MACHINE LEARNING ENGINEER NANODEGREE

Capstone Project(iWildcam 2019)

Aarthi Rajagopal

June 4, 2019.

1. Definition:

Project Overview:

Wild cams are used to collect large quantities of image data automatically. These wild cams / camera traps are used by the biologists all over the world to monitor the biodiversity and population density of animal species. The recent strides towards automating the species classification challenge in wild cams when expanded the scope, an interesting problem is encountered i.e., how do you classify a species in a new region that you may not have seen in previous training data?

To solve this, the training data and test data are prepared from different regions, namely the American Southwest and the American Northwest. The species seen in each region overlap, but aren't identical, and the challenge is to classify the test species correctly. To this end, we will allow training on our American Southwest data (from CaltechCameraTraps), on iNaturalist 2017/2018 data, and on simulated data generated from Microsoft AirSim. We have provided a taxonomy file mapping our classes into the iNat taxonomy.

To get more detailed understanding of what wildcams/camera traps are, try visiting the link provided below.
https://zslpublications.onlinelibrary.wiley.com/doi/pdf/10.1002/rse2.48

The images obtained from the wildcams are as follows,




Problem Statement:

Now, to classify the species correctly, a machine learning algorithm is used. The model is built in such a way that it is able to produce a good accuracy in classifying the image. I would consider implementing a convolutional neural network model than the multilayer perceptron as CNN works better in image classification and recognition. A pre-trained network is used to train the model.

So, while predicting the category of the species, the following strategy can be implemented.

1. Initially, the train, test data and images are loaded into the system. Preprocessing is done to reduce the image size as well as data cleaning.
2. Now, the saved reduced images are loaded and then scaled.
3. A CNN classifier model is created and then is trained.
4. A pretrained Densenet model is used in Keras.
5. A model check point function is created to calculate the accuracy.
6. The model is now validated by plotting the test vs train.

Evaluation metrics:

The evaluation criteria would be mainly on the Mean F (F1) score as mentioned in the Kaggle competition.

The traditional F-score or the balanced F-score is the harmonic mean of precision and recall.

F1 score is preferred than accuracy because with unevenly distributed data, f1 does a better job.

Considering the confusion matrix of positive and negative,

$$Accuracy = \frac{True\ positive + True\ negative}{True\ negative\ + False\ negative + True\ positive + False\ positive}$$

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$= \frac{True\ Positive}{Total\ Predicted\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

$$\text{F1} = 2\ x\ \frac{Precision * Recall}{Precision + Recall}$$

2. Analysis:

Data Exploration:

As provided by the Kaggle, the training set contains 196,157 images from 138 different locations in Southern California. The test set contains 153,730 images from 100 loactions in Idaho. The training data can also be obtained from iNaturalist 2017, iNaturalist 2018, iNaturalist 2019 and the images simulated with Microsoft AirSim.

The test set contains 153,730 images from 100 locations in Idaho.
The competition task is to label each image with one of the following label ids:

name, id

empty, 0

deer, 1

moose, 2

squirrel, 3

rodent, 4

small_mammal, 5

elk, 6

pronghorn_antelope, 7

rabbit, 8

bighorn_sheep, 9

fox, 10

coyote, 11

black_bear, 12

raccoon, 13

skunk, 14

wolf, 15

bobcat, 16

cat, 17

dog, 18

opossum, 19

bison, 20

mountain_goat, 21

mountain_lion, 22

Some classes may not occur in train or test.

The following are provided in the train and test data,

1. Category_id
2. Date_captured

3. File_name
4. Frame_num
5. Id
6. Location
7. rights_holder
8. seq_id
9. seq_num_frames
10. width
11. Height

Below is the link where the data can be got,
https://www.kaggle.com/c/iwildcam-2019-fgvc6/data

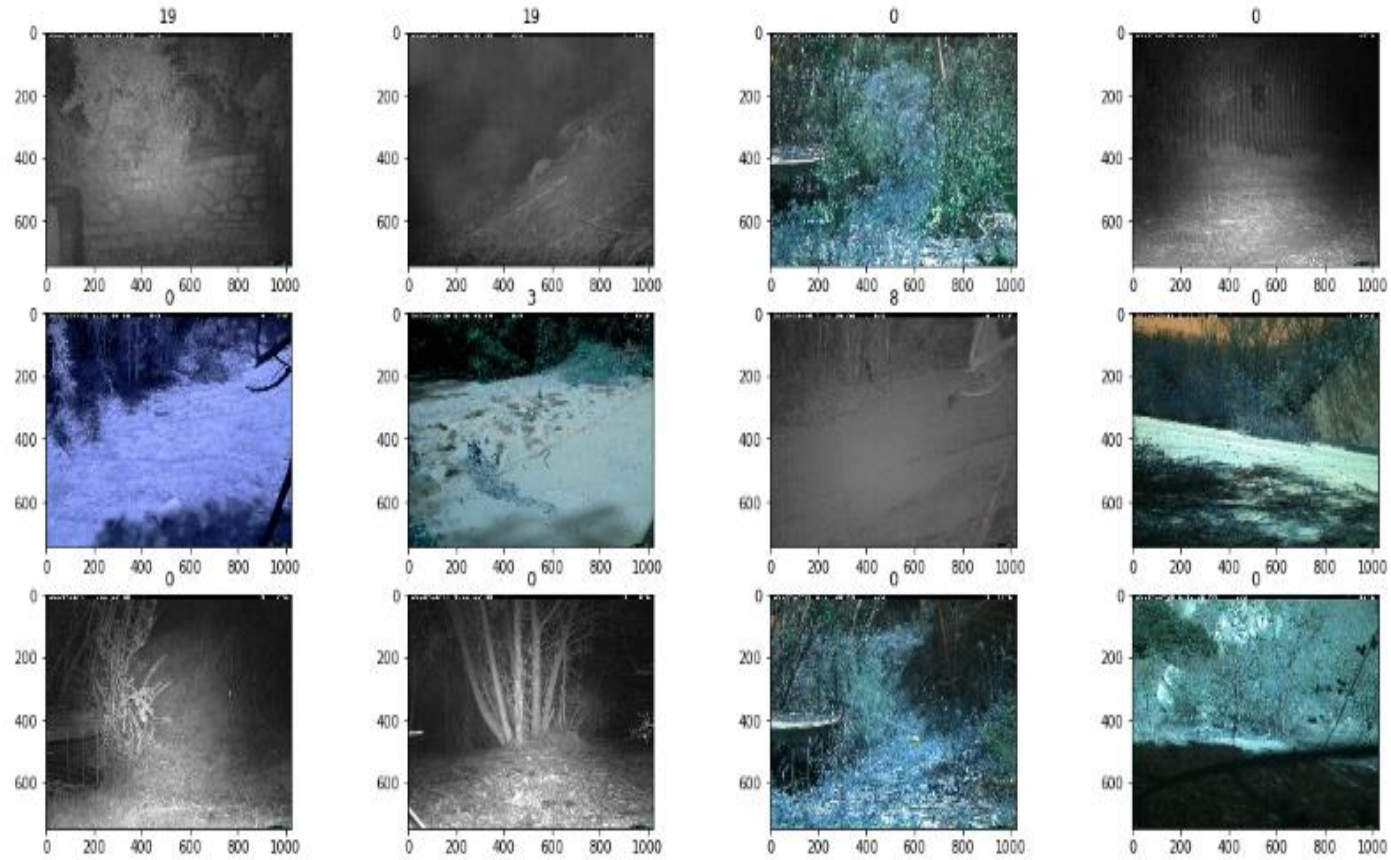
Files as described in Kaggle competition,

1. test.csv
2. train.csv
3. sample_submission.csv
4. test_images.zip
5. train_images.zip

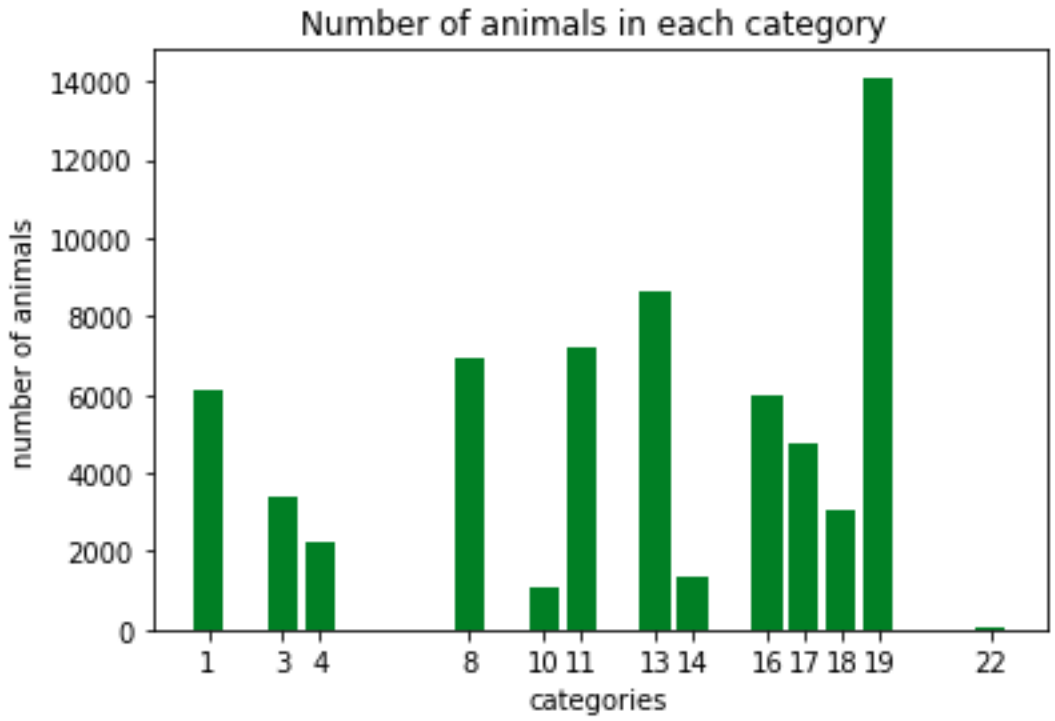
	category_id	date_captured	file_name	frame_num	id	location	rights_holder	seq_id	seq_num_fram
0	19	2011-05-13 23:43:18	5998cfa4-23d2-11e8-a6a3-ec086b02610b.jpg	1	5998cfa4-23d2-11e8-a6a3-ec086b02610b	33	Justin Brown	6f084ccc-5567-11e8-bc84-dca9047ef277	
1	19	2012-03-17 03:48:44	588a679f-23d2-11e8-a6a3-ec086b02610b.jpg	2	588a679f-23d2-11e8-a6a3-ec086b02610b	115	Justin Brown	6f12067d-5567-11e8-b3c0-dca9047ef277	
2	0	2014-05-11 11:56:46	59279ce3-23d2-11e8-a6a3-ec086b02610b.jpg	1	59279ce3-23d2-11e8-a6a3-ec086b02610b	96	Erin Boydston	6faa92d1-5567-11e8-b1ae-dca9047ef277	
3	0	2013-10-06 02:00:00	5a2af4ab-23d2-11e8-a6a3-ec086b02610b.jpg	1	5a2af4ab-23d2-11e8-a6a3-ec086b02610b	57	Erin Boydston	6f7d4702-5567-11e8-9e03-dca9047ef277	
4	0	2011-07-12 13:11:16	599fbd89-23d2-11e8-a6a3-ec086b02610b.jpg	3	599fbd89-23d2-11e8-a6a3-ec086b02610b	46	Justin Brown	6f1728a1-5567-11e8-9be7-dca9047ef277	

Exploratory visualization:

The original image as given without reducing the size is as follows,



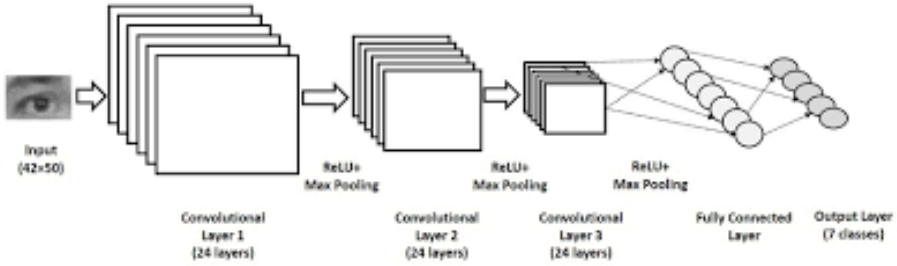
As our plot is to categorize the animal, the plot defining the number of categories is as follows,



From the graph, we can infer that , opossum is the highest number of category in the dataset and mountain lion the least . This implies the model can learn more about the highest category of animals by extracting its features and able to predict well in these cases. But as the data is unevenly distributed, there is a risk of model getting biased.

Algorithms and Techniques:

A convolutional neural network also known as CNN or comp net is a neural network that is so far been most popularly used for analyzing images. Although image analysis has been the most widespread use of CNN's they can also be used for other data analysis or classification problems as well most generally we can think of a CNN as an artificial neural network that has some type of specialization for being able to pick out or detect patterns and make sense of them. This pattern detection is what makes CNN so useful for image analysis so if a CNN is just some form of an artificial neural network what differentiates it from just a standard multi-layer perceptron or MLP well a CNN has hidden layers called convolutional layers and these layers are precisely what makes a CNN.

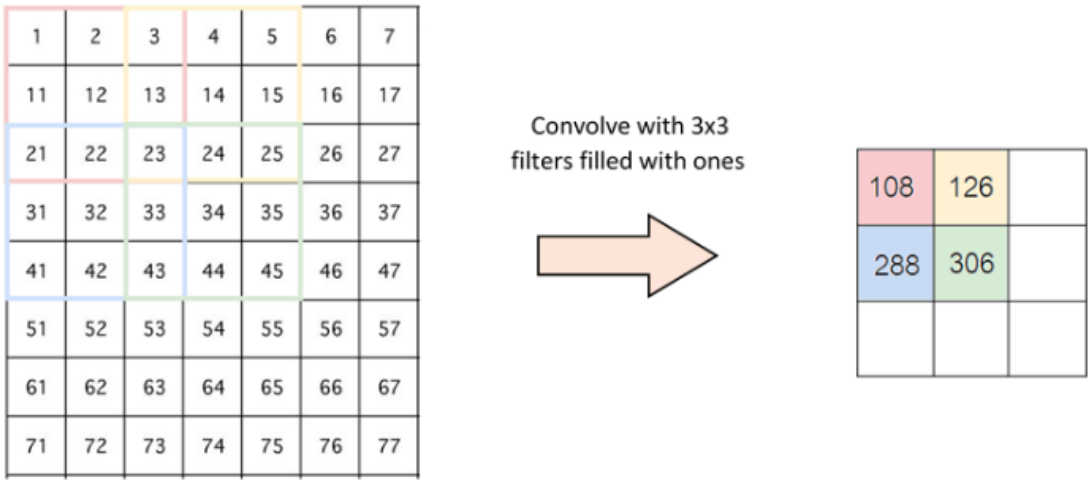


Basic architecture:

1. Initially, the input image is fed into the convolutional layer. It is the first layer in a CNN that takes the features from the image. It learns by considering the image as matrix inputs.
2. Appropriate parameters are selected according to the requirement by applying strides with filter and padding is also done for the image to perfectly fit.
3. Now, Relu (Rectified linear unit) activation function is given to apply non-linearity in the network.
4. Pooling is done to reduce the size of the image.
5. Now, more convolutional layers are added according to the need.
6. The obtained output is flattened and then given as input to fully connected layer.
7. The final result is then obtained by means of logistic regression and then the image is classified.

Strides:

It is the way by which pixels can be shifted over the input matrix. To be clear, if stride is given as 1, then the filters are moved to 1 pixel at a time.



Padding:

It is done to perfectly fit the image. This includes 2 types,

1. When the size needs to be increased so that it fits well, zero padding can be done.
2. When only few part of the input is required , the parts other than the valid ones can be dropped. This is sometimes called as valid padding.

Relu Activation function:

It is a non-linear activation function. The output is f(x) = max(0,x)
It is a piecewise activation function that outputs the input if it is positive else returns zero. It is most often used because of its good efficiency as well as easy to train the data.

Pooling:

Pooling is done to keep the important features in the data by reducing the other parameters when the image is too large. Spatial pooling reduces the dimension of the image but keeps the important information. It includes,

1. Max Pooling
2. Average Pooling
3. Sum Pooling

Max pooling chooses the largest element from the map. This also means that it takes the average pooling too. Sum Pooling is the sum of all elements in the feature map.

Dropout layer:

It is a way to prevent overfitting the data. As the name indicates, the neurons are dropped out randomly in the neural network. This indicates, the neurons in the downstream are removed on forward pass and the weight updates are also not passed.

Transfer learning:

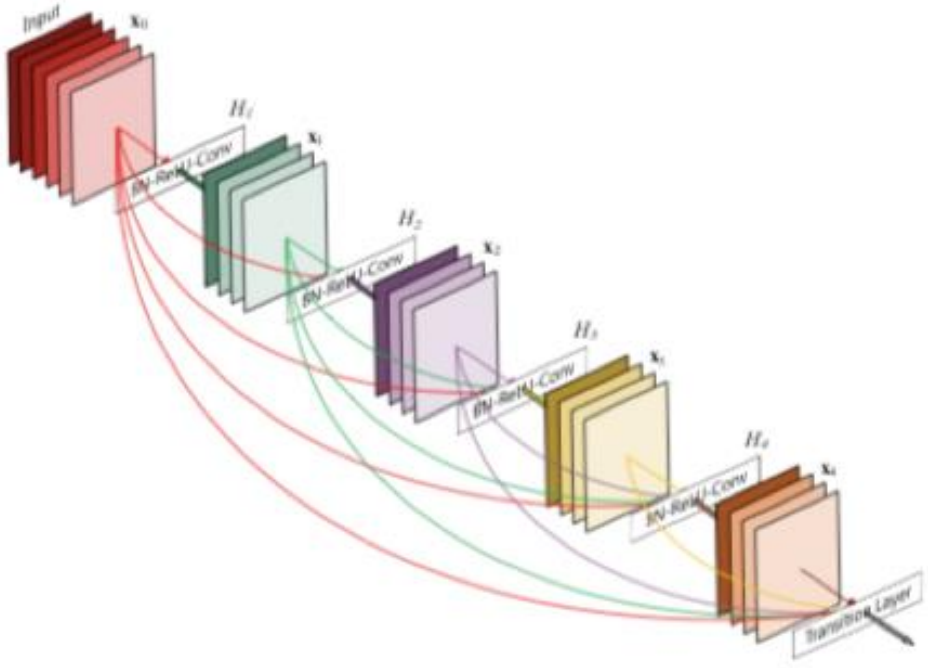
It is the technique by which pre trained network models are used. These networks are already trained well on a large dataset. Hence, utilizing this model and fine tuning them according to our need is all that is done.

In this project, I have used DenseNet121 model.

Densenet model:

In densenet, each layer takes additional inputs from all the before layers and then complements by adding its own feature map and passes it to the subsequent layers. By this way, each layers gets knowledge about all the other layers. The network looks thinner and compact. Hence, the number of channels can be less.

This enables to achieve higher computational efficiency as well as memory efficiency.



Benchmark model:

A simple CNN classifier network is taken as the bench mark model. The model that is used is as follows,

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 30, 30, 1)         28
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 1)         10
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 1)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 196)               0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               25216
_________________________________________________________________
dense_7 (Dense)              (None, 14)                1806
=================================================================
Total params: 27,060
Trainable params: 27,060
Non-trainable params: 0
_________________________________________________________________
```

This produced an accuracy of 65 percentage.

3.Methodology:

Data Preprocessing:

The steps that are included in data preprocessing are as follows,

1. All the necessary libraries are imported.
2. The train and test data as obtained from Kaggle is loaded and viewed.
3. Now, the images as given by Kaggle is viewed by calling a function.
4. A visualization plot is created for category id against the value count.
5. A function is written to adjust and reduce the size of the image.
6. Now, the resize function is called and then the shape of the resized images is seen.
7. All the resized files are now saved and used for later process.

Implementation:

1. The reduced files are loaded by means of numpy.
2. The shape of the data is viewed.
3. The test and train data are then converted by means of float 32.
4. These data are scaled to avoid over and under fitting.
5. Now, the Keras library is imported .
6. A pretrained Densenet model is called from keras application.
7. DenseNet121 with input shape (32,32,3) is loaded.
8. A sequential model is created and the dense network is added as layer.
9. Global Average Pooling of 2 dimension is done
10. A dropout layer of 0.5 is given to avoid overfitting.
11. A dense layer with output of 14 nodes is assigned.
12. Soft max activation function is used to give the probability of each output class.
13. The model is compiled with loss as categorical, adam optimizer is used for better accuracy.

14. A call back function is created in order to compute recall, accuracy and precision.
15. A Model Check point is done to define where to checkpoint the model weights.
16. The model is now fitted with the X_ train and Y_train at a validation split of 0.2 .
17. Epochs = 10 for better training the model.
18. Now, the loss of training and validation is plotted
19. Similarly, the accuracy of training and validation is plotted.
20. The submission file is loaded and then the category id is predicted.
21. Finally, it is saved into a csv file.

Challenges faced:

1. As the dataset is too large, downloading it and retrieving took a lot of time. Hence, finally I opted the Kaggle kernel.
2. Initially, I tried training in my local computer and it took long hours to completed. Hence, I used the GPU available on Kaggle.
3. There occurred a trouble while calling the DenseNet from Keras Application. Later, I figured out to switch on the internet setting in the kernel which is defaultly off .

Refinement:

I created two CNN models.

1. A basic CNN model
2. Densenet model.

Basic CNN model:

As mentioned in the benchmark, the above is the layer and model I used as a basic one.
It gave an accuracy of 65 percentage.

```
val_f1: 0.0577 - val_precision: 0.0484 - val_recall: 0.0714
Epoch 6/10
157039/157039 [==============================] - 17s 106us/step -
ss: 2.6391 - val_acc: 0.6772
val_f1: 0.0577 - val_precision: 0.0484 - val_recall: 0.0714
Epoch 7/10
157039/157039 [==============================] - 16s 105us/step -
ss: 2.6391 - val_acc: 0.6772
val_f1: 0.0577 - val_precision: 0.0484 - val_recall: 0.0714
Epoch 8/10
157039/157039 [==============================] - 16s 105us/step -
ss: 2.6391 - val_acc: 0.6772
val_f1: 0.0577 - val_precision: 0.0484 - val_recall: 0.0714
Epoch 9/10
157039/157039 [==============================] - 17s 106us/step -
ss: 2.6391 - val_acc: 0.6772
val_f1: 0.0577 - val_precision: 0.0484 - val_recall: 0.0714
Epoch 10/10
157039/157039 [==============================] - 17s 106us/step -
ss: 2.6391 - val_acc: 0.6772
val_f1: 0.0577 - val_precision: 0.0484 - val_recall: 0.0714
```

DenseNet Model:

The accuracy when used Densenet model is 90 percentage.

```
Epoch 7/10
157039/157039 [==============================] - 341s 2ms/step - lo
ss: 0.3386 - val_acc: 0.8872
val_f1: 0.6949 - val_precision: 0.7370 - val_recall: 0.6905
Epoch 8/10
157039/157039 [==============================] - 344s 2ms/step - lo
ss: 0.3407 - val_acc: 0.8847
val_f1: 0.7109 - val_precision: 0.7847 - val_recall: 0.6715
Epoch 9/10
157039/157039 [==============================] - 342s 2ms/step - lo
ss: 0.3119 - val_acc: 0.8980
val_f1: 0.7107 - val_precision: 0.7292 - val_recall: 0.7065
Epoch 10/10
157039/157039 [==============================] - 341s 2ms/step - lo
ss: 0.3023 - val_acc: 0.9011
val_f1: 0.7235 - val_precision: 0.7934 - val_recall: 0.6954
```

4. Results:

Evaluation and Validation:

The final model's summary is as follows,

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/keras/backen
45: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is d
ved in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet121 (Model)          (None, 1, 1, 1024)        7037504
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1024)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 14)                14350
=================================================================
Total params: 7,051,854
Trainable params: 6,968,206
Non-trainable params: 83,648
_________________________________________________________________
```

I have used the already trained imagenet weights for training the data.

The global average pooling is done to reduce overfitting i.e. GAP layers reduce the spatial dimensions.
Dropout layer is added to minimize the amount of randomly selected neurons.
This DenseNet model produced an accuracy of 90%

Robustness:

I initially trained by implementing my own model adding convolutional layers. The accuracy dropped. Later, I went for pre trained model using transfer learning where I used densenet to train the model and I could find my accuracy soared up to 90 percentage. To improve the robustness of the model, I added random noise to the

training data. My accuracy dropped to 84 percentage. So, I decided to stick with training without noise.

To test it, I took a image and tried predicting the category with and without adding random noise to it. I can see my model worked well when there is no noise.

In[111]:

```
img_n= random_noise(img)
result = model.predict(img_n)
result = result.argmax(axis=1)
print(result)
```

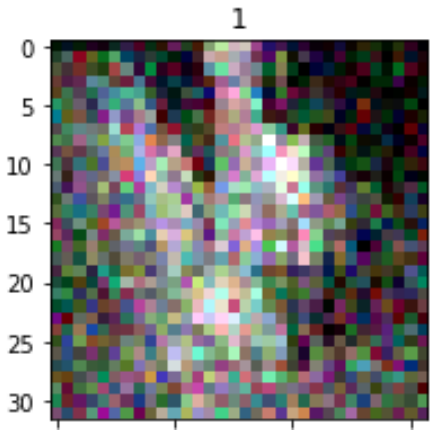
```
[1]
```

+ Code + Markdown

```
fig=plt.figure(figsize=(3,5 ))
plt.title(result[0])
plt.imshow(random_noise(img2))
```

Out[120]:

```
<matplotlib.image.AxesImage at 0x7f16d5dc7438>
```



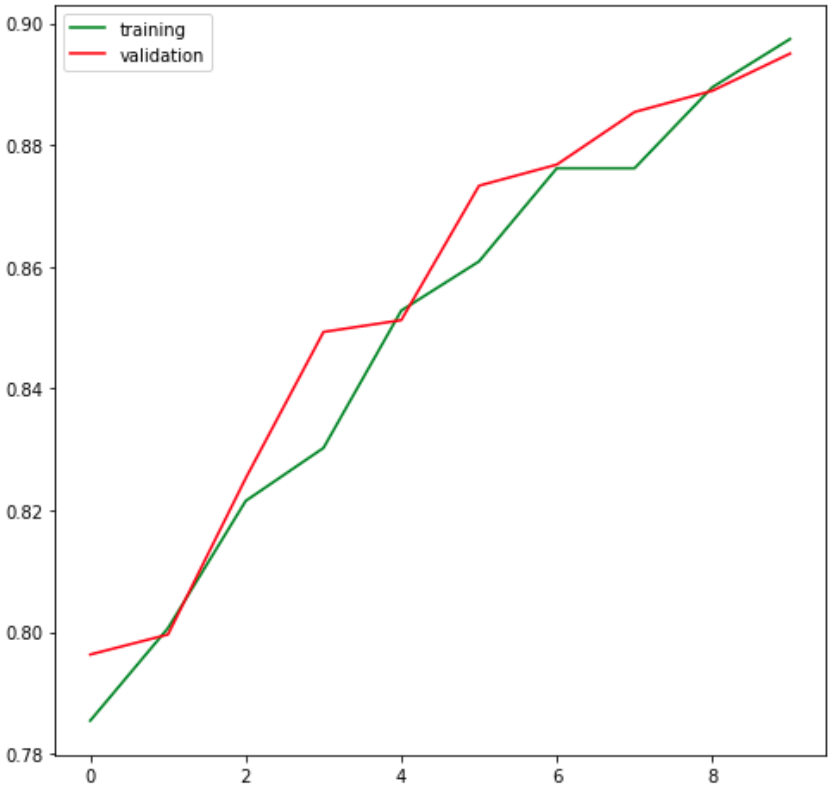
Justification:
As mentioned before, the densenet model gave a better accuracy than a basic convolutional neural network.

This is because,

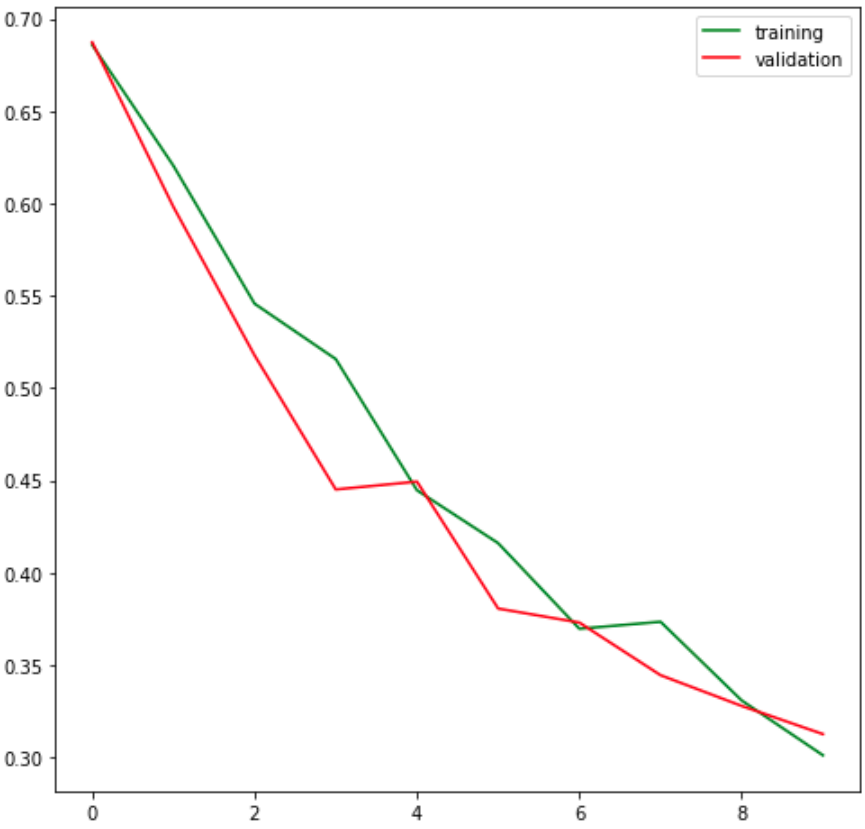
1. The Dense Net model is already trained with large dataset and hold the weight accordingly. It includes dense blocks and pooling operations.
2. These small applications produces parameter efficiency, implicit deep supervision and makes it a good fit for semantic segmentation.

The following are the visualizations of training and validation loss and accuracy.

Accuracy:



Loss:



5. Conclusion:

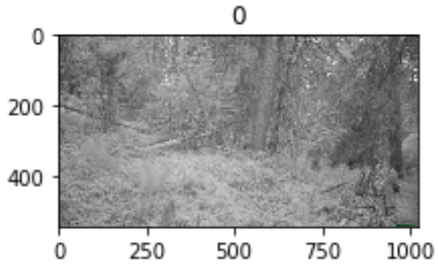
Free form visualization:

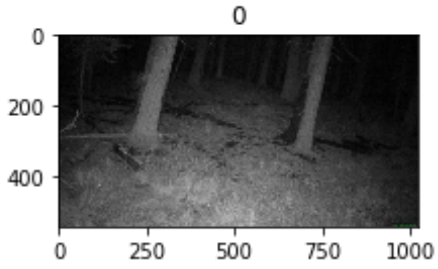
The following is the images with the predicted ids.

	Id	Predicted
0	b005e5b2-2c0b-11e9-bcad-06f10d5896c4	0
1	f2347cfe-2c11-11e9-bcad-06f10d5896c4	1
2	27cf8d26-2c0e-11e9-bcad-06f10d5896c4	0
3	f82f52c7-2c1d-11e9-bcad-06f10d5896c4	0
4	e133f50d-2c1c-11e9-bcad-06f10d5896c4	0

The following shows the image predicted with category id displayed at the top.

```
for i in range(5):
    fig=plt.figure(figsize=(11,8 ))
    img = cv2.imread(f'../input/iwildcam-2019-fgvc6/test_images/'+ submission_df.loc[i][
    fig.add_subplot(4, 2, 2)
    plt.title(submission_df.loc[i][1])
    plt.imshow(img)
```





In[99]:

```
fig=plt.figure(figsize=(11,8 ))
img = cv2.imread(f'../input/imagetotest/SM-wolverine-wildcam-1.jpg')
plt.title(result[0])
plt.imshow(img)
```

ut[99]:

```
<matplotlib.figure.AxesImage at 0x7f16d618cf98>
```

+ Code + Markdown

In[111]:

Reflection:

The summary of the whole process is as follows,

1. Get the data available on the Kaggle.
2. Import the libraries necessary.
3. Explore the data and do image size reduction process.
4. Preprocess the data by scaling and fitting.
5. Import the dense net model from keras application.
6. Design the model by adding convolutional layers.
7. Compile the model
8. Perform the call back function
9. Fit the model

10. Plot the training and validation set to view the accuracy and loss.
11. Now, predict the submission file to view the result.
12. Save the model.

I found training the model little challenging with imagenet weights. And also working with real world data was so much interesting. I am happy with the classification result my model showed up.

Improvements:

The following improvements can be considered,

1. Trying different pre-trained models like VGG16, ResNet etc.,
2. Checking if different image augmentation techniques can be implemented.
3. Building the model in Pytorch.