Carlotta Polidoro

Micol Cozzarizza

Breedify{ Dog breeds classification

[with Machine Learning and Deep Learning]

}

Introduction

Dogs are one of the most successful and diverse species that populate our planet, in fact, there exist between 195 and 500 dog breeds around the world.

Many of these breeds are totally **different** from one another, while many other present remarkable **similarity**, and are considered to be different just for some subtle details.

For these reasons, the classification of dogs into different breeds is a **very challenging task.**

In this project we tried two different approaches based on Machine Learning, and Deep Learning respectively, to understand which was the most effective to tackle this challenge.

Table Of Contents {

01 {Data Collection

Our data was collected from a publicly available dataset that we found for free on Kaggle.

Specifically, we used the Stanford Dogs Dataset, which contains 20,580 images of 120 dogs from around the world.

Machine Learning with Feature Descriptors{

< In order to build our Machine Learning model, we used the **scikit-learn** library.

We shuffled our data, and split them in training (60%), validation (10%), and test (30%) sets>

```
[ ] np.random.shuffle(images)
```

```
[ ] trainset = images[:int(0.6*len(images))]
    valset = images[int(0.6*len(images)):int(0.7*len(images))]
    testset = images[int(0.7*len(images)):]

    print('Total: {} splitted in Train: {}, Val: {} and Test: {}'.format(len(images), len(trainset), len(valset), len(testset)))
```

```
Total: 20580 splitted in Train: 12348, Val: 2057 and Test: 6175
```

Feature Descriptors{

We used 2 visual Feature Descriptors: **HOG and LBP**, and defined the extract_features() function to perform feature extraction for each image:

```python
def extract_features(images, feat_type, img_size):

    labels = []
    features = []

    for image in images:

        # open the image
        img = cv2.imread(image, 0)

        # resize the image
        img = cv2.resize(img, (img_size, img_size))

        # compute the features
        if feat_type == 'hog':
            feat = hog(img, orientations=8, pixels_per_cell=(4, 4), cells_per_block=(1, 1))
        elif feat_type == 'lbp':
            feat = np.ravel(local_binary_pattern(img, P=100, R=5))
        else:
            raise NotImplementedError('Not implemented feature!')

        # append features and labels
        features.append(feat)

        labels.append(get_labels(image))

    return features, labels
```

ML Classifiers{

For the classification task, we used 3 different classifiers: **Support Vector Machines**, **Random Forest**, and **AdaBoost**.

We defined the **decide(clf)** function, which selects one classifier at a time and runs the classification with different features and image size combinations:

```
def decide_clf (clf):
  if clf == "SVM":
    clf = svm.SVC(gamma=0.001, C=100., kernel='linear', class_weight='balanced', verbose=False, probability=False)
  elif clf == "RF":
    clf = RandomForestClassifier(n_estimators=100, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
  elif clf == "Adaboost":
    clf = AdaBoostClassifier(estimator=None,n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None)
  return clf
```

ML Classifiers{

```
e {}'.format(el, feature_type, img_size, accuracy_score(test_y, y_pred), t2-t1))
```

```
sizes = [32, 64, 128]
features = ['hog','lbp']
types_clf = ["SVM", "RF", "Adaboost"]

def decide_clf (clf):
  if clf == "SVM":
    clf = svm.SVC(gamma=0.001, C=100., kernel='linear', class_weight='balanced', verbose=False, probability=False)
  elif clf == "RF":
    clf = RandomForestClassifier(n_estimators=100, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fracti
  elif clf == "Adaboost":
    clf = AdaBoostClassifier(estimator=None,n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None)
  return clf

for img_size in sizes:
  for feature_type in features:
    t1 = time.time()
    train_x, train_y = extract_features(trainset, feature_type, img_size)     feature extraction
    val_x, val_y = extract_features(valset, feature_type, img_size)
    test_x, test_y = extract_features(testset, feature_type, img_size)
    for el in types_clf:
      clf = decide_clf(el)
      clf.fit(train_x, train_y)                                                 training, validation, and prediction
      clf.score(val_x, val_y)
      y_pred = clf.predict(test_x) #to predict the class of the given data.
      t2 = time.time()
      print('Final Accuracy for {} classifier with {} feature descriptor and {} image size: {:.3f}. Elapsed time {}'.format(el, feature_type, i
```

SVM Accuracy Results {

Feature	Image size	Accuracy	Elapsed time
HOG	32x32	0.021	153.256
LBP	32x32	0.016	450.756
HOG	64x64	0.032	242.656
LBP	64x64	0.016	1564.398
HOG	**128X128**	**0.041**	**690.136**
LBP	128x128	0.018	7812.561

}

Random Forest Accuracy Results {

Feature	Image size	Accuracy	Elapsed time
HOG	32x32	0.020	212.769
LBP	32x32	0.021	712.115
HOG	64x64	0.018	543.993
LBP	64x64	0.015	2100.234
HOG	128x128	0.018	1802.291
LBP	128x128	0.016	9012.543

}

AdaBoost Accuracy Results {

Feature	Image size	Accuracy	Elapsed time
HOG	32x32	0.012	253.476
LBP	32x32	0.014	923.209
HOG	64x64	0.015	670.137
LBP	64x64	0.011	2910.388
HOG	128x128	0.015	2456.893
LBP	128x128	0.011	12546.854

}

Deep Learning with Convolutional Neural Networks{

For our Deep Learning model, we used the **TensorFlow** library with the Keras interface.

To define the architecture of our Convolutional Neural Network, we used the **MobileNet CNN**, adapted to our case.

```python
# Our model
model = tf.keras.applications.MobileNet(
    input_shape=None,
    alpha=1.0,
    depth_multiplier=1,
    dropout=0.001,
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax"
    )
```

```python
# here we create the new Dense layer where input is the output of the second last layer
output = Dense(120, activation='softmax', name='predictions')(model.layers[-2].output)

# Then create the corresponding model
model = Model(model.input, output)
```

CNN code{

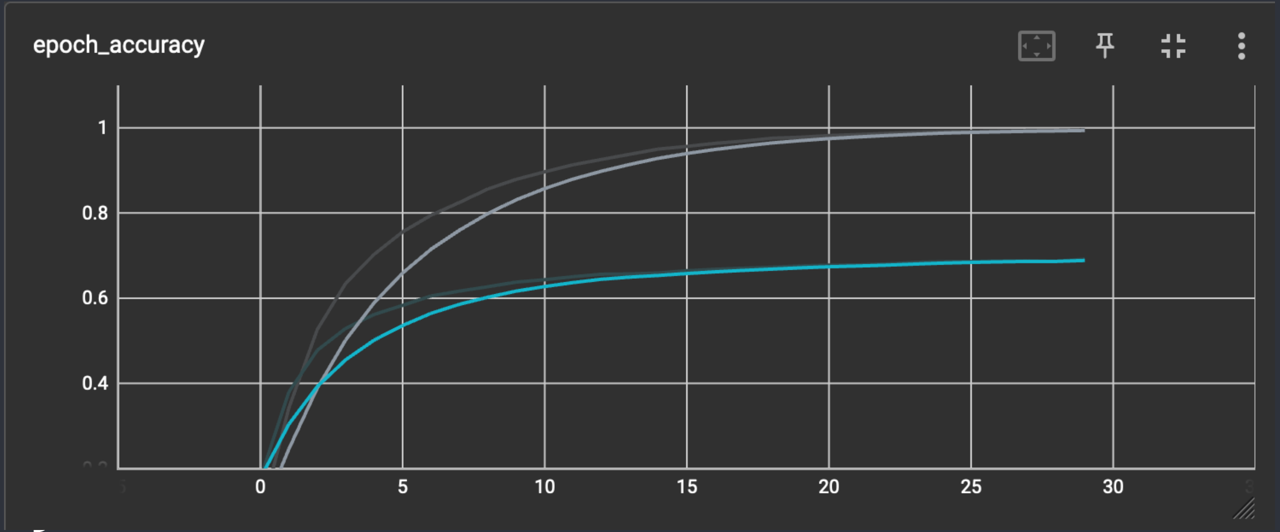
```
epochs = 30

callbacks = [
    keras.callbacks.ModelCheckpoint("save_at_{epoch}.h30"),
    tf.keras.callbacks.TensorBoard(log_dir="logs", write_graph=True, write_images=False, update_freq="epoch",)
]

model.compile(
    optimizer=keras.optimizers.SGD(1e-3),
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)
```

Here we define the epochs, callbacks, optimizers and loss function.

The final results were: 63s 191ms/step - **loss: 0.3028** - **accuracy: 0.9307**

CNN Results{

Final Accuracy: 0.9307



CNN Results{

Loss: 0.3028