kanaiyalamindiesh ■ 20 hours ago 6 min read

Wild Animal image classifier

Updated: a few seconds ago

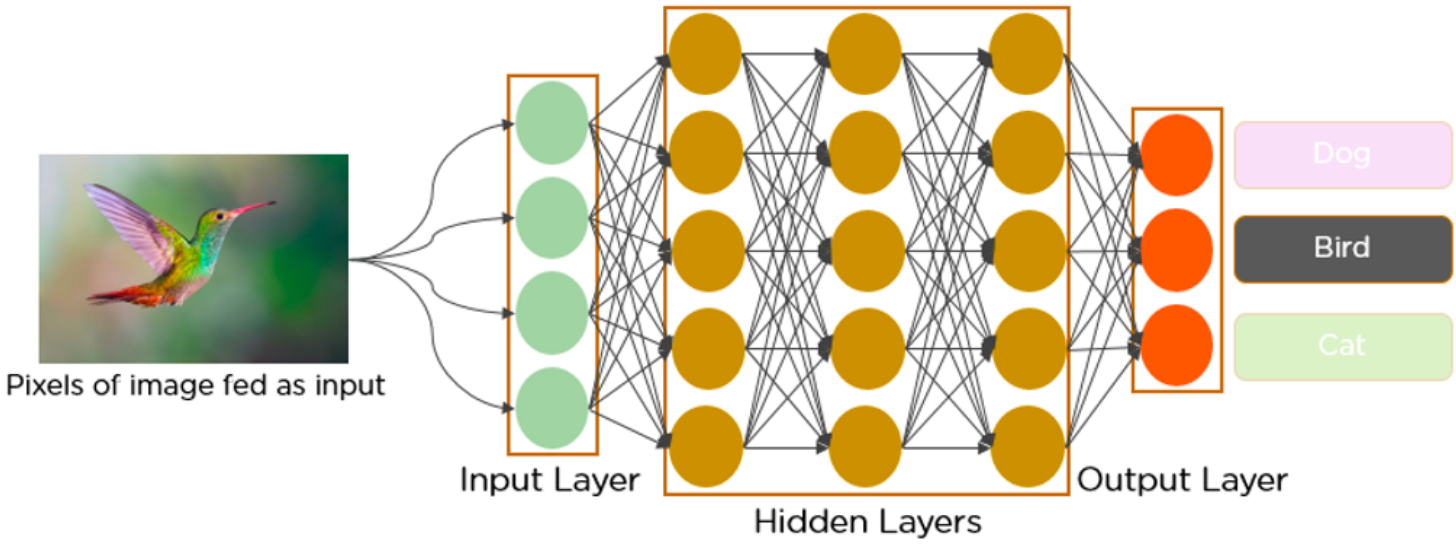
The object oif this blog is to classify images from a multiple class images.

Image Classification :

Image classification refers to the task of extracting information classes from a multiband raster image. The task of classifying and labeling groups of pixels or vectors inside an image based on certain rules is known as image classification. A model is trained to identify various image classes. A CNN model, for example, maybe trained to detect images of multiple different types of animals: tiger, cheetah, lion, wolf etc.
Depending on the interaction between the analyst and the computer during classification, there are two types of classification: supervised and unsupervised.

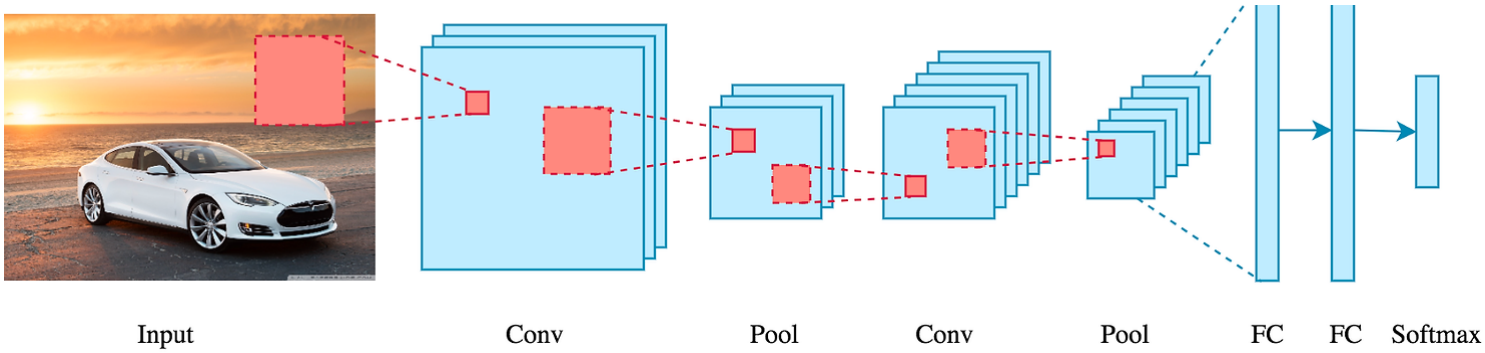
Convolutional Neural Networks:

In last few decades, deep learning has proved to be very important and powerful tool because of its capacity to deal with large datasets.



Reference:Source

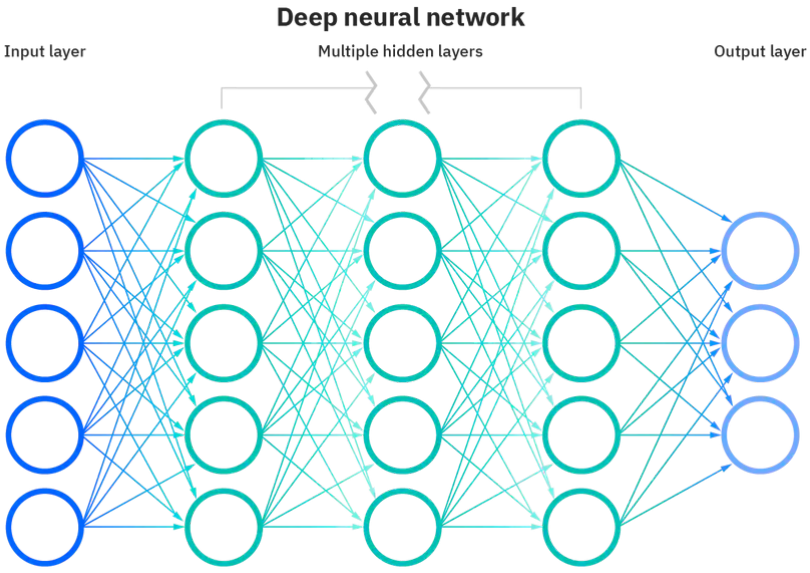
Convolutional Neural Networks are most commonly used for image classification. In early 1980 CNN can be used to detect handwritten words.CNN uses convolution instead of matrix multiplication. Convolution in mathematics is mathematical operation on two functions that produce a third function which express the shape of one is modified by the other.



Reference:Soucre

Neural Networks:

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.
Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.



Reference: Source

Building blocks of CNN:

Convultional neural networks outperforms other neural networks when handling with images, speech or audio signal inputs.

A CNN has the following layers.

Input layers:
This is the layer where input image size to the model is specified.
Activation Function:

The outcome of the neural network is determined by mathematics. This function is applied to each neuron in the network and determines whether or not the neuron should be activated based on the input relevant to the model prediction.
There are different types of activation functions they are ReLu, SoftMax, LeakyReLu etc.

Pooling layer
The pooling layer is used to reduce the number of parameters and computations in the network gradually decreasing the spatial size of the representation. It also aids in the prevention of overfitting.
Pooling methods come in a variety of shapes and sizes. The most crucial are listed below.
1.Max pooling : It is a convolution procedure in which the Kernel or feature detector recovers the maximum value of the area that it convolves.
2.Min pooling: In min pooling, the kernel extracts the minimum value of the area it convolves in the same way it extracts the minimum value of the area it convolves.
3.Average pooling: In average pooling, the kernel extracts the area's average value.
In our models, we are using max-pooling and average pooling operation. Max pooling is also called downsampling.

filters: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
Arguments

- filters: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- kernel_size: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- strides: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.
- padding: one of "valid" or "same" (case-insensitive). "valid" means no padding. "same" results in padding with zeros evenly to the left/right or up/down of the input. When padding="same" and strides=1, the output has the same size as the input.
- data_format: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch_size, height, width, channels) while channels_first corresponds to inputs with shape (batch_size, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be channels_last.

- dilation_rate: an integer or tuple/ list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.
- groups: A positive integer specifying the number of groups in which the input is split along the channel axis. Each group is convolved separately with filters / groups filters. The output is the concatenation of all the groups results along the channel axis. Input channels and filters must both be divisible by groups.
- activation: Activation function to use. If you don't specify anything, no activation is applied (see keras.activations).
- use_bias: Boolean, whether the layer uses a bias vector.
- kernel_initializer: Initializer for the kernel weights matrix (see keras.initializers). Defaults to 'glorot_uniform'.
- bias_initializer: Initializer for the bias vector (see keras.initializers). Defaults to 'zeros'.
- kernel_regularizer: Regularizer function applied to the kernel weights matrix (see keras.regularizers).
- bias_regularizer: Regularizer function applied to the bias vector (see keras.regularizers).
- activity_regularizer: Regularizer function applied to the output of the layer (its "activation") (see keras.regularizers).
- kernel_constraint: Constraint function applied to the kernel matrix (see keras.constraints).
- bias_constraint: Constraint function applied to the bias vector (see keras.constraints).

Input shape

4+D tensor with shape: batch_shape + (channels, rows, cols) if data_format='channels_first' or 4+D tensor with shape: batch_shape + (rows, cols, channels) if data_format='channels_last'.

Output shape

4+D tensor with shape: batch_shape + (filters, new_rows, new_cols) if data_format='channels_first' or 4+D tensor with shape: batch_shape + (new_rows, new_cols, filters) if data_format='channels_last'. rows and cols values might have changed due to padding.

Code Analysis:

Importing libraries required for the model building.

```
import numpy as np
import os
import PIL
import PIL.Image
import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt
import keras
import random
from PIL import Image
from numpy import asarray
```

+ Code + Markdown

Pre- Processing the data : Input is read from the kaggle dataset. The ImageDataGenerator class contains three methods: flow(), flow from directory(), and flow from dataframe(). In this we are using flow_from_directory for this model.

The directory must be set to the path where 'n' classes of folders are present.

- The target_size is the size of your input images, every image will be resized to this size.
- batch_size: No. of images to be yielded from the generator per batch.
- class_mode: Set "binary" if you have only two classes to predict, if not set to"categorical", in case if you're developing an Auto encoder system, both input and the output would probably be the same image, for this case set to "input".
- shuffle: Set True if you want to shuffle the order of the image that is being yielded, else set False.
- seed: Random seed for applying random image augmentation and shuffling the order of the image.

[99]:

```
def get_Img_dir(train_ds_gen, val_ds_gen, directory):
    train_ds = train_ds_gen.flow_from_directory(directory,
                                                target_size=(120,120),
                                                batch_size=18,
                                                subset='training',
                                                class_mode='categorical',
                                                shuffle=True,
                                                seed=223)
    val_ds = val_ds_gen.flow_from_directory(directory,
                                                      target_size=(200,200),
                                                      batch_size=18,
                                                      shuffle=True,
                                                      subset='validation',
                                                      class_mode='categorical',
                                                      seed=223)


    return train_ds, val_ds
```

Image Augmentaion:

ImageDatagenerator class provides a easy and quick way to augment our images.It provides a host of different augmentation techniques like standardization, rotation, shifts, flips, brightness change, and many more.

Data Splitting:
In this model we are splitting data as train data as 80% and Test data as 20%.

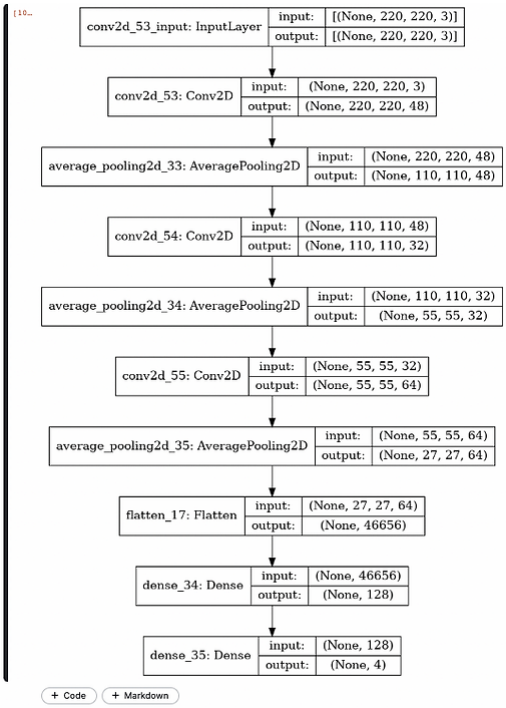
```
train_dsgen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    rotation_range=30,
    horizontal_flip=True,
    vertical_flip=True,
    width_shift_range=0.2,
    height_shift_range=0.2,
    fill_mode='nearest'
    )
val_dsgen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    rotation_range=30,
    horizontal_flip=True,
    vertical_flip=True,
    width_shift_range=0.2,
    height_shift_range=0.2,
    fill_mode='nearest'
    )

train_dataset, val_dataset = get_Img_dir(
            train_ds_gen=train_dsgen,
            val_ds_gen=val_dsgen,
            directory='/kaggle/input/african-wildlife')
```

Model :

CNN model1 is implemented with 3 convolutional layers with 64,32,128 filters and kernel size 3,3,2 respectively. Averagepooling of pool size(1,1).

```
def cnn1(input_shape):

    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Conv2D(64, 3, padding='same', input_shape = input_shape, activati
    model.add(tf.keras.layers.AveragePooling2D(pool_size = (3,3),strides= (1,1)))

    model.add(tf.keras.layers.Conv2D(32, 3, padding='same', input_shape = input_shape, activati
    model.add(tf.keras.layers.AveragePooling2D(pool_size = (3,3),strides= (1,1)))

    model.add(tf.keras.layers.Conv2D(128, 2, padding='same', activation = 'relu'))
    model.add(tf.keras.layers.AveragePooling2D(pool_size = (3,3),strides= (1,1)))

    model.add(tf.keras.layers.Flatten(input_shape = (3 ,2)))

    model.add(tf.keras.layers.Dense(units = 128, activation = 'LeakyReLU'))
    num_classes = 4

    model.add(tf.keras.layers.Dense(units = num_classes, activation = 'softmax'))

    return model

model = cnn1(input_shape=(120,120) + (3,))

tf.keras.utils.plot_model(model, show_shapes=True)
```

Output:



Model fitting:

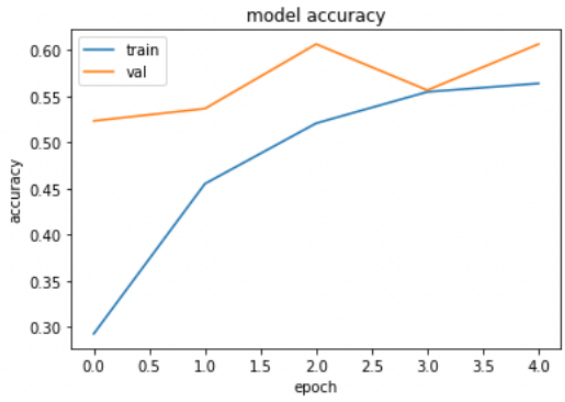
We are fitiing the model with train and validation datasets sonsidering epoch as 5.

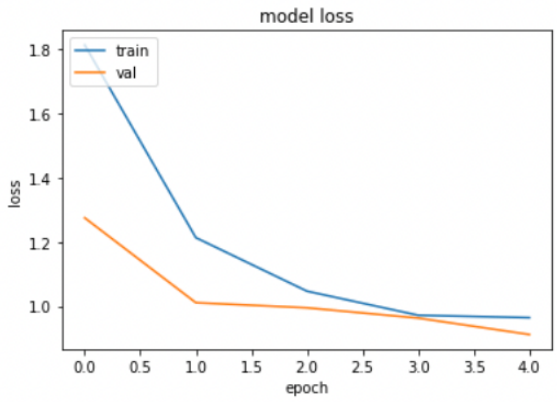
[102]:
```
epochs = 5

model.compile(
    optimizer='adam',
    loss="categorical_crossentropy",
    metrics=["categorical_accuracy"],
)

tbCallBack = keras.callbacks.TensorBoard(log_dir='logs', histogram_freq=0, write_graph=True, write_images=True)

history = model.fit(
    train_dataset, epochs=epochs, callbacks=[tbCallBack],
    validation_data=val_dataset
)
```

```
2022-12-01 03:10:04.182644: I tensorflow/core/profiler/lib/profiler_session.cc:131] Profiler session initializing.
2022-12-01 03:10:04.183043: I tensorflow/core/profiler/lib/profiler_session.cc:146] Profiler session started.
2022-12-01 03:10:04.183466: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler session tear down.
Epoch 1/5
 1/38 [..............................] - ETA: 1:19 - loss: 1.3718 - categorical_accuracy: 0.4000
2022-12-01 03:10:07.889295: I tensorflow/core/profiler/lib/profiler_session.cc:131] Profiler session initializing.
2022-12-01 03:10:07.889833: I tensorflow/core/profiler/lib/profiler_session.cc:146] Profiler session started.
 2/38 [>.............................] - ETA: 1:17 - loss: 4.7872 - categorical_accuracy: 0.3462
2022-12-01 03:10:10.072739: I tensorflow/core/profiler/lib/profiler_session.cc:66] Profiler session collecting data.
2022-12-01 03:10:10.074503: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler session tear down.
2022-12-01 03:10:10.076289: I tensorflow/core/profiler/rpc/client/save_profile.cc:136] Creating directory: ./Graph1/train/plugins/profile/2022_12_01_03_10_10

2022-12-01 03:10:10.077586: I tensorflow/core/profiler/rpc/client/save_profile.cc:142] Dumped gzipped tool data for trace.json.gz to ./Graph1/train/plugins/profile
03_10_10/913ae53e8022.trace.json.gz
2022-12-01 03:10:10.080363: I tensorflow/core/profiler/rpc/client/save_profile.cc:136] Creating directory: ./Graph1/train/plugins/profile/2022_12_01_03_10_10

2022-12-01 03:10:10.081102: I tensorflow/core/profiler/rpc/client/save_profile.cc:142] Dumped gzipped tool data for memory_profile.json.gz to ./Graph1/train/plugin
22_12_01_03_10_10/913ae53e8022.memory_profile.json.gz
2022-12-01 03:10:10.081583: I tensorflow/core/profiler/rpc/client/capture_profile.cc:251] Creating directory: ./Graph1/train/plugins/profile/2022_12_01_03_10_10
Dumped tool data for xplane.pb to ./Graph1/train/plugins/profile/2022_12_01_03_10_10/913ae53e8022.xplane.pb
Dumped tool data for overview_page.pb to ./Graph1/train/plugins/profile/2022_12_01_03_10_10/913ae53e8022.overview_page.pb
Dumped tool data for input_pipeline.pb to ./Graph1/train/plugins/profile/2022_12_01_03_10_10/913ae53e8022.input_pipeline.pb
Dumped tool data for tensorflow_stats.pb to ./Graph1/train/plugins/profile/2022_12_01_03_10_10/913ae53e8022.tensorflow_stats.pb
Dumped tool data for kernel_stats.pb to ./Graph1/train/plugins/profile/2022_12_01_03_10_10/913ae53e8022.kernel_stats.pb

38/38 [==============================] - 70s 2s/step - loss: 1.8130 - categorical_accuracy: 0.2924 - val_loss: 1.2750 - val_categorical_accuracy: 0.5233
Epoch 2/5
38/38 [==============================] - 73s 2s/step - loss: 1.2134 - categorical_accuracy: 0.4551 - val_loss: 1.0112 - val_categorical_accuracy: 0.5367
Epoch 3/5
38/38 [==============================] - 73s 2s/step - loss: 1.0469 - categorical_accuracy: 0.5208 - val_loss: 0.9953 - val_categorical_accuracy: 0.6067
Epoch 4/5
38/38 [==============================] - 68s 2s/step - loss: 0.9721 - categorical_accuracy: 0.5548 - val_loss: 0.9632 - val_categorical_accuracy: 0.5567
Epoch 5/5
38/38 [==============================] - 67s 2s/step - loss: 0.9645 - categorical_accuracy: 0.5640 - val_loss: 0.9122 - val_categorical_accuracy: 0.6067
```

The accuracy of the first model is 6.67 . Plotting model accuracy and model graph.

```
plt.plot(history.history['categorical_accuracy'])
plt.plot(history.history['val_categorical_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()


plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



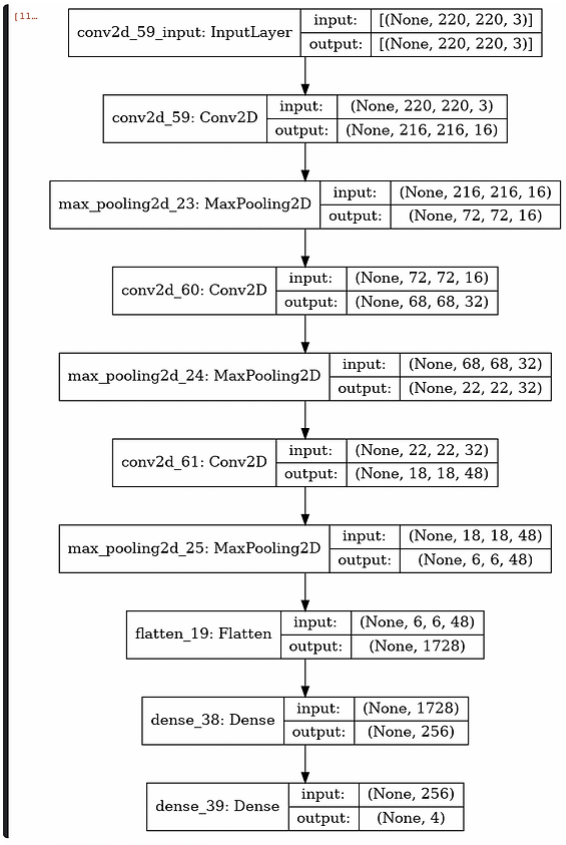


+ Code + Markdown

Model2:

Model 2 is implemented with three different Conv2D layers with 128,64,48 filter and (2,2) as kernal size respectively, by taking MaxPooling2D of pool_size (3,3). and activation function as LeakyReLu.

```python
def cnn2(input_shape):

    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Conv2D(128, 2, padding='same', input_shape = input_shape, activation = 'LeakyReLU'))
    model.add(tf.keras.layers.AveragePooling2D(pool_size = (3,3),strides= (1,1)))

    model.add(tf.keras.layers.Conv2D(64, 3, padding='same', input_shape = input_shape, activation = 'LeakyReLU'))
    model.add(tf.keras.layers.AveragePooling2D(pool_size = (3,3),strides= (1,1)))

    model.add(tf.keras.layers.Conv2D(48, 2, padding='same', activation = 'LeakyReLU'))
    model.add(tf.keras.layers.AveragePooling2D(pool_size = (3,3),strides= (1,1)))

    model.add(tf.keras.layers.Flatten(input_shape = (3 ,4)))

    model.add(tf.keras.layers.Dense(units = 128, activation = 'LeakyReLU'))
    num_classes = 4

    model.add(tf.keras.layers.Dense(units = num_classes, activation = 'softmax'))

    return model

model = cnn1(input_shape=(120,120) + (3,))

tf.keras.utils.plot_model(model, show_shapes=True)
```



Model fitting:

We are fitiing the model with train and validation datasets sonsidering epoch as 5.

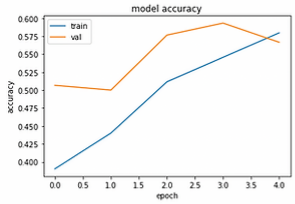
```
epochs = 5

model.compile(
    optimizer='adam',
    loss="categorical_crossentropy",
    metrics=["categorical_accuracy"],
)

tensorBoard_CallBack = keras.callbacks.TensorBoard(log_dir='logs', histogram_freq=0, write_graph=True, write_images=True)

history = model.fit(
    train_dataset, epochs=epochs, callbacks=[tensorBoard_CallBack],
    validation_data=val_dataset
)
```

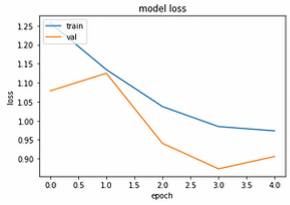
```
2022-12-01 03:18:51.229975: I tensorflow/core/profiler/lib/profiler_session.cc:131] Profiler session initializing.
2022-12-01 03:18:51.230110: I tensorflow/core/profiler/lib/profiler_session.cc:146] Profiler session started.
2022-12-01 03:18:51.230482: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler session tear down.
Epoch 1/5
 1/38 [..............................] - ETA: 1:12 - loss: 1.3806 - categorical_accuracy: 0.3125
2022-12-01 03:18:54.125719: I tensorflow/core/profiler/lib/profiler_session.cc:131] Profiler session initializing.
2022-12-01 03:18:54.126116: I tensorflow/core/profiler/lib/profiler_session.cc:146] Profiler session started.
 2/38 [>.............................] - ETA: 36s - loss: 1.4301 - categorical_accuracy: 0.2344
2022-12-01 03:18:55.269481: I tensorflow/core/profiler/lib/profiler_session.cc:66] Profiler session collecting data.
2022-12-01 03:18:55.271046: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler session tear down.
2022-12-01 03:18:55.272814: I tensorflow/core/profiler/rpc/client/save_profile.cc:136] Creating directory: ./Graph2/train/plugins/profile/2022_12_01_03_18_55

2022-12-01 03:18:55.273957: I tensorflow/core/profiler/rpc/client/save_profile.cc:142] Dumped gzipped tool data for trace.json.gz to ./Graph2/train/plugins/profile/2022_12_01
03_18_55/913ae53e8022.trace.json.gz
2022-12-01 03:18:55.277180: I tensorflow/core/profiler/rpc/client/save_profile.cc:136] Creating directory: ./Graph2/train/plugins/profile/2022_12_01_03_18_55

2022-12-01 03:18:55.278002: I tensorflow/core/profiler/rpc/client/save_profile.cc:142] Dumped gzipped tool data for memory_profile.json.gz to ./Graph2/train/plugins/profile/2
22_12_01_03_18_55/913ae53e8022.memory_profile.json.gz
2022-12-01 03:18:55.278443: I tensorflow/core/profiler/rpc/client/capture_profile.cc:251] Creating directory: ./Graph2/train/plugins/profile/2022_12_01_03_18_55
Dumped tool data for xplane.pb to ./Graph2/train/plugins/profile/2022_12_01_03_18_55/913ae53e8022.xplane.pb
Dumped tool data for overview_page.pb to ./Graph2/train/plugins/profile/2022_12_01_03_18_55/913ae53e8022.overview_page.pb
Dumped tool data for input_pipeline.pb to ./Graph2/train/plugins/profile/2022_12_01_03_18_55/913ae53e8022.input_pipeline.pb
Dumped tool data for tensorflow_stats.pb to ./Graph2/train/plugins/profile/2022_12_01_03_18_55/913ae53e8022.tensorflow_stats.pb
Dumped tool data for kernel_stats.pb to ./Graph2/train/plugins/profile/2022_12_01_03_18_55/913ae53e8022.kernel_stats.pb

38/38 [==============================] - 57s 1s/step - loss: 1.2569 - categorical_accuracy: 0.3904 - val_loss: 1.0784 - val_categorical_accuracy: 0.5067
Epoch 2/5
38/38 [==============================] - 55s 1s/step - loss: 1.1344 - categorical_accuracy: 0.4402 - val_loss: 1.1249 - val_categorical_accuracy: 0.5000
Epoch 3/5
38/38 [==============================] - 55s 1s/step - loss: 1.0374 - categorical_accuracy: 0.5116 - val_loss: 0.9407 - val_categorical_accuracy: 0.5767
Epoch 4/5
38/38 [==============================] - 56s 1s/step - loss: 0.9845 - categorical_accuracy: 0.5457 - val_loss: 0.8736 - val_categorical_accuracy: 0.5933
Epoch 5/5
38/38 [==============================] - 55s 1s/step - loss: 0.9733 - categorical_accuracy: 0.5797 - val_loss: 0.9058 - val_categorical_accuracy: 0.5667
```

+ Code + Markdown

The accuracy of the Model2 is 67% , plotting model accuracy and model graph.

```
plt.plot(history.history['categorical_accuracy'])
plt.plot(history.history['val_categorical_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()


plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```





+ Code + Markdown

We are using model.save() to save the model.

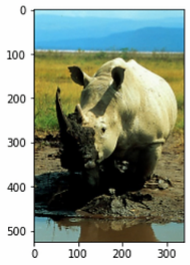
We are randomly taking 4 images and predicting them using the model.

[104]:
```
valid_data = list(zip(val_dataset.filenames,val_dataset.classes))

rand_i = [i for i in range(0,len(valid_data))]

rand_d = []
for i in random.sample(rand_i, 4):
    rand_d.append(valid_data[i])

c_i = train_dataset.class_indices
```

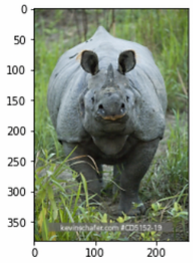
[105]:
```
c_i = {v: k for k, v in c_i.items()}
```

[106]:
```
def img_pred(f_n, model):
    image = Image.open("/kaggle/input/african-wildlife/" +f_n[0]).convert('RGB')
    temp = np.array(image)
    plt.imshow(temp)
    plt.show()
    image = image.resize((220,220))
    image = np.array(image)


    pic = asarray(image)
    pic = pic.astype('float32')

    pic /= 255.0

    uu = np.array([pic])
    yy = model.predict(uu)
    yy = yy * 100
    print("Actual:", c_i[f_n[1]])
    print("Predicted", c_i[np.argmax((yy)[0])])
```

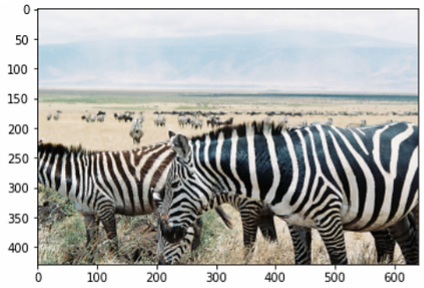
```
for i in r_d:
    img_pred(i, model)
```



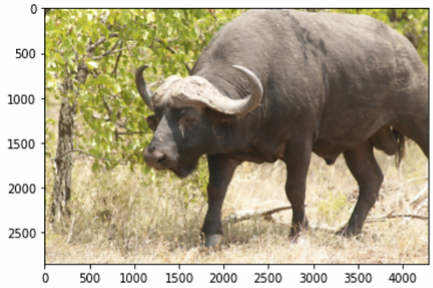
```
Actual: rhino
Predicted elephant
```



```
Actual: rhino
Predicted buffalo
```



```
Actual: zebra
Predicted zebra
```



```
Actual: buffalo
Predicted buffalo
```

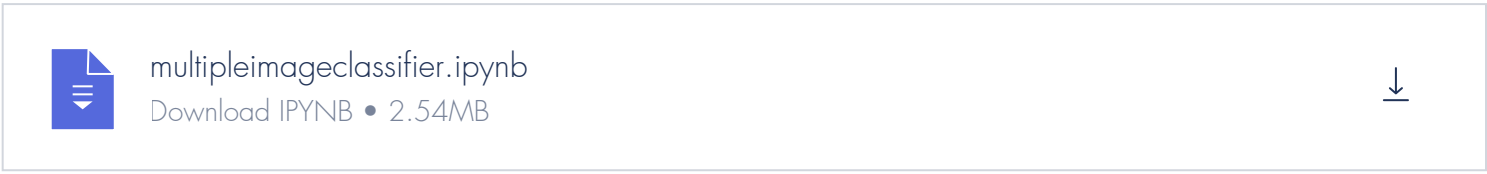
Challenges and Solution:

1. Identifying the right parameters to increase the accuracy of the model was challenging.
2. Reducing overfitting was really challenging and solved this problem by referring to available model on the internet and applying techniques like dropout layer and data augmentation.

Contribution:
I have used Maxpooling for model 1 and Averagepooling for the second model and founf the change in performance.
I have developed this model by referring to the available model and made several experiments by changing the parameters like activation function, kernel size and filters of the CNN model to increase the accuracy of the model.

- Created multiple layers to check the performance, for every model I have increased conv2d layers to check the perfomance of the model.
- Change the hyperparameters- i.e neurons per layer
- experimented on epoch values for 3 models 5,5, 28.
- Used diffferent activation functions like relu, LeakyRelu, softmax.

SourceCode:



Project Demo:
https://www.youtube.com/watch?v=0dI8_kP-xSM&ab_channel=NareshKandhyanam

References:
https://keras.io/api/layers/convolution_layers/convolution2d/
https://medium.com/@likithaveluri1998/natural-images-classification-from-scratch-e120f05691ba
https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/
https://www.ibm.com/cloud/learn/neural-networks
https://github.com/likithaveluri/natural_images_classifiers/blob/main/DM_Final_1002025528.ipynb