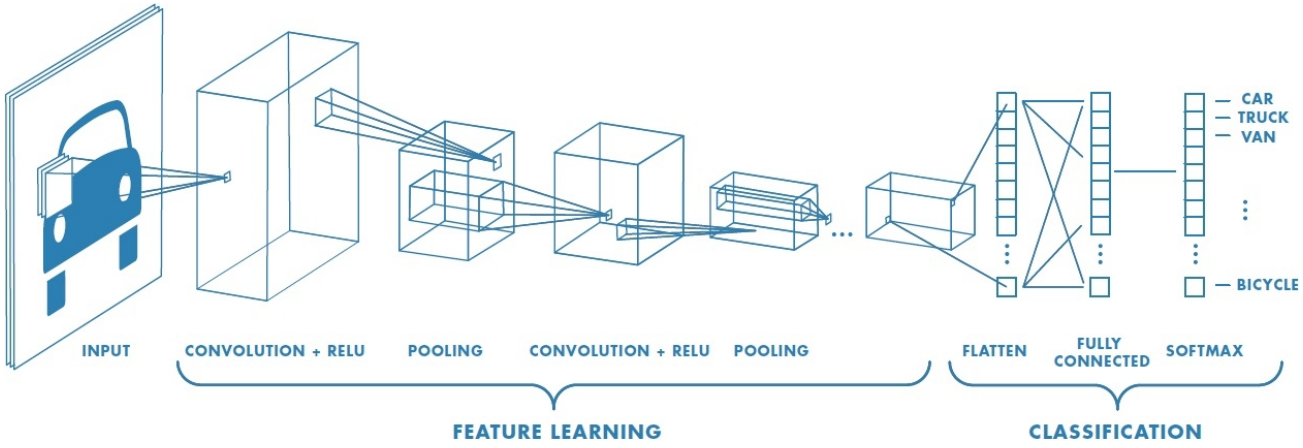
A Brief Tutorial on Artificial Neural Nets

Currently, artificial neural networks (ANNs) are one of the hottest topics in machine learning and some of the most widely used tools in data science. In the last years, they have achieved impressive performance in tasks traditionally elusive for artificial intelligence, such as image classification and speech recognition. In this notebook, we will briefly introduce the key ideas behind this learning approach, focusing on the so-called feed-forward arquitectures.



A great book for studying this topic is:

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning (adaptive computation and machine learning series). Adaptive Computation and Machine Learning series, 800.

Tools

```
For this activity, we will need a library specialized in deep learning
called Keras. We will also use sklearn, a very complete library for
machine learning in Python.


<table align='left'>
<tr>
<td><img src='figs/keras_logo.png' width='450' height='200'/></td>
<td><img src='figs/scikit.png' width='450' height='200'/></td>
</tr>
</table>
```

Please refer to https://keras.io (https://keras.io) and http://scikit-learn.org/stable/ (http://scikit-learn.org/stable/) for help regarding installation and usage.

Problem & Dataset: Ragu or Carbonara?

In this lab, we will train a neural network to solve a critical task during our stay in Italy: distinguishing between types of pasta. To start we will consider only two classes: *pasta* *al* *ragu* and *pasta* *alla* *carbonara*. As a homework, you will consider also *gnocchi* and *lasagna*. If you have not yet tried one of these dishes, I absolutely recomend you to visit one of the fantastic *osterie* in the town and exploit the opportunity to collect examples by yourself!

```
<table align='left'>
<tr>
<td><img src='figs/carbonara4.jpg' alt="Drawing" style="height:
400px;"/></td>
<td><img src='figs/ragu4.jpg' alt="Drawing" style="height: 400px;"/>
</td>
</tr>
</table>
```

We will extract images from the *Food-101* dataset, a collection of 101.000 food images organized in 101 categories, maintained by collegues of the ETH Zurich and available at https://www.vision.ee.ethz.ch/datasets_extra/food-101/ (https://www.vision.ee.ethz.ch/datasets_extra/food-101/). The images contain some amount of noise and sometimes wrong labels. All images are RGB and have been rescaled to have a maximum side length of 512 pixels, but are of variable width and length.

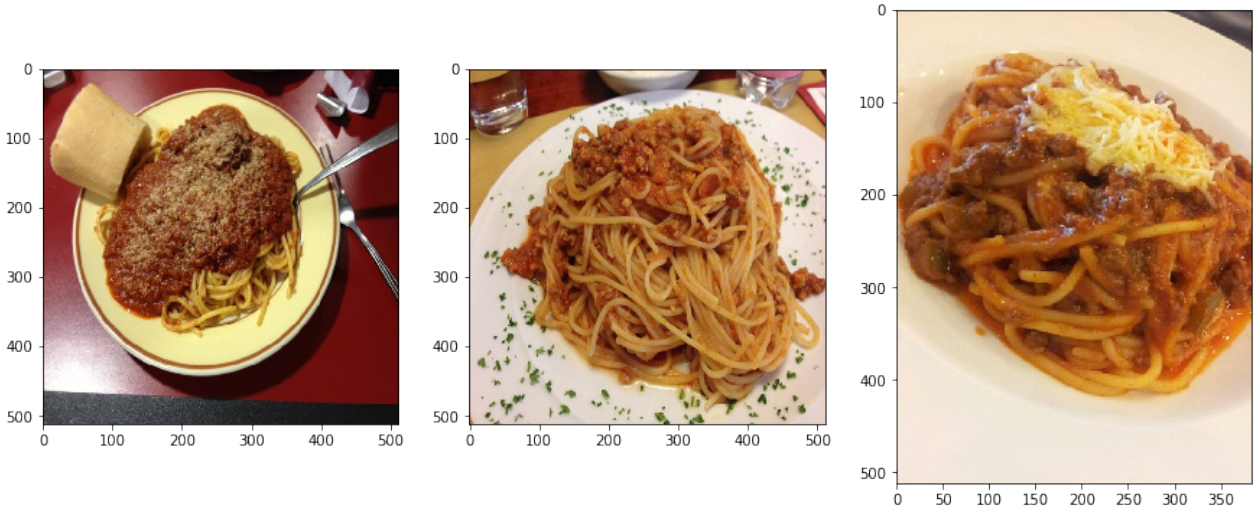
To start, verify that you have the folders required for this lab:

food/spaghetti_bolognese
food/spaghetti_carbonara
food/gnocchi
food/lasagna

If you do not have the dataset in your computer, you can download the required folders from

http://octopus.inf.utfsm.cl/~ricky/food.tar.gz (http://octopus.inf.utfsm.cl/~ricky/food.tar.gz)

In [3]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

root_dir = 'food/'
ragu_dir = 'spaghetti_bolognese/'
carbonara_dir = 'spaghetti_carbonara/'
lasagna_dir = 'lasagna/'

all_ragu = os.listdir(os.path.join(root_dir, ragu_dir))
all_carbonara = os.listdir(os.path.join(root_dir, carbonara_dir))

fig, ax = plt.subplots(1,3, frameon=False, figsize=(15, 20))
rand_img = np.random.choice(all_ragu)
img = plt.imread(os.path.join(root_dir, ragu_dir, rand_img))
ax[0].imshow(img)

rand_img = np.random.choice(all_ragu)
img = plt.imread(os.path.join(root_dir, ragu_dir, rand_img))
ax[1].imshow(img)

rand_img = np.random.choice(all_ragu)
img = plt.imread(os.path.join(root_dir, ragu_dir, rand_img))
ax[2].imshow(img)

plt.show()
```



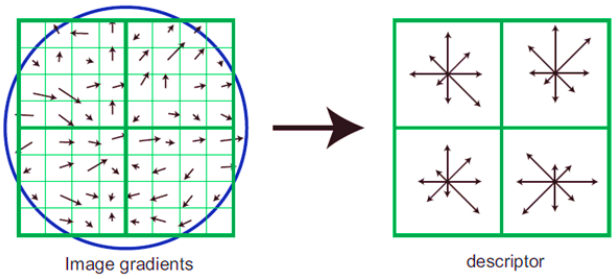
1. The SVM Meets Pasta

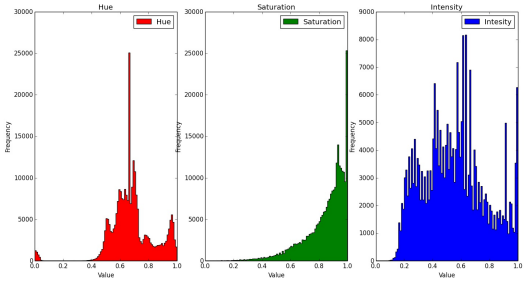
To see the importance of using good features to build a traditional classifier, we will start training an SVM for pasta recognition. We will first use the pixels as attributes and then we will try to extract more useful features.

In [ ]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

root_dir = 'food/'
ragu_dir = 'spaghetti_bolognese/'
carbonara_dir = 'spaghetti_carbonara/'
lasagna_dir = 'lasagna/'

all_ragu = os.listdir(os.path.join(root_dir, ragu_dir))
all_carbonara = os.listdir(os.path.join(root_dir, carbonara_dir))
#all_lasagna = os.listdir(os.path.join(root_dir, lasagna_dir))

import matplotlib.image as img
from skimage.transform import resize

target_w = 65
target_h = 65
all_imgs_orig = []
all_imgs = []
all_labels = []
idx = 0
min_side = 400
resize_count = 0

for img_name in all_ragu:
    img_arr = img.imread(os.path.join(root_dir, ragu_dir, img_name))
    w,h,d = img_arr.shape
    img_arr_rs = img_arr
    img_arr_rs = resize(img_arr, (target_w, target_h))
    all_imgs.append(img_arr_rs)
    all_imgs_orig.append(img_arr)
    all_labels.append(1)

for img_name in all_carbonara:
    img_arr = img.imread(os.path.join(root_dir, carbonara_dir, img_name))
    w,h,d = img_arr.shape
    img_arr_rs = img_arr
    img_arr_rs = resize(img_arr, (target_w, target_h))
    all_imgs.append(img_arr_rs)
    all_imgs_orig.append(img_arr)
```

```
    all_labels.append(0)

from sklearn.model_selection import train_test_split
from keras.utils import to_categorical


X = np.array(all_imgs)
Y = to_categorical(np.array(all_labels),num_classes=2)
Y = Y[:,0]

n,w,l,d = X.shape
X_raw = np.reshape(X,(n,w*l*d))

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_raw)

X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y, test_siz

from sklearn import svm
C_default = 1
clf = svm.SVC(kernel='linear', C=C_default)
clf.fit(X_train, Y_train)

train_accuracy_svm = clf.score(X_train,Y_train)
print(train_accuracy_svm)
test_accuracy_svm = clf.score(X_test,Y_test)
print(test_accuracy_svm)
```

Now, we will use more powerful descriptors, namely *Histograms of Oriented Gradients* (HOG) and *Color Histograms*.





In [ ]:
```
X = np.array(all_imgs)
Y = to_categorical(np.array(all_labels),num_classes=2)
Y = Y[:,0]
print(X.shape)
print(Y.shape)

from top_level_features import hog_features
from top_level_features import color_histogram_hsv
from top_level_features import extract_features
features = extract_features(X,[hog_features, color_histogram_hsv]) #extra
print X.shape
print features.shape

scaler = StandardScaler()
X = scaler.fit_transform(features)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,

from sklearn import svm
C_default = 1
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, Y_train)
clf.score(X_test,Y_test)

train_accuracy_svm = clf.score(X_train,Y_train)
print(train_accuracy_svm)
test_accuracy_svm = clf.score(X_test,Y_test)
print(test_accuracy_svm)
```

2. Motivating Neural Nets

We have discussed about the importance of working with the correct features in machine learning and data science. The performance of all the classical models for learning from examples (decision trees, logistic regression, support vectors machines, etc) heavily depends on the human ability to figure out which attributes are good *predictors* in a given task and, at the end of the day, most of the time in a machine learning project is spent actually doing feature engineering. Deep learning and, in particular, artificial neural networks (ANNs), are techniques designed to address this problem. The general idea is casting the problem of feature engineering as a learning problem and leaving the learner itself the task of determining which are the correct attributes to make the final decision.

Consider for example an SVM. You known that a linear SVM works well if you provide it with good and enough features $x_1, x_2, \dots x_d$, that make the problem almost linearly separable. The work of the SVM is to learn the weight w_i of each attribute x_i to implement a decision function of the form:

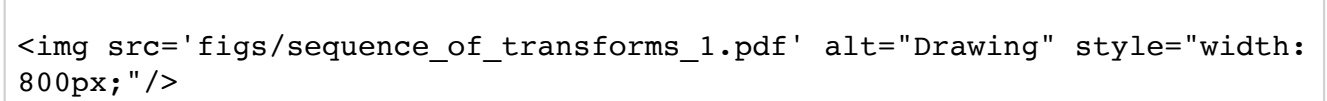
We have seen that non-linear SVMs works by introducing a feature map $\phi()$ generating new attributes $\phi_1, \phi_2, \ldots, \phi_S$ from the original set of attributes. In this approach, the feature map is selected *by hand*, probably choosing a general purpose kernel, or by designing a kernel function specialized in your problem, but, in any case, *by hand*. In neural networks, the transformation become part of the learning process. That means that we make each transformation ϕ_i to depend on a set of parameters $W_i^{(1)}$ and learn these parameters together with the parameters, let say $W_i^{(2)}$, of the final classifier. For example, if we allow each transformation ϕ_i to be a linear weighted combination of the original attributes, we obtain a network of transformations of the form:

Often, allowing non-linear transformations works better, but, at the same time, linear transformations are easy to understand and easy to learn (optimize). To combine these ideas, most neural networks design each transformation ϕ_i as a linear combination of the original attributes, followed by a non-linear but non-trainable transformation g called the *activation function* of the model.

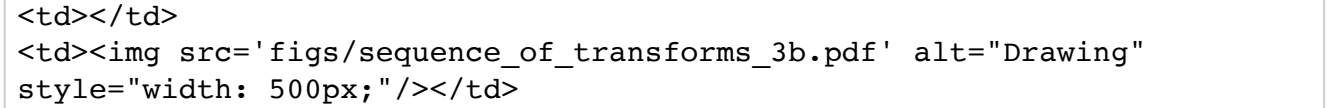
Deep neural networks appear when we iterate this idea. Maybe, the original attributes $x_1, x_2, \dots x_d$ are not powerful enough to learn good attributes $\phi_1(x), \phi_2(x), \dots, \phi_S(x)$. Maybe, we need to first learn intermediate attributes, i.e., a first layer of transformations $\phi'_1, \phi'_2, \dots, \phi'_T$ from the original attributes, and learn the transformations $\phi_1, \phi_2, \dots, \phi_S$ on top of these new intermediate features. In this way, you obtain a network of transformations that is said to have one *input layer* (just to accommodate the original attributes), two intermediate or *hidden layers* and one *output layer*.

3. Neural Network Architecture

So, a neural network attempts to approximate/learn a task $t(x)$ using a sequence of transformations $H^{(1)} \rightarrow H^{(2)} \ldots \rightarrow H^{(L)}$ that are known as *_layers_*.



By convention, the first layer is just used to copy the input and is called *_the input layer_*. The last layer produces the desired output and is called *_the output layer_*. All the intermediate layers are called *_hidden layers_*. Each transformation $H^{(\ell)}$ attempts to learn a new representation $a^{(\ell)}$ that simplifies the work of the next layer.



Each component of a transformation is an attribute that the model needs to learn. This is implemented using a linear combination of the attributes learn from the previous layer and applying to the result, a non-linear function. For historical reasons (the model was conceived as a mathematical model of biological neurons), this computational unit is known as an *_artificial neuron_* and so the entire graph of computation is known as a *_artificial neuronal network_*.

Mathematically, the operation implemented by each neuron has the following form:

where the weights $w_{st}^{(\ell)}$ connecting the attribute (neuron) t of the level ℓ with the attribute (neuron) s of the level $\ell + 1$ are parameters that need to be learnt from data.

To be more precise, networks implementing the arquitecture we have described above are known as *feed forward neural nets*. There are slightly more complicated models, e.g. *convolutional nets* used in computer vision and *recurrent neuronal nets* used for speech recognition and other sequence learning problems.

4. Creating a First Net for Pasta Recognition

Creating a neural network using the Keras library is pretty easy. You only need to specify a shape for the input layer, a shape for the output layer and stack layers in between!

In [ ]:
```
X = np.array(all_imgs)
Y = to_categorical(np.array(all_labels),num_classes=2)
Y = Y[:,0]
print(X.shape)
print(Y.shape)

from top_level_features import hog_features
from top_level_features import color_histogram_hsv
from top_level_features import extract_features
features = extract_features(X,[hog_features, color_histogram_hsv]) #extra
print X.shape
print features.shape

scaler = StandardScaler()
X = scaler.fit_transform(features)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,

model = Sequential()
model.add(Dense(100,input_dim=586))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(50))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(25))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(1))
model.add(Activation('sigmoid'))
```

5. Training the Neural Net

```
Essentially, neural networks are trained just like any other machine
learning model. You need to specify a learning goal which typically
represents a cost and then use an optimization algorithm to find the
parameters which minimize that cost. The most common training goal for
ANNs is the training error

\begin{align}
E = \frac{1}{n} \sum_\ell Q(f(x^{(\ell)},y^{(\ell)}))
\end{align}

where $(x^{(\ell)},y^{(\ell)})$, $\ell=1,2,\ldots,n$ are the training
examples and $Q(f(x^{(\ell)},y^{(\ell)}))$ is a loss function.
```

Algorithms to train neural networks are, usually, variants of gradient descent. That, is the weights of the net are optimized using an iterative algorithm, where each iteration improves a little the previous solution, just like in the Perceptron algorithm!

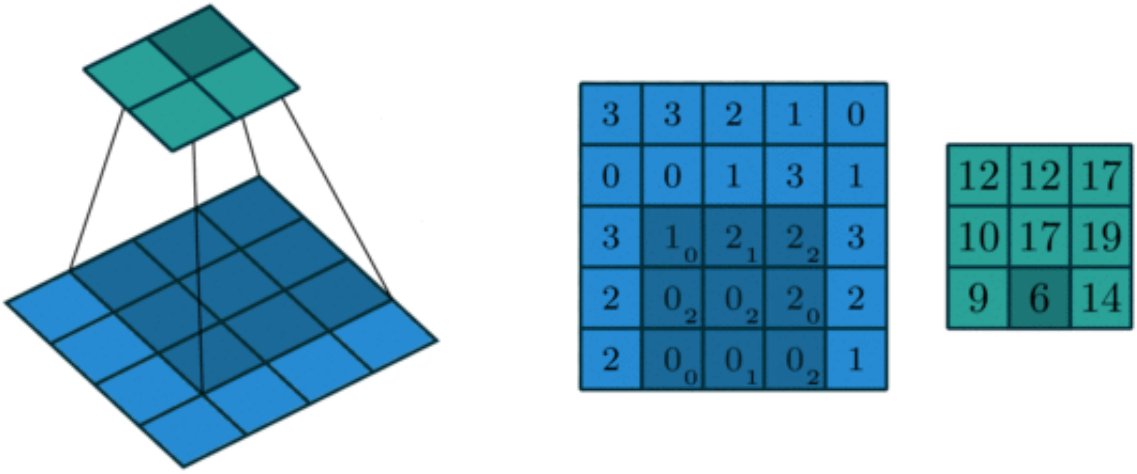
In the neural network world, each iteration of this algorithm is called *_a epoch_*. Each epoch works by predicting the values for a subset of examples, computing the loss for these cases, computing the gradient of each weight with respect to the loss and moving the weights in the direction of the negative of the gradient. The number of examples used for a round of prediction and correction is called *_the batch size_*. An epoch finishes when all the training examples have been used for perfoming corrections.

<table align='left'>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</table>

In [ ]:
```
optimizer = Adam()

model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['

batch_size = 64
n_epochs = 20

history = model.fit(X_train,Y_train,epochs=n_epochs,batch_size=batch_size
```

6. Specialized Layers

We have not been improved a lot the results of our basic SVM (Does it mean that the SVM is too good or the ANN too bad?). It is time to introduce more powerful layers, layers specialized in image classification. These layers are known as *convolutional layers*. Nets using these layers are called *convolutional neural nets* (CNNs).



Convolutional layers receive as input an image $X \in \mathbb{R}^{IST}$ of I input channels $X_i \in \mathbb{R}^{ST}$, and compute as output a new image Y, composed of J output channels $Y_j \in \mathbb{R}^{UV}$ known as \emph{feature maps}, by performing an operation of the form

$$Y_j = g\left(\sum\nolimits_i X_i * W_{ji} + b_j\right) \in \mathbb{R}^{UV},\ \forall j\ ,$$

where $*$ denotes the (2-dimensional) convolution operation

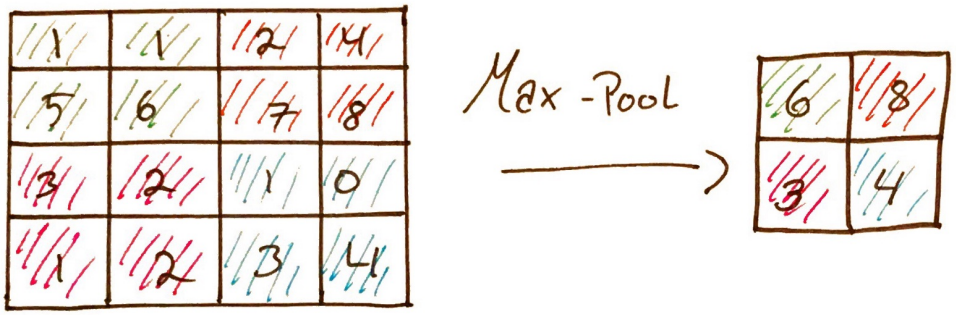
$$X_i * W_{ji}[u,v] = \sum_{p=0}^{P-1}\sum_{q=0}^{Q-1} X_i[u+p, v+q] W_{ji}[P-p, Q-q]\ ,$$

$g(\cdot)$ is a point-wise non-linear function and $W_{ji} \in \mathbb{R}^{PQ}$, $b \in \mathbb{R}^J$ are trainable parameters. The matrix W_{ji}, defining the interaction between the j-th output channel and the i-th input channel, parametrises a $P \times Q$ spatial filter implemented by the higher layer to detect or enhance some feature in the incoming image.

Convolutions have been long used in computer vision to implement useful transformations of images.

The only difference is that in neural networks the parameters of the operation are learn from data and thus determined automatically to improve in a given task.

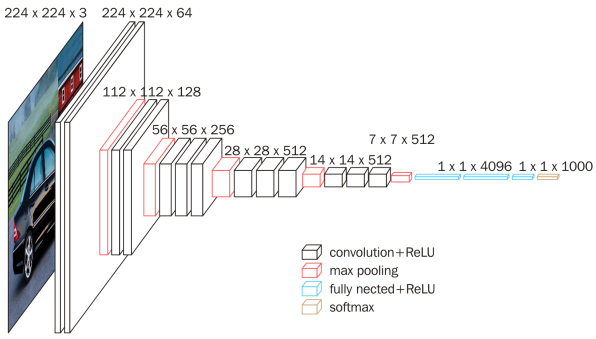
A second type of computation introduced by CNNs is *pooling*. Broadly speaking, pooling layers of a CNN implement a spatial dimensionality reduction operation designed to (i) drastically reduce the number of free parameters of the next layer, (ii) make the representation learn by the model more invariant to distortions and (iii) allow the higher layers of the model to focus on greater areas of the input image. Given an input volume pooling layers substitute a patch of pixels in the incoming image by a single representative value.

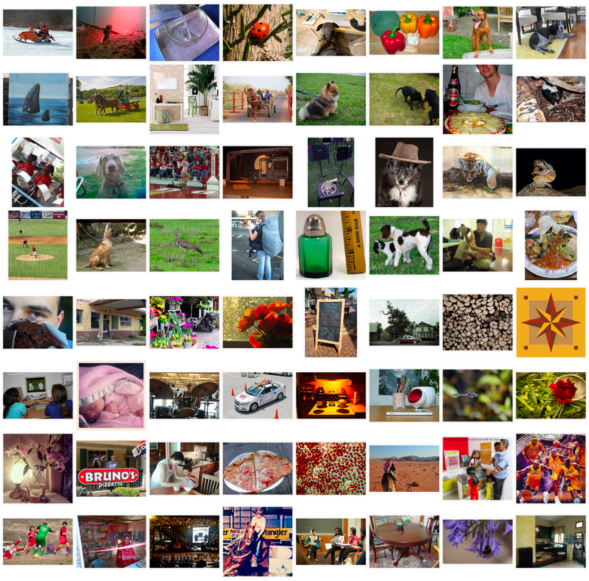


7. Creating and Training a Convolutional Neural Net

In [ ]:
```
X = np.array(all_imgs)
Y = to_categorical(np.array(all_labels),num_classes=2)
Y = Y[:,0]
print(X.shape)
print(Y.shape)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,


model = Sequential()
model.add(Conv2D(64, (3, 3), padding='same', input_shape=(target_w, targe
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

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

model.add(Flatten())  # this converts our 3D feature maps to 1D feature v
model.add(Dense(10))
model.add(Activation('relu'))
#model.add(Dropout(0.5))

model.add(Dense(1))
model.add(Activation('sigmoid'))
optimizer = Adagrad()
model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['

batch_size = 32
n_epochs = 100

history = model.fit(X_train,Y_train,epochs=n_epochs,batch_size=batch_size
```

8. Using Pre-trained Nets





¡

In [ ]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import imresize as resize

root_dir = 'food/'
ragu_dir = 'spaghetti_bolognese/'
carbonara_dir = 'spaghetti_carbonara/'
lasagna_dir = 'lasagna/'

all_ragu = os.listdir(os.path.join(root_dir, ragu_dir))
all_carbonara = os.listdir(os.path.join(root_dir, carbonara_dir))
#all_lasagna = os.listdir(os.path.join(root_dir, lasagna_dir))

import matplotlib.image as img
#from skimage.transform import resize

target_w = 299
target_h = 299
all_imgs_orig = []
all_imgs = []
all_labels = []
idx = 0
min_side = 400
resize_count = 0
```

```
for img_name in all_ragu:
    img_arr = img.imread(os.path.join(root_dir, ragu_dir, img_name))
    w,h,d = img_arr.shape
    img_arr_rs = img_arr
    img_arr_rs = resize(img_arr, (target_w, target_h))
    all_imgs.append(img_arr_rs)
    all_imgs_orig.append(img_arr)
    all_labels.append(1)

for img_name in all_carbonara:
    img_arr = img.imread(os.path.join(root_dir, carbonara_dir, img_name))
    w,h,d = img_arr.shape
    img_arr_rs = img_arr
    img_arr_rs = resize(img_arr, (target_w, target_h))
    all_imgs.append(img_arr_rs)
    all_imgs_orig.append(img_arr)
    all_labels.append(0)

# for img_name in all_lasagna:
#     img_arr = img.imread(os.path.join(root_dir, lasagna_dir, img_name))
#     w,h,d = img_arr.shape
#     img_arr_rs = img_arr
#     img_arr_rs = resize(img_arr, (target_w, target_h))
#     all_imgs.append(img_arr_rs)
#     all_imgs_orig.append(img_arr)
#     all_labels.append(2)

from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D,AveragePooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.layers import Dense, Concatenate
from keras.models import load_model, Model
from keras.optimizers import Adam
from keras.utils import to_categorical

from sklearn.model_selection import train_test_split


X = np.array(all_imgs, dtype=np.float32)
Y = to_categorical(np.array(all_labels),num_classes=2)
Y = Y[:,0]
print(X.shape)
print(Y.shape)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,


from vgg16 import VGG16
from keras.preprocessing import image
from imagenet_utils import preprocess_input
```

```
model = VGG16(weights='imagenet', include_top=False)

X = preprocess_input(X)
print(X.shape)

features = model.predict(X)

#np.save("pasta_features",features)

print(features.shape)
print(type(features))

model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same', input_shape=(9,9,512)))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature v
model.add(Dense(10))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(1))
model.add(Activation('sigmoid'))
optimizer = Adam()
model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['

batch_size = 32
n_epochs = 200

features_train, features_test, Y_train, Y_test = train_test_split(feature

history = model.fit(features_train,Y_train,epochs=n_epochs,batch_size=bat
```