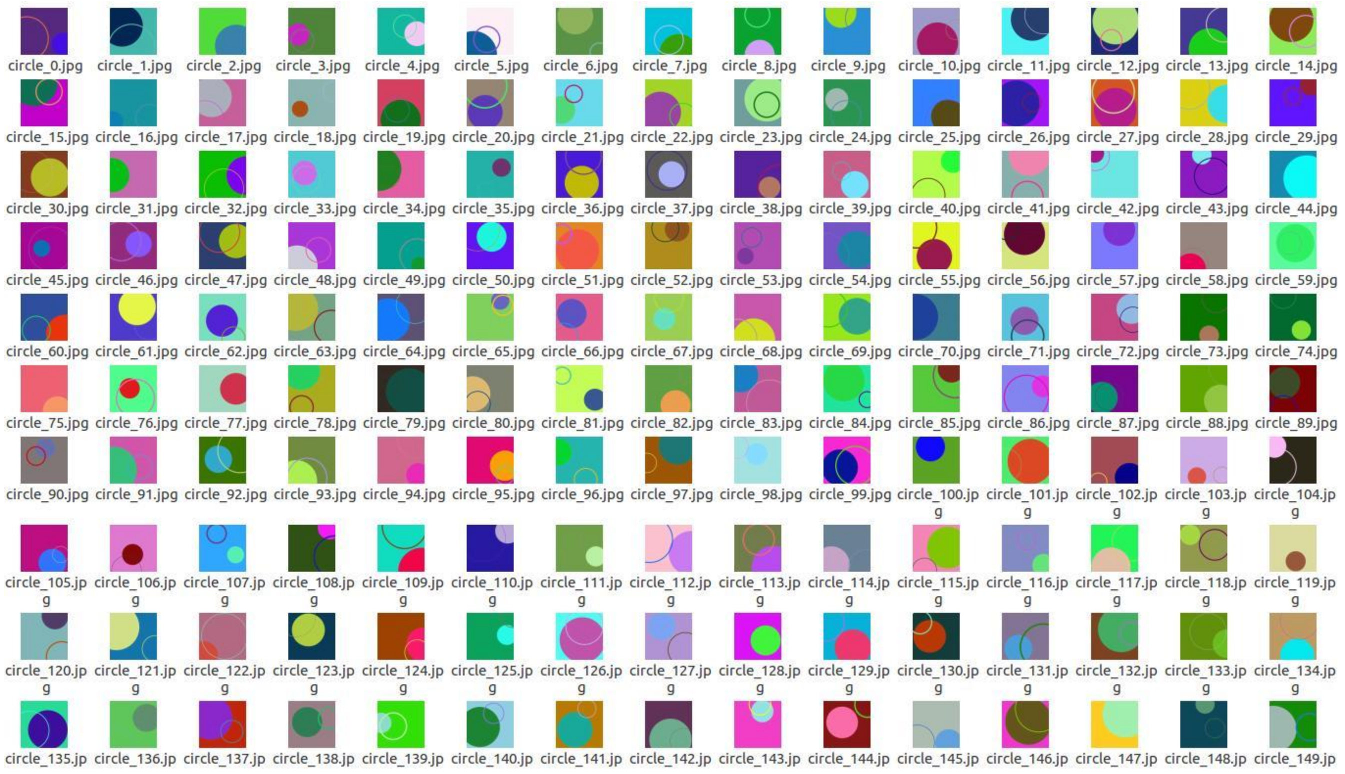
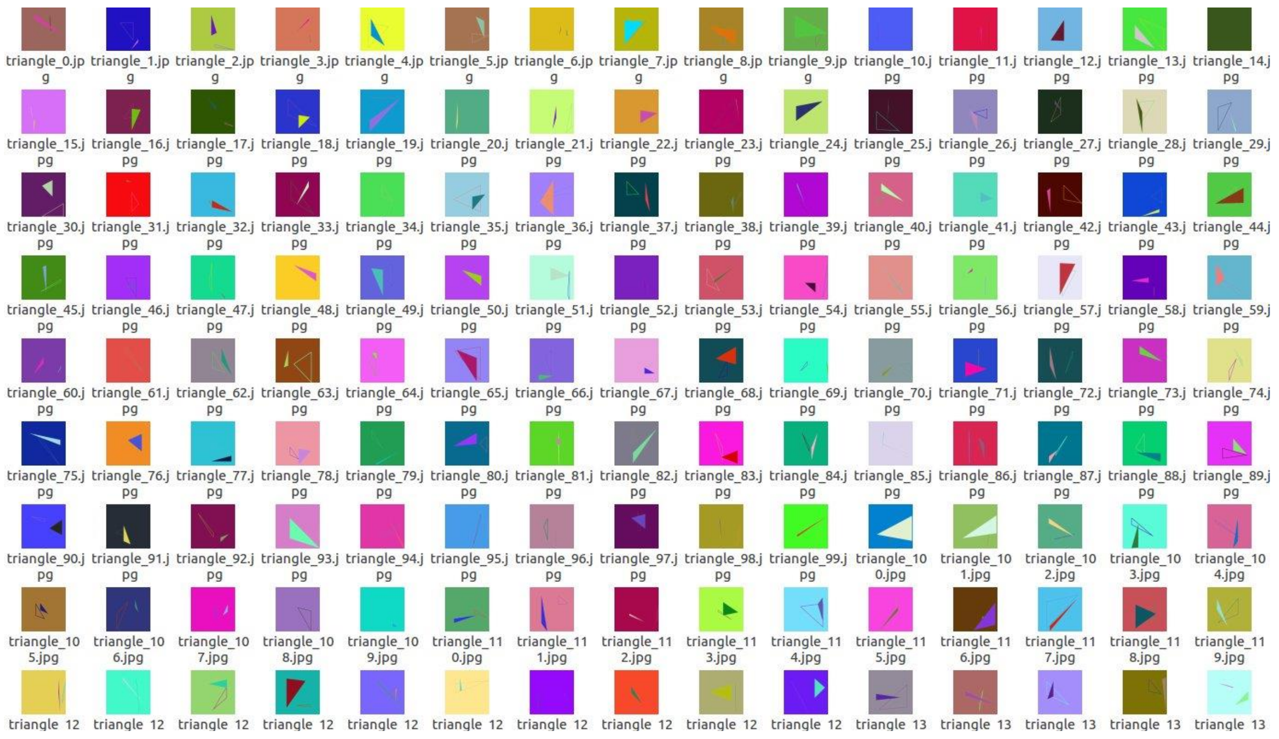
Single Output with Multi-Class Classification

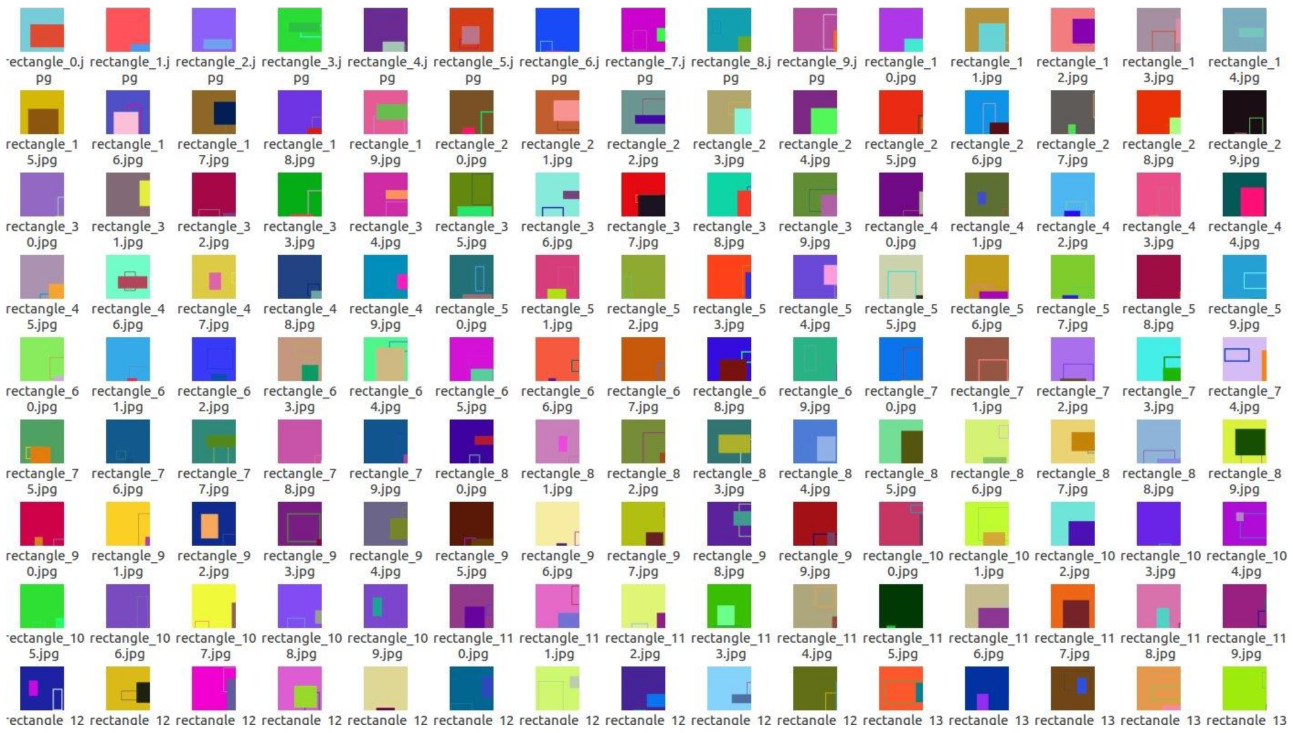
The convolutional neural network will learn using supervised training in Keras. The training and validation datasets will have a supervised output of a single label for each class in a 300 by 300 image. The dataset consisting of circles, triangles, squares and rectangles is created by the program draw.py. It uses multiprocessing of independent pool workers to draw these shapes and saving these shapes in ".png" format. Draw.py uses the cairo library for drawing the shapes and the random library to randomize the location and size of these shapes. Each 300 by 300 image will have two variations, which are the linear contours and the fill of the shape pertaining to the same class. To execute the file, simply run python draw.py --draw-func draw_circle --num-images 2000 --file-dir training_set/circle/circle_, where draw-func represents the drawing function that draws either circle, triangle, square or rectangle or a combination of these shapes if needed. There are 2,000 images per data shape with a total of 8,000 images for the training dataset. The validation dataset consists of 400 images per data shape with a total of 1,600 images, which is 20% of the training dataset.

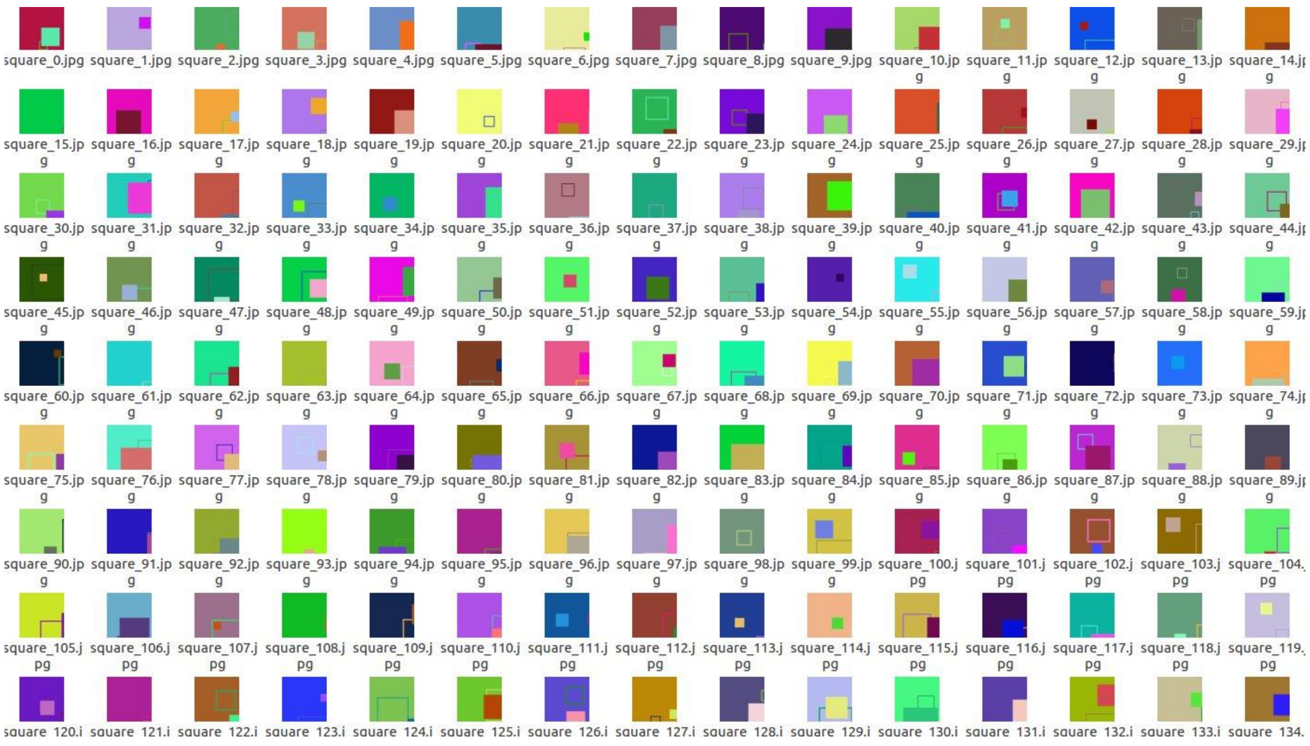
A single PNG image file has a better pixel precision, but it comes at a cost of having twice or even three times as much data storage as a JPEG file. Since the training data tend to increase when the architecture of the convolutional neural network has more layers, it is efficient



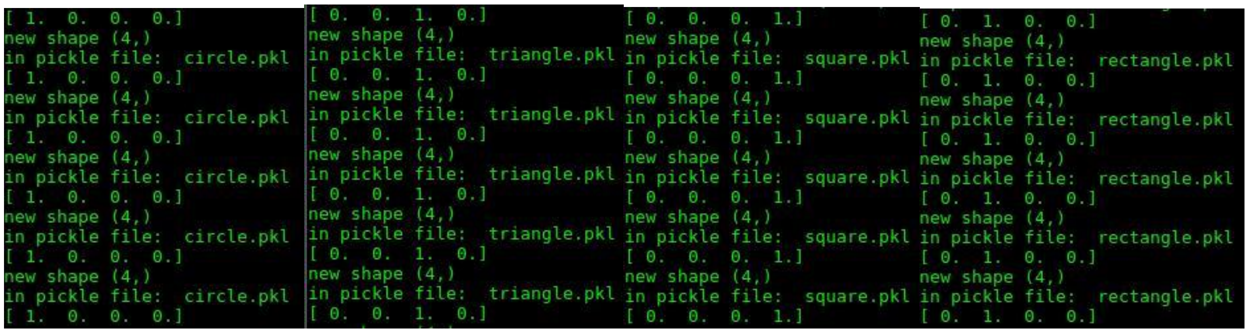
to reduce the size of the images by converting from PNG to JPEG. The program reduce_images_pickle.py reduces image sizes by adjusting a ratio of height and base decrease. It then saves the images in pickle file, where the same pickle file is opened to save the newly reduced size images into a folder.







After each image file size is reduced, pickletestcats.py is the main file that will process the reduced sized images. It is a multiprocessing program of independent pool workers, which uses the Pillow 3 module to open JPEG image files. The pool workers individually convert the JPEG image file into a numpy array of shape (300, 300, 3), where 300 represents the width and the height of the image and 3 represents the color format. The image's corresponding output label (y_label) is generated by using the Keras library of to_categorical encoding. The encoding transforms integers into different binary 1 and 0 encodings of output classes. These classes are circle, rectangle, triangle, and square. The numpy array's total size of 2000 images per shape is reduced by a half if the numpy data structure is a float-16 instead of float-32. After the execution of the program for each shape in the training and validation data, there will be eight pickle files called “circle.pkl”, “triangle.pkl”, “square.pkl”, “rectangle.pkl”, “valid_circle.pkl”, “valid_triangle.pkl”, “valid_square.pkl” and “valid_rectangle.pkl”. A tuple of (image data, output y label) is returned by the pool workers.



```
[ 1.  0.  0.  0.]
new shape (4,)
in pickle file:  circle.pkl
[ 1.  0.  0.  0.]
new shape (4,)
in pickle file:  circle.pkl
[ 1.  0.  0.  0.]
new shape (4,)
in pickle file:  circle.pkl
[ 1.  0.  0.  0.]
new shape (4,)
in pickle file:  circle.pkl
[ 1.  0.  0.  0.]
new shape (4,)
in pickle file:  circle.pkl
[ 1.  0.  0.  0.]
```

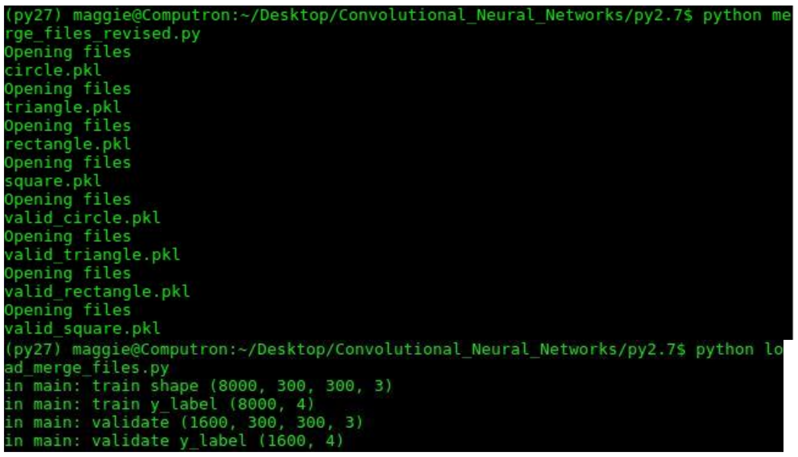
```
[ 0.  0.  1.  0.]
new shape (4,)
in pickle file:  triangle.pkl
[ 0.  0.  1.  0.]
new shape (4,)
in pickle file:  triangle.pkl
[ 0.  0.  1.  0.]
new shape (4,)
in pickle file:  triangle.pkl
[ 0.  0.  1.  0.]
new shape (4,)
in pickle file:  triangle.pkl
[ 0.  0.  1.  0.]
new shape (4,)
in pickle file:  triangle.pkl
[ 0.  0.  1.  0.]
```

```
[ 0.  0.  0.  1.]
new shape (4,)
in pickle file:  square.pkl
[ 0.  0.  0.  1.]
new shape (4,)
in pickle file:  square.pkl
[ 0.  0.  0.  1.]
new shape (4,)
in pickle file:  square.pkl
[ 0.  0.  0.  1.]
new shape (4,)
in pickle file:  square.pkl
[ 0.  0.  0.  1.]
new shape (4,)
in pickle file:  square.pkl
[ 0.  0.  0.  1.]
```

```
[ 0.  1.  0.  0.]
new shape (4,)
in pickle file:  rectangle.pkl
[ 0.  1.  0.  0.]
new shape (4,)
in pickle file:  rectangle.pkl
[ 0.  1.  0.  0.]
new shape (4,)
in pickle file:  rectangle.pkl
[ 0.  1.  0.  0.]
new shape (4,)
in pickle file:  rectangle.pkl
[ 0.  1.  0.  0.]
new shape (4,)
in pickle file:  rectangle.pkl
[ 0.  1.  0.  0.]
```

The combined pickled data of all the shapes represent the training dataset and the validation dataset, which is needed for the convolutional neural network to learn. The file merge_files_revised.py merges the shapes by using the Loader class, which is a multiprocessing module, to load the images and store them in a very large list. All the pool workers execute the loading of the images and the appending of the tuple (image data, output y label) independently. Therefore, a global storage of each large dataset list per shape is needed because each pool worker needs access to the it. I tried allocating the list using “None” times number of images. The numpy array is organized in a tuple, so the list allocation doesn't work in numpy arrays or in tuples. Memmap would not work either because the merged dataset contains a tuple, not a single memory array. After the loaded data is stored in the large lists, the program uses list concatenation to merge all the shapes together, forming the training and validation dataset.

Since the dataset is a very large list, joblib does a successful job dumping and compressing into a file, which will be loaded from load_merge_files.py. The reason for having one file for merging the dataset and another file for loading the merged dataset is that it is better to shuffle the dataset before loading into the convolutional neural network. Although the Keras' generator automatically shuffles with each run of an epoch, it is a good practice to shuffle the training dataset to allow for randomization of images. The program shuffles the dataset and loads the previously merged files by organizing the tuple into two different lists. The first one contains the numpy pixel data and the second list contains its corresponding y label output, which is needed for a supervised training of the neural network. The large list is being stored using the numpy array's memory mapping because each shape of the data pixel array is (300, 300, 3) and its corresponding y output label is (4,) where 4 represents the number of classes to be learned by the convolutional neural network.



```
(py27) maggie@Computron:~/Desktop/Convolutional_Neural_Networks/py2.7$ python me
rge_files_revised.py
Opening files
circle.pkl
Opening files
triangle.pkl
Opening files
rectangle.pkl
Opening files
square.pkl
Opening files
valid_circle.pkl
Opening files
valid_triangle.pkl
Opening files
valid_rectangle.pkl
Opening files
valid_square.pkl
(py27) maggie@Computron:~/Desktop/Convolutional_Neural_Networks/py2.7$ python lo
ad_merge_files.py
in main: train shape (8000, 300, 300, 3)
in main: train y_label (8000, 4)
in main: validate (1600, 300, 300, 3)
in main: validate y_label (1600, 4)
```

Background data without the Shape Contours

It is redundant for the convolutional neural network in cnn_copy_sobel_test.py to obtain pixelized convolutions from background pixels that do not have object contours. An image preprocessing function is needed before the image data are inputted into the first two-dimensional convolution of the network. This function get_edges is responsible for cropping the image and reducing the background space. The input image needs to be converted into grayscale for the Sobel algorithm to process. Since findContours of the opencv2 module alters the original image, a copy of the input image is needed. After finding the contours in both the x and y directions of the Sobel algorithm, the gradients in both directions clarify the contour boundaries of the image. The combined gradient adds the weights in gradients from x and y derivatives, which finalizes the pixels for the image contours.

When the coordinates of the contours are detected, the rectangular boundaries are drawn around it by using opencv2's rectangle method. A list for x coordinates and another list for y coordinates stores all the x and y coordinates returned by findContours. The minimum and maximum values of the x and y lists will determine the area of the cropping region. The locations of the rectangles are in (x,y) coordinates where the cropped version will be from the top left corner (smallest_x, smallest_y) and the bottom right opposite corner (largest_x, largest y). The original input image will be cropped by slicing the coordinates of the original array.

The layers of the convolutional neural network will gather pixels from images that are cropped and will not see the original image. To maintain the scale consistency of the size of the input image at 300 pixels by 300 pixels in RGB format, the cropped image needs to be resized in aspect ratio with a fixed width of 300 pixels. The flatten() layer of the convolutional neural network requires the input images to have the same numpy shape array; get_edges must return the same shape of (300, 300, 3). A white background of 300 pixels by 300 pixels is created and converted into an 8-bit unsigned integer numpy array. By using the paste function from the Pillow library, the resized cropped image is pasted onto the white background. The pasted, cropped image is converted back into a 32-bit numpy float value because Keras processes numpy arrays in 32-bit float values.

Convolutional Neural Network Architecture

This convolutional neural network architecture uses the Keras Sequential model that acts as instances being layered on top of each other. The convolutional layers in the beginning of the neural network gather pixels of each input data. Depending on the quality and quantity of the image data, the number of output filters will vary. The output filters are the third dimensionality of the convolutional neural layer at the z axis where a rise in output filters and convolutional layers increases the parameter of the network. Increasing the number of filters is necessary as the training data increases, so 128 filters for the first layer of the second convolutional neural network is sufficient for 8,000 JPEG images. The convolutional layer will operate for each 6 pixels by 6 pixels square area in a single 300 pixels by 300 pixels image with a stride of (1,1) overlap. Padding might be needed since 600 doesn't divide evenly by 36. “Valid” padding means that there won’t be additional padding around the borders of the input image whereas "same" padding means padding around the input to create the same dimensions as the output filters. “Same” padding drastically increases the parameters of the network. For instance, a “valid” padding with two convolutional layers and a 3-layer perceptron has 5,063,428 parameters while the “same” padding of the equivalent architecture results in 9,623,422 parameters. The rectifier activation function outputs from the first layer of the convolutional neural network. It is an easier option than the sigmoid activation function because the rectifier activation function max(0, x) creates nonlinearity with the summation of weights without as much computation as the sigmoid activation function.

After the 6 pixels by 6 pixels convolutions are individually formed and activated with the rectifier, the 6 pixels by 6 pixels pool size will down-sample the number of pixels per output filter for the first convolutional layer. Then, a second two-dimensional convolutional neural layer is added to the Sequential model. To enable a growth of learning, the number of output filters is increased twice as much from 128 in the first layer to 256 in the second convolutional layer. The stride is still (1,1) by default in Keras with a convolution size of (6,6) with no padding. The relu activation is also the summation output from the second convolutional layer. After the convolutional filters are created in the second layer, the pixels will be downsampled through maxpool. Maxpooling determines whether each pixel is similar or belong to the surrounding pixels. If it is, the pixel will be visible in the down-sampled layer whereas the pixel removed in the maxpool layer if not.

After the formation of two convolutional layers with maxpool in between, the flattening of these layers allows the convolutional network to be fed into a traditional fully connected neural network. Keras requires that the flattening of the pixelized data input to have a defined shape in the format (width, height, color). The drawback of using convolutional layers is that the images must be the same size. However, convolutional layers are capable of extracting pixels and characteristics from the image data and processes them in layered filters. The basic multi-layer perceptron is not capable of directly extracting features from pixels. In the fully connected layer of the network, a Dense layer is added onto the Sequential model. A Dense layer represents a basic perceptron where 512 neurons are fully connected from the flattening layer to the output of the first Dense layer. The output again will be summed and calculated by the rectifier function. The dropout at the rate of 0.35 is added for the neural network to prevent overfitting. The neurons will be dropped out from the network, removing their connective weight functionality from the next Dense layer of the network.

The final layer will have four units because the number of output classes is four. The activation function for the last layer will be softmax, which squashes the probabilities of the output classes from zero to one. After the architecture of the neural network is defined as a Sequential model, the convolutional and dense layers for the single-label output with multi-class classification will be compiled. The optimizer for compiling the convolutional neural network is one of the most crucial parameters of the learning process. This is because the optimizer determines the learning rate, decay and momentum of the network for correct weight updates during the forward-pass and backpropogation.

Data input into the convolutional neural network architecture

After the architecture of the network, the optimizer and the loss functions are defined, the convolutional neural network will be feed the input images. Keras image generator uses data augmentation to create variations of the image data ranging from scaling, rotating to flipping in a default batch size of 32 per data input. The training data generator consists of rescaling at 1./255, shearing at 0.2, zooming at 0.2, horizontal flip and preprocessing function get_edges.

Training and validation data generators will fit the Sequential model on batches of 32 with real-time data augmentation (Keras.io). The validate data generator will be rescaled at 1./255 and will also be preprocessed with the function get_edges. Fitting the validation and training data generator will "compute quantities required for feature-wise normalization such as standard deviation, mean, and principal components if ZCA whitening is applied," (Keras.io). Data augmentation will prevent overfitting if the augmented data has informative pixels. The early stopping of the convolutional neural network is applied to also prevent overfitting. The network will stop training after three epochs of nondecreasing improvement in the loss values. Finally, after the convolutional neural network is learning through adjusting weight parameters through supervised training with categorical cross entropy loss, the Sequential model is evaluated on a validation set in terms of the validation loss and the validation accuracy. The layers of the model and their corresponding output shapes are printed via the summary() method. The Sequential model is saved as a hierarchical data format (HDF5) file and will be loaded in another file for the testing set, which will predict and evaluate the output classes of the input image data. The testing set file makes predictions based from the trained model that has the network architecture, weight and loss values and optimizer.

Cropped Shapes with Different Dimensions Using Spatial Pyramid Pooling

The spatial pyramid pooling functions as a layer in between the convolutional layers. It will be used to see if the loss of the convolutional neural network can be reduced. The program cnn_sobel_main.py saves the cropped images of different dimensions, 300 pixels by 300 pixels and 300 pixels by 150 pixels in their respective folders. The images are pickled into a tuple of image data array and output labels, merged and split into two separate lists of image data and categorical label encodings for training and validation sets. The cnn_sobel_spp.py file does not use the preprocessing function get_edges because the processed images are already cropped, scaled and saved accordingly in cnn_sobel_main.py. The architecture with spatial pyramid pooling will use Keras Sequential layered model. Spatial pyramid pooling allows different image sizes to be incorporated into the convolutional neural network by replacing the flatten layer with the spatial pyramid pooling([1, 2, 4]) layer. The max-pool layer is also not needed into between the two-dimensional convolutional layer. The parameter [1, 2, 4] list array represents 3 regions with 1, 2x2 and 4x4 max pools, which creates 21 outputs per feature map, (yhenon/keras-spp). The architecture of the network is as follows: convolutional two-dimensional layer, max-pool layer, dropout layer, spatial pyramid pooling layer, dense Layer, dropout layer and dense layer for output classes. The Sequential model is then fitted for the training dataset with 300 pixels by 300 pixels and the other training dataset with 300 pixels by 150 pixels.

Inception Model using Keras Functional API

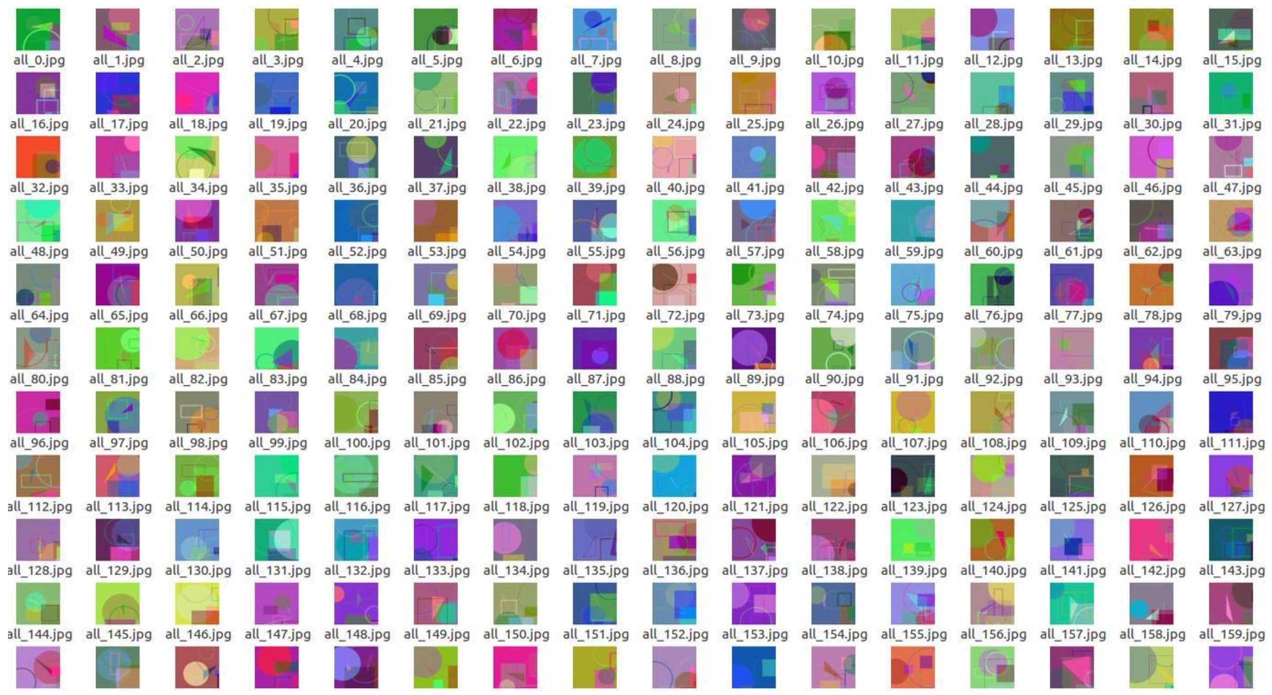
A smaller version of Google's Inception model architecture can be created using Keras functional API. The Input instance allows a defined image size of the shape (300, 300, 3), where Input represents the input data being loaded from the pickle file. The Inception module architecture consists of piling two-dimensional convolutions on top of each other; these layers are called towers. The first layer of the tower will obtain the vectors of input image matrices from Input(). It is followed by the second consecutive layer of the two-dimensional convolutional layer, which will obtain the output from the first layer of the tower. The layering of these two consecutive convolutional layers can continue to as many towers as needed. The layers of the tower are concatenated into a single output and then flattened for the towered layers to feed into the fully connected layer. The fully connected layer follows the dense instance where the first dense layer gets data from the flattened output. There will be dropout after the first dense layer, which gets its input from the first fully connected layer. The functional input into the layer modules continues as needed where the final dense layer contains four neurons for soft-max activation to a probabilistic output class. An inception model is then created from the functional layering by using the model instance where the inputs represent the input data and the outputs represent the resulting layer, which is the fully connected layer. The data generator, early stopping and the fitting of the inception model remains the same as the previous convolutional neural network file.

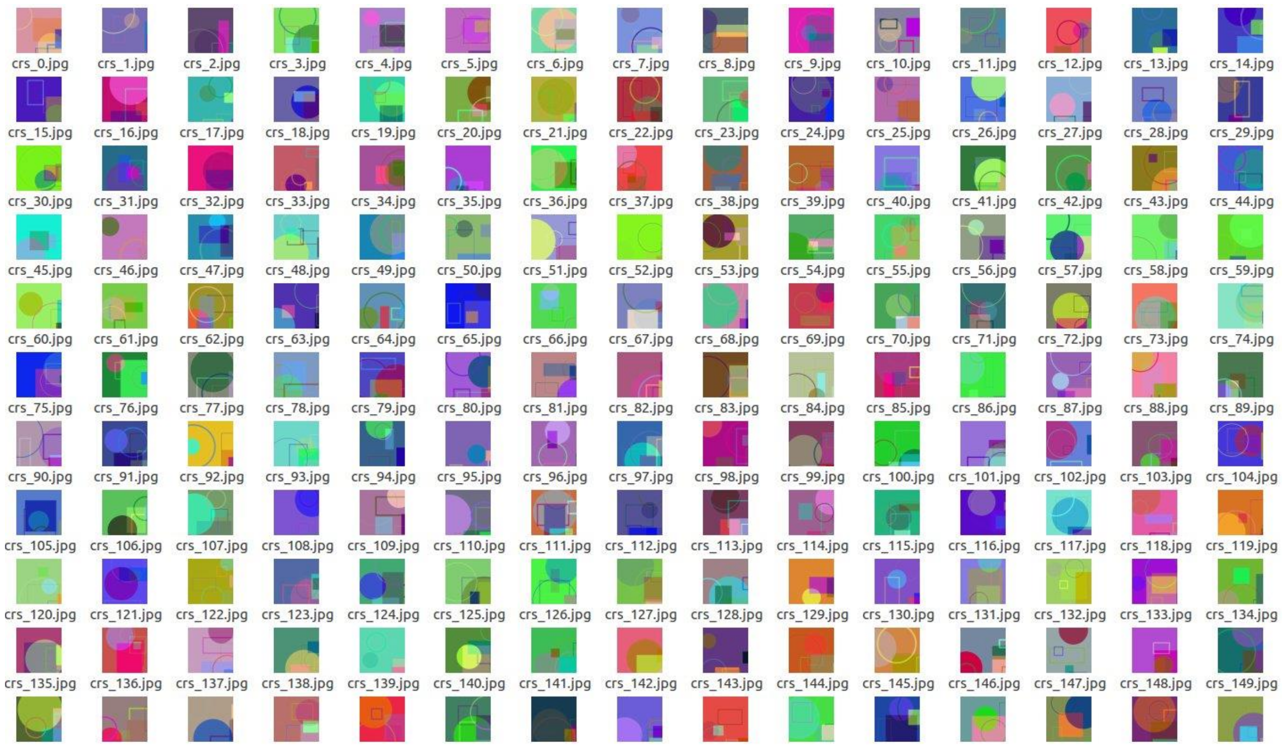
Multi-output (Y-label) Multi-Class Classification

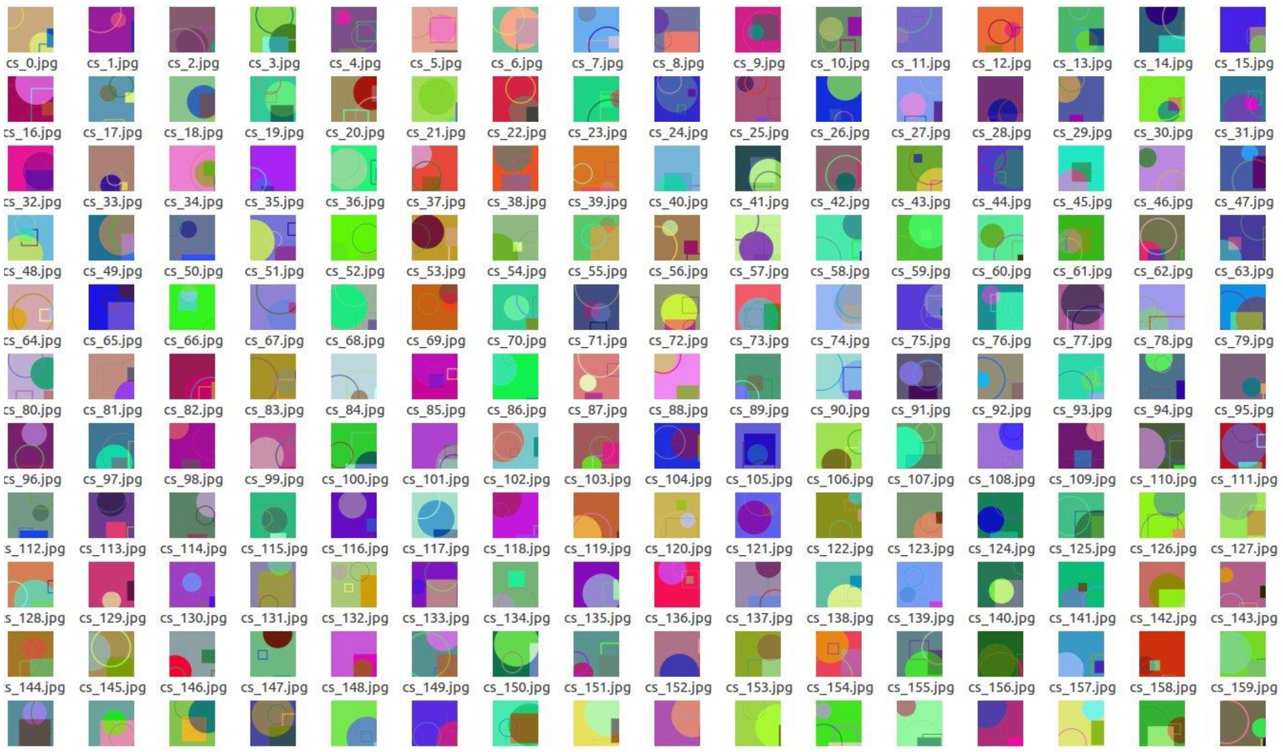
The multi-output classification is different from the single-output classification during the compilation of the Sequential model as well at the last dense layer. The last dense layer is responsible for generating the output classes in supervised training. At the last dense layer, the number of neurons, which is the number of classes, remains the same. The sigmoid activation function is used instead of soft-max because soft-max makes it more evident for a single output classification. Multi-label classification does not have a chance probability of an image pertaining to a single class. A multi-label output image contains multiple classes, so the sigmoid activation will smooth out multiple probabilities of classes pertaining to the data image. The loss function for the Sequential model will be in binary cross-entropy instead of categorical cross-entropy because the binary cross-entropy represents an array of zero or one label of an input image where zero doesn't belong to the class and one belongs to the class. For instance, the array [1 1 1 1] represents that the input image has all the shapes belonging to all the classes.

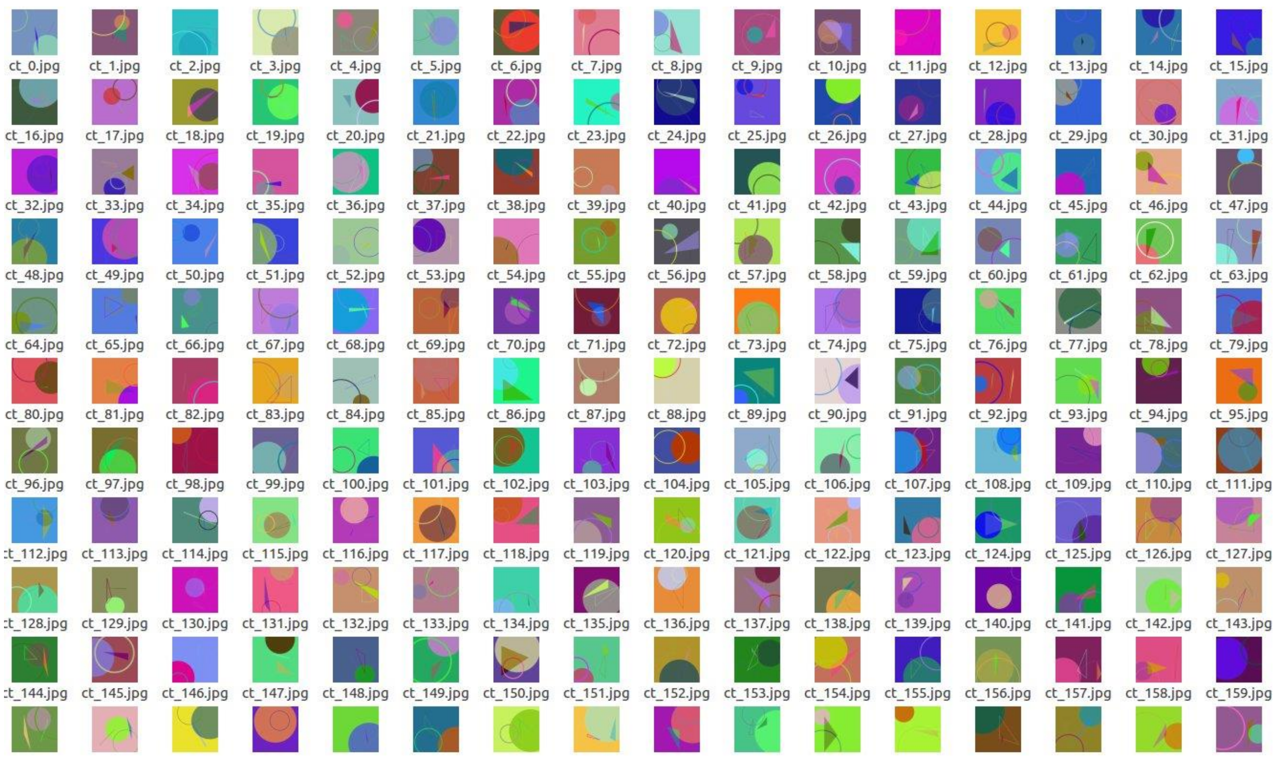
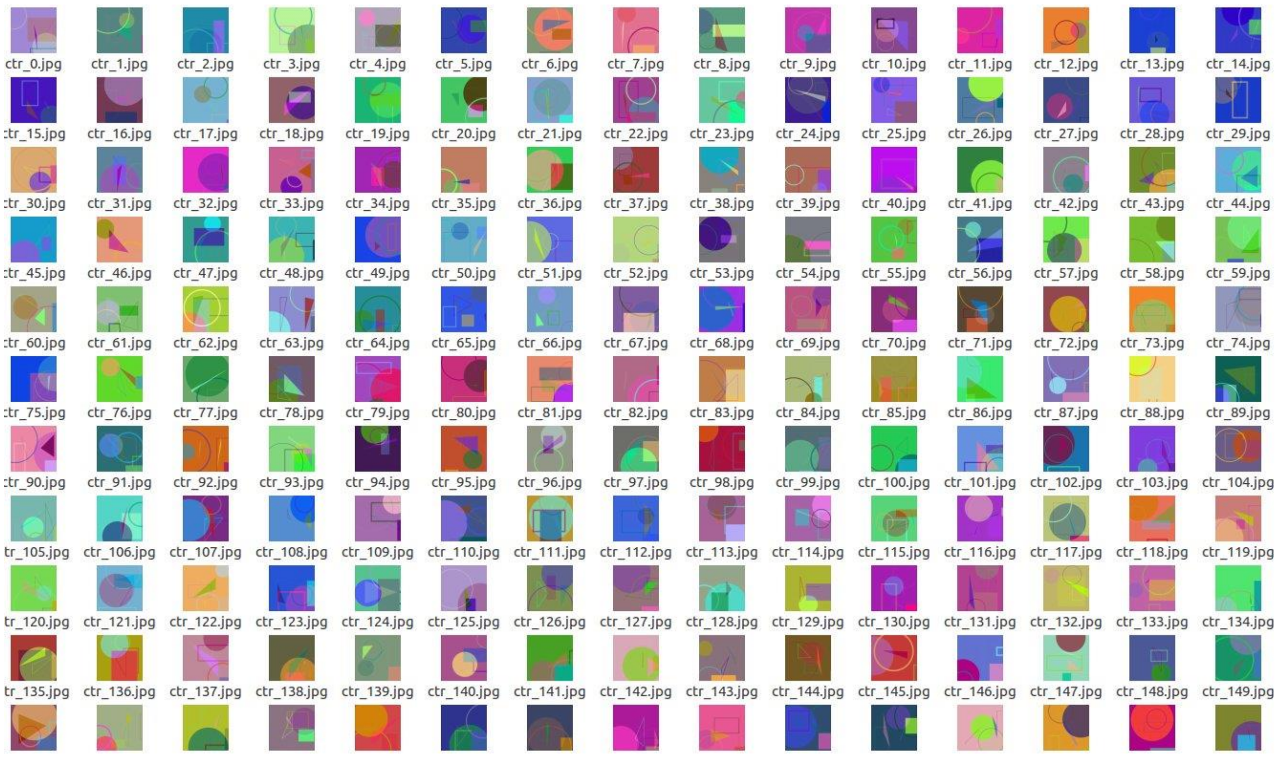
The difference between the single-label output with multi-class classification and multi-label output with multi-class classification is that the multi-label output will learn the intersection of different shapes from different classes not just a single class. The training dataset will consist of two or more classes on a single image consisting of the labels (circle, triangle, square, rectangle), (circle, rectangle), (circle, rectangle, square), (circle, square), (circle, triangle), (circle, triangle, rectangle), (circle, triangle, square), (rectangle, square), (triangle, rectangle), (triangle, rectangle, square) and (triangle, square) to create a consistent layered effect of the images, the same images from the previous training dataset on single-label output is used. For the layering of the shapes on top of each other, each pickle file representing a shape is loaded and returned as a numpy array. Concatenating the numpy array, rather than as a list in the single-label output, will combine the layers of the image data, creating an intersection of different shapes.

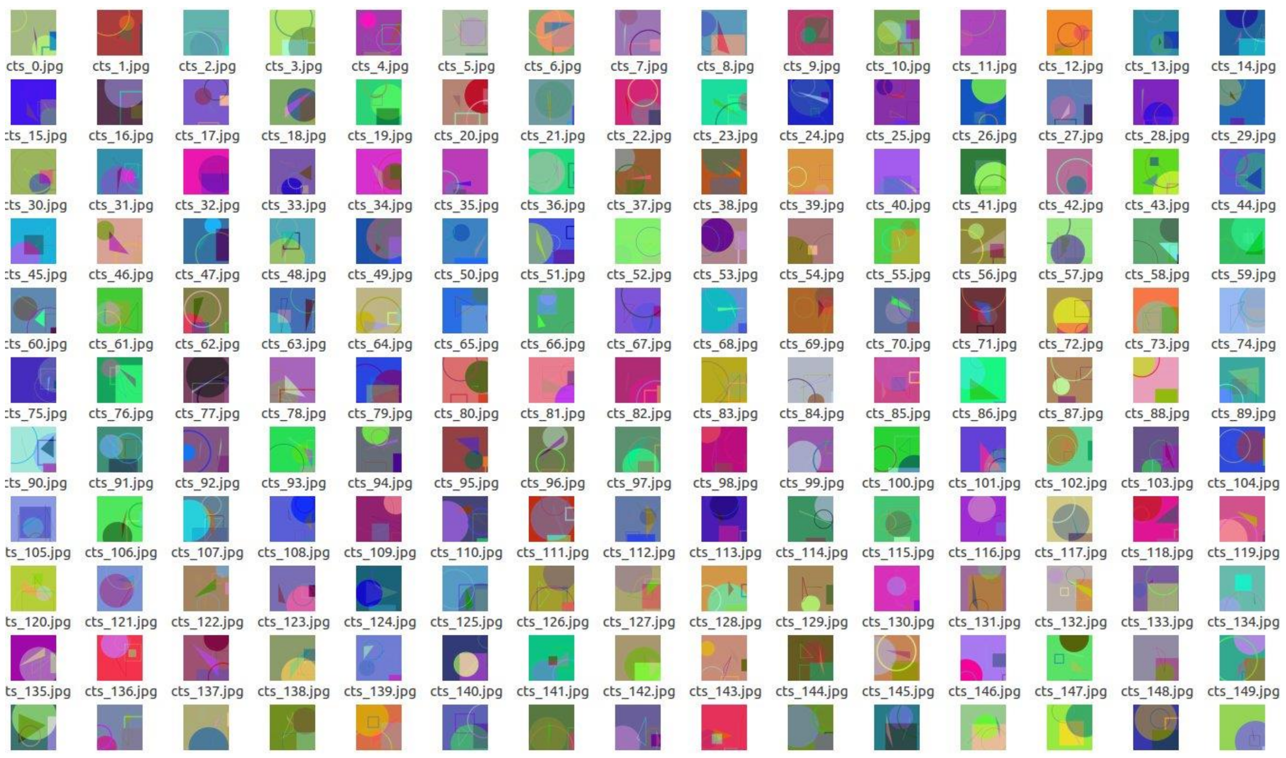
The file merge_pkl_files.py is an option for merge where the intersections of shapes are created by draw.py. It does so by returning each pickled shape as a numpy array and merging them into layers in the next corresponding file. Since each pickled file from the training dataset consists of 2,000 images per shape, the memory mapping of numpy arrays are used for numpy array data in the shape (number of shapes, 300, 300, 3) and their categorical encoding in the shape (number of shapes, 4). The Loader() class uses multiprocessing of independent pool workers to return serialized pickled numpy arrays into function results(), where list arrays are stored globally. A pickle file called individual_data_shapes.pkl is created to represent the memory mapping of the pixelized data of each shape dataset for training and validation. The program load_merge_intersection.py merges all numpy arrays that describes each shape's pixels into layers in the next corresponding file. Each numpy array of data pixels represents a layer of information rather than as a merged list. So, the training dataset will consist of a combination of each shape layer, creating layered intersections of the same shape data of the original dataset. This is for the addition of the convolutional neural network where each training data will have multiple outputs as a label of shapes instead of a single output. This creates two pickle files by running the same program twice with different variables and pickle files called intersection_shapes_one.pkl and intersection_shapes_two.pkl. The point of these pickle files is for another program, save_pickle_images.py, to save these as images by loading the intersected layers of these pickle files.

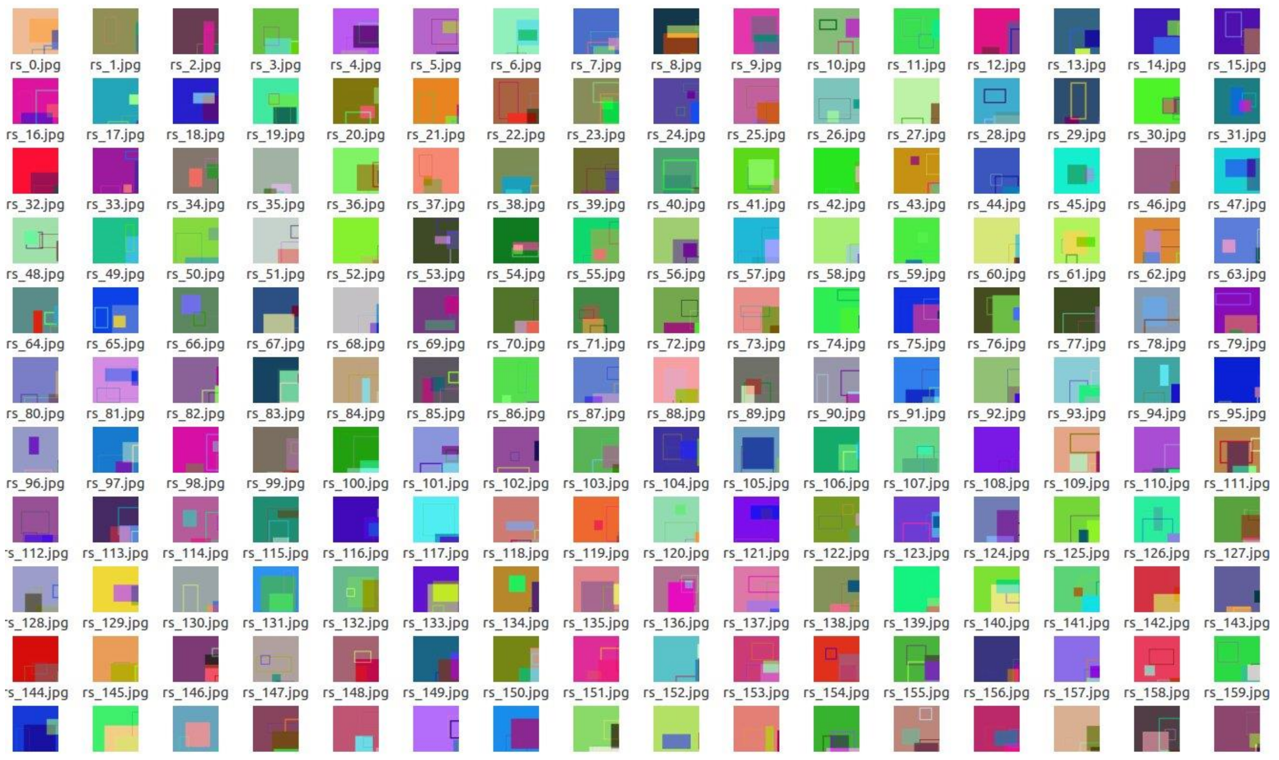

all_0.jpg all_1.jpg all_2.jpg all_3.jpg all_4.jpg all_5.jpg all_6.jpg all_7.jpg all_8.jpg all_9.jpg all_10.jpg all_11.jpg all_12.jpg all_13.jpg all_14.jpg all_15.jpg
all_16.jpg all_17.jpg all_18.jpg all_19.jpg all_20.jpg all_21.jpg all_22.jpg all_23.jpg all_24.jpg all_25.jpg all_26.jpg all_27.jpg all_28.jpg all_29.jpg all_30.jpg all_31.jpg
all_32.jpg all_33.jpg all_34.jpg all_35.jpg all_36.jpg all_37.jpg all_38.jpg all_39.jpg all_40.jpg all_41.jpg all_42.jpg all_43.jpg all_44.jpg all_45.jpg all_46.jpg all_47.jpg
all_48.jpg all_49.jpg all_50.jpg all_51.jpg all_52.jpg all_53.jpg all_54.jpg all_55.jpg all_56.jpg all_57.jpg all_58.jpg all_59.jpg all_60.jpg all_61.jpg all_62.jpg all_63.jpg
all_64.jpg all_65.jpg all_66.jpg all_67.jpg all_68.jpg all_69.jpg all_70.jpg all_71.jpg all_72.jpg all_73.jpg all_74.jpg all_75.jpg all_76.jpg all_77.jpg all_78.jpg all_79.jpg
all_80.jpg all_81.jpg all_82.jpg all_83.jpg all_84.jpg all_85.jpg all_86.jpg all_87.jpg all_88.jpg all_89.jpg all_90.jpg all_91.jpg all_92.jpg all_93.jpg all_94.jpg all_95.jpg
all_96.jpg all_97.jpg all_98.jpg all_99.jpg all_100.jpg all_101.jpg all_102.jpg all_103.jpg all_104.jpg all_105.jpg all_106.jpg all_107.jpg all_108.jpg all_109.jpg all_110.jpg all_111.jpg
all_112.jpg all_113.jpg all_114.jpg all_115.jpg all_116.jpg all_117.jpg all_118.jpg all_119.jpg all_120.jpg all_121.jpg all_122.jpg all_123.jpg all_124.jpg all_125.jpg all_126.jpg all_127.jpg
all_128.jpg all_129.jpg all_130.jpg all_131.jpg all_132.jpg all_133.jpg all_134.jpg all_135.jpg all_136.jpg all_137.jpg all_138.jpg all_139.jpg all_140.jpg all_141.jpg all_142.jpg all_143.jpg
all_144.jpg all_145.jpg all_146.jpg all_147.jpg all_148.jpg all_149.jpg all_150.jpg all_151.jpg all_152.jpg all_153.jpg all_154.jpg all_155.jpg all_156.jpg all_157.jpg all_158.jpg all_159.jpg

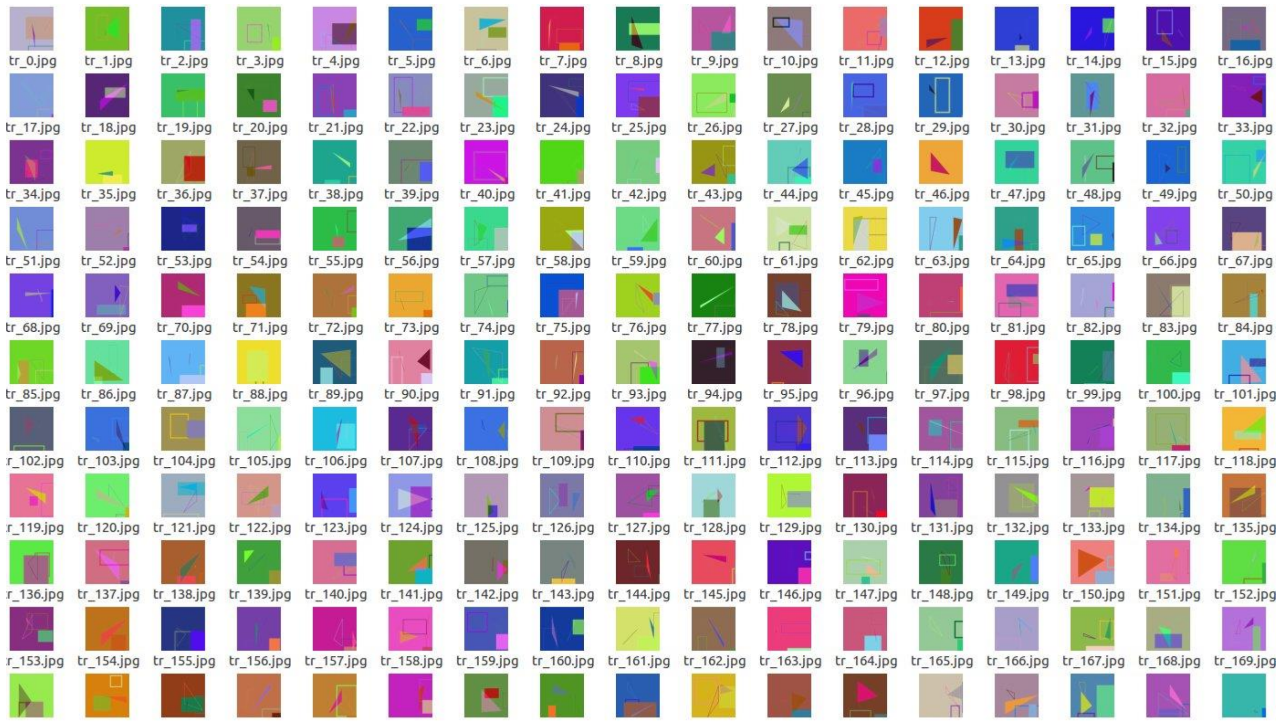

crs_0.jpg
crs_1.jpg
crs_2.jpg
crs_3.jpg
crs_4.jpg
crs_5.jpg
crs_6.jpg
crs_7.jpg
crs_8.jpg
crs_9.jpg
crs_10.jpg
crs_11.jpg
crs_12.jpg
crs_13.jpg
crs_14.jpg
crs_15.jpg
crs_16.jpg
crs_17.jpg
crs_18.jpg
crs_19.jpg
crs_20.jpg
crs_21.jpg
crs_22.jpg
crs_23.jpg
crs_24.jpg
crs_25.jpg
crs_26.jpg
crs_27.jpg
crs_28.jpg
crs_29.jpg
crs_30.jpg
crs_31.jpg
crs_32.jpg
crs_33.jpg
crs_34.jpg
crs_35.jpg
crs_36.jpg
crs_37.jpg
crs_38.jpg
crs_39.jpg
crs_40.jpg
crs_41.jpg
crs_42.jpg
crs_43.jpg
crs_44.jpg
crs_45.jpg
crs_46.jpg
crs_47.jpg
crs_48.jpg
crs_49.jpg
crs_50.jpg
crs_51.jpg
crs_52.jpg
crs_53.jpg
crs_54.jpg
crs_55.jpg
crs_56.jpg
crs_57.jpg
crs_58.jpg
crs_59.jpg
crs_60.jpg
crs_61.jpg
crs_62.jpg
crs_63.jpg
crs_64.jpg
crs_65.jpg
crs_66.jpg
crs_67.jpg
crs_68.jpg
crs_69.jpg
crs_70.jpg
crs_71.jpg
crs_72.jpg
crs_73.jpg
crs_74.jpg
crs_75.jpg
crs_76.jpg
crs_77.jpg
crs_78.jpg
crs_79.jpg
crs_80.jpg
crs_81.jpg
crs_82.jpg
crs_83.jpg
crs_84.jpg
crs_85.jpg
crs_86.jpg
crs_87.jpg
crs_88.jpg
crs_89.jpg
crs_90.jpg
crs_91.jpg
crs_92.jpg
crs_93.jpg
crs_94.jpg
crs_95.jpg
crs_96.jpg
crs_97.jpg
crs_98.jpg
crs_99.jpg
crs_100.jpg
crs_101.jpg
crs_102.jpg
crs_103.jpg
crs_104.jpg
crs_105.jpg
crs_106.jpg
crs_107.jpg
crs_108.jpg
crs_109.jpg
crs_110.jpg
crs_111.jpg
crs_112.jpg
crs_113.jpg
crs_114.jpg
crs_115.jpg
crs_116.jpg
crs_117.jpg
crs_118.jpg
crs_119.jpg
crs_120.jpg
crs_121.jpg
crs_122.jpg
crs_123.jpg
crs_124.jpg
crs_125.jpg
crs_126.jpg
crs_127.jpg
crs_128.jpg
crs_129.jpg
crs_130.jpg
crs_131.jpg
crs_132.jpg
crs_133.jpg
crs_134.jpg
crs_135.jpg
crs_136.jpg
crs_137.jpg
crs_138.jpg
crs_139.jpg
crs_140.jpg
crs_141.jpg
crs_142.jpg
crs_143.jpg
crs_144.jpg
crs_145.jpg
crs_146.jpg
crs_147.jpg
crs_148.jpg
crs_149.jpg

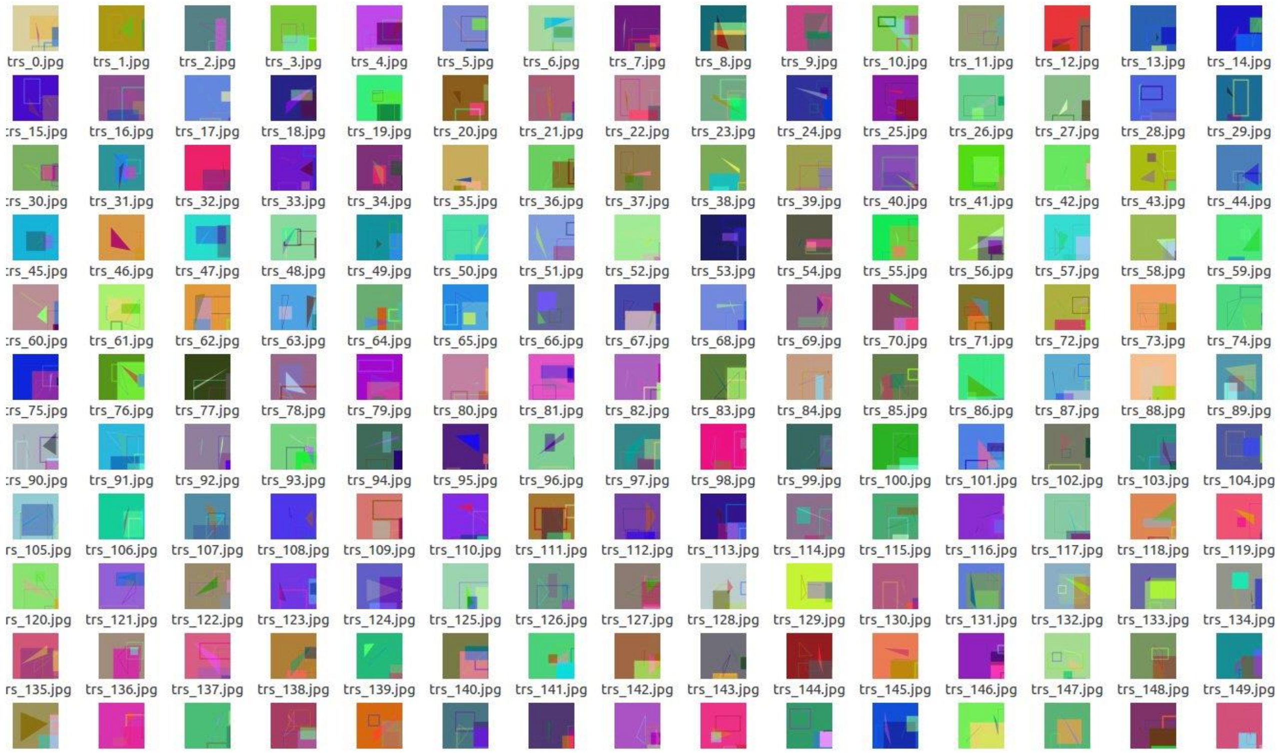

cs_0.jpg
cs_1.jpg
cs_2.jpg
cs_3.jpg
cs_4.jpg
cs_5.jpg
cs_6.jpg
cs_7.jpg
cs_8.jpg
cs_9.jpg
cs_10.jpg
cs_11.jpg
cs_12.jpg
cs_13.jpg
cs_14.jpg
cs_15.jpg
cs_16.jpg
cs_17.jpg
cs_18.jpg
cs_19.jpg
cs_20.jpg
cs_21.jpg
cs_22.jpg
cs_23.jpg
cs_24.jpg
cs_25.jpg
cs_26.jpg
cs_27.jpg
cs_28.jpg
cs_29.jpg
cs_30.jpg
cs_31.jpg
cs_32.jpg
cs_33.jpg
cs_34.jpg
cs_35.jpg
cs_36.jpg
cs_37.jpg
cs_38.jpg
cs_39.jpg
cs_40.jpg
cs_41.jpg
cs_42.jpg
cs_43.jpg
cs_44.jpg
cs_45.jpg
cs_46.jpg
cs_47.jpg
cs_48.jpg
cs_49.jpg
cs_50.jpg
cs_51.jpg
cs_52.jpg
cs_53.jpg
cs_54.jpg
cs_55.jpg
cs_56.jpg
cs_57.jpg
cs_58.jpg
cs_59.jpg
cs_60.jpg
cs_61.jpg
cs_62.jpg
cs_63.jpg
cs_64.jpg
cs_65.jpg
cs_66.jpg
cs_67.jpg
cs_68.jpg
cs_69.jpg
cs_70.jpg
cs_71.jpg
cs_72.jpg
cs_73.jpg
cs_74.jpg
cs_75.jpg
cs_76.jpg
cs_77.jpg
cs_78.jpg
cs_79.jpg
cs_80.jpg
cs_81.jpg
cs_82.jpg
cs_83.jpg
cs_84.jpg
cs_85.jpg
cs_86.jpg
cs_87.jpg
cs_88.jpg
cs_89.jpg
cs_90.jpg
cs_91.jpg
cs_92.jpg
cs_93.jpg
cs_94.jpg
cs_95.jpg
cs_96.jpg
cs_97.jpg
cs_98.jpg
cs_99.jpg
cs_100.jpg
cs_101.jpg
cs_102.jpg
cs_103.jpg
cs_104.jpg
cs_105.jpg
cs_106.jpg
cs_107.jpg
cs_108.jpg
cs_109.jpg
cs_110.jpg
cs_111.jpg
s_112.jpg
cs_113.jpg
cs_114.jpg
cs_115.jpg
cs_116.jpg
cs_117.jpg
cs_118.jpg
cs_119.jpg
cs_120.jpg
cs_121.jpg
cs_122.jpg
cs_123.jpg
cs_124.jpg
cs_125.jpg
cs_126.jpg
cs_127.jpg
s_128.jpg
cs_129.jpg
cs_130.jpg
cs_131.jpg
cs_132.jpg
cs_133.jpg
cs_134.jpg
cs_135.jpg
cs_136.jpg
cs_137.jpg
cs_138.jpg
cs_139.jpg
cs_140.jpg
cs_141.jpg
cs_142.jpg
cs_143.jpg
s_144.jpg
cs_145.jpg
cs_146.jpg
cs_147.jpg
cs_148.jpg
cs_149.jpg
cs_150.jpg
cs_151.jpg
cs_152.jpg
cs_153.jpg
cs_154.jpg
cs_155.jpg
cs_156.jpg
cs_157.jpg
cs_158.jpg
cs_159.jpg

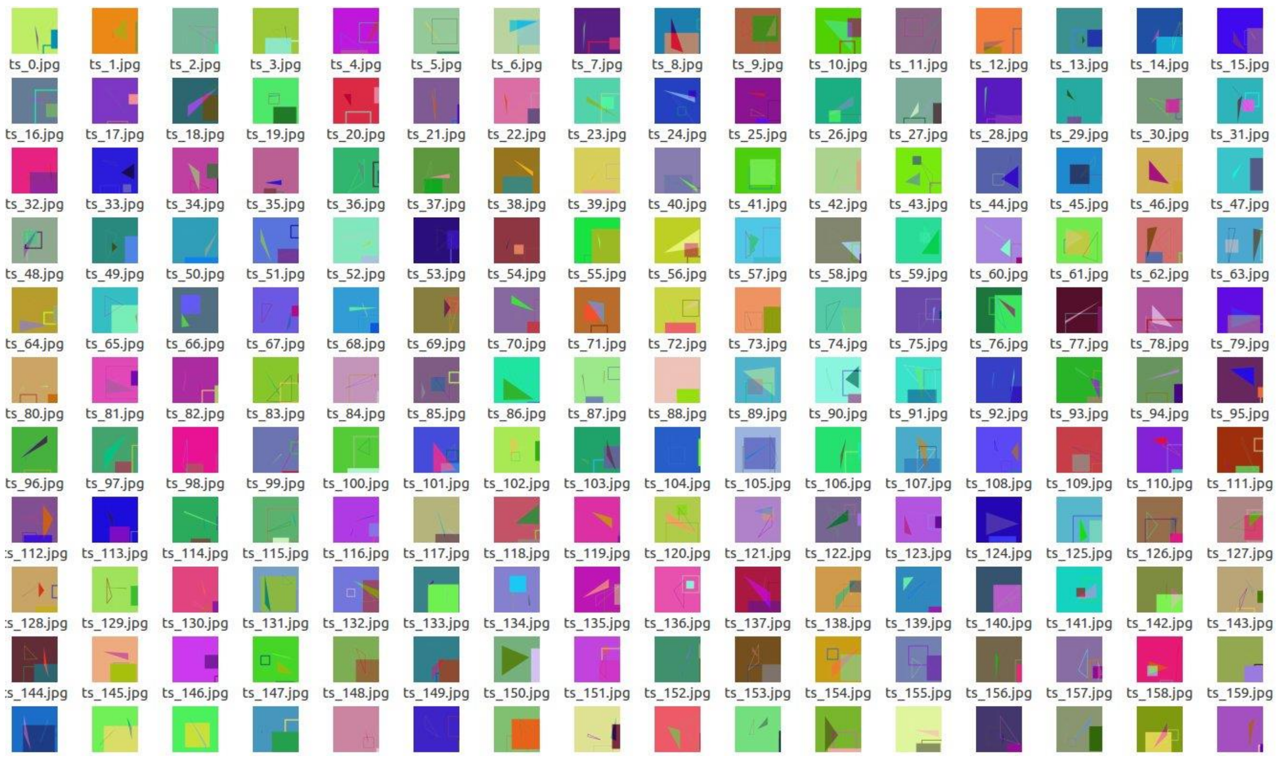

ct_0.jpg ct_1.jpg ct_2.jpg ct_3.jpg ct_4.jpg ct_5.jpg ct_6.jpg ct_7.jpg ct_8.jpg ct_9.jpg ct_10.jpg ct_11.jpg ct_12.jpg ct_13.jpg ct_14.jpg ct_15.jpg
ct_16.jpg ct_17.jpg ct_18.jpg ct_19.jpg ct_20.jpg ct_21.jpg ct_22.jpg ct_23.jpg ct_24.jpg ct_25.jpg ct_26.jpg ct_27.jpg ct_28.jpg ct_29.jpg ct_30.jpg ct_31.jpg
ct_32.jpg ct_33.jpg ct_34.jpg ct_35.jpg ct_36.jpg ct_37.jpg ct_38.jpg ct_39.jpg ct_40.jpg ct_41.jpg ct_42.jpg ct_43.jpg ct_44.jpg ct_45.jpg ct_46.jpg ct_47.jpg
ct_48.jpg ct_49.jpg ct_50.jpg ct_51.jpg ct_52.jpg ct_53.jpg ct_54.jpg ct_55.jpg ct_56.jpg ct_57.jpg ct_58.jpg ct_59.jpg ct_60.jpg ct_61.jpg ct_62.jpg ct_63.jpg
ct_64.jpg ct_65.jpg ct_66.jpg ct_67.jpg ct_68.jpg ct_69.jpg ct_70.jpg ct_71.jpg ct_72.jpg ct_73.jpg ct_74.jpg ct_75.jpg ct_76.jpg ct_77.jpg ct_78.jpg ct_79.jpg
ct_80.jpg ct_81.jpg ct_82.jpg ct_83.jpg ct_84.jpg ct_85.jpg ct_86.jpg ct_87.jpg ct_88.jpg ct_89.jpg ct_90.jpg ct_91.jpg ct_92.jpg ct_93.jpg ct_94.jpg ct_95.jpg
ct_96.jpg ct_97.jpg ct_98.jpg ct_99.jpg ct_100.jpg ct_101.jpg ct_102.jpg ct_103.jpg ct_104.jpg ct_105.jpg ct_106.jpg ct_107.jpg ct_108.jpg ct_109.jpg ct_110.jpg ct_111.jpg
t_112.jpg ct_113.jpg ct_114.jpg ct_115.jpg ct_116.jpg ct_117.jpg ct_118.jpg ct_119.jpg ct_120.jpg ct_121.jpg ct_122.jpg ct_123.jpg ct_124.jpg ct_125.jpg ct_126.jpg ct_127.jpg
t_128.jpg ct_129.jpg ct_130.jpg ct_131.jpg ct_132.jpg ct_133.jpg ct_134.jpg ct_135.jpg ct_136.jpg ct_137.jpg ct_138.jpg ct_139.jpg ct_140.jpg ct_141.jpg ct_142.jpg ct_143.jpg
t_144.jpg ct_145.jpg ct_146.jpg ct_147.jpg ct_148.jpg ct_149.jpg ct_150.jpg ct_151.jpg ct_152.jpg ct_153.jpg ct_154.jpg ct_155.jpg ct_156.jpg ct_157.jpg ct_158.jpg ct_159.jpg

ctr_0.jpg ctr_1.jpg ctr_2.jpg ctr_3.jpg ctr_4.jpg ctr_5.jpg ctr_6.jpg ctr_7.jpg ctr_8.jpg ctr_9.jpg ctr_10.jpg ctr_11.jpg ctr_12.jpg ctr_13.jpg ctr_14.jpg
tr_15.jpg ctr_16.jpg ctr_17.jpg ctr_18.jpg ctr_19.jpg ctr_20.jpg ctr_21.jpg ctr_22.jpg ctr_23.jpg ctr_24.jpg ctr_25.jpg ctr_26.jpg ctr_27.jpg ctr_28.jpg ctr_29.jpg
tr_30.jpg ctr_31.jpg ctr_32.jpg ctr_33.jpg ctr_34.jpg ctr_35.jpg ctr_36.jpg ctr_37.jpg ctr_38.jpg ctr_39.jpg ctr_40.jpg ctr_41.jpg ctr_42.jpg ctr_43.jpg ctr_44.jpg
tr_45.jpg ctr_46.jpg ctr_47.jpg ctr_48.jpg ctr_49.jpg ctr_50.jpg ctr_51.jpg ctr_52.jpg ctr_53.jpg ctr_54.jpg ctr_55.jpg ctr_56.jpg ctr_57.jpg ctr_58.jpg ctr_59.jpg
tr_60.jpg ctr_61.jpg ctr_62.jpg ctr_63.jpg ctr_64.jpg ctr_65.jpg ctr_66.jpg ctr_67.jpg ctr_68.jpg ctr_69.jpg ctr_70.jpg ctr_71.jpg ctr_72.jpg ctr_73.jpg ctr_74.jpg
tr_75.jpg ctr_76.jpg ctr_77.jpg ctr_78.jpg ctr_79.jpg ctr_80.jpg ctr_81.jpg ctr_82.jpg ctr_83.jpg ctr_84.jpg ctr_85.jpg ctr_86.jpg ctr_87.jpg ctr_88.jpg ctr_89.jpg
tr_90.jpg ctr_91.jpg ctr_92.jpg ctr_93.jpg ctr_94.jpg ctr_95.jpg ctr_96.jpg ctr_97.jpg ctr_98.jpg ctr_99.jpg ctr_100.jpg ctr_101.jpg ctr_102.jpg ctr_103.jpg ctr_104.jpg
tr_105.jpg ctr_106.jpg ctr_107.jpg ctr_108.jpg ctr_109.jpg ctr_110.jpg ctr_111.jpg ctr_112.jpg ctr_113.jpg ctr_114.jpg ctr_115.jpg ctr_116.jpg ctr_117.jpg ctr_118.jpg ctr_119.jpg
tr_120.jpg ctr_121.jpg ctr_122.jpg ctr_123.jpg ctr_124.jpg ctr_125.jpg ctr_126.jpg ctr_127.jpg ctr_128.jpg ctr_129.jpg ctr_130.jpg ctr_131.jpg ctr_132.jpg ctr_133.jpg ctr_134.jpg
tr_135.jpg ctr_136.jpg ctr_137.jpg ctr_138.jpg ctr_139.jpg ctr_140.jpg ctr_141.jpg ctr_142.jpg ctr_143.jpg ctr_144.jpg ctr_145.jpg ctr_146.jpg ctr_147.jpg ctr_148.jpg ctr_149.jpg


cts_0.jpg cts_1.jpg cts_2.jpg cts_3.jpg cts_4.jpg cts_5.jpg cts_6.jpg cts_7.jpg cts_8.jpg cts_9.jpg cts_10.jpg cts_11.jpg cts_12.jpg cts_13.jpg cts_14.jpg
ts_15.jpg cts_16.jpg cts_17.jpg cts_18.jpg cts_19.jpg cts_20.jpg cts_21.jpg cts_22.jpg cts_23.jpg cts_24.jpg cts_25.jpg cts_26.jpg cts_27.jpg cts_28.jpg cts_29.jpg
ts_30.jpg cts_31.jpg cts_32.jpg cts_33.jpg cts_34.jpg cts_35.jpg cts_36.jpg cts_37.jpg cts_38.jpg cts_39.jpg cts_40.jpg cts_41.jpg cts_42.jpg cts_43.jpg cts_44.jpg
ts_45.jpg cts_46.jpg cts_47.jpg cts_48.jpg cts_49.jpg cts_50.jpg cts_51.jpg cts_52.jpg cts_53.jpg cts_54.jpg cts_55.jpg cts_56.jpg cts_57.jpg cts_58.jpg cts_59.jpg
ts_60.jpg cts_61.jpg cts_62.jpg cts_63.jpg cts_64.jpg cts_65.jpg cts_66.jpg cts_67.jpg cts_68.jpg cts_69.jpg cts_70.jpg cts_71.jpg cts_72.jpg cts_73.jpg cts_74.jpg
ts_75.jpg cts_76.jpg cts_77.jpg cts_78.jpg cts_79.jpg cts_80.jpg cts_81.jpg cts_82.jpg cts_83.jpg cts_84.jpg cts_85.jpg cts_86.jpg cts_87.jpg cts_88.jpg cts_89.jpg
ts_90.jpg cts_91.jpg cts_92.jpg cts_93.jpg cts_94.jpg cts_95.jpg cts_96.jpg cts_97.jpg cts_98.jpg cts_99.jpg cts_100.jpg cts_101.jpg cts_102.jpg cts_103.jpg cts_104.jpg
ts_105.jpg cts_106.jpg cts_107.jpg cts_108.jpg cts_109.jpg cts_110.jpg cts_111.jpg cts_112.jpg cts_113.jpg cts_114.jpg cts_115.jpg cts_116.jpg cts_117.jpg cts_118.jpg cts_119.jpg
ts_120.jpg cts_121.jpg cts_122.jpg cts_123.jpg cts_124.jpg cts_125.jpg cts_126.jpg cts_127.jpg cts_128.jpg cts_129.jpg cts_130.jpg cts_131.jpg cts_132.jpg cts_133.jpg cts_134.jpg
ts_135.jpg cts_136.jpg cts_137.jpg cts_138.jpg cts_139.jpg cts_140.jpg cts_141.jpg cts_142.jpg cts_143.jpg cts_144.jpg cts_145.jpg cts_146.jpg cts_147.jpg cts_148.jpg cts_149.jpg

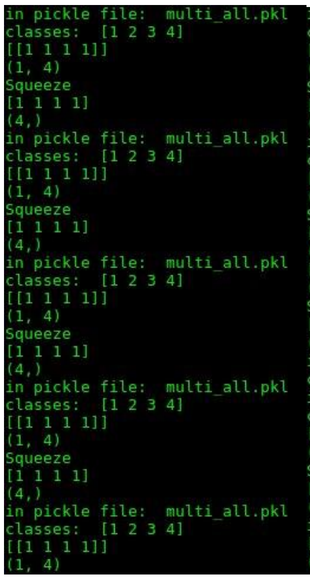

rs_0.jpg rs_1.jpg rs_2.jpg rs_3.jpg rs_4.jpg rs_5.jpg rs_6.jpg rs_7.jpg rs_8.jpg rs_9.jpg rs_10.jpg rs_11.jpg rs_12.jpg rs_13.jpg rs_14.jpg rs_15.jpg
rs_16.jpg rs_17.jpg rs_18.jpg rs_19.jpg rs_20.jpg rs_21.jpg rs_22.jpg rs_23.jpg rs_24.jpg rs_25.jpg rs_26.jpg rs_27.jpg rs_28.jpg rs_29.jpg rs_30.jpg rs_31.jpg
rs_32.jpg rs_33.jpg rs_34.jpg rs_35.jpg rs_36.jpg rs_37.jpg rs_38.jpg rs_39.jpg rs_40.jpg rs_41.jpg rs_42.jpg rs_43.jpg rs_44.jpg rs_45.jpg rs_46.jpg rs_47.jpg
rs_48.jpg rs_49.jpg rs_50.jpg rs_51.jpg rs_52.jpg rs_53.jpg rs_54.jpg rs_55.jpg rs_56.jpg rs_57.jpg rs_58.jpg rs_59.jpg rs_60.jpg rs_61.jpg rs_62.jpg rs_63.jpg
rs_64.jpg rs_65.jpg rs_66.jpg rs_67.jpg rs_68.jpg rs_69.jpg rs_70.jpg rs_71.jpg rs_72.jpg rs_73.jpg rs_74.jpg rs_75.jpg rs_76.jpg rs_77.jpg rs_78.jpg rs_79.jpg
rs_80.jpg rs_81.jpg rs_82.jpg rs_83.jpg rs_84.jpg rs_85.jpg rs_86.jpg rs_87.jpg rs_88.jpg rs_89.jpg rs_90.jpg rs_91.jpg rs_92.jpg rs_93.jpg rs_94.jpg rs_95.jpg
rs_96.jpg rs_97.jpg rs_98.jpg rs_99.jpg rs_100.jpg rs_101.jpg rs_102.jpg rs_103.jpg rs_104.jpg rs_105.jpg rs_106.jpg rs_107.jpg rs_108.jpg rs_109.jpg rs_110.jpg rs_111.jpg
s_112.jpg rs_113.jpg rs_114.jpg rs_115.jpg rs_116.jpg rs_117.jpg rs_118.jpg rs_119.jpg rs_120.jpg rs_121.jpg rs_122.jpg rs_123.jpg rs_124.jpg rs_125.jpg rs_126.jpg rs_127.jpg
s_128.jpg rs_129.jpg rs_130.jpg rs_131.jpg rs_132.jpg rs_133.jpg rs_134.jpg rs_135.jpg rs_136.jpg rs_137.jpg rs_138.jpg rs_139.jpg rs_140.jpg rs_141.jpg rs_142.jpg rs_143.jpg
s_144.jpg rs_145.jpg rs_146.jpg rs_147.jpg rs_148.jpg rs_149.jpg rs_150.jpg rs_151.jpg rs_152.jpg rs_153.jpg rs_154.jpg rs_155.jpg rs_156.jpg rs_157.jpg rs_158.jpg rs_159.jpg


tr_0.jpg
tr_1.jpg
tr_2.jpg
tr_3.jpg
tr_4.jpg
tr_5.jpg
tr_6.jpg
tr_7.jpg
tr_8.jpg
tr_9.jpg
tr_10.jpg
tr_11.jpg
tr_12.jpg
tr_13.jpg
tr_14.jpg
tr_15.jpg
tr_16.jpg
tr_17.jpg
tr_18.jpg
tr_19.jpg
tr_20.jpg
tr_21.jpg
tr_22.jpg
tr_23.jpg
tr_24.jpg
tr_25.jpg
tr_26.jpg
tr_27.jpg
tr_28.jpg
tr_29.jpg
tr_30.jpg
tr_31.jpg
tr_32.jpg
tr_33.jpg
tr_34.jpg
tr_35.jpg
tr_36.jpg
tr_37.jpg
tr_38.jpg
tr_39.jpg
tr_40.jpg
tr_41.jpg
tr_42.jpg
tr_43.jpg
tr_44.jpg
tr_45.jpg
tr_46.jpg
tr_47.jpg
tr_48.jpg
tr_49.jpg
tr_50.jpg
tr_51.jpg
tr_52.jpg
tr_53.jpg
tr_54.jpg
tr_55.jpg
tr_56.jpg
tr_57.jpg
tr_58.jpg
tr_59.jpg
tr_60.jpg
tr_61.jpg
tr_62.jpg
tr_63.jpg
tr_64.jpg
tr_65.jpg
tr_66.jpg
tr_67.jpg
tr_68.jpg
tr_69.jpg
tr_70.jpg
tr_71.jpg
tr_72.jpg
tr_73.jpg
tr_74.jpg
tr_75.jpg
tr_76.jpg
tr_77.jpg
tr_78.jpg
tr_79.jpg
tr_80.jpg
tr_81.jpg
tr_82.jpg
tr_83.jpg
tr_84.jpg
tr_85.jpg
tr_86.jpg
tr_87.jpg
tr_88.jpg
tr_89.jpg
tr_90.jpg
tr_91.jpg
tr_92.jpg
tr_93.jpg
tr_94.jpg
tr_95.jpg
tr_96.jpg
tr_97.jpg
tr_98.jpg
tr_99.jpg
tr_100.jpg
tr_101.jpg
r_102.jpg
tr_103.jpg
tr_104.jpg
tr_105.jpg
tr_106.jpg
tr_107.jpg
tr_108.jpg
tr_109.jpg
tr_110.jpg
tr_111.jpg
tr_112.jpg
tr_113.jpg
tr_114.jpg
tr_115.jpg
tr_116.jpg
tr_117.jpg
tr_118.jpg
r_119.jpg
tr_120.jpg
tr_121.jpg
tr_122.jpg
tr_123.jpg
tr_124.jpg
tr_125.jpg
tr_126.jpg
tr_127.jpg
tr_128.jpg
tr_129.jpg
tr_130.jpg
tr_131.jpg
tr_132.jpg
tr_133.jpg
tr_134.jpg
tr_135.jpg
r_136.jpg
tr_137.jpg
tr_138.jpg
tr_139.jpg
tr_140.jpg
tr_141.jpg
tr_142.jpg
tr_143.jpg
tr_144.jpg
tr_145.jpg
tr_146.jpg
tr_147.jpg
tr_148.jpg
tr_149.jpg
tr_150.jpg
tr_151.jpg
tr_152.jpg
r_153.jpg
tr_154.jpg
tr_155.jpg
tr_156.jpg
tr_157.jpg
tr_158.jpg
tr_159.jpg
tr_160.jpg
tr_161.jpg
tr_162.jpg
tr_163.jpg
tr_164.jpg
tr_165.jpg
tr_166.jpg
tr_167.jpg
tr_168.jpg
tr_169.jpg

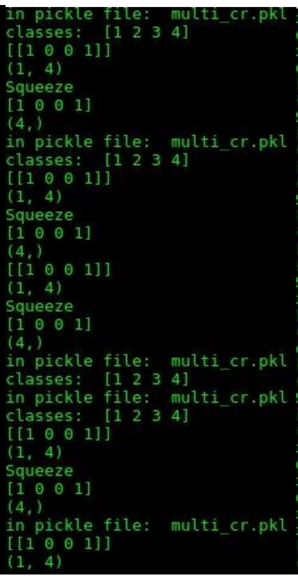

trs_0.jpg
trs_1.jpg
trs_2.jpg
trs_3.jpg
trs_4.jpg
trs_5.jpg
trs_6.jpg
trs_7.jpg
trs_8.jpg
trs_9.jpg
trs_10.jpg
trs_11.jpg
trs_12.jpg
trs_13.jpg
trs_14.jpg
rs_15.jpg
trs_16.jpg
trs_17.jpg
trs_18.jpg
trs_19.jpg
trs_20.jpg
trs_21.jpg
trs_22.jpg
trs_23.jpg
trs_24.jpg
trs_25.jpg
trs_26.jpg
trs_27.jpg
trs_28.jpg
trs_29.jpg
rs_30.jpg
trs_31.jpg
trs_32.jpg
trs_33.jpg
trs_34.jpg
trs_35.jpg
trs_36.jpg
trs_37.jpg
trs_38.jpg
trs_39.jpg
trs_40.jpg
trs_41.jpg
trs_42.jpg
trs_43.jpg
trs_44.jpg
rs_45.jpg
trs_46.jpg
trs_47.jpg
trs_48.jpg
trs_49.jpg
trs_50.jpg
trs_51.jpg
trs_52.jpg
trs_53.jpg
trs_54.jpg
trs_55.jpg
trs_56.jpg
trs_57.jpg
trs_58.jpg
trs_59.jpg
rs_60.jpg
trs_61.jpg
trs_62.jpg
trs_63.jpg
trs_64.jpg
trs_65.jpg
trs_66.jpg
trs_67.jpg
trs_68.jpg
trs_69.jpg
trs_70.jpg
trs_71.jpg
trs_72.jpg
trs_73.jpg
trs_74.jpg
rs_75.jpg
trs_76.jpg
trs_77.jpg
trs_78.jpg
trs_79.jpg
trs_80.jpg
trs_81.jpg
trs_82.jpg
trs_83.jpg
trs_84.jpg
trs_85.jpg
trs_86.jpg
trs_87.jpg
trs_88.jpg
trs_89.jpg
rs_90.jpg
trs_91.jpg
trs_92.jpg
trs_93.jpg
trs_94.jpg
trs_95.jpg
trs_96.jpg
trs_97.jpg
trs_98.jpg
trs_99.jpg
trs_100.jpg
trs_101.jpg
trs_102.jpg
trs_103.jpg
trs_104.jpg
rs_105.jpg
trs_106.jpg
trs_107.jpg
trs_108.jpg
trs_109.jpg
trs_110.jpg
trs_111.jpg
trs_112.jpg
trs_113.jpg
trs_114.jpg
trs_115.jpg
trs_116.jpg
trs_117.jpg
trs_118.jpg
trs_119.jpg
rs_120.jpg
trs_121.jpg
trs_122.jpg
trs_123.jpg
trs_124.jpg
trs_125.jpg
trs_126.jpg
trs_127.jpg
trs_128.jpg
trs_129.jpg
trs_130.jpg
trs_131.jpg
trs_132.jpg
trs_133.jpg
trs_134.jpg
rs_135.jpg
trs_136.jpg
trs_137.jpg
trs_138.jpg
trs_139.jpg
trs_140.jpg
trs_141.jpg
trs_142.jpg
trs_143.jpg
trs_144.jpg
trs_145.jpg
trs_146.jpg
trs_147.jpg
trs_148.jpg
trs_149.jpg



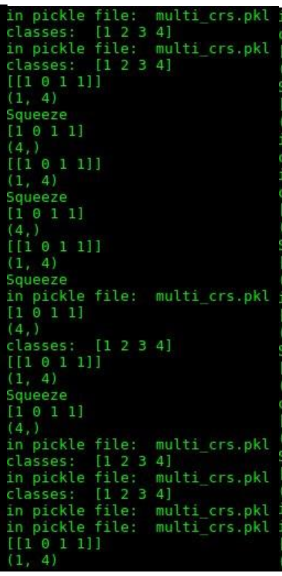
After the images for the multi-label output with multi-class classification are created, the images will be organized in a pickle file like the one with the single-label output. The difference is in the second data structure, which is y-label, that is part of the (data image, y-label) tuple. This y-label will be created by the multi-label binarizer class. It is a numpy array in binary representing zero for no class representation and one for class representation. For example, an image that consists of all shapes will have the y-label [1 1 1 1]. An image that has a circle, triangle and rectangle will have the y_label [1 1 0 1]. The numpy.squeeze function is needed to remove the extra shape axis created by multi-label binarizer so that Keras can process the numpy list structure correctly.



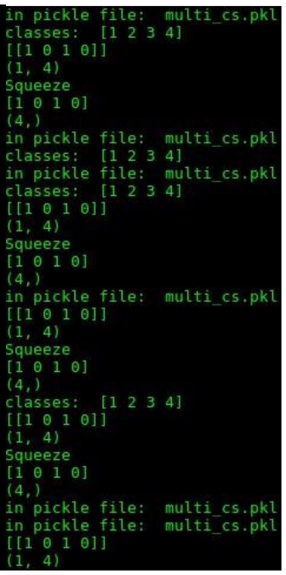
```
in pickle file:  multi_all.pkl
classes:  [1 2 3 4]
[[1 1 1 1]]
(1, 4)
Squeeze
[1 1 1 1]
(4,)
in pickle file:  multi_all.pkl
classes:  [1 2 3 4]
[[1 1 1 1]]
(1, 4)
Squeeze
[1 1 1 1]
(4,)
in pickle file:  multi_all.pkl
classes:  [1 2 3 4]
[[1 1 1 1]]
(1, 4)
Squeeze
[1 1 1 1]
(4,)
in pickle file:  multi_all.pkl
classes:  [1 2 3 4]
[[1 1 1 1]]
(1, 4)
Squeeze
[1 1 1 1]
(4,)
in pickle file:  multi_all.pkl
classes:  [1 2 3 4]
[[1 1 1 1]]
(1, 4)
```



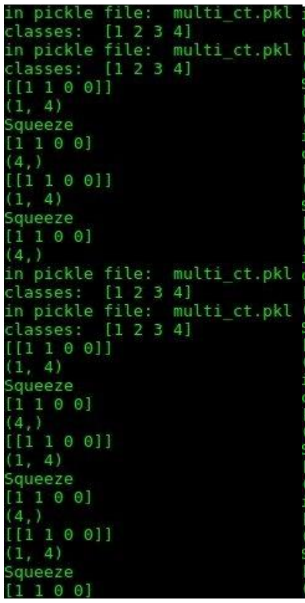
```
in pickle file:  multi_cr.pkl
classes:  [1 2 3 4]
[[1 0 0 1]]
(1, 4)
Squeeze
[1 0 0 1]
(4,)
in pickle file:  multi_cr.pkl
classes:  [1 2 3 4]
[[1 0 0 1]]
(1, 4)
Squeeze
[1 0 0 1]
(4,)
[[1 0 0 1]]
(1, 4)
Squeeze
[1 0 0 1]
(4,)
in pickle file:  multi_cr.pkl
classes:  [1 2 3 4]
in pickle file:  multi_cr.pkl
classes:  [1 2 3 4]
[[1 0 0 1]]
(1, 4)
Squeeze
[1 0 0 1]
(4,)
in pickle file:  multi_cr.pkl
[[1 0 0 1]]
(1, 4)
```



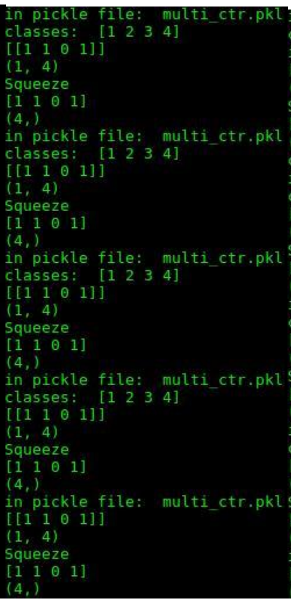
```
in pickle file:  multi_crs.pkl
classes:  [1 2 3 4]
in pickle file:  multi_crs.pkl
classes:  [1 2 3 4]
[[1 0 1 1]]
(1, 4)
Squeeze
[1 0 1 1]
(4,)
[[1 0 1 1]]
(1, 4)
Squeeze
[1 0 1 1]
(4,)
[[1 0 1 1]]
(1, 4)
Squeeze
in pickle file:  multi_crs.pkl
[1 0 1 1]
(4,)
classes:  [1 2 3 4]
[[1 0 1 1]]
(1, 4)
Squeeze
[1 0 1 1]
(4,)
in pickle file:  multi_crs.pkl
classes:  [1 2 3 4]
in pickle file:  multi_crs.pkl
classes:  [1 2 3 4]
in pickle file:  multi_crs.pkl
in pickle file:  multi_crs.pkl
[[1 0 1 1]]
(1, 4)
```



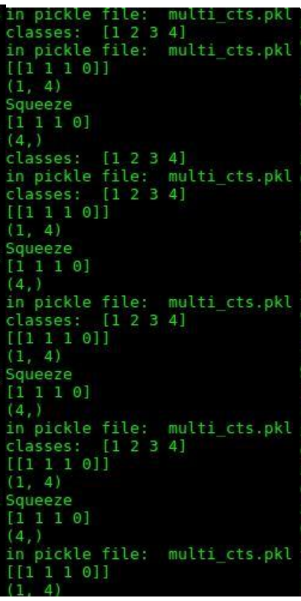
```
in pickle file:  multi_cs.pkl
classes:  [1 2 3 4]
[[1 0 1 0]]
(1, 4)
Squeeze
[1 0 1 0]
(4,)
in pickle file:  multi_cs.pkl
classes:  [1 2 3 4]
in pickle file:  multi_cs.pkl
classes:  [1 2 3 4]
[[1 0 1 0]]
(1, 4)
Squeeze
[1 0 1 0]
(4,)
in pickle file:  multi_cs.pkl
[[1 0 1 0]]
(1, 4)
Squeeze
[1 0 1 0]
(4,)
classes:  [1 2 3 4]
[[1 0 1 0]]
(1, 4)
Squeeze
[1 0 1 0]
(4,)
in pickle file:  multi_cs.pkl
in pickle file:  multi_cs.pkl
[[1 0 1 0]]
(1, 4)
```



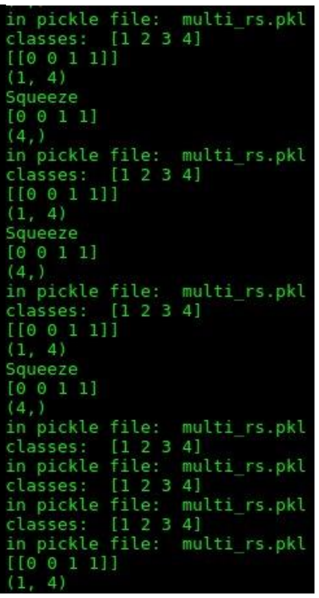
```
in pickle file:  multi_ct.pkl
classes:  [1 2 3 4]
in pickle file:  multi_ct.pkl
classes:  [1 2 3 4]
[[1 1 0 0]]
(1, 4)
Squeeze
[1 1 0 0]
(4,)
[[1 1 0 0]]
(1, 4)
Squeeze
[1 1 0 0]
(4,)
in pickle file:  multi_ct.pkl
classes:  [1 2 3 4]
in pickle file:  multi_ct.pkl
classes:  [1 2 3 4]
[[1 1 0 0]]
(1, 4)
Squeeze
[1 1 0 0]
(4,)
[[1 1 0 0]]
(1, 4)
Squeeze
[1 1 0 0]
(4,)
[[1 1 0 0]]
(1, 4)
Squeeze
[1 1 0 0]
```



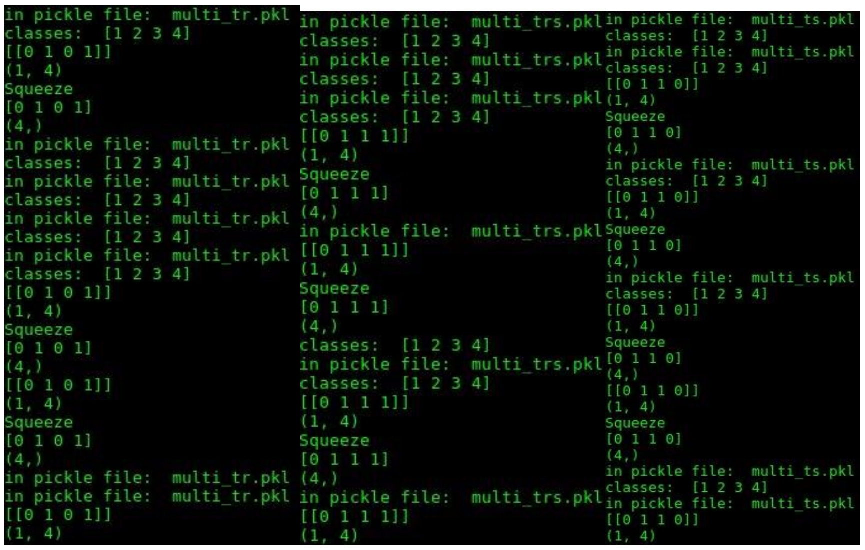
```
in pickle file:  multi_ctr.pkl
classes:  [1 2 3 4]
[[1 1 0 1]]
(1, 4)
Squeeze
[1 1 0 1]
(4,)
in pickle file:  multi_ctr.pkl
classes:  [1 2 3 4]
[[1 1 0 1]]
(1, 4)
Squeeze
[1 1 0 1]
(4,)
in pickle file:  multi_ctr.pkl
classes:  [1 2 3 4]
[[1 1 0 1]]
(1, 4)
Squeeze
[1 1 0 1]
(4,)
in pickle file:  multi_ctr.pkl
classes:  [1 2 3 4]
[[1 1 0 1]]
(1, 4)
Squeeze
[1 1 0 1]
(4,)
in pickle file:  multi_ctr.pkl
[[1 1 0 1]]
(1, 4)
Squeeze
[1 1 0 1]
(4,)
```



```
in pickle file:  multi_cts.pkl
classes:  [1 2 3 4]
in pickle file:  multi_cts.pkl
[[1 1 1 0]]
(1, 4)
Squeeze
[1 1 1 0]
(4,)
classes:  [1 2 3 4]
in pickle file:  multi_cts.pkl
classes:  [1 2 3 4]
[[1 1 1 0]]
(1, 4)
Squeeze
[1 1 1 0]
(4,)
in pickle file:  multi_cts.pkl
classes:  [1 2 3 4]
[[1 1 1 0]]
(1, 4)
Squeeze
[1 1 1 0]
(4,)
in pickle file:  multi_cts.pkl
classes:  [1 2 3 4]
[[1 1 1 0]]
(1, 4)
Squeeze
[1 1 1 0]
(4,)
in pickle file:  multi_cts.pkl
[[1 1 1 0]]
(1, 4)
```



```
in pickle file:  multi_rs.pkl
classes:  [1 2 3 4]
[[0 0 1 1]]
(1, 4)
Squeeze
[0 0 1 1]
(4,)
in pickle file:  multi_rs.pkl
classes:  [1 2 3 4]
[[0 0 1 1]]
(1, 4)
Squeeze
[0 0 1 1]
(4,)
in pickle file:  multi_rs.pkl
classes:  [1 2 3 4]
[[0 0 1 1]]
(1, 4)
Squeeze
[0 0 1 1]
(4,)
in pickle file:  multi_rs.pkl
classes:  [1 2 3 4]
in pickle file:  multi_rs.pkl
classes:  [1 2 3 4]
in pickle file:  multi_rs.pkl
classes:  [1 2 3 4]
in pickle file:  multi_rs.pkl
[[0 0 1 1]]
(1, 4)
```

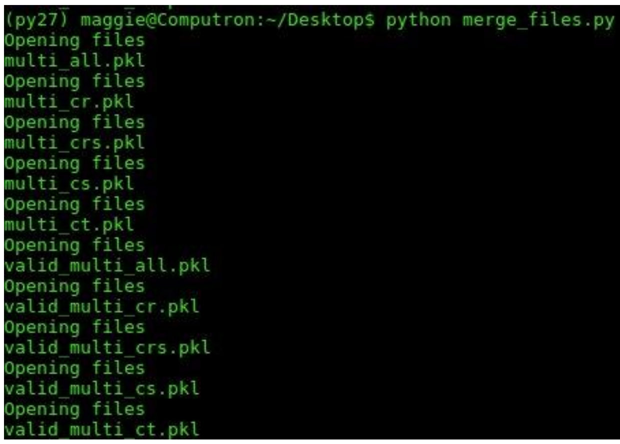


```
in pickle file:  multi_tr.pkl
classes:  [1 2 3 4]
[[0 1 0 1]]
(1, 4)
Squeeze
[0 1 0 1]
(4,)
in pickle file:  multi_tr.pkl
classes:  [1 2 3 4]
in pickle file:  multi_tr.pkl
classes:  [1 2 3 4]
in pickle file:  multi_tr.pkl
classes:  [1 2 3 4]
in pickle file:  multi_tr.pkl
classes:  [1 2 3 4]
[[0 1 0 1]]
(1, 4)
Squeeze
[0 1 0 1]
(4,)
[[0 1 0 1]]
(1, 4)
Squeeze
[0 1 0 1]
(4,)
in pickle file:  multi_tr.pkl
in pickle file:  multi_tr.pkl
[[0 1 0 1]]
(1, 4)
```

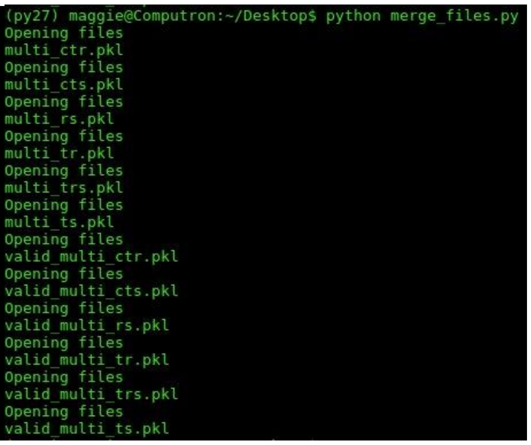
```
in pickle file:  multi_trs.pkl
classes:  [1 2 3 4]
in pickle file:  multi_trs.pkl
classes:  [1 2 3 4]
in pickle file:  multi_trs.pkl
classes:  [1 2 3 4]
[[0 1 1 1]]
(1, 4)
Squeeze
[0 1 1 1]
(4,)
in pickle file:  multi_trs.pkl
[[0 1 1 1]]
(1, 4)
Squeeze
[0 1 1 1]
(4,)
classes:  [1 2 3 4]
in pickle file:  multi_trs.pkl
classes:  [1 2 3 4]
[[0 1 1 1]]
(1, 4)
Squeeze
[0 1 1 1]
(4,)
in pickle file:  multi_trs.pkl
[[0 1 1 1]]
(1, 4)
```

```
in pickle file:  multi_ts.pkl
classes:  [1 2 3 4]
in pickle file:  multi_ts.pkl
classes:  [1 2 3 4]
[[0 1 1 0]]
(1, 4)
Squeeze
[0 1 1 0]
(4,)
in pickle file:  multi_ts.pkl
classes:  [1 2 3 4]
[[0 1 1 0]]
(1, 4)
Squeeze
[0 1 1 0]
(4,)
in pickle file:  multi_ts.pkl
classes:  [1 2 3 4]
[[0 1 1 0]]
(1, 4)
Squeeze
[0 1 1 0]
(4,)
[[0 1 1 0]]
(1, 4)
Squeeze
[0 1 1 0]
(4,)
in pickle file:  multi_ts.pkl
classes:  [1 2 3 4]
in pickle file:  multi_ts.pkl
[[0 1 1 0]]
(1, 4)
```

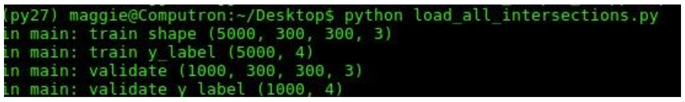
After these pickle files are created, the program merge_files.py merges all types of intersections into a single large list for training and validation data. It is very similar to the merge_files.py for single label output for multi-class classification. The difference is that there are a lot more files to load and merge. Since each intersection dataset consists of 2,000 images with a total of 22,000 images is a lot of repetitive data, half of each dataset is merged instead of the entire set with a total of 11,000 images. The validation data for the intersections consists of 200 images for each multi-label class, with a total of 2200, which is 20% of the training dataset. It dumps all the merged dataset, training dataset and validation dataset into a pickle file called data_shapes.pkl in order for load_all_intersections.py to create two large lists of the intersected data: the numpy image data list and its corresponding multi-label output. Similar to the single-label output, the multi-label output for the training dataset and the validation dataset are shuffled in order to randomize the data input for the convolutional neural network.



```
(py27) maggie@Computron:~/Desktop$ python merge_files.py
Opening files
multi_all.pkl
Opening files
multi_cr.pkl
Opening files
multi_crs.pkl
Opening files
multi_cs.pkl
Opening files
multi_ct.pkl
Opening files
valid_multi_all.pkl
Opening files
valid_multi_cr.pkl
Opening files
valid_multi_crs.pkl
Opening files
valid_multi_cs.pkl
Opening files
valid_multi_ct.pkl
```



```
(py27) maggie@Computron:~/Desktop$ python merge_files.py
Opening files
multi_ctr.pkl
Opening files
multi_cts.pkl
Opening files
multi_rs.pkl
Opening files
multi_tr.pkl
Opening files
multi_trs.pkl
Opening files
multi_ts.pkl
Opening files
valid_multi_ctr.pkl
Opening files
valid_multi_cts.pkl
Opening files
valid_multi_rs.pkl
Opening files
valid_multi_tr.pkl
Opening files
valid_multi_trs.pkl
Opening files
valid_multi_ts.pkl
```



```
(py27) maggie@Computron:~/Desktop$ python load_all_intersections.py
in main: train shape (5000, 300, 300, 3)
in main: train y_label (5000, 4)
in main: validate (1000, 300, 300, 3)
in main: validate y label (1000, 4)
```

Conclusion of the optimal Convolutional Neural Network Architecture

It is unclear which neural network architecture works better for generalization: 2 Dense layers or 3 Dense layers with the final Dense layer representing the output classes of the convolutional neural network. I am not sure if the white background from the cropped images should pertain to a class. Will that decrease the loss of the neural network? The lowest loss I can obtain from training the network is 0.3466. This has the architecture in the chart below:

adamax = Adamax(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0) #prev 0.002

Number of Filters	64		128			512	512	512	4
Layer Type	CONV_2D	MAX POOL	CONV_2D	MAXPOOL	Flatten()	Dense()	Dense()	Dense()	Dense()
Conv. Size	(6,6)	(6,6)	(6,6)	(6,6)		Dropout (0.35)	Dropout (0.35)	Dropout (0.35)	
Padding	valid		valid						
Activation	relu		relu			relu	relu	relu	softmax

Epoch 30/50

250/250 [==============================] - 233s - loss: 0.3466 - acc: 0.8096 - val_loss: 0.3560 - val_acc: 0.7869

Test loss: 4.95138476849

Test accuracy 0.673125

Total params: 4,041,156

Trainable params: 4,041,156

Layer(type)	Output Shape	Param #
conv2d_1(Conv2D)	(None, 295, 295, 64)	6976

max_pooling2d_1(MaxPooling2D)	(None, 49, 49, 64)	0
conv2d_2(Conv2D)	(None, 44, 44, 128)	295040
max_pooling2d_2(MaxPooling2D)	(None, 7, 7, 128)	0
flatten_1(Flatten)	(None, 6272)	0
dense_1(Dense)	(None, 512)	3211776
dropout_1(Dropout)	(None, 512)	0
dense_2(Dense)	(None, 512)	262656
dropout_2(Dropout)	(None, 512)	0
dense_3(Dense)	(None, 512)	262656
dropout_3(Dropout)	(None, 512)	0
dense_4(Dense)	(None, 4)	2052