PREDICTING HAND DRAWN SKETCHES USING GOOGLE'S QUICKDRAW! DATASET IN REAL TIME

Vansh Bajaj , Poshan Bastola and Dr. Roman Tankelevich
Department of Computer Science
California State University , Long Beach , USA
vansh.bajaj@student.csulb.edu,poshan.bastola@student.csulb.edu,roman.tankelvich@csulb.edu

ABSTRACT

Image recognition has been at the forefront of development of Neural networks . Advances in Deep Learning has enabled previously incomprehensible tasks such as image classification , speech recognition possible in real-time . Current advances in Neural Networks such as RNN's and GAN's have even enabled conditional and unconditional generation of hand-drawn sketches in real-time. However, these advanced techniques are still being developed , implemented and optimized so that they can be implemented in the real world.

Hence, In this paper we propose – 1.) A CNN based model that is capable in receiving input through OpenCV and classifying the received input into classes in real-time. 2.) Performance evaluation of the model in terms of loss function and model accuracy. 3.) Model limitations of our currently developed model it's future scope.

KEYWORDS – Convolutional Neural Network(CNN) , Recurrent Neural Network (RNN), QuickDraw, Convolution Layer , Maxpooling, Flatten layer , Dense Layer , Dropout Layer, SoftMax Layer , multinomial logistic regression , ADAM optimizer , Cross Entropy Loss.

1 INTRODUCTION

Human Beings have inherent ability to express themselves and communicate with others by drawing images and sketches . The sketches are expressed as a short sequence of strokes . There have been major advancements in this field of generative modelling of images using neural networks . Various models have been proposed for this such as Generative Adversarial Network (GANs) , Variational Inference and Autoregressive models . However , most of these models proposed have been targeted towards generating low resolution pixel images of objects . Our motivation for this project is Google's Sketch-RNN[1]. The developers at Google Brain gave a very fundamental idea which we have incorporated in our model.

[1] A Neural Representation of Sketch Drawings - David Ha, Douglas Eck. (Ha & Eck, 2018) (Image Recognition, 2017)

That idea is - Each sketch should be treated as a set of pen stroke actions. The single-event pen stroke actions should be treated as multi-state events. In other words , each sketch is treated as a vector of five elements namely , (Δx,Δy, p1,p2, p3). The first two elements are the offset distance in the x and y directions of the pen from the previous point. The last 3 elements represents a binary one-hot vector of 3 possible states. The first pen state,p1, indicates that the pen is currently touching the paper, and that a line will be drawn connecting the next point with the current point. The second pen state, p2, indicates that the pen will be lifted from the paper after the current point, and that no line will be drawn next. The final pen state, p3, indicates that the drawing has ended, and subsequent points, including the current point, will not be rendered.

Architecturally , Sketch-RNN is described as a Sequence-to-Sequence Variational Autoencoder (VAE). Encoder is a bidirectional RNN that takes in a sketch and outputs a latent vector of a specific size. Decoder is an autoregressive RNN that generates sketches based on a given latent vector. Specifically , Decoder only model can be trained to suggest ways to complete incomplete sketches and Decoder-Encoder model can be trained for conditional and unconditional generation of images.

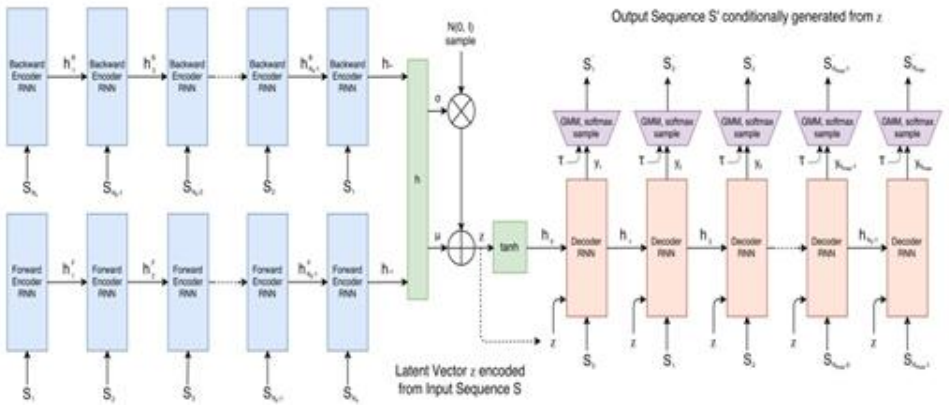

Figure 1 - Architecture of Google's Sketch-RNN

2 RELATED WORK

There has been a lot of work related to algorithms and models that mimics a human beings creative process. In 2013, Paul The Robot was proposed . The underlying algorithm was responsible for controlling the mechanical arm that sketches lines on the canvas with a programmable creative style to mimic a digitized portrait of a person . Reinforcement Learning based approaches have also been developed that to discover a set of paint brush strokes that can best represent a given input photograph . Neural Network-based approaches have been developed for generative models of images, although the majority of neural network-related research on image generation deal with pixel images . Earlier work on vector image generation made use of Hidden Markov Models to synthesize lines and curves required to generate a hand drawn sketch . Most recent work has been done on generating handwriting with Recurrent Neural Networks to generate continuous data points .

3 DATA SET

One of the main factors that limiting research in the space of generative vector drawings was the lack of publicly available datasets . Previous available datasets such as Sketch datasets contain 20k images and was used to explore feature extraction techniques . The Sketchy dataset contained 70k vector sketches along with pixels images for several classes . ShadowDraw was a publicly available dataset 30k images combined with extracted vectorized features .The dataset we use comes from an online game from the same name QuickDraw![2] . The goal of the game is to draw an object from a particular class in less than 20 seconds . Each class of the Quickdraw! is a dataset of 70k training samples , 2.5k Validation samples and 2.5k Test samples. However , due to limitations in computational resources we have trained 15 classes only.

4 CNN

Each component of the model is explained in the following sections.

Neural Networks transform the received input through a series of hidden layers each of which are made up of a set of neurons, where each neuron is fully connected to all the neurons in the previous layer. Neurons in the single layer are completely independent and do not share any connections. The last fully connected layer is known as the output layer and represents the class scores . However , Neural Networks don't scale well to full images. For example , consider the CIFAR-10 dataset , which is a collection of images sized as 32*32*3. Hence a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights. This amount of weights is still manageable. If the image size is increased substantially to let's say , 200*200*3 then the total weight of the neurons comes out to be 200*200*3 = 120,000 weights. Also, were going to need several of these neurons so that the parameters add up quickly. This is clearly wasteful, and a large number of parameters would lead to overfitting.

The model we are using is a convolutional neural network[3] .Convolutional Neural networks constrain the architecture of input images in a sensible way . CNN arranges neurons in 3 layers : width, height and depth . Every layer has a simple API that is transformation of an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters.

[2] (Lab, 2018)

[3] (Jaswal & Sowmya.V, June 2014)

5 LAYERS

Each of these layers and their functions are explained below -

5.1 CONVOLUTION LAYER

The Convolution Layer[4] is the core building block of a CNN that does most of the computational heavy lifting. We are using 3 Convolutional Layers in our model . The Convolution layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height) but extends through the full depth of the input volume. In our model , The first convolution layer accepts an input size of 24*24*128 . The second convolutional layer accepts an input size 9*9*64 and the third convolutional layer accepts an input size of 3*3*32. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

In our model, we have 3 convolution layers. The first layer consists of 128 nodes with filter size 5. The second layer consists of 64 layers with filter size 4. The third convolution layer consist of 32 nodes with filter size 3.Size of input image is 28*28. The activation function is relu for all convolution layers.

```
model = Sequential()image_x,image_y,1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2
model.add(Conv2D(128, (5, 5), input_shape=(, 2), padding='same'))
model.add(Conv2D(64, (4, 4), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
```

[4] (Advanced Convolutional Neural Networks, 2017)

5.2 MAXPOOL LAYER

Max Pooling is a sample-based discretization process. The objective is to down sample an input representation by reducing its dimensionality and allowing for assumptions to be made about features in the sub-regions.

Maxpooling is done by applying a max filter or a maxpool to non-overlapping subregions of initial representation.

For each region in the filter , we take the max of that region and create a new output matrix where each element is the max of the region in the original input.

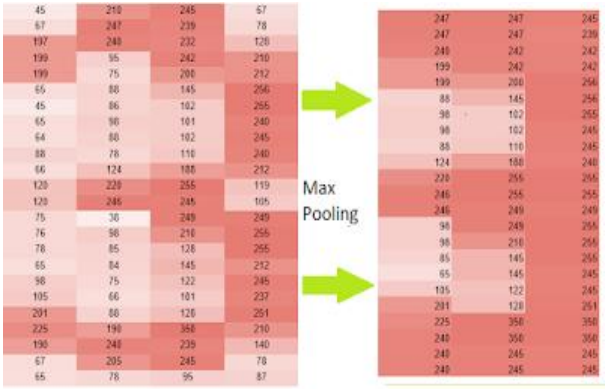

Figure 2- Dimensionality reduction using Maxpool

Maxpooling with a reference image of 3 with stride 1 . Stride is the number of steps that the pooling matrix will jump .

In our model ,there Maxpooling layers. All the pooling layers consists of 2 pool size , stride length is 2 and padding size as 2.

```
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2,2), padding='same'))
model.add(Conv2D(128, (5, 5), input_shape=(, 2), padding='same'))
model.add(Conv2D(64, (4, 4), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
```

5.3 DENSE LAYER

A dense layer is just a regular layer of neurons. Each neuron receives input from all the neurons in the previous layer, thus densely connected. The layer has a weight matrix **W,** a bias vector **b,** and the activations of previous layer **a.** The following is the docstring of class Dense from the keras documentation:

output = activation(dot(input, kernel) + bias)

where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer.

Our model consists of 3 dense layers. The first dense layer has 512 fully connected nodes. The second layer consists of 128 fully connected nodes and the third layer consists of 15 classes for output. The first two dense layers use Relu as their activation function and last dense layer uses SoftMax activation function.

```
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.6))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.6))
model.add(Dense(classes, activation='softmax'))
```

5.4 DROPOUT LAYER

Dropout is a regularization technique that is used to reduce overfitting. During training , individual nodes and their incoming and outgoing nodes are either discarded or kept each with the probability of 1-p and p respectively . As a result, a reduced network is generated . Once reduced network is trained . The removed nodes and their weights are reinserted after the training is done.

In our model, we have used a dropout of 0.6 to avoid overfitting. There are two dropout layers.

5.5 FLATTEN LAYER

After a pooled feature map is generated the pooled feature map is ‘flattened’. Flattening is the process of converting all the resultant 2 dimensional arrays into a single long continuous linear vector. Flattening is needed to make the use of fully connected layers after processing from convolution layers . Fully connected layers don't have a local limitation like convolutional layers (which only observe some local part of an image by using convolutional filters).Each feature

map channel in the output of a CNN layer is a "flattened" 2D array created by adding the results of multiple 2D kernels (one for each channel in the input layer).

5.6 SOFTMAX LAYER

SoftMax[5] layer is the last layer of a neural network just before the output layer. The SoftMax layer must have same number of nodes as the output layer. SoftMax assigns decimal probabilities to each class in a multi-class problem . The sum total of every decimal probabilities must be equal to one .

Mathematically SoftMax is defined as –

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

5.7 ReLU ACTIVATION FUNCTION

The Rectified Linear Unit[6] computes the function-

$$f(x) = max(x, 0)$$

A rectified linear unit has output 0 if the input is less than 0, and raw output otherwise. That is, if the input is greater than 0, the output is equal to the input. In other words, the activation is simply thresholded at zero . It is computationally inexpensive as compared to sigmoid and tanh functions and greatly increases the convergence of stochastic gradient descent as compared to tanh and sigmoid functions.

[5] (Singh, 2018)
[6] (Stanford,2018)

6 METHODOLOGY

The methodology for building and testing the model is explained below -

1.) The QuickDraw dataset was downloaded in numpy.bitmap format.
2.) CNN model is then built with the following layers –

- 3 * Convolution Layers
- 3 * MaxPool Layers
- 3 * Dense Layers
- 2 * Dropout Layers
- 1 * Flatten Layer
- 1 * SoftMax Layer

3.) Cross Entropy loss function and ADAM optimizer is used for Training.
4.) OpenCV is incorporated in the model to provide user input.
5.) Webcam is used to register the input to the CNN.
6.) CNN classifies hand drawn input based on the 15 trained classes of objects.

7 TRAINING

To train our model we made use of multinomial logistic regression that is SoftMax regression. SoftMax applies non-linearity to the output of the network and calculates the cross entropy between normalized predictions and the label index. It is technically known as Cross Entropy Loss Function. We used ADAM as the optimizer. The model was trained for 16 epochs.

The training time was 17 seconds each epoch except second epoch which took 67 seconds. The training time was quite fast because of use of Nvidia GTX 1060 GPU with 8GB of RAM. However , upon increasing the number of classes the training time went up exponentially with reduction in accuracy to as low as 46% .We wanted to test the limitations of our CNN model and our latest hardware, so we tried to train the entire QuickDraw! Dataset which could not be done as our hardware was not up to the mark. Applying more convolution layers or flattening layers didn’t seem to help either as a lot of features were lost which also resulted in a drop of accuracy. Hence to maximize accuracy we had to sacrifice the number of classes that were trained . The first epoch had loss with 0.78 and accuracy of 79% which went up to 95% by the last epoch whereas the loss was 0.16.

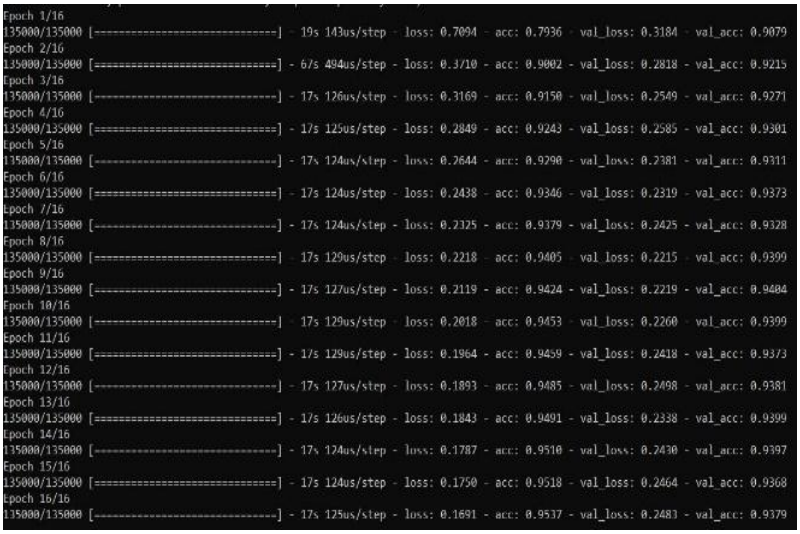

Figure 3- Training the model

7.1 CROSS - ENTROPY LOSS FUNCTION

Cross Entropy loss [7]function is used to measure the performance of a classification model whose output value is a probability between 0 and 1 . The Cross-Entropy Loss formula is given as -

$$c = \sum_{0}^{n} p_i \log(1/q_i)$$

Minimizing this function will enable our model to move towards a desired distribution.

7.2 ADAM OPTIMIZER

ADAM[8] optimization algorithm is seen as an extension to stochastic gradient descent . The name is derived from adaptive moment estimation .

ADAM is known to combine two extensions of stochastic gradient algorithm –

AdaGrad – AdaGrad maintains a different learning rate for each of the parameters . Known to improve performance with sparse gradients.

RMSProp – Similar to AdaGrad in the sense that it also assigns per – parameter learning rates however these rates are adapted based on the average of recent magnitudes of gradients for the weights.

---

[7] (Singh, 2018)
[8] (Browniee, 2017)

ADAM realizes both the benefits of these extensions.

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
summary = model.fit(train_x, train_y, validation_data=(test_x, test_y), epochs=16, batch_size=64,
          callbacks=[TensorBoard(log_dir="QuickDraw")])
```

Our model uses cross entropy loss function and Adam optimizer. We earlier used Stochastic Gradient Descent, but the accuracy of the model was lower than that when using ADAM.

8 HARDWARE SPECIFICATION AND ENVIRONMENT:

CPU: Intel(R) Core i7 - 8th Gen @ 2.20GHz
GPU: NVidia GeForce GTX 1060
CPU RAM: 16GB
Dedicated GPU RAM: 6 GB

8.1 ENVIRONMENT:

Microsoft Windows 10 64-bit
TensorFlow APIr1.12
Python 3.6
Anaconda 2.3.0
Py-opencv 3.4.2

9 TESTING AND EVALUATION

Testing and evaluation were done using OpenCV API to draw doodles on the screen. The model predicted the hand drawn doodles with good accuracy but there were some instances where the model was off by a large factor for some hand drawn doodles. The problem is also the OpenCV API which can pick up unwanted point during drawing that can add noise to what we are drawing.
However, the results were pretty good. It correctly classified almost most of the hand drawn doodles into one of the 15 classes.

Captures 1: Testing the model.

10 RESULTS

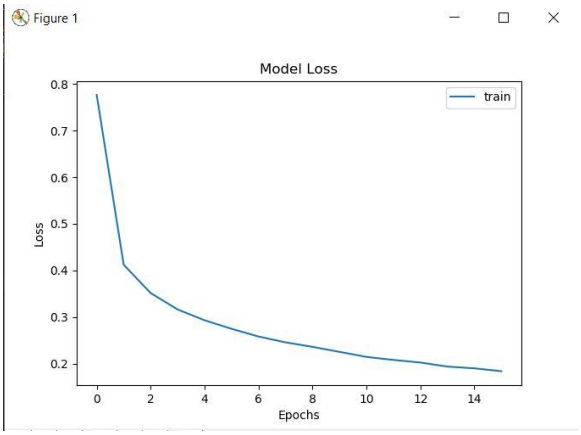

Figure 4 - Graph depicting Model Loss values

Loss at the start of the training period was fairly high at 0.78 . Loss was fairly constant after 11 epochs with slight reduction till it reached 0.20 at the 16th epoch.

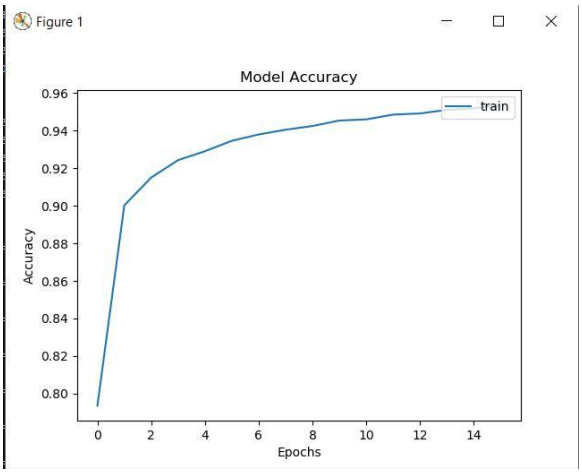

Figure 5 - Graph depicting Model Accuracy.

Model accuracy was 75% at epoch 0 . After 12 epochs the accuracy was 94% which stayed constant till 16th epoch.

11 MODEL LIMITATIONS

Our CNN based model was able to achieve high accuracy and low loss although , our model is still somewhat unpredictable with complex images. Due to memory limitations
we were only able to train 15 classes of the 345-class dataset . The model would have had much better performance with complex images if it had been trained on more classes. Also, our model is limited by its traditional architecture . Newer RNN's are able to handle arbitrary inputs and outputs along with other advantages such as maintenance of an internal state to capture information.

12 CONCLUSION

Even though our model is a traditional CNN that is trained on far less number of classes than Google's Sketch-RNN. Although we were limited by time and computational resources, we have managed to show the application of the basic principle that was proposed by Google Brain developers that a sketch should be treated as a vector with five elements. This simple idea has fueled the research for both conditional and unconditional generation of images . Sketch-RNN has the potential to enable many creative applications . Even a decoder only model can assist an artist's creative process by suggesting ways to finish an incomplete sketch. It can be used to generate a large number of similar but unique designs for textile and wallpaper prints. Sketch-RNN can be combined with supervised cross-domain pixel image generation to generate photo-realistic images . [9]

13 BIBLIOGRAPHY

Advanced Convolutional Neural Networks. (2017). Retrieved from tensorflow.org: https://www.tensorflow.org/tutorials/images/deep_cnn

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., & Bengio, S. (2016). Generating Sentences from a Continuous Space. *SIGNLL Conference on Computational Natural Language Learning (CONLL)*, 1-3.

Browniee, J. (2017, July 3). *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. Retrieved from machinelearningmastery.com: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

Ha, D., & Eck, D. (2018). A Neural Representation of Sketch Drawings. *ICLR 2018* (pp. 1-9;). ICLR Conference.

Image Recognition.(2017). Retrieved from Tensorflow.org: https://www.tensorflow.org/tutorials/images/image_recognition

[9] (Ha & Eck, 2018)

Jaswal, D., & Sowmya.V, K. (June 2014). Image Classification Using Convolutional NeuralNetworks. *International Journal of Scientific & Engineering Research*, 1-8.

Lab, G. C. (2018). *The Quick, Draw! Dataset*. Retrieved from github.com: https://github.com/googlecreativelab/quickdraw-dataset

Ruder, S. (2016, January 19). *An overview of gradient descent optimization algorithms*. Retrieved from ruder.io: http://ruder.io/optimizing-gradient-descent/index.html#adam

Singh, V. (2018, Apr 3). *Understanding Entropy, Cross-Entropy and Cross-Entropy Loss*. Retrieved from Medium.com: https://medium.com/@vijendra1125/understanding-entropy-cross-entropy-and-softmax-3b79d9b23c8a

Stanford. (2018, Spring). *Convolution Neural Networks for Visual Recognition*. Retrieved from Github.com: http://cs231n.github.io/convolutional-networks/

14 APPENDIX

14.1 SOURCE CODE

```
def Model(image_x, image_y):
    classes = 15
    model = Sequential()image_x,image_y,1), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2,2), padding='same'))
    model.add(Conv2D(128, (5, 5), input_shape=(, 2), padding='same'))
    model.add(Conv2D(64, (4, 4), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.6))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.6))
    model.add(Dense(classes, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    filepath = "WeightsOfQuickDraw.h5"
    checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
    callbacks_list = [checkpoint]
    return model, callbacks_list
```

Captures 2 - Model Architecture

```
def main():
    with open("features", "rb") as f:
        features = np.array(pickle.load(f))
    with open("labels", "rb") as f:
        labels = np.array(pickle.load(f))
    features, labels = shuffle(features, labels)
    labels = np_utils.to_categorical(labels)
    train_x, test_x, train_y, test_y = train_test_split(features, labels, random_state=0,test_size=0.1)
    train_x = train_x.reshape(train_x.shape[0], 28, 28, 1)
    test_x = test_x.reshape(test_x.shape[0], 28, 28, 1)
    model, callbacks_list = Model(28, 28)
    print_summary(model)
    summary = model.fit(train_x, train_y, validation_data=(test_x, test_y), epochs=16, batch_size=64,
              callbacks=[TensorBoard(log_dir="QuickDraw")])
    #plot result acc
    plt.plot(summary.history['acc'])
    plt.title("Model Accuracy")
    plt.ylabel("Accuracy")
    plt.xlabel("Epochs")
    plt.legend(['train'], loc='upper right')
    plt.show()

    #plot loss result
    plt.plot(summary.history['loss'])
    plt.title("Model Loss")
    plt.ylabel("Loss")
    plt.xlabel("Epochs")
    plt.legend(['train'], loc='upper right')
    plt.show()
    model.save('QuickDraw.h5')
```

Captures 3 - Training and Testing

```
def main():
    Lower_blue = np.array([110, 50, 50])
    Upper_blue = np.array([130, 255, 255])
    blackboard = np.zeros((480, 640, 3), dtype=np.uint8)
    digit = np.zeros((200, 200, 3), dtype=np.uint8)
    emojis = get_emojis()
    cap = cv2.VideoCapture(0)
    pts = deque(maxlen=512)
    pred_class = 0

    while (cap.isOpened()):
        ret, img = cap.read()
        img = cv2.flip(img, 1)
        hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        kernel = np.ones((5, 5), np.uint8)
        mask = cv2.inRange(hsv, Lower_blue, Upper_blue)
        mask = cv2.erode(mask, kernel, iterations=4)

        cv2.imshow("mask", mask)
        cnts, heir = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2:]
        center = None

        if len(cnts) >= 1:
            cnt = max(cnts, key=cv2.contourArea)
            if cv2.contourArea(cnt) > 200:
                print("Counters > 200")
                ((x, y), radius) = cv2.minEnclosingCircle(cnt)
                cv2.circle(img, (int(x), int(y)), int(radius), (0, 255, 0), 2)
                cv2.circle(img, center, 5, (0, 0, 255), -1)
                M = cv2.moments(cnt)
                center = (int(M['m10'] / M['m00']), int(M['m01'] / M['m00']))
                pts.appendleft(center)
                for i in range(1, len(pts)):
                    if pts[i - 1] is None or pts[i] is None:
                        continue
                    cv2.line(blackboard, pts[i - 1], pts[i], (255, 255, 255), 7)
                    cv2.line(img, pts[i - 1], pts[i], (0, 0, 255), 2)
        elif len(cnts) == 0:
            if len(pts) != []:
                print("Counters < 200")
                blackboard_gray = cv2.cvtColor(blackboard, cv2.COLOR_BGR2GRAY)
                blur1 = cv2.medianBlur(blackboard_gray, 15)
                blur1 = cv2.GaussianBlur(blur1, (5, 5), 0)
                thresh1 = cv2.threshold(blur1, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
                blackboard_cnts = cv2.findContours(thresh1.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)[1]
                if len(blackboard_cnts) >= 1:
                    cnt = max(blackboard_cnts, key=cv2.contourArea)
                    print(cv2.contourArea(cnt))
                    if cv2.contourArea(cnt) > 2000:
                        print("GREATER THAN 2000")
                        x, y, w, h = cv2.boundingRect(cnt)
                        digit = blackboard_gray[y:y + h, x:x + w]
                        pred_probab, pred_class = predict_model(model, digit)
                        print(pred_class, pred_probab)

            pts = deque(maxlen=512)
            blackboard = np.zeros((480, 640, 3), dtype=np.uint8)
            img = overlay(img, emojis[pred_class], 400, 250, 100, 100)
        cv2.imshow("Frame", img)
```

Captures 4 - Using OpenCV to input sketches.