Article By Jayant Arsode

ARTIFICIAL NEURAL NETWORKS

- Artificial Neural Networks (ANNs) are computational processing systems of which are heavily inspired by way biological nervous systems (such as the human brain) operate.

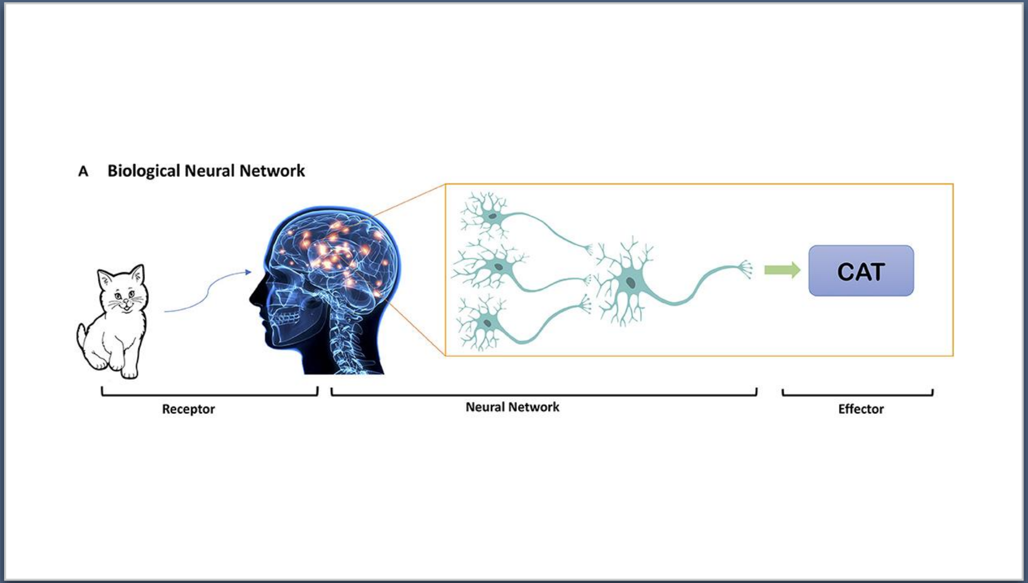
- They have small nodes called as neurons.

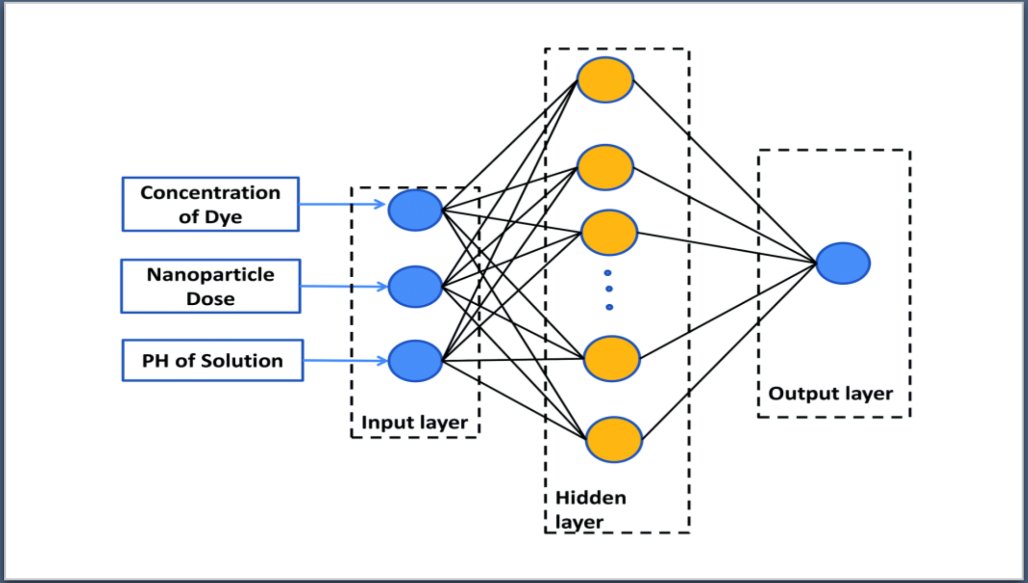
- There are three layers in ANN Input layer, hidden layer, Output layer.

- They take features as input and give labels or continues numbers as output.

- They can also classify images but like MNIST dataset having small image of 28 x28 dimension having in which first layer have 784 weights while for RGB images 64x64x3 that is 12,288.

- Also due to so much computational data our cost will increase so we need to find an optimized way to extract image data.





Article By Jayant Arsode

CONVOLUTION NEURAL NETWORKS

- Convolution Neural Networks (CNNs) are Supervised learning types of Deep learning algorithm that deals **multiple features and classifier in joint** fashion.
- Perform much better than ANN’s in image classification as it extract features thus reducing parameters.
- Image classification and segmentation problems also can be used for NLP.

Courtesy: internet

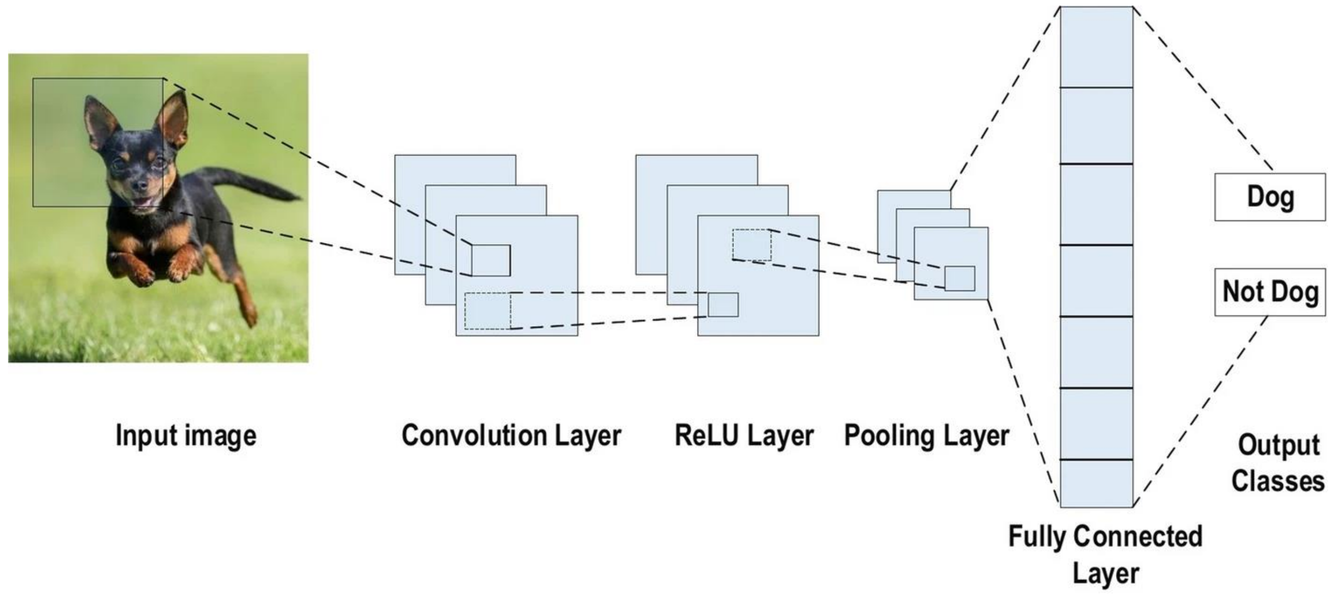
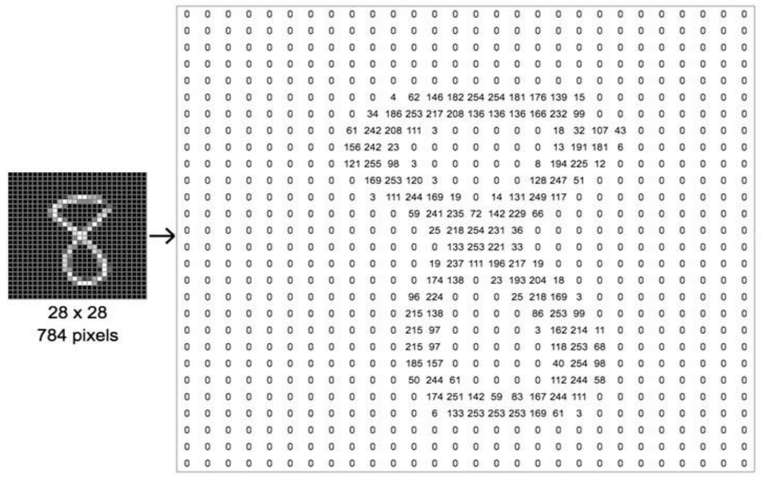


Fig. 1: Architecture of Convolution Neural Networks (CNNs)

INPUT

- An Image is a matrix of pixel values
- If we consider image then colors are represented by 8-bit binary number with pixel value ranging from 1 to 255.
- If we consider RGB image each pixel will have a combined values of R,G and B.
- Features are called weights.
- Input format

  Height x Width x Depth



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 1.1 Input For A Gray Scale Image

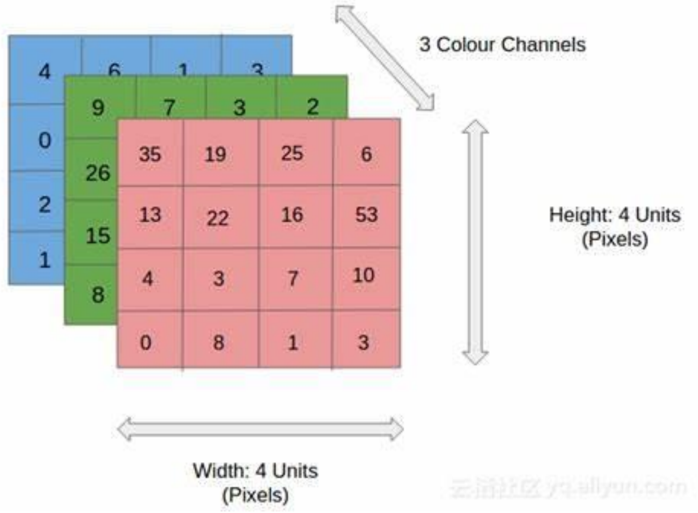


Fig. 1.2 Input For A Colored Image With RGB Channels

CONVOLUTIONAL LAYER

- In CNN architecture, the most significant component is the convolutional layer. It consists of a collection of convolutional filters (so-called kernels). The input image, expressed as N-dimensional metrics, is convolved with these filters to generate the output feature map.
- **Kernel/Filter/Mask:**
  - Multiple filters are automatically learned CNN during training on a particular training dataset.
  - At the end of the training, you would have a unique set of filter values that are used for detecting specific features in the dataset.
- **Stride**
  - Denotes how many steps we are moving in each steps in convolution.
  - By default, it is one.

courtesy: codebasics

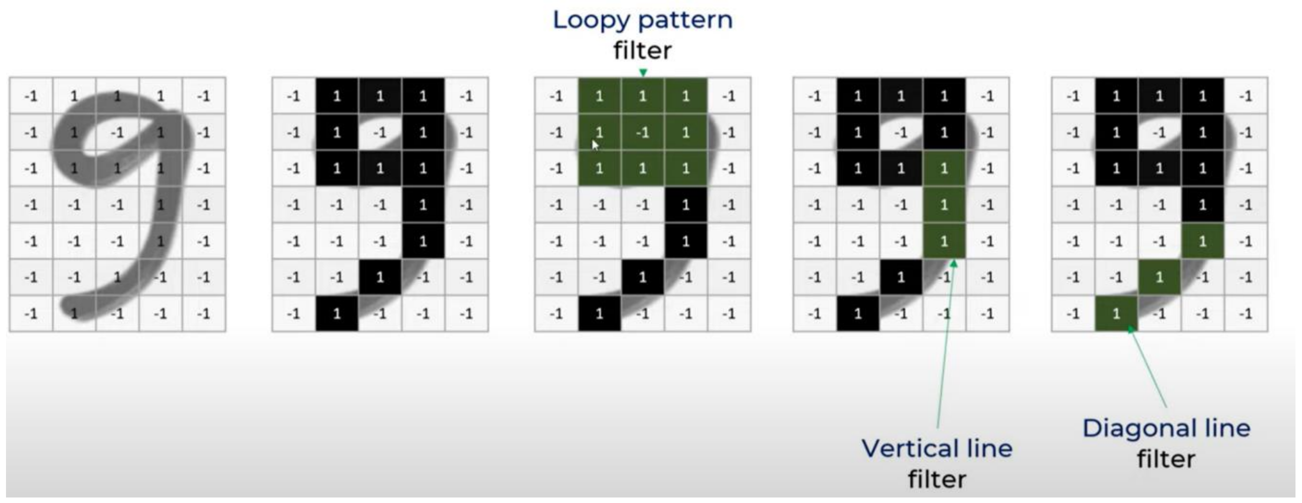


Fig. 1.2: Extracting patterns in images

courtesy: codebasics

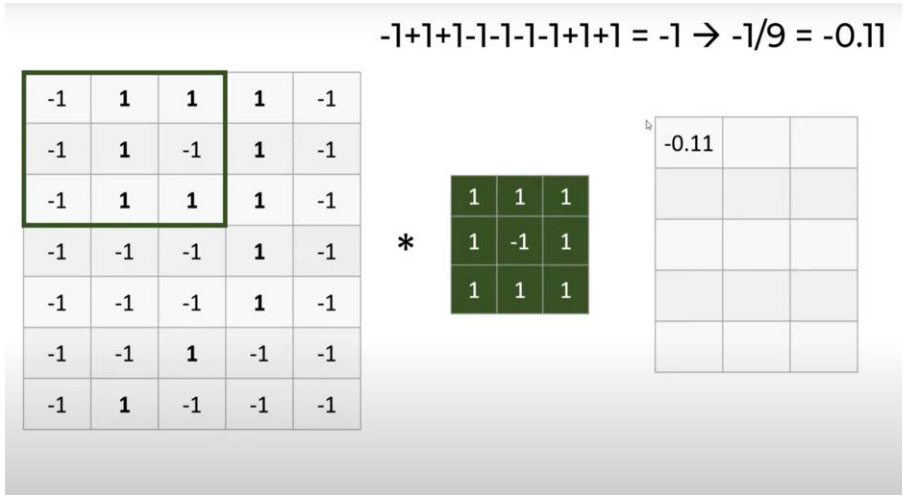
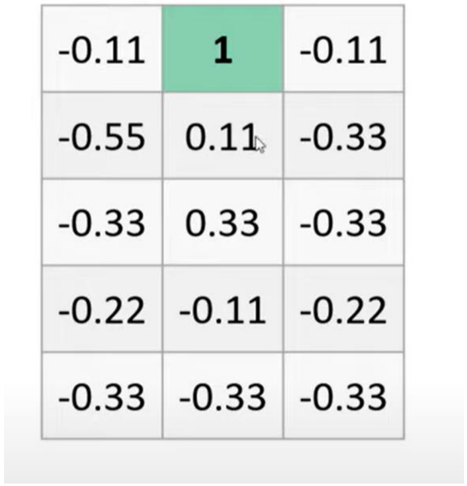


Fig. 1.2.1: Process of Applying convolution filter.

courtesy: codebasics



-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33

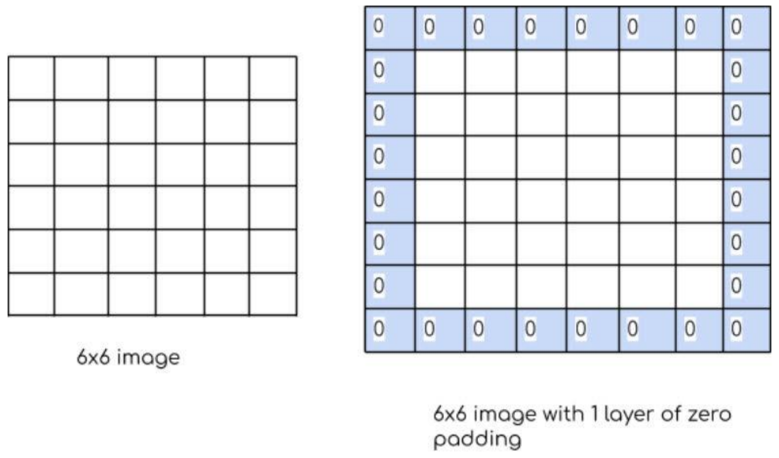
Fig. 1.2.2: Final Feature MAP

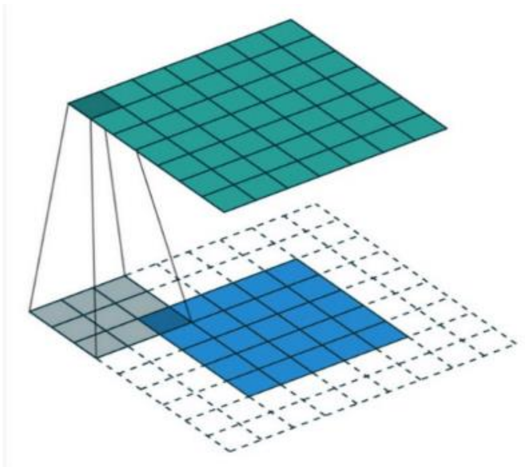
PADDING

- Sometimes the pixel of images at edges do not participate in convolution operation for even participation of image we can use padding.
- Padding is simply a process of adding layers of zeros to our input images so as to avoid the problems mentioned above.
- Type of padding:

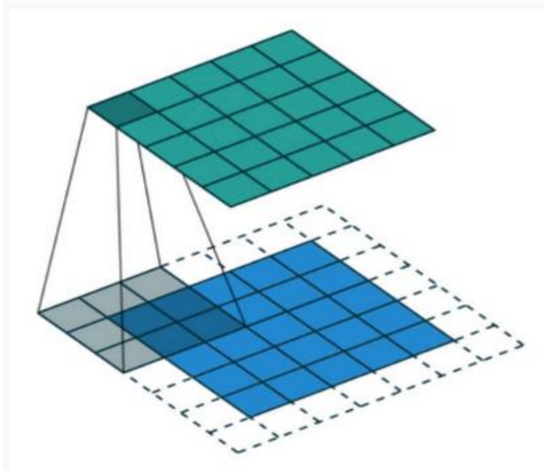
1) Full Padding.
2) Same Padding.

Figure showing simple padding=1





Full Padding-It add zeros in such a way that all pixel are visited same amount of time.

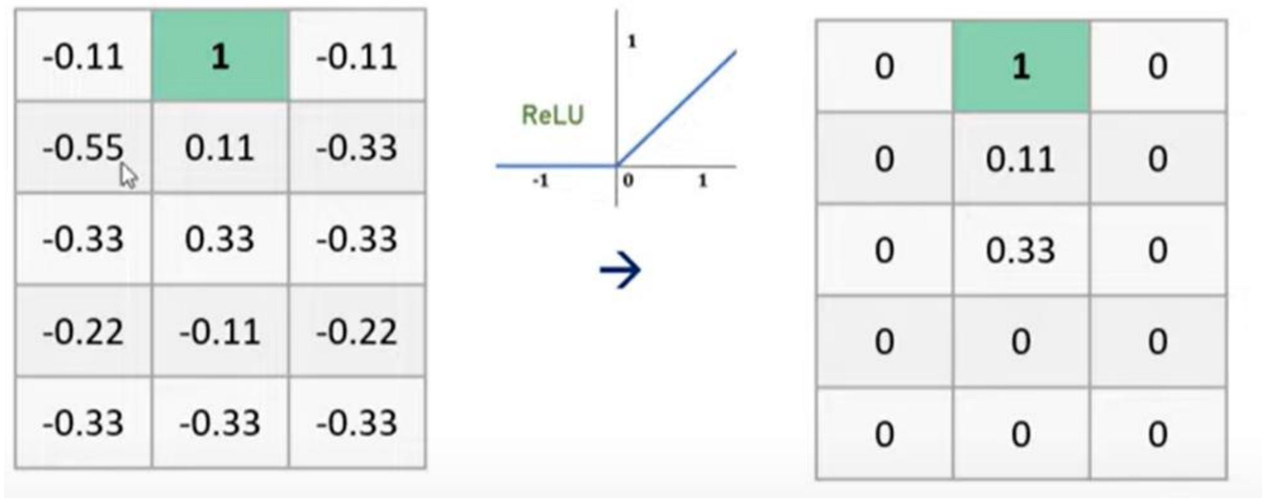


Same Padding-It add zeros in such a way that input matrix and output are equal.

CONVOLUTIONAL LAYER WITH RELU(RECTIFIED LINEAR UNIT)

- RELU is kind of optimization function used to optimize the feature map here and reduces the computational cost by removing unwanted pixels sharpening image.
- In this optimization it convert all negative values in feature map to zero.
- There are also sigmoid optimization function but relu works best with CNNs.

courtesy: codebasics



-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33

0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

Fig. 1.3: Optimization of feature map using RELU.

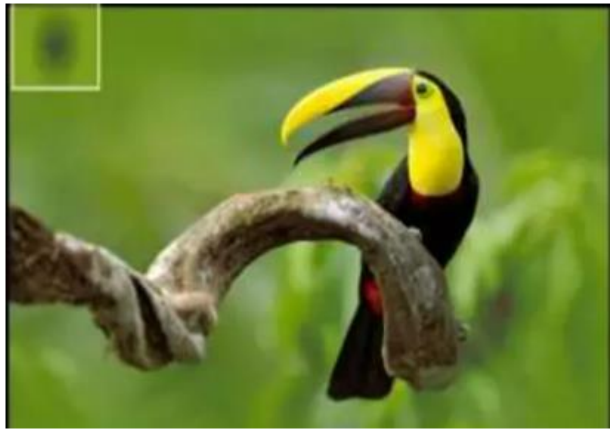


Fig. 1.3.1: Raw Image.

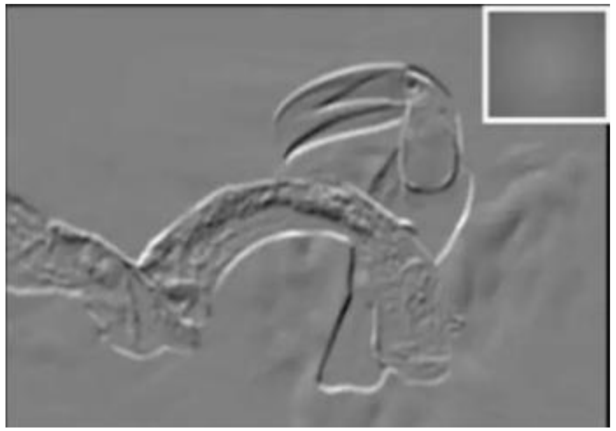
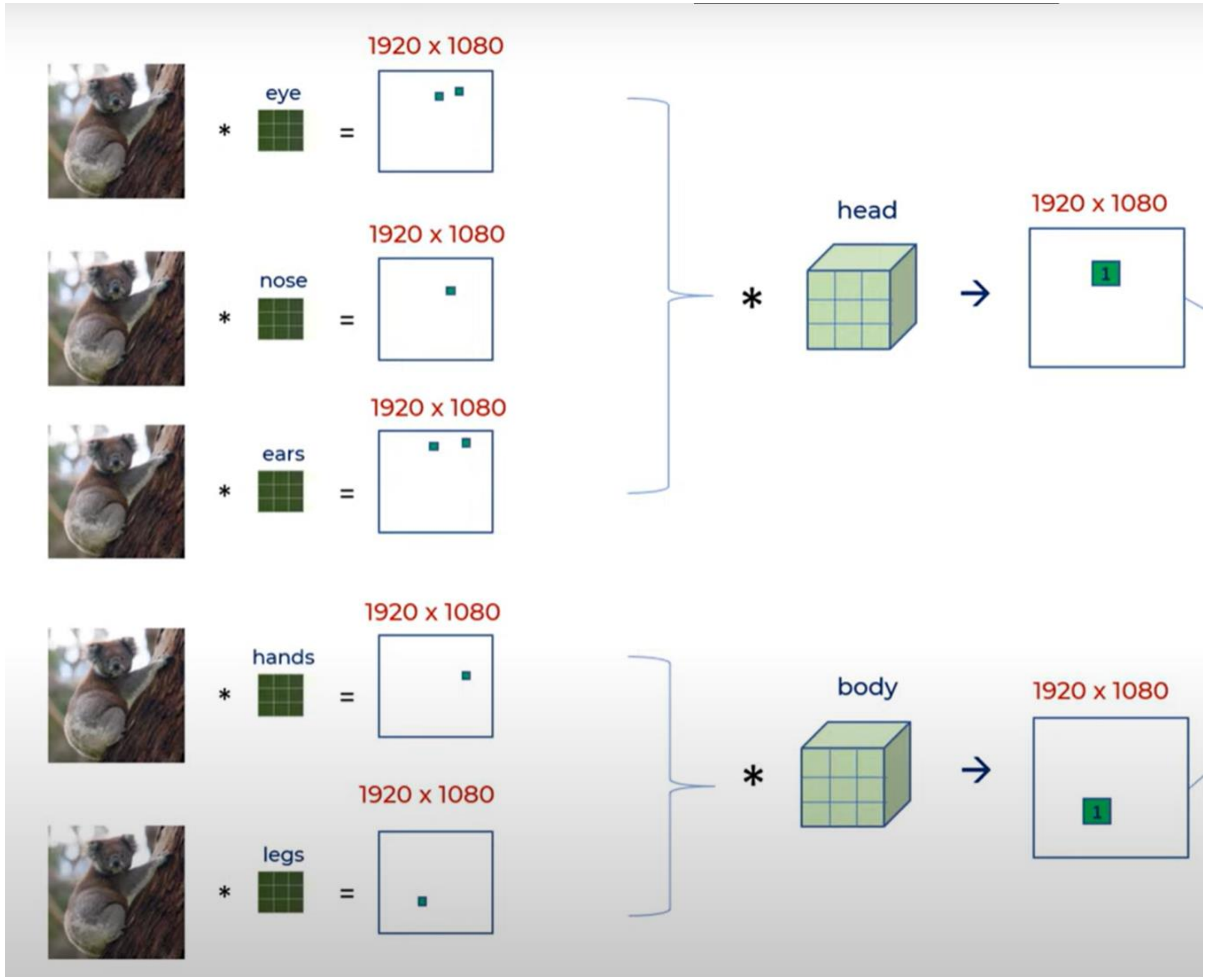


Fig. 1.3.2: Image after using Convolution+RELU.

CONVOLUTIONAL LAYER ON A RGB IMAGE WITH SIZE 1920 X 1080

- We **convolve** different feature map to get a complex map.
- As given feature map have **1920x1080** size so **cost of computation is more**.
- Also even after convolution operation our feature map is big thus to extract important features we use **pooling layer**.

courtesy: codebasics



POOLING LAYER

Pooling layer is used to reduce size of feature map and it's sub sampling

STRIDE – It is number of pixel we move forward after each computation.

MAX Pooling - It choose feature with maximum value from feature map.

Average Polling – It do average on all values of feature map.

Global Average Pooling – It averages all values of feature map

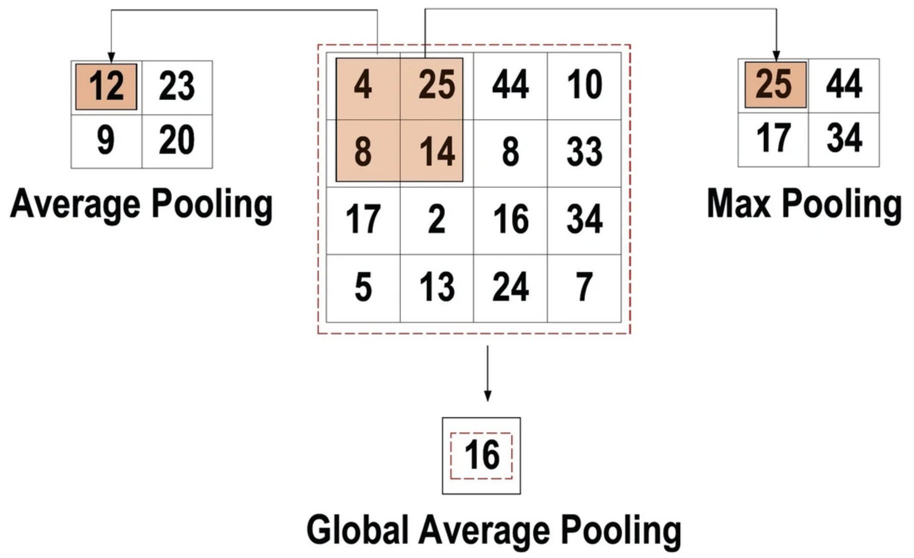


Fig. 1.4: Different pooling with stride = 2 on feature map

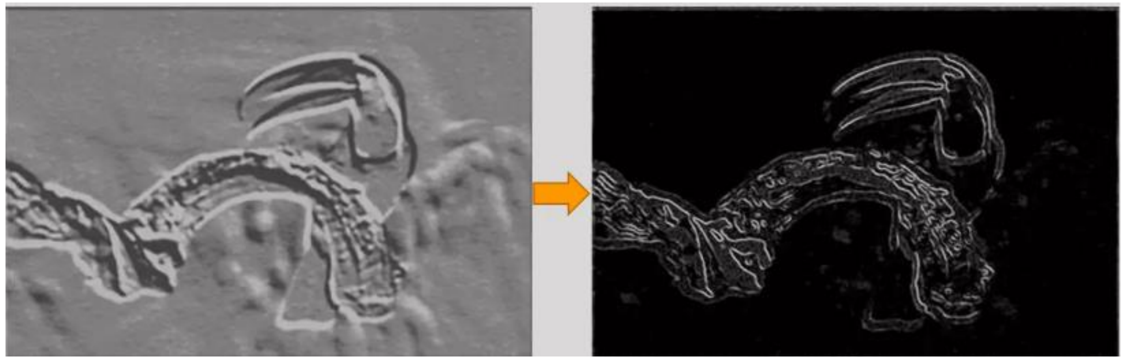


Fig. 1.4.1: Convolution To Pooled Image

FLATTENING AND FULLY CONNECTED LAYER

- **FLATTEN LAYER** – It used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector.
- **FULLY CONNECTED LAYER** - The fully-connected layer is simply an artificial neural network where parameter training take place.

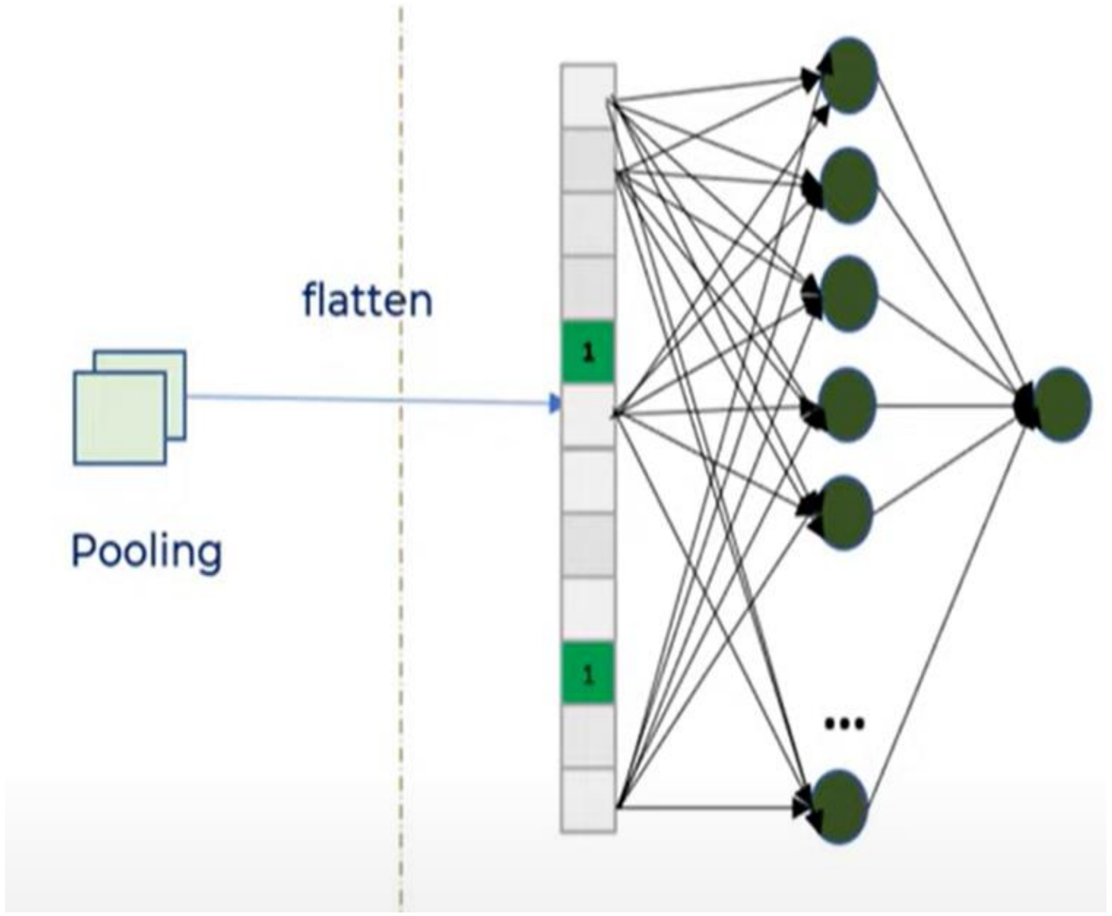


Fig. 1.5: Flattening and fully connected layer in CNN

Article by Jayant Arsode

cnn-cifar100

April 14, 2023

1 Image recognition using CNN on CIFAR-100 Dataset

[31]:
```
from keras.datasets import cifar100
from keras.layers import Dense,Conv2D,MaxPooling2D,Flatten,Dropout
from keras.callbacks import EarlyStopping
from keras.utils import load_img,to_categorical
from keras.models import Sequential
import matplotlib.pyplot as plt
from pathlib import Path
import numpy as np
import seaborn as sns

(X_train,y_train),(X_test,y_test)=cifar100.load_data()
```

2 The CIFAR-100 dataset

This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a “fine” label (the class to which it belongs) and a “coarse” label (the superclass to which it belongs). Here is the list of classes in the CIFAR-100:

Superclass ————————-> Classes

1) aquatic mammals ————————-> beaver, dolphin, otter, seal, whale
2) fish ————————-> aquarium fish, flatfish, ray, shark, trout
3) flowers ————————-> orchids, poppies, roses, sunflowers, tulips
4) food containers ————————-> bottles, bowls, cans, cups, plates
5) fruit and vegetables ————————-> apples, mushrooms, oranges, pears, sweet peppers
6) household electrical devices ————————-> clock, computer keyboard, lamp, telephone, television
7) household furniture ————————-> bed, chair, couch, table, wardrobe
8) insects ————————-> bee, beetle, butterfly, caterpillar, cockroach
9) large carnivores ————————-> bear, leopard, lion, tiger, wolf

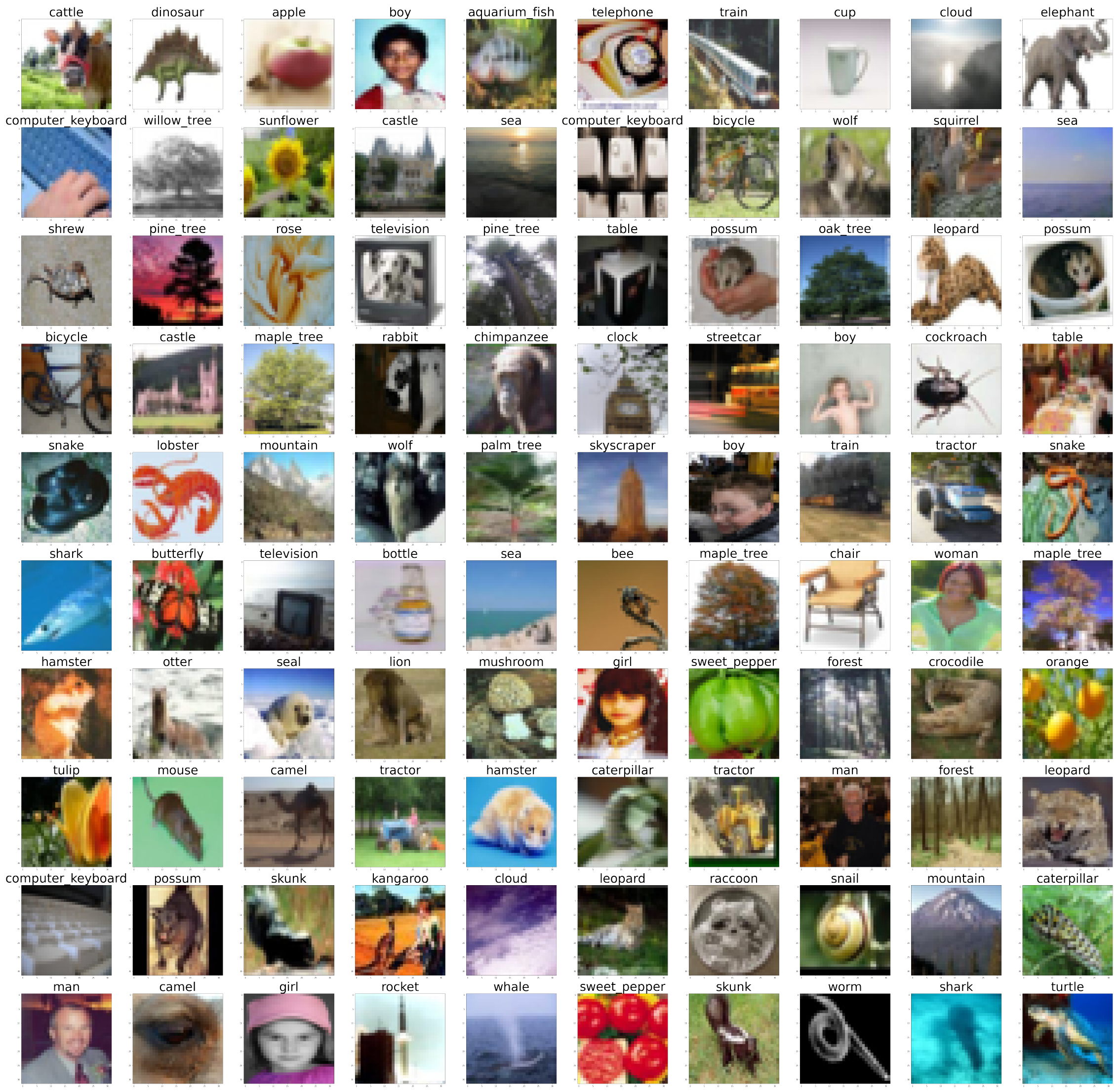
10) large man-made outdoor things ————–> bridge, castle, house, road, skyscraper
11) large natural outdoor scenes ————–> cloud, forest, mountain, plain, sea
12) large omnivores and herbivores ————–> camel, cattle, chimpanzee, elephant, kangaroo
13) medium-sized mammals ————–> fox, porcupine, possum, raccoon, skunk
14) non-insect invertebrates ————–> crab, lobster, snail, spider, worm
15) people ————–> baby, boy, girl, man, woman
16) reptiles ————–> crocodile, dinosaur, lizard, snake, turtle
17) small mammals ————–> hamster, mouse, rabbit, shrew, squirrel
18) trees ————–> maple, oak, palm, pine, willow
19) vehicles 1 ————–> bicycle, bus, motorcycle, pickup truck, train
20) vehicles 2 ————–> lawn-mower, rocket, streetcar, tank, tractor

3 CIFAR-100 Dataset class labels

[32]:
```
labels =  ['apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed',
           'bee', 'beetle', 'bicycle', 'bottle', 'bowl', 'boy', 'bridge',␣
 ↪'bus', 'butterfly',
           'camel', 'can', 'castle', 'caterpillar', 'cattle', 'chair',␣
 ↪'chimpanzee', 'clock',
           'cloud', 'cockroach', 'couch', 'crab', 'crocodile', 'cup',
           'dinosaur', 'dolphin', 'elephant', 'flatfish', 'forest', 'fox',␣
 ↪'girl',
           'hamster', 'house', 'kangaroo', 'computer_keyboard',
           'lamp', 'lawn_mower', 'leopard', 'lion', 'lizard', 'lobster', 'man',
           'maple_tree', 'motorcycle', 'mountain', 'mouse', 'mushroom',
           'oak_tree', 'orange', 'orchid', 'otter', 'palm_tree', 'pear',
           'pickup_truck', 'pine_tree', 'plain', 'plate', 'poppy', 'porcupine',␣
 ↪'possum',
           'rabbit', 'raccoon', 'ray', 'road', 'rocket', 'rose',
           'sea', 'seal', 'shark', 'shrew', 'skunk', 'skyscraper', 'snail',
           'snake', 'spider', 'squirrel', 'streetcar', 'sunflower',␣
 ↪'sweet_pepper',
           'table', 'tank', 'telephone', 'television', 'tiger', 'tractor',
           'train', 'trout', 'tulip', 'turtle',
           'wardrobe', 'whale', 'willow_tree', 'wolf', 'woman', 'worm']
```

4 All The Images With Labels in CIFAR100 Dataset

```
[33]: fig,ax=plt.subplots(10,10,figsize=(96,96))
      ax=ax.ravel()
      for i in range(0,100):
          image=X_train[i]
          label=int(y_train[i])
          ax[i].imshow(image)
          ax[i].set_title(labels[label],fontdict={'size':60})
```



5 Normalizing Data

Dividing it by 255 as pixel ranges from 0 to 255 and 255 is maximum value

[34]:
```
# normalize data
X_train=X_train.astype("float32")
X_test=X_test.astype("float32")

X_train/=255.0
X_test/=255.0
```

6 One-hot encoding label

[35]:
```
y_train=to_categorical(y_train,100)
y_test=to_categorical(y_test,100)
print('One hot Encoded: ',y_train[0])
```

```
One hot Encoded:  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0.]
```

7 Shape of Train and Test Data

Shape is (32,32,3) as image is of size 32x32 and 3 RGB(Red-Green-Blue) Channels

[36]:
```
print("Shape of X_train: ",X_train.shape)
print("Shape of y_train: ",y_train.shape)
print("Shape of X_test: ",X_test.shape)
print("Shape of y_test: ",y_test.shape)
```

```
Shape of X_train:  (50000, 32, 32, 3)
Shape of y_train:  (50000, 100)
Shape of X_test:  (10000, 32, 32, 3)
Shape of y_test:  (10000, 100)
```

8 CNN Architecture

[55]:
```
# model creation
model=Sequential()
model.add(Conv2D(100,(2,2), input_shape=(32,32,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3)))

model.add(Conv2D(80,(2,2),padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(50,(2,2),padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(512,activation='relu'))
model.add(Dense(100, activation='softmax'))
```

9 Compiling Model And Gettin Model Summary

- categorical_crossentrophy as loss as it is multiclass classification and will return a array with probability of all classes.
- adam is one of fameous optimizer .
- metric here we are using accuracy for evaluation as metric.

[56]:
```
#compile model
model.compile(
loss='categorical_crossentropy',
optimizer='adam',
metrics=['accuracy'])

#model summary

model.summary()
```

```
Model: "sequential_7"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_28 (Conv2D)          (None, 31, 31, 100)       1300

 max_pooling2d_21 (MaxPoolin  (None, 10, 10, 100)      0
 g2D)

 conv2d_29 (Conv2D)          (None, 10, 10, 80)        32080

 max_pooling2d_22 (MaxPoolin  (None, 5, 5, 80)         0
 g2D)

 conv2d_30 (Conv2D)          (None, 5, 5, 50)          16050

 max_pooling2d_23 (MaxPoolin  (None, 2, 2, 50)         0
 g2D)

 flatten_7 (Flatten)         (None, 200)               0

 dense_21 (Dense)            (None, 1024)              205824

 dense_22 (Dense)            (None, 512)               524800
 dense_23 (Dense)            (None, 100)              51300

=================================================================
Total params: 831,354
Trainable params: 831,354
Non-trainable params: 0
_________________________________________________________________
```

10 Training Model

- Epochs: It refers to the one cycle of all training data through neural network.
- Batch_size: It refers to the number of example after which weights should be modified.
- EarlyStopping: It lets the model stop training if a given paramter is not changing here it is accuracy and patient means consecutive times the change is not occuring.

[57]:
```
# to train model

EPOCHS=50

Batch_size=32

callback=EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)

history=model.fit(X_train,y_train,
         batch_size=Batch_size,
         validation_data=(X_test,y_test),
         callbacks=callback,
         epochs=EPOCHS,
         verbose=1,
         shuffle=True)
```

```
Epoch 1/50
1563/1563 [==============================] - 40s 25ms/step - loss: 3.8497 -
accuracy: 0.1013 - val_loss: 3.4019 - val_accuracy: 0.1748
Epoch 2/50
1563/1563 [==============================] - 43s 28ms/step - loss: 3.1594 -
accuracy: 0.2193 - val_loss: 3.0102 - val_accuracy: 0.2483
Epoch 3/50
1563/1563 [==============================] - 44s 28ms/step - loss: 2.8296 -
accuracy: 0.2825 - val_loss: 2.8163 - val_accuracy: 0.2943
Epoch 4/50
1563/1563 [==============================] - 44s 28ms/step - loss: 2.6141 -
accuracy: 0.3272 - val_loss: 2.7811 - val_accuracy: 0.3078
Epoch 5/50
1563/1563 [==============================] - 43s 28ms/step - loss: 2.4455 -
accuracy: 0.3609 - val_loss: 2.6050 - val_accuracy: 0.3419
Epoch 6/50
```

```
1563/1563 [==============================] - 47s 30ms/step - loss: 2.2901 -
accuracy: 0.3934 - val_loss: 2.5362 - val_accuracy: 0.3524
Epoch 7/50
1563/1563 [==============================] - 49s 31ms/step - loss: 2.1471 -
accuracy: 0.4243 - val_loss: 2.4571 - val_accuracy: 0.3777
Epoch 8/50
1563/1563 [==============================] - 44s 28ms/step - loss: 2.0160 -
accuracy: 0.4529 - val_loss: 2.5043 - val_accuracy: 0.3660
Epoch 9/50
1563/1563 [==============================] - 44s 28ms/step - loss: 1.8841 -
accuracy: 0.4830 - val_loss: 2.5080 - val_accuracy: 0.3715
Epoch 10/50
1563/1563 [==============================] - 44s 28ms/step - loss: 1.7502 -
accuracy: 0.5151 - val_loss: 2.5362 - val_accuracy: 0.3822
Epoch 11/50
1563/1563 [==============================] - 44s 28ms/step - loss: 1.6168 -
accuracy: 0.5436 - val_loss: 2.6246 - val_accuracy: 0.3736
Epoch 11: early stopping
```

11 Observations:

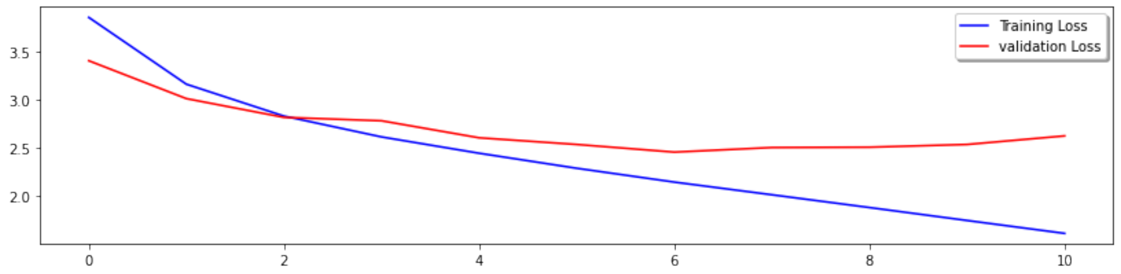
- We trained the model and got accuracy of 54.36% on training data and 37.36 on validation data.

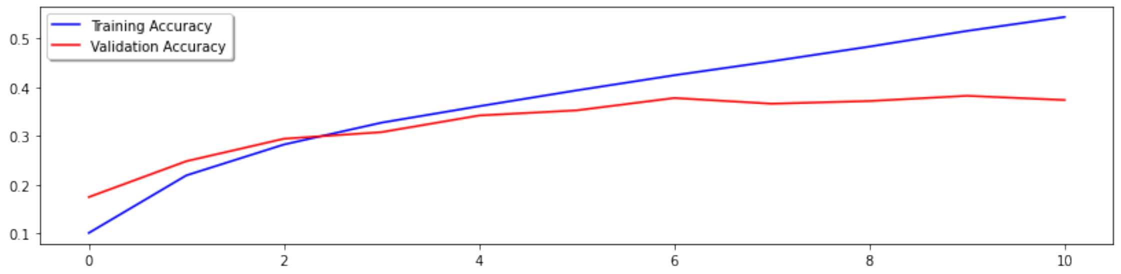
12 Show Loss and Accuracy Plots

[58]:
```
fig, ax = plt.subplots(2, 1,figsize=(14,7))

ax[0].plot(history.history['loss'], color='b', label="Training Loss")
ax[0].plot(history.history['val_loss'], color='r', label="validation␣
  ↪Loss",axes=ax[0])
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training Accuracy")
ax[1].plot(history.history['val_accuracy'], color='r', label="Validation␣
  ↪Accuracy")
legend = ax[1].legend(loc='best', shadow=True)
```





13 Saving Trained Model

[59]:
```
from keras.models import save_model,load_model
model.save("image_reco_cifar100")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
3 of 3). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: image_reco_cifar100\assets

INFO:tensorflow:Assets written to: image_reco_cifar100\assets
```

14 Saving Trained Weights

[60]:
```
model.save_weights("model_weights_cifar100.h5")
```

15 Making predictions on the images

[61]:
```
from keras.models import model_from_json
import numpy as np
```

16 load an image file to test

[73]:
```
from keras.utils import load_img,img_to_array
```

17 Loading Saved Model

[64]:
```
model_loaded=load_model("image_reco_cifar100")
```

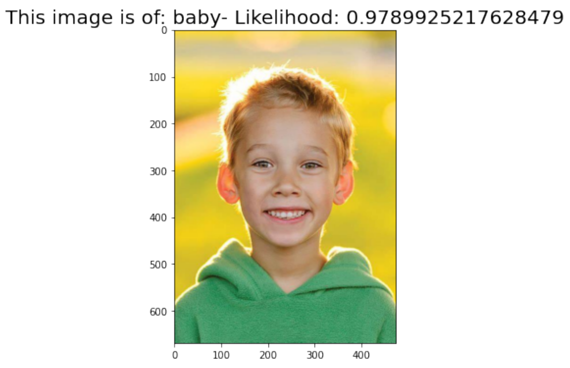
18 Creating Image Detctor

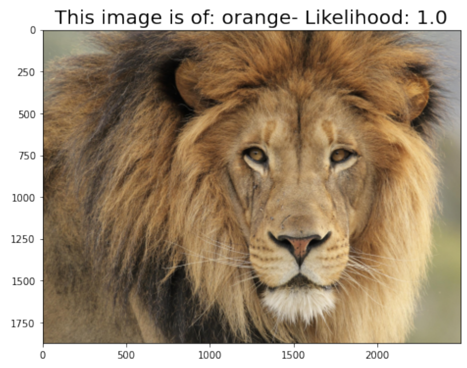
- It will load image
- Convert it to numpy array
- Expand it's dimension
- As it is a sparse matrix of result get result with highest probability using argmax
- Also it return probability of prediction

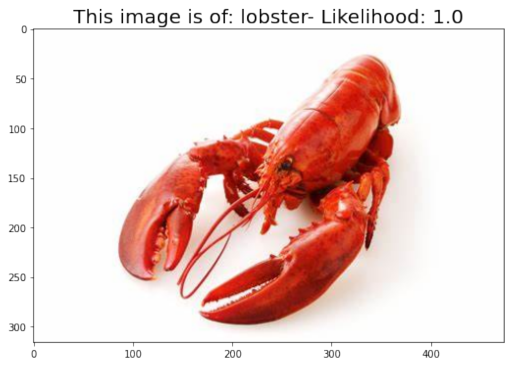
[89]:
```
def image_predictor(img_path):
    img= load_img(img_path,target_size=(32,32))
    img_show=load_img(img_path)
    image_to_test= img_to_array(img)
    list_of_images= np.expand_dims(img,axis=0)
    results= model_loaded.predict(list_of_images)
    single_result= results[0]
    most_likely_class_index= int(np.argmax(single_result))
    class_likelihood= single_result[most_likely_class_index]
    class_label= labels[most_likely_class_index]
    return img_show,class_likelihood,class_label
```

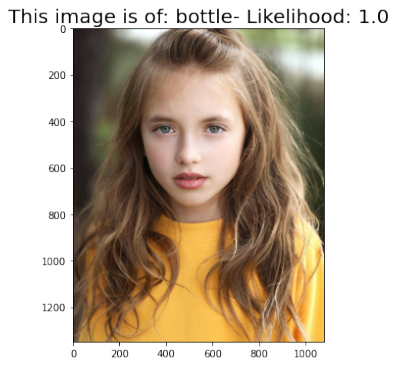
[95]:
```
# Print the result
list_image_to_test=['boy.jpg','lion.jpg','lobster.jpg','girl.jpg','tv.
 ↪jpg','sunflower.jpg']
fig,ax=plt.subplots(3,2,figsize=(20,20))
ax=ax.ravel()
for i in range(len(list_image_to_test)):
    img,class_likelihood,class_lable=image_predictor(list_image_to_test[i])
    ax[i].imshow(img)
    ax[i].set_title(f'This image is of: {class_lable}- Likelihood:␣
 ↪{class_likelihood}',fontdict={'size':20})
```

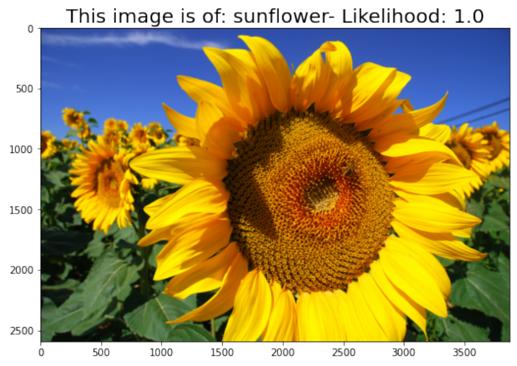
```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 16ms/step
```











19 Conclusion

- We can see it classified televison,baby,lobster,sunflower
- Also it missclassifed girl as bottel,lion as orange

```
[ ]:
```