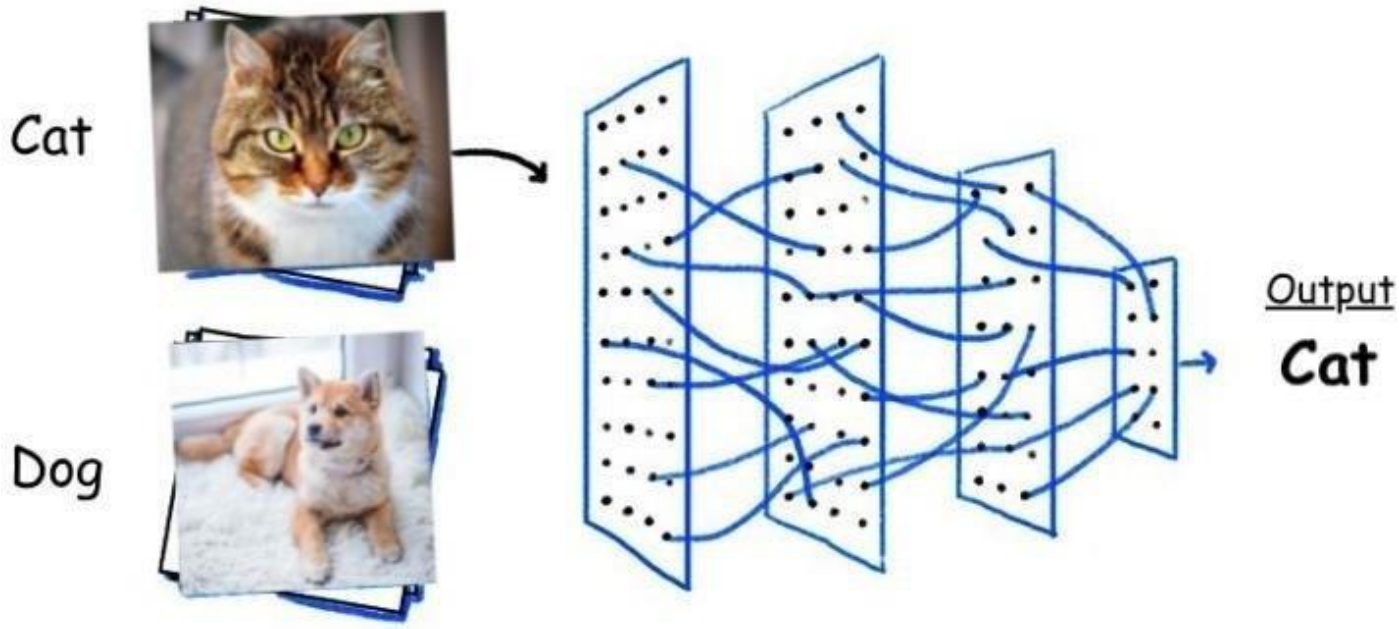
# *Cats Vs Dogs Identification Using CNN*




## What Is A CNN Model?

In deep learning, a convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery. Unlike the classical image recognition where you define the image features yourself, CNN takes the image's raw pixel data, trains the model, then extracts the features automatically for better classification.

There are various types of CNN architecture like AlexNet, VGGNet, ResNet, and GoogleNet.

CNN is mainly composed of an Input layer, Convolution layers, Activation function (ReLU), Pooling layer, Flattening, Fully connected layer, and Softmax or Sigmoid function depending on whether you're doing multiclass or binary classification.

## The Input Layer

The input layer is the first layer in any CNN model. It is the layer that receives the image for the CNN model. The dimension of the image along with the image channels is passed to the input layer.

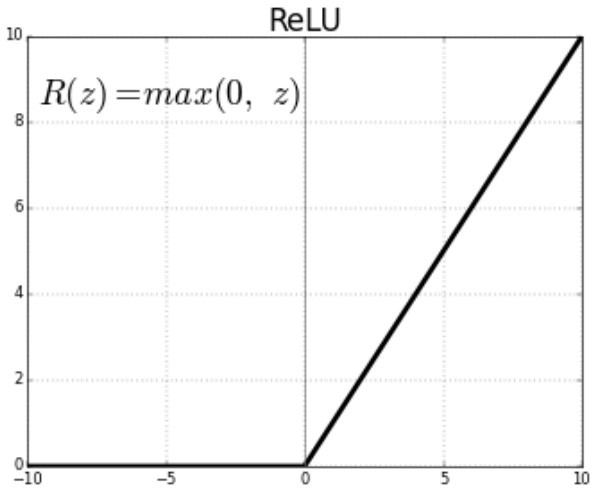
## The Convolution Layer

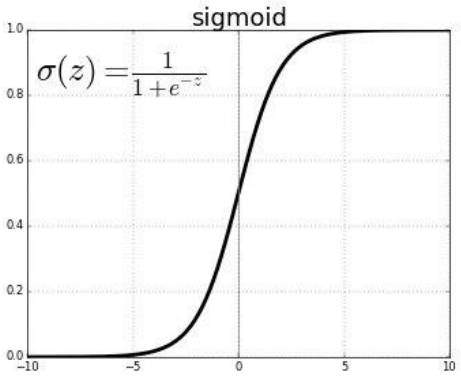
This is the first layer to extract features from the image. These extractions are done with the help of filters. It does a mathematical operation on the image with the filter and preserves the relationship between pixels by learning image features.

## Activation Function

The activation function is a node that is put at the end of or in between Neural Networks. They help to decide if the neuron would fire or not. The most popular activation functions in the CNN model are ReLu, Sigmoid, and Softmax.

ReLU:- The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero.






Sigmoid:- This function is mostly used for the model which has binary outputs and is added to the output layer. It exists between 0 and 1.

Softmax:- This Function is also added in the output layer where we have multiple classes to predict. Softmax assigns decimal probabilities to each class in a multi-class problem. Those decimal probabilities must add up to 1.0

## The Pooling Layer

Pooling layers are used to reduce the dimensions of the feature maps. It reduces the number of trainable parameters and hence the number of computations required. There are mainly two types of pooling layers used MaxPool and Average Pool.

## Flattening

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 that can be passed to fully connected layers.

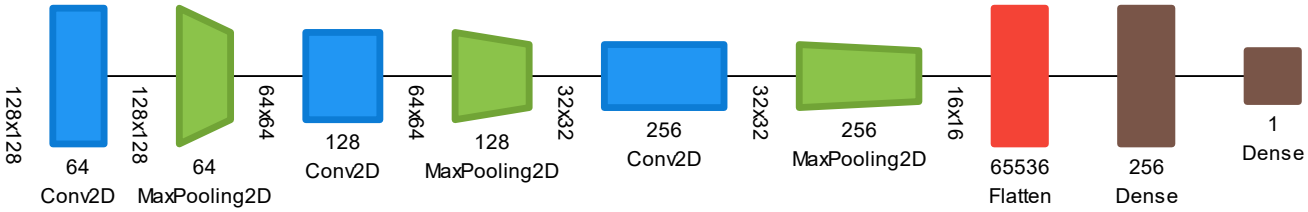
## Fully Connected Layers

Fully Connected Layers form the last few layers in the network. The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened and then fed into the fully connected layer.

# Cat-vs-Dogs Classification Custom Model Architecture

The Custom Made Cat-vs-Dog Classification Model consists of 3 convolutional layers with 3 max-pooling layers and 1 fully connected layer.

Its architecture is given below.

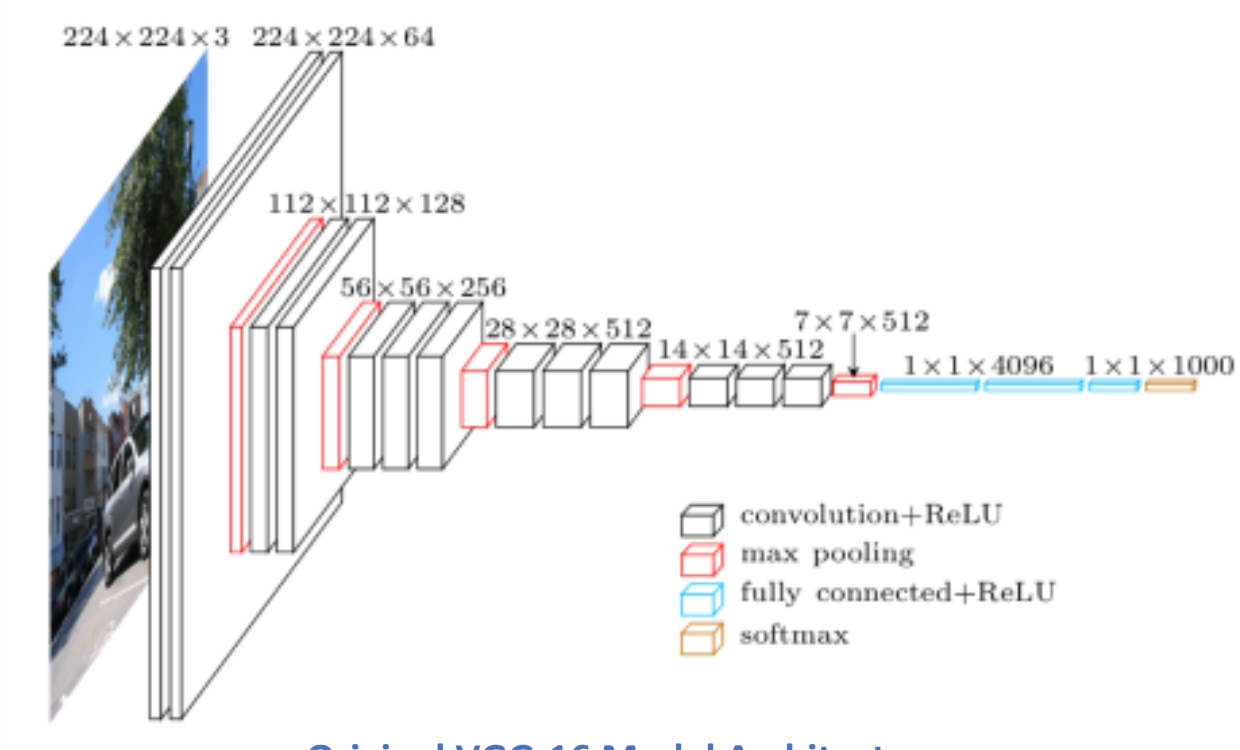



| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 128, 128, 64) | 4864 |
| max_pooling2d (MaxPooling2D) | (None, 64, 64, 64) | 0 |
| conv2d_1 (Conv2D) | (None, 64, 64, 128) | 73856 |
| max_pooling2d_1 (MaxPooling2 | (None, 32, 32, 128) | 0 |
| conv2d_2 (Conv2D) | (None, 32, 32, 256) | 295168 |
| max_pooling2d_2 (MaxPooling2 | (None, 16, 16, 256) | 0 |
| flatten (Flatten) | (None, 65536) | 0 |
| dense (Dense) | (None, 256) | 16777472 |
| dense_1 (Dense) | (None, 1) | 257 |

Total params: 17,151,617
Trainable params: 17,151,617
Non-trainable params: 0

# VGG-16 Model Architecture

The original VGG-16 model consists of 16 convolutional layers, 5 pooling layers, and 4 Dense layers. However, for my use case, I have replaced all the 4 Dense layers with a single Dense layer followed by a dropout layer.




**Original VGG-16 Model Architecture**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_1 (InputLayer) | [(None, 128, 128, 3)] | 0 |
| block1_conv1 (Conv2D) | (None, 128, 128, 64) | 1792 |
| block1_conv2 (Conv2D) | (None, 128, 128, 64) | 36928 |
| block1_pool (MaxPooling2D) | (None, 64, 64, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 64, 64, 128) | 73856 |
| block2_conv2 (Conv2D) | (None, 64, 64, 128) | 147584 |
| block2_pool (MaxPooling2D) | (None, 32, 32, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 32, 32, 256) | 295168 |
| block3_conv2 (Conv2D) | (None, 32, 32, 256) | 590080 |
| block3_conv3 (Conv2D) | (None, 32, 32, 256) | 590080 |
| block3_pool (MaxPooling2D) | (None, 16, 16, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 16, 16, 512) | 1180160 |
| block4_conv2 (Conv2D) | (None, 16, 16, 512) | 2359808 |
| block4_conv3 (Conv2D) | (None, 16, 16, 512) | 2359808 |
| block4_pool (MaxPooling2D) | (None, 8, 8, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 8, 8, 512) | 2359808 |
| block5_conv2 (Conv2D) | (None, 8, 8, 512) | 2359808 |
| block5_conv3 (Conv2D) | (None, 8, 8, 512) | 2359808 |
| block5_pool (MaxPooling2D) | (None, 4, 4, 512) | 0 |
| flatten_1 (Flatten) | (None, 8192) | 0 |
| dropout (Dropout) | (None, 8192) | 0 |
| dense_2 (Dense) | (None, 1) | 8193 |

Total params: 14,722,881
Trainable params: 8,193
Non-trainable params: 14,714,688

**Modified VGG-16 Model Architecture**

# Deep Learning Concepts Used

## *Neural Network*

It is formed by interconnected neurons. These neurons have weights, and bias which is updated during the network training. The network consists of an input layer, hidden layers, and an output layer. The hidden layer is where the processing of data is done.

## *Activation Functions*

The activation function translates the input signals to output signals. We apply a nonlinear function like an activation function to a linear function coming from the input layer. Common activation functions are sigmoid, Relu, Softmax, and tanh.

## *Loss Functions*

When we build a network, the network tries to predict the output as close as possible to the actual value. We measure this accuracy of the network using the loss function. Our motive while running the network is to increase our prediction accuracy and to reduce the error, hence minimizing the cost function.

## *Gradient Descent*

Gradient descent is an optimization algorithm for minimizing the loss. The basic idea behind it is that the gradient of the function should try to find the direction of the steeper section (towards minima).

## *Batches*

While training a neural network, instead of sending the entire input in one go, we divide the input into several parts of equal size randomly. Larger batch sizes help in reducing the training time along with the cost of little low accuracy.

## *Epochs*

An epoch is defined as a single training iteration of all batches in both forward and backpropagation. This means 1 epoch is a single forward and backward pass of the entire input data.

## *Batch Normalization*

Batch Normalization is done to ensure that the distribution of data is the same as the next layer hoped to get. When we are training the neural network, the weights are changed after each step of gradient descent. It is performed on the output of layers of each batch.

## *Transfer Learning*

Transfer Learning is a machine learning method where we reuse a pre-trained model as the starting point for a model on a new task. Since the model already has weights and biases so with a little tweak one can get high accuracy with comparatively less training time.

## Dataset

The dataset consists of 4000 images of cats and dogs each for training and 1000 images of both for test in directories. The image sizes of all images are different and are colored images.

These datasets are loaded from the directory into the Training and Testing set and a part of the training set (11%) will be used as a validation set while training. Further, it is divided into the batch size of 32, and the image size is rescaled to 128 by 128 with the help of data augmentation.

## Data Augmentation

It is a commonly used technique in image processing used to create more training data. It modifies the original training images a bit for example by rotating, flipping, zooming in and out, etc.

## Image Pre-Processing

Image preprocessing is the steps taken to format images before they are used by model training and inference. This includes resizing, orienting, and color corrections.

A few of them which are used in my image is explained below.

### *Rescaling*

Every digital image is formed by pixels having values in the range 0~255. 0. Similarly, in color images also for all three channels, the range is from 0~255.

Since 255 is the maximum pixel value. Rescale 1./255 is to transform every pixel value from range [0,255] -> [0,1].

This is done because some images are in a high pixel range, and some are in a low pixel range. The images are all sharing the same model, weights, and learning rate. So if we don't scale it equally some differences will occur in the images and either high loss or weak loss is generated. Also as training neural networks with this range gets efficient

## *Resizing*

Resizing is used as the images in the data set are of different dimensions and some of them are quite large. The larger the image the more data can be gathered by training time will also increase tremendously. That's why resizing each image to a small size is necessary. Images are generally resized to 32x32, 128x128, 256x256 or 512x512.

In my model, I have resized it to 128x128.

# *Project Details*

In this project, I have built two CNN models. One from scratch and the other by applying transfer learning over the pre-trained VGG-16 model. They were trained with the help of the dataset from here. First I pre-processed the image by rescaling It and applying some image augmentations like horizontal flip, rotation by 40 deg, and also resized it to 128x128. And then I split the training dataset into training and validation sets. All this was done with the help of ImageDataGenerator from the keras.preprocessing library.

Then, I First created a simple CNN model with 3 Convolution layers each followed by a max-pooling layer. Each layer had ReLU as its activation function except the output layer which had a sigmoid.

Then I compiled the model with adam as optimizer and binary_crossentropy as the loss function.

For training, I also used callbacks and early stopping as I didn't know how much longer it will take to train my model. With a batch size of 46, the model was trained for 26 epochs and the result was as follows:-

- Training time around 23 minutes with Kaggle's GPU
- Training accuracy is around 79%
- Final validation accuracy is around 77%
- The test accuracy is around 81%.

Then I applied transfer learning. I downloaded the VGG-16 model with imagenet weights and froze all the layers so that they do not learn. Then, I replace all fully connected layers with a dropout layer and then the output layer directly. Using the same callbacks as used for the custom model, I re-trained the model and after just 7 epochs the results were as follows:-

- Training time around 7 minutes with Kaggle's GPU
- Training accuracy is around 85%
- Final validation accuracy is around 84%
- The test accuracy is around 87%.

# *Conclusion*

In this project, two ML models were built to identify an image as a cat or a dog with the help of a convolutional neural network. Both models performed very well, the custom model giving an accuracy of 81 percent and the pre-trained model giving an accuracy of 87 percent on the same test dataset. The main difference was the number of epochs and the time taken for the training. The custom model trained for 26 epochs for 23 minutes while the pre-trained model trained for 7 epochs in 6 minutes and also gave accuracy higher than the custom model. The pre-trained models and the transfer learning concepts are life saviors if one has hardware limitations.