Assignment No. 06

Problem Statement

1. Load in a pre-trained CNN model trained on a large dataset
2. Freeze parameters (weights) in model's lower convolutional layers
3. Define the network architecture using keras
4. Add custom classifier with several layers of trainable parameters to model
5. Train classifier layers on training data available for task
6. Fine-tune hyper parameters and unfreeze more layers as needed

Solution Expected

Object detection using Transfer Learning of CNN architectures by. Load in a pre-trained CNN model trained on a large dataset . Freeze parameters (weights) in model's lower convolutional layers .. Add custom classifier with several layers of trainable parameters to model.

Objectives to be achieved

1. Understand how to use Tensorflow Eager and Keras Layers to build a neural network architecture.
2. Understand how a model is trained and evaluated.
3. Transfer learning is flexible, allowing the use of pre-trained models directly as feature extraction preprocessing and integrated into entirely new models.
4. Keras provides convenient access to many top performing models on the ImageNet image recognition tasks such as VGG, Inception, and ResNet.
5. Our main goal is to train a neural network (using Keras) to obtain > 90% accuracy on Caltech dataset.

Methodology to be used

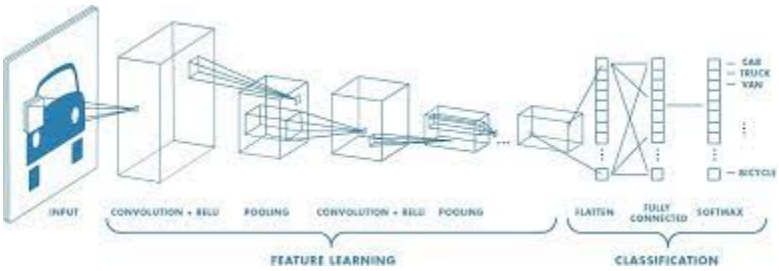
Deep Learning Convolutional Neural Networks Keras

Justification with Theory/Literature

Deep Learning

Deep learning attempts to mimic the human brain—albeit far from matching its ability—enabling systems to cluster data and make predictions with incredible accuracy. Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain—albeit far from matching its ability—allowing it to "learn" from large amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy. Deep learning drives many artificial intelligence (AI) applications and services that improve automation, performing analytical and physical tasks without human intervention. Deep learning technologylies behind everyday products and services (such as digital assistants, voice-enabled TV remotes, and credit card fraud detection) as well as emerging technologies (such as self-driving cars). [1]

Convolution Neural Network



Convolutional Neural Network is one of the main categories to do image classification and image recognition in neural networks. Scene labeling, objects detections, and face recognition, etc., are some of the areas where convolutional neural networks are widely used. CNN takes an image as input, which is classified and process under a certain category such as dog, cat, lion, tiger, etc. The computer sees an image as an array of pixels and depends on the resolution of the image. Based on image resolution, it will see as h *w* d, where h= height w= width and d= dimension. For example, An RGB image is 6 *6* 3 array of the matrix, and the grayscale image is 4 *4* 1 array of the matrix. In CNN, each input image will pass through a sequence of convolution layers along with pooling, fully connected layers, filters (Also known as kernels). After that, we will apply the Soft-max function to classify an object with probabilistic values 0 and 1.

Keras

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research. Keras is: Simple -- but not simplistic. Keras reduces developer cognitive load to free you to focus on the parts of the problem that really matter. Flexible -- Keras adopts the principle of progressive disclosure of complexity: simple workflows should be quick and easy, while arbitrarily advanced workflows should be possible via a clear path that builds upon what you've already learned. Powerful -- Keras provides industry-strength performance and scalability: it is used by organizations and companies including NASA, YouTube, or Waymo. [4]

TensorFlow 2

TensorFlow 2 is an end-to-end, open-source machine learning platform. You can think of it as an infrastructure layer for differentiable programming. It combines four key abilities: Efficiently executing low-level tensor operations on CPU, GPU, or TPU. Computing the gradient of arbitrary differentiable expressions. Scaling computation to many devices, such as clusters ofhundreds of GPUs. Exporting programs ("graphs") to external runtimes such as servers, browsers, mobile and embedded devices. [4]

PyTorch

PYTorch is an open source machine learning library used for developing and training neural network based deep learning models. It is primarily developed by Facebook’s AI research group. PyTorch can be used with Python as well as a C++. Naturally, the Python interface is more polished. PyTorch is an optimized Deep Learning tensor library based on Python and Torch and is mainly used for applications using GPUs and CPUs. PyTorch is favored over other Deep Learning frameworks like TensorFlow and Keras since it uses dynamic computation graphs and is completely Pythonic. It allows scientists, developers, and neural network debuggers to run and test portions of the code in real-time. Thus, users don’t have to wait for the entire code to be implemented to check if a part of the code works or not.

Data Augmentation

TensorFlow Data augmentation (DA) and transfer learning (TL) were used to increase the size of the training set and to optimize the models for the new dataset. The effect of the different breathing patterns on the performance of the classifiers was also studied.

In [2]:

```
# example of using a pre-trained model as a classifier
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
from keras.applications.vgg16 import VGG16
# load an image from file
image = load_img('dog.jpeg', target_size=(224, 224))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
# load the model
model = VGG16()
# predict the probability across all output classes
yhat = model.predict(image)
# convert the probabilities to class labels
label = decode_predictions(yhat)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/v
gg16_weights_tf_dim_ordering_tf_kernels.h5
553467096/553467096 [==============================] - 127s 0us/step
1/1 [==============================] - 2s 2s/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet
_class_index.json
35363/35363 [==============================] - 0s 0us/step
pug (90.15%)
```

In [3]:

```
# load an image from file
image = load_img('dog1.jpeg', target_size=(224, 224))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
# load the model
```

```
model = VGG16()
# predict the probability across all output classes
yhat = model.predict(image)
# convert the probabilities to class labels
label = decode_predictions(yhat)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
1/1 [==============================] - 1s 1s/step
German_shepherd (99.95%)
```

In [5]:

```
# load an image from file
image = load_img('dog2.jpg', target_size=(224, 224))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
# load the model
model = VGG16()
# predict the probability across all output classes
yhat = model.predict(image)
# convert the probabilities to class labels
label = decode_predictions(yhat)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
1/1 [==============================] - 3s 3s/step
Pekinese (35.46%)
```

In [ ]: