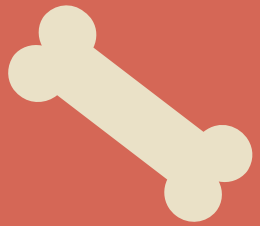
A Transfer Learning Approach to Fine-grained Dog Breed Classification

Sean Hansen
Jon Comisky



Overview

- Our project's source code
- The changes we made
- Our pipeline
- The training iterations we ran
- The output of our tests
- Our final model results

Source

https://github.com/nikhilroxtomar/Dog-Breed-Classifier-using-TF2.0/blob/master/train.py

```python
import os
import numpy as np
import pandas as pd
import cv2
from glob import glob

import tensorflow as tf
from tensorflow.keras.layers import *
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam

from sklearn.model_selection import train_test_split

def build_model(size, num_classes):
    inputs = Input((size, size, 3))
    backbone = MobileNetV2(input_tensor=inputs, include_top=False, weights="imagenet")
    backbone.trainable = True
    x = backbone.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.2)(x)
    x = Dense(1024, activation="relu")(x)
    x = Dense(num_classes, activation="softmax")(x)

    model = tf.keras.Model(inputs, x)
    return model

def read_image(path, size):
    image = cv2.imread(path, cv2.IMREAD_COLOR)
    image = cv2.resize(image, (size, size))
    image = image / 255.0
    image = image.astype(np.float32)
    return image

def parse_data(x, y):
    x = x.decode()

    num_class = 120
    size = 224

    image = read_image(x, size)
    label = [0] * num_class
    label[y] = 1
    label = np.array(label)
    label = label.astype(np.int32)

    return image, label

def tf_parse(x, y):
    x, y = tf.numpy_function(parse_data, [x, y], [tf.float32, tf.int32])
    x.set_shape((224, 224, 3))
    y.set_shape((120))
    return x, y

def tf_dataset(x, y, batch=8):
    dataset = tf.data.Dataset.from_tensor_slices((x, y))
    dataset = dataset.map(tf_parse)
    dataset = dataset.batch(batch)
    dataset = dataset.repeat()
    return dataset

if __name__ == "__main__":
    path = "Dog Breed Identification/"
    train_path = os.path.join(path, "train/*")
    test_path = os.path.join(path, "test/*")
    labels_path = os.path.join(path, "labels.csv")

    labels_df = pd.read_csv(labels_path)
    breed = labels_df["breed"].unique()
    print("Number of Breed: ", len(breed))

    breed2id = {name: i for i, name in enumerate(breed)}

    ids = glob(train_path)
    labels = []

    for image_id in ids:
        image_id = image_id.split("/")[-1].split(".")[0]
        breed_name = list(labels_df[labels_df.id == image_id]["breed"])[0]
        breed_idx = breed2id[breed_name]
        labels.append(breed_idx)

    ids = ids[:1000]
    labels = labels[:1000]

    ## Spliting the dataset
    train_x, valid_x = train_test_split(ids, test_size=0.2, random_state=42)
    train_y, valid_y = train_test_split(labels, test_size=0.2, random_state=42)

    ## Parameters
    size = 224
    num_classes = 120
    lr = 1e-4
    batch = 16
    epochs = 10

    ## Model
    model = build_model(size, num_classes)
    model.compile(loss="categorical_crossentropy", optimizer=Adam(lr), metrics=["acc"])
    # model.summary()

    ## Dataset
    train_dataset = tf_dataset(train_x, train_y, batch=batch)
    valid_dataset = tf_dataset(valid_x, valid_y, batch=batch)

    ## Training
    callbacks = [
        ModelCheckpoint("model.h5", verbose=1, save_best_only=True),
        ReduceLROnPlateau(factor=0.1, patience=5, min_lr=1e-6)
    ]
    train_steps = (len(train_x)//batch) + 1
    valid_steps = (len(valid_x)//batch) + 1
    model.fit(train_dataset,
        steps_per_epoch=train_steps,
        validation_steps=valid_steps,
        validation_data=valid_dataset,
        epochs=epochs,
        callbacks=callbacks)
```

Training

Testing

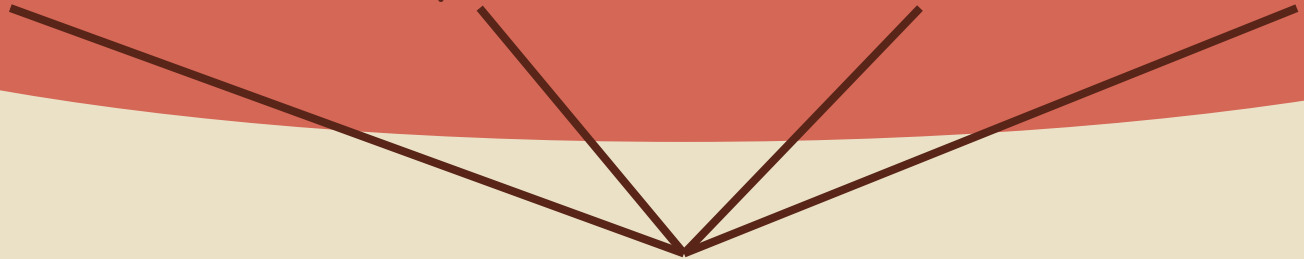
```python
import os
import numpy as np
import pandas as pd
import cv2
from glob import glob
from tqdm import tqdm
import tensorflow as tf
from sklearn.model_selection import train_test_split

def read_image(path, size):
    image = cv2.imread(path, cv2.IMREAD_COLOR)
    image = cv2.resize(image, (size, size))
    image = image / 255.0
    image = image.astype(np.float32)
    return image

if __name__ == "__main__":
    path = "Dog Breed Identification/"
    train_path = os.path.join(path, "train/*")
    test_path = os.path.join(path, "test/*")
    labels_path = os.path.join(path, "labels.csv")

    labels_df = pd.read_csv(labels_path)
    breed = labels_df["breed"].unique()
    print("Number of Breed: ", len(breed))

    breed2id = {name: i for i, name in enumerate(breed)}
    id2breed = {i: name for i, name in enumerate(breed)}

    ids = glob(train_path)
    labels = []

    for image_id in ids:
        image_id = image_id.split("/")[-1].split(".")[0]
        breed_name = list(labels_df[labels_df.id == image_id]["breed"])[0]
        breed_idx = breed2id[breed_name]
        labels.append(breed_idx)

    ids = ids[:1000]
    labels = labels[:1000]

    ## Spliting the dataset
    train_x, valid_x = train_test_split(ids, test_size=0.2, random_state=42)
    train_y, valid_y = train_test_split(labels, test_size=0.2, random_state=42)

    ## Model
    model = tf.keras.models.load_model("model.h5")

    for i, path in tqdm(enumerate(valid_x[:10])):
        image = read_image(path, 224)
        image = np.expand_dims(image, axis=0)
        pred = model.predict(image)[0]
        label_idx = np.argmax(pred)
        breed_name = id2breed[label_idx]

        ori_breed = id2breed[valid_y[i]]
        ori_image = cv2.imread(path, cv2.IMREAD_COLOR)

        ori_image = cv2.putText(ori_image, breed_name, (0, 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 1)
        ori_image = cv2.putText(ori_image, ori_breed, (0, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)

        cv2.imwrite(f"save/valid_{i}.png", ori_image)
```

Our changes

Our repository

Pipelines

ResNet50 Xception DenseNet201 VGG19



Dropout(0.25)(x)

BatchNormalization()(x)

Dropout(0.25)(x)

GlobalAveragePooling2D()(x)

Dropout(0.25)(x)

Dense(1024, activity_regularizer=regularizers.l2())(x)

Dropout(0.25)(x)

Dense(num_classes, activation="softmax", activity_regularizer=regularizers.l2())(x)

Flatten()(x)

Data Split

90% Train 18,522

10% Validation 2,058

Test

Train

20 Epochs; No Data Augmentation;
Image-Net; LR = 1e-1

Lower LR

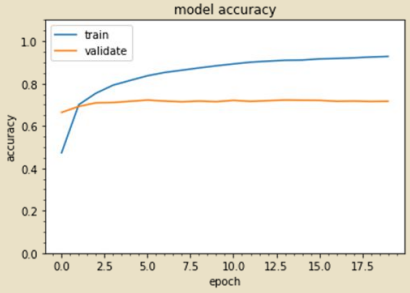
20 Epochs; No Data Augmentation;
Train Callback; LR = 1e-2

Fine Tune

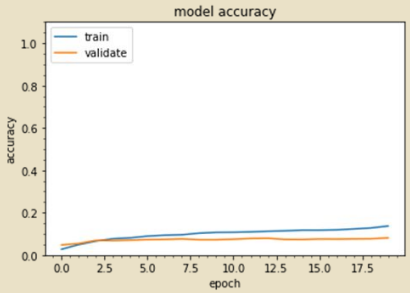
20 Epochs; No Data Augmentation;
Lower LR Callback; LR = 1e-2

Training Accuracy

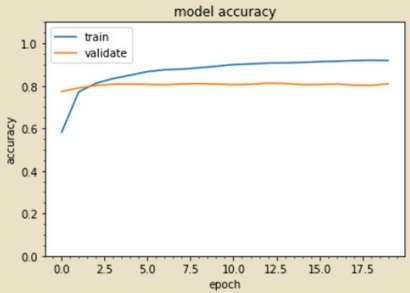
ResNet50



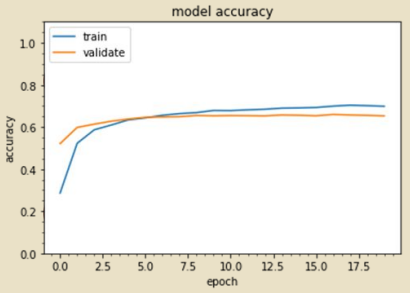
Xception



DenseNet201

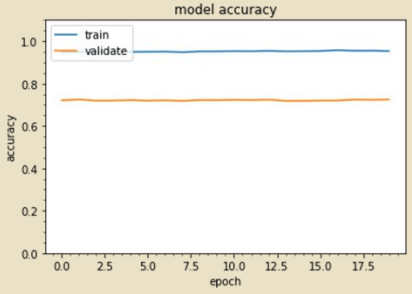


VGG19

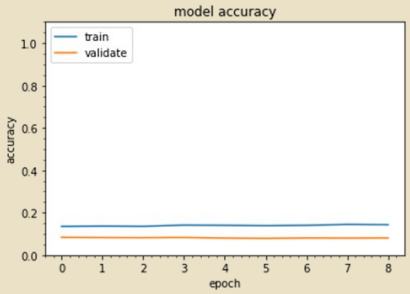


Lower LR Accuracy

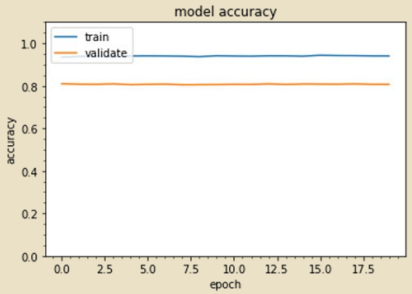
ResNet50



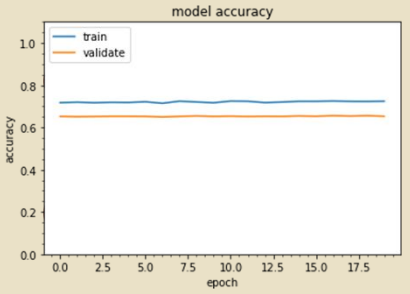
Xception



DenseNet201



VGG19



Overall Accuracy

ResNet50

Train:
92.79% -> 95.30%

Validate:
71.62% -> 72.50%

Xception

Train:
13.70% -> 8.11%

Validate:
14.31% -> 8.07%

DenseNet201

Train:
91.92% -> 94.05%

Validate:
80.95% -> 80.71%

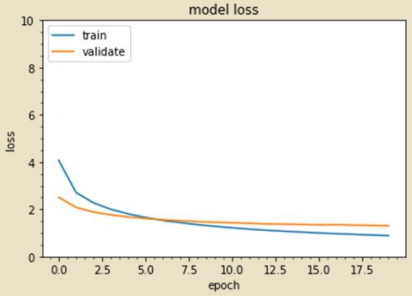
VGG19

Train:
69.87% -> 72.44%

Validate:
65.31% -> 65.31%

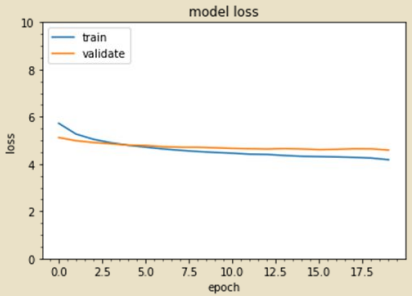
Overall Loss

ResNet50



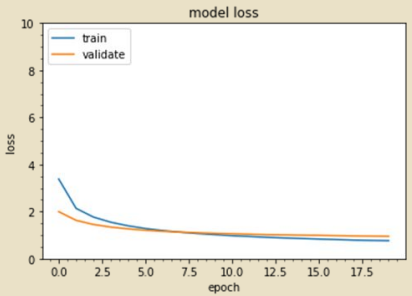
Train: 0.8832
Validate: 1.3031

Xception



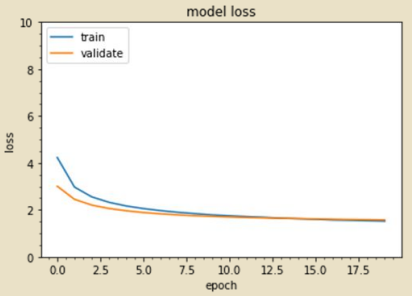
Train: 4.1828
Validate: 4.5909

DenseNet201



Train: 0.7611
Validate: 0.9486

VGG19



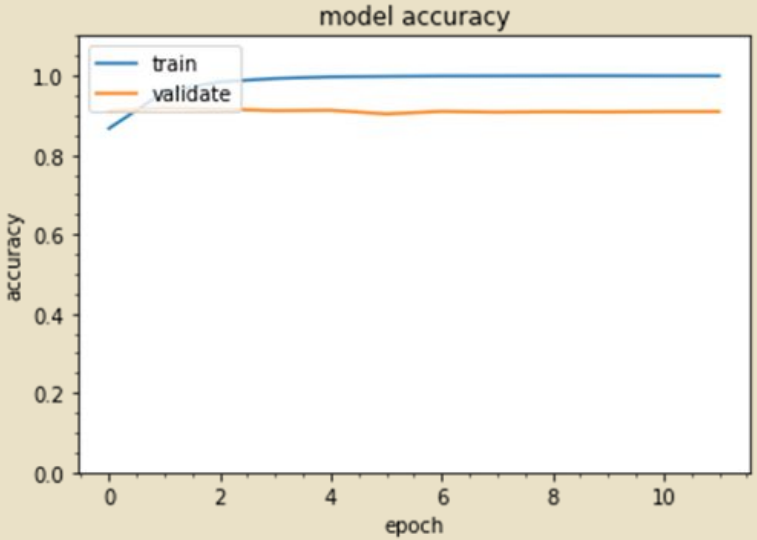
Train: 1.5191
Validate: 1.5720

Fine Tune

99.9% Train

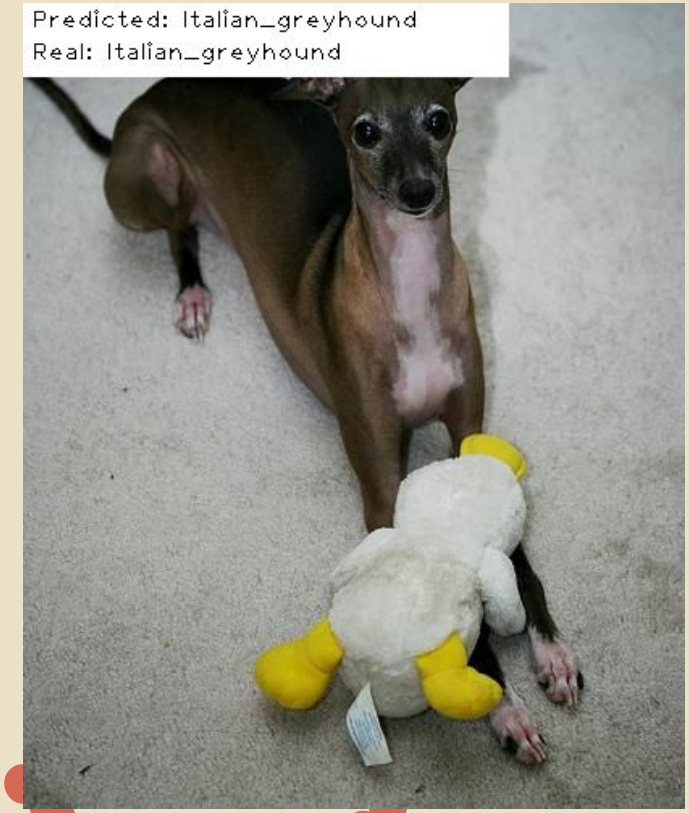
90.9% Validate

98.0% Test



Loss stopped improving after 12 epochs

Results


Predicted: Italian_greyhound
Real: Italian_greyhound


Predicted: Italian_greyhound
Real: Italian_greyhound

Results


Predicted: basset
Real: basset


Predicted: Walker_hound
Real: Walker_hound

Results


Predicted: Newfoundland
Real: Newfoundland


Predicted: schipperke
Real: schipperke

Results


Predicted: curly_coated_retriever
Real: curly_coated_retriever


Predicted: curly_coa
Real: curly_coated_r


Predicted: Labrador_retriever
Real: Labrador_retriever

Results

Conclusions

DenseNet201 = Best Model

Transfer Learning

More Training != Better Results

Fine Tuning = Better Results

Testing Results > Validation