Selfie Filters Addition With Speech Recognition On Facial Keypoints Using Convolutional Neural Networks

Srishti Tomar
Akshit Goyal
Chad Bloxham
Mehul Kothari

Problem Description

- Facial KeyPoint detection is a challenging problem in the field of computer vision since the facial features vary greatly from one individual to another. In 2018, Snapchat released speech recognition lenses that animate when users speak simple English words.

- We have detected facial keypoints in real time and then attempted to add and change selfie filters using voice commands which is still under research.

Code Requirements

- Python
- OpenCv
- Keras Libraries
- Speech_Recognition & Py-Audio
- NLTK

Methodology

Data Description

- This dataset on Kaggle allows us to train a model to detect the facial keypoints given a facial image.
- It was provided by Dr. Yoshua Bengio of the University of Montreal.
- Each datapoint in the dataset contains space separated pixel values of the images in a sequential order and the last 30 values of the datapoint represent 15 pairs of coordinates of the key points on the face.
- We have trained a CNN model to solve this classic deep learning problem.

Data Pre-Processing

- Uniform aspect ratio: One of the first steps is to ensure that the images have the same size and aspect ratio.
- Image Scaling: Once we’ve ensured that all images are square (or have some predetermined aspect ratio), it’s time to scale each image appropriately.
- Normalizing Image Inputs : Helps in faster convergence.
- Dimensionality Reduction : Changing it into grayscale

Why 96*96 and GrayScale?

- It depends upon the primary memory available to you and the size of the training dataset you are using.
- In my case, I have a laptop with 8GB RAM.
- We tried to resize the image in 200*200. It was giving out of memory error. Then I changed the resolution to 96*96 and it was running fine.
- As using R,G,B will stack three matrices stacked over each other and instead of 3 value we have 1 value for each pixel., reducing dimensionality and efficient processing

left_eye_center_x	left_eye_center_y	right_eye_center_x	right_eye_center_y	left_eye_inner_corner_	left_eye_i
66.03356391	39.00227368	30.22700752	36.4216782	59.58207519	39.64742
1 43 41 39 43 39 38 42 45 49 55 51 50 52 48 45 44 52 56 92 128 13 132 141 146 145 143 34 137 146 150 146 140 136 131 129 1					
64.33293617	34.9700766	29.9492766	33.44871489	58.85617021	35.27435
65.05705263	34.90964211	30.90378947	34.90964211	59.412	36.32097
10 101 96 92 91 86 88 91 98 103 107 112 110 115 116 74 48 71 115 123 63 36 42 56 67 68 96 151 172 162 158 158 158 157 157 6					
65.22573913	37.26177391	32.02309565	37.26177391	60.00333913	39.12718
2 161 166 168 176 186 161 99 57 21 0 1 1 1 1 1 1 1 1 1 1 2 1 0 79 78 77 65 41 19 5 0 1 1 1 1 1 0 9 24 36 77 136 176 179 184 183 1					
66.72530061	39.62126135	32.24480982	38.0420319	58.56588957	39.62126
09 109 107 104 47 17 24 19 26 33 28 24 28 32 45 51 51 57 85 102 115 136 103 131 135 135 141 137 132 165 142 89 135 137 112					
69.68074766	39.96874766	29.1835514	37.56336449	62.86429907	40.16927
64.13186577	34.29004027	29.57895302	33.13804027	57.79715436	35.15404
67.4688932	39.41345243	29.35596117	39.6217165	59.55495146	40.45477
5 93 88 94 139 112 129 119 102 121 126 111 110 94 67 132 201 213 221 226 223 223 227 229 232 231 226 212 196 184 190 204 2					

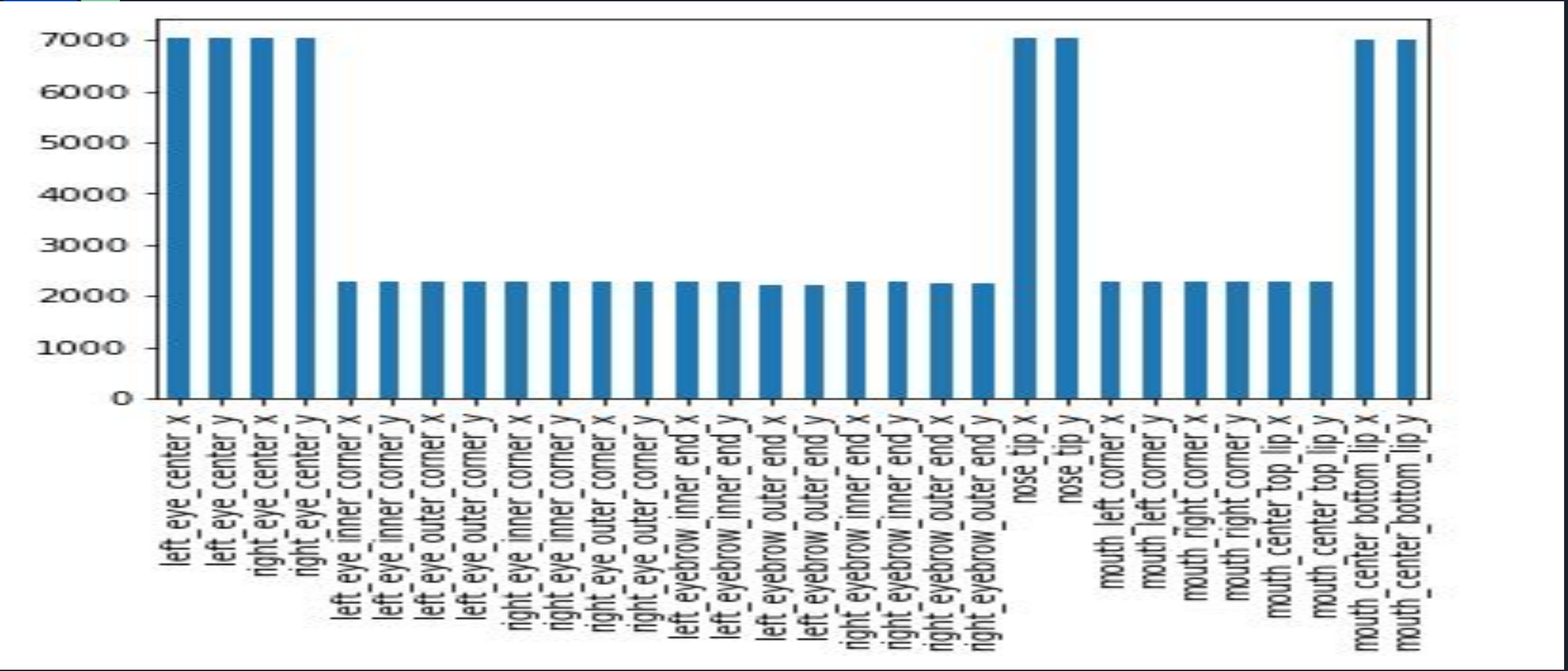
Loading And Analyzing The Data

The Image column contains the face data for which the 30 first columns represent the keypoint data (15 x-coordinates and 15 y-coordinates)

```
def string2image(string):
    """Converts a string to a numpy array."""
    return np.array([int(item) for item in string.split()]).reshape((96, 96))

def plot_faces(nrows=5, ncols=5):
    """Randomly displays some faces from the training data."""
    selection = np.random.choice(df.index, size=(nrows*ncols), replace=False)
    image_strings = df.loc[selection]['Image']
    fig, axes = plt.subplots(figsize=(10, 10), nrows=nrows, ncols=ncols)
    for string, ax in zip(image_strings, axes.ravel()):
        ax.imshow(string2image(string), cmap='gray')
        ax.axis('off')
```

Statistical Analysis Of Images

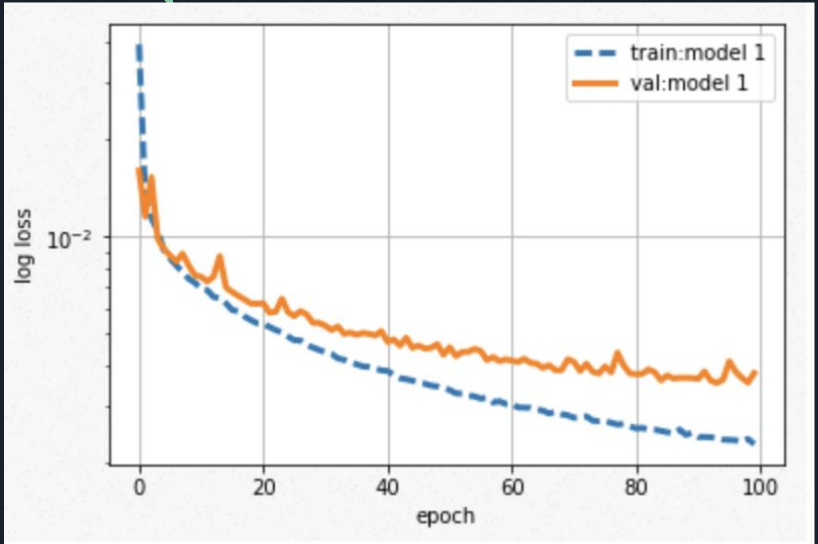
This plot tells us is that in this dataset, only 2000 images are "high quality" with all keypoints, while 5000 other images are "low quality" with only 4 keypoints labelled.

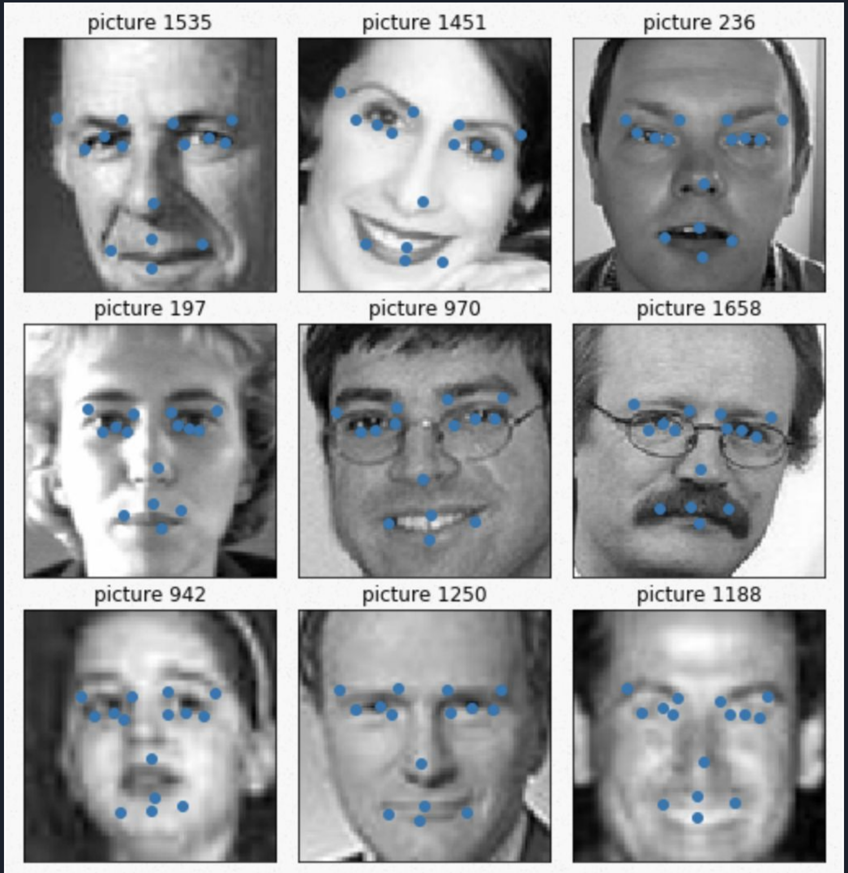


Single layer Feed forward network for setting the baseline performance

```
model = Sequential()
model.add(Dense(100,input_dim=X.shape[1]))
model.add(Activation('relu'))
model.add(Dense(30))


sgd = SGD(lr=0.01, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)
hist = model.fit(X, y, nb_epoch=100, validation_split=0.2,verbose=False)
```

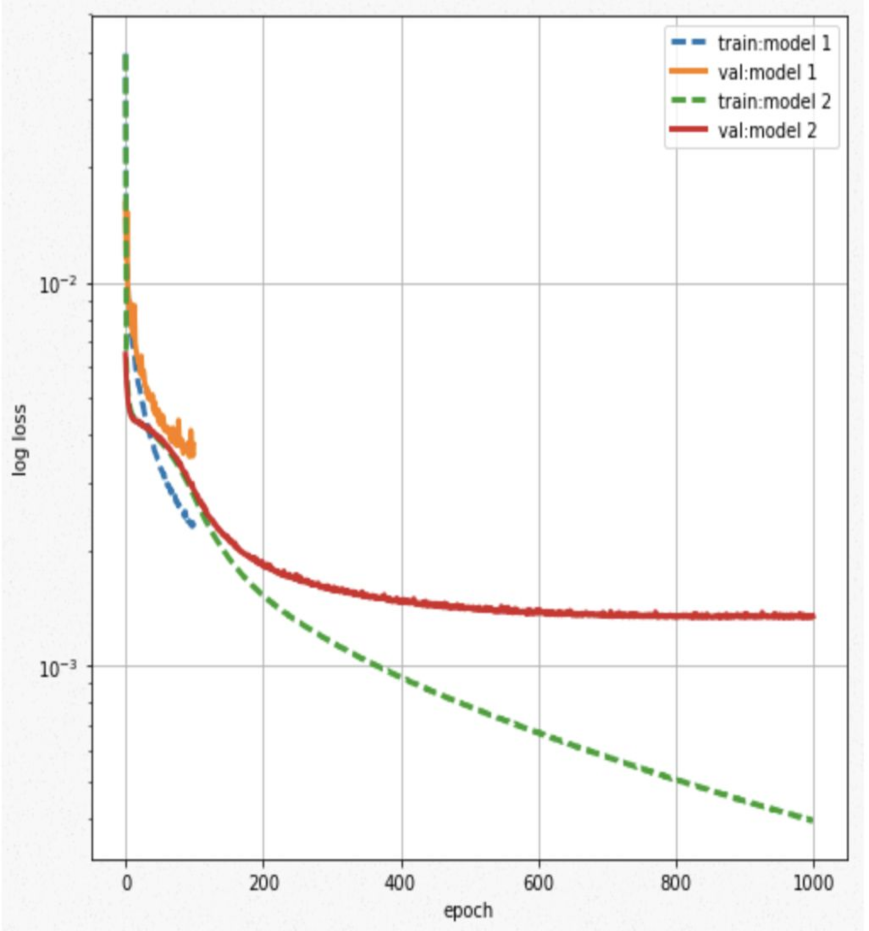
Plotting Our Training Curves With This Model





Convolutional Neural Network

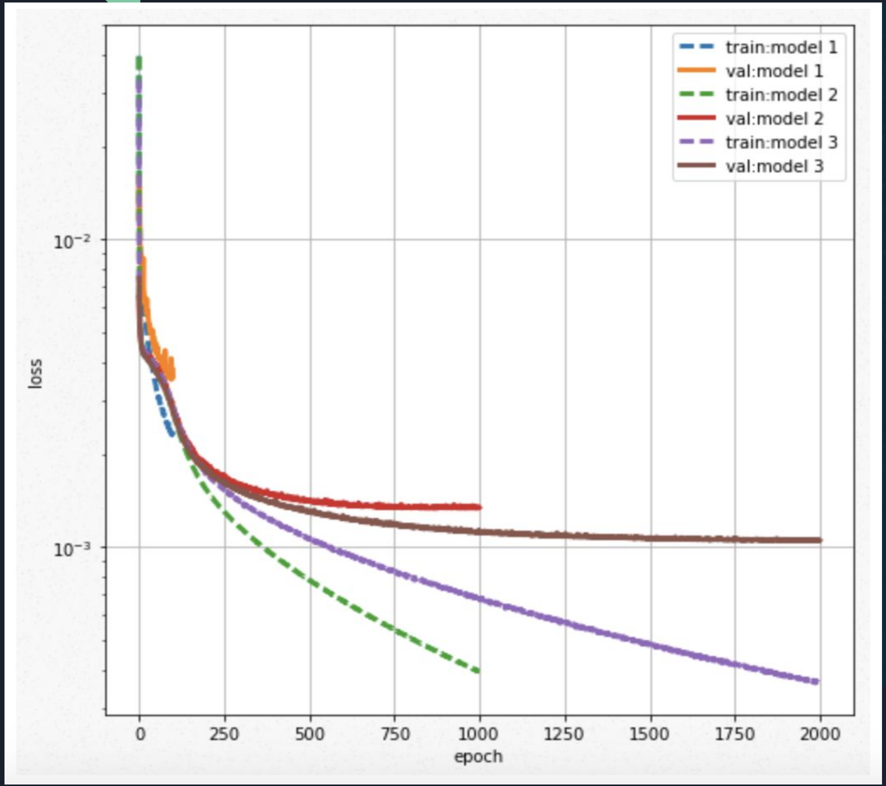
```
model = Sequential()
model.add(Conv2D(32,(3, 3), input_shape = (96, 96, 1)))
model.add(Activation('relu')) ## 96 - 3 + 2
model.add(MaxPooling2D(pool_size = (2,2))) ## 96 - (3-1)*2
if withDropout:
    model.add(Dropout(0.1))

model.add(Conv2D(64,(2,2)))
model.add(Activation('relu')) ##
model.add(MaxPooling2D(pool_size = (2,2)))
if withDropout:
    model.add(Dropout(0.1))

model.add(Conv2D(128,(2,2)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
if withDropout:
    model.add(Dropout(0.1))

model.add(Flatten())

model.add(Dense(500))
model.add(Activation('relu'))
if withDropout:
    model.add(Dropout(0.1))

model.add(Dense(500))
model.add(Activation('relu'))
if withDropout:
    model.add(Dropout(0.1))

model.add(Dense(30))
sgd = SGD(lr=0.01,momentum = 0.9,nesterov=True)
model.compile(loss="mean_squared_error",optimizer=sgd)
return(model)
```



The plot compares the difference between the baseline model and CNN. The Loss is relatively lower using CNN. The results are better.

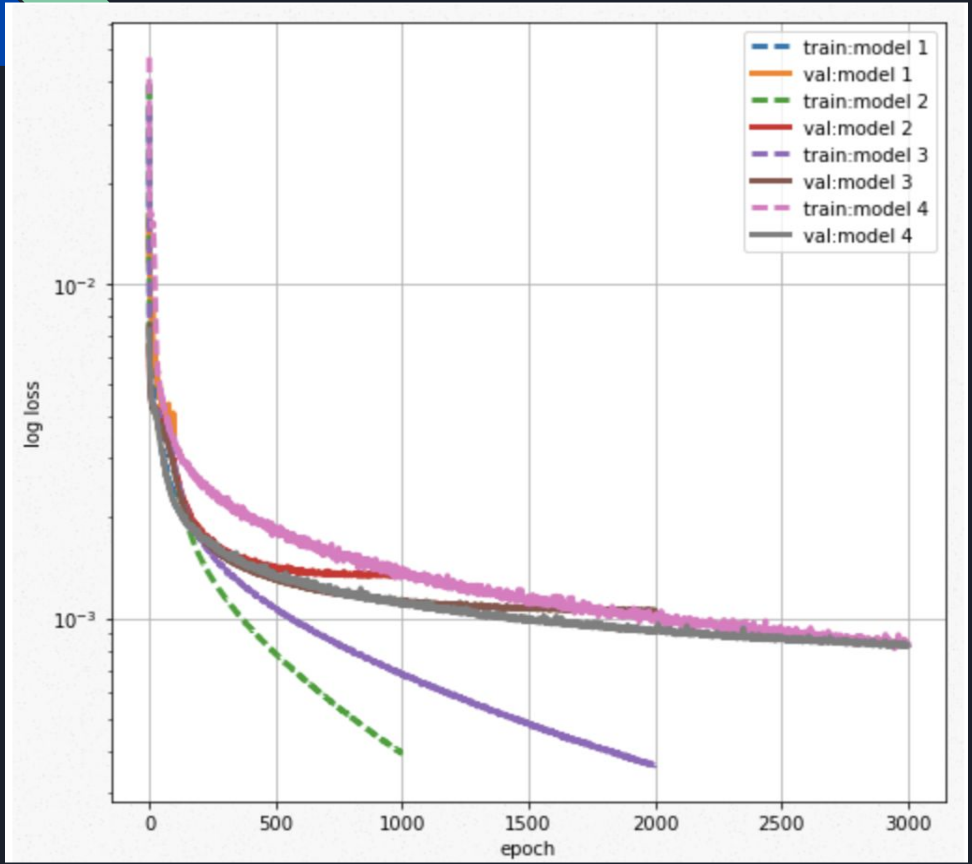
Different techniques used to improve model accuracy

- Data Augmentation with flipped pictures




```
batch_size = X_batch.shape[0]
indices = np.random.choice(batch_size, batch_size/2, replace=False)

X_batch[indices] = X_batch[indices, :, ::-1,:]
y_batch[indices, ::2] = y_batch[indices, ::2] * -1

# flip left eye to right eye, left mouth to right mouth and so on ..
for a, b in self.flip_indices:
    y_batch[indices, a], y_batch[indices, b] = (
            y_batch[indices, b], y_batch[indices, a]
        )
```

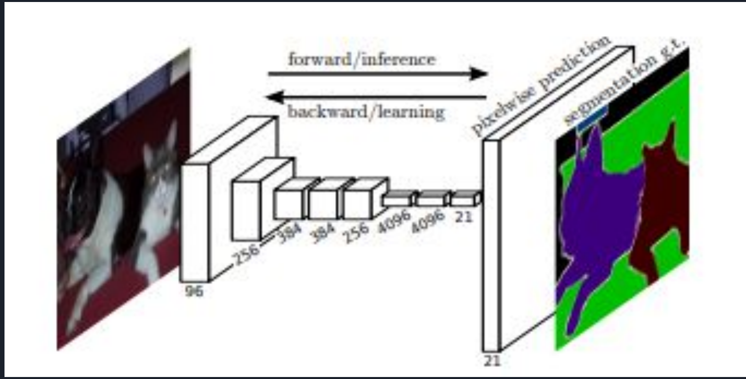
● Data Augmentation with Shifting Pictures



We even tried adding dropout layer to serve the purpose of regularisation but there was not much improvement to the model because shifting images serves the same purpose

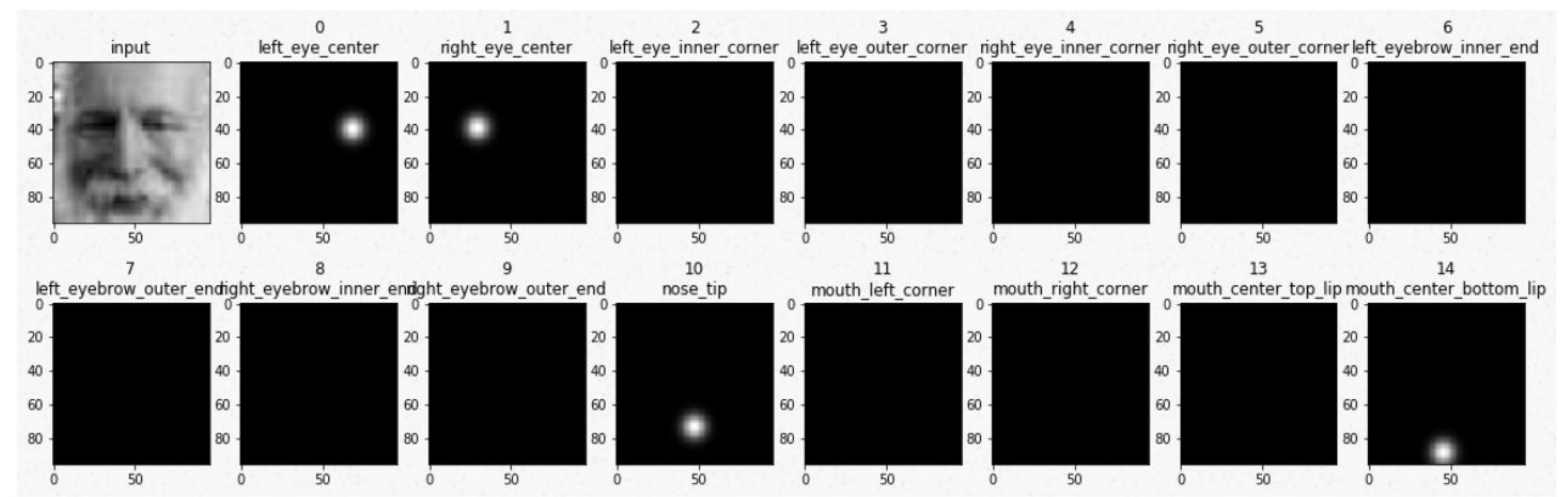
Alternate Method for Keypoint Detection

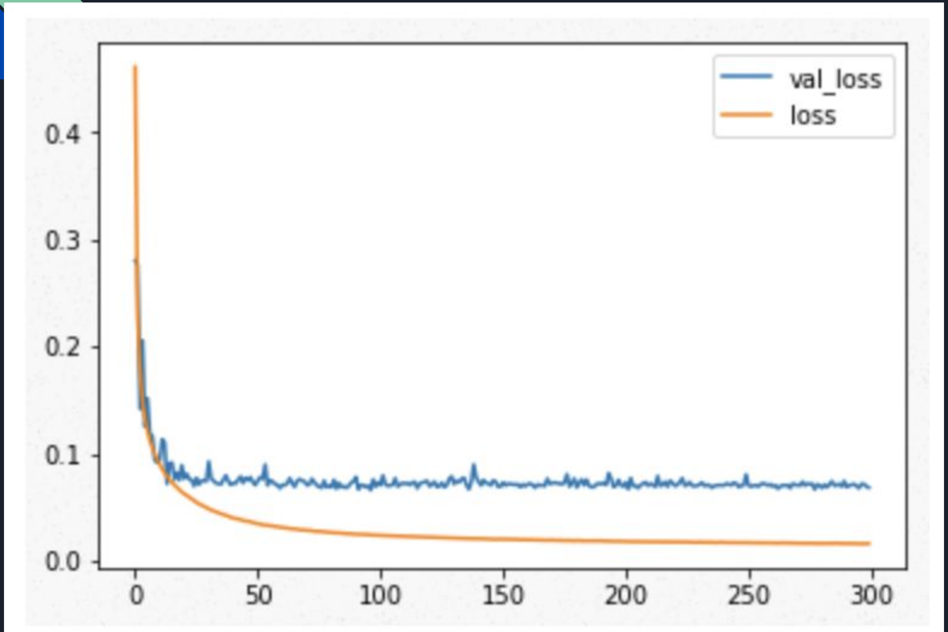
- “Fully Convolutional Networks for Semantic Segmentation” by Long, Shelhamer, and Darrell
- Performs image segmentation on a per-pixel basis
- Favors convolutional layers over fully connected layers between pooling steps
- Uses a skip architecture to integrate basic, general semantic object information with finer details such as appearance information (facial features)
- For facial keypoint detection, the network outputs a “heat map” of detected facial features
- Applicable to general object detection



Fully Convolutional Network(FCN)

- (x,y)-coordinates of the landmarks are transformed to "heatmap" using some kernels e.g. Gaussian kernel. Then the problem becomes estimating the value of the heatmap at every pixel just like object detection problem where the goal is to estimate the object's class at every pixel. Interesting!





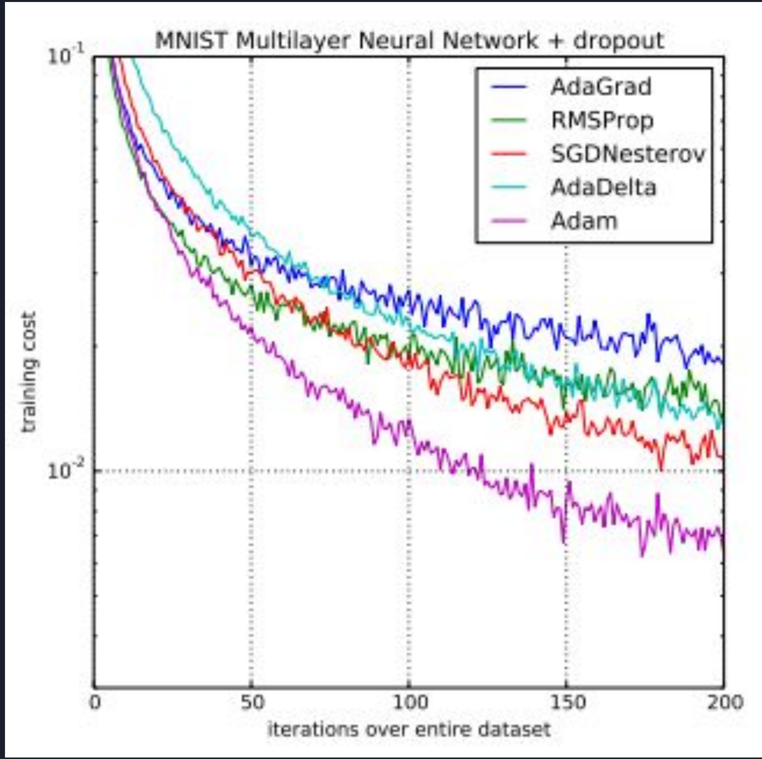
The main difference is that the fully **convolutional** net is learning filters every where. Even the decision-making layers at the end of the network are filters.

A fully convolutional net tries to learn representations and make decisions based on **local** spatial input. Appending a fully connected layer enables the network to learn something using **global** information where the spatial arrangement of the input falls away and need not apply.

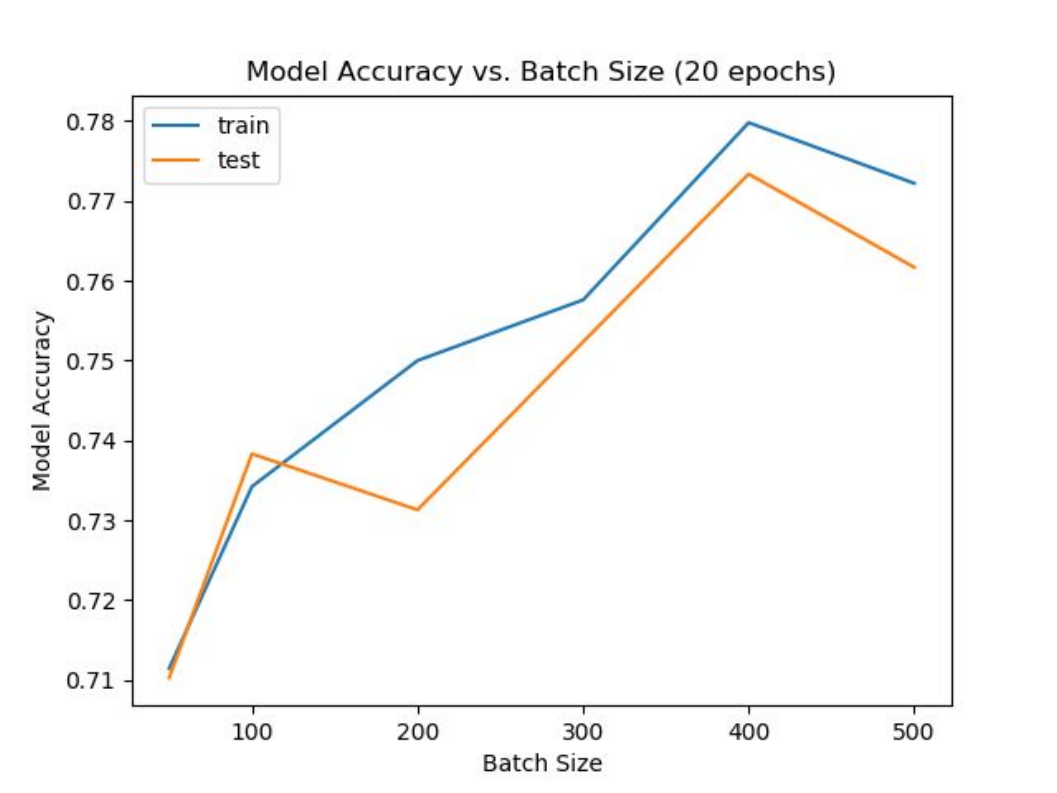
Although we found this method interesting, the accuracy of the previously discussed CNN was sufficient for our application, and we had already implemented a method to overlay our filters using the generated keypoints of that method

Training Details

- Adam Optimization Algorithm:
  - Variation of Stochastic Gradient Descent that allows each parameter to have a separate learning rate
  - Popular choice for deep learning problems. Achieves good results at a faster rate than other learning algorithms
- Error function:
  - Mean squared error between detected facial keypoints and labeled keypoints

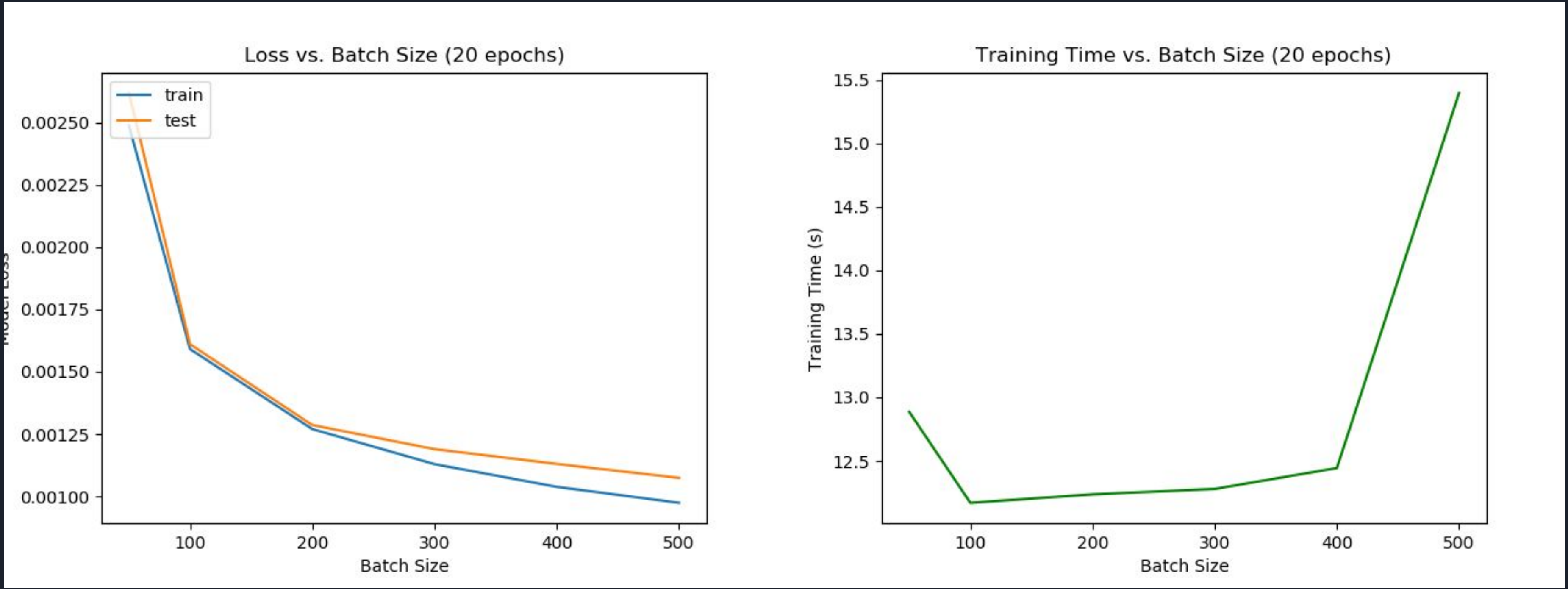


Accuracy Dependence on Batch Size



- We performed test to determine the optimum batch size to use for training of the model
- We found that loss continues to decrease for batch sizes over 400, but at a diminishing rate when training time is taken into consideration:

Loss and Training Time vs. Batch Size



How good is the result?

Application

```python
from my_CNN_model import *
import cv2
import numpy as np

# Load the model built in the previous step
my_model = load_my_CNN_model('my_model')

# Face cascade to detect faces
face_cascade = cv2.CascadeClassifier('cascades/haarcascade_frontalface_default.xml')

# Define the upper and lower boundaries for a color to be considered "Blue"
blueLower = np.array([100, 60, 60])
blueUpper = np.array([140, 255, 255])

# Define a 5x5 kernel for erosion and dilation
kernel = np.ones((5, 5), np.uint8)

# Define filters
filters = ['images/sunglasses.png', 'images/sunglasses_2.png', 'images/sunglasses_3.jpg', 'images/s
filterIndex = 0

# Load the video - 0 for webcam input
camera = cv2.VideoCapture(0)
```

- Initializing stuff like, code to read webcam inputs, detecting faces, using our CNN model, using speech simultaneously with video frames.

Detect face in the input using cascade classifier object & creating a filter switch that will show different colors applied during speech.

```
# changing the sunglasses, Hat & Mustache with speech
if speechChange[0] in sunglasses_Filter:
    filterIndex = sunglasses_Filter[speechChange[0]]
    hatIndex = hat_Filter[speechChange[0]]
    DefaultValue = RGBValue[speechChange[0]]

# Add the 'Filter' To Show The color change of the filter objects on the face
frame = cv2.rectangle(frame, (500,10), (620,65), (235,50,50), -1)
cv2.putText(frame, "FILTER", (512, 37), cv2.FONT_HERSHEY_SIMPLEX, 0.5, DefaultValue, 2, cv2.LINE_AA)
```

```
while True:

    (grabbed, frame) = camera.read()
    frame = cv2.flip(frame, 1)
    frame2 = np.copy(frame)

    # Convert to HSV and GRAY for convenience
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces using the haar cascade object
    faces = face_cascade.detectMultiScale(gray, 1.25, 6)
```

- Getting the filters using speech and applying it to the video frames.
- OpenCv is used to read inputs from camera frame by frame.

Detecting Facial KeyPoints Using The Model

```
# Loop over all the faces found in the frame
    for (x, y, w, h) in faces:
        # Make the faces ready for the model (normalize, resize and stuff)
        gray_face = gray[y:y+h, x:x+w]
        color_face = frame[y:y+h, x:x+w]

        # Normalize to match the input format of the model - Range of pixel to [0, 1]
        gray_normalized = gray_face / 255

        # Resize it to 96x96 to match the input format of the model
        original_shape = gray_face.shape # A Copy for future reference
        face_resized = cv2.resize(gray_normalized, (96, 96), interpolation = cv2.INTER_AREA)
        face_resized_copy = face_resized.copy()
        face_resized = face_resized.reshape(1, 96, 96, 1)

        # Predict the keypoints using the model
        keypoints = my_model.predict(face_resized)

        # De-Normalize the keypoints values
        keypoints = keypoints * 48 + 48

        # Map the Keypoints back to the original image
        face_resized_color = cv2.resize(color_face, (96, 96), interpolation = cv2.INTER_AREA)
        face_resized_color2 = np.copy(face_resized_color)

        # Pair the keypoints together - (x1, y1)
        points = []
        for i, co in enumerate(keypoints[0][0::2]):
            points.append((co, keypoints[0][1::2][i]))
```

- Resizing the image to 96*96 pixels to match the input format of the model.
- After some preprocessing the keypoints are mapped to original image.

Adding the Hat Filters

- Cascade Classifier to find the regions of the captured image corresponding to faces
- The face cascade detection returns:
- (x, y) = position of upper-left corner of bounding box
- (w, h) = width and height of bounding box
- For each hat image, the proper offsets from (x, y) were computed in terms of w and h so that the hat images will adjust correctly as faces move

Adding Moustache and Sunglass Filters

- The width, height, and positional offsets of the moustache and sunglasses are computed using the positions of relevant keypoints:
- Sunglasses: keypoints 7, 8, 9, and 10 (lower nose and eye region)
- Moustache: keypoints 10, 11, 12, 13 (lower nose and mouth region)

Some examples of the filters used





Voice Recognition and Multithreading

- To incorporate voice and filter addition, multithreading was implemented
- Thread 1- Video Capture and selfie filters
- Thread 2- Speech Recognition.
- A Shared Variable “speechToAction” is used between both the threads. If no input provided, filters continue working with default values.

Application

- Extension of this application as a part of shopping app/website so that users can try on products virtually before buying.

Future Scope

- Reducing the lag while changing filters
- Improving model accuracy by changing hyperparameters
- Including more filter options and better image resolution
- Accommodating different face orientations
    - Would require a training dataset with faces at various angles
    - Detect additional keypoints to determine the orientation
    - Would require additional filter images for different possible head poses/orientations

References

- OpenCV Face Detection Documentation: https://docs.opencv.org/3.4.3/d7/d8b/tutorial_py_face_detection.html
- Facial Keypoints Detection: https://www.kaggle.com/c/facial-keypoints-detection
- Speech Recognition: https://pypi.org/project/SpeechRecognition/

Thank You!