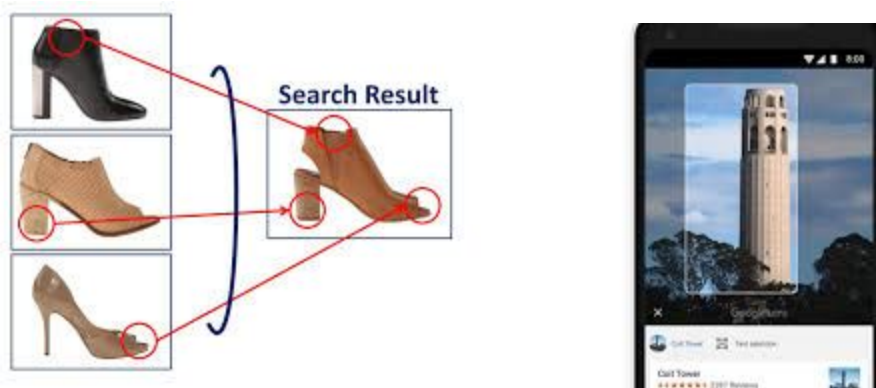


# Visual Recognition

Understanding what happens behind the scenes in Convnets.

---



## Introduction

Convolutional Neural Networks or Convnets for short are a form of Artificial Neural networks, their behaviour that differentiates them from regular Dense Neural networks are

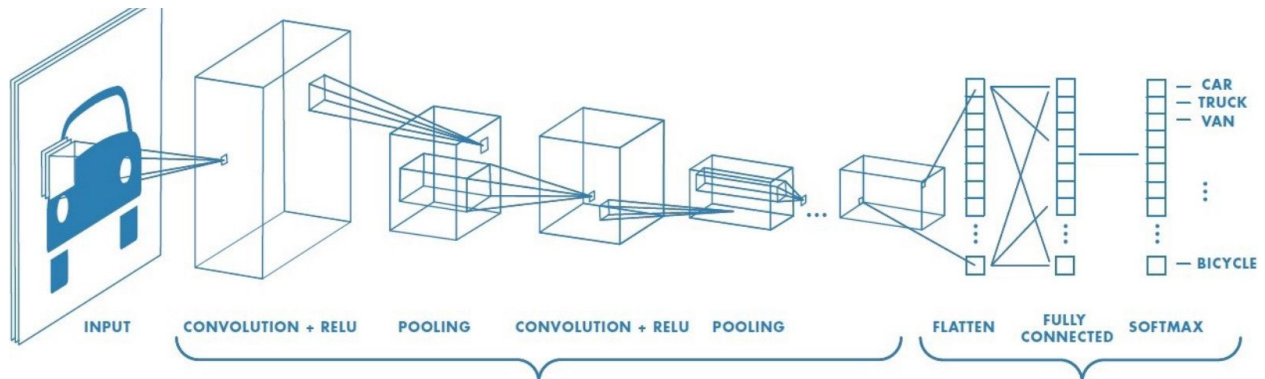
1. They can identify and recognize local patterns to build high level features
2. They can identify patterns invariantly in space.

So, in this report we are going to present what Convnets are? How do they complement then DNNs? How does a convnet process an image?

---

---

## What are Convnets?



Convnets are a class of Deep neural networks which find their successful use in image/pattern recognition. As you can infer from the image above that CNN is technically a preprocessing step before the processed image is fed to the DNNs. CNN layers consist of input layer , output layer and multiple hidden layers which typically consist of activation function, pooling layers and fully connected layers.

The major differences between a DNN algorithm and a CNN is

1. DNN learn from global patterns in their input feature space while CNN's learn local patterns.
2. The patterns that CNNs learn are translation invariant unlike DNN
3. CNN can learn Spatial hierarchies unlike DNNs.

## Dataset

Here we are using dataset from kaggle which consist of

```
total training cat images: 3001
total training dog images: 3006
total test cat images: 1012
total test dog images: 1013
total validation cat images: 1000
total validation dog images: 1000
```

---

The data has been separated into training, validation and test data sets

## Model building without regularization

```
[23] from keras import models
      from keras import layers

↳ Using TensorFlow backend.

[24] model = models.Sequential()
      model.add(layers.Conv2D(32, (3,3), activation="relu",
                             input_shape=(150,150,3))) # you'll see later that we will resize all image to 150x150
      model.add(layers.MaxPooling2D((2,2)))

      model.add(layers.Conv2D(64, (3,3), activation='relu'))
      model.add(layers.MaxPooling2D((2,2)))

      model.add(layers.Conv2D(128, (3,3), activation='relu'))
      model.add(layers.MaxPooling2D((2,2)))

      model.add(layers.Conv2D(128, (3,3), activation='relu'))
      model.add(layers.MaxPooling2D((2,2)))

      model.add(layers.Flatten())
      model.add(layers.Dense(512, activation="relu"))
      model.add(layers.Dense(1, activation="sigmoid"))

↳ WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/op_def_library.py:263:
Instructions for updating:
Colocations handled automatically by placer.
```

So, from the figure is a code snippet of how we build the model, this pretty straight forward way of building a model.

```
[25] model.summary()
```



Layer (type)	Output Shape	Param #
=====		
conv2d_1 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_1 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_2 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_3 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_3 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_4 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_4 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 512)	3211776
dense_2 (Dense)	(None, 1)	513
=====		
Total params: 3,453,121		
Trainable params: 3,453,121		
Non-trainable params: 0		

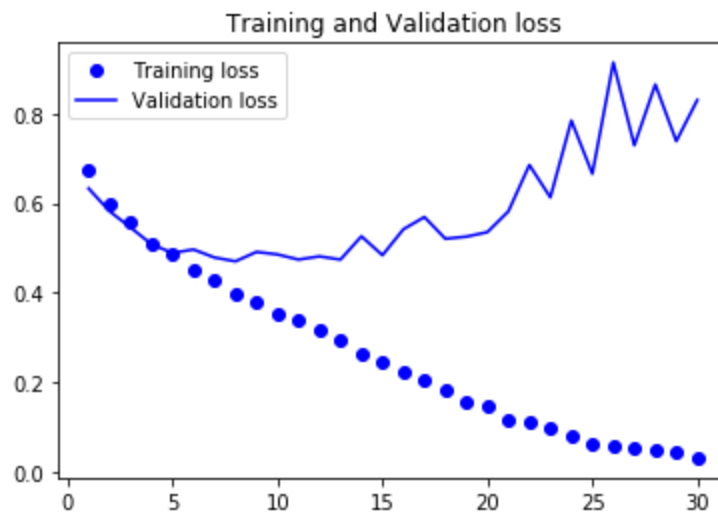
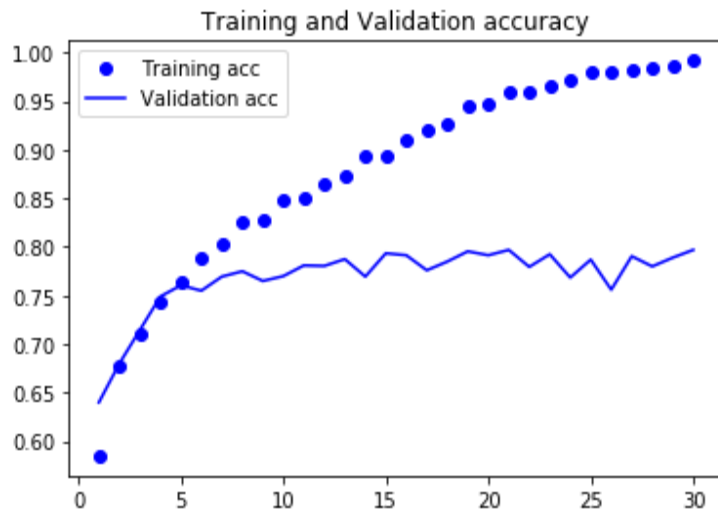
This figure above is a output snippet of model that we built:

- We can see that how different layers are arranged and how the size of the shape is changing after every layer and each layer has different number of outputs.

---

## Overfitting:

As you can see that the training accuracy goes on increasing while the validation accuracy stalls at around 75%, which is just ok. but if you look at the loss plot, you see that the training loss goes on decreasing but the validation loss hits its minimum at around 14 epochs but after that loss goes on increasing which indicates overfitting of the model

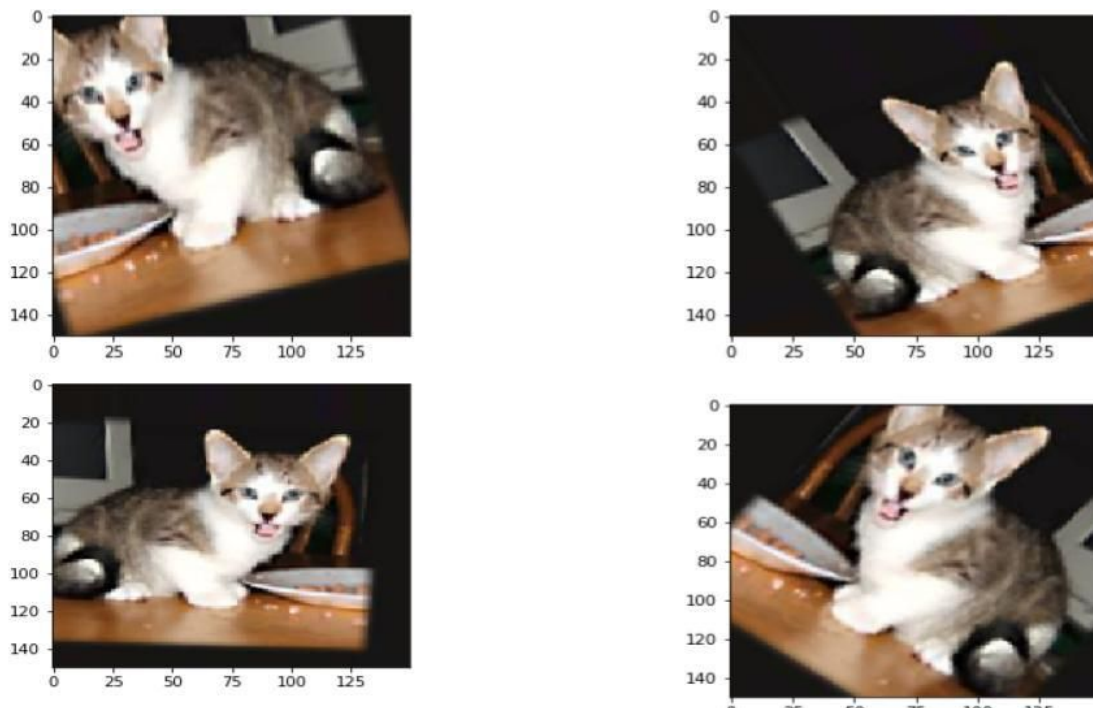


---

## Avoiding Overfitting:

### Data Augmentation:

It is the process of generating more training data from existing training samples. this process uses a number of transformations on the existing images to yield new samples for training. The goal of augmentation is first increase the number of samples such that the model doesn't see the image twice while training. In Keras augmentation is done by using a combination of transformation( to be applied on the images) via the ImageDataGenerator instance.[**Source code in python NB**]



Here you can see that by using the Data Augmentation provided by keras we can cope up with less training data.

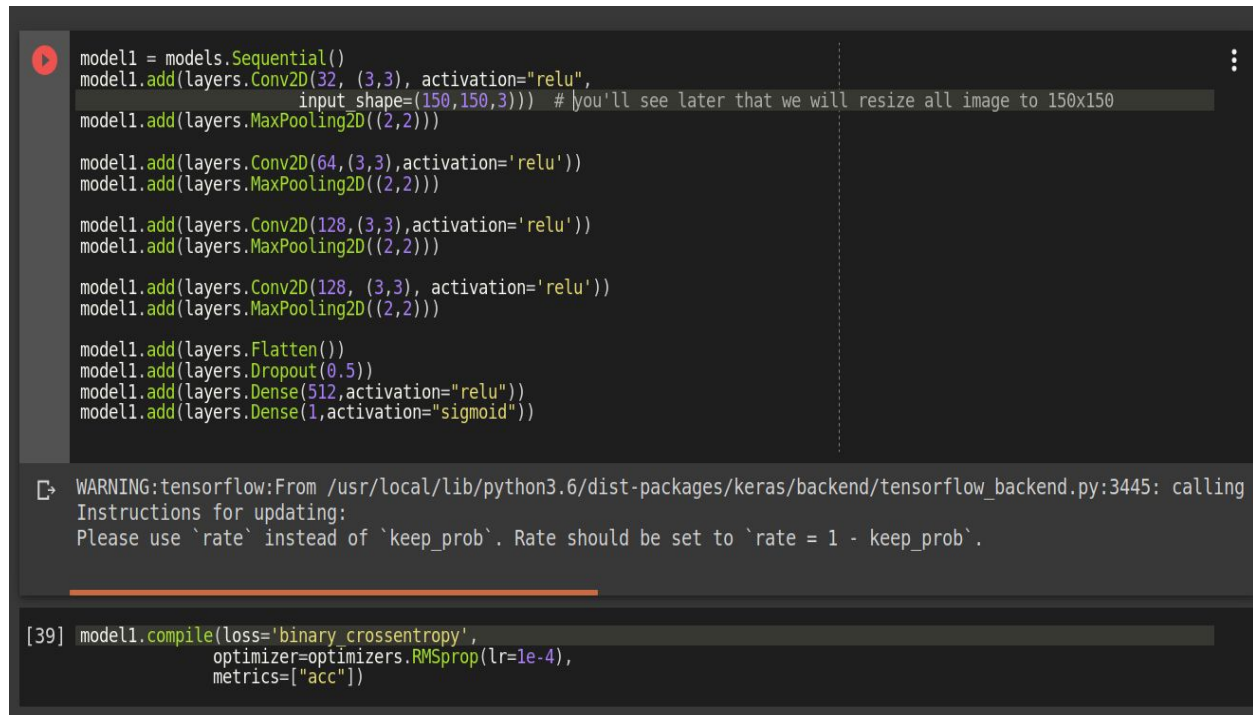
### But does Data Augmentation really helped?

So, even though your model would never see the same image twice but your augmented images are just slight transformations from the real image so, they still have high

---

correlation which might be a factor for overfitting , that's one of the reason we use Dropout to avoid this.

## Building model with regularization



```
model1 = models.Sequential()
model1.add(layers.Conv2D(32, (3,3), activation="relu",
                        input_shape=(150,150,3))) # you'll see later that we will resize all image to 150x150
model1.add(layers.MaxPooling2D((2,2)))

model1.add(layers.Conv2D(64, (3,3), activation='relu'))
model1.add(layers.MaxPooling2D((2,2)))

model1.add(layers.Conv2D(128, (3,3), activation='relu'))
model1.add(layers.MaxPooling2D((2,2)))

model1.add(layers.Conv2D(128, (3,3), activation='relu'))
model1.add(layers.MaxPooling2D((2,2)))

model1.add(layers.Flatten())
model1.add(layers.Dropout(0.5))
model1.add(layers.Dense(512, activation="relu"))
model1.add(layers.Dense(1, activation="sigmoid"))

[39] model1.compile(loss='binary_crossentropy',
                  optimizer=optimizers.RMSprop(lr=1e-4),
                  metrics=["acc"])
```

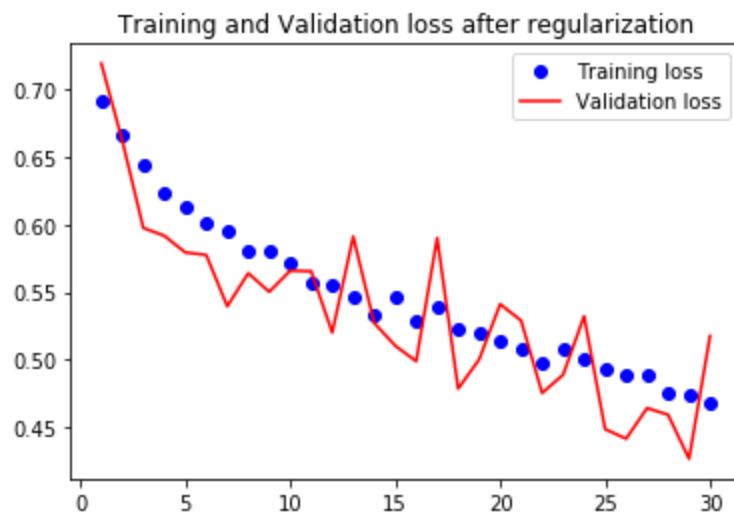
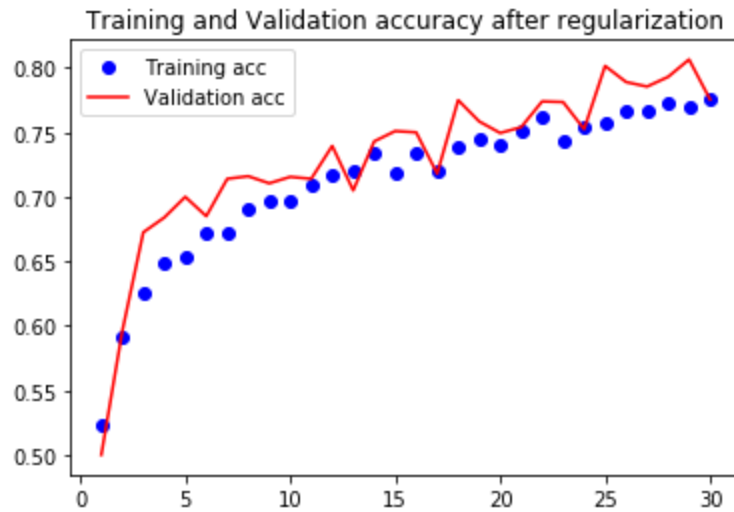
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow\_backend.py:3445: calling Instructions for updating:  
Please use `rate` instead of `keep\_prob`. Rate should be set to `rate = 1 - keep\_prob`.

Notice the use of dropout and introduction of learning rate when using the RMSprop optimizer.

---

## Checking for performance of the regularized model

Here you can see that after applying data augmentation and dropout the model is performing well to the validation set this avoiding over fitting



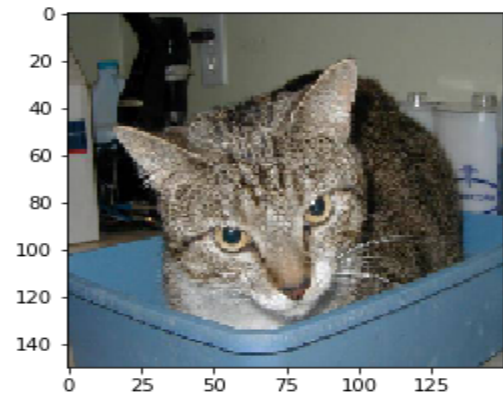


---

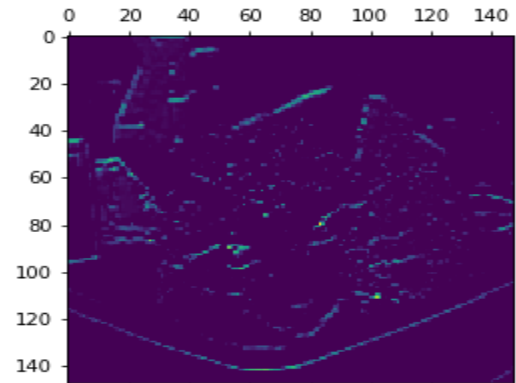
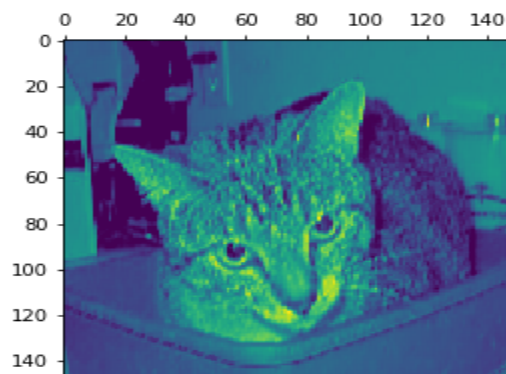
## Visualizing what convnets learn

So, next what we tried was to see the outputs of each layer to find how the input is decomposed into different filters learned by the network.

Say this on the right is the input image

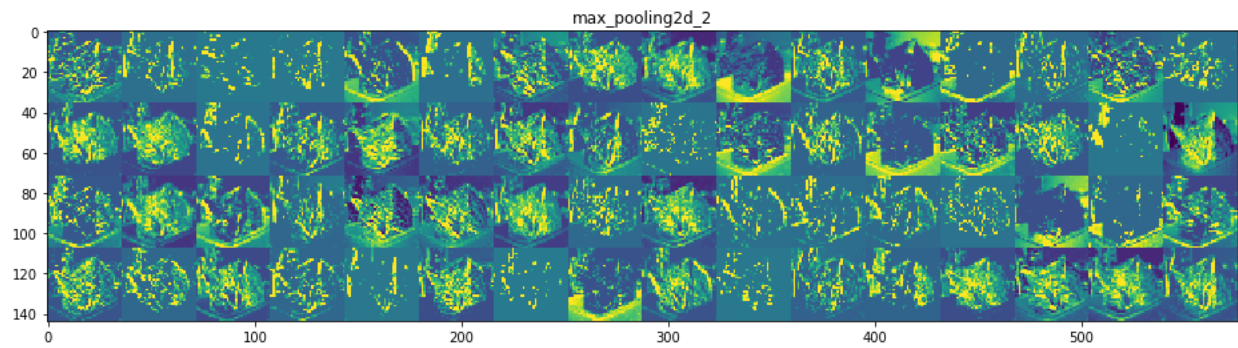
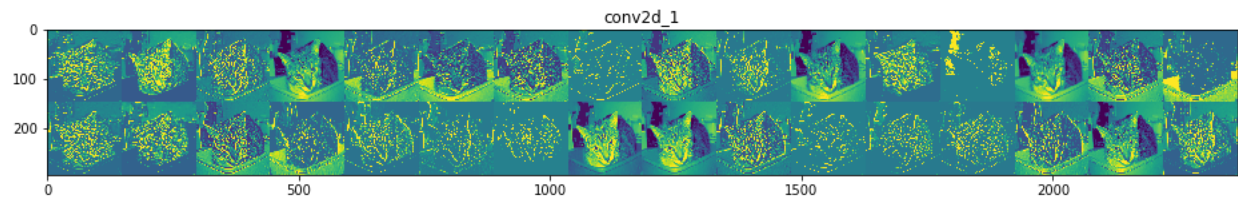


These two are the outputs of random filters from which we can infer that each filter looks for some specific features which evolve in each layer.

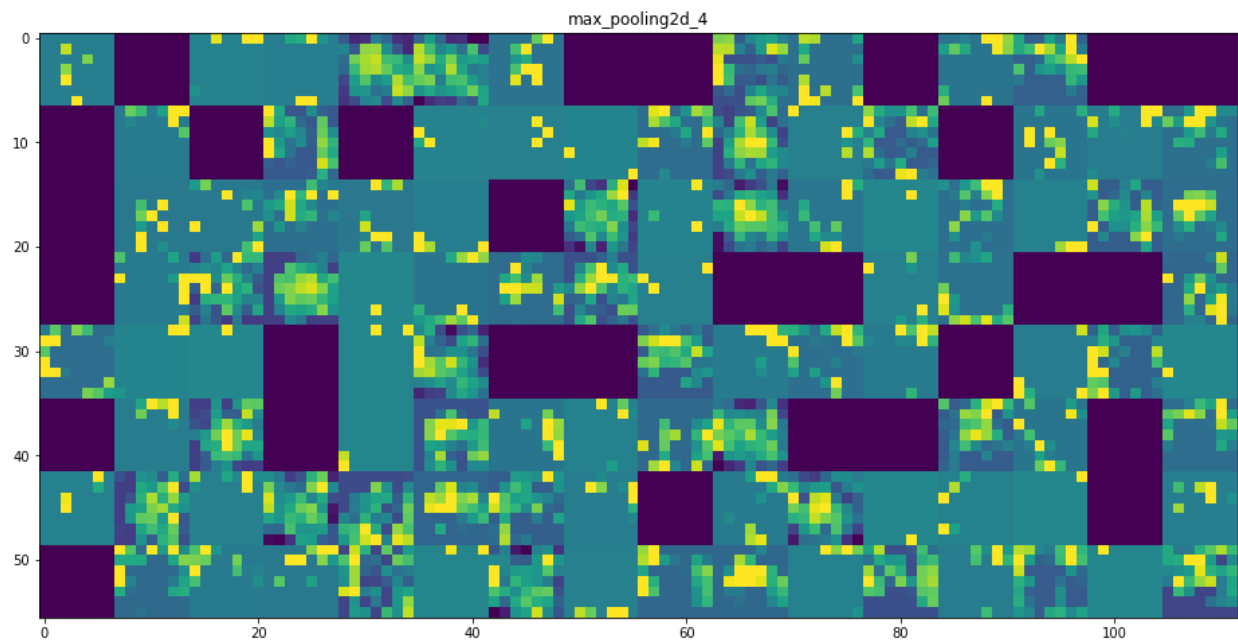


---

To get a better understanding we produced the outputs of a few layers .



So, see that as we go deeper into the layers the features become more and more abstract unnoticeable to the human.



---

References:

<https://keras.io/>

<https://matplotlib.org/>

<https://www.python.org/>

[www.kaggle.com](http://www.kaggle.com)

DeepLearning with python ; Francois Chollet