`from keras.models import Sequential`: This imports the `Sequential` class from the `keras.models` module. `Sequential` is a linear stack of neural network layers.

`from keras.layers import Conv2D`: This imports the `Conv2D` layer class from the `keras.layers` module. `Conv2D` is a 2-dimensional convolutional layer that performs convolutional operations on input data.

`from keras.layers import MaxPooling2D`: This imports the `MaxPooling2D` layer class from the `keras.layers` module. `MaxPooling2D` performs max pooling operations on input data, reducing the spatial dimensions.

`from keras.layers import Flatten`: This imports the `Flatten` layer class from the `keras.layers` module. `Flatten` flattens the input into a 1-dimensional array, which is useful when transitioning from convolutional layers to fully connected layers.

`from keras.preprocessing.image import ImageDataGenerator`: This imports the `ImageDataGenerator` class from the `keras.preprocessing.image` module. `ImageDataGenerator` generates batches of augmented/normalized data from image files.

`from keras.layers import Dense`: This imports the `Dense` layer class from the `keras.layers` module. `Dense` is a fully connected layer that performs element-wise operations on input data.

`from keras.layers import BatchNormalization`: This imports the `BatchNormalization` layer class from the `keras.layers` module. `BatchNormalization` normalizes the activations of the previous layer, which can help with the training process.

`from keras.layers import Dropout`: This imports the `Dropout` layer class from the `keras.layers` module. `Dropout` applies dropout regularization to the input, randomly setting a fraction of input units to 0 during training, which helps prevent overfitting.

`model = Sequential()`: This creates an instance of the `Sequential` model.

`model.add(Conv2D(32, kernel_size = (3, 3), activation='relu', input_shape=(128,128, 3)))`: This adds a 2D convolutional layer to the model with 32 filters, a kernel size of 3x3, ReLU activation function, and an input shape of (128, 128, 3). The input shape corresponds to 128x128 RGB images.

`model.add(MaxPooling2D(pool_size=(2,2)))`: This adds a max pooling layer to the model with a pool size of 2x2.

`model.add(BatchNormalization())`: This adds a batch normalization layer to the model, which normalizes the activations of the previous layer.

`model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))`: This adds another 2D convolutional layer to the model with 64 filters and a kernel size of 3x3.

`model.add(MaxPooling2D(pool_size=(2,2)))`: This adds another max pooling layer to the model with a pool size of 2x2.

`model.add(BatchNormalization())`: This adds another batch normalization layer to the model.

`model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))`: This adds another 2D convolutional layer to the model with 64 filters and a kernel size of 3x3.

`model.add(MaxPooling2D(pool_size=(2,2)))`: This adds another max pooling layer to the model with a pool size of 2x2.

`model.add(BatchNormalization())`: This adds another batch normalization layer to the model.

`model.add(Conv2D(96, kernel_size=(3,3), activation='relu'))`: This adds another 2D convolutional layer to the model with 96 filters and a kernel size of 3x3.

`model.add(MaxPooling2D(pool_size=(2,2)))`: This adds another max pooling layer to the model with a pool size of 2x2.

`model.add(BatchNormalization())`: This adds another batch normalization layer to the model.

`model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))`: This adds another 2D convolutional layer to the model with 32 filters and a kernel size of 3x3.

`model.add(MaxPooling2D(pool_size=(2,2)))`: This adds another max pooling layer to the model with a pool size of 2x2.

`model.add(BatchNormalization())`: This adds another batch normalization layer to the model.

`model.add(Dropout(0.2))`: This adds a dropout layer to the model with a dropout rate of 0.2, which randomly sets 20% of the input units to 0 during training.

`model.add(Flatten())`: This adds a flatten layer to the model, which converts the multidimensional output from the previous layer into a 1D array.

`model.add(Dense(128, activation='relu'))`: This adds a fully connected layer to the model with 128 units and a ReLU activation function.

`model.add(Dropout(0.3))`: This adds another dropout layer to the model with a dropout rate of 0.3.

`model.add(Dense(25, activation = 'softmax'))`: This adds the final fully connected layer to the model with 25 units (assuming you have 25 classes) and a softmax activation function, which outputs probabilities for each class.

`model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])`: This compiles the model with the Adam optimizer, categorical cross-entropy loss function (since it's a multi-class classification problem), and accuracy as the metric to evaluate the model's performance.

`train_datagen = ImageDataGenerator(rescale = None, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True)`: This creates an instance of `ImageDataGenerator` for the training set. It specifies rescaling, shear range, zoom range, and horizontal flip as data augmentation techniques.

`test_datagen = ImageDataGenerator(rescale = 1./255)`: This creates an instance of `ImageDataGenerator` for the test set. It only performs rescaling.

`training_set = train_datagen.flow_from_directory('dataset/train', target_size = (128, 128), batch_size = 32, class_mode = 'categorical')`: This generates batches of augmented/normalized data from the training set directory. It sets the target image size to 128x128, batch size to 32, and uses categorical labels.

`print(test_datagen)`: This prints information about the test data generator.

`labels = (training_set.class_indices)`: This retrieves the class labels and their corresponding indices from the training set.

`print(labels)`: This prints the class labels and their indices.

`test_set = test_datagen.flow_from_directory('dataset/test', target_size = (128, 128), batch_size = 32