DOG'S BREED IDENTIFICATION

DOMESTICATED CARNIVOROUS MAMMAL

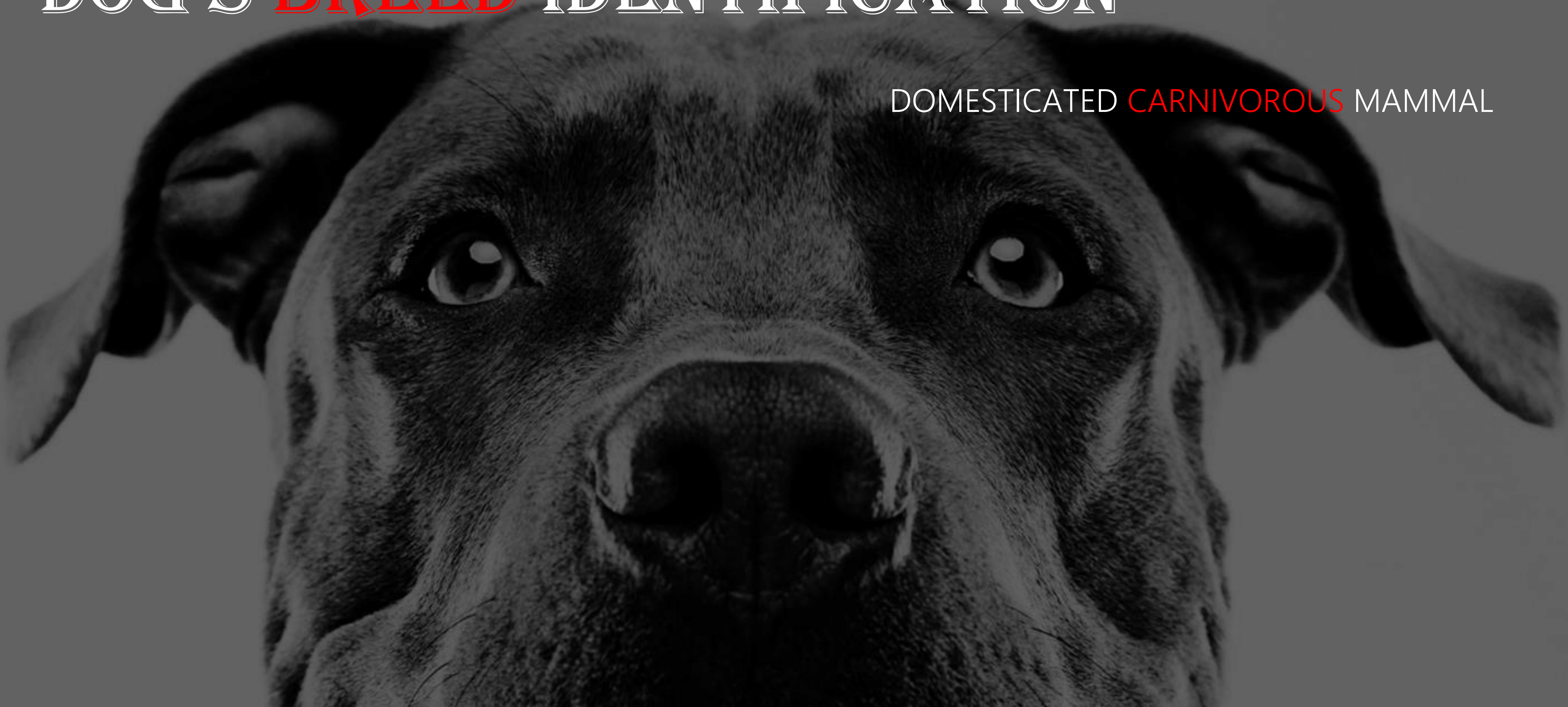
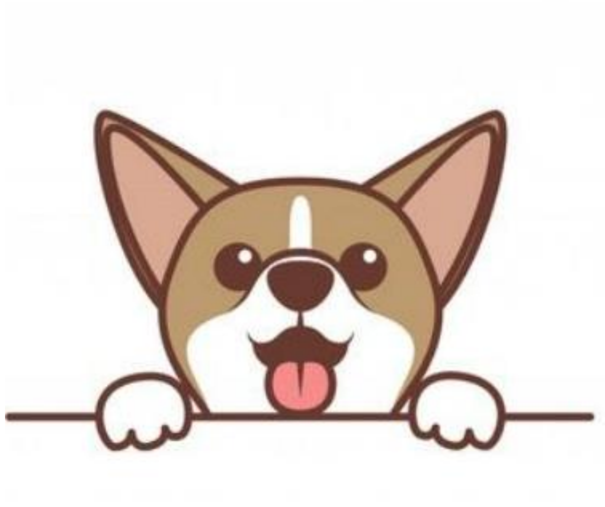


TABLE OF CONTENTS



INTRODUCTION

- With over 200 registered dog breeds and a whole host of other crossbreeds and types, choosing your perfect canine companion might seem a bit daunting at first. Different types of dogs have different needs. If you have a Terrier, for example, he will love digging, whereas a Scenthound would prefer to follow a trail to a hidden stash! A Livestock Protection dog may be happy on his own for long periods of time, but a Toy Dog needs lots more attention from you to feel content.
- So getting to know your dog's personality and behavioural needs is vital to keep them as happy as possible.
- Put simply, dogs began to recognise that humans could provide food for them, while humans realised that some types of dogs were really good at certain jobs. Humans living alongside dogs was beneficial to both and so our ancestors began to selectively breed these dogs with those jobs in mind. At the same time, dogs evolved to succeed in the ever-changing environment in which they were living.

INTRODUCTION

- If you travel around the world and look at village dog populations, you will see far more similarities than differences. Left to its own devices, the domestic dog is pretty similar no matter what country they come from. They are medium-sized, smoothish-coated, of various shades of brown with tulip shaped ears and a tail with a white tip (for easy communication) that is often held over their back. They may be slightly smaller in hot climates and larger with more coat in cold climates but basically, they are all very similar. They live alongside the human population but do not have a relationship with them.
- By contrast, when you look at the types of dogs we live with today, there couldn't be a wider variety in terms of size, shape, coat-type and personality. And, your dog’s nutritional needs differ and vary with different breeds. This is why it is important for you to choose food that caters to the specific needs of your dog’s breed to ensure overall growth & development.
- This is why if we want to understand our dogs, we need to know a little bit about how the companion dog has changed in the 15,000 years they have been living and working alongside humans.

PROBLEM STATEMENT

- We are provided with a training set and a test set of images of dogs. Each image has a filename that is its unique id. The dataset comprises 120 breeds of dogs. The goal is to create a classifier capable of determining a dog's breed from a photo. The list of breeds is as follows
- Task Description Who's a good dog? Who likes ear scratches? Well, it seems those fancy deep neural networks don't have all the answers. However, maybe they can answer that ubiquitous question we all ask when meeting a four-legged stranger: what kind of good pup is that? In this Task, we were provided a strictly canine subset of ImageNet in order to practice fine-grained image categorization. How well we can tell our Norfolk Terriers from our Norwich Terriers? With 120 breeds of dogs and a limited number training images per class, we might find the problem more, err, ruff than we anticipated.
- Support: Dog-Breed-Identification-using-CNN-with-Keras has a low active ecosystem.
- Quality: Dog-Breed-Identification-using-CNN-with-Keras has no bugs reported.
- Security: Dog-Breed-Identification-using-CNN-with-Keras has no vulnerabilities reported, and its dependent libraries have no vulnerabilities reported.
- License: Dog-Breed-Identification-using-CNN-with-Keras does not have a standard license declared.
- Re-use: Dog-Breed-Identification-using-CNN-with-Keras releases are not available. You will need to build from source code and install.

10,000 labeled images of breeds

ABSTRACT

- The current paper presents a fine-grained image recognition problem, one of multi-class classification, namely determining the breed of a dog in a given image. The presented system employs innovative methods in deep learning, including convolutional neural networks. Two different networks are trained and evaluated on the Stanford Dogs dataset. The usage/evaluation of convolutional neural networks is presented through a software system. It contains a central server and a mobile client, which includes components and libraries for evaluating on a neural network in both online and offline environments.
- With an enormous amount of effort being put into the field, multiclass classification has proven to be particularly challenging. Hence, in the present study the researcher focuses on achieving multiclass classification on dog breed identification using state of the art deep learning techniques.
- Dogs are domesticated mammals, not natural wild animals. They have been bred by humans for a long time. Today, some dogs are used as pets, others are used to help humans do their work. It’s a significant task for the owners to care and maintain their pet dog. For that, they need to know the breed of the dog to train and cure disease. The current paper presents a fine-grained image recognition problem, identifying the breed of a dog in a given image which includes convolution neural networks. The network is trained and evaluated on the Stanford Dogs Dataset. By using web scraping, the data from various websites are collected and rendered in the application.

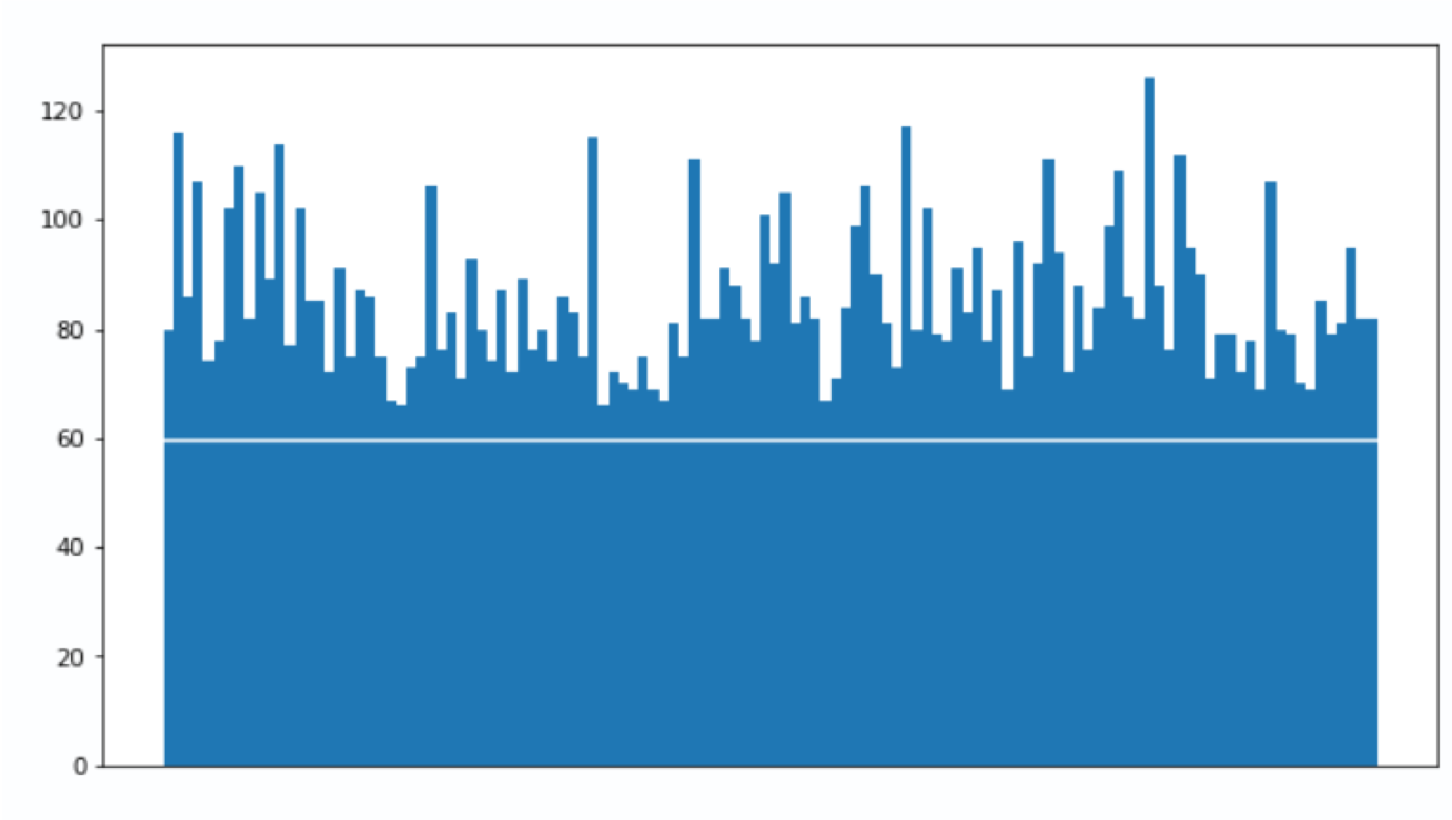
LITERATURE REVIEW

AUTHOR	INSIGHTS
Gunter, L. M., Barber, R. T., & Wynne, C. D. (2018).	A canine identity crisis: Genetic breed heritage testing of shelter dogs. *PloS one,* 13(8), e0202633.
Voith, V. L., Ingram, E., Mitsouras, K., & Irizarry, K. (2009).	Comparison of Adoption Agency Breed Identification and DNA Breed Identification of Dogs. *Journal of Applied Animal Welfare Science*, *12*(3), 253-262. doi:10.1080/10888700902956151
Voith, V. L., Trevejo, R., Dowling-Guyer, S., Chadik, C., Marder, A., Johnson, V., & Irizarry, K. (2013).	Comparison of visual and DNA breed identification of dogs and inter-observer reliability, *American Journal of Sociological Research*, *3*(2) 17-29. doi: 10.5923/j.sociology.20130302.02.
Olson, K. R., Levy, J. K., Norby, B., Crandall, M. M., Broadhurst, J. E., Jacks, S., Barton, R. C.,	Inconsistent identification of pit bull-type dogs by shelter staff. *The Veterinary Journal*, *206,* 197-202.
Simpson, R. J., Simpson, K., & VanKavage, L. (2012).	Rethinking dog breed identification in veterinary practice. *Journal of the American Veterinary Medical Association*, *241*(9), 1163-1166.
Bradley, J. (2017)	Defaming Rover: Error-Based Latent Rhetoric in the Medical Literature on Dog Bites.

DATASET

- The dataset is taken from the Dog Breed Identification competition hosted, a data science and machine learning competitions hosting platform. It contains approximately 10,000 labeled images, each of them depicts a dog from one of 120 breeds, and the same amount of testing data. Generally, these images have different resolutions, various zoom levels, they could have more than one dog shown, and were taken in various lighting conditions. Below are a few examples from the dataset.
- The dog breeds represented in the dataset are more or less balanced (i.e. each of them has a comparable number of observations) with 59 samples per breed on average. Below is the distribution of dogs breeds in the dataset.


Irish Terrier
Staffordshire Bullterrier
Beagle
Bernese Mountain Dog
Appenzeller
Weimaraner
Beagle
Ibizan Hound
Komondor
Afghan Hound
Border Terrier
Appenzeller
Norfolk Terrier
Brabancon Griffon
Kuvasz
Welsh Springer Spaniel



As our brief analysis shows, the analyzed dataset is not too sophisticated for modern deep learning architectures and has quite a simple structure. Therefore, we can expect good results and high accuracy for all breeds represented in the dataset.

MOTIVATION

- For purebred dogs, dog breed identifier tests can help prevent inbreeding and ensure genetic diversity. Health. Dog breed identifier tests can help you better understand your dog’s genetic composition, as well as what health risks your dog may be at risk for due to his genetic makeup.
- **Understanding Behavior.** Much of a dog’s personality traits and little quirks can be explained (at least in part) by their doggy genealogy. Whether your dog likes to bark, dig, or herd, many behaviors can be better understood with dog breed identifier tests.
- **Weight Prediction.** Dog breed identity testing can help predict the adult expected size and weight of your puppy. This allows you to plan ahead and better understand what your dog will need for the future, as well as what a healthy size will be.
- **Prevent Inbreeding.** For purebred dogs, dog breed identifier tests can help prevent inbreeding and ensure genetic diversity.
- **Health.** Dog breed identifier tests can help you better understand your dog’s genetic composition, as well as what health risks your dog may be at risk for due to his genetic makeup. Some breeds are much more prone to certain diseases and sicknesses than others. Knowing your dog’s breeds can help you become proactive in preventing possible health threats and ensure you dog stays healthy.
- **For Fun:** Lastly, understanding your dog’s genetic identity can be tons of fun! Of course you love your mutt regardless, but it’s still interesting and amusing to discover your canine’s great grandparents and understand his or her family tree!

OBJECTIVE

Many breeds have been selected over the years for specific tasks like herding other animals, guarding people or property, or spending long days hunting or running races. These genetics also influence the dog's behavior in the home. Research what the breed or mix is/was intended to do. Working breeds, in general, tend to be high-energy and need a lot more exercise. Herding breeds may tend to chase things that move. Guarding breeds may patrol fences or doorways and see visitors or neighbors as intruders. Hunting breeds may tend to follow their noses, including over and under fences. Racing breeds need a lot of chances to run fast in safe areas. Consider how the dog's genetic task program might mesh or conflict with the needs and desires of your family. Before choosing a pet, consider initial and recurring costs, home environment, size, temperament, and physical characteristics of the dog. Consider his training, exercising, and grooming needs. Consider your lifestyle. Then consider yourself lucky to have the right dog for your family..!

STEPS

Step 1: Import Datasets

Obviously, to be able to build an algorithm intended to identify dogs we will need some “dog data”. A lot of it. Thankfully, for this project Udacity is providing a decent number of dog images including the corresponding breed labels. Concretely, the image data comprises 8351 dog images and 133 separate dog breed names. Since the app has the additional task to assign the most resembling dog breed to a given human face, we also need a dataset with human faces. The dataset provided by Udacity includes 13233 images from the labeled faces in the wild dataset.

Step 2: Detect Humans

This seems to be a somewhat surprising step in the development of a dog identification app, but it is necessary for its extra job to assign the most resembling dog breed to a given human face. In order to detect human faces in images we will use OpenCV's implementation of Haar feature-based cascade classifiers. The approach of this classifier is based on the concept of Haar-like features, which is widely used in the field of object recognition because of its convincing calculation speed.

Step 3: Detect Dogs

Now that we have a pretty decent algorithm to detect human faces in images we surely want to build a similar function for dog detection. Unfortunately, at the moment there is no comparable "dog detector" available for OpenCV's CascadeClassifiers. Therefore, we choose another approach by employing an image classification model which has been pre-trained on the vast image database of ImageNet. More specifically, we will use the high-level deep learning API Keras to load the ResNet-50 convolutional neural network and run images through this model. For a specific image the network predicts probabilites for each of 1000 image categories in total.

Step 4: Create a CNN to Classify Dog Breeds (from Scratch)

Now we will come to the really interesting part and tackle the implementation of the app's principal task to tell the correct dog breed label from an image of a dog. We could make things easy and just use the pre-trained model from step two and predict the dog breed labels defined in the categories of the ImageNet dataset. But of course it's much more exciting, interesting and educational to build our own solution, so here we go! Before we start building our own classifier, a few words about convolutional neural networks.Convolutional neural networks (CNNs) are a class of deep neural networks primarily used in the analysis of images.

Step 5: Use a CNN to Classify Dog Breeds (using Transfer Learning)

The general idea behind transfer learning is the fact that it is much easier to teach specialized skills to a subject that already has basic knowledge in the specific domain. There are a lot of neural network models out there that already specialize in image recognition and have been trained on a huge amount of data. Our strategy now is to take advantage of such pre-trained networks and our plan can be outlined

Step 6: Create a CNN to Classify Dog Breeds (using Transfer Learning)

We will now take step 4 as a template and define our own CNN using transfer learning. We choose InceptionV3 as the network that should provide us with the features for our training layers. Inception is another high performing model on the ImageNet dataset and its power lies in the fact that the network could be designed much deeper than other models by introducing subnetworks called *inception modules*.

Step 7: Write your Algorithm

So, let's now collect the achievements and findings from the previous steps and write an algorithm that takes an image of a dog or a human und spits out a dog breed along with 4 sample images of the specific breed.

Step 8: Test your Algorithm

Finally, let's test our algorithm with a few test images.

CONCLUSION

In this project we developed several approaches for the development of an app for the identification of dog breeds, and we achieved our best results with the application of a transfer learning model. We obtained an accuracy of 83% in our tests. We also learned how to build convolution networks from scratch, which was a very educational undertaking, even though we soon realized that there are significantly more promising methods, particularly with the application of transfer learning.

- We could gather more training data.
- We could employ data augmentation to prevent overfitting.
- We could add more layers to make our model more complex and hopefully more powerful.
- We could extend our training time and add more epochs to the training.

IMPLEMENTATION

Import

```
[ ] # Imports
    import os
    import sys
    import numpy as np
    import pandas as pd
    import cv2
    import time
    import json
    from IPython.core.display import HTML
    from matplotlib import pyplot as plt
    import matplotlib.ticker as mticker
    %matplotlib inline

    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.python.keras import backend as K
    from tensorflow.keras.models import Model, Sequential
    from tensorflow.keras import layers
    from tensorflow.keras import activations
    from tensorflow.keras import optimizers
    from tensorflow.keras import losses
    from tensorflow.keras import initializers
    from tensorflow.keras import regularizers
    from tensorflow.keras.utils import to_categorical, plot_model
    from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, Callback
    # Import pretrained models
    from tensorflow.keras.applications import ResNet50V2,VGG16, InceptionV3, MobileNetV2, DenseNet121

    from sklearn.model_selection import train_test_split
    from sklearn.metrics import classification_report

    import xml.etree.ElementTree as ET # for parsing XML
    from PIL import Image # to read images
    import tensorflow_datasets as tfds
    import tensorflow_addons as tfa
```

```
[ ] # https://www.tensorflow.org/guide/data_performance
    AUTOTUNE = tf.data.experimental.AUTOTUNE
```

```
[ ] print("tensorflow version", tf.__version__)
    print("keras version", tf.keras.__version__)

    tensorflow version 2.3.0
    keras version 2.4.0
```

Test GPU + RAM

```
[ ] gpu_info = !nvidia-smi
    gpu_info = '\n'.join(gpu_info)
    if gpu_info.find('failed') >= 0:
      print('Select the Runtime > "Change runtime type" menu to enable a GPU accelerator, ')
      print('and then re-execute this cell.')
    else:
      print(gpu_info)


    from psutil import virtual_memory
    ram_gb = virtual_memory().total / 1e9
    print('Your runtime has {:.1f} gigabytes of available RAM\n'.format(ram_gb))

    if ram_gb < 20:
      print('To enable a high-RAM runtime, select the Runtime > "Change runtime type"')
      print('menu, and then select High-RAM in the Runtime shape dropdown. Then, ')
      print('re-execute this cell.')
    else:
      print('You are using a high-RAM runtime!')
```

```
Sun Sep 20 18:39:45 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.66       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   33C    P0    23W / 300W |      0MiB / 16130MiB |      0%      Default |
|                               |                      |                 ERR! |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
Your runtime has 27.4 gigabytes of available RAM

You are using a high-RAM runtime!
```

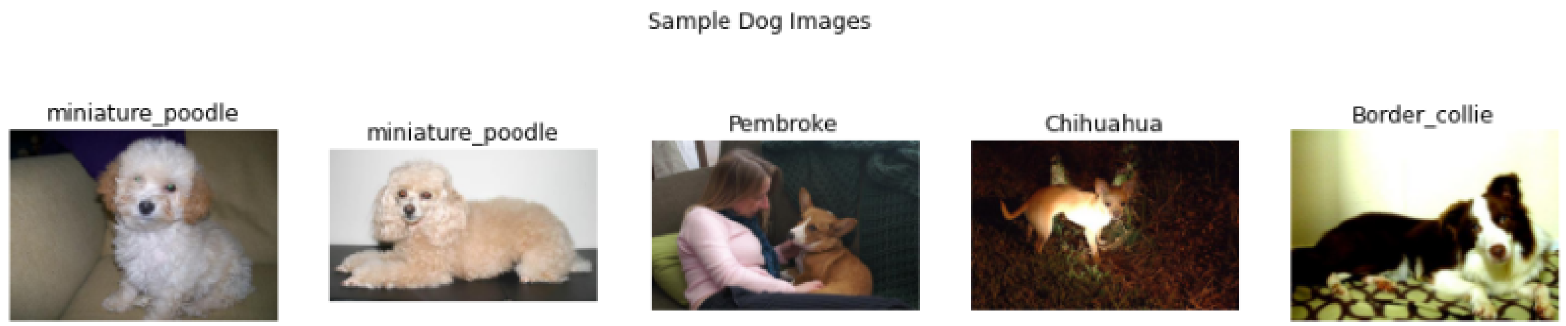
Dog DataSet

```
[ ] !rm -rf DatasetStore
```

```
[ ] import requests
    import tarfile
    dataset_path = "DatasetStore"

    # Download and extract dataset
    if not os.path.exists(dataset_path):
      os.mkdir(dataset_path)
      packet_url = "http://vision.stanford.edu/aditya86/ImageNetDogs/images.tar"
      packet_file = os.path.basename(packet_url)
      packet_file = os.path.join(dataset_path, packet_file)
      with requests.get(packet_url, stream=True) as r:
          r.raise_for_status()
          with open(packet_file, 'wb') as f:
              for chunk in r.iter_content(chunk_size=8192):
                  f.write(chunk)

      with tarfile.open(packet_file) as tfile:
        tfile.extractall(dataset_path)

      packet_url = "http://vision.stanford.edu/aditya86/ImageNetDogs/annotation.tar"
      packet_file = os.path.basename(packet_url)
      packet_file = os.path.join(dataset_path, packet_file)
      with requests.get(packet_url, stream=True) as r:
          r.raise_for_status()
          with open(packet_file, 'wb') as f:
              for chunk in r.iter_content(chunk_size=8192):
                  f.write(chunk)

      with tarfile.open(packet_file) as tfile:
        tfile.extractall(dataset_path)
```

Display some training images

```
[ ] # https://www.kaggle.com/gtimoshaz/dataset-reading-demo

    breed_list = os.listdir('DatasetStore/Annotation/') # list of all breeds for further demo

    # Train images
    fig = plt.figure(figsize=(15,8))
    for i in range(15):
        axs = fig.add_subplot(3,5,i+1)
        breed = np.random.choice(breed_list) # random breed
        dog = np.random.choice(os.listdir('DatasetStore/Annotation/' + breed)) # random image
        img = Image.open('DatasetStore/Images/' + breed + '/' + dog + '.jpg')
        tree = ET.parse('DatasetStore/Annotation/' + breed + '/' + dog) # init parser for file given
        root = tree.getroot()
        object_1 = root.findall('object')[0]; # finding all dogs. An array
        name = object_1.find('name').text;
        axs.set_title(name)
        plt.imshow(img)
        plt.axis('off')

    plt.suptitle("Sample Dog Images")
    plt.show()
```




```
[ ] breed_list = os.listdir('DatasetStore/Annotation/'); # list of all breeds for further demo
    breed_list.sort()
    for i,breed in enumerate(breed_list):
      breed_list[i] = breed[10:];

    # Create label index for easy lookup
    label2index = dict((name, index) for index, name in enumerate(breed_list))
    index2label = dict((index, name) for index, name in enumerate(breed_list))
    print(breed_list[:3])
```

```
['Chihuahua', 'Japanese_spaniel', 'Maltese_dog']
```

Load data

```
breed_list = os.listdir('DatasetStore/Annotation/') # list of all breeds for further demo
breed_list.sort()
# Create label index for easy lookup
label2index = dict((name, index) for index, name in enumerate(breed_list))
index2label = dict((index, name) for index, name in enumerate(breed_list))

images = []
annotations =[]
for breed in breed_list:
  image_files = os.listdir('DatasetStore/Images/' + breed)
  image_files.sort()
  images.extend([os.path.join('DatasetStore/Images/',breed,f) for f in image_files])
  annotations.extend([os.path.join('DatasetStore/Annotation/',breed,f.replace(".jpg","")) for f in image_files])

for idx, ann in enumerate(annotations):
    annotations[idx] = ann.split("/")[2] # add dog breed name

# Prepare train test validate datasets
Xs = np.asarray(images)
Ys = np.asarray(annotations)

print('Xs shape',Xs.shape)
print(Xs[:5])
print('Ys shape',Ys.shape)
print(Ys[:5])

# Split into train_validate + test data
train_validate_x,test_x, train_validate_y, test_y = train_test_split(Xs,Ys,test_size=0.1)

print("train_validate_x shape:",train_validate_x.shape)
print('train_validate_x[:5]:',train_validate_x[:5])
print("train_validate_y shape:",train_validate_y.shape)
print('train_validate_y[:5]:',train_validate_y[:5])

print("test_x shape:",test_x.shape)
print('test_x[:5]:',test_x[:5])
print("test_y shape:",test_y.shape)
print('test_y[:5]:',test_y[:5])
```

```
Xs shape (20580,)
['DatasetStore/Images/n02085620-Chihuahua/n02085620_10074.jpg'
 'DatasetStore/Images/n02085620-Chihuahua/n02085620_10131.jpg'
 'DatasetStore/Images/n02085620-Chihuahua/n02085620_10621.jpg'
 'DatasetStore/Images/n02085620-Chihuahua/n02085620_1073.jpg'
 'DatasetStore/Images/n02085620-Chihuahua/n02085620_10976.jpg']
Ys shape (20580,)
['n02085620-Chihuahua' 'n02085620-Chihuahua' 'n02085620-Chihuahua'
 'n02085620-Chihuahua' 'n02085620-Chihuahua']
train_validate_x shape: (18522,)
train_validate_x[:5]: ['DatasetStore/Images/n02100735-English_setter/n02100735_523.jpg'
 'DatasetStore/Images/n02089078-black-and-tan_coonhound/n02089078_188.jpg'
 'DatasetStore/Images/n02113978-Mexican_hairless/n02113978_341.jpg'
 'DatasetStore/Images/n02095570-Lakeland_terrier/n02095570_3213.jpg'
 'DatasetStore/Images/n02106030-collie/n02106030_15172.jpg']
train_validate_y shape: (18522,)
train_validate_y[:5]: ['n02100735-English_setter' 'n02089078-black-and-tan_coonhound'
 'n02113978-Mexican_hairless' 'n02095570-Lakeland_terrier'
 'n02106030-collie']
test_x shape: (2058,)
test_x[:5]: ['DatasetStore/Images/n02112350-keeshond/n02112350_4282.jpg'
 'DatasetStore/Images/n02089867-Walker_hound/n02089867_600.jpg'
 'DatasetStore/Images/n02097209-standard_schnauzer/n02097209_2629.jpg'
 'DatasetStore/Images/n02109961-Eskimo_dog/n02109961_12118.jpg'
 'DatasetStore/Images/n02094433-Yorkshire_terrier/n02094433_1824.jpg']
test_y shape: (2058,)
test_y[:5]: ['n02112350-keeshond' 'n02089867-Walker_hound'
 'n02097209-standard_schnauzer' 'n02109961-Eskimo_dog'
 'n02094433-Yorkshire_terrier']
```

```python
# View a few train images
fig = plt.figure(figsize=(15,10))

for idx in range(9):
  sample_input = cv2.imread(train_validate_x[idx])
  sample_input = cv2.cvtColor(sample_input, cv2.COLOR_BGR2RGB)
  breed = train_validate_y[idx];
  axs = fig.add_subplot(3,3,idx+1)
  axs.set_title(breed)
  plt.imshow(sample_input)
  plt.axis('off')

plt.show();
```

Build Data Generator

```
[ ] validation_percent = 0.2
    image_width = 128
    image_height = 128
    num_channels = 3
    num_classes = len(breed_list);

    epochs = 30
    train_batch_size = 32
    validation_batch_size = 32
    test_batch_size = 32
    train_shuffle_size = train_batch_size * 3
    validation_shuffle_size = validation_batch_size * 3

    # Split data into train / validation
    train_x, validate_x, train_y, validate_y = train_test_split(train_validate_x, train_validate_y, test_size=validation_percent)

    #  Converts to binary class matrix (One-hot-encoded)
    train_processed_y = np.asarray([label2index[label] for label in train_y])
    validate_processed_y = np.asarray([label2index[label] for label in validate_y])
    test_processed_y = np.asarray([label2index[label] for label in test_y])
    train_processed_y = to_categorical(train_processed_y, num_classes=num_classes, dtype='float32')
    validate_processed_y = to_categorical(validate_processed_y, num_classes=num_classes, dtype='float32')
    test_processed_y = to_categorical(test_processed_y, num_classes=num_classes, dtype='float32')

    train_data_count = train_x.shape[0]
    steps_per_epoch = np.int(train_data_count / train_batch_size)
    validation_data_count = validate_x.shape[0]
    validation_steps = np.int(validation_data_count / validation_batch_size)
```

```
    # Train data
    # Shuffle
    train_data = train_data.shuffle(train_data_count)
    # Apply all data processing logic
    for process in train_data_process_list:
        train_data = train_data.map(process, num_parallel_calls=AUTOTUNE)

    train_data = train_data.repeat(epochs).batch(train_batch_size)

    # Validation data
    # Shuffle
    validation_data = validation_data.shuffle(validation_data_count)
    # Apply all data processing logic
    for process in validate_data_process_list:
        validation_data = validation_data.map(process, num_parallel_calls=AUTOTUNE)

    validation_data = validation_data.repeat(epochs).batch(validation_batch_size)

    # Test data
    # Apply all data processing logic
    for process in test_data_process_list:
        test_data = test_data.map(process, num_parallel_calls=AUTOTUNE)
    test_data = test_data.repeat(1).batch(test_batch_size)

    return train_data, validation_data, test_data

train_data, validation_data, test_data = build_data_generators()
print("train_data",train_data)
print("validation_data",validation_data)
print("test_data",test_data)
```

```
train_data <BatchDataset shapes: ((None, 128, 128, 3), (None, 120)), types: (tf.float32, tf.float32)>
validation_data <BatchDataset shapes: ((None, 128, 128, 3), (None, 120)), types: (tf.float32, tf.float32)>
test_data <BatchDataset shapes: ((None, 128, 128, 3), (None, 120)), types: (tf.float32, tf.float32)>
```

Utility Function

```
[ ]

class JsonEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, np.integer):
            return int(obj)
        elif isinstance(obj, np.floating):
            return float(obj)
        elif isinstance(obj, decimal.Decimal):
            return float(obj)
        elif isinstance(obj, np.ndarray):
            return obj.tolist()
        else:
            return super(JsonEncoder, self).default(obj)

def get_model_metrics():
    with open("./SavedModels/model_metrics.json") as json_file:
        model_metrics = json.load(json_file)

    return model_metrics

def save_model_metrics(model_name="model_1",metrics={}):
    if os.path.exists("./SavedModels/model_metrics.json"):
        with open("./SavedModels/model_metrics.json") as json_file:
            model_metrics = json.load(json_file)
    else:
        model_metrics = {}

    model_metrics[model_name] = metrics

    # Save the json
    with open("./SavedModels/model_metrics.json", 'w') as json_file:
        json_file.write(json.dumps(model_metrics, cls=JsonEncoder))

def save_model(path="./SavedModels",model_name="model01"):
    filename = "./SavedModels/"
    os.makedirs(os.path.dirname(filename), exist_ok=True)
    # Save the enitire model (structure + weights)
    model.save(os.path.join(path,model_name+".hdf5"))
```

```
trainable_parameters = model.count_params()

# Save model metrics
metrics ={
    "trainable_parameters":trainable_parameters,
    "execution_time":execution_time,
    "loss":evaluation_results[0],
    "accuracy":evaluation_results[1],
    "model_size":model_size,
    "learning_rate":learning_rate,
    "batch_size":batch_size,
    'momentum': momentum,
    "epochs":epochs,
    "optimizer":type(optimizer).__name__
}
save_model_metrics(model_name=model.name,metrics=metrics)
```

Compare all models

```
[ ] # Compare model metrics
    view_metrics = pd.read_json("./SavedModels/model_metrics.json")
    view_metrics = view_metrics.T
    # Format columns
    view_metrics['accuracy'] = view_metrics['accuracy']*100
    view_metrics['accuracy'] = view_metrics['accuracy'].map('{:,.2f}%'.format)

    view_metrics['trainable_parameters'] = view_metrics['trainable_parameters'].map('{:,.0f}'.format)
    view_metrics['execution_time'] = view_metrics['execution_time'].map('{:,.2f} mins'.format)
    #view_metrics['loss'] = view_metrics['loss'].map('{:,.2f}'.format)
    view_metrics['model_size'] = view_metrics['model_size']/1000000
    view_metrics['model_size'] = view_metrics['model_size'].map('{:,.0f} MB'.format)
    print('Number of models:',view_metrics.shape[0])

    Number of models: 3
```

VGG16

Use VGG16 as the base and fine tune the last conv2d block for our problem

Build model

```
[ ] # vgg16 with fine tuning the last conv2d base
    vgg16 = VGG16(include_top=False, weights='imagenet', input_shape=(image_height,image_width,3))

    def view_layers(model):
        layers = model.layers
        layers_list = []

        for idx, layer in enumerate(layers):
            layers_list.append({
                'layer': type(layer).__name__,
                'trainable':layer.trainable
            })

        df = pd.DataFrame(layers_list)

        return df

    layers_df = view_layers(vgg16)
    print(layers_df[10:])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
          layer  trainable
10  MaxPooling2D      True
11        Conv2D      True
12        Conv2D      True
13        Conv2D      True
14  MaxPooling2D      True
15        Conv2D      True
16        Conv2D      True
17        Conv2D      True
18  MaxPooling2D      True
```

Training params

```
#############################
# Training Params
#############################
learning_rate = 0.001
batch_size = 32
epochs = 50
#############################


# Set all layers as trainable false execpt last conv block
for layer in vgg16.layers[:-4]:
    layer.trainable = False

# Input
model_input = vgg16.layers[0].input

# Final pool layer
hidden = vgg16.layers[-1]
print("Pool Layer",hidden)

# Flatten
hidden = layers.Flatten()(hidden.output)

#  Hidden Layer, Classification Block
hidden = layers.Dense(units=1024, activation='relu')(hidden)
hidden = layers.Dense(units=1024, activation='relu')(hidden)

# Output Layer
output = layers.Dense(units=num_classes, activation='softmax')(hidden)

# Build model
model = Model(model_input, output, name='VGG16')

# Optimizer
optimizer = optimizers.SGD(lr=learning_rate)

# Loss
loss = losses.categorical_crossentropy

# Compile
model.compile(loss=loss,
                  optimizer=optimizer,
                  metrics=['accuracy'])
```

```python
# Optimizer
optimizer = optimizers.SGD(lr=learning_rate)

# Loss
loss = losses.categorical_crossentropy

# Compile
model.compile(loss=loss,
                  optimizer=optimizer,
                  metrics=['accuracy'])

#print(model.summary())
layers_df = view_layers(model)
print(layers_df.head(25))
```

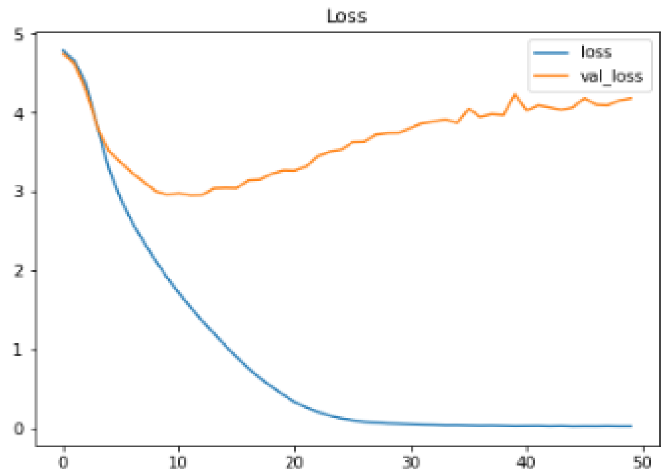
```
Pool Layer <tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7fc517d55a20>
           layer  trainable
0     InputLayer      False
1         Conv2D      False
2         Conv2D      False
3   MaxPooling2D      False
4         Conv2D      False
5         Conv2D      False
6   MaxPooling2D      False
7         Conv2D      False
8         Conv2D      False
9         Conv2D      False
10  MaxPooling2D      False
11        Conv2D      False
12        Conv2D      False
13        Conv2D      False
14  MaxPooling2D      False
15        Conv2D       True
16        Conv2D       True
17        Conv2D       True
18  MaxPooling2D       True
19       Flatten       True
20         Dense       True
21         Dense       True
22         Dense       True
```

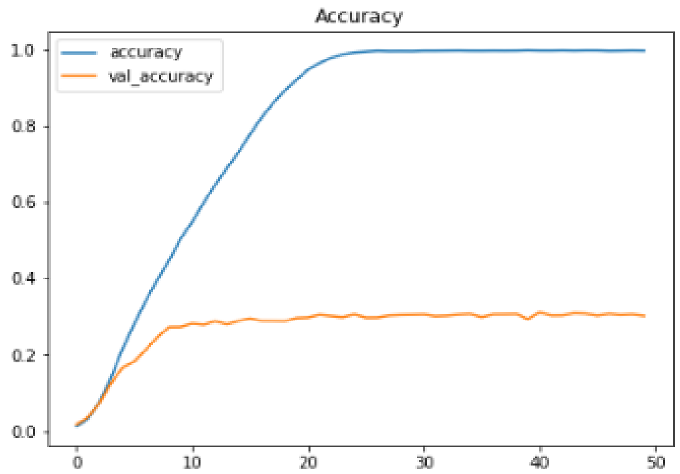
Train model

```
[ ] # Train model

    # Early Stopping
    earlystopping = EarlyStopping(monitor='val_accuracy', patience=10);

    # Model Checkpoint
    checkpoint_filepath = './Checkpoints/checkpoint_VGG16'
    model_checkpoint_callback = ModelCheckpoint(
        filepath=checkpoint_filepath,
        save_weights_only=True,
        monitor='val_accuracy',
        verbose=1,
        mode='max',
        save_best_only=True)


    start_time = time.time()
    training_results = model.fit(
            train_data,
            validation_data=validation_data,
            callbacks=[earlystopping,model_checkpoint_callback],
            epochs=epochs,
            verbose=1,
            steps_per_epoch=steps_per_epoch,
            validation_steps=validation_steps)
    execution_time = (time.time() - start_time)/60.0
    print("Training execution time (mins)",execution_time)
```

Evaluate and Save

```
[ ] # Evaluate and Save model
    evaluate_save_model(model,training_results,test_data,execution_time, learning_rate, batch_size, epochs, optimizer)
```






```
65/65 [==============================] - 1s 22ms/step - loss: 3.9470 - accuracy: 0.3246
Evaluation results: [loss, accuracy] [3.947021722793579, 0.32458698749542236]
```

ResNet50V2 with Adam optimizer

```
[ ] resnet50_v2 = ResNet50V2(
        include_top=False,
        input_shape=(128, 128, 3)
    )
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5
94674944/94668760 [==============================] - 1s 0us/step
```

Build model

Build model

```
[ ] # Build model for Resnet
    def build_resnet_model(model_name = 'ResNet50V2',print_summary=True):
      # Set all layers as hidden
      for layer in resnet50_v2.layers:
          layer.trainable = False

      # Input
      model_input = resnet50_v2.layers[0].input

      # Extract final pool layer
      hidden = resnet50_v2.layers[-1]

      # Flatten
      hidden = layers.Flatten()(hidden.output)

      # Output Layer
      output = layers.Dense(units=120, activation='softmax')(hidden)

      # Create model
      model = Model(model_input, output, name=model_name)

      # Print the model architecture
```

Training params

```
[ ] ##############################
    # Training Params
    ##############################
    batch_size = 32
    epochs = 50
    ##############################

    # Early Stopping
    earlystopping = EarlyStopping(monitor='val_accuracy', patience=10);

    # Model Checkpoint
    checkpoint_filepath = './Checkpoints/checkpoint_ResNet50V2'
    model_checkpoint_callback = ModelCheckpoint(
        filepath=checkpoint_filepath,
        save_weights_only=True,
        monitor='val_accuracy',
        verbose=1,
        mode='max',
        save_best_only=True)

    # Build the model
    model = build_resnet_model()

    # Optimier
    optimizer = optimizers.Adam()

    # Loss
    loss = losses.categorical_crossentropy

    # Compile
    model.compile(loss=loss,
                      optimizer=optimizer,
                      metrics=['accuracy'])
```

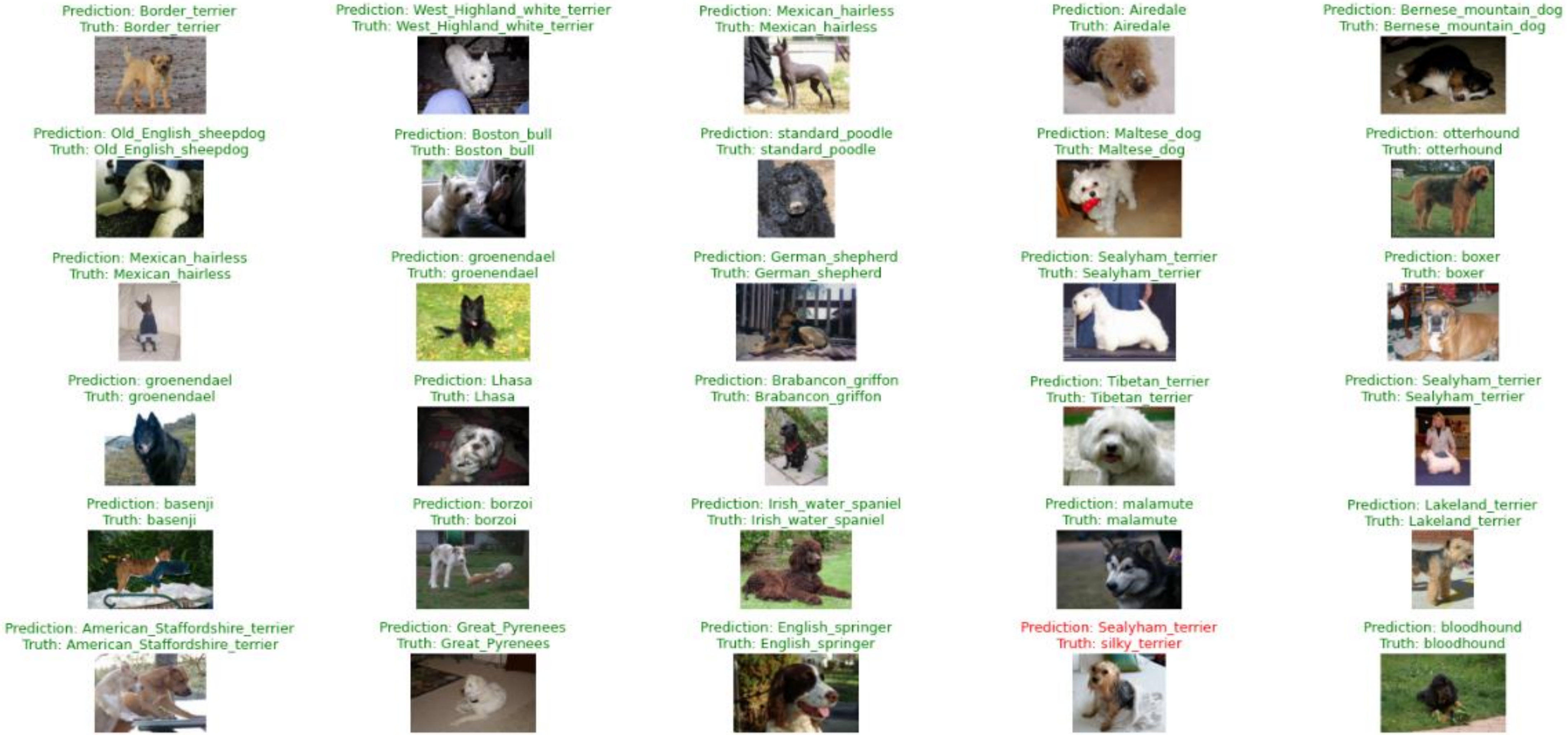
```
conv5_block2_preact_relu (Activ (None, 4, 4, 2048)   0         conv5_block2_preact_bn[0][0]
conv5_block2_1_conv (Conv2D)    (None, 4, 4, 512)    1048576   conv5_block2_preact_relu[0][0]
conv5_block2_1_bn (BatchNormali (None, 4, 4, 512)    2048      conv5_block2_1_conv[0][0]
conv5_block2_1_relu (Activation (None, 4, 4, 512)    0         conv5_block2_1_bn[0][0]
conv5_block2_2_pad (ZeroPadding (None, 6, 6, 512)    0         conv5_block2_1_relu[0][0]
conv5_block2_2_conv (Conv2D)    (None, 4, 4, 512)    2359296   conv5_block2_2_pad[0][0]
conv5_block2_2_bn (BatchNormali (None, 4, 4, 512)    2048      conv5_block2_2_conv[0][0]
conv5_block2_2_relu (Activation (None, 4, 4, 512)    0         conv5_block2_2_bn[0][0]
conv5_block2_3_conv (Conv2D)    (None, 4, 4, 2048)   1050624   conv5_block2_2_relu[0][0]
conv5_block2_out (Add)          (None, 4, 4, 2048)   0         conv5_block1_out[0][0]
                                                               conv5_block2_3_conv[0][0]
conv5_block3_preact_bn (BatchNo (None, 4, 4, 2048)   8192      conv5_block2_out[0][0]
conv5_block3_preact_relu (Activ (None, 4, 4, 2048)   0         conv5_block3_preact_bn[0][0]
conv5_block3_1_conv (Conv2D)    (None, 4, 4, 512)    1048576   conv5_block3_preact_relu[0][0]
conv5_block3_1_bn (BatchNormali (None, 4, 4, 512)    2048      conv5_block3_1_conv[0][0]
conv5_block3_1_relu (Activation (None, 4, 4, 512)    0         conv5_block3_1_bn[0][0]
conv5_block3_2_pad (ZeroPadding (None, 6, 6, 512)    0         conv5_block3_1_relu[0][0]
conv5_block3_2_conv (Conv2D)    (None, 4, 4, 512)    2359296   conv5_block3_2_pad[0][0]
conv5_block3_2_bn (BatchNormali (None, 4, 4, 512)    2048      conv5_block3_2_conv[0][0]
conv5_block3_2_relu (Activation (None, 4, 4, 512)    0         conv5_block3_2_bn[0][0]
conv5_block3_3_conv (Conv2D)    (None, 4, 4, 2048)   1050624   conv5_block3_2_relu[0][0]
conv5_block3_out (Add)          (None, 4, 4, 2048)   0         conv5_block2_out[0][0]
                                                               conv5_block3_3_conv[0][0]
post_bn (BatchNormalization)    (None, 4, 4, 2048)   8192      conv5_block3_out[0][0]
post_relu (Activation)          (None, 4, 4, 2048)   0         post_bn[0][0]
flatten (Flatten)               (None, 32768)        0         post_relu[0][0]
dense (Dense)                   (None, 120)          3932280   flatten[0][0]
==================================================================================================
```

```
conv3_block1_2_conv (Conv2D)    (None, 16, 16, 128)  147456    conv3_block1_2_pad[0][0]
conv3_block1_2_bn (BatchNormali (None, 16, 16, 128)  512       conv3_block1_2_conv[0][0]
conv3_block1_2_relu (Activation (None, 16, 16, 128)  0         conv3_block1_2_bn[0][0]
conv3_block1_0_conv (Conv2D)    (None, 16, 16, 512)  131584    conv3_block1_preact_relu[0][0]
conv3_block1_3_conv (Conv2D)    (None, 16, 16, 512)  66048     conv3_block1_2_relu[0][0]
conv3_block1_out (Add)          (None, 16, 16, 512)  0         conv3_block1_0_conv[0][0]
                                                               conv3_block1_3_conv[0][0]
conv3_block2_preact_bn (BatchNo (None, 16, 16, 512)  2048      conv3_block1_out[0][0]
conv3_block2_preact_relu (Activ (None, 16, 16, 512)  0         conv3_block2_preact_bn[0][0]
conv3_block2_1_conv (Conv2D)    (None, 16, 16, 128)  65536     conv3_block2_preact_relu[0][0]
conv3_block2_1_bn (BatchNormali (None, 16, 16, 128)  512       conv3_block2_1_conv[0][0]
conv3_block2_1_relu (Activation (None, 16, 16, 128)  0         conv3_block2_1_bn[0][0]
conv3_block2_2_pad (ZeroPadding (None, 18, 18, 128)  0         conv3_block2_1_relu[0][0]
conv3_block2_2_conv (Conv2D)    (None, 16, 16, 128)  147456    conv3_block2_2_pad[0][0]
conv3_block2_2_bn (BatchNormali (None, 16, 16, 128)  512       conv3_block2_2_conv[0][0]
conv3_block2_2_relu (Activation (None, 16, 16, 128)  0         conv3_block2_2_bn[0][0]
conv3_block2_3_conv (Conv2D)    (None, 16, 16, 512)  66048     conv3_block2_2_relu[0][0]
conv3_block2_out (Add)          (None, 16, 16, 512)  0         conv3_block1_out[0][0]
                                                               conv3_block2_3_conv[0][0]
conv3_block3_preact_bn (BatchNo (None, 16, 16, 512)  2048      conv3_block2_out[0][0]
conv3_block3_preact_relu (Activ (None, 16, 16, 512)  0         conv3_block3_preact_bn[0][0]
conv3_block3_1_conv (Conv2D)    (None, 16, 16, 128)  65536     conv3_block3_preact_relu[0][0]
conv3_block3_1_bn (BatchNormali (None, 16, 16, 128)  512       conv3_block3_1_conv[0][0]
conv3_block3_1_relu (Activation (None, 16, 16, 128)  0         conv3_block3_1_bn[0][0]
conv3_block3_2_pad (ZeroPadding (None, 18, 18, 128)  0         conv3_block3_1_relu[0][0]
conv3_block3_2_conv (Conv2D)    (None, 16, 16, 128)  147456    conv3_block3_2_pad[0][0]
conv3_block3_2_bn (BatchNormali (None, 16, 16, 128)  512       conv3_block3_2_conv[0][0]
conv3_block3_2_relu (Activation (None, 16, 16, 128)  0         conv3_block3_2_bn[0][0]
```

Display predictions

```
[ ] # load best model with best weights
    prediction_model = tf.keras.models.load_model('./SavedModels/ResNet50V2_DataAug.hdf5')
    checkpoint_path = './Checkpoints/checkpoint_ResNet50V2DataAug'
    prediction_model.load_weights(checkpoint_path)

    # make predictions
    test_predictions = prediction_model.predict(test_data)

    # Load Test images
    test_x_display = []
    for path in test_x:
        # read image
        image = cv2.imread(path)
        # convert to rgb
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        # Train x
        test_x_display.append(image)

    # Convert to numpy array
    test_x_display = np.asarray(test_x_display)

    # add true and predicted breed for each dog image
    # mark it green if prediction is true, otherwise red
    # count the total number of true predictions for the first 100 images
    true_predict = 0
    false_predict = 0
    fig = plt.figure(figsize=(20,16))
    for i,file in enumerate(test_x_display[:50]):
        axs = fig.add_subplot(10,5,i+1)
        axs.set_aspect('equal')
        predicted_breed = index2label[test_predictions.argmax(axis=1)[i]][10:] # [10:] truncates leading unnecessary letters
        true_breed = test_y[i][10:]
        # color code true/false predictions
        if true_breed == predicted_breed:
          axs.set_title('Prediction: ' + predicted_breed + '\n' + 'Truth: ' + true_breed,color='green')
          true_predict += 1
        else:
          axs.set_title('Prediction: ' + predicted_breed + '\n' + 'Truth: ' + true_breed,color='red')
          false_predict += 1
        plt.imshow(test_x_display[i])
        plt.axis('off')
    plt.tight_layout()
    plt.show()

    print('# of true predictions: ', true_predict)
    print('# of false predictions: ', false_predict)
```

Prediction: keeshond
Truth: keeshond
Prediction: beagle
Truth: Walker_hound
Prediction: soft-coated_wheaten_terrier
Truth: standard_schnauzer
Prediction: Eskimo_dog
Truth: Eskimo_dog
Prediction: Australian_terrier
Truth: Yorkshire_terrier
Prediction: miniature_schnauzer
Truth: miniature_schnauzer
Prediction: Boston_bull
Truth: toy_terrier
Prediction: Cardigan
Truth: EntleBucher
Prediction: Chihuahua
Truth: American_Staffordshire_terrier
Prediction: bull_mastiff
Truth: bull_mastiff
Prediction: Irish_wolfhound
Truth: Scottish_deerhound
Prediction: soft-coated_wheaten_terrier
Truth: clumber
Prediction: Japanese_spaniel
Truth: Japanese_spaniel
Prediction: cairn
Truth: cairn
Prediction: French_bulldog
Truth: French_bulldog
Prediction: kelpie
Truth: kelpie
Prediction: flat-coated_retriever
Truth: flat-coated_retriever
Prediction: Blenheim_spaniel
Truth: Blenheim_spaniel
Prediction: Brabancon_griffon
Truth: Brabancon_griffon
Prediction: Irish_setter
Truth: Irish_setter
Prediction: Border_terrier
Truth: Border_terrier
Prediction: West_Highland_white_terrier
Truth: West_Highland_white_terrier
Prediction: Mexican_hairless
Truth: Mexican_hairless
Prediction: Airedale
Truth: Airedale
Prediction: Bernese_mountain_dog
Truth: Bernese_mountain_dog
Prediction: Old_English_sheepdog
Truth: Old_English_sheepdog
Prediction: Boston_bull
Truth: Boston_bull
Prediction: standard_poodle
Truth: standard_poodle
Prediction: Maltese_dog
Truth: Maltese_dog
Prediction: otterhound
Truth: otterhound
Prediction: Mexican_hairless
Truth: Mexican_hairless
Prediction: groenendael
Truth: groenendael
Prediction: German_shepherd
Truth: German_shepherd
Prediction: Sealyham_terrier
Truth: Sealyham_terrier
Prediction: boxer
Truth: boxer




```
# of true predictions:  41
# of false predictions:  9
```

RESULT

- 65/65 [==] - 1s 21ms/step - loss: 92.6662 - accuracy: 0.4368
- Evaluation results: [loss, accuracy] [92.66619873046875, 0.43683186173439026]
- true predictions: 41 , false predictions: 9
- The accuracy we have achieved by using this model is 82.05%
- Overall, we consider our results to be a success given the high number of breeds in this fine-grained classification problem. We are able to effectively predict the correct breed over 50% of the time in one guess, a result that very few humans could match given the high variability both between and within the 166 different breeds contained in the dataset.

THANK YOU

RA2011026010100 – VAISHALI V
RA2011026010060- SHIVANI R
RA2011026010119 – PARKAVI