Necessary Installations:

pip install pandas

pip install numpy

pip install opencv-python

pip install matplotlib

pip install Pillow

pip install scikit-learn

pip install keras

pip install tensorflow

Add a ! beforehand

Model Description:

This is a deep learning model written in Python using the Keras library. The model is a convolutional neural network (CNN) that is designed for image classification tasks. The architecture of the model is as follows:

1. The model starts with a Conv2D layer with 96 filters, a kernel size of 11x11, and a stride of 4x4. The input shape is specified as a 3-dimensional tensor with dimensions 224 x 224 x 3. The padding is set to 'valid', which means that no padding is added to the input. The layer also applies L2 regularization with a strength of 0.01 to the kernel weights. This layer is followed by an activation function of Rectified Linear Unit (ReLU).
2. Next, a MaxPooling2D layer is added with a pool size of 2x2 and a stride of 2x2. This layer performs down-sampling of the input by taking the maximum value of non-overlapping 2x2 regions.
3. Another Conv2D layer is added with 256 filters, a kernel size of 11x11, and a stride of 1x1. The padding is set to 'valid', and L2 regularization is applied

with a strength of 0.01. This layer is followed by an activation function of ReLU.

4. Another MaxPooling2D layer is added with the same pool size and stride as the previous one.
5. Another Conv2D layer is added with 384 filters, a kernel size of 3x3, and a stride of 1x1. The padding is set to 'valid', and no regularization is applied. This layer is followed by an activation function of ReLU and a Dropout layer with a rate of 0.4. Dropout is a technique for regularization that randomly sets a fraction of the input units to 0 during training.
6. The output of the previous layer is flattened into a 1-dimensional tensor.
7. A Dense layer with 4096 units is added with ReLU activation. This layer has an input shape of 224x224x3, which is the flattened output from the previous layer.
8. Another Dense layer with 4096 units is added with ReLU activation.
9. Another Dense layer with 1000 units is added with ReLU activation. This layer is designed for image classification tasks where the goal is to predict the probabilities of an input image belonging to one of 1000 classes.
10. A final Dense layer with 1 unit is added with sigmoid activation. This layer is designed for binary classification tasks where the goal is to predict whether an input image belongs to a particular class or not.

Train and Validation split:

The `train_test_split` function takes three arguments:

`train_images`: The input images for the training set.

`train_labels`: The labels for the training set.

`test_size`: The fraction of the data that should be reserved for the validation set. In this case, it is set to 0.15, which means that 15% of the data will be used for validation and the remaining 85% will be used for training.

Prediction

The `model.predict()` function takes an input data and returns the predicted output. In this case, `test_images` is the input data and `predictions` is the predicted output.

The first transformation applied to the `predictions` array is a sigmoid transformation using the NumPy `exp` function. This transformation is used to convert the raw predictions into a range between 0 and 1, which can be interpreted as probabilities.

The second transformation applied to the `predictions` array is a normalization using the NumPy `min` and `max` functions. This transformation scales the range of the predictions to be between 0 and 1.

Next, the function calculates the mean of the normalized predictions using the NumPy `mean` function.

Finally, the function iterates over each prediction in the `predictions` array and compares it to the mean. If the prediction is less than or equal to the mean, the function appends a string to the `new_predictions` list indicating that the prediction is "good". Otherwise, it appends a string indicating that the prediction is "not-good".