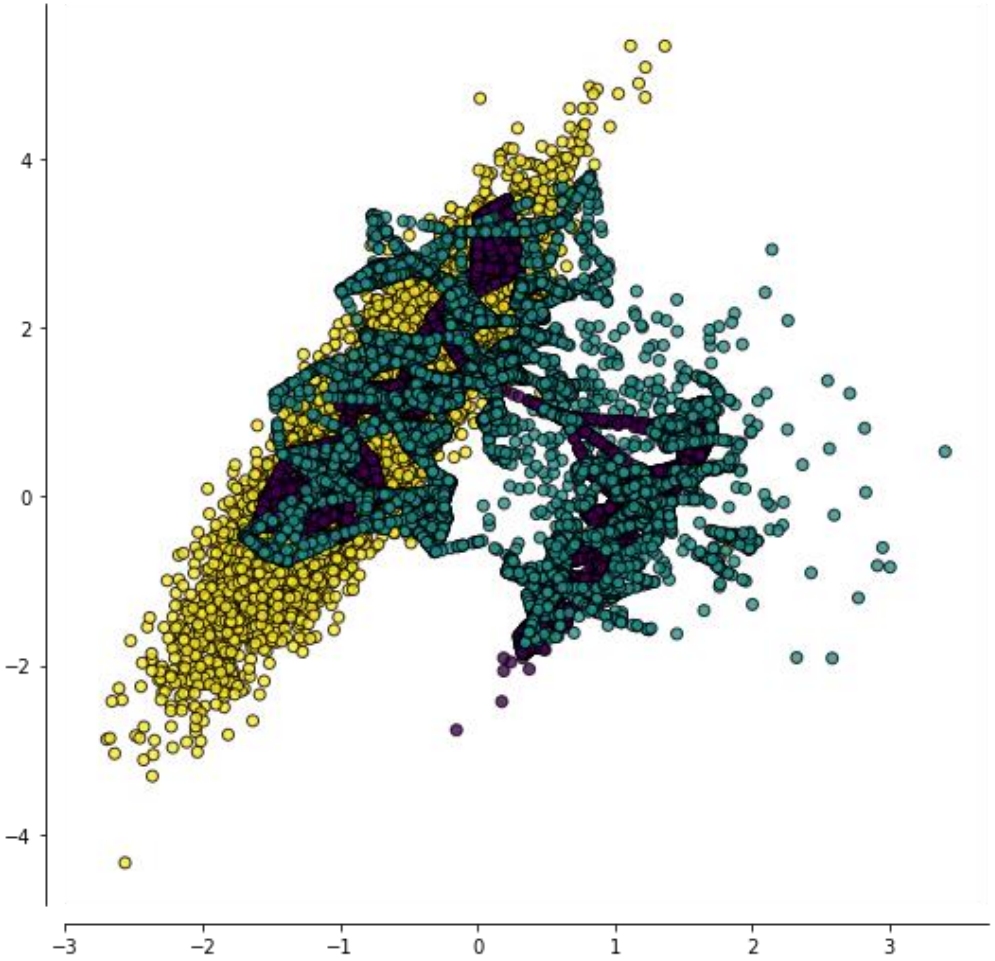
- Adaptively generating minority data samples according to their distributions:
  - more synthetic data is generated for minority class samples that are harder to learn compared to those minority samples that are easier to learn.
- Advantages:
  - Not just copy the same minority data.
  - Generates more data for “harder to learn” examples.
- Disadvantages:
  - For minority examples that are sparsely distributed, each neighborhood may only contain 1 minority example.
  - Precision of ADASYN may suffer due to adaptability nature.

ADASYN Algorithm

- Step 1: Calculate the ratio of minority to majority examples: $\mathrm{d} = \frac{m_s}{m_l}$
  - m_s and m_l are the amount of minority and majority class examples, respectively.
- Step 2: Calculate the total number of synthetic minority data to generate: $G = (m_l - m_s)\beta$
  - G is the total number of minority data to generate. β is the ratio of $\frac{\#\ of\ minority}{\#\ of\ majority}$ desired after ADASYN.
- Step 3: Find the k-Nearest Neighbors of each minority example and calculate the r_i value. After this step, each minority example should be associated with a different neighborhood.
  - $r_i = \frac{\#majority}{k}$
- Step 4: Normalize the r_i values as $\widehat{r_i}$ so that the sum of all r_i values equals to 1.
- Step 5: Calculate the amount of synthetic examples to generate per neighborhood. $G_i = G\widehat{r_i}$
- Step 6: Generate G_i data for each neighborhood. First, take the minority example for the neighborhood, x_i. Then, randomly select another minority example within that neighborhood, xz_i. The new synthetic example: $s_i = x_i + (x_{zi} - x_i)\lambda$

ADASYN Algorithm - Results





What to Use: SMOTE for Imbalanced Classification with Python

https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html

https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTENC.html

https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.BorderlineSMOTE.html

SMOOTE_code.zip included into this lecture folder

What to Use?

https://imbalanced-learn.org/stable/install.html

Prerequisites

The imbalanced-learn package requires the following dependencies:

- python (>=3.6)
- numpy (>=1.13.3)
- scipy (>=0.19.1)
- scikit-learn (>=0.23)
- keras 2 (optional)
- tensorflow (optional)

https://github.com/scikit-learn-contrib/imbalanced-learn file **imbalanced-learn-master2020.zip** is included into this lecture folder

Imbalanced Datasets

https://machinelearningmastery.com/imbalanced-multiclass-classification-with-the-glass-identification-dataset/

https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

https://debuggercafe.com/class-accuracies-for-imbalanced-data-in-deep-learning-image-recognition/

https://www.kdnuggets.com/2018/12/handling-imbalanced-datasets-deep-learning.html

https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced

Ensemble

Summary of Ensemble Techniques

Varying Training Data

- k-fold Cross-Validation Ensemble
- Bootstrap Aggregation (bagging) Ensemble
- Random Training Subset Ensemble

Varying Models

- Multiple Training Run Ensemble
- Hyperparameter Tuning Ensemble
- Snapshot Ensemble
- Horizontal Epochs Ensemble
- Vertical Representational Ensemble

Varying Combinations

- Model Averaging Ensemble
- Weighted Average Ensemble
- Stacked Generalization (stacking) Ensemble
- Boosting Ensemble
- Model Weight Averaging Ensemble

NOTE: We will discuss Ensemble Approaches based on Varying Combinations

Ensemble Methods

- Ensemble Building is the leading winning strategy for machine learning competitions and often the technique used for solving real-world problems.
- What often happens is that while solving a problem or participating in a competition you end up with several trained models, each one with some differences to another - and we end up picking up your best model based on your best evaluation scores.
- Every member of the ensemble makes a contribution to the final output and individual weaknesses are offset by the contribution the other members.

Ensemble Methods: General Idea (1)

- The main motivation for using an ensemble is to find a hypothesis that is not necessarily contained within the hypothesis space of the models from which it is built.
- A machine learning ensemble consists of only a concrete finite set of alternative models.
- Empirically, ensembles tend to yield better results when there is a significant diversity among the models.

Ensemble Methods: General Idea (2)

- Construct a set of classifiers from the training data
- Predict class label of test records by combining the predictions made by multiple classifiers
- Ensemble Learning is all about learning how to best combine predictions from multiple existing models (called the **base-learners**).
- The combined learned model is named the **meta-learner**.
- There are several flavors of Ensemble Learning. We focus on two strategies:

  #1 Stacking and **#2 Weighted Average Ensemble**

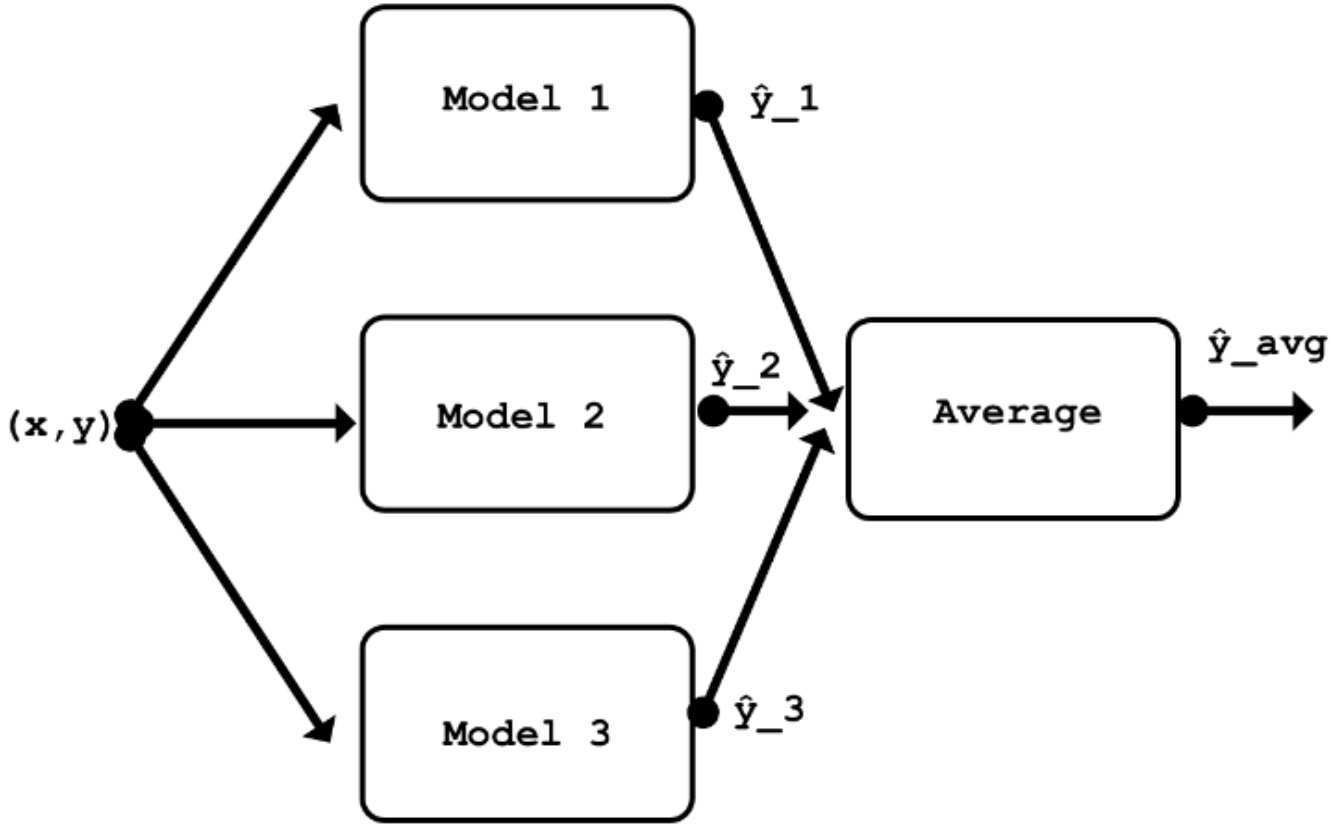
Ensemble Methods: #Stacking Approach

- Stacking is one many different types of ensembles; stacking is one of them.
- It is one of the more general types and can theoretically represent any other ensemble technique.
- Stacking involves training a learning algorithm to combine the predictions of several other learning algorithms.

https://en.wikipedia.org/wiki/Ensemble_learning

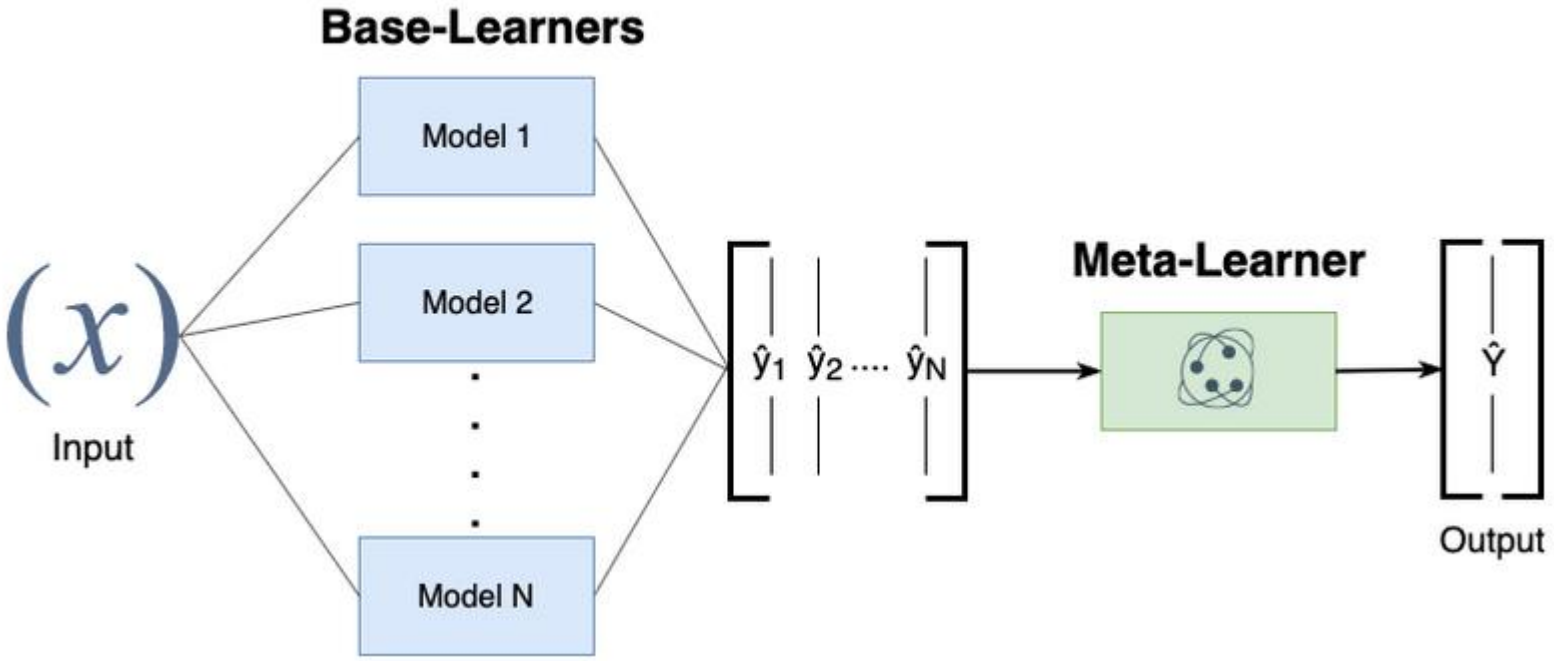
Ensemble Methods: Idea Behind the Stacking

- One of the simplest forms of ensembling is stacking, which involves taking an average of outputs of models in the ensemble. Since averaging doesn't take any parameters, there is no need to train this ensemble (only its models).



Ensemble Methods: #Stacking (1)

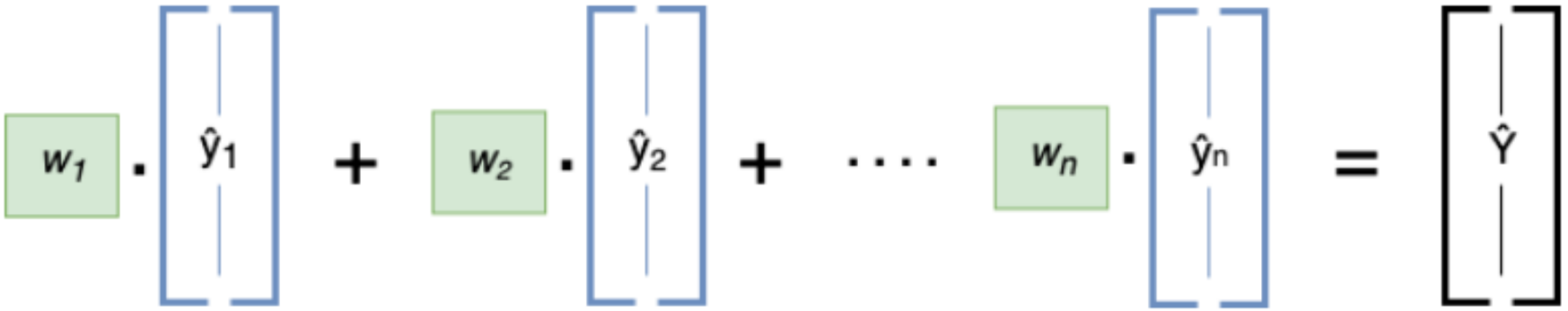
In stacking the output of the base-learners are taken as input for training a meta-learner, that learns how to best combine the base-learners predictions.



- **Stacking** combines multiple predictive models in order to generate a new combined model.

Ensemble Methods: The Weighted Average Ensemble

- This method weights the contribution of each ensemble member based on their performance on **validation dataset**. Models with better contribution receive a higher weight.



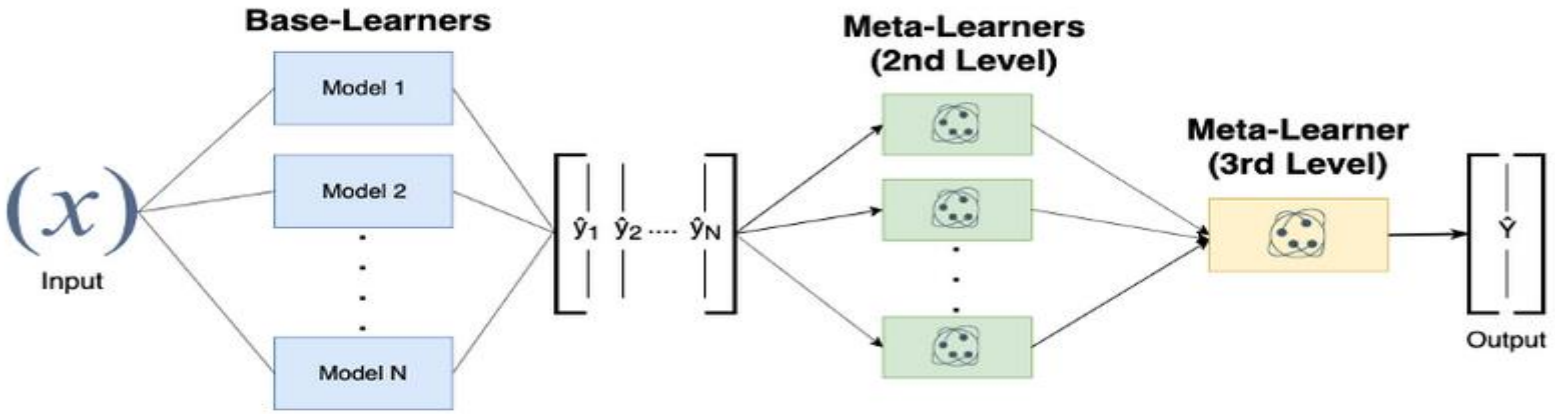
- **Weighted Average Ensemble** is all about weighting the predictions of each base-model generating a combined prediction.

Stacking vs. Weighted Average Ensemble

- The main difference between both methods is that in stacking, the meta-learner takes every single output of the base-learners as a training instance, learning how to best map the base-learner decisions into an improved output.
- The meta-learner can be any classic known machine learning model. The weighted average ensemble on the other hand is just about optimizing weights that are used for weighting all outputs of the base-learner and taking the weighted average.
- In weighted average ensemble there are no meta-learners (besides the weights). Here the number of weights is equal to the number of existing base-learners.
- https://github.com/jcborges/DeepStack
- Included into this Lecture folder file **DeepStack-master.zip.**

3rd Level Stacking with the scikit-learn StackingClassifier

- Scikit-learn 0.22 introduces a *StackingClassifier* and *StackingRegressor* allowing you to have a stack of scikit-learn estimators with *a final classifier or a regressor*. Nice! We can now leverage the scikit-learn Stacking interface for building a 3rd level meta-learner with ***DeepStack***:



- Example of a 3-levels Stacking : https://gist.github.com/jcborges/c2c32bdd1d4ed4b453f9faa0a6ec7781#file-multilevelstacking-py
- Included into this Lecture folder **MultiLevelStacking.zip.**

DeepStack vs. scikit-learn

- But why do we need *DeepStack* if *scikit-learn* is already supporting stacking?
- *DeepStack* is developed to be used in cases for which we already have some pre-trained models and wish to bundle (ensemble) them together creating an even more powerful model. I.e., we don't create or train your base-learners with *DeepStack*.
- *DeepStack* is also generic and is not dependent on the library used for creating the base-learners (*keras*, *pyTorch*, *tensorflow*, etc.).
- The *scikit-learn* stacking API supports creating (training) base-learners from scratch with scikit-learn models.
- For using *DeepStack* with the output of any your *pyTorch*, *tensorflow*, etc. models and train meta-learners for them check the class ***deepstack.base.Member***.
  https://github.com/jcborges/DeepStack/blob/master/deepstack/base.py
- You can also specify any custom target scoring function for the ensembles. *DeepStack* also allow you to save and load your ensembles for later optimization.

Ensemble Methods - References

- **Ensemble Learning**. (n.d.). In *Wikipedia*. https://en.wikipedia.org/wiki/Ensemble_learning
- D. Opitz and R. Maclin (1999) "**Popular Ensemble Methods: An Empirical Study**", Volume 11, pages 169–198 (available at https://www.d.umn.edu/~rmaclin/publications/opitz-jair99.pdf)

https://arxiv.org/pdf/1106.0257.pdf

- **Learning Multiple Layers of Features from Tiny Images**, Alex Krizhevsky, 2009.
- arXiv:1412.6806v3
- arXiv:1312.4400v3
- https://blog.statsbot.co/ensemble-learning-d1dcd548e936
- https://github.com/vsmolyakov/experiments_with_python/blob/master/chp01/ensemble_methods.ipynb