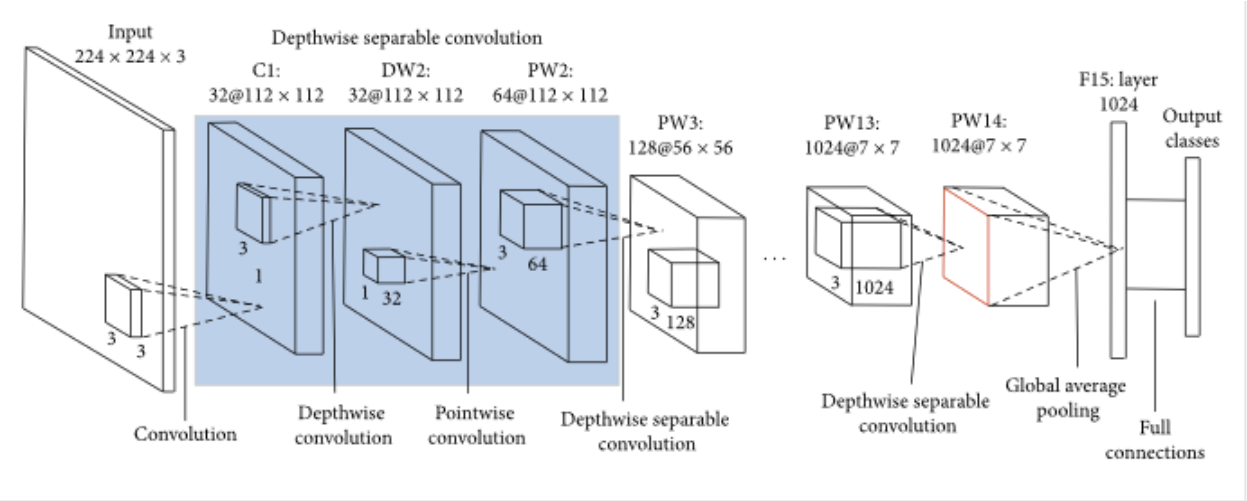
Explanation

MobileNet Architecture for image classification

MobileNet uses a Convolutional Neural Network (CNN) architecture model to classify images. It is open sourced by Google. MobileNet architecture uses very less computing power to run. This makes it a perfect fit for mobile devices, embedded systems, and computers to run without GPUs.

Below shows the architecture and the number of layers of a pre-trained MobileNet architecture model based on convolutional neural network:



To understand CNN's architecture and how it works, we must understand the following few points.

Convolution Neural Networks combines deep neural networks and a set of operations known as convolutions.

CNN's are a class of deep learning techniques popularly used to solve computer vision tasks. As they learn directly from input data, they are beneficial for finding patterns in images, enabling them to perform tasks such as face recognition, object detection, and scene recognition.

They are applied in solving computer vision and facial recognition tasks in smart cameras and self-driving cars.

Before we learn how CNNs work, it is important to understand why we need CNNs and why they are better than other algorithms?

1. CNN's produce innovative recognition results.
2. CNN's automatically learn a hierarchy of features from inputs. This eliminates the need for hand engineering the extraction of features.
3. CNNs can developed on pre-existing networks. Thus, CNN's can be retrained to perform new recognition tasks.

The CNN architecture consists of a stacking of three building blocks:

- Convolution layers
- Pooling layers, i.e., max pooling
- Fully connected (FC) layers

Convolution layer

A convolution layer is a key component of the CNN architecture. This layer helps us perform feature extractions on input data using the convolution operation. The convolution operation involves performing an element-wise multiplication between the filter's weights and the patch of the input image with the same dimensions. Finally, the resulting output values are added together.

Convolution Operation

We slide the m x n filter over the input image, perform element-wise multiplication, and add the outputs.

Besides, on the CNN architecture, we can note that feature extraction is achieved using both convolution layers (linear operation) and an activation function (non-linear operation). We always add the activation function to each layer of convolution. Rectified Linear Unit (ReLU) is the most used activation function in deep learning.

Rectified Linear Unit (ReLU)

From the image above, we note that it combines a convolution layer with the ReLU layer. The purpose of the ReLU activation function is to introduce non-linearity into the network.

Pooling layer

A pooling layer is non-linear down-sampling that reduces the dimensionality of the feature maps of an input image, making its representation smaller and more manageable. This process reduces the number of parameters and computation resources in the network.

Max Pooling

Max pooling is the most popular among the pooling operations. It extracts patches from the input feature maps, picks the largest value on each patch, and discards the rest.

Average Pooling

Besides, average pooling is another type of pooling operation that's worth mentioning. It performs down-sampling by performing an average of all values in a feature map. Thus, picking this averaged value as the new value of the feature space.

Fully Connected layer

The Fully Connected (FC) layer is the last building block in a Convolution Neural Network. The output feature maps of the layer before the FC layer are transformed into a one-dimension (1D) array of numbers, i.e., flattened. That layer is connected to one or more fully connected layers.

The FC layer maps the extracted features from the convolution and pooling layers onto the final output, i.e., classification.

Lastly, once the CNN has derived an output using a probability function such as softmax, we compare this output with the output we want the network to give. The difference between the two gives us the error in our network.

The CNNs weights are then updated, and it minimizes the objective function through a process known as backpropagation, as we will see in the image below. We can describe backpropagation as the backward propagation of errors.

It's fine-tuning the weights and bias so it can give us the output we want. It's the core algorithm behind how neural networks learn

MobileNet Architecture Implementation

1. Prepare the data to be trained. This could be a single dataset, or a labelled set of data

2. Import all the necessary modules like TensorFlow and its associated modules and functions. Also import NumPy and OpenCV cv2.

3. In General, the implementation of the single image classification was carried out in the following steps through the implementation of my created function:
    - Read the image or dataset we have prepared.
    - Display the image to check that it matches the dimension that MobileNet would want to use.
    - Implement the MobileNet architecture function for the application.
    - Prepare the image for the predictions
    - Run the algorithm to predict the image
    - Decode the outcome of the prediction.

4. Read the image or dataset we have prepared.

The image was gotten online and stored the working directory of the project. A function is then defined which utilizes OpenCV

```
def read_image(an_image):
    """This function reads any image from a given path"""
    image = cv2.imread(an_image)
    return image
```

This would read the image data from the location

4. Next, we display this image to ensure it is in the correct format that the MobileNet model would be comfortable with using. Once again, we define a function that utilizes the inbuilt OpenCV function to show image.

```
def display_image(window_name, image_read):
    """This function any image from a given path"""
    cv2.imshow(window_name, image_read)
    cv2.waitKey(0)
```

5. Our image is ready, and we can now run the first part of the MobileNet Architecture which download a MobileNet function *tf.keras.applications.mobilenet.MobileNet*. This function returns a Keras image classification model, optionally loaded with weights pre-trained on ImageNet.
From the TensorFlow manual, the parameters arguments of this function are listed below from this source TensorFlow

Parameter Argument	
`input_shape`	Optional shape tuple, only to be specified if include_top is False (otherwise the input shape has to be (224, 224, 3) (with channels_last data format) or (3, 224, 224) (with channels_first data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 32. E.g. (200, 200, 3) would be one valid value. Default to None. input_shape will be ignored if the input_tensor is provided.
`alpha`	Controls the width of the network. This is known as the width multiplier in the MobileNet paper. - If alpha < 1.0, proportionally

	decreases the number of filters in each layer. - If alpha > 1.0, proportionally increases the number of filters in each layer. - If alpha = 1, default number of filters from the paper are used at each layer. Default to 1.0.
depth_multiplier	Depth multiplier for depth-wise convolution. This is called the resolution multiplier in the MobileNet paper. Default to 1.0
dropout	Dropout rate. Default to 0.001.
include_top	Boolean, whether to include the fully connected layer at the top of the network. Default to True.
weights	One of None (random initialization), 'imagenet' (pre-training on ImageNet), or the path to the weights file to be loaded. Default to imagenet.
input_tensor	Optional Keras tensor (i.e., output of layers.Input()) to use as image input for the model. input_tensor is useful for sharing inputs between multiple different networks. Default to None.
pooling	Optional pooling mode for feature extraction when include_top is False. • None (default) means that the output of the model will be the 4D tensor output of the last convolutional block. • avg means that global average pooling will be applied to the output of the last convolutional block, and thus the output of the model will be a 2D tensor. • max means that global max pooling will be applied.
classes	Optional number of classes to classify images into, only to be specified if include_top is True, and if no weights argument is specified. Defaults to 1000.
classifier_activation	A str or callable. The activation function to use on the "top" layer. Ignored unless include_top=True. Set classifier_activation=None to return the logits of the "top" layer. When loading pretrained weights, classifier_activation can only be None or "softmax".
**kwargs	For backwards compatibility only.

This function would be implemented in a defined function which returns the mobile net as shown below

```
def mobilenet_arch():
   mobile_net = tf.keras.applications.mobilenet.MobileNet(input_shape=(224, 224, 3),
   alpha=1.0,
   include_top=True,
   weights="imagenet",
   input_tensor=None,
   classes=1000,
   pooling=None,
   classifier_activation="softmax",
   )

   return mobile_net
```

6. Here we now need to preprocess the input that has been built on the MobileNet in the step above. Since the MobileNet architecture is run on Keras and each Keras application expects a specific kind of input preprocessing. Thus, we use the inbuilt pre-processing function inside our defined function as shown below:

```
def prepare_image(a_file):
    img = image.load_img(a_file, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array_expanded_dims = np.expand_dims(img_array, axis=0)
    return tf.keras.applications.mobilenet.preprocess_input(img_array_expanded_dims)
```

7. The prediction is done through the algorithm MobileNet prediction function

```
1/1 [==============================] - 1s 738ms/step
[[('n02111889', 'Samoyed', 0.9932219), ('n02112137', 'chow', 0.0023243665), ('n02109961', 'Eskimo_dog', 0.0018303135), ('n02114
548', 'white_wolf', 0.0011486099), ('n02112018', 'Pomeranian', 0.0005273827)]]
```

```
algo_predictions = mobilenet.predict(preprocessed_image)
```

8. Finally, we decode the outcome of the prediction where the classification is done with the probability of certainty that the classification is correct.

```
outcome = imagenet_utils.decode_predictions(algo_predictions)
```

Prediction Results:

A couple of images were passed into the model. The image from which the result displayed below was captured was that of a Samoyed dog. The MobileNet architecture with ImageNet give a series of prediction as shown below:

Samoyed = 0.9932219 i.e., 99.3%

Chow = 0.0023 i.e., 0.23%

Eskimo dog = 0.0018 i.e., 0.18%

White wolf = 0.001148 i.e., 0.1148 %

Pomeranian = negligible based on result shown below.

The prediction was very correct as the image is that of a Samoyed, a type of Dog.