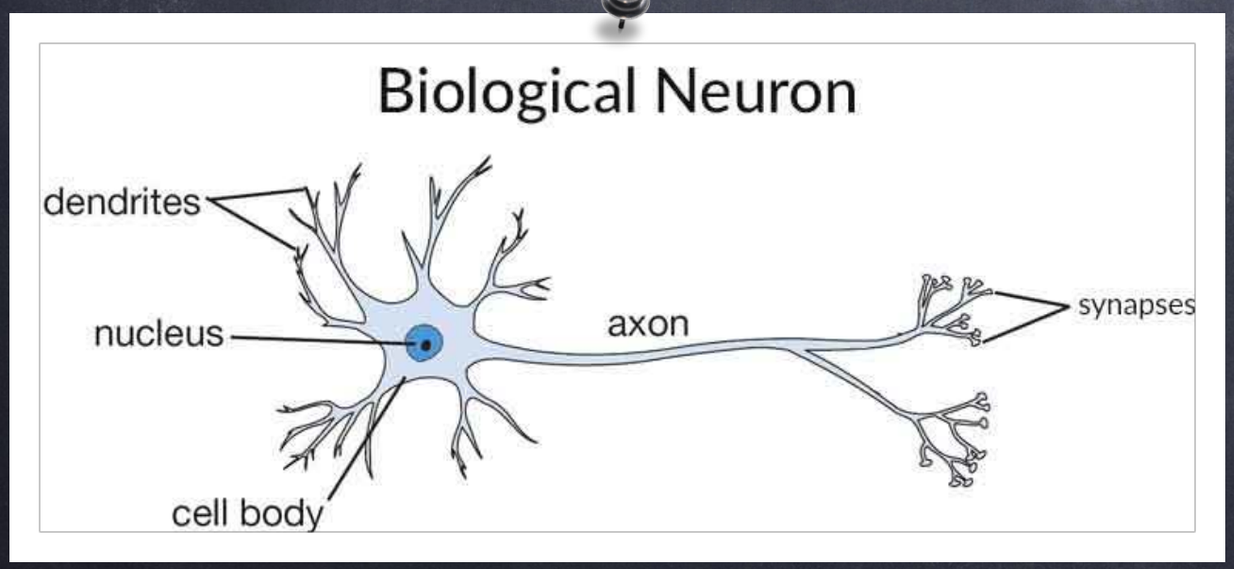
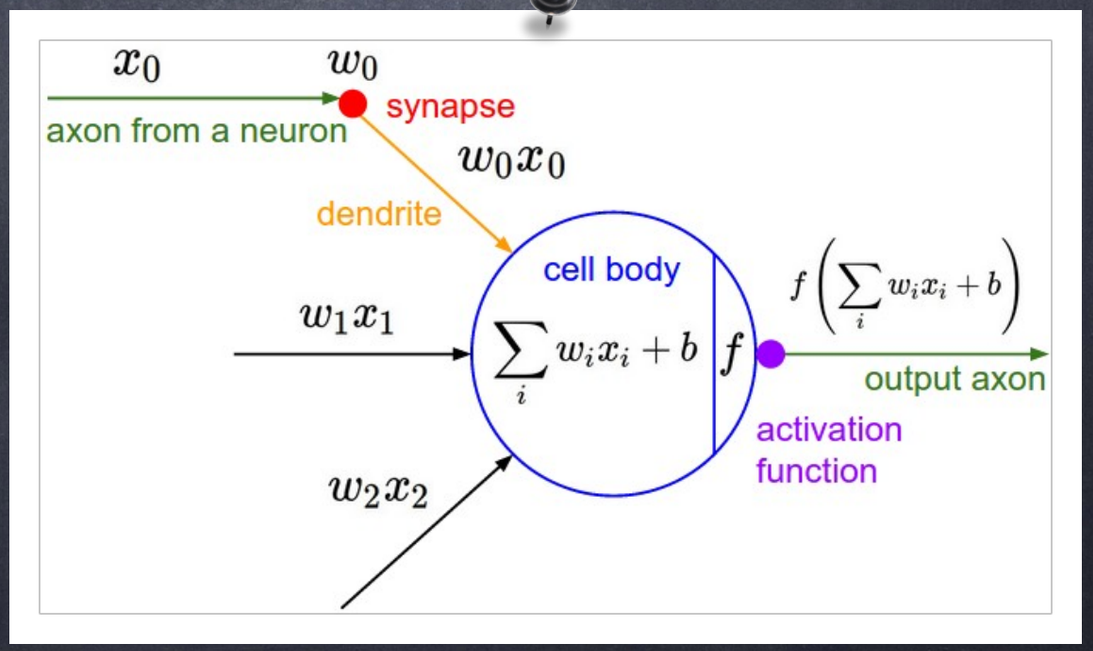
The Simpsons Characters Data

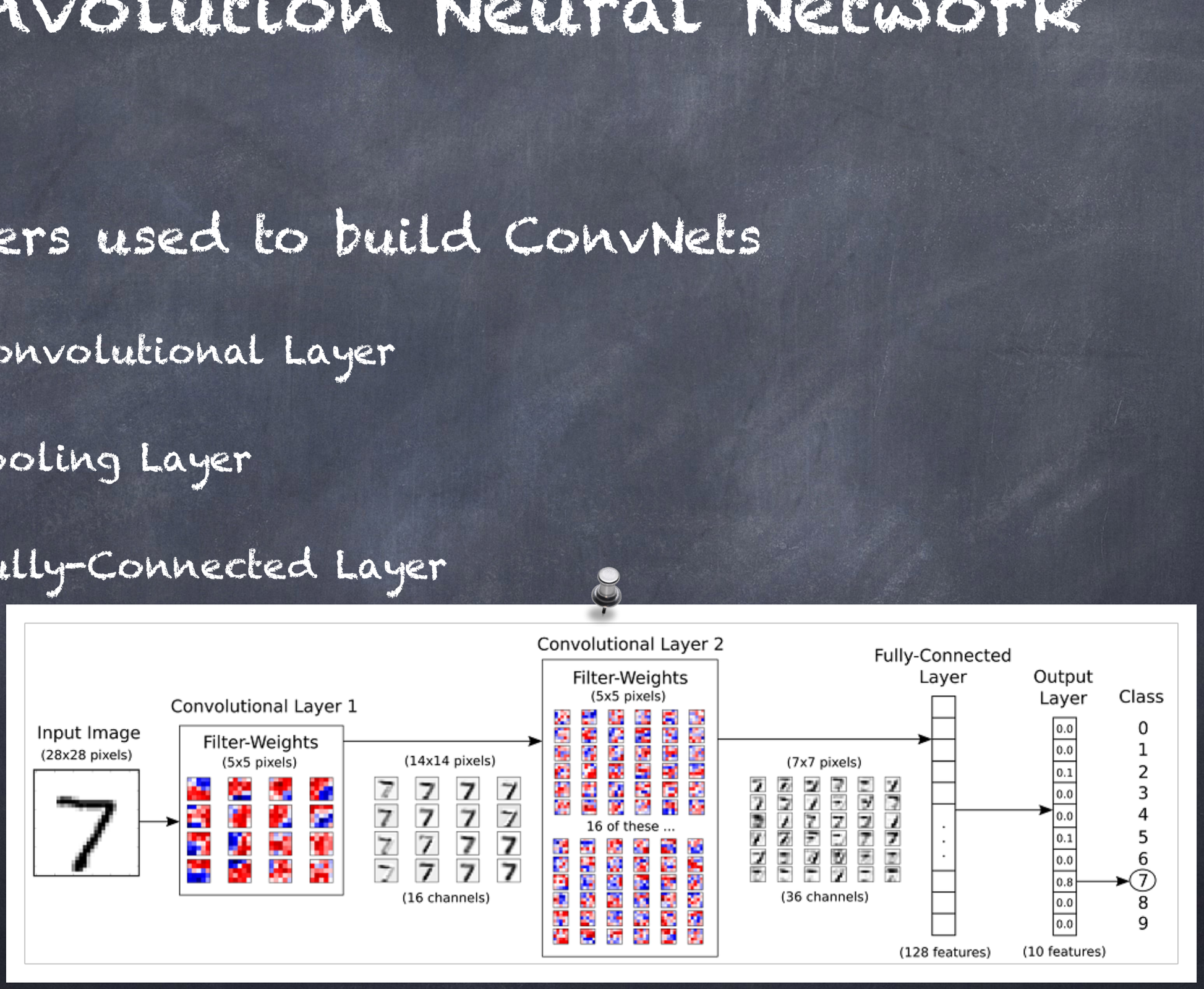
U10416008丁杰





Convolution Neural Network

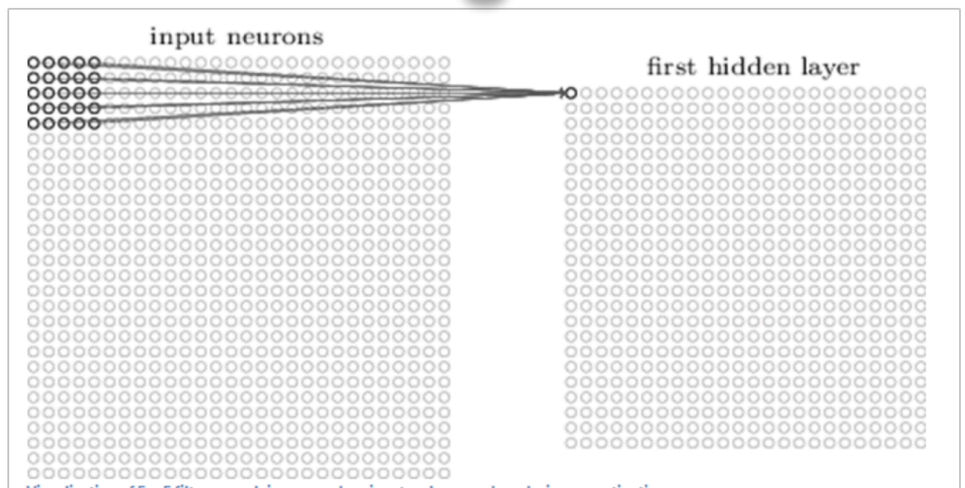
- Layers used to build ConvNets
  - Convolutional Layer
  - Pooling Layer
  - Fully-Connected Layer



Convolutional Layer

- The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume.

- For example, the input is a 32 (width) x 32 (height) x 3 (channel-RGB) array of pixel values. Suppose the dimensions of the filter is 5 (width) x 5 (height) x 3 (channel-RGB), After the filter is sliding, or convolving, around the input image, you will find out that what you're left with is a 28 x 28 x 1 array of numbers, which we call an activation map or feature map.



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

Convolutional Layer

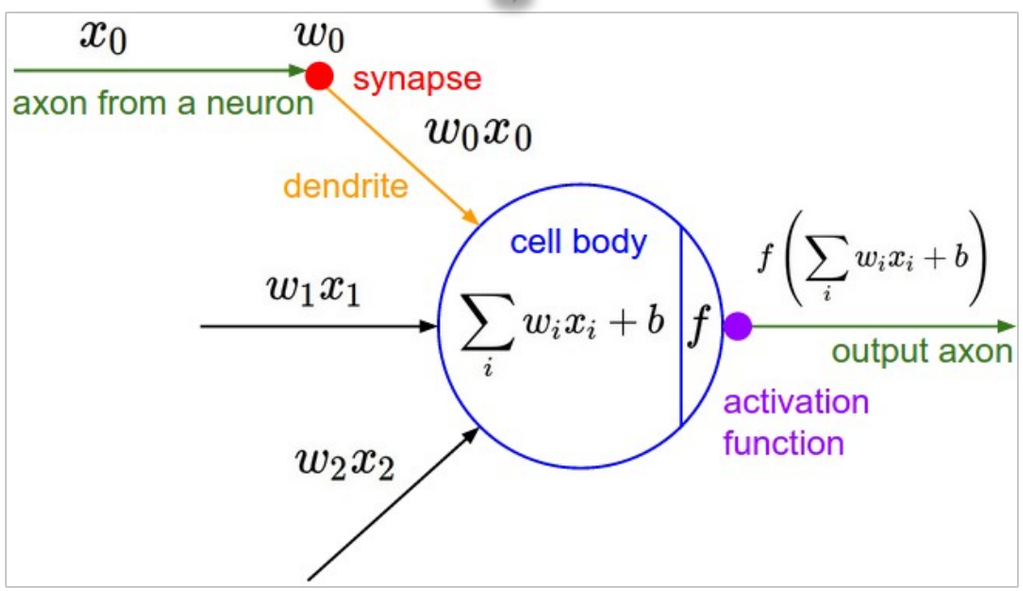
- Spatial arrangement
  - receptive field : a local region of the input volume (equivalently this is the filter size)
  - Depth : it corresponds to the number of filters we would like to use, each learning to look for something different in the input.
  - Stride : When the stride is 1 (2) then we move the filters one (two) pixel at a time. This will produce smaller output volumes spatially.
  - zero-padding : it will allow us to control the spatial size of the output volumes.

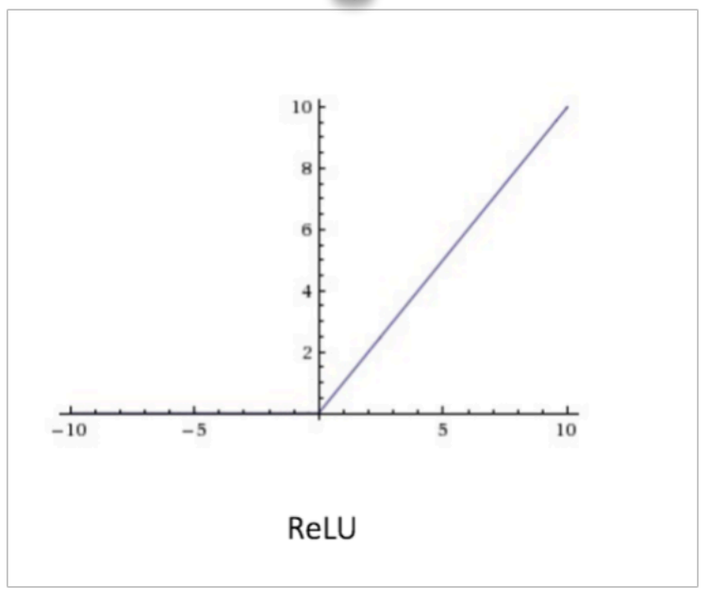
Convolutional Layer

- BackPropagation
  - forward pass : take a training image and pass it through the whole network
  - loss function : We want to get to a point where the predicted label (output of the ConvNet) is the same as the training label. In order to get there. Minimize the amount of loss.
  - backward pass : determining which weights contributed most to the loss and finding ways to adjust them so that the loss decreases.
  - weight update : all the weights of the filters and update them so that they change in the opposite direction of the gradient.

Activation Function - ReLU

- RELU layer will apply an elementwise activation function, such as the max(0,x) thresholding at zero. This leaves the size of the volume unchanged





Activation Function - ReLU

- Why ReLU ?
- One major benefit is the reduced likelihood of the gradient to vanish. In contrast, the gradient of sigmoids becomes increasingly small as the absolute value of x increases.
- The other benefit is sparsity(when $x \leq 0$). The more such units that exist in a layer the more sparse the resulting representation. In contrast, sigmoids are always likely to generate some non-zero value resulting in dense representations.

Pooling Layer

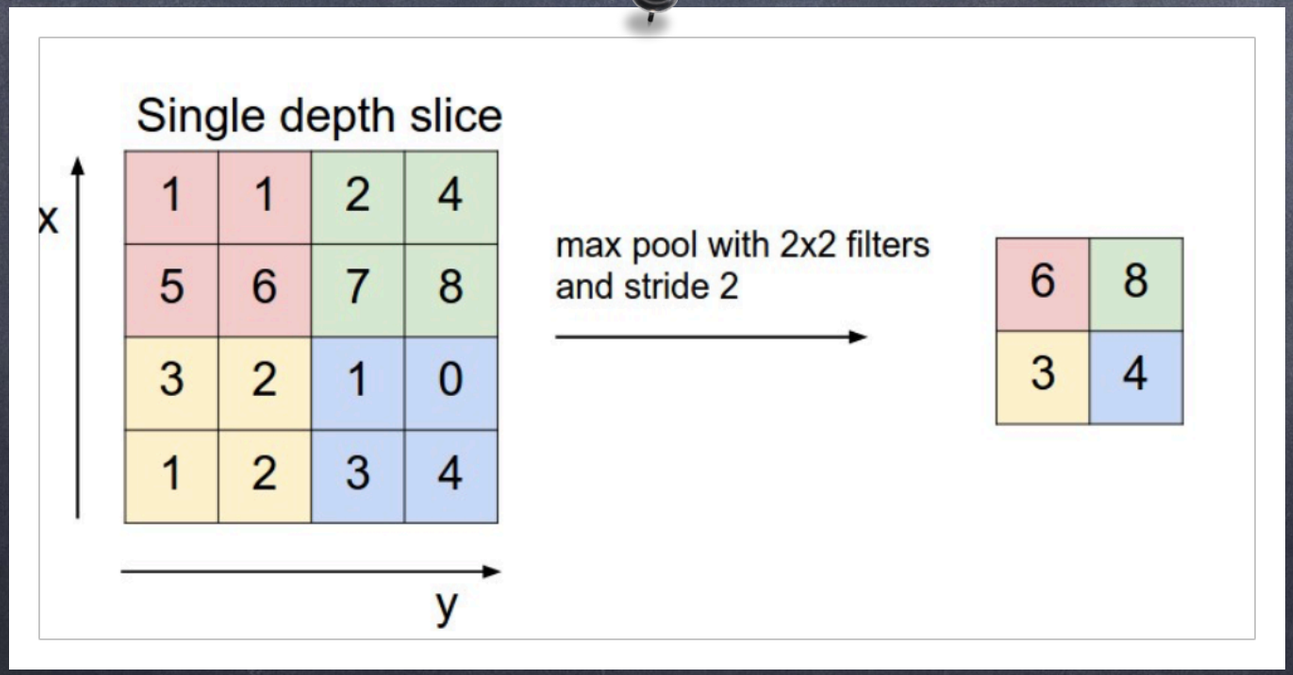
- Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.

- operates independently on every depth slice of the input and resizes it spatially, using the MAX operation.

- In addition to max pooling, the pooling units can also perform other functions, such as average pooling or even L2-norm pooling. Max pooling operation has been shown to work better in practice.

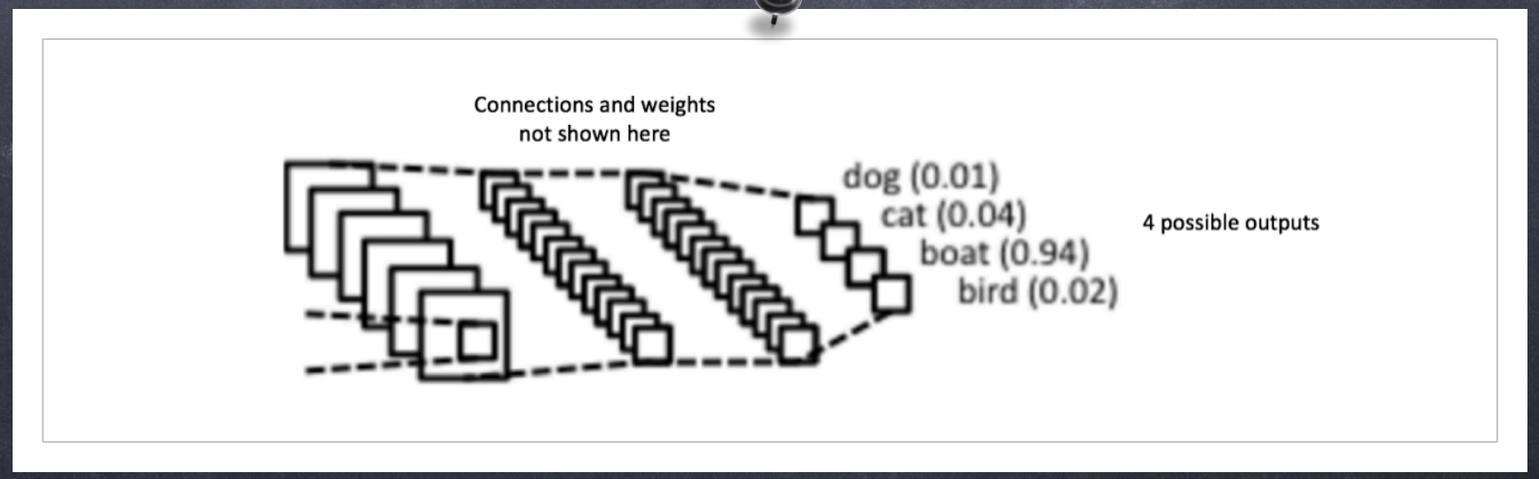
Pooling Layer

- The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations.

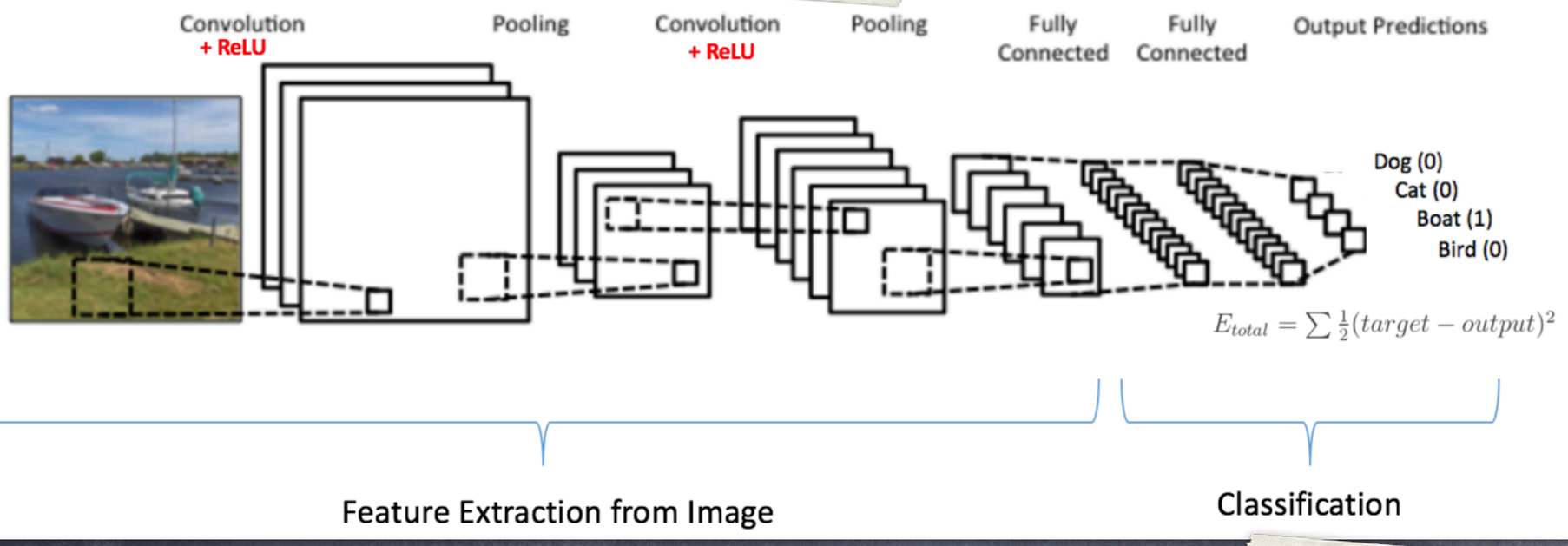


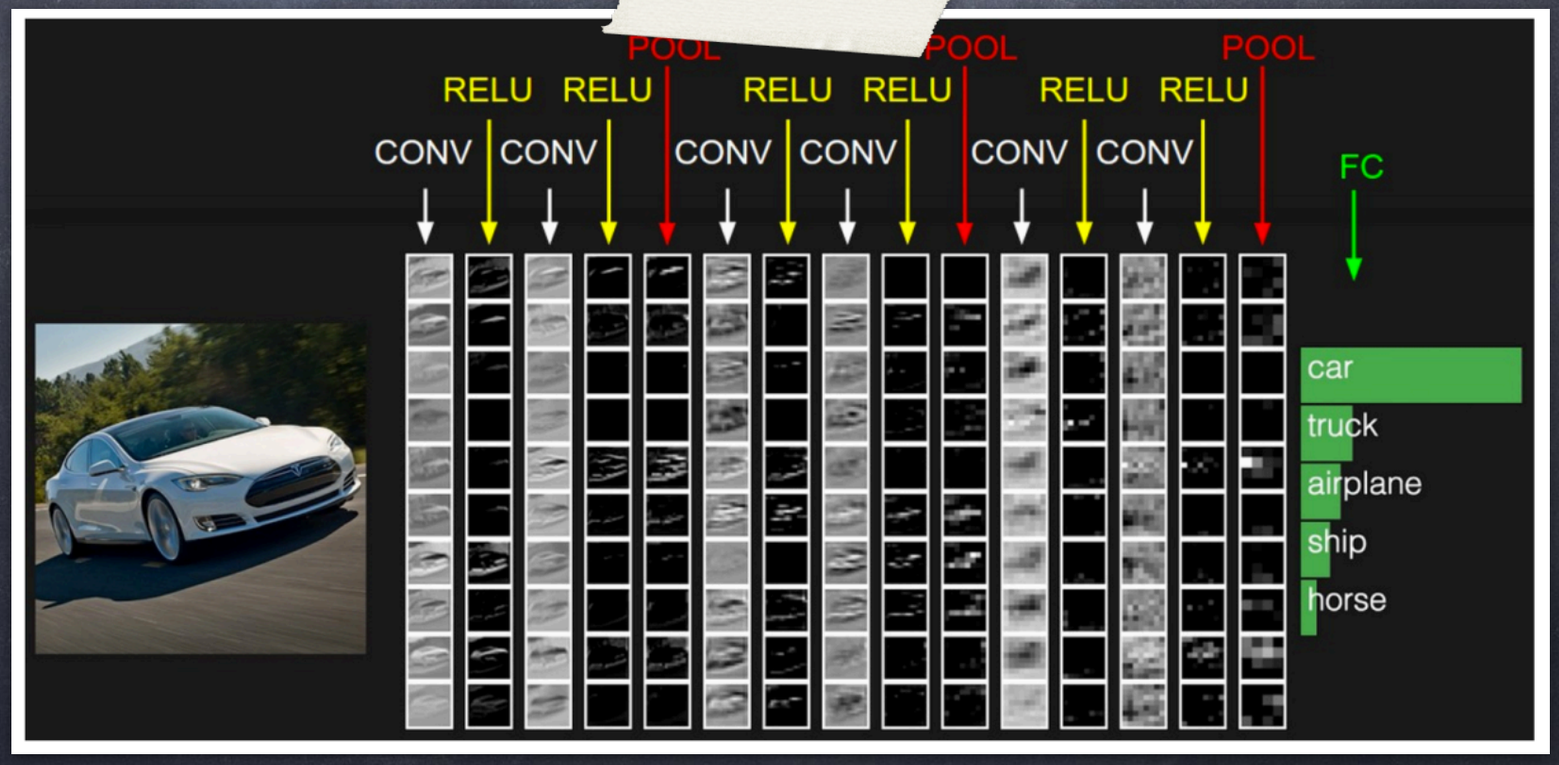
Fully-Connected Layer

- The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.
- The Fully Connected layer is a traditional Multi Layer Perceptron that uses a **softmax** (or SVM) activation function in the output layer



Putting it all together





Other

- Loss Function (Cross Entropy)
- Optimizer (Rmsprop)
- Regularization (Dropout)
- Validation

Loss Function - Cross Entropy

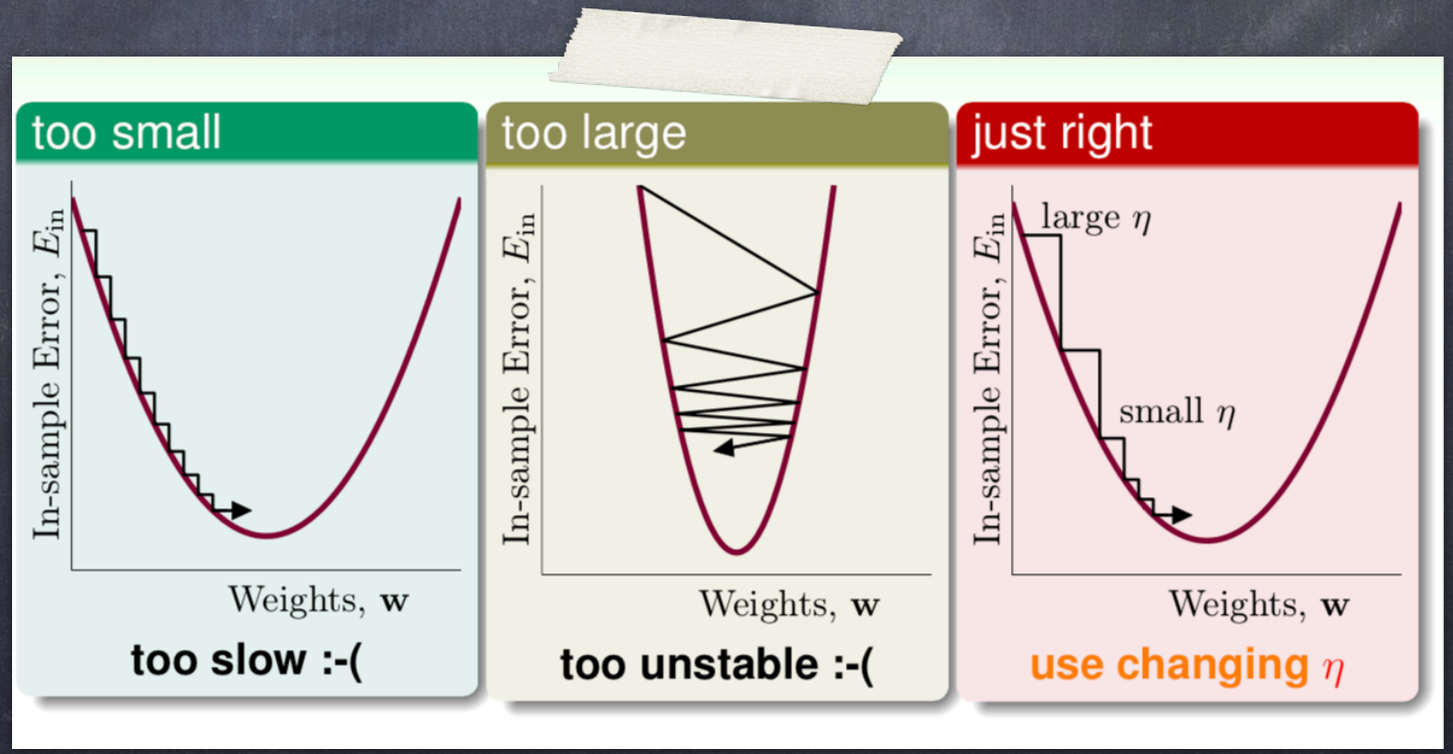
- Cross Entropy is a Loss Function in the model used to classify the data into multiple class

$$\text{cross-entropy} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{k} t_{i,j}\log(p_{i,j})$$

Optimizer

- Optimization algorithms helps us to minimize an Error function E(x) which is simply a mathematical function dependent on the Model's internal learnable parameters which are used in computing the target values(Y) from the set of predictors(X) used in the model.

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \nabla E_{\text{in}}(\mathbf{w}_t)$$



Optimizer - RMSprop

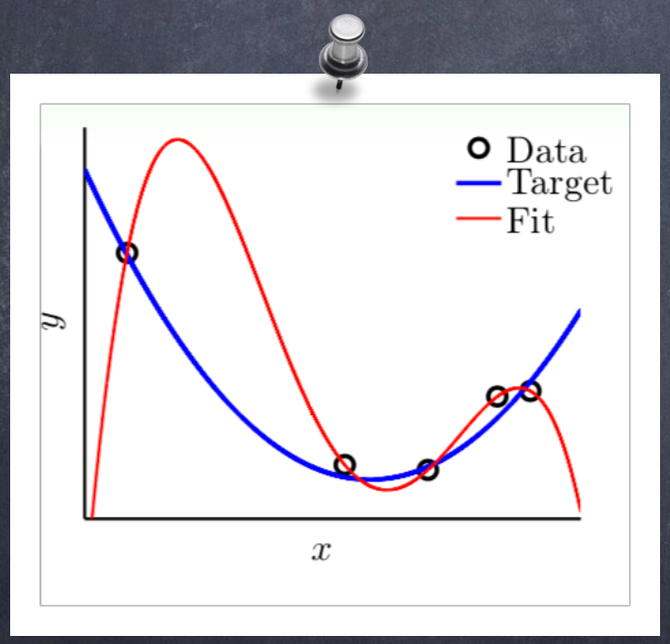
- RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton

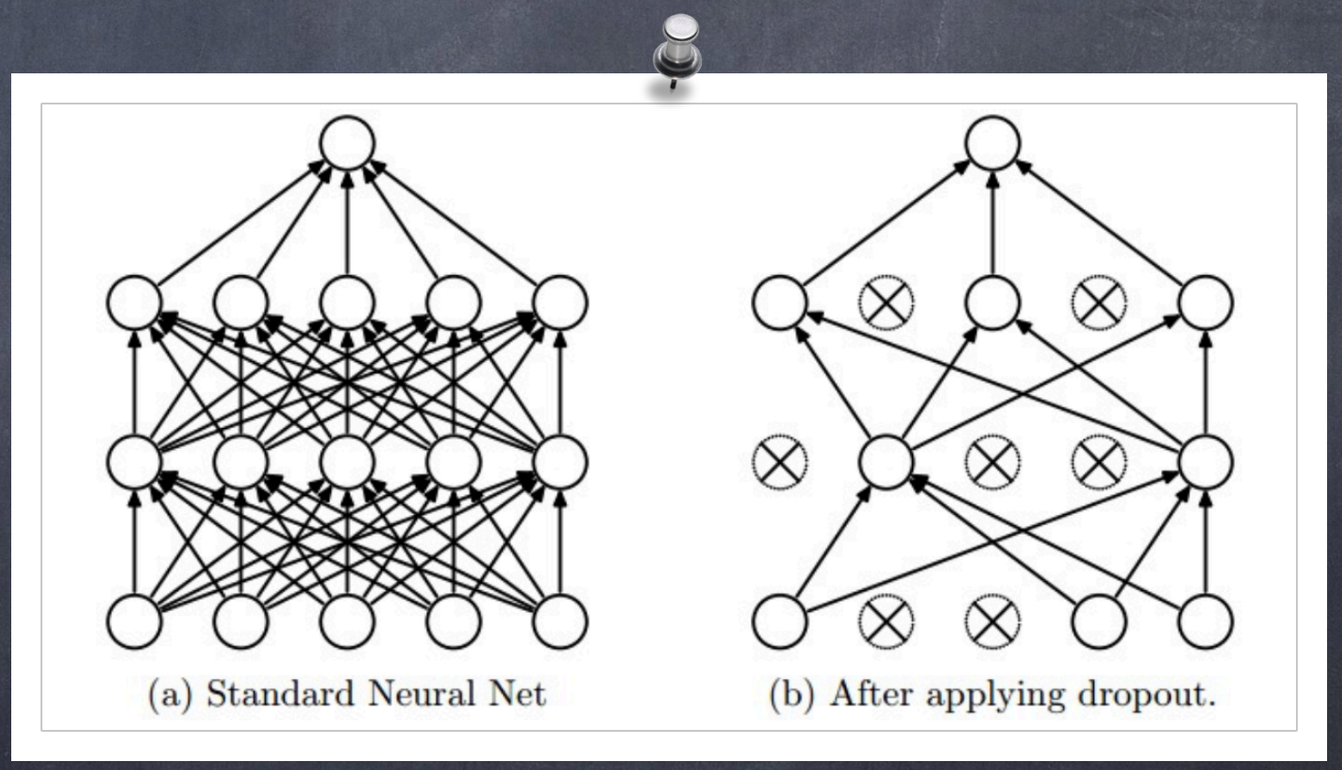
$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Regularization - DropOut

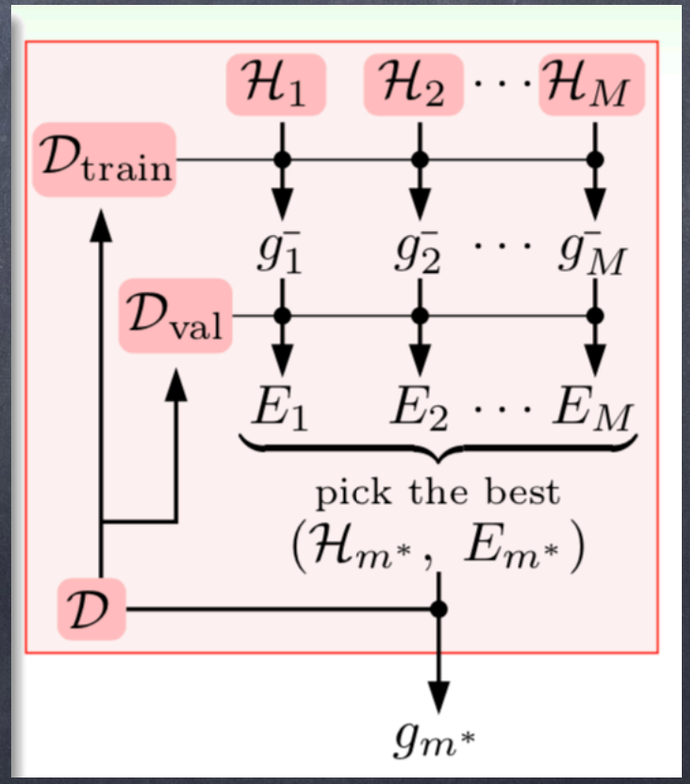
- A Way to Prevent Neural Networks from Overfitting





Validation

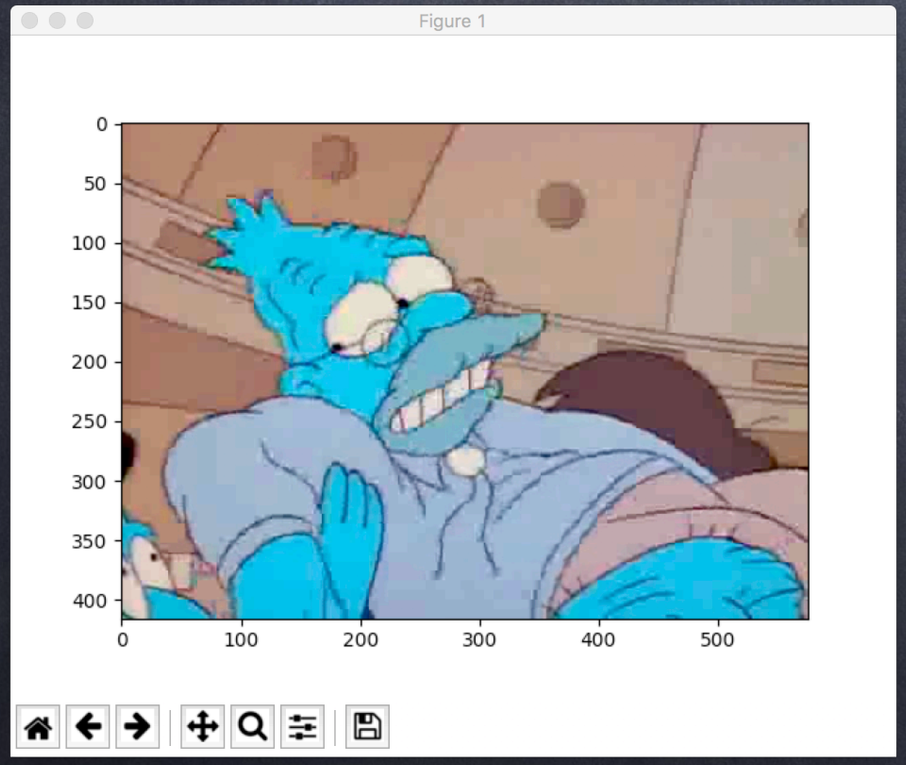
- You can't use a model trained by known data to predict a unknown data



Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.

In order to visualize how CNN works, I test one data of my dataset.




```
ann = Sequential()
x = Conv2D(filters=64,kernel_size=(3,3),input_shape=(64,64,3))
ann.add(x)
ann.add(Activation('relu'))
##ann.add(Conv2D(64, (3, 3)))
##ann.add(Activation('relu'))
##ann.load_weights('test.h5')
x2 = Conv2D(filters=64,kernel_size=(3,3))
ann.add(x2)
ann.add(Activation('relu'))
ann.add(MaxPooling2D(pool_size=(2, 2)))

ann.load_weights('test2.h5')
```

Keras

- In Conv2D(filters = N , kernel_size = (width , height), strides, padding, input_shape) :
  - filters , kernal_size : randomly create the N filters with kernal_size
  - padding : if padding = 'same' , the size of data after a filter sliding is equal to before sliding.
  - input_shape : the size of input_size(width , height , channel(RGB)) => only first layer need to assign

Keras

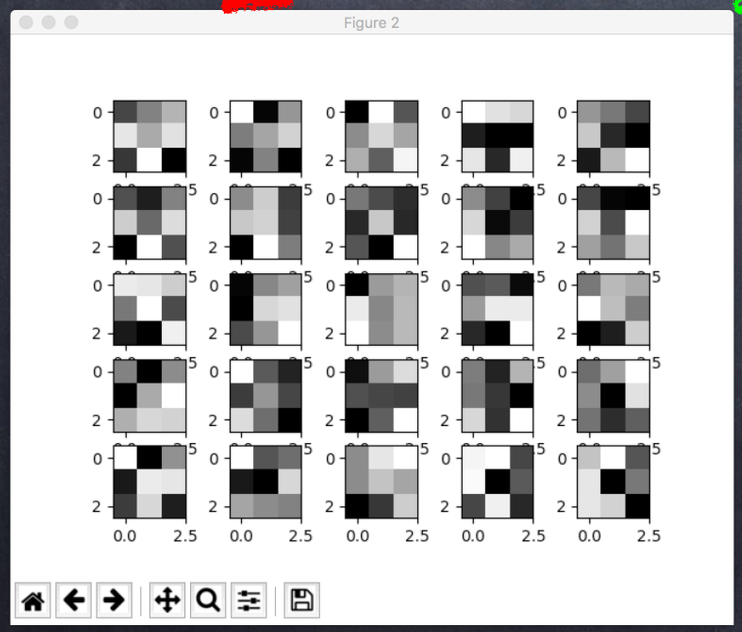
- Here are some example for Conv2D filters(weights) :

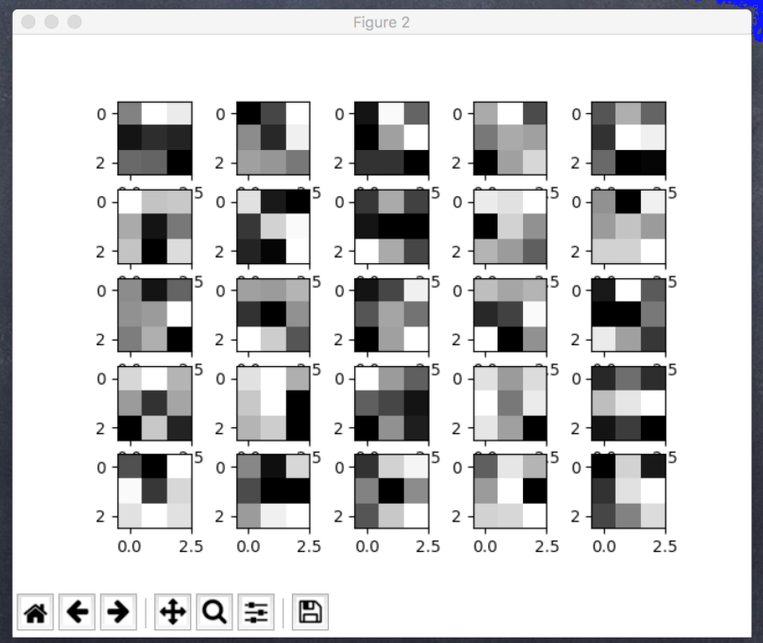
```
binary was not
(3, 3, 3, 64)
```

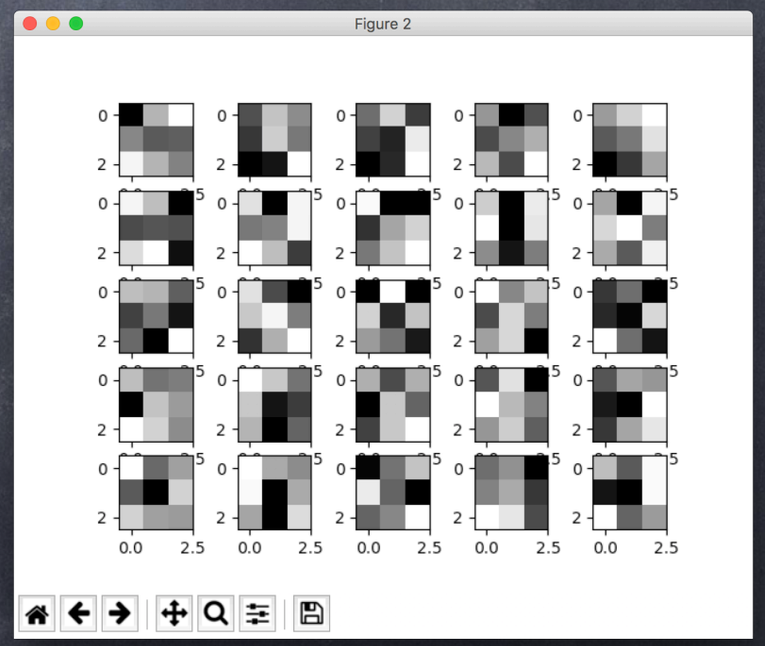
The shape of weights of the first layer

size of channel

size of weights (next size of Channels)







Keras

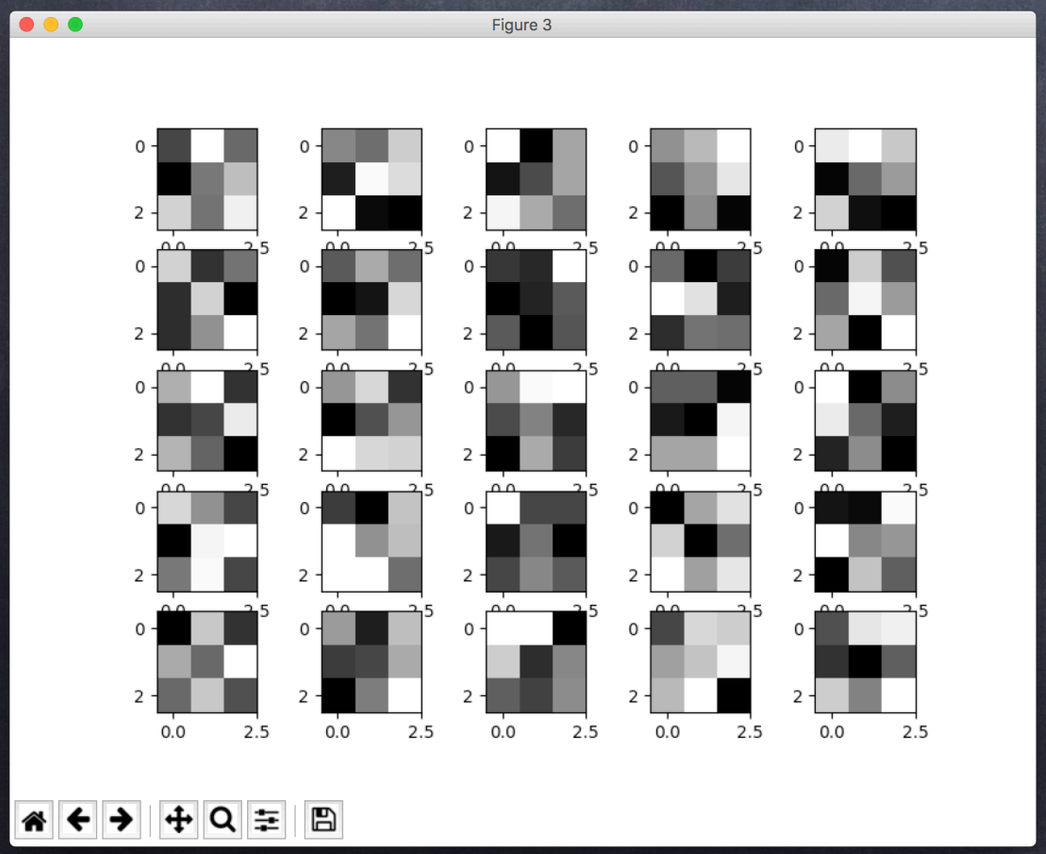
- Here are some example for Conv2D filters(weights) :

```
(3, 3, 64, 64)
```

The shape of weights of the second layer

size of channel(the size of weight in first layer)

size of weights (next size of Channels)



Weight of the first channel in second layer

Keras

- In Activation(activation)
  - Activation : a function to deal with each neural .ex : ReLU , sigmoid (binary classification) , softmax (multi classification)
- In MaxPooling2D(pool_size)
  - pool_size : a size of pooling matrix.

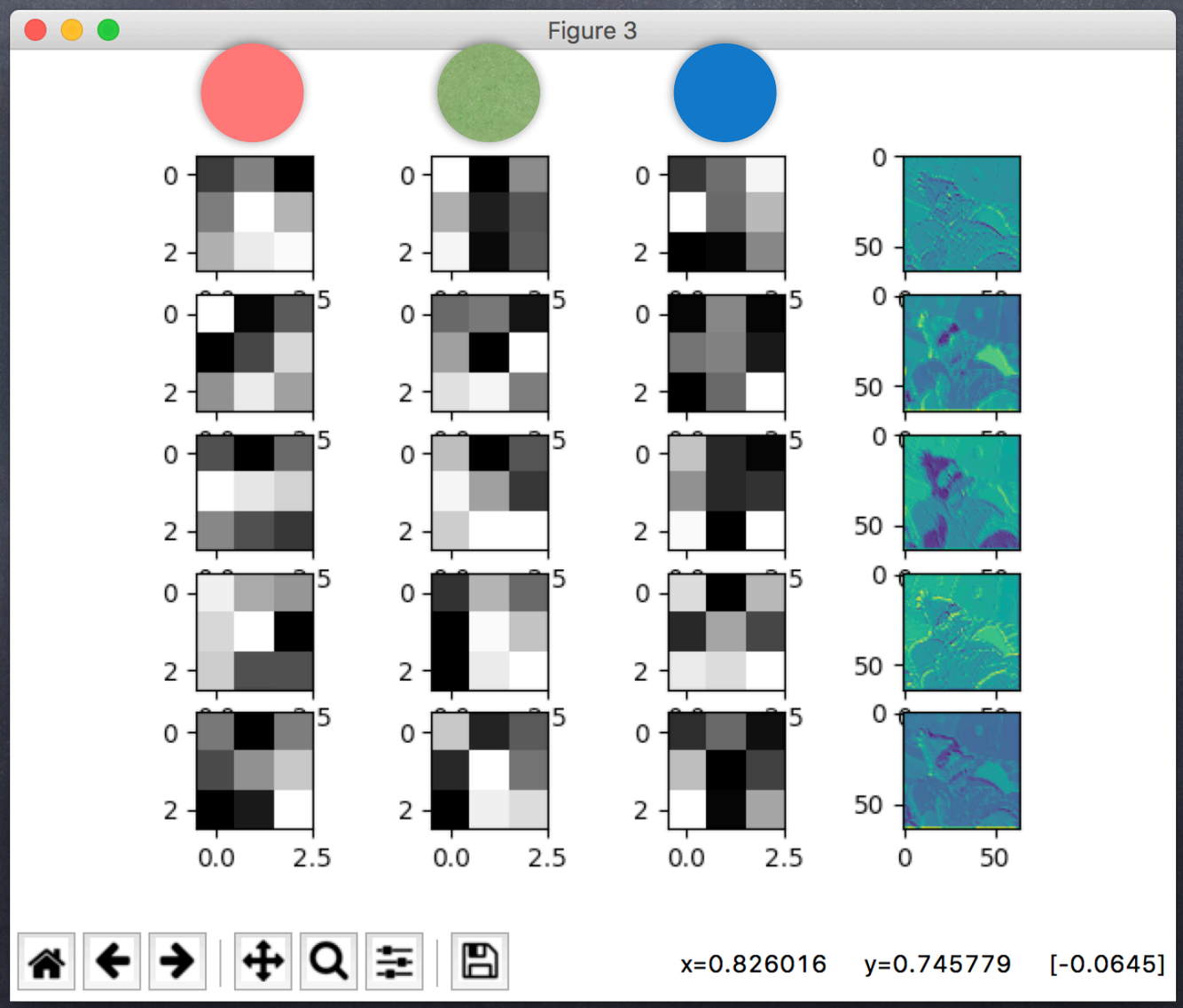
```
origin size:
(64, 64, 3)
after pooling size:
(30, 30, 64)
```

```
(3, 3, 64, 64)
origin size:
(64, 64, 3)
after pooling size:
(32, 32, 64)
```

padding = 'same'

Keras

- Here are some example for :
  Input -> Conv

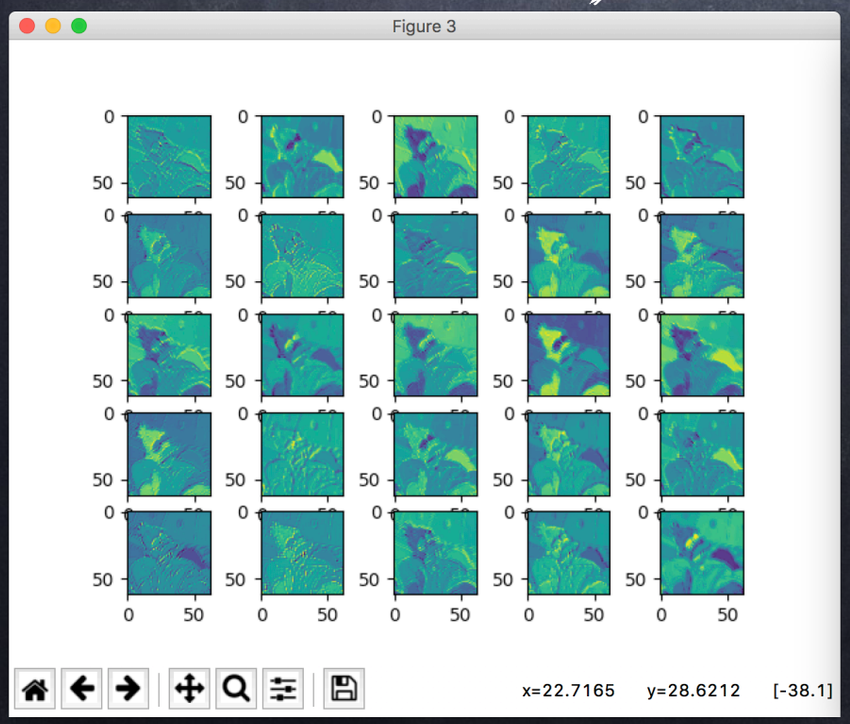


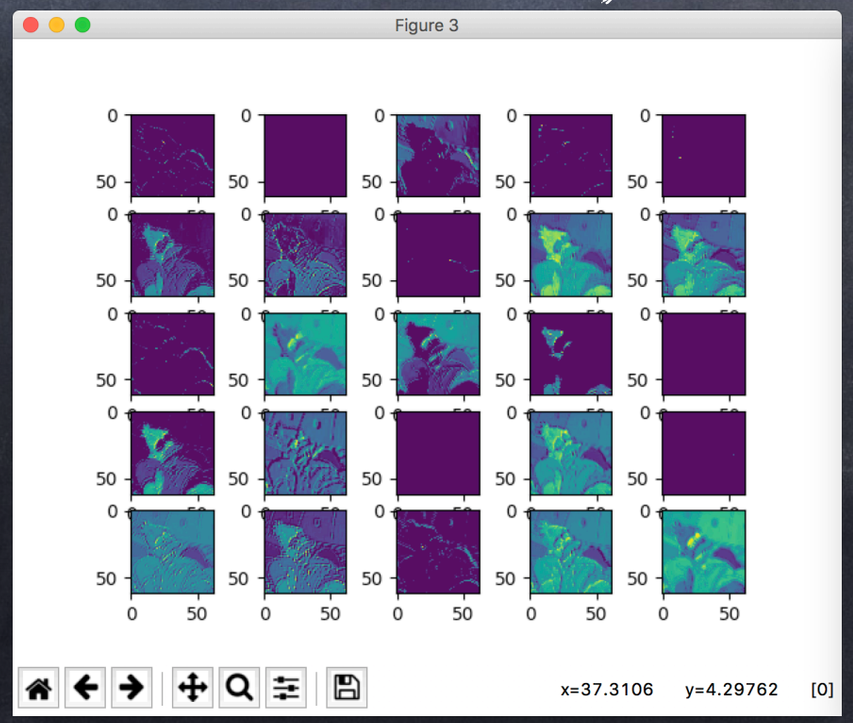
Keras

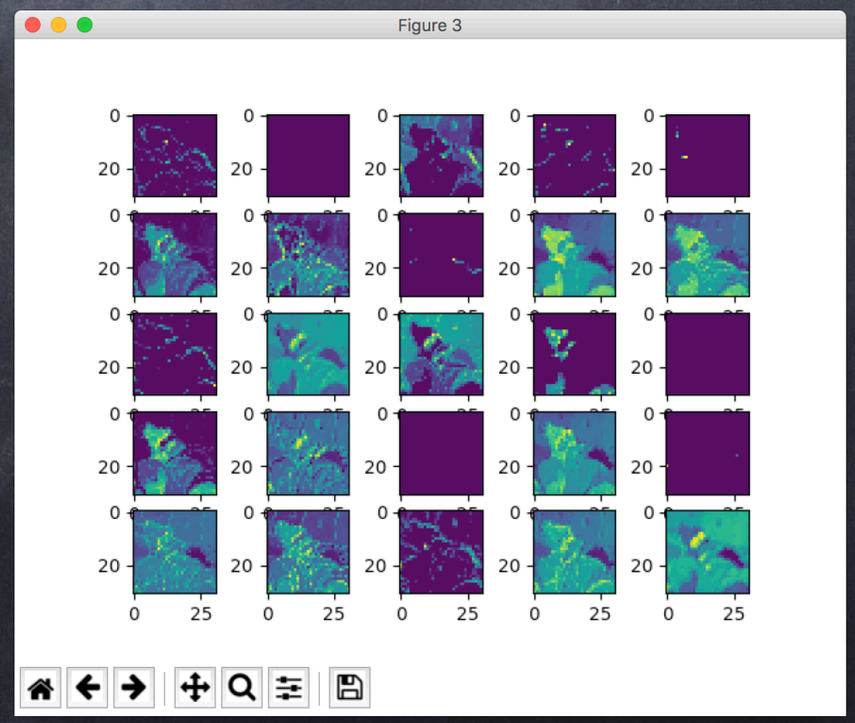
- Here are some example for :
  Conv -> Relu -> MaxPooling

Relu

MaxPooling





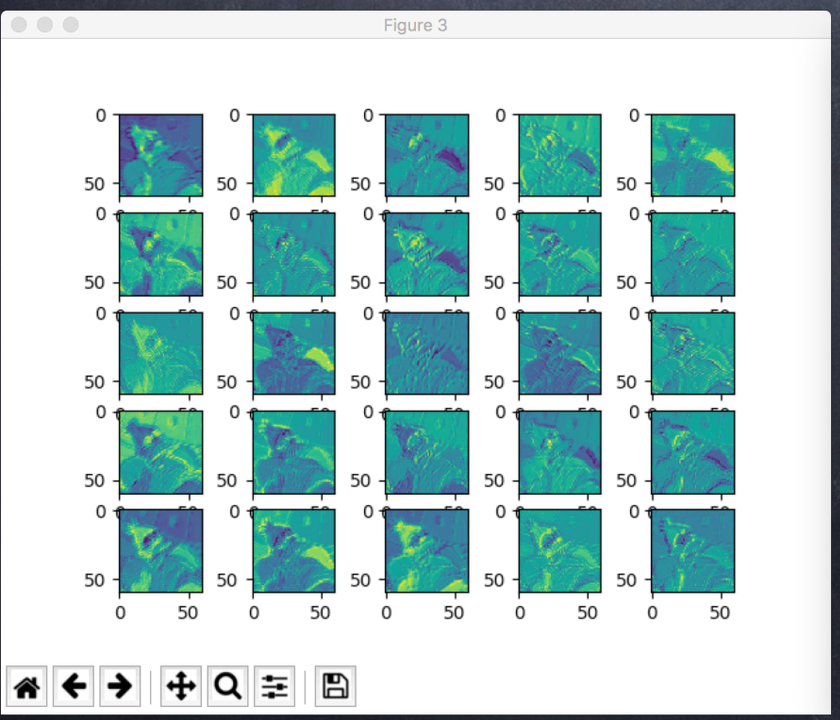


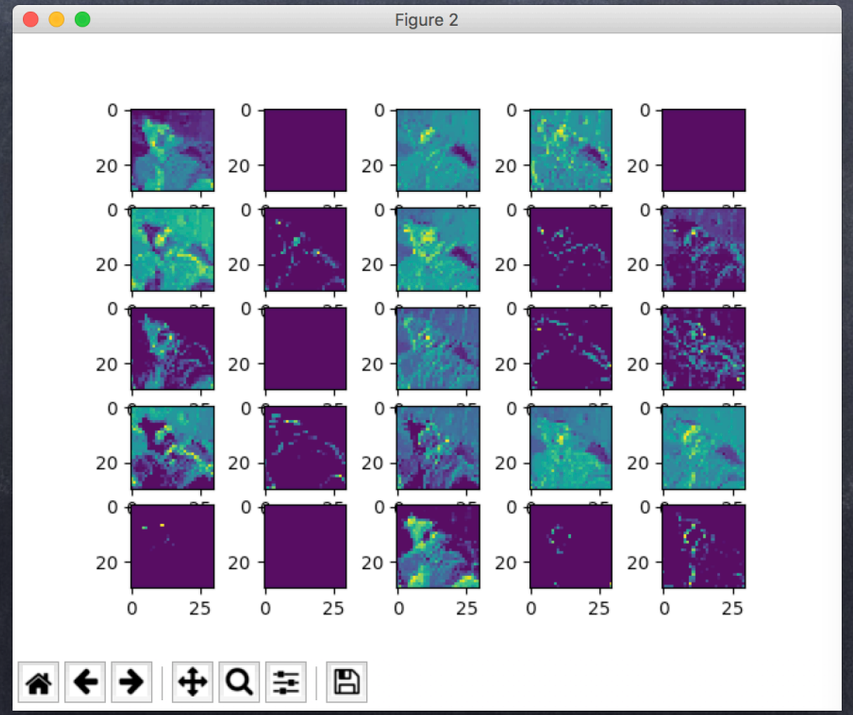
Keras

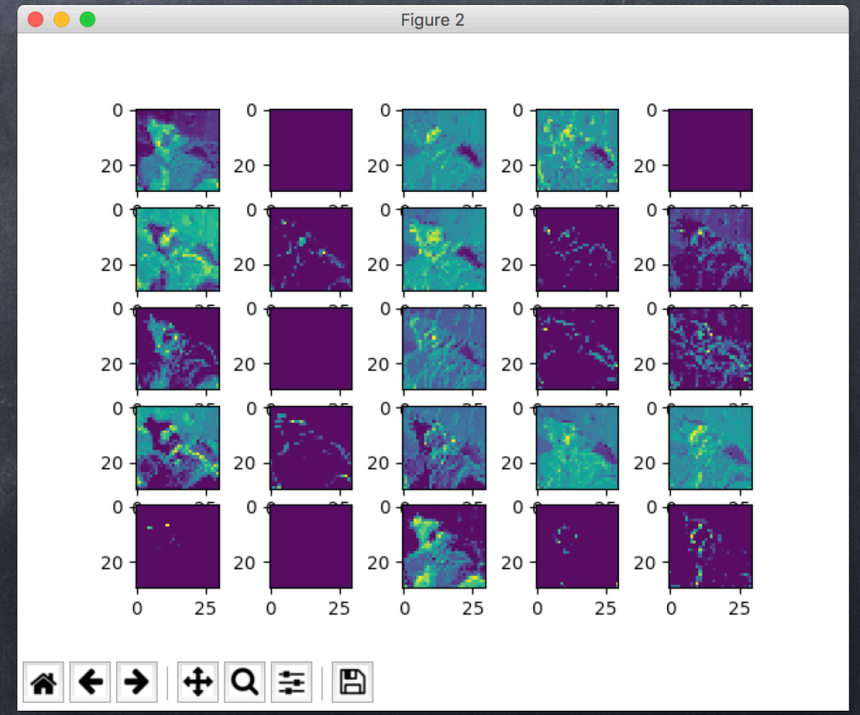
- Here are some example for :
  Conv -> Relu -> Conv -> Relu -> MaxPooling

Relu

MaxPooling







Keras

- Let's train and classify our dataset
  - Step 1. New a Convolution Neural Network Model

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same', input_shape=input_shape))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)
return model, opt
```

Keras

- Step 2. load train data and set 1/5 of the data to be the valid data

```
X_train_Ori, y_train_Ori = load_train_set("simpsons_dataset")
(X_train, X_test, y_train, y_test) = train_test_split(X_train_Ori, y_train_Ori,
    test_size=0.2)
```

Keras

- Step 3. Model

```
datagen = ImageDataGenerator(
    featurewise_center=False,  # set input mean to 0 over the dataset
    samplewise_center=False,  # set each sample mean to 0
    featurewise_std_normalization=False,  # divide inputs by std of the dataset
    samplewise_std_normalization=False,  # divide each input by its std
    zca_whitening=False,  # apply ZCA whitening
    rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
    width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
    horizontal_flip=True,  # randomly flip images
    vertical_flip=False)  # randomly flip images
datagen.fit(X_train)



filepath="my_simpsons.model4.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=0, save_best_only=True, mode='max')
callbacks_list = [checkpoint]
H = model.fit_generator(datagen.flow(X_train, y_train,
                                     batch_size=batch_size),
                                    steps_per_epoch=X_train.shape[0] // batch_size,
                                    epochs=30,
                                    callbacks=callbacks_list,
                                    validation_data=(X_test, y_test))
score = model.evaluate(X_test, y_test, verbose=0)
print('\nKeras CNN #2B - accuracy:', score[1])
return model , H
```

Keras

- In Step 3.
  - The ModelCheckpoint for choosing the better weights set between the file of weight and the current weight you trained
- Step 4. load file of weight and set how the model compiles

```
def load_model_from_checkpoint(weights_path, input_shape=(pic_size,pic_size,3)):
    model, opt = create_model_four_conv(input_shape)
    model.load_weights(weights_path)
    model.compile(loss='categorical_crossentropy',
                  optimizer=opt,
                  metrics=['accuracy'])

    return model
```

Keras

- Step 5. Train 1

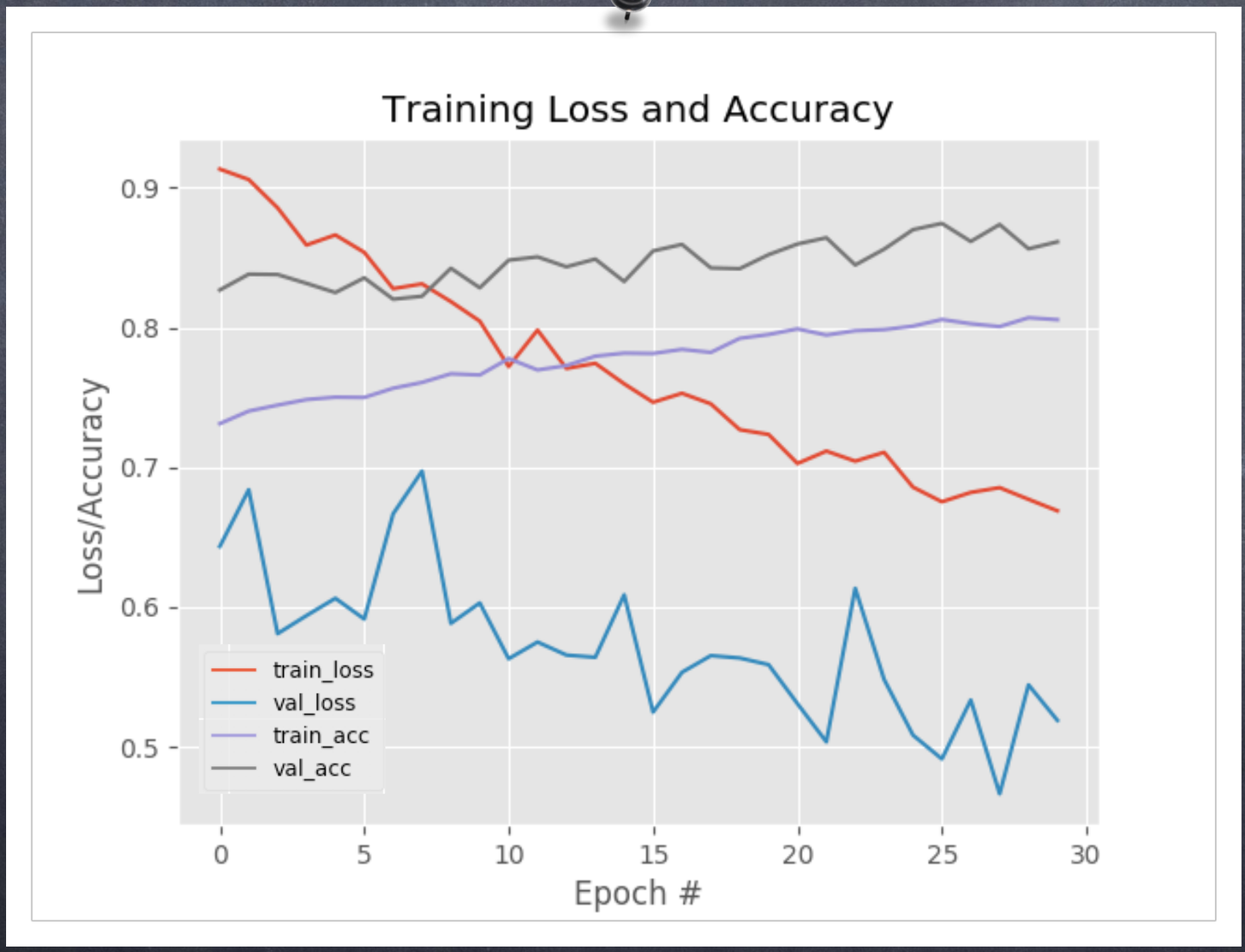
Keras

- Step 5. Score after Train 1

	precision	recall	f1-score	support
abraham_grampa_simpson	0.89	0.67	0.76	48
apu_nahasapeemapetilon	0.92	0.96	0.94	50
bart_simpson	0.54	0.80	0.65	50
charles_montgomery_burns	0.67	0.85	0.75	48
chief_wiggum	0.88	0.90	0.89	50
comic_book_guy	0.89	0.63	0.74	49
edna_krabappel	1.00	0.80	0.89	50
homer_simpson	0.62	0.84	0.71	50
kent_brockman	0.91	0.84	0.87	50
krusty_the_clown	0.93	1.00	0.96	50
lisa_simpson	0.87	0.54	0.67	50
marge_simpson	0.94	0.90	0.92	50
milhouse_van_houten	0.94	0.94	0.94	49
moe_szyslak	0.91	0.84	0.87	50
ned_flanders	0.77	0.88	0.82	49
nelson_muntz	0.92	0.66	0.77	50
principal_skinner	0.78	0.94	0.85	50
sideshow_bob	0.94	0.94	0.94	47
avg / total	0.85	0.83	0.83	890

Keras

- Step 5. Train 2



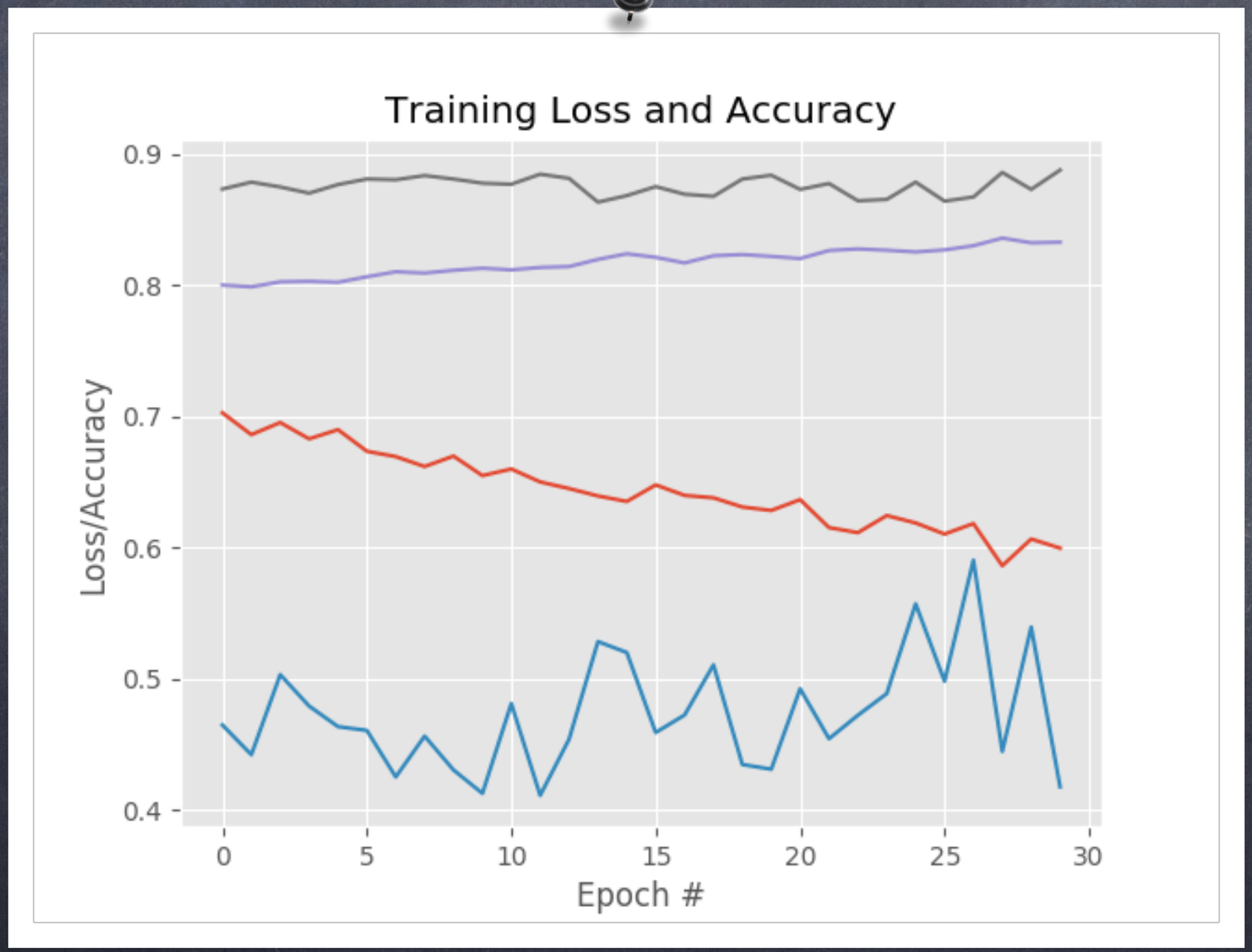
Keras

- Step 5. score after Train 2

	precision	recall	f1-score	support
abraham_grampa_simpson	0.88	0.73	0.80	48
apu_nahasapeemapetilon	0.94	0.96	0.95	50
bart_simpson	0.84	0.84	0.84	50
charles_montgomery_burns	0.66	0.85	0.75	48
chief_wiggum	0.82	0.92	0.87	50
comic_book_guy	0.97	0.63	0.77	49
edna_krabappel	0.98	0.86	0.91	50
homer_simpson	0.70	0.92	0.79	50
kent_brockman	0.92	0.90	0.91	50
krusty_the_clown	0.91	1.00	0.95	50
lisa_simpson	0.97	0.76	0.85	50
marge_simpson	0.98	0.98	0.98	50
milhouse_van_houten	0.92	0.98	0.95	49
moe_szyslak	0.88	0.90	0.89	50
ned_flanders	0.90	0.92	0.91	49
nelson_muntz	1.00	0.70	0.82	50
principal_skinner	0.91	0.98	0.94	50
sideshow_bob	0.87	1.00	0.93	47
avg / total	0.89	0.88	0.88	890

Keras

- Step 5. Train 3



Keras

- Step 5. score after Train 3

	precision	recall	f1-score	support
abraham_grampa_simpson	0.90	0.75	0.82	48
apu_nahasapeemapetilon	0.88	0.98	0.92	50
bart_simpson	0.88	0.74	0.80	50
charles_montgomery_burns	0.86	0.88	0.87	48
chief_wiggum	0.92	0.90	0.91	50
comic_book_guy	0.93	0.80	0.86	49
edna_krabappel	0.94	0.92	0.93	50
homer_simpson	0.74	0.96	0.83	50
kent_brockman	0.98	0.92	0.95	50
krusty_the_clown	0.96	1.00	0.98	50
lisa_simpson	0.82	0.84	0.83	50
marge_simpson	0.94	0.96	0.95	50
milhouse_van_houten	0.87	0.98	0.92	49
moe_szyslak	0.92	0.88	0.90	50
ned_flanders	0.94	0.98	0.96	49
nelson_muntz	1.00	0.84	0.91	50
principal_skinner	0.94	0.98	0.96	50
sideshow_bob	0.94	0.98	0.96	47
avg / total	0.91	0.90	0.90	890

Keras

- Step 6. show

Reference

- https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
- http://solring-blog.logdown.com/posts/302641-dl-convolutional-neural-network
- http://mropengate.blogspot.tw/2017/02/deep-learning-role-of-activation.html
- https://en.wikipedia.org/wiki/Backpropagation
- http://cs231n.github.io/convolutional-networks/
- http://cs231n.github.io/understanding-cnn/
- https://stackoverflow.com/questions/43305891/how-to-correctly-get-layer-weights-from-conv2d-in-keras?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa
- https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
- https://medium.com/alex-attia-blog/the-simpsons-character-recognition-using-keras-d8e1796eae36
- https://www.coursera.org/learn/ntumlone-algorithmicfoundations/home/welcome
- https://medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c
- Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.