PCA FOR VGG

This code explains the process of using pretrained weights (VGG16) as feature extractors for both neural network and traditional machine learning (Random Forest). It also uses dimensional reduction using PCA to reduce the number of features for speedy training.

Read images while using loop then resize them. Convert list into arrays and extract labels from text to integer by using sklearn preprocessing

## **Train Test Split dataset**

## **Scaling Feautres**

Scale pixel values to between 0 and 1

One hot encode y values for neural network. Not needed for Random Forest

## **Load VGG model with trained weights**

#Load VGG model with imagenet trained weights and without classifier/fully connected layers

#We will use this as feature extractor.

Make loaded layers as non-trainable. This is important as we want to work with pre-trained weights

#Now, let us extract features using VGG imagenet weights

#Train features

#test features

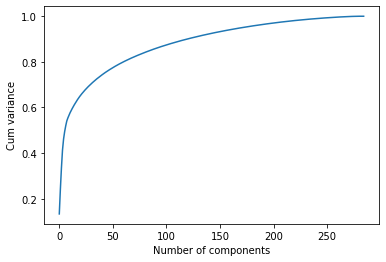
## **Using PCA to reduce dimensions**

Reduce dimensions using PCA

#First verfiy the ideal number of PCA components to not lose much information.

#Try to retain 90% information, so look where the curve starts to flatten.

#Remember that the n\_components must be lower than the number of rows or columns (features)



This plot shows that maximum number of PCA components are more than 285 but our rows are limit to 285 we can pick this value. On this value we have acheived highest cum variance almost equals to 1.

Extract train and test PCA Features from PCA analysis then fit and transform train and test data to gain train and test features.

#Pick the optimal number of components. This is how many features we will have

#for our machine learning

## **Train data into the model**

#If we want 90% information captured we can also try ...

#pca=PCA(0.9)

#principalComponents = pca.fit\_transform(X\_for\_RF)

############## Neural Network Approach ##################

##Add hidden dense layers and final output/classifier layer.

Model: "model\_5"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

input\_7 (InputLayer) [(None, 285)] 0

dense\_10 (Dense) (None, 256) 73216

dense\_11 (Dense) (None, 7) 1799

=================================================================

Total params: 75,015

Trainable params: 75,015

Epoch 20/20

9/9 [==============================] - 0s 4ms/step - loss: 2.5097e-09 - categorical\_accuracy: 1.0000

Total execution time with PCA is: 0:00:01.343187

## **Now, predict our Test data by using model**

### Find overall accuracy by comparing actual and predict outputs

##Print overall accuracy

Accuracy = 1.0

#Confusion Matrix - verify accuracy of each class

