Dog breed identification using transfer learning

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS90703
PROJECT NAME	Dog breed identification using transfer learning
MAXIMUM MARKS	4 MARKS

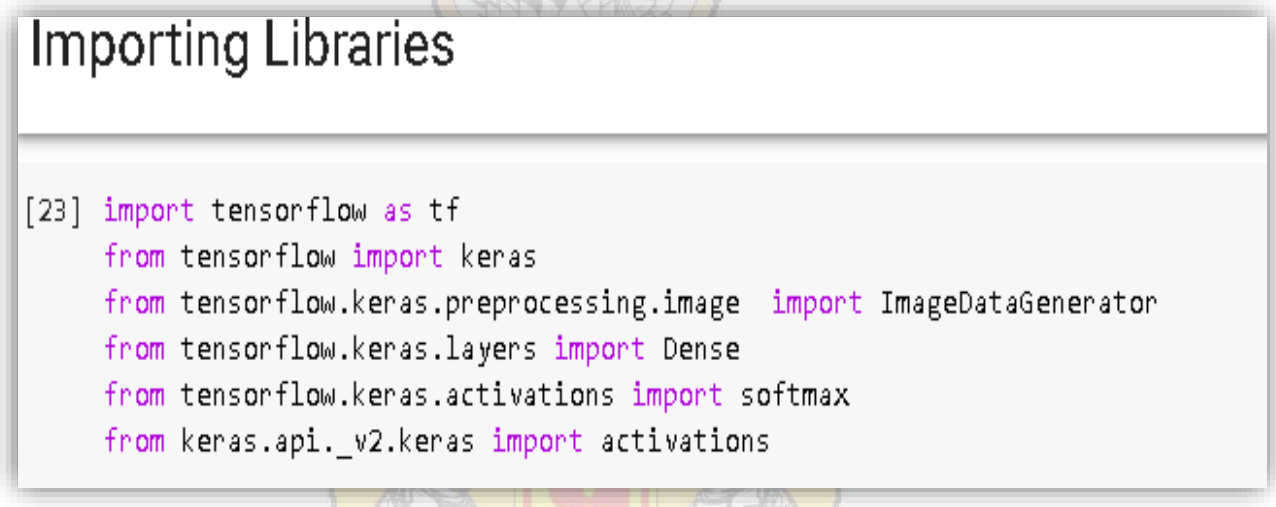
Model Building

Now it's time to build our Convolutional Neural Networking using vgg19 which contains an input layer along with the convolution, max-pooling, and finally an output layer.

Link: https://thesmartbridge.com/documents/spsaimldocs/CNNflow.pdf

1: Importing the Model Building Libraries

Importing the necessary libraries



2: Importing the VGG19 model:

To initialize the VGG19 model, the weights are usually pre-trained on the ImageNet dataset, which is a large-scale dataset of images belonging to 1,000 different categories. These pre-trained weights can be downloaded from the internet, and they can be used as a starting point to fine-tune the model for a specific task, such as object recognition or classification.

```
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

3: Initializing the model:

- The model will be initialized with the pre-trained weights from the ImageNet dataset, and the last fully connected layer will be excluded from the model architecture.
- The loop that follows freezes the weights of all the layers in the VGG19 model by setting

  `i.trainable=False` for each layer in the model. This is done to prevent the weights from being updated during training, as the model is already pre-trained on a large dataset.
- Finally, a `Flatten()` layer is added to the output of the VGG19 model to convert the output tensor into a 1D tensor.
- The resulting model can be used as a feature extractor for transfer learning or as a starting point for building a new model on top of it.

```
[ ] Image_size=[224,224]

[ ] sol=VGG19(input_shape=Image_size + [3] , weights='imagenet' , include_top = False)

[ ] for i in sol.layers:
      i.trainable = False

[ ] y=Flatten()(sol.output)
```

4: Adding Fully connected Layers

- For information regarding CNN Layers refer to the link Link: https://victorzhou.com/blog/intro-to-cnns-part-1/
- As the input image contains three channels, we are specifying the input shape as (128,128,3).
- We are adding a output layer with activation function as “softmax”.

```
[ ]  from keras.api._v2.keras import activations
     final = Dense(20, activation = 'softmax')(y)
```

A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer. The number of neurons in the Dense layer is the same as the number of classes in the training set.The neurons in the last Dense layer, use softmax activation to convert their outputs into respective probabilities.

Understanding the model is a very important phase to properly use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers.

```
vgg19_model.summary()

Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168
 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080
 block3_conv4 (Conv2D)       (None, 56, 56, 256)       590080
 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0
 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160
 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808
 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808
 block4_conv4 (Conv2D)       (None, 28, 28, 512)       2359808
 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0
 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808
```

5: Configure The Learning Process

- The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations inthe learning process. Keras requires a loss function during the model compilation process.
- Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer.
- Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process.

```
vgg19_model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['Accuracy'])
```

6: Train The model

Now, let us train our model with our image dataset. The model is trained for 6 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 30 epochs and probably there is further scope to improve the model.

fit_generator functions used to train a deep learning neural network.

Arguments:

- steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.
- Epochs: an integer and number of epochs we want to train our model for.
- validation_data can be either:
    - an inputs and targets list
    - a generator
    - an inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.
- validation_steps: only if the validation_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is

stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

```
[ ]  vgg19_model.fit(generator, epochs = 6)

     Epoch 1/6
     53/53 [==============================] - 1353s 25s/step - loss: 126.8239 - Accuracy: 0.1218
     Epoch 2/6
     53/53 [==============================] - 1352s 26s/step - loss: 30.5051 - Accuracy: 0.6049
     Epoch 3/6
     53/53 [==============================] - 1336s 25s/step - loss: 3.8852 - Accuracy: 0.9008
     Epoch 4/6
     53/53 [==============================] - 1333s 25s/step - loss: 1.1688 - Accuracy: 0.9584
     Epoch 5/6
     53/53 [==============================] - 1332s 25s/step - loss: 1.6495 - Accuracy: 0.9548
     Epoch 6/6
     53/53 [==============================] - 1287s 24s/step - loss: 0.2561 - Accuracy: 0.9857
     <keras.callbacks.History at 0x7f66172dd540>
```

7: Save the Model

The model is saved with .h5 extension as follows.

An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

```
[ ]  vgg19_model.save("dogbreed.h5")
```

8: Test The model

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data.

```
[ ] import numpy as np
    from keras.preprocessing import image
    from keras.applications.vgg16 import preprocess_input
    from tensorflow.keras.preprocessing.image import load_img, img_to_array

[ ] img_path ='/content/test/000621fb3cbb32d8935728e48679680e.jpg'

[ ] img = load_img(img_path, target_size=(224, 224))
    x = img_to_array(img)
    x = preprocess_input(x)
    preds = vgg19_model.predict(np.array([x]))

    1/1 [==============================] - 1s 1s/step

[ ] class_names=['affenpinscher','beagle','appenzeller','basset','bluetick','boxer','cairn','doberman','german_shepherd','golden_retriever','kelpie','komondor','leonberg',

[ ] class_names[np.argmax(preds)]
```

Taking an image as input and checking the results

By using the model we are predicting the output for the given input image. The predicted class index name will be printed here.