Understanding of a Convolutional Neural Network

INFO 7390 Project Report

Varsha Premani
Student
Northeastern University
Massachusetts, U.S

Yashraj Swarnkar
Student
Northeastern University
Massachusetts, U.S

***Abstract*— The term Deep Learning or Deep Neural Network refers to Artificial Neural Networks (ANN) with multi layers . Over the last few decades, it has been considered to be one of the most powerful tools, and has become very popular in the literature as it is able to handle a huge amount of data. The interest in having deeper hidden layers has recently begun to surpass classical methods performance in different fields; especially in pattern recognition. One of the most popular deep neural networks is the Convolutional Neural Network (CNN). It takes this name from mathematical linear operation between matrixes called convolution. As a part of this project, we tune the hyperparameters involved in modeling the network to improve the learning capabilities of the network.**

Keywords*— *machine learning, artificial neural networks, deep learning, convolutional neural networks ,computer vision, Image classification

I. INTRODUCTION

Convolutional neural networks are deep artificial neural networks primarily used to classify images, cluster them by similarity and perform object recognition within scenes. CNN algorithm can identify faces, individuals, street signs, tumors, platypuses and many other aspects of visual data. In purely mathematical terms, convolution is a function derived from two given functions by integration which expresses how the shape of one is modified by the other. The model consists of two convolutional layers , relu activation, followed by a deep layer. The convolution formula is $(f * g)(t) \stackrel{\text{def}}{=} \int f(\tau)g(t - \infty - \infty \tau)\, d\tau$

CNN consists of multiple layers; including convolutional layer, non-linearity layer, maxpooling layer and fully connected layer. The convolutional and fully- connected layers have parameters but the pooling and non-linearity layers don't have parameters. The CNN has an excellent performance in machine learning problems. Specially the applications that deal with image data, such as largest image classification data set (Image Net), computer vision, and in natural language processing (NLP) and the results achieved were very amazing . In this paper we will explain and define all the elements and important issues related to CNN, and how these elements work. In addition, we will also state the parameters that effect CNN efficiency. This paper assumes that the readers have adequate knowledge about both machine learning and artificial neural network.

This paper is organized as follows: Background gives the general area and specific problem we addressed. Data preprocessing introduces the basic image to array conversion of our dataset and the normalization of images. In Result, we compared the result of different models and the impact of different hyperparameter of same method. Some graphs included in this part which visualizes the performance of model. Conclusion part gives the conclusion of this paper.

II. ABOUT DATA

The dataset contains 500 rendered images of different types of horses in different stances in different locations. It additionally contains 527 rendered pictures of humans in different postures and areas. Emphasis has been taken to ensure diversity of humans, and to that end there are both men and women as well as Asian, Black, South Asian, and Caucasians present in the training set. The validation set adds 6 different figures of different gender, race and poses to ensure breadth of data.

III. DATA PREPROCESSING

The dataset for this project is: Humans vs Horse classification dataset. The Link is: http://www.laurencemoroney.com/horses-or-humans-dataset/. In this dataset, there are 1027 images in train and 256 images in test.

Every image is a matrix of pixel values. The range of values that can be encoded in each pixel depends upon its bit size. Most commonly, we have 8 bit or 1 Byte-sized pixels. Thus the possible range of values a single pixel can represent is [0, 255]. However, with colored images, particularly RGB (Red, Green, Blue)-based images, the presence of separate color channels (3 in the case of RGB images) introduces an additional 'depth' field to the data, making the input 3-dimensional.

Hence, for a given RGB image of size, say 255×255 (Width x Height) pixels, we'll have 3 matrices associated with each image, one for each of the color channels. Thus, the image in its entirety, constitutes a 3-dimensional structure called the Input Volume (255x255x3).

In order to Normalize this data, we need the values in Float. So, we are converting them. Dividing the data with 255 will normalize the pixel intensity values.

```
print('1st training image as array',x_train[0])
```

```
1st training image as array [[[249. 255. 253.]
  [251. 255. 253.]
  [254. 255. 255.]
  ...
  [255. 255. 255.]
  [255. 255. 255.]
  [255. 255. 255.]]

 [[251. 255. 253.]
  [252. 255. 253.]
  [254. 255. 255.]
  ...
  [255. 255. 255.]
  [255. 255. 255.]
  [255. 255. 255.]]

 [[254. 255. 255.]
  [254. 255. 255.]
  [254. 255. 255.]
  ...
  [255. 255. 255.]
  [255. 255. 255.]
  [255. 255. 255.]]

 ...
```

Fig. 1. Before Normalization

Figure 2 is the figure which shows the distribution after normalization.

```
# time to re-scale so that all the pixel values lie within 0 to 1
x_train = x_train.astype('float32')/255
x_test = x_test.astype('float32')/255
x_train[0]
```

```
array([[[0.9764706 , 1.        , 0.99215686],
        [0.9843137 , 1.        , 0.99215686],
        [0.99607843, 1.        , 1.        ],
        ...,
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ]],

       [[0.9843137 , 1.        , 0.99215686],
        [0.9882353 , 1.        , 0.99215686],
        [0.99607843, 1.        , 1.        ],
        ...,
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ]],

       [[0.99607843, 1.        , 1.        ],
        [0.99607843, 1.        , 1.        ],
        [0.99607843, 1.        , 1.        ],
        ...,
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ]],

       ...,
```

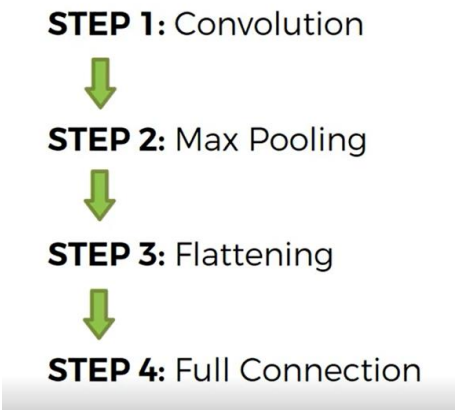
Fig. 2. After normalization

IV. CNN AND ITS ELEMENTS

Convolutional Neural Networks (CNNs) are a class of Artificial Neural Networks (ANNs) which have proven to be very effective for this type of task. They have certain characteristics that share resemblance with how human beings recognize patterns in visual imagery. Convolutional neural networks share the characteristics of multilayer perceptron (and may be said to be composed of individual MLPs, although this analogy remains a bit vague): they have one input layer, one output layer and a set of – at minimum one – hidden layer(s) in between.

Components of the network are:

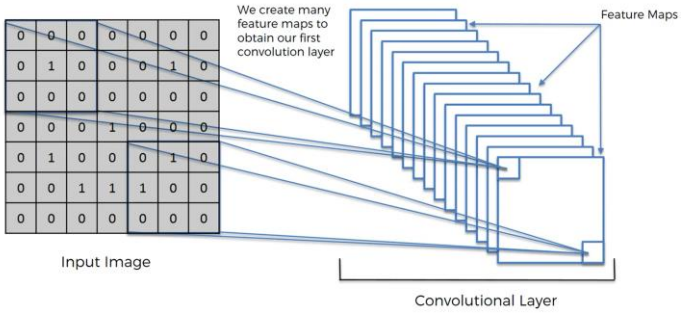
1. Layers:



A. Convolutional layers

In a CNN, the first layer that comes after the input layer is a so-called **convolutional layer.** We have the input layer on the left. This layer contains the actual image – a matrix/tensor/whatever you wish to call it of shape (width, height, depth). In our case, since we used a 300×300 RGB image above, our shape is (300, 300, 3).

A convolutional layer thus consists of a set of filters which all look at different parts of the image. The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. There are two types of results to the operation — one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying Valid Padding in case of the former, or Same Padding in the case of the latter.

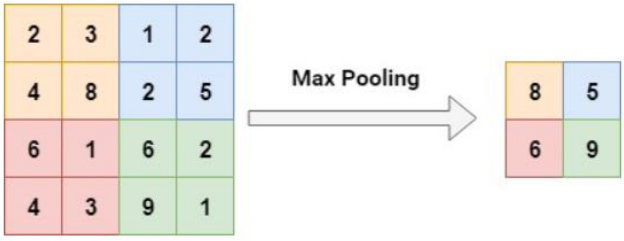


B. Pooling Layer

Like the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

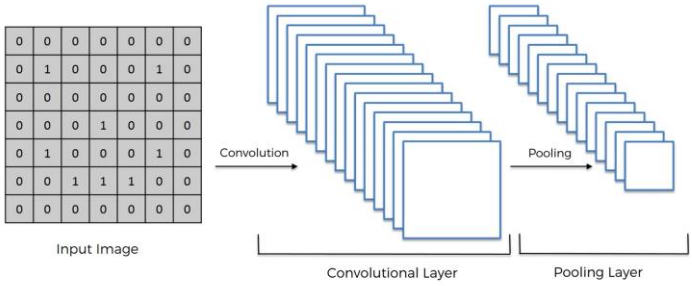
There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.

Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that Max Pooling performs a lot better than Average Pooling.



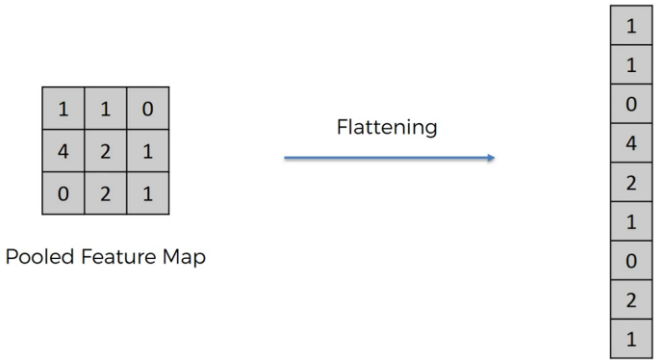
The Convolutional Layer and the Pooling Layer together form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low levels details even further, but at the cost of more computational power.

After going through the above process, we have successfully enabled the model to understand the features. Moving on, we are going to flatten the final output and feed it to a regular Neural Network for classification purposes.



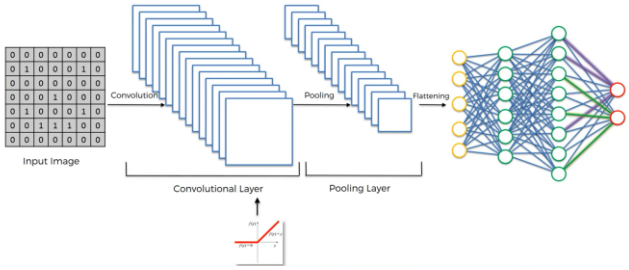
C. Flattening Layer

Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a fully connected layer. In other words, we put all the pixel data in one line and make connections with the final layer.



D. Fully connected layer

The objective of a fully connected layer is to take the results of the convolution/pooling process and use them to classify the image into a label (in a simple classification example).
The fully connected part of the CNN network goes through its own backpropagation process to determine the most accurate weights. Each neuron receives weights that prioritize the most appropriate label. Finally, the neurons "vote" on each of the labels, and the winner of that vote is the classification decision.



2. Activation Function:

The activation function is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function). The Activation Functions can be basically divided into 2 types-

i. Linear Activation Function

Since the function is a line or linear. Therefore, the output of the functions will not be confined between any range. It doesn't help with the complexity or various parameters of usual data that is fed to the neural networks.

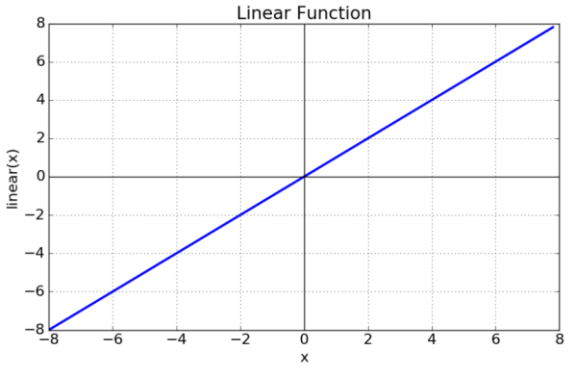


Fig: Linear Activation Function

ii. Non-linear Activation Functions

The Nonlinear Activation Functions are the most used activation functions. Nonlinearity helps to makes the graph look something like this:

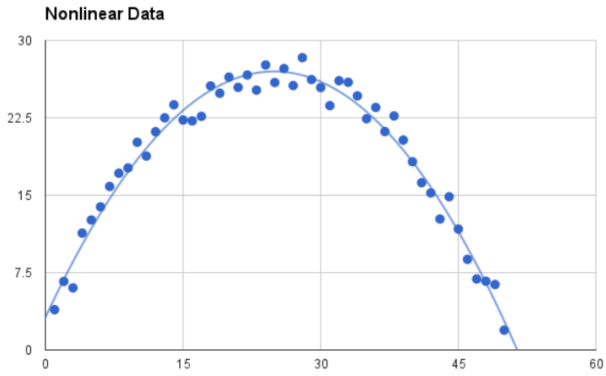


Fig: Non-linear Activation Function

The Nonlinear Activation Functions are mainly divided based on their range or curves-

a. Sigmoid or Logistic Activation Function

The Sigmoid Function curve looks like a S-shape. The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we must predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.
The softmax function is a more generalized logistic activation function which is used for multiclass classification.

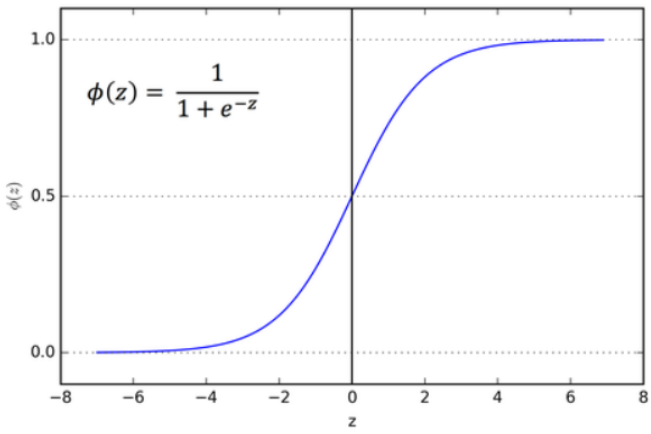


Fig: Sigmoid Function

b. Tanh or hyperbolic tangent Activation Function

tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped). The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.

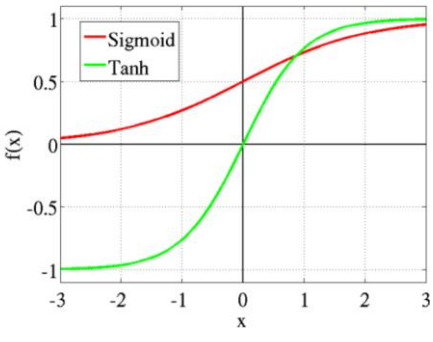


Fig: tanh v/s Logistic Sigmoid

c. ReLU (Rectified Linear Unit) Activation Function

The ReLU is the most used activation function in the world right now. Since, it is used in almost all the convolutional neural networks or deep learning. It is half rectified (from bottom). f(z) is zero when z is less than zero and f(z) is equal to z when z is

above or equal to zero.

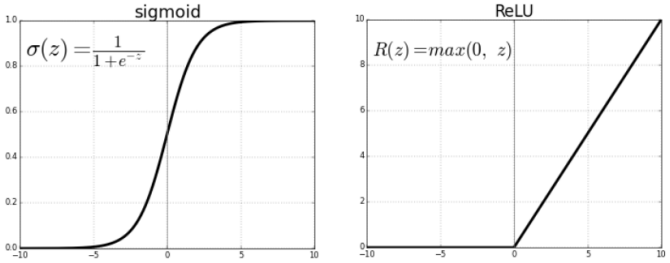


Fig: ReLU v/s Logistic Sigmoid

3. **Cost or Loss function**

The cost or loss function has an important job in that it must faithfully distill all aspects of the model down into a single number in such a way that improvements in that number are a sign of a better model. The cost function reduces all the various good and bad aspects of a possibly complex system down to a single number, a scalar value, which allows candidate solutions to be ranked and compared.

a. Mean Squared Error Loss

Mean Squared Error loss, or MSE for short, is calculated as the average of the squared differences between the predicted and actual values. The result is always positive regardless of the sign of the predicted and actual values and a perfect value is 0.0. The loss value is minimized, although it can be used in a maximization optimization process by making the score negative.

b. Cross-Entropy Loss (or Log Loss)

Cross-entropy loss is often simply referred to as "cross-entropy," "logarithmic loss," "logistic loss," or "log loss" for short. Each predicted probability is compared to the actual class output value (0 or 1) and a score is calculated that penalizes the probability based on the distance from the expected value. The penalty is logarithmic, offering a small score for small differences (0.1 or 0.2) and enormous score for a large difference (0.9 or 1.0). Cross-entropy loss is minimized, where smaller values represent a better model than larger values. A model that predicts perfect probabilities has a cross entropy or log loss of 0.0.Cross-entropy for a binary or two class prediction problem is calculated as the average cross entropy across all examples.

4. Optimizers

Optimizers tie together the loss function and model parameters by updating the model in response to the output of the loss function. In simpler terms, optimizers shape and mold your model into its most accurate possible form by futzing with the weights. The loss function is the guide to the terrain, telling the optimizer when it's moving in the right or wrong direction.

a. Stochastic Gradient Descent

Instead of calculating the gradients for all of your training examples on every pass of gradient descent, it's sometimes more efficient to only use a subset of the training examples each time. Stochastic gradient descent is an implementation that either uses batches of examples at a time or random examples on each pass.

b. RMSprop:

RMSprop is a special version of Adagrad developed by Professor Geoffrey Hinton in his neural nets class. Instead of letting all the gradients accumulate for momentum, it only accumulates gradients in a fixed window. RMSprop is like Adaprop, which is another optimizer that seeks to solve some of the issues that Adagrad leaves open.

c. Adam:

Adam stands for adaptive moment estimation and is another way of using past gradients to calculate current gradients. Adam also utilizes the concept of momentum by adding fractions of previous gradients to the current one. This optimizer has become widespread and is practically accepted for use in training neural nets.

It's easy to get lost in the complexity of some of these new optimizers. We just need to remember that they all have the same goal: minimizing our loss function. Even the most complex ways of doing that are simple at their core.

5. Network Initializers:

Initializations define the way to set the initial random weights of Keras layers. The aim of weight initialization is to prevent layer activation outputs from exploding or vanishing during a forward pass through a deep neural network. If either occurs, loss gradients will either be too large or too small to flow backwards beneficially, and the network will take longer to converge, if it is even able to do so at all.

A good initialization of the network is one of the most important prerequisites for achieving rapid convergence of training and avoiding the problem of gradient disappearance. The bias parameters can be initialized to zero, while weight parameters must be carefully introduced to break the symmetry between hidden units of the same layer. For example, if the network is not initialized properly, each layer measures its own input in k, the final output measures the original input in k^L, where L is the number of layers. In this case, the value of $k > 1$ leads to extremely high initial layer values, while the value of $k < 1$ leads to diminishing output value and gradients.

Zeros: Initializer that generates tensors initialized to 0.

Ones: Initializer that generates tensors initialized to 1.

Constant: Initializer that generates tensors initialized to a constant value.

RandomNormal: Initializer that generates tensors with a normal distribution.

RandomUniform: Initializer that generates tensors with a uniform distribution.

TruncatedNormal: Initializer that generates a truncated normal distribution.

VarianceScaling: Initializer capable of adapting its scale to the shape of weights.

Orthogonal: Initializer that generates a random orthogonal matrix.

Identity: Initializer that generates the identity matrix.

lecun_uniform: LeCun uniform initializer.

glorot_normal: Glorot normal initializer, also called Xavier normal initializer.

glorot_uniform: Glorot uniform initializer, also called Xavier uniform initializer.

he_normal: He normal initializer.

lecun_normal: LeCun normal initializer.

he_uniform: He uniform variance scaling initializer.

6. Number of epochs:

The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.

One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. For example, as above, an epoch that has one batch is called the batch gradient

descent learning algorithm.

V. Step by Step Implementation of Base Model

a. Import all the libraries which are required for implementation of model.

```
[ ] import os
    import keras
    import zipfile
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import matplotlib.style  as style
    from google.colab import files
    from sklearn.metrics     import accuracy_score, roc_auc_score
    from keras.layers        import Dense, Dropout, Flatten, MaxPooling2D,Convolu
    from keras.models        import Sequential, Model, load_model
    from keras.optimizers    import Adam, sgd, Adagrad
    from keras.preprocessing import image
    from keras.preprocessing.image import ImageDataGenerator
```

```
Using TensorFlow backend.
```

b. Upload the dataset and defining dataset directories.

```
[ ] local_zip = '/tmp/horse-or-human.zip'
    zip_ref = zipfile.ZipFile(local_zip, 'r')
    zip_ref.extractall('/tmp/horse-or-human')
    local_zip = '/tmp/validation-horse-or-human.zip'
    zip_ref = zipfile.ZipFile(local_zip, 'r')
    zip_ref.extractall('/tmp/validation-horse-or-human')
    zip_ref.close()
```

Defining the directories for the datasets

```
[ ] # Directory with our training horse pictures
    train_horse_dir = os.path.join('/tmp/horse-or-human/horses')

    # Directory with our training human pictures
    train_human_dir = os.path.join('/tmp/horse-or-human/humans')

    # Directory with our training horse pictures
    validation_horse_dir = os.path.join('/tmp/validation-horse-or-human/horses')

    # Directory with our training human pictures
    validation_human_dir = os.path.join('/tmp/validation-horse-or-human/humans')
```

c. Building CNN model from Scratch:

ImageDataGenerator will read pictures from source folders, convert them to float32 tensors, and feed them (with their labels) to the network. We'll have one generator for the training images and one for the validation images. Our generators will yield batches of images of size 300x300 and their labels (binary). We preprocessed our images by normalizing the pixel values to be in the [0, 1] range (originally all values are in the [0, 255] range). We have normalized our data using rescale parameter. ImageDataGenerator allows you to instantiate generators of augmented image batches via .flow_from_directory(directory)

```
[ ] batch_size = 128
    input_size = (300,300)
```

Normalizing the dataset using rescale paramerter. *ImageDataGenerator* allows you to instantiate generators of augmented image batches via .flow_from_directory(directory)

```
[ ]
    # All images will be rescaled by 1./255
    train_datagen = ImageDataGenerator(rescale=1/255)
    validation_datagen = ImageDataGenerator(rescale=1/255)

    # Flow training images in batches of 128 using train_datagen generator
    train_generator = train_datagen.flow_from_directory(
            '/tmp/horse-or-human/',  # This is the source directory for training images
            target_size=(300, 300),  # All images will be resized to 150x150
            batch_size=128,
            # Since we use binary_crossentropy loss, we need binary labels
            class_mode='binary')

    # Flow training images in batches of 128 using train_datagen generator
    validation_generator = validation_datagen.flow_from_directory(
            '/tmp/validation-horse-or-human/',  # This is the source directory for training images
            target_size=(300, 300),  # All images will be resized to 150x150
            batch_size=32,
            # Since we use binary_crossentropy loss, we need binary labels
            class_mode='binary')
```

```
Found 1027 images belonging to 2 classes.
Found 256 images belonging to 2 classes.
```

d. Building CNN model from Scratch:

Initialize the model. We initialize our model by using Sequential function.

```
model = Sequential()
```

Inserting first convolution layer in the model. Layer is having 16 convolution layer having size 3*3. Activation function is present in the layer to increase nonlinearity in the model

```
model.add(Convolution2D(16, (3,3), activation='relu', input_shape=(
```

To reduce dimension of layers we use pooling layer in the model.

```
model.add(MaxPooling2D(pool_size = (2,2)))
```

Flattening layer will convert 2D image in to 1D vector.

```
model.add(Flatten())
```

A dense layer that takes that vector and generates probabilities for 2 target labels, using a SoftMax activation function. Dropout are added in model to avoid overfitting.

```
# 512 neuron hidden layer
model.add(Dense(512, activation = "relu",init = 'normal'))
# Only 1 output neuron. It will contain a value from 0-1 where 0 for 1 class ('horses') and 1 for the other ('humans')
model.add(Dropout(0.3))
model.add(Dense(1, activation = "sigmoid"))
```

e. Complete CNN model.

In the model we have 5 convolution layers, each with a pooling layer following it.

```
model = Sequential()
model.add(Convolution2D(16, (3,3), activation='relu', input_shape=(300, 300, 3)))
model.add(MaxPooling2D(pool_size = (2,2)))
# The second convolution
model.add(Convolution2D(32, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
# The third convolution
model.add(Convolution2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
# The fourth convolution
model.add(Convolution2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
# The fifth convolution
model.add(Convolution2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
# Flatten the results to feed into a DNN
model.add(Flatten())
# 512 neuron hidden layer
model.add(Dense(512, activation = "relu",init = 'normal'))
# Only 1 output neuron. It will contain a value from 0-1 where 0 for 1 class ('horses') a
model.add(Dropout(0.3))
model.add(Dense(1, activation = "sigmoid"))

model.compile(optimizer = 'adam',loss = 'binary_crossentropy', metrics = ['accuracy'])
model.summary()
```

model.summary() will tell about summary of model.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 298, 298, 16)      448
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 149, 149, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 147, 147, 32)      4640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 73, 73, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 71, 71, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 35, 35, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 33, 33, 64)        36928
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1606144
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 1,704,097
Trainable params: 1,704,097
```

f. Training of model using .fit() method

```
history = model.fit(
      train_generator,
      steps_per_epoch=train_generator.samples // batch_size,
      epochs=15,
      verbose=1,
      validation_data = validation_generator,
      validation_steps=validation_generator.samples // batch_size)
```

For model training, we will configure the specification by compiling the model.

```
model.compile(loss='adam',
              optimizer='binary_crossentropy',
              metrics=['accuracy'])
```

g. Test the model

Now we have trained our model. Now it's time to test the model. From below snippet code you can test this model.

```
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/' + fn
  img = image.load_img(path, target_size=(300, 300))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=10)

  print("%.2f" % classes[0])

  from IPython.display import Image
  display(Image(filename=fn))

  if classes[0]>0.5:
    print(fn + " is a human")

  else:
    print(fn + " is a horse")
```

```
Choose Files  No file chosen   Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to en
Saving IMG-4682 (1).jpg to IMG-4682 (1).jpg
1.00
```

```
IMG-4682 (1).jpg is a human
```

Code Implementation:

The code repository to implementation of the same along with various hyperparameter tuning: https://colab.research.google.com/drive/12SIY-hpV2H55XyejOkTazJLAWtsL7rlU#scrollTo=qNf45ptBhzVz

OBSERVATIONS:
Based on code implemented, we were able to draw the observations as below:

1. model (Base Model)
No. of convolution layers : **5** , Activation Function : **relu** , Optimizer : **Adam**, Loss Function : **Binary Cross Entropy**, Epoch : **15**, Initializer : **normal**
Accuracy : 0.9967
Validation Accuracy : 0.8594

2. model1 (Changed Activation function in Base Model)
No. of convolution layers : 5 , Activation Function : **tanh** , Optimizer : Adam, Loss Function : Binary Cross Entropy, Epoch : 15, Initializer : normal
Accuracy : 0.9432
Validation Accuracy : 0.6055

3. model2 (Changed Cost Function function in Base Model)
No. of convolution layers : 5 , Activation Function : relu , Optimizer : Adam, Loss Function : **mean_square_error**, Epoch : 15, Initializer : normal
Accuracy : 0.8906
Validation Accuracy : 0.7773

4. model (Changed no. of epoch in Base Model)
No. of convolution layers : 5 , Activation Function : relu , Optimizer : Adam, Loss Function : Binary Cross Entropy, Epoch : **30**, Initializer : normal
Accuracy : 1.000
Validation Accuracy : 0.8438

5. model3 (Changed optimizer fuction in Base Model)
No. of convolution layers : 5 , Activation Function : relu , Optimizer : **sgd (Stochastic Gradient Descent)**, Loss Function : Binary Cross Entropy, Epoch : 15, Initializer : normal
Accuracy : 0.9393
Validation Accuracy : 0.8320

6. model4 (Changed no. of convolution layers in Base Model)
No. of convolution layers : **2** , Activation Function : relu , Optimizer : Adam, Loss Function : Binary Cross Entropy, Epoch : 15, Initializer : normal
Accuracy : 0.4182
Validation Accuracy : 0.7656

7. model5 (Changed initializtion function in base model)
No. of convolution layers : 5 , Activation Function : relu , Optimizer : Adam, Loss Function : Binary Cross Entropy, Epoch : 15, Initializer : **glorot_normal**
Accuracy : 0.9219
Validation Accuracy : 0.7891

REFERENCES

[1] https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[2] http://www.slideshare.net/hanneshapke/introduction-to-convolutionalneural-networks.

[3] O. Abdel-hamid, L. Deng, and D. Yu, "Exploring Convolutional Neural Network Structures and Optimization Techniques for Speech Recognition," no. August, pp. 3366–3370, 2013.

[4] http://www.deeplearningbook.org/contents/convnets.html

[5] Taigman, Y., Yang, M., Ranzato, M.A. and Wolf, L., 2014. Deepface: Closing the gap to human-level performance in face verification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1701-1708).

[6] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient- based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp.2278-2324

[7] https://algorithmia.com/blog/introduction-to-optimizers

[8] https://cs231n.github.io/neural-networks-1/

[9] https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6