iWildCam Project

Bangkit 2020

Outline

- The Dataset
- The Classification Model
  - Baseline Model
  - Pre-trained Model
- Implementations in Indonesia
- Android App Applications

The Dataset

What is the dataset?

- Collections of wild animals images taken from camera traps in Southern California
- Training set includes a total of 196,157 images from 138 different camera locations.
- Training datasets are divided into: training(60%), validation (20%) and testing dataset (20%)
- It comprises of 14 classes: 'bobcat', 'opossum', 'coyote', 'raccoon', 'dog', 'cat', 'squirrel', 'rabbit', 'skunk', 'rodent', 'deer', 'fox', 'mountain lion', 'empty'

Why the dataset?

- Exploring AI for sustainability
  - To learn how AI helps solving toughest environmental challenges
- Exploring challenges in real-life dataset
  - To learn about computer vision challenges in solving real-life problems

Data Challenges

Illumination

Motion Blur

Data Challenges

Small ROI

Occlusion

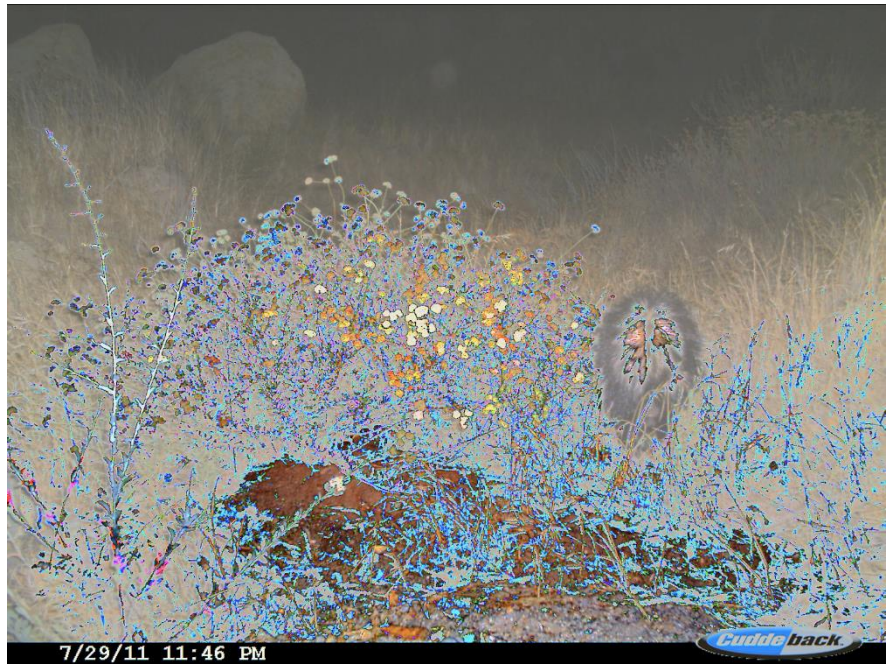
Data Challenges

Perspective

Weather

Data Challenges

Camera Malfunctions



Non Animal Variability

Data Challenges

Temporal Changes

The Classification Model

The model to classify different animals

Baseline CNN Architecture

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 32)        9248
_________________________________________________________________
flatten_2 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 14)                87822
=================================================================
Total params: 97,966
Trainable params: 97,966
Non-trainable params: 0
_________________________________________________________________
```

One set of convolution layer, using only 15,000 random samples from training dataset

Baseline CNN Generator

```
test_datagen =tf.keras.preprocessing.image.ImageDataGenerator(rescale = 1./255)

train_datagen=tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255,
                                    validation_split=0.25,

                                   )
```

```
train_generator=train_datagen.flow_from_dataframe(
                    dataframe=x_train,
                    directory="../output/kaggle/working/train_images/",
                    x_col="file_name",
                    y_col="classes_wild",
                    subset="training",
                    batch_size=32,
                    seed=424,
                    shuffle=True,
                    class_mode="categorical",
                    target_size=(32, 32))

valid_generator=train_datagen.flow_from_dataframe(
                    dataframe=x_train,
                    directory="../output/kaggle/working/train_images/",
                    x_col="file_name",
                    y_col="classes_wild",
                    subset="validation",
                    batch_size=32,
                    seed=424,
                    shuffle=True,
                    class_mode="categorical",
                    target_size=(32, 32))
```

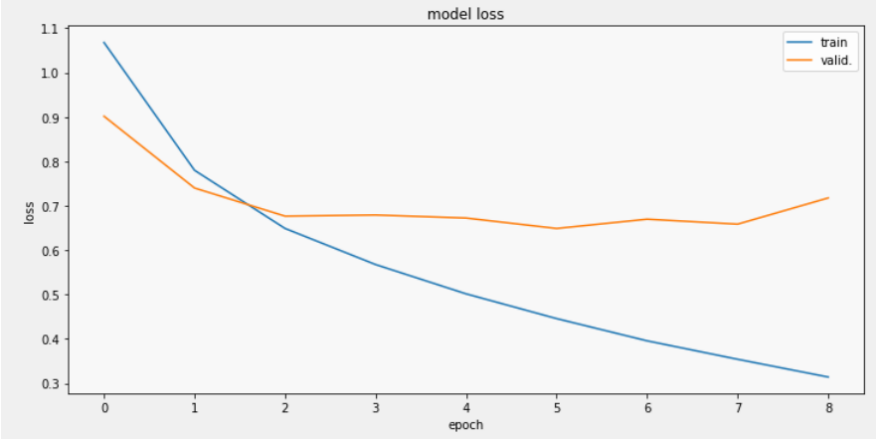
```
test_generator = test_datagen.flow_from_dataframe(
                    dataframe=x_test,
                    directory="../output/kaggle/working/train_images/",
                    x_col="file_name",
                    y_col="classes_wild",
                    batch_size=32,
                    seed=424,
                    shuffle=True,
                    class_mode="categorical",
                    target_size=(32, 32))
```

Baseline CNN Hyperparameters

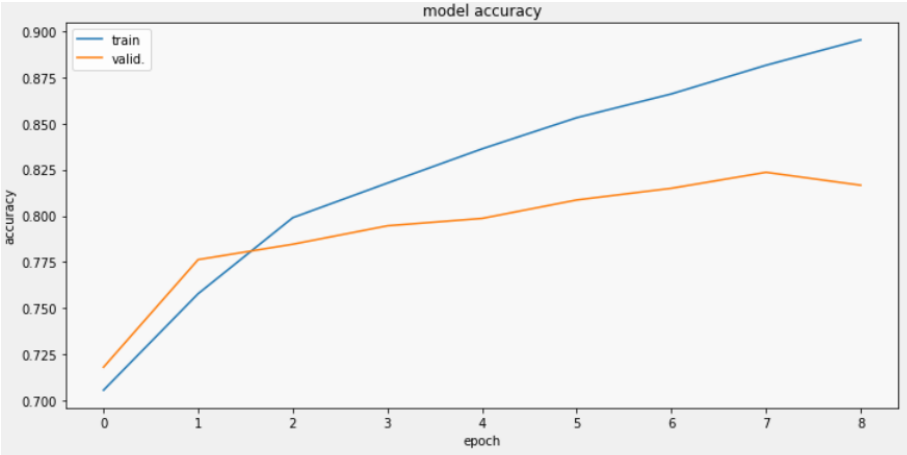
```
model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics = ['accuracy']
)

history = model.fit_generator(generator=train_generator,
                              steps_per_epoch=128,
                              validation_data=valid_generator,
                              validation_steps=64,
                              epochs=15,
                              callbacks = [early],
                              )
```

Baseline CNN Results

Train vs Validation Loss



Train vs Validation Accuracy




```
test_loss: 0.6878410671302613 and test_acc: 0.8096666932106018
```

Improvements

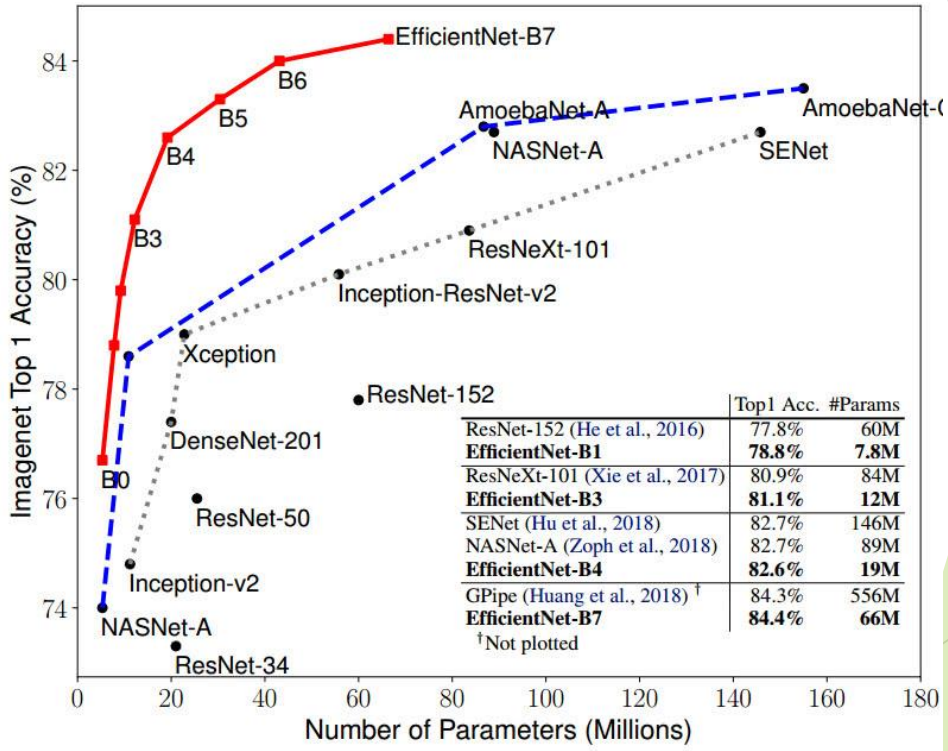
- Learning Rate
  - Experiments: 0.01 – 0.005, learning rate decay
  - Selected: lr=0.005, decay=1e-6
  - Finding: accuracy, stability and training times varies among various learning rate
- Optimizer
  - Experiments: Adam and RMSProp
  - Selected : Adam
  - Finding : doesn't get much differences . We stick with Adam.
- Input Size
  - Experiments : 32, 64 and 128
  - Selected : 128 x 128
  - Finding : 128 increases training time but give improvement in final accuracy
- Batch Size
  - Experiments: 32 and 64
  - Selected : 64
  - Finding: performance improves when using 64

Improvements

- Using Pretrained Model
  - Experiments :
    - EfficientNetB0, EfficientNetB3, EfficientNetB6, DenseNet
    - Fine tuning EfficientNetB0
  - Selected : with EfficientNetB0 with unfreeze layer block6a_expand_conv
  - Finding : all models gave accuracy around 0.8. Selected model has the highest accuracy and smallest accuracy difference between training-test dataset
- Replace dense layer with Global Average Pooling
  - Selected: Global Average Pooling
  - Finding: reduce number of training parameters -> reduce training times
- **Dropout**
  - Experiments: +/- dropout
  - Selected: add dropout
  - Finding: reduce overfitting, reduce training time
- **Callbacks**
  - Experiments: Early stopping, Learning Rate on Plateau
  - Selected: Adding early stopping only
  - Finding: adding both reduce model performances

Pretrained Model: EfficientNet

- Much smaller in size
- Scales more efficiently
- The number of parameters is reduces by magnitudes, while achieving state-of-the-art results on ImageNet.



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

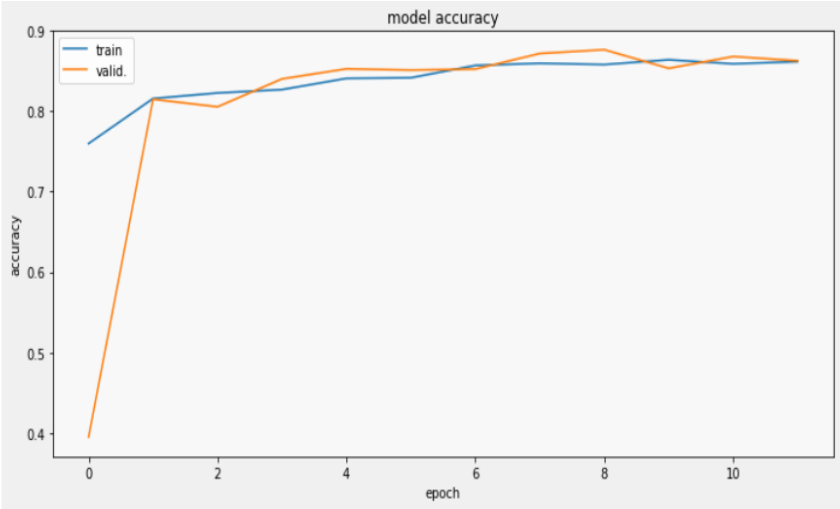
Final Model Architecture

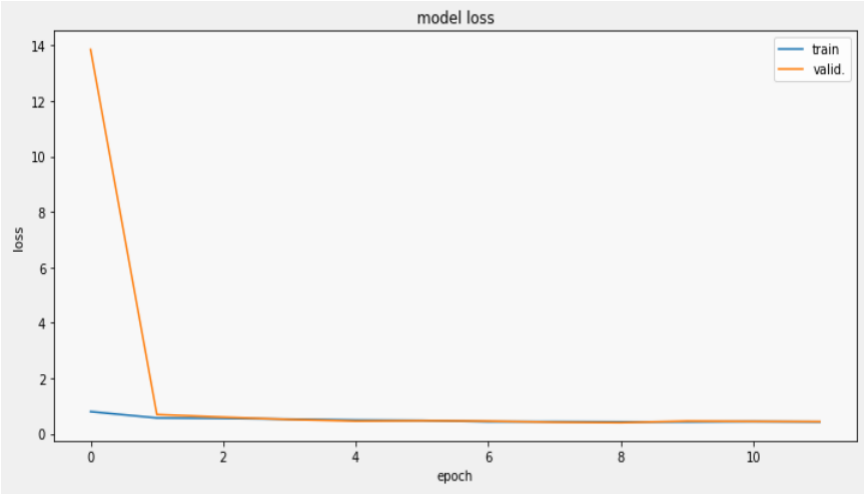
```
effnet_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
efficientnet-b0 (Model)      (None, 4, 4, 1280)        4049564
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 14)                17934
=================================================================
Total params: 4,067,498
Trainable params: 3,173,674
Non-trainable params: 893,824
_________________________________________________________________
```

```
pre_trained_model.trainable = True

set_trainable = False
for layer in pre_trained_model.layers:
    if layer.name == 'block6a_expand_conv':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

Final Model Results






```
test_loss, test_acc =effnet_model.evaluate_generator(test_generator, steps=32)
print('test_loss_effnet: {} and test_acc_effnet: {} '.format(test_loss, test_acc))
```

```
test_loss_effnet: 0.4516219198703766 and test_acc_effnet: 0.85400390625
```

What we tried but didn’t work

- Image Augmentation
  - Several augmentation techniques have been explored but it didn’t boost model performance
- Image preprocessing
  - Several Image preprocessing have been explored like CLAHE, etc.
  - Obstacles:
    - Don’t fit kaggle RAM.
  - Details are on github repo

Documentations

- The dataset can be found at:
  - https://www.kaggle.com/c/iwildcam-2019-fgvc6
- To get the tflite model run jupyter notebook at the following link:
  - https://github.com/pascalisnala/iWildCam/tree/master/ML/Notebook
  - Run Deeper_TL_iWildCam2019_EffNetLite_B0.ipynb
- To run the app:
  - https://github.com/pascalisnala/iWildCam
  - clone/pull the project from github,
  - open the Android/WildCam Project from Android Studio

Implementations

Implementation in Indonesia

- Indonesia is home to 1,531 species of bird, 515 species of mammal, 270 species of amphibian, 35 species of primate and 38,000 species of plant.
- Indonesia has been ranked as:
  - Number 1 with the most mammal species
  - Number 4 with the most known animal species after Brazil, Colombia and Peru

Biodiversity Problems in Indonesia

Reality Checks

- Ironically, it's also the country with highest deforestation rate
- 31.1 % of all species in Indonesia are endemic with 9.9% of the total number of species threatened by poaching, wildlife trade, logging and agricultural development.
- The **loss and degradation of biodiversity** negatively affects every level of the planet, especially the poor and vulnerable: women, children and indigenous people,

Existing Conservation works

- Using drones to keep track of orangutans and illegal forest clearing
- Manually deactivate traps & sweep areas for snares
- Using hi-tech tools to identify illegal wildlife sales by using cutting-edge DNA barcoding and smartphone apps

How will AI help?

- Camera traps have been widely used in other countries to help wildlife conservations.
- This real-life application has been supported by big tech companies such as Google and Microsoft.
- Applying AI technologies will help in :
  - Removing manual works in identifying animals in the pics, so conservation scientists can spend more times in doing ecology analyses
  - Identifying locations where wild animals being illegally poached
  - Identifying locations where snares and traps are located
  - Identifying poachers
  - Identifying areas where conflict occurs between wild animals, human, & industries
  - Tracking deforestation and its impact
  - Raising awareness to public

Android App

Deployment

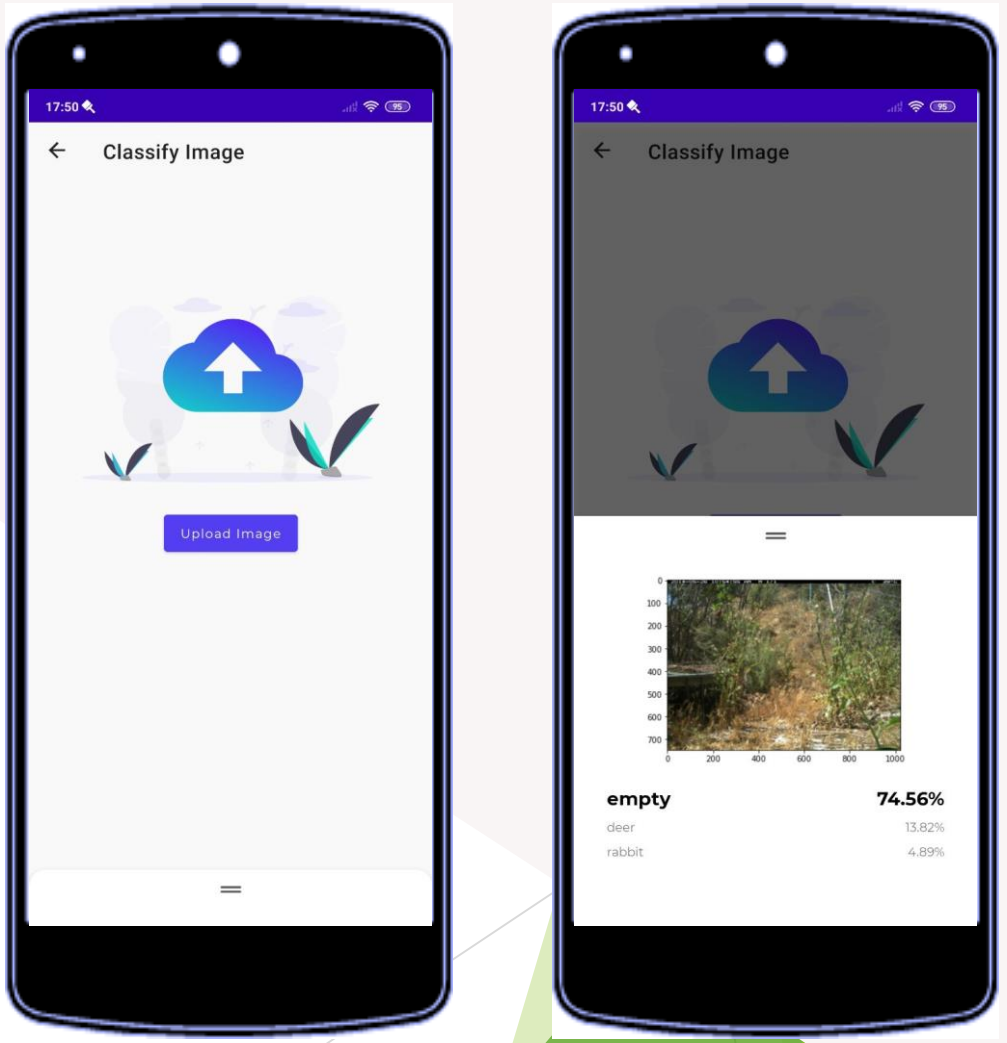
The purpose of this stage is to make the model that we have created can be used by the public for any purpose.

Our deployment is by converting the model we made previously into a Tensorflow Lite model. Thus, the model can be embedded into an android application.

Deployment (cont'd)

The reason for choosing Android and Tensorflow Lite as a technique used for deployment is in accordance with the possibility of the implementation being done in Indonesia which to classify animal captured in wildlife situation using camera traps.

So we want this application to be used anywhere not limited to location or any kind of internet connection



Deployment (cont'd)

Technique	Benefits	Hardware
Dynamic range quantization	4x smaller, 2x-3x speedup	CPU
Full integer quantization	4x smaller, 3x+ speedup	CPU, Edge TPU, Microcontrollers
Float16 quantization	2x smaller, GPU acceleration	CPU, GPU

We use dynamic range post training quantization to reduce the size of the model which converted into Tensorflow Lite model. By doing this method, we can reduce the size of the model up to 4x smaller. This method is done to maintain the application size, so it is not too big but still able to produce a classification that is almost the same as the original model

Application Demo