Presentation of BSIS NS 3A - Group 5

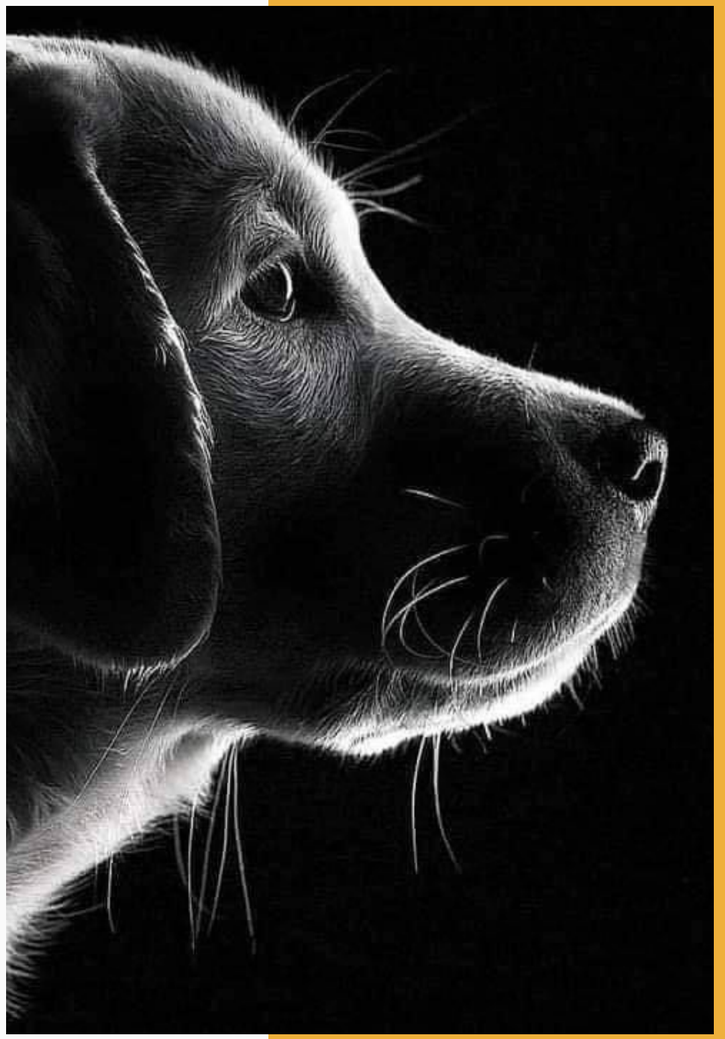
An Image Classifier on Dog Breeds using Deep Learning

FINAL PROJECT

IS Professional Elective 3: Data Mining

2023

Overview



Introduction

- The goal is to train a model using Convolutional Neural Network (CNN) Algorithm to recognize images of dogs in datasets.
- The model will predict dog breed using deep learning in Python.
- After analyzing the input image, the breed will be predicted.
- The model will be extended on Anvil Web App Builder

Review of Related Literature

Recognition of dog breeds has been helpful in many ways. The researchers use two references to solely help with the identification of dog breeds. These references are: Dog Breed Identification Using Deep Learning, and Knowing Your Dog Breed: Identifying a Dog Breed with Deep Learning. These two references help students choose what approach, language, and algorithm they are going to use. Hence, students decided to use Tensorflow Keras API for approach, Python for programming language, and CNN for algorithm.

Review of Related Literature

Dog Breed Identification Using Deep Learning

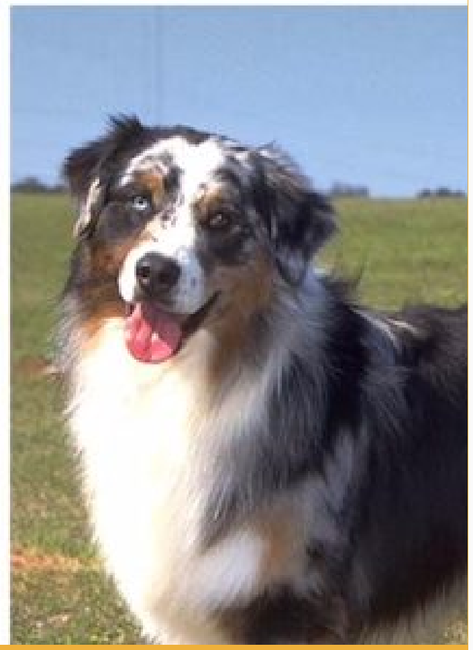
The current paper presents a fine-grained image recognition problem, one of multi-class classification, namely determining the breed of a dog in a given image. The presented system employs innovative methods in deep learning, including convolutional neural networks. Two different networks are trained 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.

A. Varshney, A. Katiyar, A. K. Singh and S. S. Chauhan, "Dog Breed Classification Using Deep Learning," 2021 International Conference on Intelligent Technologies (CONIT), 2021, pp. 1-5, doi: 10.1109/CONIT51480.2021.9498338. https://ieeexplore.ieee.org/document/9498338

Knowing Your Dog Breed: Identifying a Dog Breed with Deep Learning

In the paper, the authors have proposed two models to classify dogs according to their breeds. During the study, they came across many types of functionality levels that were not taken in previous studies too. Also, the approach also works on the main concept of transfer learning which deals with data augmentation technique with its properties to increase the size of data set, after which accuracy levels are matched or it is compared with both the models so that a comparison can be made for both the models and the classification is also done with a profound approach

*Z. Ráduly, C. Sulyok, Z. Vadászi and A. Zölde, "Dog Breed Identification Using Deep Learning," 2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY), 2018, pp. 000271-000276, doi: 10.1109/SISY.2018.8524715.*https://ieeexplore.ieee.org/document/8524715



Implementation

Deep Learning Model (DenseNet 121)

We used DenseNet121 architecture to identify the breed of a dog. The DenseNet121 is a popular deep learning architecture used in image classification tasks. It is a convolutional neural network that uses dense connections to improve the flow of information through the network.

Python (Programming Language)

Python is used because of its simplicity, readability, and ease of use. In addition, it is also because of libraries and tools. Python is used because it can handle repeated prototyping, which makes it easier for researchers to test and train different models.

Keras Python

The TensorFlow Keras is used in defining and training deep learning models. This makes it easier for researchers to train the model for image classification tasks.

Methodology

1. Data Cleaning and Breed Labelling

The first step in this process is to prepare the dataset. The dataset should contain images of dogs and their corresponding breed labels.

how many breeds and pictures we have

```
[ ]  breed_list = os.listdir("../content/dogbreedidentification/images/Images/")

     num_classes = len(breed_list)
     print("{} breeds".format(num_classes))

     n_total_images = 0
     for breed in breed_list:
         n_total_images += len(os.listdir("../content/dogbreedidentification/images/Images/{}".format(breed)))
     print("{} images".format(n_total_images))
```

```
63 breeds
10810 images
```

label strings and numbers mapping

```
[ ]  label_maps = {}
     label_maps_rev = {}
     for i, v in enumerate(breed_list):
         label_maps.update({v: i})
         label_maps_rev.update({i : v})
```

Methodology

2. Data Preprocessing

- Resize images to uniform size to focus on the dog
- Crop images to show only the dog
- Resize cropped images to uniform size of 224x224 and store in separate folders for each breed
- Transform breed labels into numerical labels for use as target values in training process

cropping pics

```
%%time

# copy from https://www.kaggle.com/gabrielloye/dogs-inception-pytorch-implementation
# reduce the background noise

os.mkdir('data')
for breed in breed_list:
    os.mkdir('data/' + breed)
print('Created {} folders to store cropped images of the different breeds.'.format(len(os.listdir('data'))))

for breed in os.listdir('data'):
    for file in os.listdir('/content/dogbreedidentification/annotations/Annotation/{}'.format(breed)):
        img = Image.open('/content/dogbreedidentification/images/Images/{}/{}.jpg'.format(breed, file))
        tree = ET.parse('/content/dogbreedidentification/annotations/Annotation/{}/{}'.format(breed, file))
        xmin = int(tree.getroot().findall('object')[0].find('bndbox').find('xmin').text)
        xmax = int(tree.getroot().findall('object')[0].find('bndbox').find('xmax').text)
        ymin = int(tree.getroot().findall('object')[0].find('bndbox').find('ymin').text)
        ymax = int(tree.getroot().findall('object')[0].find('bndbox').find('ymax').text)
        img = img.crop((xmin, ymin, xmax, ymax))
        img = img.convert('RGB')
        img = img.resize((224, 224))
        img.save('data/' + breed + '/' + file + '.jpg')
```

```
Created 63 folders to store cropped images of the different breeds.
CPU times: user 1min 6s, sys: 1.26 s, total: 1min 7s
Wall time: 1min 7s
```

Methodology

3. Data Augmentation

image generator with augment

```
[ ]  batch_size = 32

     class ImageGenerator(Sequence):

         def __init__(self, paths, targets, batch_size, shape, augment=False):
             self.paths = paths
             self.targets = targets
             self.batch_size = batch_size
             self.shape = shape
             self.augment = augment

         def __len__(self):
             return int(np.ceil(len(self.paths) / float(self.batch_size)))

         def __getitem__(self, idx):
             batch_paths = self.paths[idx * self.batch_size : (idx + 1) * self.batch_size]
             x = np.zeros((len(batch_paths), self.shape[0], self.shape[1], self.shape[2]), dtype=np.float32)
             y = np.zeros((self.batch_size, num_classes, 1))
             for i, path in enumerate(batch_paths):
                 x[i] = self.__load_image(path)
             y = self.targets[idx * self.batch_size : (idx + 1) * self.batch_size]
             return x, y

         def __iter__(self):
             for item in (self[i] for i in range(len(self))):
                 yield item

         def __load_image(self, path):
             image = imread(path)
             image = preprocess_input(image)
             if self.augment:
                 seq = iaa.Sequential([
                     iaa.OneOf([
                         iaa.Fliplr(0.5),
                         iaa.Flipud(0.5),
                         iaa.CropAndPad(percent=(-0.25, 0.25)),
                         iaa.Crop(percent=(0, 0.1)),
                         iaa.Sometimes(0.5,
                             iaa.GaussianBlur(sigma=(0, 0.5))
                         ),
                         iaa.Affine(
                             scale={"x": (0.8, 1.2), "y": (0.8, 1.2)},
                             translate_percent={"x": (-0.2, 0.2), "y": (-0.2, 0.2)},
                             rotate=(-180, 180),
                             shear=(-8, 8)
                         )
                     ])
                 ], random_order=True)
                 image = seq.augment_image(image)
             return image
```

Methodology

4. Model Selection

DenseNet121 model selected for its capability in handling image classification tasks

Dense Convolutional Network (DenseNet), connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections - one between each layer and its subsequent layer - our network has L(L+1)/2 direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters.

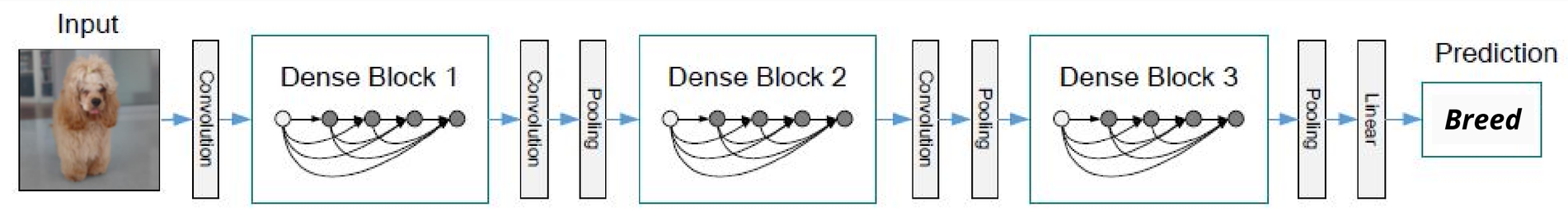
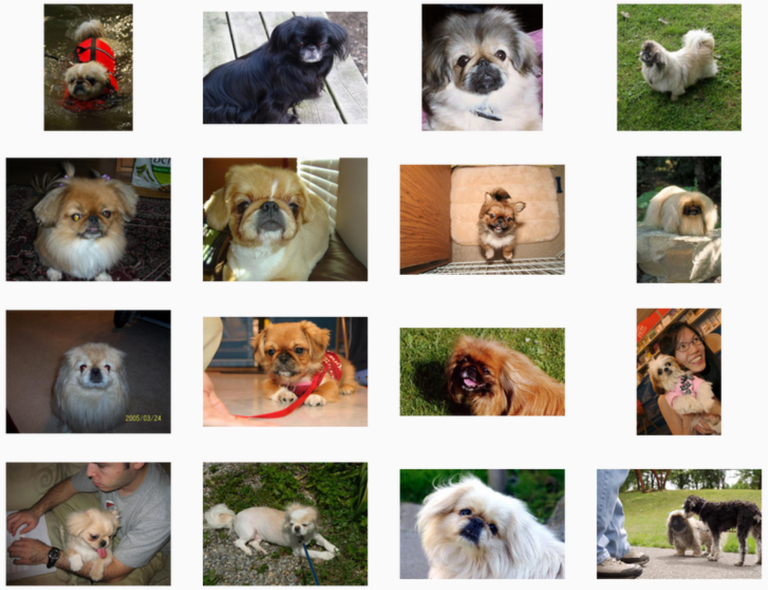


Figure 1: DesNet CNN Layers

Methodology

5. Model Fine-tuning

The pre-trained DenseNet121 model is fine-tuned by retraining the last few layers to adapt the model to the specific dog breed dataset. This step is done to ensure that the model can perform well on the specific dataset.



keras pretrain densenet121 model

```
[ ]  inp = Input((224, 224, 3))
     backbone = DenseNet121(input_tensor=inp,
                            weights='/content/DenseNet-BC-121-32-no-top.h5',
                            include_top=False)
     x = backbone.output
     x = GlobalAveragePooling2D()(x)
     x = Dense(1024, activation="relu")(x)
     x = Dropout(0.5)(x)
     x = Dense(512, activation="relu")(x)
     x = Dropout(0.5)(x)
     outp = Dense(num_classes, activation="softmax")(x)

     model = Model(inp, outp)
```

```
[ ]  for layer in model.layers[:-6]:
         layer.trainable = False
```

Methodology

6. Model Optimizing

Compiling the pre-trained model using the Adam optimization algorithm and using accuracy as the evaluation metric.

```
[ ]  model.compile(optimizer="adam",
                   loss="categorical_crossentropy",
                   metrics=["acc"])
```

7. Training the Model

The final step is to train all the model on the prepared dataset. The model should be trained for a number of epochs (25 to 50), with a batch size of (32,64). After training, the model can be evaluated on a validation set to assess its performance. The model can then be used to predict the breed of new dogs.

```
[ ]  history = model.fit_generator(generator=train_gen,
                                   steps_per_epoch=len(train_gen),
                                   validation_data=val_gen,
                                   validation_steps=len(val_gen),
                                   epochs=1)
```

Methodology

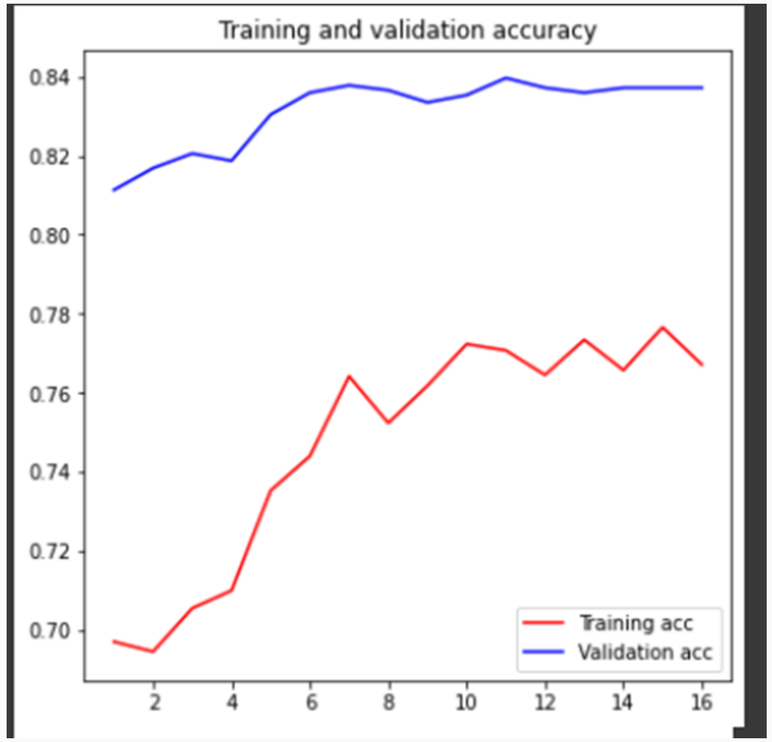
```
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
144/144 [==============================] - 49s 338ms/step - loss: 0.9197 - acc: 0.7098 - val_loss: 0.5591 - val_acc: 0.8187 - lr: 0.0010
Epoch 5/40
144/144 [==============================] - ETA: 0s - loss: 0.8180 - acc: 0.7352
Epoch 5: val_acc improved from 0.82059 to 0.83046, saving model to dog_breed_classifier_model.h5
144/144 [==============================] - 51s 354ms/step - loss: 0.8180 - acc: 0.7352 - val_loss: 0.5069 - val_acc: 0.8305 - lr: 2.0000e-04
Epoch 6/40
144/144 [==============================] - ETA: 0s - loss: 0.7902 - acc: 0.7439
Epoch 6: val_acc improved from 0.83046 to 0.83600, saving model to dog_breed_classifier_model.h5
144/144 [==============================] - 50s 344ms/step - loss: 0.7902 - acc: 0.7439 - val_loss: 0.5045 - val_acc: 0.8360 - lr: 2.0000e-04
Epoch 7/40
144/144 [==============================] - ETA: 0s - loss: 0.7526 - acc: 0.7641
Epoch 7: val_acc improved from 0.83600 to 0.83785, saving model to dog_breed_classifier_model.h5
144/144 [==============================] - 50s 347ms/step - loss: 0.7526 - acc: 0.7641 - val_loss: 0.4962 - val_acc: 0.8379 - lr: 2.0000e-04
Epoch 8/40
144/144 [==============================] - ETA: 0s - loss: 0.7611 - acc: 0.7523
Epoch 8: val_acc did not improve from 0.83785

Epoch 8: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
144/144 [==============================] - 49s 341ms/step - loss: 0.7611 - acc: 0.7523 - val_loss: 0.5016 - val_acc: 0.8366 - lr: 2.0000e-04
Epoch 9/40
144/144 [==============================] - ETA: 0s - loss: 0.7500 - acc: 0.7618
Epoch 9: val_acc did not improve from 0.83785
144/144 [==============================] - 52s 358ms/step - loss: 0.7500 - acc: 0.7618 - val_loss: 0.4949 - val_acc: 0.8335 - lr: 4.0000e-05
Epoch 10/40
144/144 [==============================] - ETA: 0s - loss: 0.7070 - acc: 0.7723
Epoch 10: val_acc did not improve from 0.83785
144/144 [==============================] - 49s 341ms/step - loss: 0.7070 - acc: 0.7723 - val_loss: 0.4931 - val_acc: 0.8354 - lr: 4.0000e-05
Epoch 11/40
144/144 [==============================] - ETA: 0s - loss: 0.7274 - acc: 0.7707
Epoch 11: val_acc improved from 0.83785 to 0.83970, saving model to dog_breed_classifier_model.h5
144/144 [==============================] - 50s 346ms/step - loss: 0.7274 - acc: 0.7707 - val_loss: 0.4891 - val_acc: 0.8397 - lr: 4.0000e-05
Epoch 12/40
144/144 [==============================] - ETA: 0s - loss: 0.7325 - acc: 0.7645
Epoch 12: val_acc did not improve from 0.83970

Epoch 12: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.
144/144 [==============================] - 51s 352ms/step - loss: 0.7325 - acc: 0.7645 - val_loss: 0.4911 - val_acc: 0.8372 - lr: 4.0000e-05
Epoch 13/40
144/144 [==============================] - ETA: 0s - loss: 0.6993 - acc: 0.7734
Epoch 13: val_acc did not improve from 0.83970
144/144 [==============================] - 51s 352ms/step - loss: 0.6993 - acc: 0.7734 - val_loss: 0.4905 - val_acc: 0.8360 - lr: 8.0000e-06
Epoch 14/40
144/144 [==============================] - ETA: 0s - loss: 0.7296 - acc: 0.7657
Epoch 14: val_acc did not improve from 0.83970

Epoch 14: ReduceLROnPlateau reducing learning rate to 1.6000001778593287e-06.
144/144 [==============================] - 51s 352ms/step - loss: 0.7296 - acc: 0.7657 - val_loss: 0.4904 - val_acc: 0.8372 - lr: 8.0000e-06
Epoch 15/40
144/144 [==============================] - ETA: 0s - loss: 0.7080 - acc: 0.7766
Epoch 15: val_acc did not improve from 0.83970
144/144 [==============================] - 48s 335ms/step - loss: 0.7080 - acc: 0.7766 - val_loss: 0.4904 - val_acc: 0.8372 - lr: 1.6000e-06
Epoch 16/40
144/144 [==============================] - ETA: 0s - loss: 0.7295 - acc: 0.7671
Epoch 16: val_acc did not improve from 0.83970

Epoch 16: ReduceLROnPlateau reducing learning rate to 3.2000002647691870e-07.
144/144 [==============================] - 50s 349ms/step - loss: 0.7295 - acc: 0.7671 - val_loss: 0.4903 - val_acc: 0.8372 - lr: 1.6000e-06
```

Results

The trained model achieved an accuracy of 83.97% on the training set. This result indicates that the DenseNet121 architecture is capable of accurately identifying the breed of a dog based on its image.

```
print("Accuracy:", (max(val_acc)*100),"%")
```

```
Accuracy: 83.97040963172913 %
```



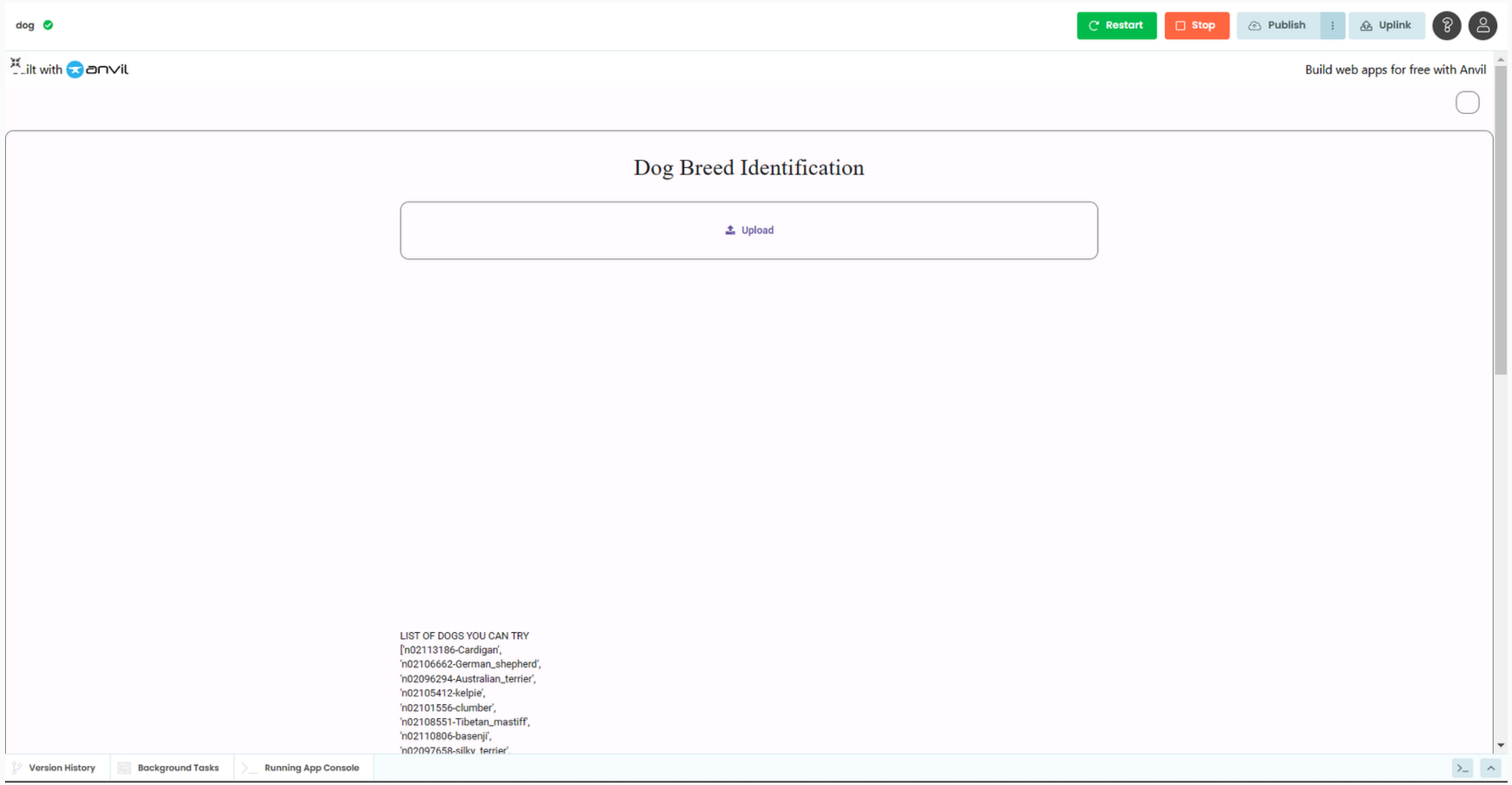
Results

```
1/1 [==============================] - 0s 247ms/step
100.00%          pug
0.00%    bull_mastiff
0.00%    malinois
0.00%    standard_schnauzer
0.00%    Pekinese
```

```
1/1 [==============================] - 0s 90ms/step
98.64%   chow
0.91%    Leonberg
0.18%    collie
0.12%    Pomeranian
0.05%    golden_retriever
```

Results

Application on Anvil Web Builder



Results

Application on Anvil Web Builder

Dog Breed Identification

1 file selected

Result: chow | Confidence: 98.64%
Result: Leonberg | Confidence: 0.91%
Result: collie | Confidence: 0.18%
Result: Pomeranian | Confidence: 0.12%
Result: golden_retriever | Confidence: 0.05%

LIST OF DOGS YOU CAN TRY
['n02113186-Cardigan',
'n02106662-German_shepherd',
'n02096294-Australian_terrier',
'n02105412-kelpie',

Conclusion and Recommendation

The DenseNet121 architecture has shown to be a capable model for dog breed identification. The results obtained in this research indicate that the model can accurately identify the breed of a dog based on its image. This research provides a foundation for future work in the field of dog breed identification and can be used as a reference for other researchers.

However, there is still room for improvement in terms of accuracy. To further improve the accuracy in dog breed recognition, it is recommended to continue research in the following areas:

- Improved Data: Using larger, more diverse, and higher quality datasets for training and testing can enhance the performance of the model.
- Advanced Preprocessing: Applying more sophisticated preprocessing techniques, such as semantic segmentation, can help minimize the impact of background noise on the model's performance.
- Alternative Models: Trying other deep learning models, such as ResNet and Inception, can provide useful information on which models are most suitable for this task.
- Transfer Learning: Investigating the use of transfer learning with other pre-trained models may result in further improvements in accuracy.

Overall, this research showcases the potential of DenseNet121 for dog breed recognition and highlights the significance of carefully considering data quality and model selection to achieve the best results.