A ResNet model on Birds 450 dataset

Group Members: Nitesh Kumar (AP19110010542), Suyash Dayal (AP19110010530), Swikriti Khadke (AP19110010555)

Introduction and Description of implemented CNN Model: In recent years, deep learning has attracted a lot of attention. A class of artificial neural networks that has been a leading technique in computer vision tasks is Convolutional Neural Network. It was developed in object recognition competition in 2012 ((ILSVRC). In several subjects, CNN performed at an expert level which includes medical research. Radiology researchers have, unsurprisingly, become increasingly interested in CNN's potential. It is beings used in the fields like lesion detection, classification, segmentation, image reconstruction, and natural language processing. Knowing this cutting-edge approach would be beneficial for clinical radiologists as well as researchers that use CNN for their work in radiology and medical imaging, as deep learning may soon affect their field of practice. CNN should be used to process data with a grid pattern such as images. CNN was built taking inspiration from the organization of visual cortex of an animal. CNN was built to automatically and adaptively learn spatial hierarchies of features. A standard CNN is made from low to high level patterns, convolution, pooling and fully linked layers are the three types of layers. Convolution and pooling layers in order one and two do feature extraction, whereas a fully connected layer in order three maps the extracted features into the output, such as classification. In CNN, which is made up of a stack of mathematical operations, including convolution, a specialized kind of linear operation, a convolution layer is crucial. Since a feature may appear anywhere in a digital image, the pixel values are stored in a two-dimensional (2D) grid, or array of numbers, and a small grid of parameters known as the kernel, an optimizable feature extractor, is applied at each image position, making CNNs extremely effective for image processing. Extracted features may gradually and hierarchically become more sophisticated as one layer feeds its output into the following layer. Training is the process of minimizing the difference between outputs and ground truth labels using an optimization technique like as backpropagation and gradient descent, among others. It involves improving parameters such as kernels. A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning method that can take in an input image, give various elements and objects in the image importance (learnable weights and biases), and be able to distinguish between them. Comparatively speaking, a ConvNet requires substantially less pre-processing than other classification techniques. ConvNets have the capacity to learn these filters and properties, whereas in primitive techniques filters are hand-engineered.

A ConvNet's architecture was influenced by how the Visual Cortex is organised and is similar to the connectivity network of neurons in the human brain. Only in this constrained area of the visual field, known as the Receptive Field, do individual neurons react to stimuli. The entire visual field is covered by a series of such fields that overlap. The procedure yields two different types of results: one where the dimensionality of the convolved feature is decreased as compared to the input, and the other where it is either increased or stays the same. Applying Valid Padding in the first instance or Same Padding in the second accomplishes this. The Pooling layer, like the Convolutional Layer, is in charge of shrinking the Convolved Feature's spatial size. Through dimensionality reduction, the amount of computing power needed to process the data will be reduced. Furthermore, it aids in properly training the model by allowing the extraction of dominating characteristics that are rotational and positional invariant. Max Pooling and Average Pooling are the two different types of pooling. The maximum value from the area of the image that the Kernel has covered is returned by Max Pooling. The average of all the values from the area of the image covered by the Kernel is what is returned by average pooling, on the other hand. Additionally, Max Pooling functions as a noise suppressant. It also does de-noising and dimensionality reduction in addition to completely discarding the noisy activations. Average Pooling, on the other hand, merely carries out dimensionality reduction as a noise-suppressing strategy. Therefore, we can conclude that Max Pooling outperforms Average Pooling significantly. The i-th layer of a convolutional neural network is made up of the convolutional layer and the pooling layer. The number of these layers may be expanded to capture even more minute details, but doing so will require more computer power depending on how complex the images are. There are numerous CNN architectures that may be used, and these architectures have been essential in creating the algorithms that power and will continue to power AI as a whole in the near future. Below is a list of a few of them:

1. LeNet
2. AlexNet
3. VGGNet
4. GoogLeNet
5. ResNet
6. ZFNet

Shaoqing Ren, Kaiming He, Jian Sun, and Xiangyu Zhang introduced the Residual Network (ResNet), one of the well-known deep learning models, in their study. 2015 saw the publication of "Deep Residual Learning for Image Recognition". One of the most well-liked and effective deep learning models to date is the ResNet model. Instead of learning unreferenced functions, Residual Networks, or ResNets, train residual functions with reference to the layer inputs. Residual nets allow these layers to suit a residual mapping rather than expecting that each few stacked layers directly match a desired underlying mapping. They build networks by piling residual blocks on top of one another; for example, a ResNet-50 uses fifty layers of these blocks.

Dataset Description: The dataset used has been extracted from Kaggle. Here, we have data set of 450 bird species. It has 70,626 training images, 22500 test images(5 images per species) and 2250 validation images(5 images per species). This is a very high quality dataset where there is only one bird in each image and the bird typically takes up at least 50% of the pixels in the image. As a result even a moderately complex model will achieve training and test accuracies in the mid 90% range. All images are 224 X 224 X 3 color images in jpg format. Data set includes a train set, test set and validation set. Each set contains 450 sub directories, one for each bird species.

Code and Output:

```
import os
import tensorflow as tf
import glob
import pathlib
from tensorflow import keras
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.models import Model
from tensorflow.keras.applications import ResNet101V2
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.layers import Dense,Conv2D,Flatten,MaxPool2D,Dropout,BatchNormalization,Activation
```

```
train="D:\\ml\\PSC Project\\train"
test="D:\\ml\\PSC Project\\test"
valid="D:\\ml\\PSC Project\\valid"
```

```
len(os.listdir(train))
```

```
4
```

```
len(os.listdir(test))
```

```
4
```

```
len(os.listdir(valid))
```

```
4
```

```
def process(data):
    path=pathlib.Path(data)#converting the dtring to path
    filepaths=list(path.glob(r"*/*.jpg"))#Going through all the subpaths
    labels=list(map(lambda x: os.path.split(os.path.split(x)[0])[1],filepaths))#Separating the label from filepath and storing it
    df1=pd.Series(filepaths,name='filepaths').astype(str)
    df2=pd.Series(labels,name='labels')
    df=pd.concat([df1,df2],axis=1)#Making the dataframe
    return df
```

```
df_train=process(train)
df_test=process(test)
df_valid=process(valid)
```

```
df_train.head()
```

	filepaths	labels
0	D:\ml\PSC Project\train\ABBOTTS BABBLER\001.jpg	ABBOTTS BABBLER
1	D:\ml\PSC Project\train\ABBOTTS BABBLER\002.jpg	ABBOTTS BABBLER
2	D:\ml\PSC Project\train\ABBOTTS BABBLER\003.jpg	ABBOTTS BABBLER
3	D:\ml\PSC Project\train\ABBOTTS BABBLER\004.jpg	ABBOTTS BABBLER
4	D:\ml\PSC Project\train\ABBOTTS BABBLER\005.jpg	ABBOTTS BABBLER

Data Visualization

```
df_train=df_train.sample(frac=1).reset_index(drop=True)#Shuffling the dataframe so we can get random bird pictures
fig,axes=plt.subplots(nrows=6,ncols=4,figsize=(12,12))

for i,ax in enumerate(axes.flat):
    x=plt.imread(df_train['filepaths'][i])#reading the image
    ax.imshow(x)
    ax.set_title(df_train['labels'][i])
plt.tight_layout()
plt.show()
```

Data Augmentation

```
train_generator=ImageDataGenerator(rescale=1/4)
test_generator=ImageDataGenerator(rescale=1/4)
valid_generator=ImageDataGenerator(rescale=1/4)
```

Reading the images from dataframe

```
train_image=train_generator.flow_from_dataframe(dataframe=df_train,
                                                x_col='filepaths',
                                                y_col='labels',
                                                target_size=(224,224),
                                                color_mode='rgb',
                                                class_mode='categorical',
                                                batch_size=256,
                                                subset='training',
                                                random_seed=42)
```

```
test_image = test_generator.flow_from_dataframe(
    dataframe=df_test,
    x_col='filepaths',
    y_col='labels',
    target_size=(224,224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=256
)
```

```
valid_image = test_generator.flow_from_dataframe(
    dataframe=df_valid,
    x_col='filepaths',
    y_col='labels',
    subset='training',
    target_size=(224,224),
    color_mode='rgb',
    class_mode='categorical',
```

```
    batch_size=256
)
```

```
Found 662 validated image filenames belonging to 4 classes.
Found 20 validated image filenames belonging to 4 classes.
Found 20 validated image filenames belonging to 4 classes.
```

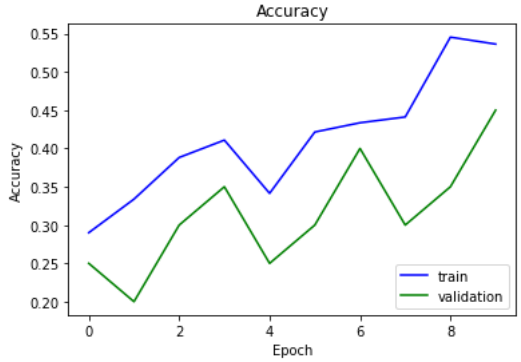
Model building

```
pretrained_model = ResNet101V2(
    input_shape=(224,224, 3),
    include_top=False,
    weights='imagenet',
    pooling='avg'
)
pretrained_model.trainable = False #We don't want to train again th resnet


inputs = pretrained_model.input

x = Dense(120, activation='relu')(pretrained_model.output)
x = Dense(120, activation='relu')(x)#adding some custom layers of our coice

outputs = Dense(4, activation='softmax')(x)
#output choice
model = Model(inputs=inputs, outputs=outputs)


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


mo_fit=model.fit(train_image,validation_data=valid_image,epochs=10)
```

```
Epoch 1/10
3/3 [==============================] - 67s 22s/step - loss: 7.8820 - accuracy: 0.2900 - val_loss: 11.1289 - val_accuracy: 0.2500
Epoch 2/10
3/3 [==============================] - 74s 24s/step - loss: 10.9657 - accuracy: 0.3338 - val_loss: 11.6077 - val_accuracy: 0.2000
Epoch 3/10
3/3 [==============================] - 76s 24s/step - loss: 9.6110 - accuracy: 0.3882 - val_loss: 6.8647 - val_accuracy: 0.3000
Epoch 4/10
3/3 [==============================] - 72s 22s/step - loss: 4.7745 - accuracy: 0.4109 - val_loss: 3.9740 - val_accuracy: 0.3500
Epoch 5/10
3/3 [==============================] - 71s 22s/step - loss: 3.2850 - accuracy: 0.3414 - val_loss: 3.8496 - val_accuracy: 0.2500
Epoch 6/10
3/3 [==============================] - 72s 28s/step - loss: 2.4914 - accuracy: 0.4215 - val_loss: 4.5166 - val_accuracy: 0.3000
Epoch 7/10
3/3 [==============================] - 73s 23s/step - loss: 2.6494 - accuracy: 0.4335 - val_loss: 2.8035 - val_accuracy: 0.4000
Epoch 8/10
3/3 [==============================] - 72s 22s/step - loss: 1.9710 - accuracy: 0.4411 - val_loss: 2.4030 - val_accuracy: 0.3000
Epoch 9/10
3/3 [==============================] - 71s 28s/step - loss: 1.4636 - accuracy: 0.5453 - val_loss: 2.3165 - val_accuracy: 0.3500
Epoch 10/10
3/3 [==============================] - 72s 28s/step - loss: 1.5935 - accuracy: 0.5363 - val_loss: 1.5437 - val_accuracy: 0.4500
```

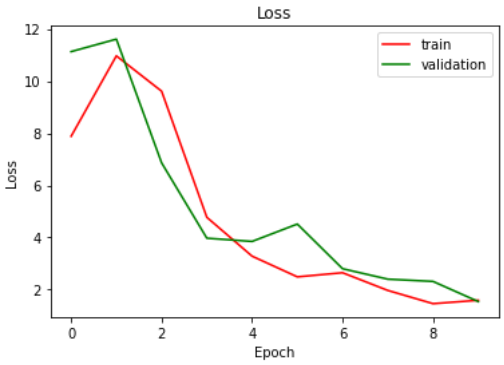
```
plt.plot(mo_fit.history['accuracy'],c='blue')
plt.plot(mo_fit.history['val_accuracy'],c='green')
plt.title('Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train','validation'],loc='lower right')
```

```
<matplotlib.legend.Legend at 0x1e08afd6e00>
```



```
plt.plot(mo_fit.history['loss'],c='red')
plt.plot(mo_fit.history['val_loss'],c='green')
plt.title('Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train','validation'],loc='upper right')
```

```
<matplotlib.legend.Legend at 0x1e08b05fa60>
```




```
results = model.evaluate(train_image, verbose=0)

print("train Loss: {:.5f}".format(results[0]))
print("train Accuracy: {:.2f}%".format(results[1] * 100))
```

```
train Loss: 1.09393
train Accuracy: 60.27%
```

```
results = model.evaluate(valid_image, verbose=0)

print("valid Loss: {:.5f}".format(results[0]))
print("valid Accuracy: {:.2f}%".format(results[1] * 100))
```

```
valid Loss: 1.54374
valid Accuracy: 45.00%
```

```
results = model.evaluate(test_image, verbose=0)

print("Test Loss: {:.5f}".format(results[0]))
print("Test Accuracy: {:.2f}%".format(results[1] * 100))
```

```
Test Loss: 1.51095
Test Accuracy: 35.00%
```

References:

1. https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9
2. https://arxiv.org/abs/1511.08458
3. https://medium.com/appyhigh-technology-blog/convolutional-neural-networks-a-brief-history- of-their-evolution/
4. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the- eli5-way-3bd2b1164a53
5. https://content.iospress.com/articles/journal-of-intelligent-and-fuzzy-systems/ifs210925
6. https://firiuza.medium.com/optimizers-for-training-neural-networks-e0196662e21e
7. https://myrtle.ai/learn/how-to-train-your-resnet-5-hyperparameters/
8. https://towardsdatascience.com/improve-your-model-performance-with-bayesian- optimization-hyperparameter-tuning-4dbd7fe25b62
9. https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models- python-keras/