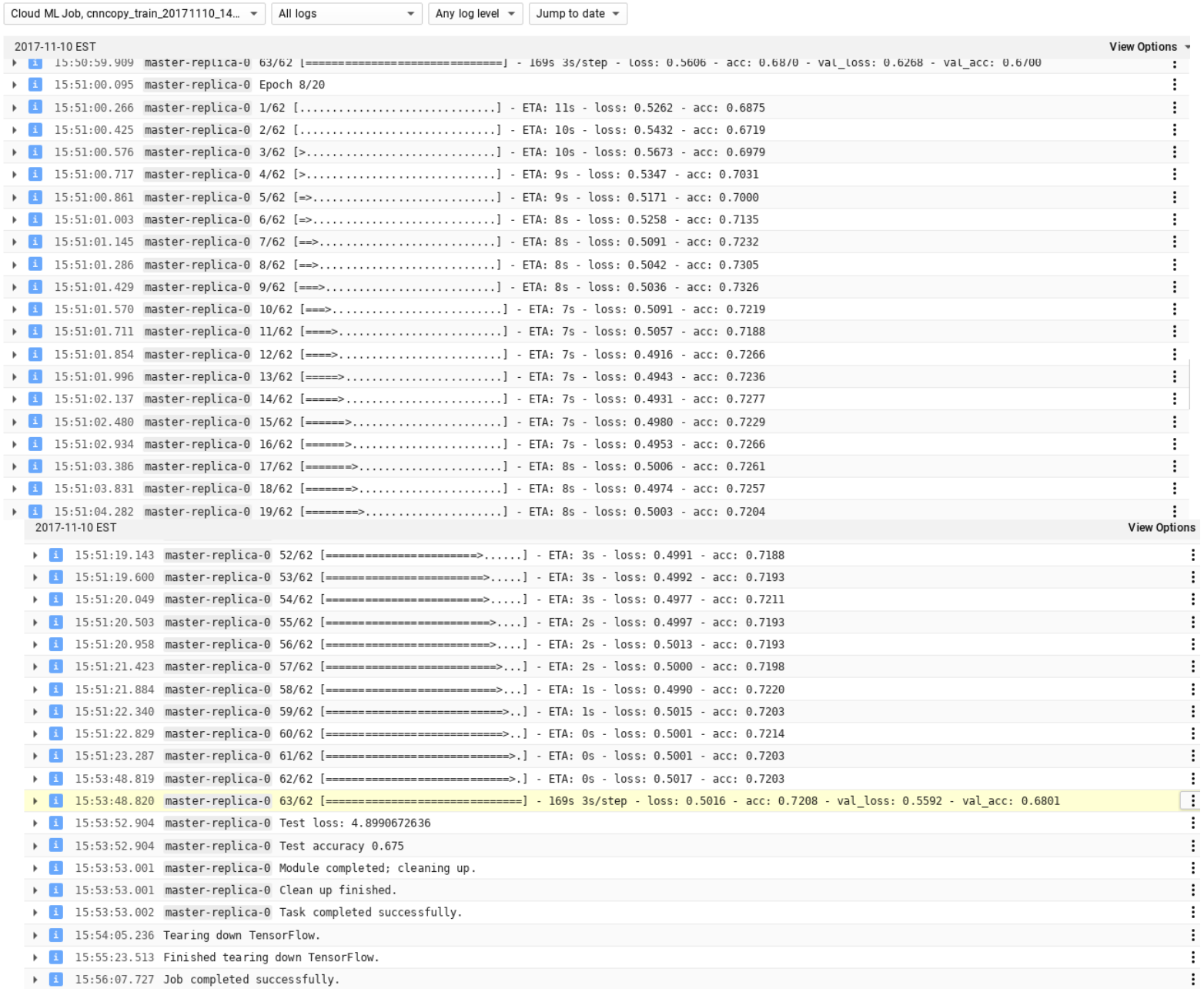
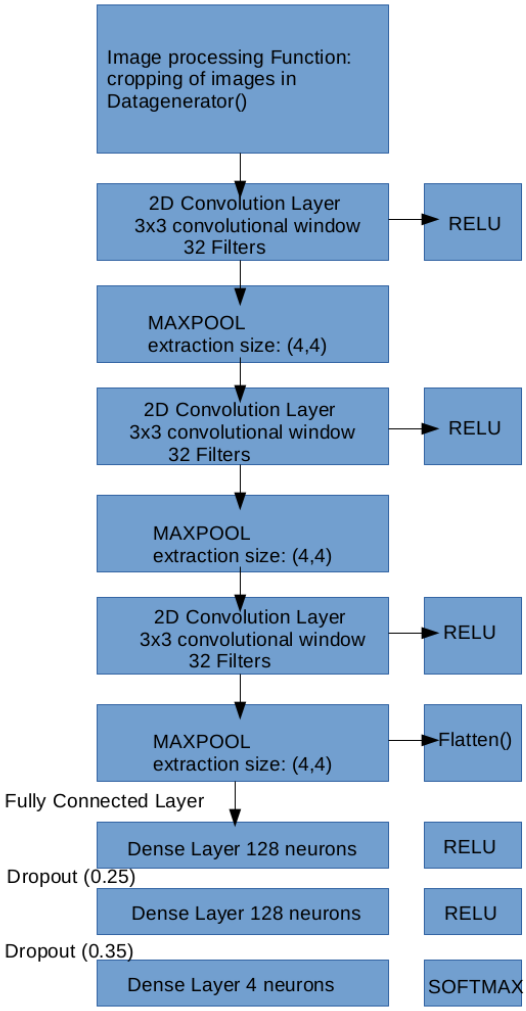
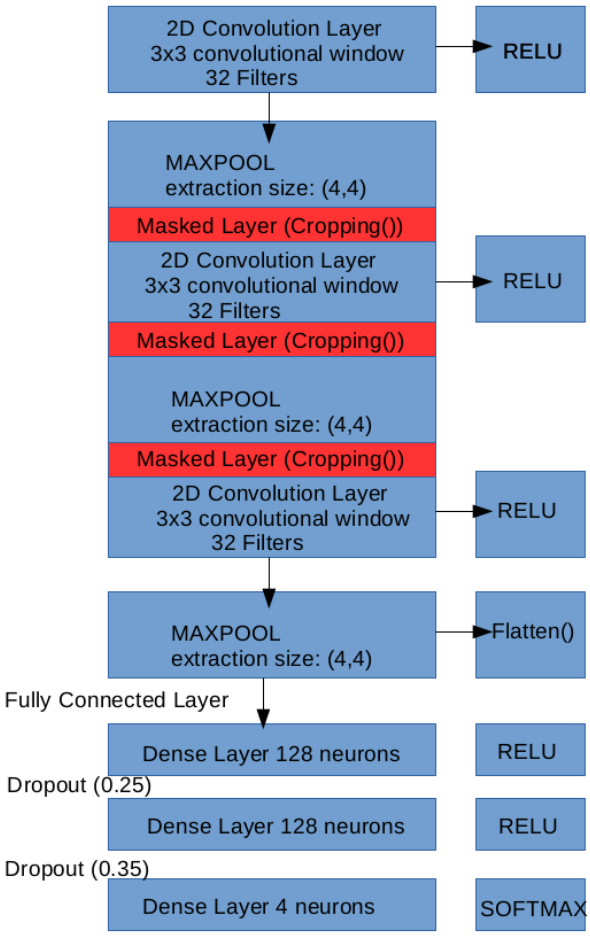
Since the neural network had a validation loss of 6.4028 with a validation accuracy of 0.58769, I changed the optimizer from rms_prop back to adam and reduced the 2D Maxpool size from (8,8) to (4,4). There was no problem loading the dataset of 1,000 images per each shape, with a total of 4,000 images. During the loading of the second half of the dataset, a callback function in keras called early stopping will stop the learning of the network to prevent overfitting. The convolutional neural network stopped at the 8/20 epoch in the loading of the second half of the dataset, which I think is too early for the neural network to stop. It might have stopped because the dataset is insufficient, I will increase the dataset by twice as much as the previous dataset after incorporating the cropping function in Keras. The validation loss is at 4.8 percent with a validation accuracy of 67.5%.

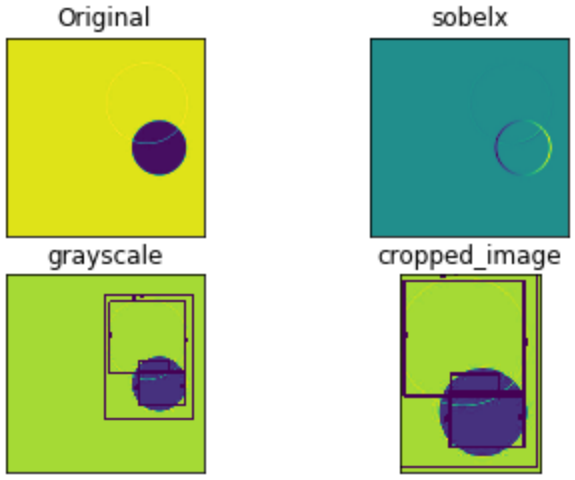


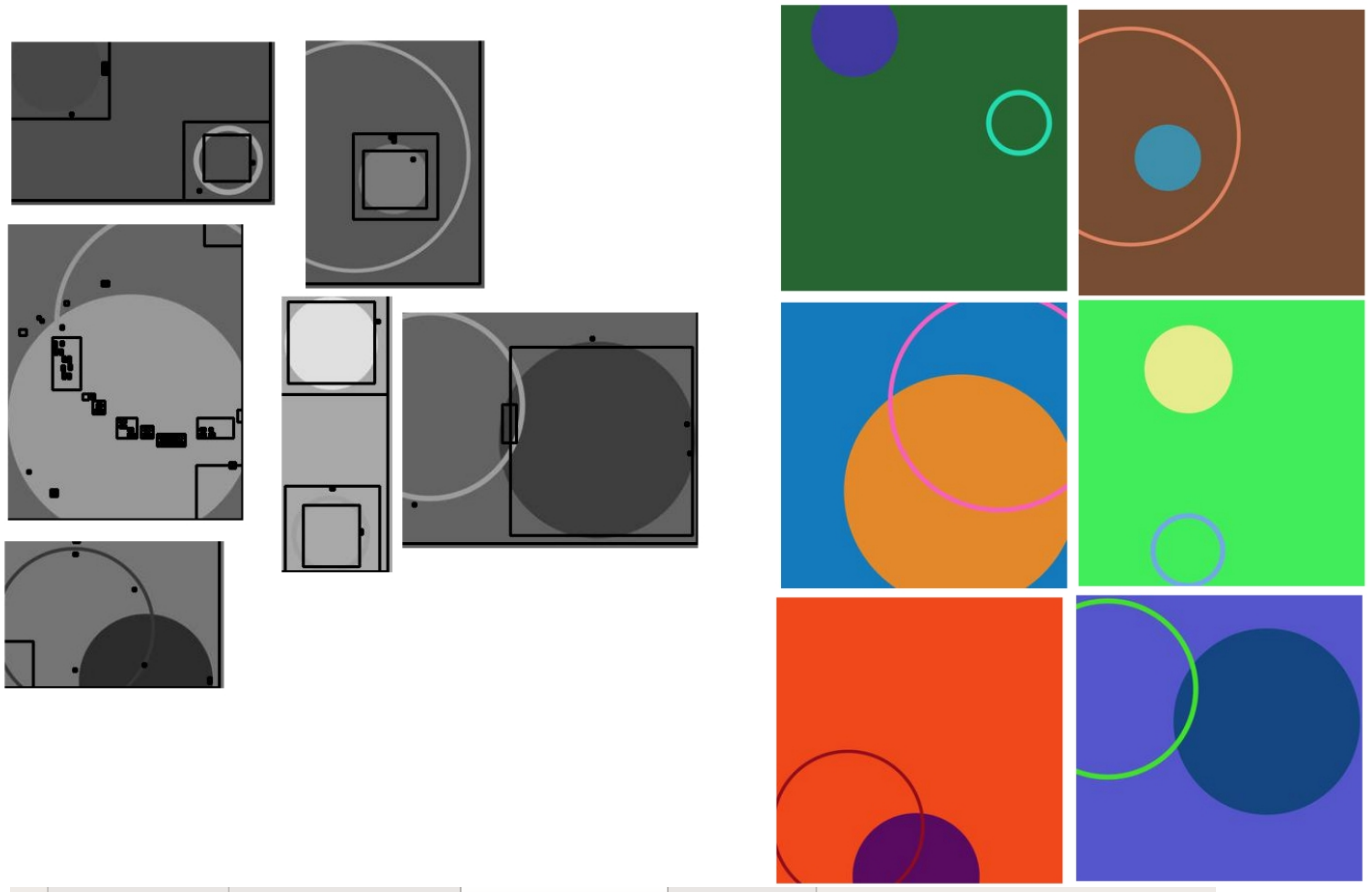
I made changes to the architecture of the network where I won't incorporate dropout during the beginning of the convolutional extractions. This is because it is important to retain as much important information about the shapes since I will be incorporating a cropping function either as a unified data generator as a callback parameter or ask a layer mask between the convolution layer. Dropout will occur after the convolutional layers are flattened and in between the 2-layer network. The network will still use stochastic learning since it's convenient and generates good learning rates. Below are the diagrams that explains the two modified designs of the neural network:





In order to make the convolutional neural network learn better, I created a cropping function that finds the boundaries of the shapes and crops accordingly. This cropping function will be applied as a parameter in the data_generator in the convolutional neural network loader file. This function is needed because the unnecessary background data that does not contain the pixels of the shapes will create more noise in the neural network during the convolutional 2D process of image extraction. The cropping function will make the neural network focus on the shapes in background instead of extracting colors of backgrounds with no important features in them besides colors. The function uses the cv2 module which is having problem with dependencies from python 2 to python 3 in the anaconda installations. The module works in python 3 but not in python 2 so adjustment to fix this issue will be needed. The procedure of the program is to use the sobel algorithm to detect the contours of the edges. The images have to be in grayscale beforehand, but the convolutional neural network extract grayscale images because the function will return the coordinates necessary for cropping. The function will create a boundary rectangle around the found contours and crop as necessary. Here are the black and white outputs of the cropped images:





cnncopy.py | pickletestcats.py | sobel_copy.py | draw.py

```python
"""
Created on Sun Nov 12 18:41:03 2017
Program finds contours and creates a rectangular boundary around it
This program will be implemented in the cnn loader file
@author: maggie
"""
import cv2
from matplotlib import pyplot as plt
import scipy.misc

"""code to get boundaries of contour shapes and crops the images based on the location of the edges"""
def get_edges(image_path):
    img = cv2.imread(image_path)
    #requires the image to be in grayscale
    grayscale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Gaussian to remove noise from
    img = cv2.GaussianBlur(grayscale,(3,3),0)
    # Edge Detection Filter: use sobel algorithm to detect edges
    sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
    sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)
    gradient_x = cv2.convertScaleAbs(sobelx)
    gradient_y = cv2.convertScaleAbs(sobely)
    #combine two sobel gradients
    gradient_t = cv2.addWeighted(gradient_x, 0.5, gradient_y, 0.5, 0)

    #Threshold filter
    #create threshold to separate edges from background
    #retval, thresh_gray = cv2.threshold(grayscale, thresh=127, maxval=255, type=cv2.THRESH_BINARY)
    im2,contours,hierarchy = cv2.findContours(gradient_t, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(gradient_t,contours,0,(0,0,255),2)

    #with each contour, find 2 sets of coordnates to draw bounding rectangle
    edge_list_x = []
    edge_list_y = []
    for c in contours:
        x, y, width, height = cv2.boundingRect(c)
        cv2.rectangle(grayscale, (x,y), (x+width, y+height), (0,255,0), 2) #draw green rectangle
        edge_list_x.append(x)
        edge_list_y.append(y)
        edge_list_x.append(x+width)
        edge_list_y.append(y+height)
```

```
43
44      smallest_x = min(edge_list_x)
45      smallest_y = min (edge_list_y)
46      largest_x = max(edge_list_x)
47      largest_y = max(edge_list_y)
48
49      offset = 5
50      #cv2.rectangle(grayscale, (smallest_x, smallest_y), (largest_x, largest_y), (0,255,0), 2)
51
52      # use this function to crop in keras 2Dcropping (extra layer between CNN layers)
53      crop_image = grayscale [smallest_y + offset:largest_y + offset,
54                              smallest_x + offset:largest_x + offset]
55
56      plt.subplot(2,2,1),plt.imshow(img)
57      plt.title('Original'), plt.xticks([]), plt.yticks([])
58      plt.subplot(2,2,2),plt.imshow(sobelx)
59      plt.title('sobelx'), plt.xticks([]), plt.yticks([])
60      plt.subplot(2,2,3),plt.imshow(grayscale)
61      plt.title('grayscale'), plt.xticks([]), plt.yticks([])
62      plt.subplot(2,2,4),plt.imshow(crop_image)
63      plt.title('cropped_image'), plt.xticks([]), plt.yticks([])
64
65      return crop_image
66
67  if __name__ == '__main__':
68      crop = get_edges("test_images/circle101.jpg")
69      scipy.misc.imsave("circle7.jpg", crop)
```

cnn_copy_sobel.py ✖ | sobel_copy.py ✖

```
55      # a decrease in accuracy  @ adeshpande3.github.io''
56      classifier.add(Conv2D(32, (3, 3), input_shape = (300, 300, 3), activation = 'relu'))
57
58      # Step 2:
59      # Max Pooling downsamples the number pixels per neuron and create a max
60      # number that describes those features in a pool_size of 2 by 2
61      # change pool size from (2,2) to (8,8) to (4,4)
62      classifier.add(MaxPooling2D(pool_size = (4, 4)))
63
64      # Adding a second convolutional layer, which is the same as the first one
65      classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
66      # change pool size from (2,2) to (8,8)
67      classifier.add(MaxPooling2D(pool_size = (4, 4)))
68      # Dropout layers at the second convolutional layer before flattening
69      #classifier.add(Dropout(0.25)) #don't add dropout here to fix it? 11/11 Saturday
70
71      # Step 3: Flattening the convolutional layers for input into a fully
72      # connected layer
73      classifier.add(Flatten())
74
75      # Step 4:
76      # Fully connected: Dense function is used to add a fully connected
77      # 3 layer perceptron at the end
78      classifier.add(Dense(units = 128, activation = 'relu'))
79      # dropout at the first layer perceptron
80      classifier.add(Dropout(0.25))
81      # adding second hidden layer - remove if accuracy decreases or loss increases
82      classifier.add(Dense(units = 128, activation = 'relu'))
83      classifier.add(Dropout(0.35))
84      # softmax classifier as an activation from the last layer perceptron
85      # units represent number of output classes
86      # the output classes are triangle, rectangle, square, circle
87      classifier.add(Dense(units = 4, activation = 'softmax'))
88
89      # Compiling the CNN:
90      # check if optimizer adam is good, categorical_crossentropy is for
91      # multi-class network, multilabel with intersection needs binary_crossentropy
92      # and sigmoid activations
93      # change from adam to rmsprop
94      classifier.compile(optimizer = 'adam',
95                         loss = 'categorical_crossentropy',
96                         metrics = ['accuracy'])
```

```
    # Part 2:
    # Feeding CNN the input images and fitting the CNN
    # CNN uses data augmentation configuration to prevent overfitting
    # datagen augmentation is for training data input
    datagen = ImageDataGenerator(featurewise_center = True,
                                 featurewise_std_normalization = True,
                                 rescale = 1./255,
                                 shear_range = 0.2,
                                 zoom_range = 0.2,
                                 horizontal_flip = True)
                                 preprocessing_function = get_edges(train_shape_dataset))

    # augmentation configuration for rescaling images used for validation
    validate_datagen = ImageDataGenerator(rescale = 1./255)

    # the test set data augmentation only rescales the images
    # is this enough to test the network correctly? if you want a more manual
    # representation of fitting the input data use for loop
    validate_datagen.fit(validate_shape_dataset)
    validate_generator = datagen.flow(validate_shape_dataset,
                                      validate_y_dataset,
                                      batch_size = 32)

    # the code below fits the training data that is loaded by pickle file
    # to prevent memory error, 1/2 of the number of data inputs are feed first
    # an epoch define the input being run once from
    # the architecture of the cnn is:
    # 2DConv -> ReLU -> MaxPool -> 2DConv -> ReLU -> MaxPool -> Flatten() ->
    # Fully connected 2-layer neural network
    # 128 neurons for the first layer -> ReLU -> 128 for hidden layer -> ReLU
    # -> 3 neurons for output layer -> softmax

    # compute quantities required for featurewise normalization
    datagen.fit(train_shape_dataset)
    #early_stopping = EarlyStopping(monitor = 'val_loss', patience = 2)
    # fits the model on batches with real-time data augmentation
    train_generator = datagen.flow(train_shape_dataset,
                                   train_y_dataset,
                                   batch_size = 32)
    classifier.fit_generator(train_generator, #train generator
                             steps_per_epoch = len(train_shape_dataset) / 32,
                             epochs = 20,
                             validation_data =  validate_generator,
```

The ImageDataGenerator will be normalized via featurewise_center and featurewise_std_normalization in order to make the convolutional neural network gather features that have equal significance. Sometimes the shapes blend with the background environment, so normalizing the data will make the unfilled and filled shapes the same. In this code, the cropping function of data shapes will be incorporated as a function in data generator of keras. This means that the images will be cropped before being fed into the network. To maintain the original images of the network, I can use the cropping as a layer mask in between the 2D convolution architecture by using Kera's 2D cropping:

classifier.add(Conv2D(32, (3, 3), input_shape = (300, 300, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (4, 4)))
# Adding a second convolutional layer, which is the same as the first one
classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
# change pool size from (2,2) to (8,8)
classifier.add(MaxPooling2D(pool_size = (4, 4)))
# Dropout layers at the second convolutional layer before flattening
keras.layers.Cropping2D(cropping=get_edges(train_shape_dataset), data_format=None)
# Step 3: Flattening the convolutional layers for input into a fully
# connected layer
classifier.add(Flatten())