Dog breed recognition algorithms

Lăzărescu Andreea

Moldovan Mihaela

Why dogs?

This field has gotten a lot of attention because of its practical applications in areas like pet adoption, veterinary medicine, and canine research.

Accurate breed recognition assists in behavior prediction, health assessment, and customized care.

Manual recognition can be challenging due to the vast number of breeds.

Identifying a dog’s breed from an image requires the algorithm to learn and distinguish subtle differences between breeds.

Norfolk Terrier

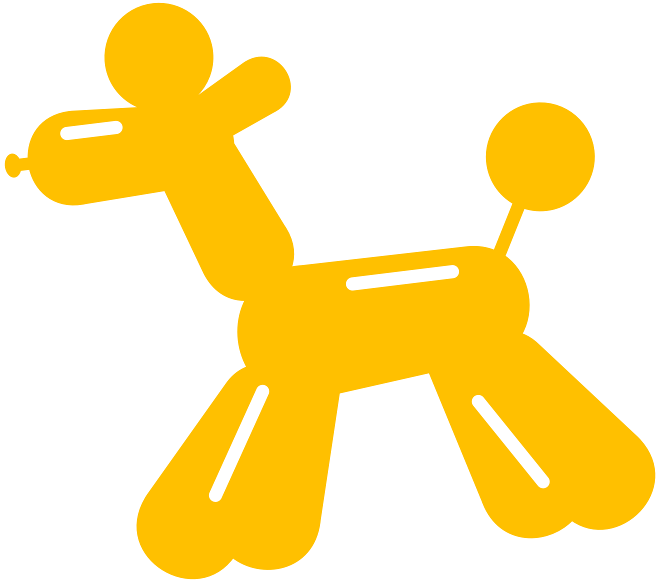
Norwich Terrier

Related work

Many studies have been done to advance the state of the art in dog breed recognition algorithms.

Aspects including dataset development, algorithm design, feature extraction, and performance evaluation criteria have all been the subject of prior research.

[Kaggle](#) has hosted a competition to practice fine-grained image categorization where they offered a dataset with 120 breeds of dogs and a limited number training images per class.



Data

- The dataset used in the experiment is a subset of ImageNet, specifically curated for fine-grained image categorization of dogs.
- The dataset consists of a training set and a test set of dog images. The training set contains a limited number of images per dog breed, with a total of 120 different breeds.
- Each image in the dataset is assigned a unique ID, which corresponds to its filename.

Algorithm – Feature Extraction

- We used multiple pre-trained models: InceptionV3, Xception, InceptionResNetV2, NASNetLarge.
- For each of them we extracted the features and then we concatenated them into a future map.

```python
def get_features(model_name, model_preprocessor, input_size, data):

    input_layer = Input(input_size)
    preprocessor = Lambda(model_preprocessor)(input_layer)
    base_model = model_name(weights='imagenet', include_top=False,
                            input_shape=input_size)(preprocessor)
    avg = GlobalAveragePooling2D()(base_model)
    feature_extractor = Model(inputs = input_layer, outputs = avg)

    #Extract feature.
    feature_maps = feature_extractor.predict(data, verbose=1)
    print('Feature maps shape: ', feature_maps.shape)
    return feature_maps
```

```python
# Extract features using InceptionV3
from keras.applications.inception_v3 import InceptionV3, preprocess_input
inception_preprocessor = preprocess_input
inception_features = get_features(InceptionV3,
                                  inception_preprocessor,
                                  img_size, X)
```

```
320/320 [==============================] - 711s 2s/step
Feature maps shape:  (10222, 2048)
```

```python
# Extract features using Xception
from keras.applications.xception import Xception, preprocess_input
xception_preprocessor = preprocess_input
xception_features = get_features(Xception,
                                 xception_preprocessor,
                                 img_size, X)
```

```
320/320 [==============================] - 1229s 4s/step
Feature maps shape:  (10222, 2048)
```

```python
# Extract features using InceptionResNetV2
from keras.applications.inception_resnet_v2 import InceptionResNetV2, preprocess_input
inc_resnet_preprocessor = preprocess_input
inc_resnet_features = get_features(InceptionResNetV2,
                                   inc_resnet_preprocessor,
                                   img_size, X)
```

```
320/320 [==============================] - 1782s 6s/step
Feature maps shape:  (10222, 1536)
```

```python
# Extract features using NASNetLarge
from keras.applications.nasnet import NASNetLarge, preprocess_input
nasnet_preprocessor = preprocess_input
nasnet_features = get_features(NASNetLarge,
                               nasnet_preprocessor,
                               img_size, X)
```

```
320/320 [==============================] - 3827s 12s/step
Feature maps shape:  (10222, 4032)
```

```python
#Creating final featuremap by combining all extracted features

final_features = np.concatenate([inception_features,
                                 xception_features,
                                 nasnet_features,
                                 inc_resnet_features,], axis=-1) #axis=-1 to concatinate horizontally

print('Final feature maps shape', final_features.shape)
```

```
Final feature maps shape (10222, 9664)
```

Algorithm – Model

- We used a sequential model

Hyperparameters are:

- batch_size= 128
- epochs= 50
- learn_rate= .001
- sgd=SGD(learning_rate=learn_rate,

momentum=.9,

nesterov=False)

- adam=Adam(

learning_rate=learn_rate,

beta_1=0.9,

beta_2=0.999,

epsilon=None,

amsgrad=False)

```
#Prepare Deep net

model = Sequential()
# model.add(Dense(1028,input_shape=(final_features.shape[1],)))
model.add(Dropout(0.7,input_shape=(final_features.shape[1],)))
model.add(Dense(n_classes,activation= 'softmax'))

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

#Training the model.
history = model.fit(final_features, y,
            batch_size=batch_size,
            epochs=epochs,
            validation_split=0.2,
            callbacks=[lrr,EarlyStop])
```

Model accuracy

- The performance of the trained model in predicting the dog’s breed was evaluated using several metrics, including accuracy, precision, recall, and F1-score.
- The obtained results demonstrate that the trained model exhibits strong performance in identifying the dog’s breed, achieving high accuracy and precision, as well as maintaining a high recall rate.

Metric	Value
Accuracy	0.93
Precision	0.9465
Recall	0.93
F1-Score	0.9303


```
[ ] Image('golden.jpg')
```

```
[ ] #reading the image and converting it into an np array

    img_g = load_img('golden.jpg',target_size = img_size)
    img_g = np.expand_dims(img_g, axis=0) # as we trained our model in (row,
    # img_g
```

```
[ ] img_g.shape

    (1, 331, 331, 3)
```

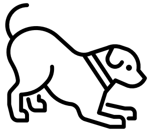
```
[ ] # #Predict test labels given test data features.
    test_features = extact_features(img_g)
    predg = model.predict(test_features)
    print(f"Predicted label: {classes[np.argmax(predg[0])]}")
    print(f"Probability of prediction): {round(np.max(predg[0])) * 100} %")

    1/1 [==============================] - 1s 1s/step
    Feature maps shape:  (1, 2048)
    1/1 [==============================] - 1s 972ms/step
    Feature maps shape:  (1, 2048)
    1/1 [==============================] - 6s 6s/step
    Feature maps shape:  (1, 4032)
    WARNING:tensorflow:5 out of the last 328 calls to <function Model.make_pr
    1/1 [==============================] - 3s 3s/step
    Feature maps shape:  (1, 1536)
    Final feature maps shape (1, 9664)
    1/1 [==============================] - 0s 22ms/step
    Predicted label: golden_retriever
    Probability of prediction): 100 %
```

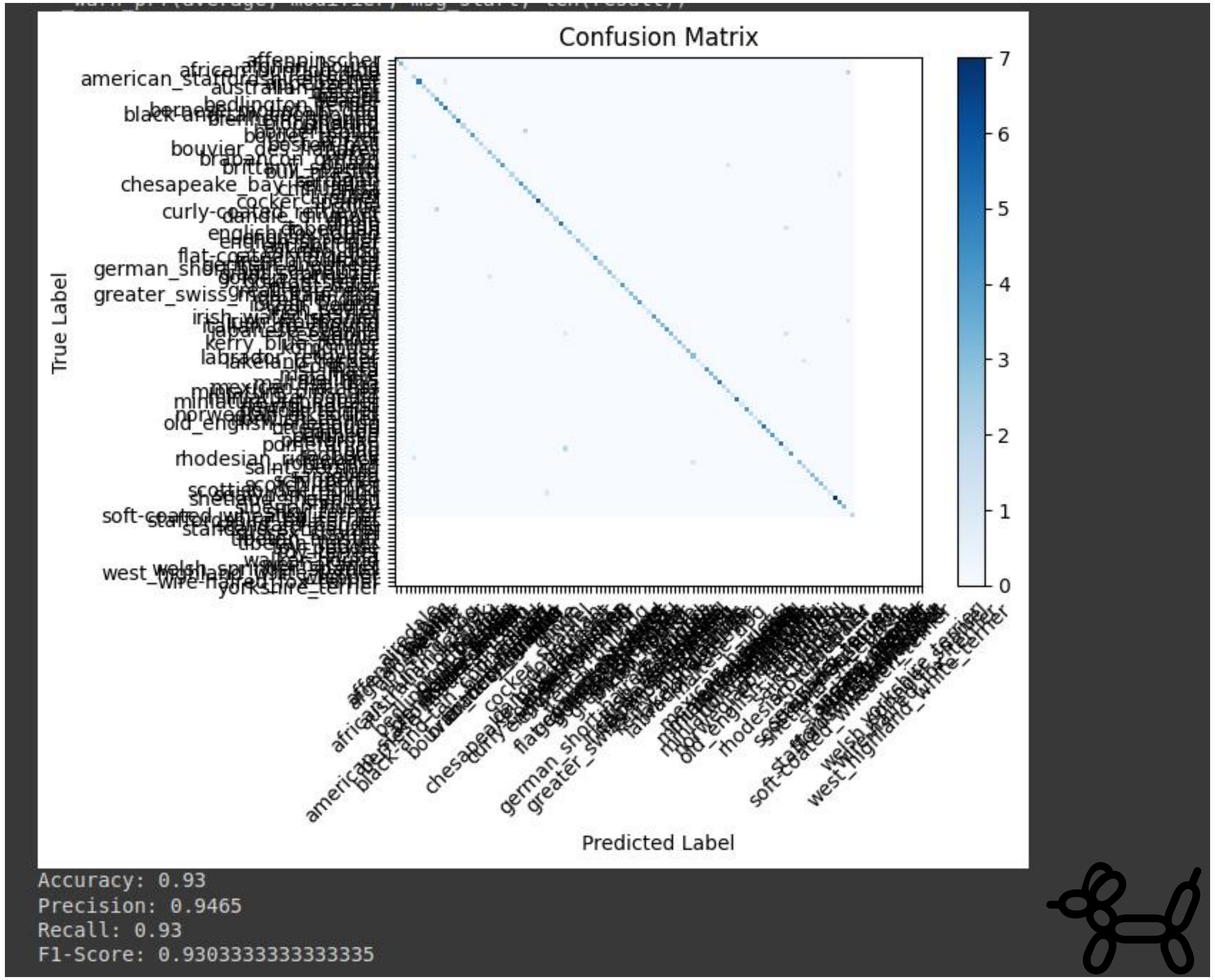
Results

This finding is supported by a comparison with other models, as discussed in "A new dataset of dog breed images and a benchmark for fine-grained classification"

Our research has shown that using an ensemble of multiple models, such as InceptionV3, Xception, InceptionResNetV2, and NASNetLarge, results in better performance than using a single model.



Model	Backbone	Batchsize	Epochs	Accuracy
Inception V3	-	64	200	77.66%
WS-DAN	Inception	12	80	86.404%
PMG	ResNet50	16	200	83.52%
TBMSL-Net	ResNet50	6	200	83.7%
Ours	InceptionV3, Xception, InceptionResNetV2, NASNetLarge	128	50	93%



Conclusion

- **The ensemble method takes advantage of the diversity and complementary strengths of individual models, resulting in improved accuracy and robustness in dog breed classification.**
- **We mitigate the limitations and biases inherent in any single model by combining predictions from multiple models, resulting in more reliable and accurate predictions.**
- **Further research can look into different model combinations, ensemble strategies, and architectural variations to improve performance even more**