Dog Breed Image Recognition

Sarah Gates, Jesse Borg, Luis Ahumada

Agenda

- Project Definition
  - Scope / Features / Data Sources / Expected Outcomes
- Project Architecture
  - Logical Structure / Data Flow
- Project Implementation
  - Cloud Services / Inputs & Outputs
- Demo
- Questions

Scope

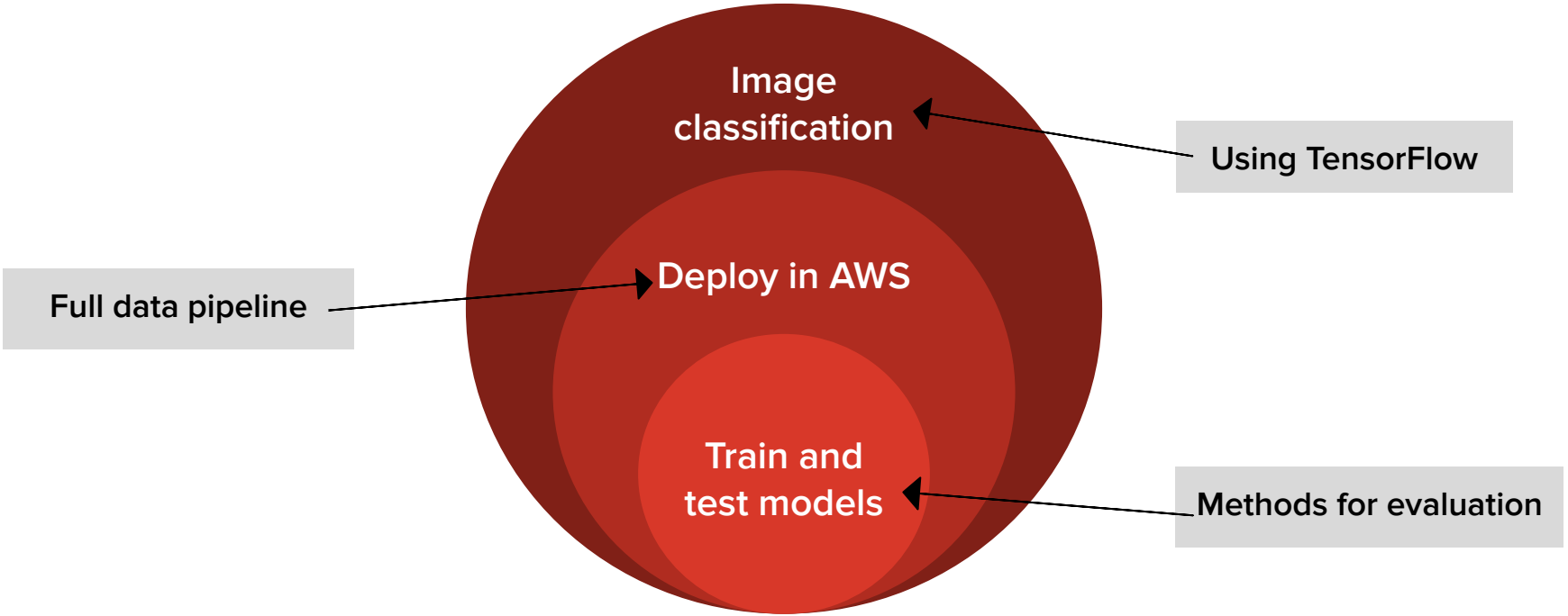


Image classification
Deploy in AWS
Train and test models
Using TensorFlow
Full data pipeline
Methods for evaluation

Features

Images stored in S3

Python scripts to run machine learning models

EC2 Instances to deploy scripts

Evaluate model success rate based on loss and accuracy

Data Sources

- Stanford Dogs dataset
- Images of 120 breeds of dogs from around the world
  - Number of categories: 120
  - Number of images: 20,580
  - Annotations: Class labels, Bounding boxes
- There are 20,580 images
  - 12,000 are used for training
  - 8,580 for testing

Expected Outcomes

- Fine-grained image categorization of dog breeds

Project Architecture

Preprocessing Data

- Extract dataset from TensorFlow
- Store images using S3 buckets

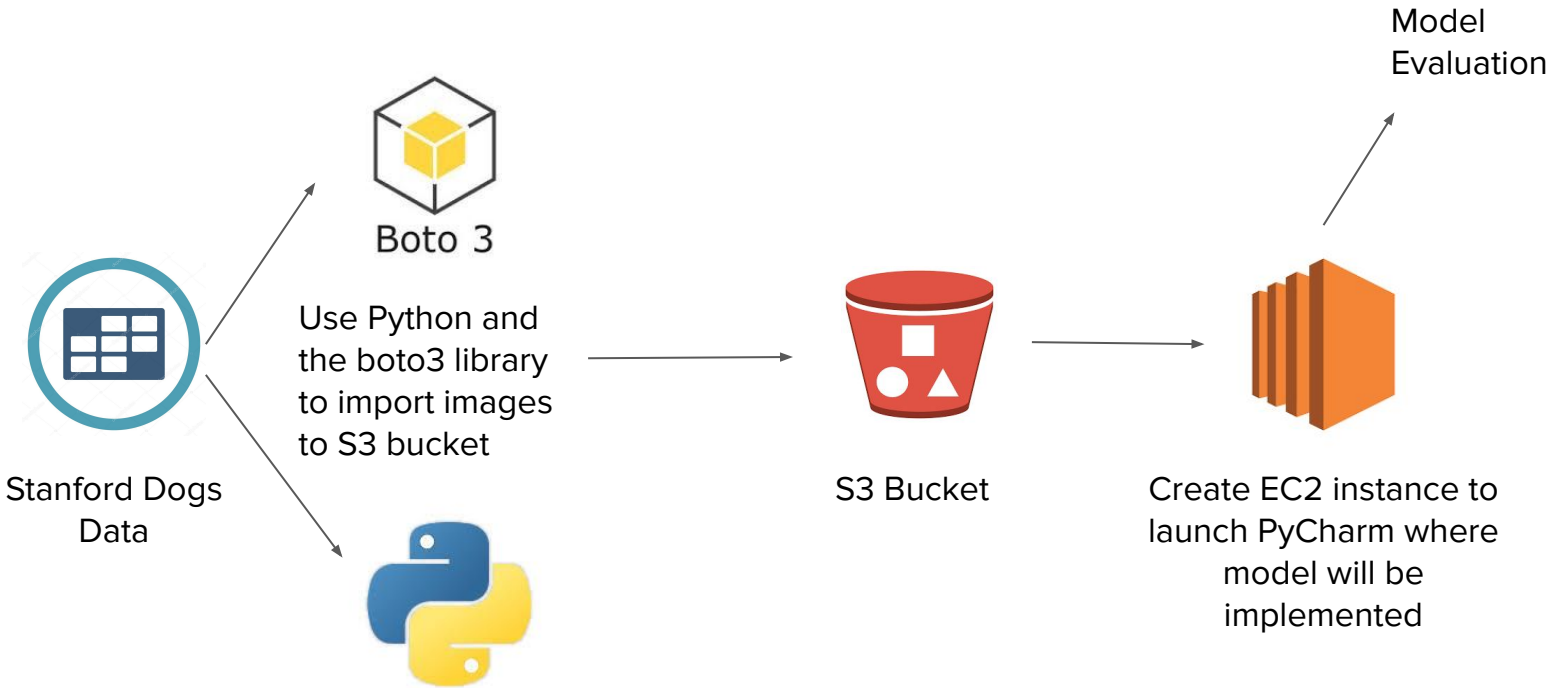
Model

- Use a machine learning model from previous project - test to see if it works
- Setup EC2 instances to access Python from SSH
- Use model to categorize dog breeds

Validation

- Measure accuracy and loss as metrics for successful classification

Data Flow


Model
Evaluation
Boto 3
Use Python and
the boto3 library
to import images
to S3 bucket
Stanford Dogs
Data
S3 Bucket
Create EC2 instance to
launch PyCharm where
model will be
implemented

Project Implementation

Configuration

- Create logins for all group members into a single account, managed by the root account
- Create a security group granting users with required permissions
- Create an S3 bucket
- Create an EC2 instance

Deployment

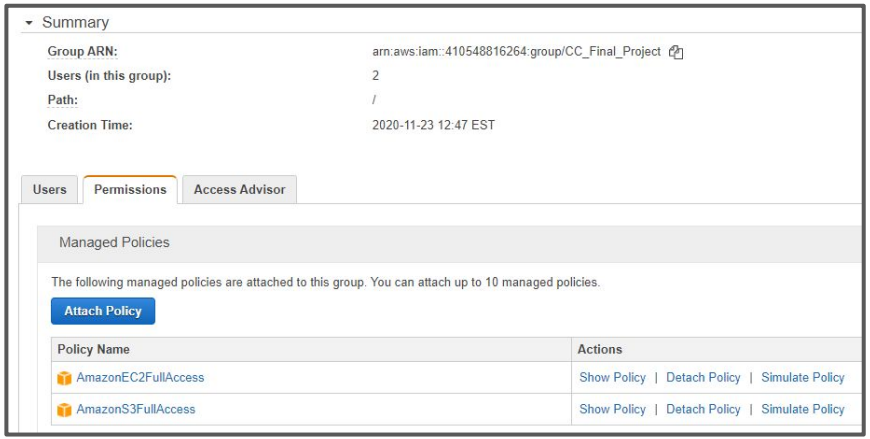
- SSH into EC2 instance and launch Python
- Use Boto 3 to load image data into the S3 bucket
- Run the developed code to train the model
- Initiate the model on testing data

Results

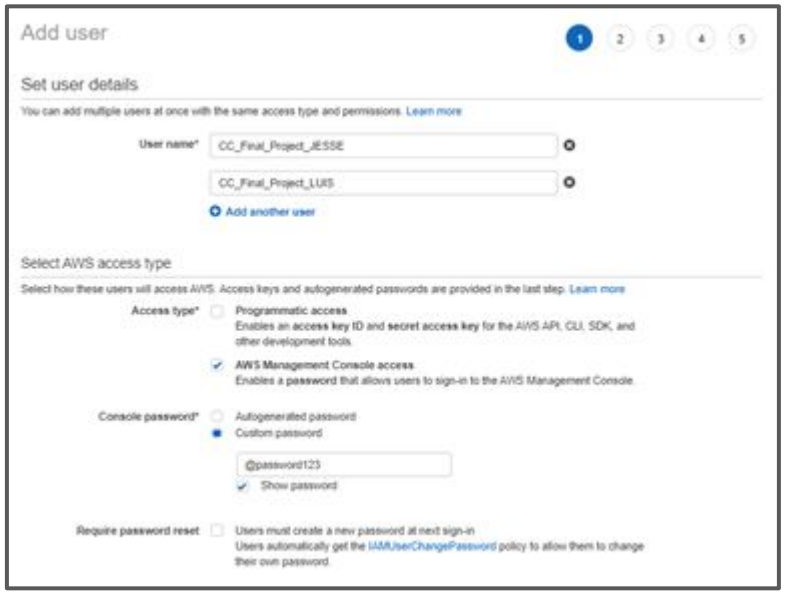
- Validate that code ran as expected and images are categorized via S3 in AWS console
- Review loss and accuracy metrics output in Python

Demo

Configuration



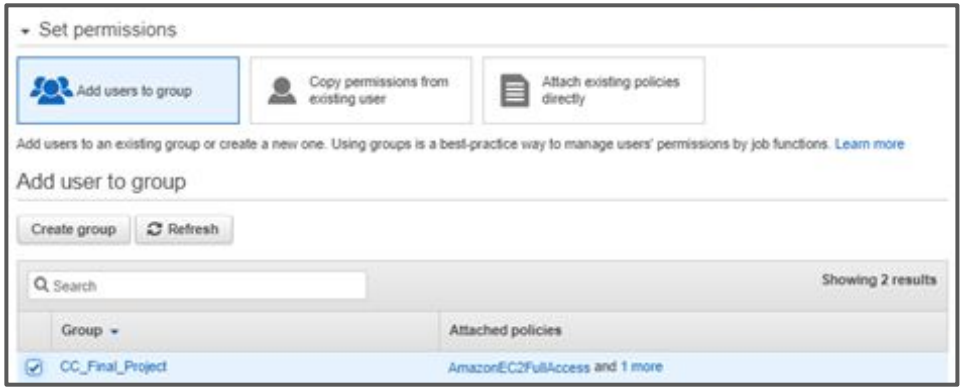
Step 1: Created security group



Step 2: Created users and set up login credentials

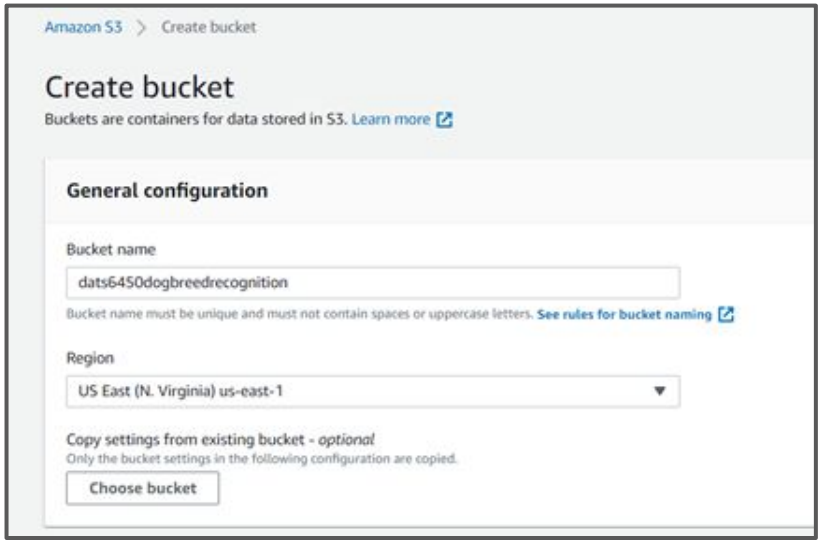
Demo

Configuration



Step 3: Added the users to the security group

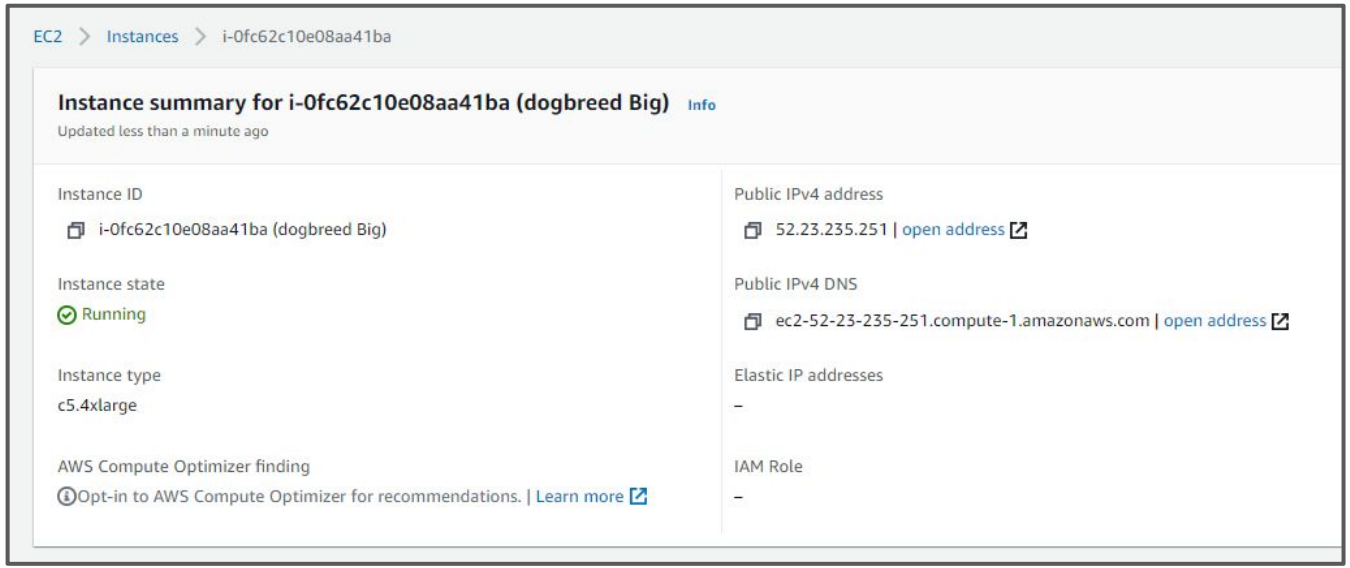
Step 4: Created one S3 bucket



Demo

Configuration

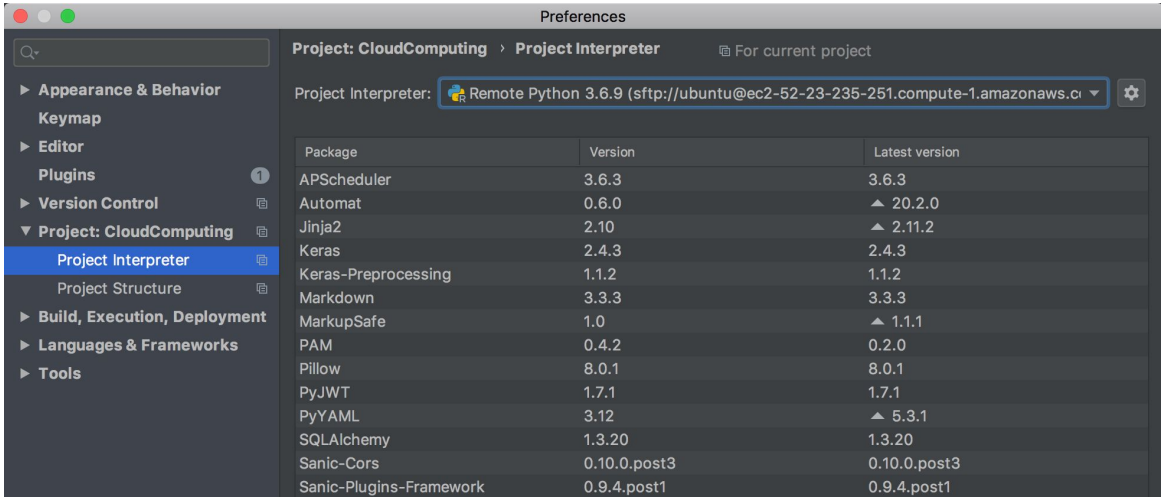
Step 5: Created an EC2 instance
Step 6: Managed instance size constraints



Demo

Deployment

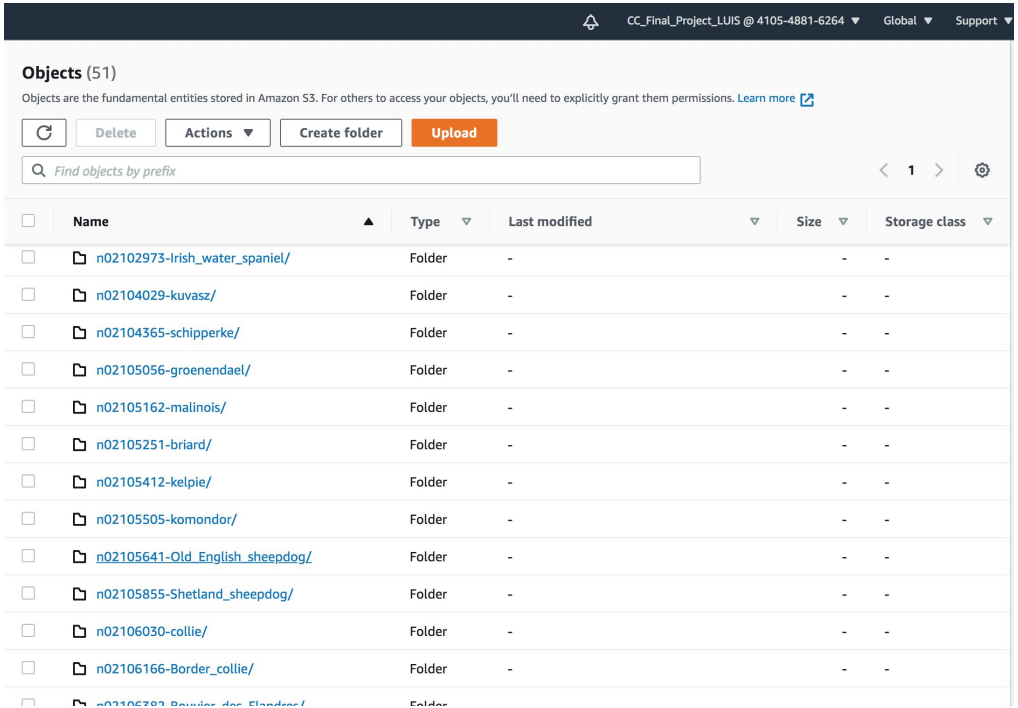
Step 1: SSH into the EC2 instance and launched python



Demo

Deployment

Step 2: Ensure images uploaded into S3 bucket



Demo

Deployment

Step 3: modified the developed code to run smoothly in PyCharm then initiated model training

```
#%%
import tensorflow_datasets as tfds

# Get the name of the data
data_name = 'stanford_dogs'

# Load data
data, info = tfds.load(name=data_name, data_dir=abspath + 'data/', as_supervised=True, with_info=True)

# Get the name of the target
target = 'label'

#%%
# Get the classes
classes = info.features['label'].names

# Print the classes
print(classes)

# Get the number of classes
n_classes = info.features['label'].num_classes

# Print the number of classes
print(info.features['label'].num_classes)
```

Demo

Deployment

Step 4: modified the developed code to run smoothly in PyCharm then initiated model training

```
#%%
# Set the training, validation and testing split
split_train, split_valid, split_test = 'train[:70%]', 'train[70%:]', 'test'

# Get the training data
data_train = tfds.load(name=data_name, split=split_train, data_dir=abspath + 'data/', as_supervised=True)

# Get the validation data
data_valid = tfds.load(name=data_name, split=split_valid, data_dir=abspath + 'data/', as_supervised=True)

# Get the testing data
data_test = tfds.load(name=data_name, split=split_test, data_dir=abspath + 'data/', as_supervised=True)

#%%
# Resize the data for the pretrained model
# Set the default input size for the pretrained model
input_size = [299, 299]
```

Demo

Deployment

Step 4: modified the developed code to run smoothly in PyCharm then initiated model training

```
#%%
# Build the architecture of the model

# Add the pretrained layers
pretrained_model = keras.applications.xception.Xception(include_top=False, weights='imagenet')

# Add GlobalAveragePooling2D layer
average_pooling = keras.layers.GlobalAveragePooling2D()(pretrained_model.output)

# Add the output layer
output = keras.layers.Dense(n_classes, activation='softmax')(average_pooling)

# Get the model
model = keras.Model(inputs=pretrained_model.input, outputs=output)
```

Demo

Deployment

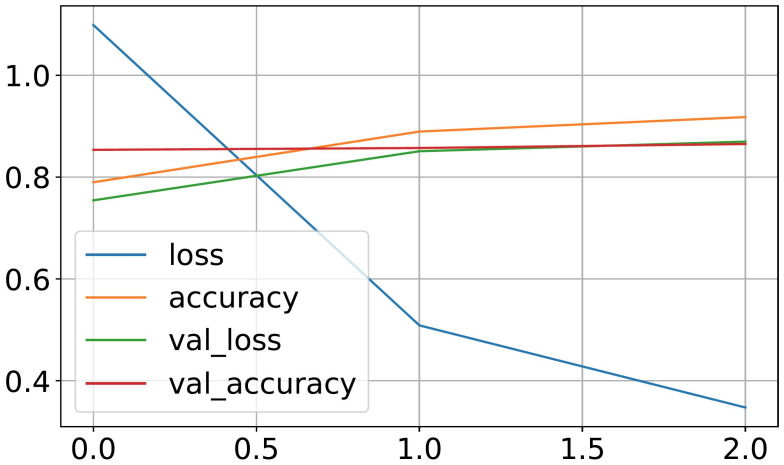
Step 4: modified the developed code to run smoothly in PyCharm then initiated model training

```
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2020-11-30 02:39:33.910092: I tensorflow/core/platform/profile_utils/cpu_utils.cc:104] CPU Frequency: 2999995000 Hz
2020-11-30 02:39:33.910609: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x5844550 initialized for platform Host (this doe
2020-11-30 02:39:33.910635: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): Host, Default Version
['n02085620-chihuahua', 'n02085782-japanese_spaniel', 'n02085936-maltese_dog', 'n02086079-pekinese', 'n02086240-shih-tzu', 'n02086646-blenh
  'n02087046-toy_terrier', 'n02087394-rhodesian_ridgeback', 'n02088094-afghan_hound', 'n02088238-basset', 'n02088364-beagle', 'n02088466-bl
  'n02089078-black-and-tan_coonhound', 'n02089867-walker_hound', 'n02089973-english_foxhound', 'n02090379-redbone', 'n02090622-borzoi', 'n0
  'n02091032-italian_greyhound', 'n02091134-whippet', 'n02091244-ibizan_hound', 'n02091467-norwegian_elkhound', 'n02091635-otterhound', 'n0
  'n02092339-weimaraner', 'n02093256-staffordshire_bullterrier', 'n02093428-american_staffordshire_terrier', 'n02093647-bedlington_terrier'
  'n02093859-kerry_blue_terrier', 'n02093991-irish_terrier', 'n02094114-norfolk_terrier', 'n02094258-norwich_terrier', 'n02094433-yorkshire
  'n02095570-lakeland_terrier', 'n02095889-sealyham_terrier', 'n02096051-airedale', 'n02096177-cairn', 'n02096294-australian_terrier', 'n02
  'n02097047-miniature_schnauzer', 'n02097130-giant_schnauzer', 'n02097209-standard_schnauzer', 'n02097298-scotch_terrier', 'n02097474-tibe
  'n02098105-soft-coated_wheaten_terrier', 'n02098286-west_highland_white_terrier', 'n02098413-lhasa', 'n02099267-flat-coated_retriever', '
  'n02099601-golden_retriever', 'n02099712-labrador_retriever', 'n02099849-chesapeake_bay_retriever', 'n02100236-german_short-haired_pointe
  'n02100735-english_setter', 'n02100877-irish_setter', 'n02101006-gordon_setter', 'n02101388-brittany_spaniel', 'n02101556-clumber', 'n021
  'n02102177-welsh_springer_spaniel', 'n02102318-cocker_spaniel', 'n02102480-sussex_spaniel', 'n02102973-irish_water_spaniel', 'n02104029-k
  'n02105056-groenendael', 'n02105162-malinois', 'n02105251-briard', 'n02105412-kelpie', 'n02105505-komondor', 'n02105641-old_english_sheep
  'n02106030-collie', 'n02106166-border_collie', 'n02106382-bouvier_des_flandres', 'n02106550-rottweiler', 'n02106662-german_shepherd', 'n0
  'n02107312-miniature_pinscher', 'n02107574-greater_swiss_mountain_dog', 'n02107683-bernese_mountain_dog', 'n02107908-appenzeller', 'n0210
  'n02108422-bull_mastiff', 'n02108551-tibetan_mastiff', 'n02108915-french_bulldog', 'n02109047-great_dane', 'n02109525-saint_bernard', 'n0
  'n02110185-siberian_husky', 'n02110627-affenpinscher', 'n02110806-basenji', 'n02110958-pug', 'n02111129-leonberg', 'n02111277-newfoundlan
  'n02111889-samoyed', 'n02112018-pomeranian', 'n02112137-chow', 'n02112350-keeshond', 'n02112706-brabancon_griffon', 'n02113023-pembroke',
  'n02113712-miniature_poodle', 'n02113799-standard_poodle', 'n02113978-mexican_hairless', 'n02115641-dingo', 'n02115913-dhole', 'n02116738
120
Epoch 1/5
525/525 [==============================] - 424s 808ms/step - loss: 1.1033 - accuracy: 0.7889 - val_loss: 0.7466 - val_accuracy: 0.8558
Epoch 2/5
525/525 [==============================] - 420s 800ms/step - loss: 0.5049 - accuracy: 0.8904 - val_loss: 0.8228 - val_accuracy: 0.8628
Epoch 3/5
525/525 [==============================] - 422s 803ms/step - loss: 0.3521 - accuracy: 0.9231 - val_loss: 0.8728 - val_accuracy: 0.8658
Epoch 1/5
160/525 [========>.....................] - ETA: 16:41 - loss: 2.0587 - accuracy: 0.4762
```

Demo

Results

Step 1: Saving model on h5 format after fitting and compiling.



model_compiled.h5

model.h5

Demo

Results

Step 2: Accuracy on test set

```
537/537 [==============================] - 76s 141ms/step - loss: 0.8236 - accuracy: 0.8480
```

Demo

Results

Step 2: Accuracy on test set

```
537/537 [==============================] - 76s 141ms/step - loss: 0.8236 - accuracy: 0.8480
```

Thanks for Listening!

Questions?