GENERAL CONFEDERATION OF LABOR OF VIETNAM

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

FINAL PROJECT

MACHINE LEARNING

Instructing Lecturer: **MR. LÊ ANH CƯỜNG**

Student's name: **ĐÀO HOÀNG GIANG – 518H0088**

NGUYỄN HUỲNH TÚ - 518H0679

Class : **18H50301**

Course**: 22**

HO CHI MINH CITY, 2020

GENERAL CONFEDERATION OF LABOR OF VIETNAM

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

FINAL PROJECT

MACHINE LEARNING

Instructing Lecturer: **MR. LÊ ANH CƯỜNG**

Student’s name: **ĐÀO HOÀNG GIANG – 518H0088**
NGUYỄN HUỲNH TÚ - 518H0679

Class **: 18H50301**

Course**: 22**

HO CHI MINH CITY, 2020

ACKNOWLEDGEMENT

We would like to thank the Faculty of Information Technology, University

Ton Duc Thang has created favorable conditions for us to study and implement the sub-topic of this treatise.

We would like to express our deep gratitude to the devoted Mr. Le Anh Cuong guide the owner to tell us during the implementation of the topic.

We would like to thank our teachers in the Information Technology department devotedly teaching and equipping her with valuable knowledge for the past year.

We would like to thank the attention, help, and support of the brothers and sisters in the process of doing the thesis essay. Although I have tried to complete the thesis in scope and ability certainly will not avoid the shortcomings.

We are looking forward to receiving sympathy, suggestions, and dedicated instruction from teachers and friends.

THE PROJECT WAS COMPLETED AT TON DUC THANG UNIVERSITY

I pledge that this is a product of our own project and is under the guidance of Mr. Phạm Thái Kỳ Trung. The content of research results in this subject is honest and not published in any form before. The data in the tables used for the analysis, comment, and evaluation were collected by the authors themselves from various sources indicated in the reference section.

In addition, many comments and assessments as well as data from other authors and organizations have been used in the project, with references and annotations.

If any fraud is found, I am fully responsible for the content of my project. Ton Duc Thang University is not involved in any copyright infringement of copyright infringement in the course of implementation (if any).

Ho Chi Minh, April 9th, 2020

Author

(sign and write full name)

Đào Hoàng Giang

Nguyễn Huỳnh Tú

EVALUATION OF INSTRUCTING LECTURER

Confirmation of the instructor

Ho Chi Minh City, 2020

(sign and write full name)

The assessment of the teacher marked

Ho Chi Minh City, 2020

(sign and write full name)

TABLE OF CONTENT

I. DATASET:

We choose Abalone Data Set from UCI to predict the age of abalone from physical measurements.

Data Set Characteristics:	Multivariate	**Number of Instances:**	4177	**Area:**	Life
Attribute Characteristics:	Categorical, Integer, Real	**Number of Attributes:**	8	**Date Donated**	1995-12-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	1011348

Data Set Information:
Predicting the age of abalone from physical measurements. The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope
Attribute Information:
Given is the attribute name, attribute type, the measurement unit and a brief description. The number of rings is the value to predict: either as a continuous value or as a classification problem.

Name / Data Type / Measurement Unit / Description

Sex / nominal / -- / M, F, and I (infant)
Length / continuous / mm / Longest shell measurement
Diameter / continuous / mm / perpendicular to length
Height / continuous / mm / with meat in shell
Whole weight / continuous / grams / whole abalone
Shucked weight / continuous / grams / weight of meat
Viscera weight / continuous / grams / gut weight (after bleeding)
Shell weight / continuous / grams / after being dried
Rings / integer / -- / +1.5 gives the age in years

```
The shape of the DataFrame is (4177, 9)
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7
...	...	...	...	...	...	...	...	...	...
95	M	0.665	0.535	0.195	1.6060	0.5755	0.3880	0.480	14
96	M	0.535	0.435	0.150	0.7250	0.2690	0.1385	0.250	9
97	M	0.470	0.375	0.130	0.5230	0.2140	0.1320	0.145	8
98	M	0.470	0.370	0.130	0.5225	0.2010	0.1330	0.165	7
99	F	0.475	0.375	0.125	0.5785	0.2775	0.0850	0.155	10

100 rows × 9 columns

II. BALANCING DATA

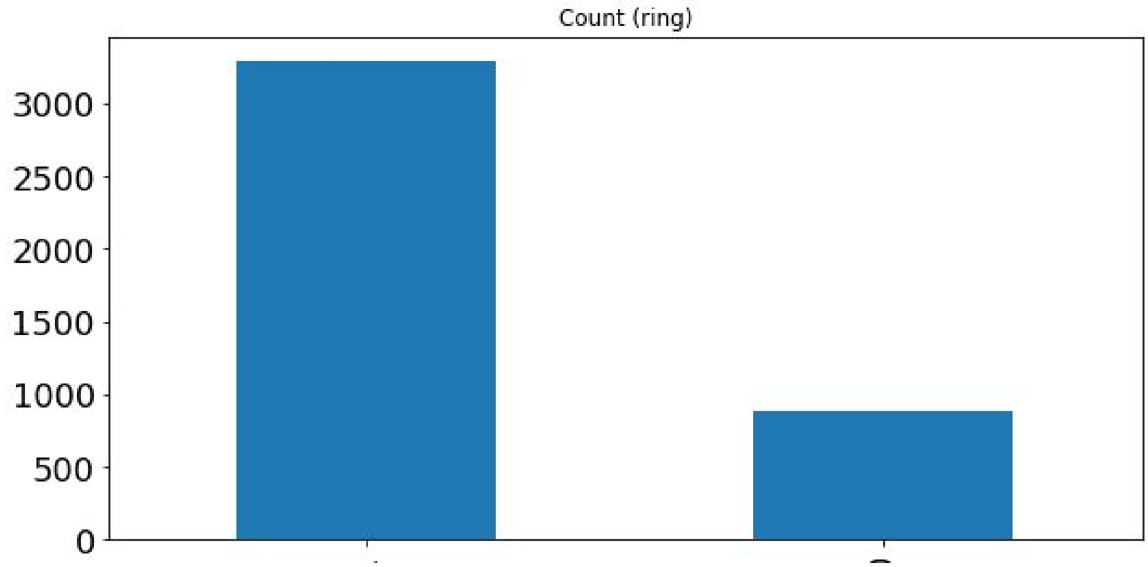
Our dataset has 4177 samples include:

- Class 0 (no): 882
- Class 1(yes): 3295

The proportion of class 0 and class 1 is 0.27:1 therefore the dataset is unbalanced.

```
Class 0: 882
Class 1: 3295
Proportion: 0.27 : 1
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f80d7516630>
```



So we need to balance the dataset by using Up-sample Minority Class

Up-sampling is the process of randomly duplicating observations from the minority class in order to reinforce its signal.

First, we'll import the resampling module from Scikit-Learn:

```python
from sklearn.utils import resample
```

Next, we'll create a new DataFrame with an up-sampled minority class. Here are the steps:

1. First, we'll separate observations from each class into different DataFrames.
2. Next, we'll resample the minority class with replacement, setting the number of samples to match that of the majority class.
3. Finally, we'll combine the up-sampled minority class DataFrame with the original majority class DataFrame.

```
# Separate majority and minority classes
df_majority = df[df.Rings==0]
df_minority = df[df.Rings==1]

# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,                    # sample with replacement
                                 n_samples=df[df.Rings==0].shape[0],    # to match majority class
                                 random_state=123)                # reproducible results

# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_majority, df_minority_upsampled])

# Display new class counts
df_upsampled.Rings.value_counts()
# 1    882
# 0    882
# Name: balance, dtype: int64
```

```
1    882
0    882
Name: Rings, dtype: int64
```

The ratio of the two classes is now 1:1

III. PREPROCESSING DATA

- Check data info

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sex             4177 non-null   object
 1   Length          4177 non-null   float64
 2   Diameter        4177 non-null   float64
 3   Height          4177 non-null   float64
 4   Whole weight    4177 non-null   float64
 5   Shucked weight  4177 non-null   float64
 6   Viscera weight  4177 non-null   float64
 7   Shell weight    4177 non-null   float64
 8   Rings           4177 non-null   int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
```

- Label Encoding refers to converting the labels into the numeric form so as to convert it into the machine-readable form.

```
# Label Encoder
# 0: Female     1: Infant       2: Male

LE = LabelEncoder()
df['Sex'] = LE.fit_transform(df['Sex'])
df['Sex'].head(10)
```

```
0    2
1    2
2    0
3    2
4    1
5    1
6    0
7    0
8    2
9    0
Name: Sex, dtype: int64
```

- Scaling data:

```
] #scale the data
  from sklearn.preprocessing import MinMaxScaler
  scaler = MinMaxScaler()
  scaler.fit(X)
  X = scaler.transform(X)
```

IV. CLASSIFICATION MODELS:

- **Logistic Regression**

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
LG_classifier = LogisticRegression()
LG_classifier.fit(X_train,y_train)

y_pred_LG = LG_classifier.predict(X_test)

acc_score_LG = metrics.accuracy_score(y_pred_LG,y_test)
print(' Logistic Regression accuracy: ',acc_score_LG)

print(classification_report(y_test,y_pred_LG))
```

```
 Logistic Regression accuracy:  0.8205741626794258
              precision    recall  f1-score   support

           0       0.69      0.15      0.24       163
           1       0.83      0.98      0.90       673

    accuracy                           0.82       836
   macro avg       0.76      0.57      0.57       836
weighted avg       0.80      0.82      0.77       836
```

- **K Nearest Neighbors**

```
# k-Neighbors
from sklearn.neighbors import KNeighborsClassifier
knn_classifer = KNeighborsClassifier()
knn_classifer.fit(X_train,y_train)

y_pred_knn = knn_classifer.predict(X_test)

acc_score_knn = metrics.accuracy_score(y_pred_knn,y_test)
print('k-Neighbors accuracy: ',acc_score_knn)

print(classification_report(y_test,y_pred_knn))
```

```
k-Neighbors accuracy:  0.8229665071770335
              precision    recall  f1-score   support

           0       0.56      0.41      0.48       163
           1       0.87      0.92      0.89       673

    accuracy                           0.82       836
   macro avg       0.71      0.67      0.68       836
weighted avg       0.81      0.82      0.81       836
```

- **Support Vector Machine**

```
# kernel{‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’}
# support vector machine
from sklearn.svm import SVC
svm_classifier = SVC(kernel='linear', gamma = 0.01, C = 100)
svm_classifier.fit(X_train,y_train)

y_pred_svm = svm_classifier.predict(X_test)

acc_score_svm = metrics.accuracy_score(y_pred_svm,y_test)
print('SVM (linear) accuracy: ',acc_score_svm)

print(classification_report(y_test,y_pred_svm))
```

```
SVM (linear) accuracy:  0.8433014354066986
              precision    recall  f1-score   support

           0       0.70      0.34      0.46       163
           1       0.86      0.96      0.91       673

    accuracy                           0.84       836
   macro avg       0.78      0.65      0.68       836
weighted avg       0.83      0.84      0.82       836
```

```
# support vector machine
from sklearn.svm import SVC
svm_classifier = SVC(kernel='sigmoid')
svm_classifier.fit(X_train,y_train)

y_pred_svm = svm_classifier.predict(X_test)

acc_score_svm = metrics.accuracy_score(y_pred_svm,y_test)
print('SVM (sigmoid) accuracy: ', acc_score_svm)

print(classification_report(y_test,y_pred_svm))
```

```
SVM (sigmoid) accuracy:  0.6698564593301436
              precision    recall  f1-score   support

           0       0.18      0.20      0.19       163
           1       0.80      0.78      0.79       673

    accuracy                           0.67       836
   macro avg       0.49      0.49      0.49       836
weighted avg       0.68      0.67      0.68       836
```

```
# support vector machine
from sklearn.svm import SVC
svm_classifier = SVC(kernel='poly')
svm_classifier.fit(X_train,y_train)

y_pred_svm = svm_classifier.predict(X_test)

acc_score_svm = metrics.accuracy_score(y_pred_svm,y_test)
print('SVM (poly) accuracy: ', acc_score_svm)

print(classification_report(y_test,y_pred_svm))
```

```
SVM (poly) accuracy:  0.8277511961722488
              precision    recall  f1-score   support

           0       0.70      0.20      0.31       163
           1       0.84      0.98      0.90       673

    accuracy                           0.83       836
   macro avg       0.77      0.59      0.61       836
weighted avg       0.81      0.83      0.79       836
```

```
# support vector machine
from sklearn.svm import SVC
svm_classifier = SVC()
svm_classifier.fit(X_train,y_train)

y_pred_svm = svm_classifier.predict(X_test)

acc_score_svm = metrics.accuracy_score(y_pred_svm,y_test)
print('SVM (rbf) accuracy: ', acc_score_svm)

print(classification_report(y_test,y_pred_svm))
```

```
SVM (rbf) accuracy:  0.8433014354066986
              precision    recall  f1-score   support

           0       0.82      0.25      0.38       163
           1       0.84      0.99      0.91       673

    accuracy                           0.84       836
   macro avg       0.83      0.62      0.65       836
weighted avg       0.84      0.84      0.81       836
```

- **Naive Bayes**

```
# Multinominal naive bayes
from sklearn.naive_bayes import GaussianNB
gnb_classifer = GaussianNB()
gnb_classifer.fit(X_train, y_train)

y_pred_gnb = gnb_classifer.predict(X_test)
acc_score_gnb = metrics.accuracy_score(y_pred_gnb,y_test)
print('Multinominal NB model accuracy: ', acc_score_gnb)

print(classification_report(y_test,y_pred_gnb))
```

```
Multinominal NB model accuracy:  0.7655502392344498
              precision    recall  f1-score   support

           0       0.39      0.36      0.37       163
           1       0.85      0.86      0.86       673

    accuracy                           0.77       836
   macro avg       0.62      0.61      0.61       836
weighted avg       0.76      0.77      0.76       836
```

- **Multilayer Perceptron**

```
#MultiLayer Perceptron
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(64,64),activation='relu',solver='sgd',batch_size=64, max_iter=1000)
mlp.fit(X_train, y_train)
y_pred = mlp.predict(X_test)
print('MultiLayer Perceptron accuracy is',accuracy_score(y_pred,y_test))
print(classification_report(y_test,y_pred))
```

```
MultiLayer Perceptron accuracy is 0.8528708133971292
              precision    recall  f1-score   support

           0       0.72      0.40      0.52       163
           1       0.87      0.96      0.91       673

    accuracy                           0.85       836
   macro avg       0.79      0.68      0.72       836
weighted avg       0.84      0.85      0.84       836
```

V. Train the model using the batch size and epoch number.

- Import necessary library, Keras API

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
from sklearn.datasets import make_moons
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
from matplotlib import pyplot
```

- Define model. The model is optimized using the binary cross-entropy loss function, suitable for binary classification problems and the efficient Adam version of gradient descent.

```
# define model
model = Sequential()
model.add(Dense(500, input_dim=8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

- Callbacks provide a way to execute code and interact with the training model process automatically. Callbacks can be provided to the fit() function via the “callbacks” argument.

```
# fit model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=100,batch_size=40, verbose=2, callbacks=[es])
```

- Early stopping is a method that allows you to specify an arbitrarily large number of training epochs and stop training once the model performance stops improving on a hold-out validation dataset. Early stopping requires that a validation dataset is evaluated during training

```
# simple early stopping
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=200)
```

- Model evaluation

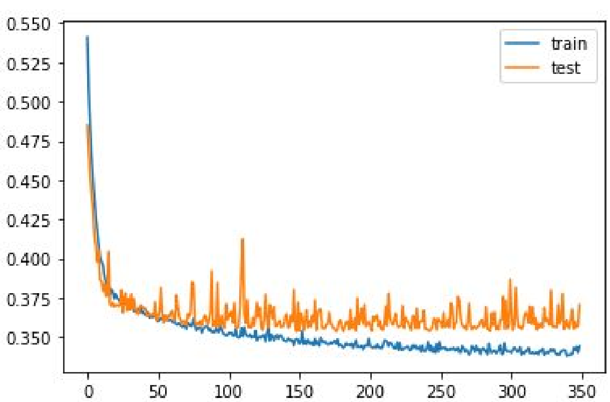
```
# evaluate the model
_, train_acc = model.evaluate(X_train, y_train, verbose=0)
_, test_acc = model.evaluate(X_train, y_train, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

- Result with 2000 epochs

```
Epoch 349/2000
84/84 - 0s - loss: 0.3443 - accuracy: 0.8530 - val_loss: 0.3707 - val_accuracy: 0.8445
Epoch 00349: early stopping
Train: 0.858, Test: 0.858
```

- Display training history

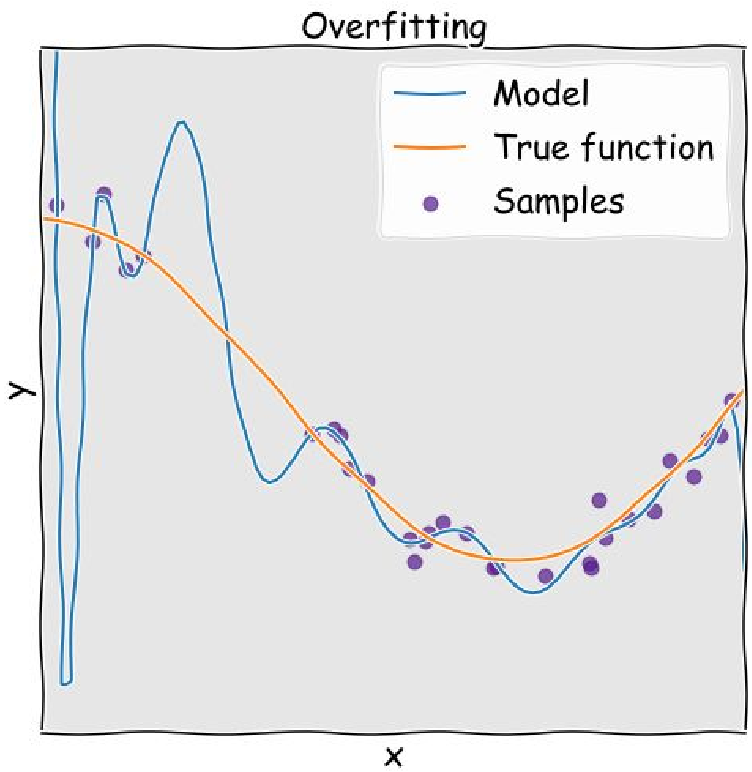
```
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

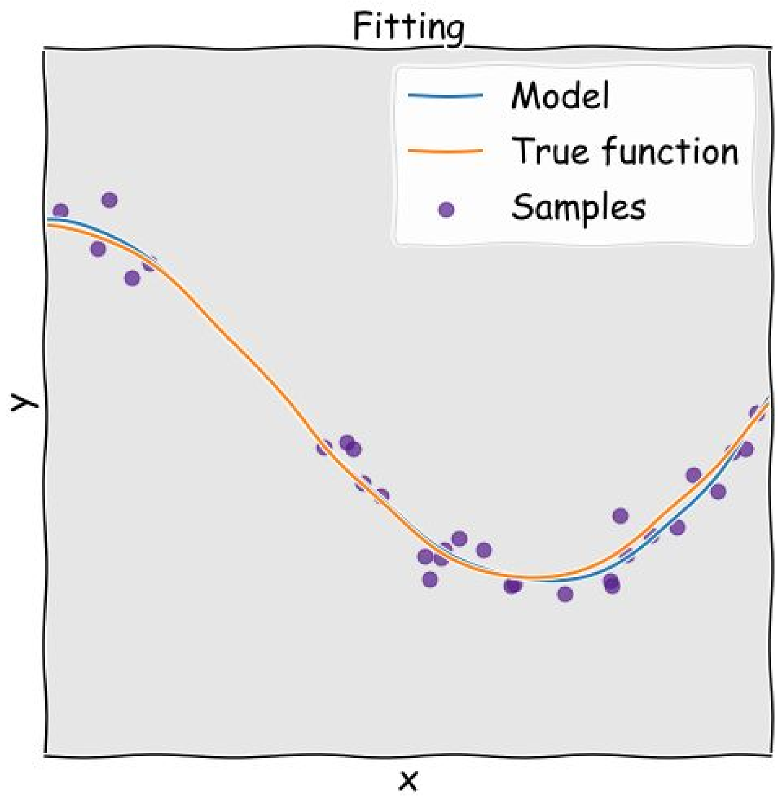


VI. OVERFITTING PROBLEM

Overfitting occurs when our model is too complex to capture the underlying relationships in the data. A model that performs well on training data, but poorly on test data is overfit.

An example of overfitting. The model function has too much complexity (parameters) to fit the true function correctly.

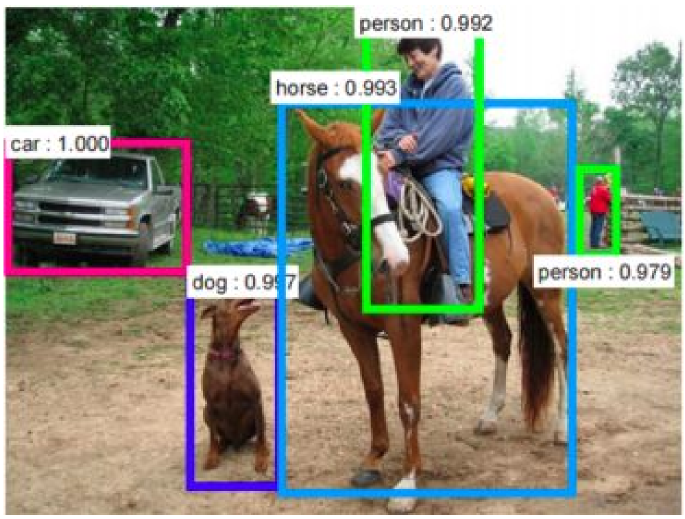




- **Avoiding Overfitting**
  - **Hold back a test set**: training your model with around 2/3 or 3/4 of the data and using the rest for testing the resulting model.
  - **Resampling with Cross-validation:** generates multiple train-test splits and fits the model with each split in turn. Use cross-validation to prove that your model performs well on different cuts of unseen data.
  - **Feature selection**: Remove excessive features
  - **Regularization**: make a model become a simpler version of itself
  - **Early Stopping:** stop the algorithm before the loss function reaches too small a value.

VII.COVOLUTIONAL NEURAL NETWORK

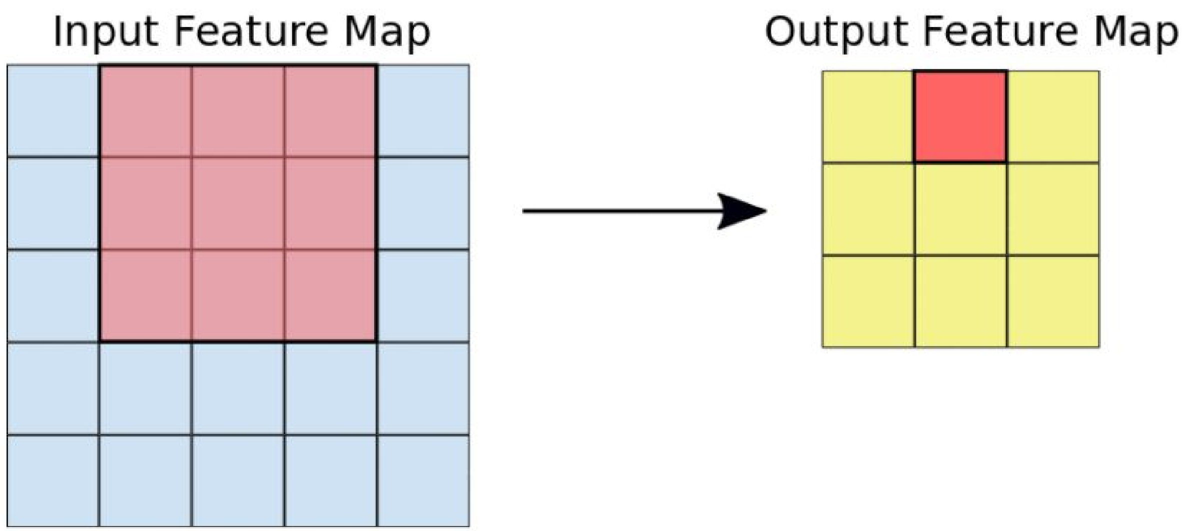
A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other.



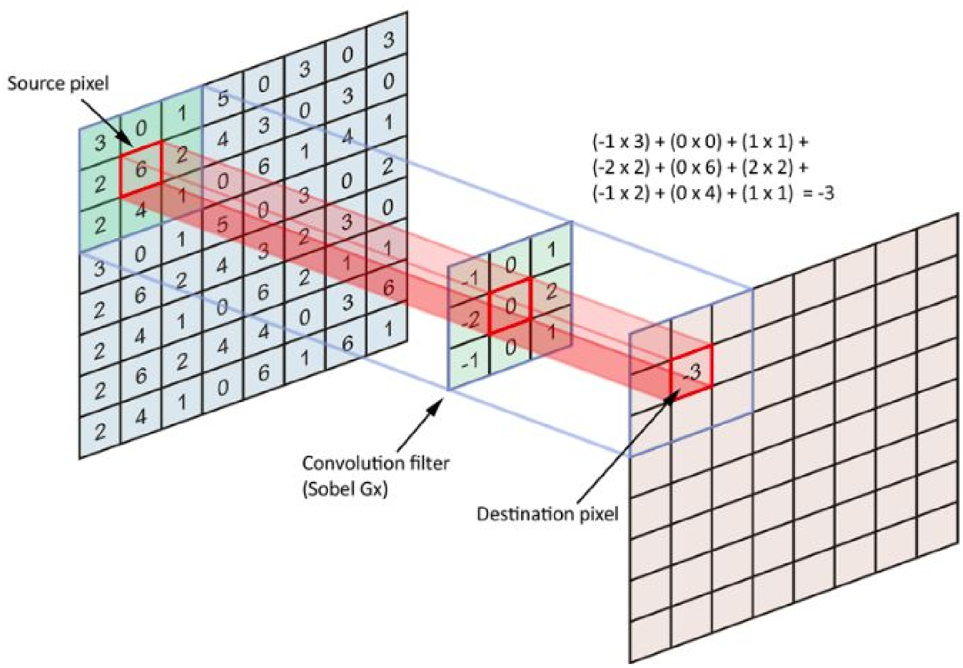
1. Convolution

A convolution extracts tiles of the input feature map, and applies filters to them to compute new features, producing an output feature map, or convolved feature (which may have a different size and depth than the input feature map). Convolutions are defined by two parameters:

- **Size of the tiles that are extracted** (typically 3x3 or 5x5 pixels).
- **The depth of the output feature map**, which corresponds to the number of filters that are applied.



In reality convolutions are performed in 3D. Each image is namely represented as a 3D matrix with a dimension for width, height, and depth. Depth is a dimension because of the colours channels used in an image (RGB).

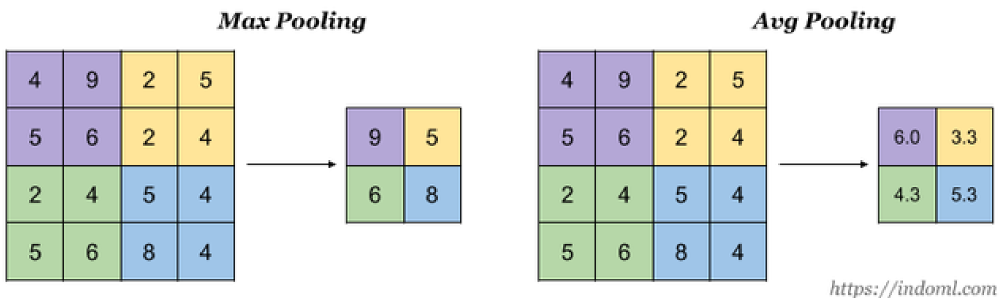


2. ReLU

Following each convolution operation, the CNN applies a Rectified Linear Unit (ReLU) transformation to the convolved feature, in order to introduce nonlinearity into the model. The ReLU function, **F(x)=max(0,x)** , returns x for all values of $x > 0$, and returns 0 for all values of $x \leq 0$.

3. Pooling

Pooling layer is used to reduce the size of the representations and to speed up calculations, as well as to make some of the features it detects a bit more robust.



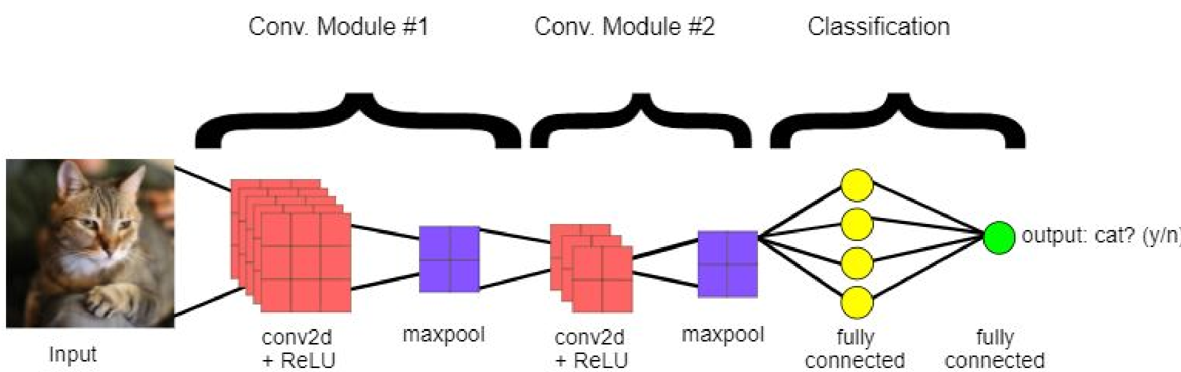
Properties of pooling layer:

- it has hyper-parameters:
    - **size** (*f*)
    - **stride** (*s*)
    - **type** (max or avg)

4. Fully Connected Layers

Perform classification based on the features extracted by the convolutions. Typically, the final fully connected layer contains a softmax activation function, which outputs a probability value from 0 to 1 for each of the classification labels the model is trying to predict.

5 .The end-to-end structure of a convolutional neural network



Summary