CONCEPTS & ENVIRONMENT SET-UPS

Make sure that your laptop can run basic Python3 programs. (e.g. Hello World)
If you haven't installed Python, please proceed to install PyCharm.

https://www.youtube.com/watch?v=cHa85et7LK0
The installation instruction starts from 2:40.

IMAGE CLASSIFICATION WORKSHOP

BY

On Campus

The Institution of Engineering and Technology

LET'S START

Outline

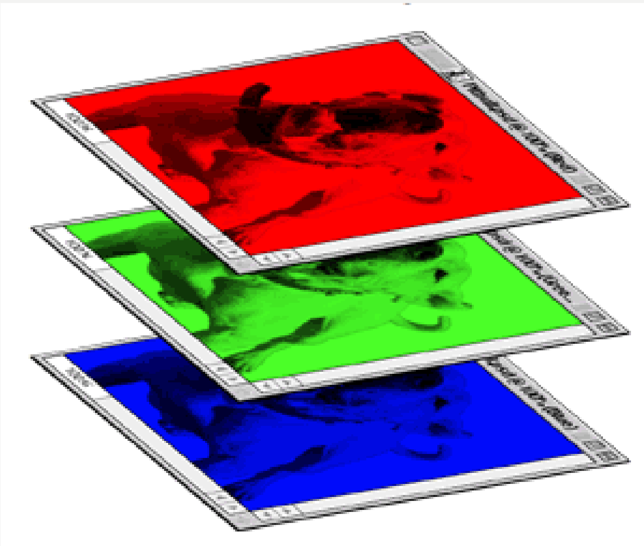
1. Image Classification (IC) Concepts & Environment Set-ups
2. Image formatting & processing
3. Model Built-up & Model Training
4. Make predictions with the Model

CONCEPTS & ENVIRONMENT SET-UPS

How do Computers achieve **Image Classification**?

When a computer takes an image as input, it will see an array of pixel values.

Depending on the resolution and size of the image, it will see a WIDTH x HEIGHT x 3 array of numbers. For example, suppose we have a colorful image with a size 480 x 480. The array seen by the computer will be 480 x 480 x 3.



The computer will then output **<u>the probability</u>** of the image being a certain class based on the manipulation of the array.

CONCEPTS & ENVIRONMENT SET-UPS

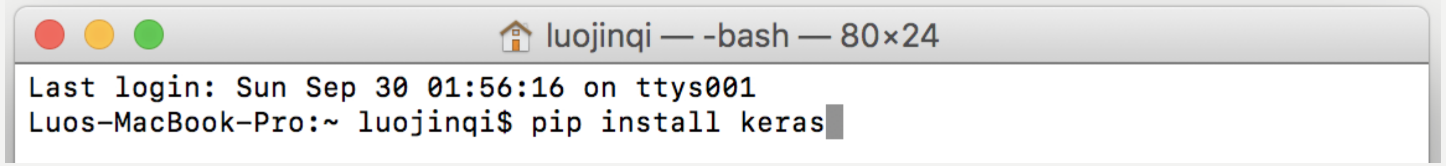
How is it relevant to Python?

Keras with TensorFlow is a high-level library set in Python to build and train **Neural Networks**.

1. User friendly
2. Modular and composable
3. Easy to extend
4. Developed by Google – High Quality!

Easy to start with 0 basic knowledges!

CONCEPTS & ENVIRONMENT SET-UPS



The operation is the same in Windows Command Window & MacOS Terminal:
pip install numpy
pip install scipy
pip install tensorflow
pip install keras

Please also do these after pip install:
pip3 install keras
pip3 install tensorflow

That will install the libraries on your Python3 environment.

CONCEPTS & ENVIRONMENT SET-UPS

Download the zipped pack from link:
That contains all the materials that we need today. (Slides also included)

Run **test_environment.py** to check whether your libraries have been installed.

Wait a few seconds to see the output of
Ignore the Warning Notes which programmers never regard.

If the message "Congrats" is shown, the your environment is ready to go.
Inform the speaker if any bugs happen.

CONCEPTS & ENVIRONMENT SET-UPS

Type the following lines of codes to set up your coding environment.
(Or just copy the corresponding part from **sample_model.py**)

Now you are ready to process the images and train the model.
Before we leap: We have to know that the model training is actually calculating the parameters which will help the model make correct decisions.

```
# coding: utf-8


from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Conv1D, MaxPooling1D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
```

IMAGE FORMATTING & PROCESSING

Approaches that alter the training data in ways that **change the array representation while keeping the object (label) the same** are known as **data augmentation**.

It allows us to artificially **expand our dataset**. Some popular augmentations are horizontal flips, random crops, translations, rotations, and so on.

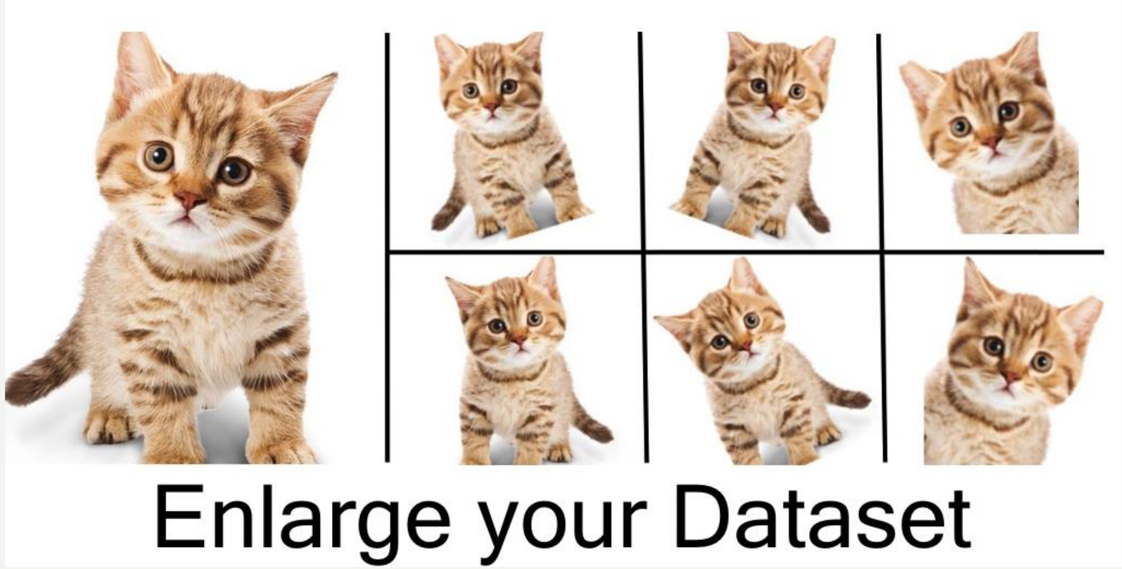


IMAGE FORMATTING & PROCESSING

Here we implement a generator from Keras.
(Type or copy the ImageDataGenerator from **sample_model.py)**

```
datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

rotation_range: Degree range for random rotations
width_shift_range: Fraction range of total width for shifting.
height_shift_range: Fraction range of total height for shifting.
share_range
zoom_range: Fraction range for random zoom.
horizontal_flip: Randomly flip inputs horizontally.
fill_mode

If you cannot remember so much terms, don't worry as this not the key things in IC.

IMAGE FORMATTING & PROCESSING

Then we set up the generator for **training sets** and **validation sets**.
(Type or copy the generator from **sample_model.py)**

```
batch_size = 128

# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.12,
        zoom_range=0.12,
        horizontal_flip=True
        )

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

IMAGE FORMATTING & PROCESSING

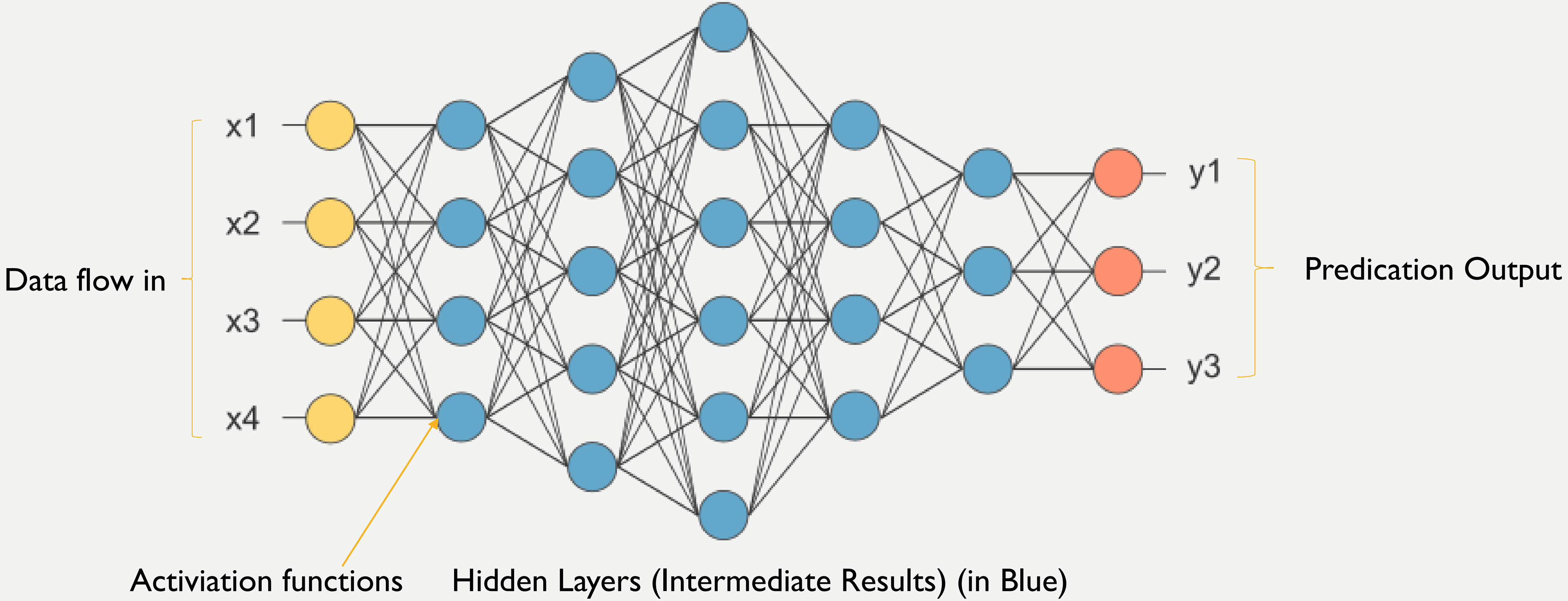
The training and validation images are ready to **flow in** the program!
(Type or copy the code from **sample_model.py)**

```
# Below is a generator that will read pictures found in
# subfolers of 'Training_Images', and indefinitely generate
# batches of augmented image data. Each batch is of size 128.
train_generator = train_datagen.flow_from_directory(
        "C:/Users/xxx/Desktop/IET IC Workshop/Training Images",  # this is the target directory
        target_size=(100, 100),  # all images will be resized to 100x100
        batch_size=batch_size,
        class_mode='categorical')
```

```
# this is a similar flow-in, for validation data
validation_generator = test_datagen.flow_from_directory(
        'C:/Users/xxx/Desktop/IET IC Workshop/Validation Images',
        target_size=(100, 100),
        batch_size=batch_size,
        class_mode='categorical')
```

MODEL BUILT-UP & MODEL TRAINING

Neural Networks Models are basically
the combination of different Math Manipulations & Activation Functions.



MODEL BUILT-UP & MODEL TRAINING

Here we implement a naïve neural network by hand.

```
model = Sequential()

model.add(Conv2D(100, (3, 3), input_shape=(100, 100, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(100, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(200, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

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(3)) #3 types of images

model.add(Activation('softmax'))

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

This is the whole view of the neural network.

Details of each part will be explained in the following slides.

MODEL BUILT-UP & MODEL TRAINING

Here we implement a naïve neural network by hand.

```
model = Sequential()
```

Sequential():

The function Sequential() is providing an “empty box” for the further implementation.

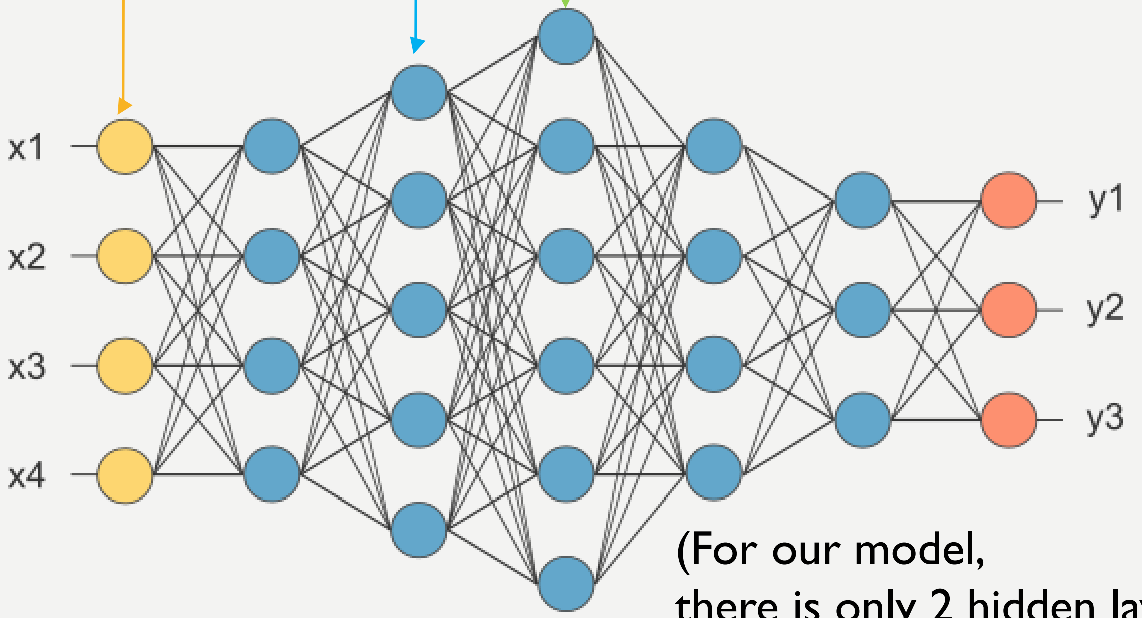
In other words, it is used to initialize the neural network model so that later we can add layers to this model.

MODEL BUILT-UP & MODEL TRAINING

Here we implement a naïve neural network by hand.

```
model.add(Conv2D(100, (3, 3), input_shape=(100, 100, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(100, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(200, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Simple Analogy of
Neural Network Layers

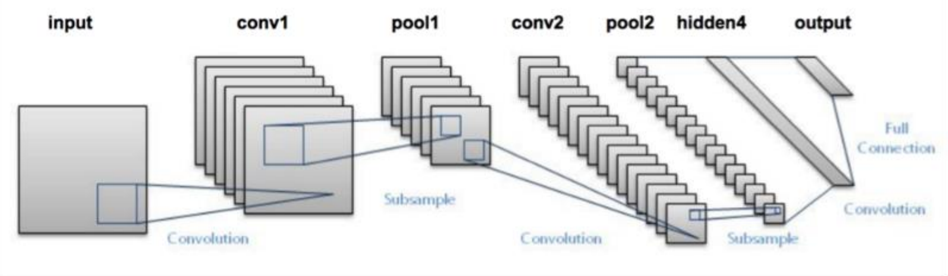


(For our model,
there is only 2 hidden layer)

MODEL BUILT-UP & MODEL TRAINING

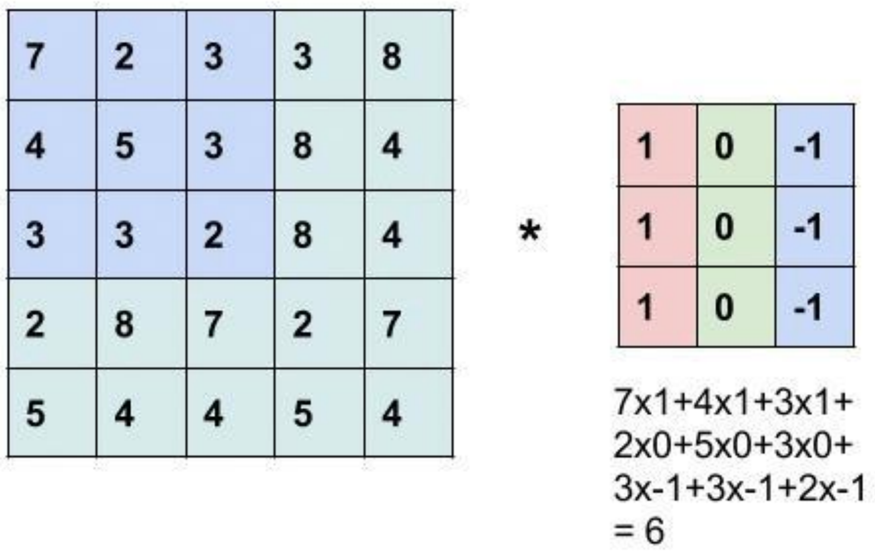
Here we implement a naïve neural network by hand.

```
model.add(Conv2D(100, (3, 3), input_shape=(100, 100, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(100, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(200, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```



Conv2D:
The convolution is usually the first step of a **Convolutional Neural Network (CNN)** on the training images. Each of the convolutions can be imagined as **a flashlight** that is shedding light upon and sliding over the image.

The flashlight in this layer is looking for **specific features**. If they find the features they are looking for, **they produce a high activation**. Since we are working on images which are 2D arrays we're using Conv2D.

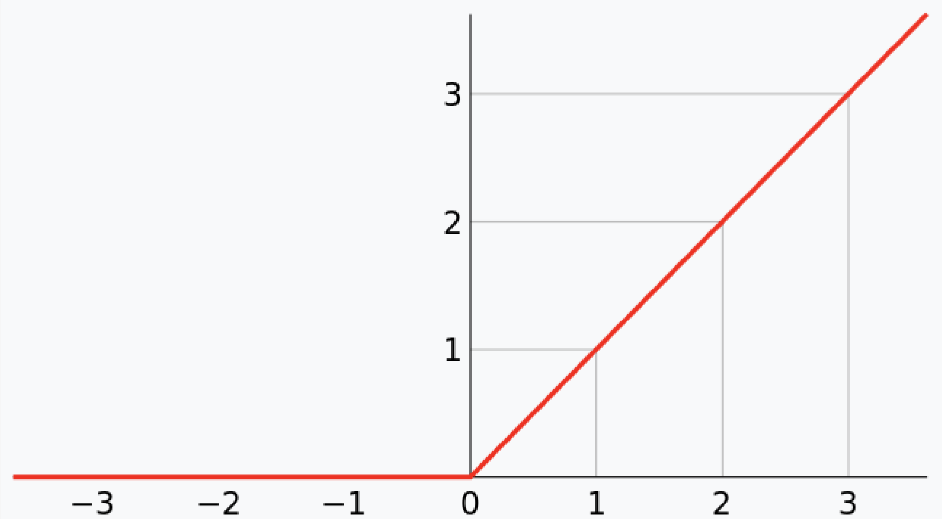


How Conv2D works

MODEL BUILT-UP & MODEL TRAINING

Here we implement a naïve neural network by hand.

```
model.add(Conv2D(100, (3, 3), input_shape=(100, 100, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(100, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(200, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```



ReLU:

Rectified Linear Unit (ReLU) is a kind of widely-used activation functions in CNN. It finds wide applications in the area of Computer Vision and Speech Recognition.

$$f(x) = \max(0, x)$$

"f(X) = X for any non-negative X"

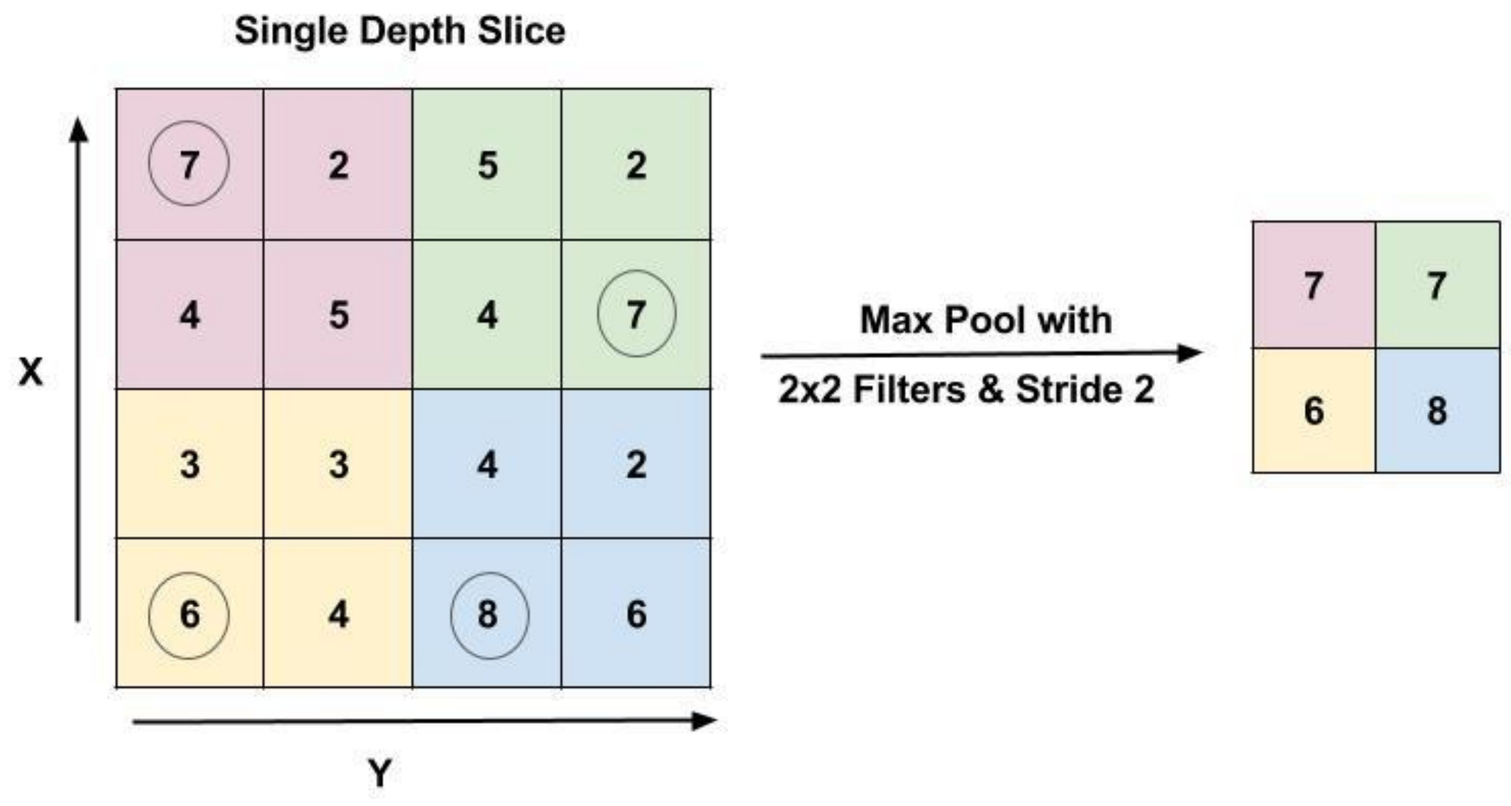
MODEL BUILT-UP & MODEL TRAINING

Here we implement a naïve neural network by hand.

```
model.add(Conv2D(100, (3, 3), input_shape=(100, 100, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(100, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(200, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

MaxPooling2D:
MaxPooling2D is used for pooling operation. Pooling is mostly used immediately after the convolution to **reduce the overwhelming data size** (only width and height).

This reduces the number of parameters, hence reducing the computation and increasing efficiency.



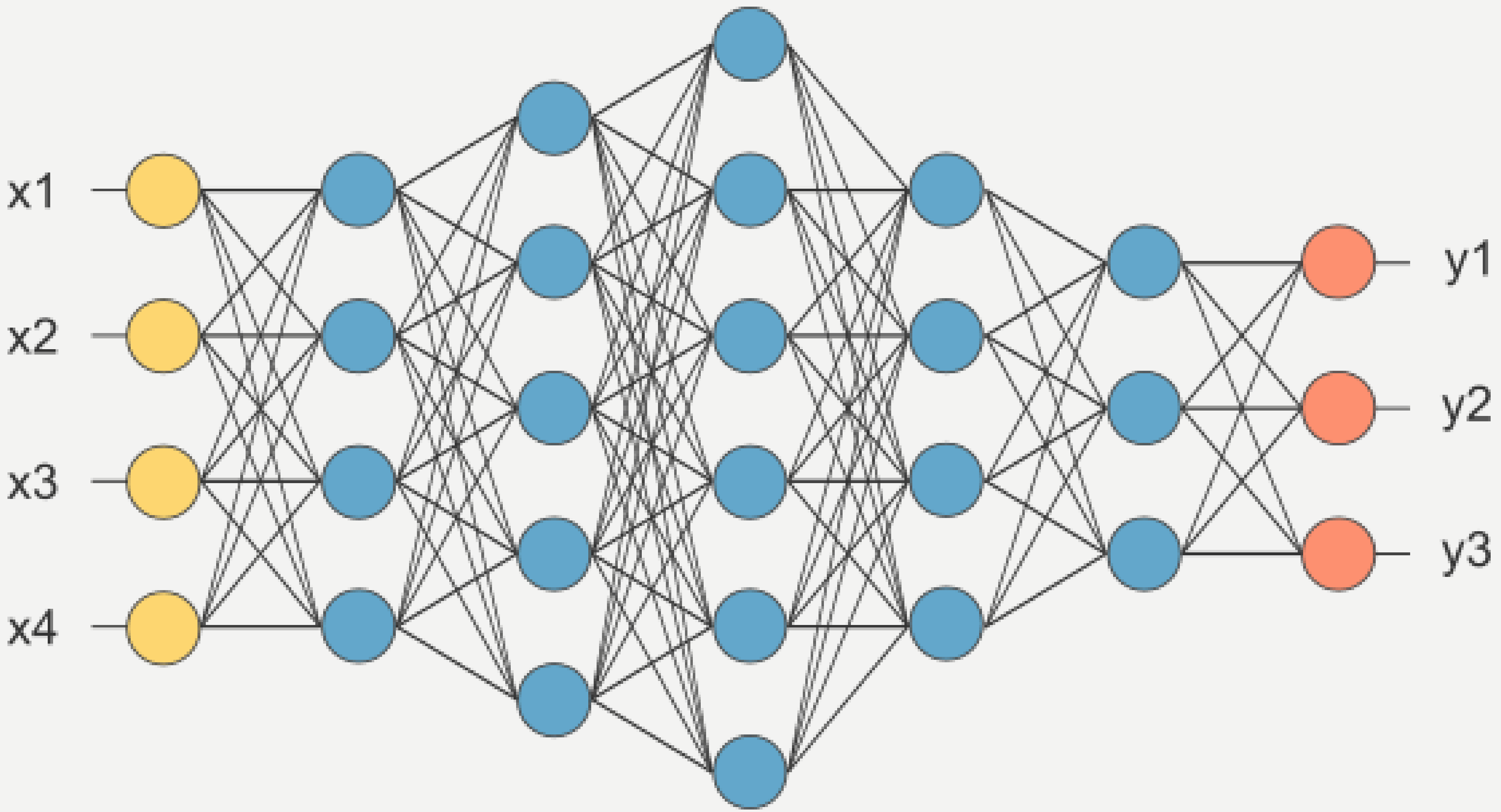
How MaxPooling2D works

MODEL BUILT-UP & MODEL TRAINING

Here we implement a naïve neural network by hand.

```
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(3)) #3 types of images

model.add(Activation('softmax'))

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

Flatten: Flattening is the process of converting all 3-D arrays into a single long continuous linear vector.
Dense: We also import Dense to perform the full connection of the neural network. A dense layer is a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous layer, thus densely connected.



Densely Connected

MODEL BUILT-UP & MODEL TRAINING

Here we implement a naïve neural network by hand.

```
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(3)) #3 types of images

model.add(Activation('softmax'))

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

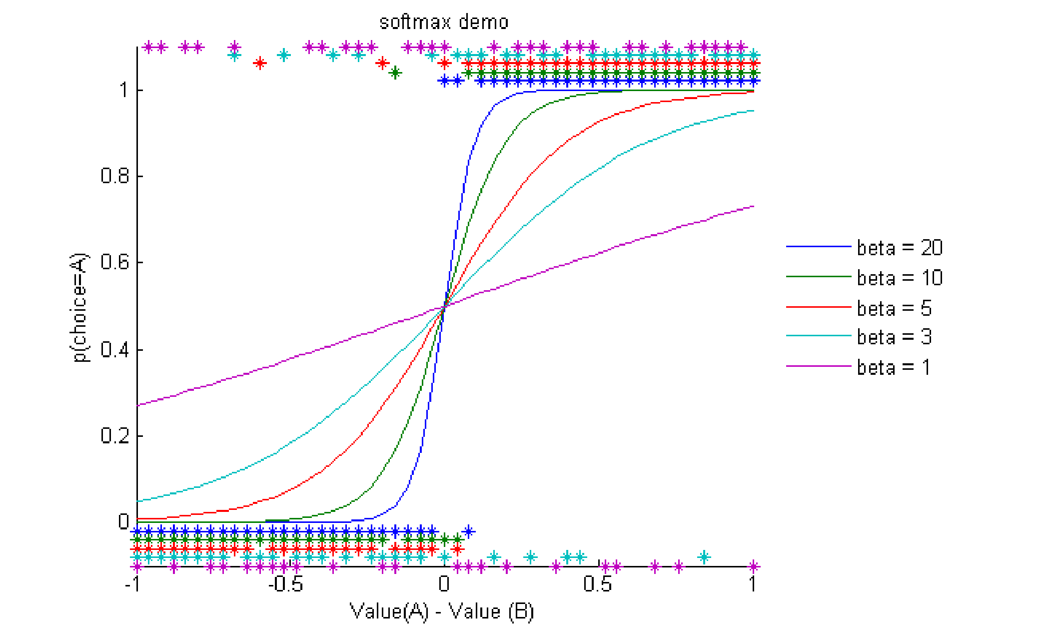
Flatten: Flattening is the process of converting all 3-D arrays into **a single long continuous linear vector.**
Dense: We also import Dense to perform the full connection of the neural network. A dense layer is a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous layer, thus densely connected.

MODEL BUILT-UP & MODEL TRAINING

Here we implement a naïve neural network by hand.

```
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(3)) #3 types of images

model.add(Activation('softmax'))

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

Softmax:
The smooth function that ranges in (0,1).
It converges input X into a **correspondingly-large** value in (0,1)



MODEL BUILT-UP & MODEL TRAINING

Here we implement a naïve neural network by hand.

```
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(3)) #3 types of images

model.add(Activation('softmax'))

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

Lastly, the **model.compile** gather up all the settings above and packed up the model for further training.

MODEL BUILT-UP & MODEL TRAINING

Here we implement a naïve neural network by hand.

```
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(3)) #3 types of images

model.add(Activation('softmax'))

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

loss:

The function is to represent **the difference between the actual output and the predicted output** by machine. During our training of the model, we will attempt to minimize the value. **binary_crossentropy** is usually used for a binary problem.

MODEL BUILT-UP & MODEL TRAINING

Here we implement a naïve neural network by hand.

```
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(3)) #3 types of images

model.add(Activation('softmax'))

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

optimizer: The function is to update the internal parameters of a model in such a way that **the loss function is minimized.**

metrics: List of data. accuracy means the percentage of correct answers. Metric values are recorded at the end of each epoch on the training dataset. **It represents how well the model works just like the GPA.**

MODEL BUILT-UP & MODEL TRAINING

Here we implement a naïve neural network models by hand.

```
model = Sequential()

model.add(Conv2D(100, (3, 3), input_shape=(100, 100, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(100, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(200, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

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(3)) #3 types of images

model.add(Activation('softmax'))

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

Up to now, we have briefly explained how the functions in the model works.

If you cannot memorize all the terms, don't worry. Many engineers just use pre-implemented models proposed by CS research institutions (e.g. CMU, UC Berkeley, Tsinghua, NTU, ~~NUS~~)

MODEL BUILT-UP & MODEL TRAINING

Last step: set the configuration to save the **trained** parameter of your model

```
bacth_size = 128

model.fit_generator(
        train_generator,
        steps_per_epoch=30000// batch_size,
        epochs=3,
        validation_data=validation_generator,
        validation_steps=30000 // batch_size)
model.save_weights('result_parameters_weight.h5')
```

validation data:
The data generator we have created previously for validation image data.

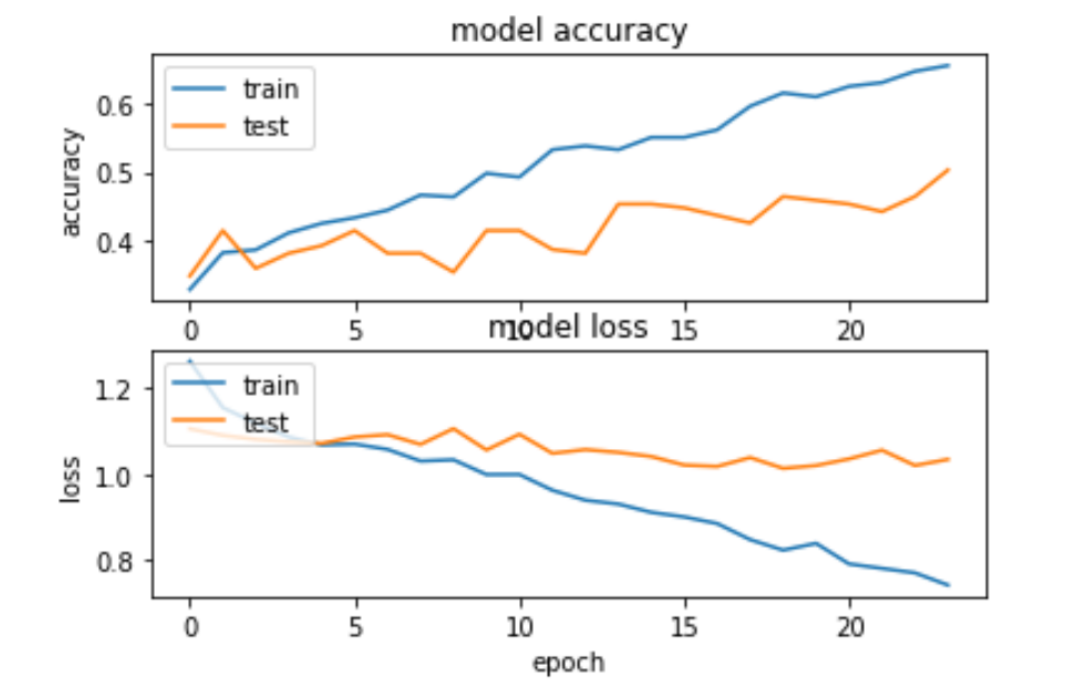
generator:
The data generator we have created previously for training image data.

epochs:
For each single epoch, the neural network is trained on every training samples to update its parameters.

MODEL BUILT-UP & MODEL TRAINING

Run the whole file and wait until it terminates.

This may take a few minutes. Please inform the speaker if any bugs happen.



MAKE PREDICTIONS WITH THE MODEL

Now you have a **result.h5** file which stores the parameter that you calculated.

You can begin using this result to classify Baby Hats, Pants & Shirts images.

Do remember to put your own pictures of jpg format in the **your-own-images** folder and edit the **predict_with_trained_parameters.py** with your own file path.

Run this py file to see the classification result of your images.