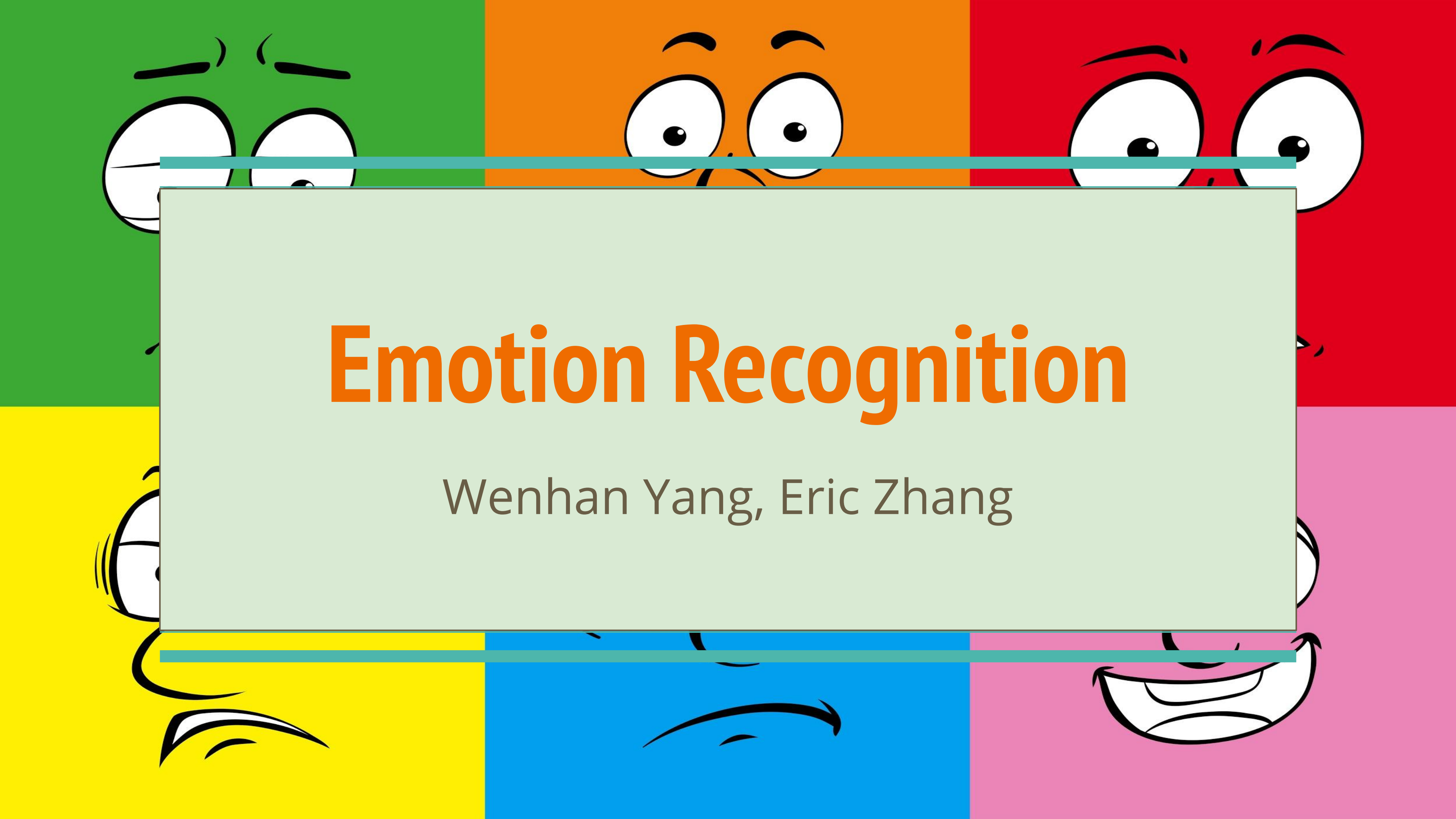
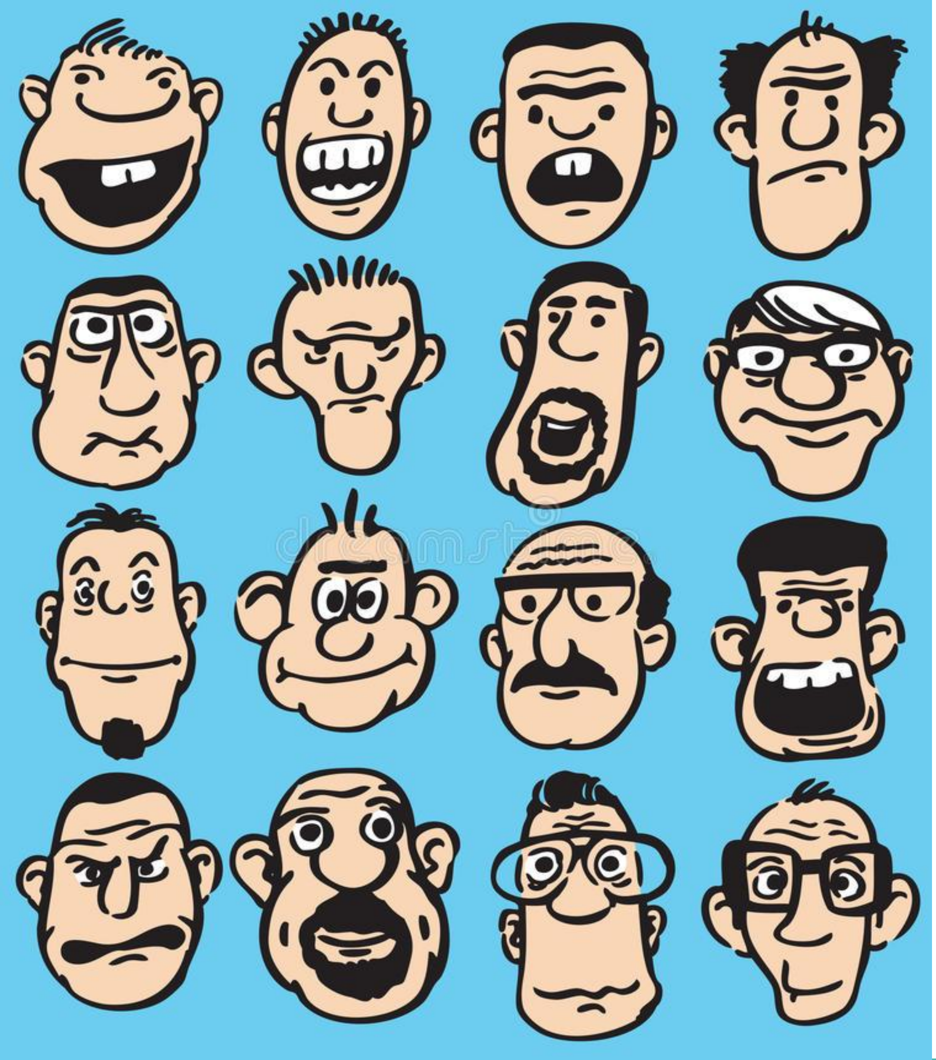


Emotion Recognition

Wenhan Yang, Eric Zhang

Outline

- Introduction
- Overview of Dataset
- Model Design
- Results and Discussion
- Future Studies



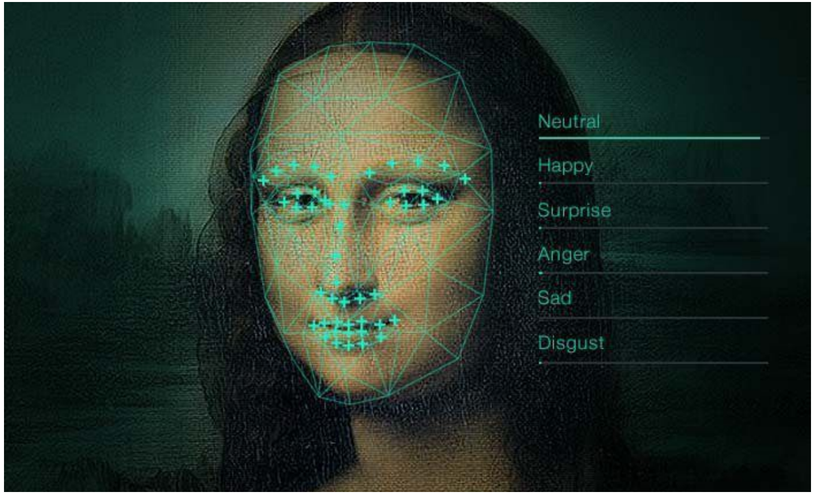
1. INTRODUCTION TO

EMOTION RECOGNITION

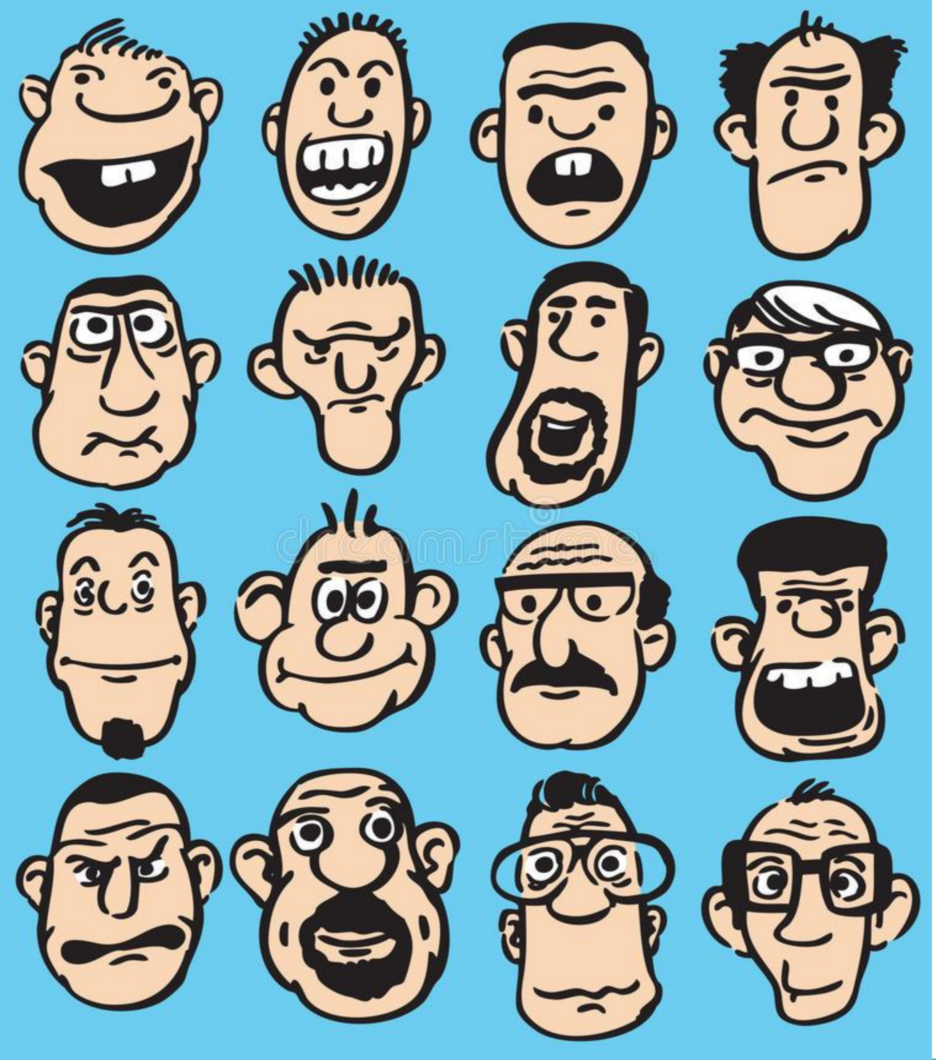
Introduction

Real-time interaction

- Recommender systems
- Security (Driving)



Emotion Detection System



2. OVERVIEW OF DATASET

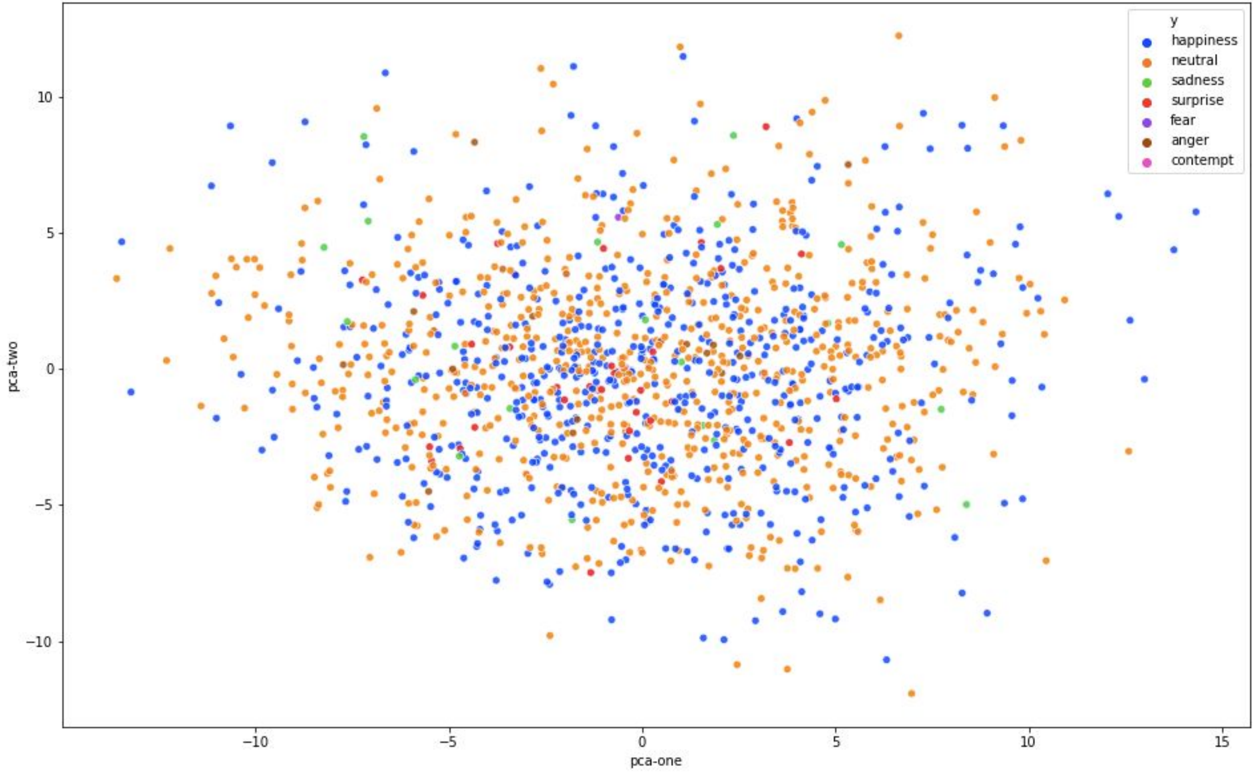
Overview of Dataset

- **Dataset:** *https://github.com/muxspace/facial_expressions*
- **Raw data format: (picture, label)**
  - **Picture:** 350x350 pixel. Mostly Grayscale, a few RGB with different dimension.
  - **Label:** { 'anger', 'contempt', 'disgust', 'fear', 'happiness', 'neutral', 'sadness', 'surprise' }
  - **Example:**

( , ‘happiness’)

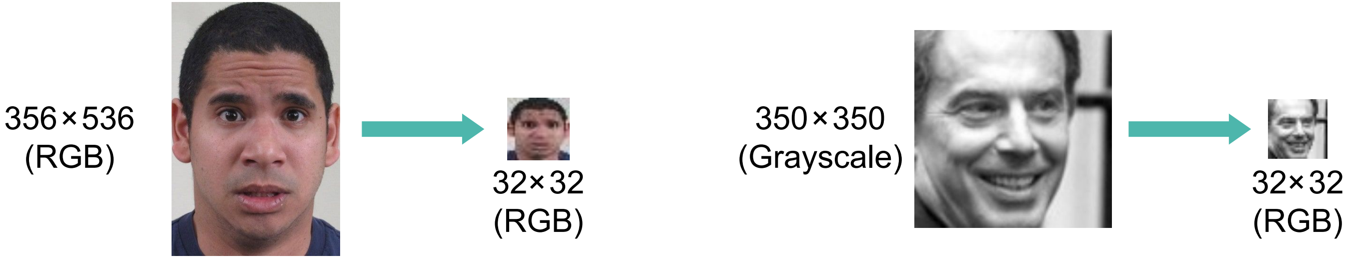
Visualizing Data Features

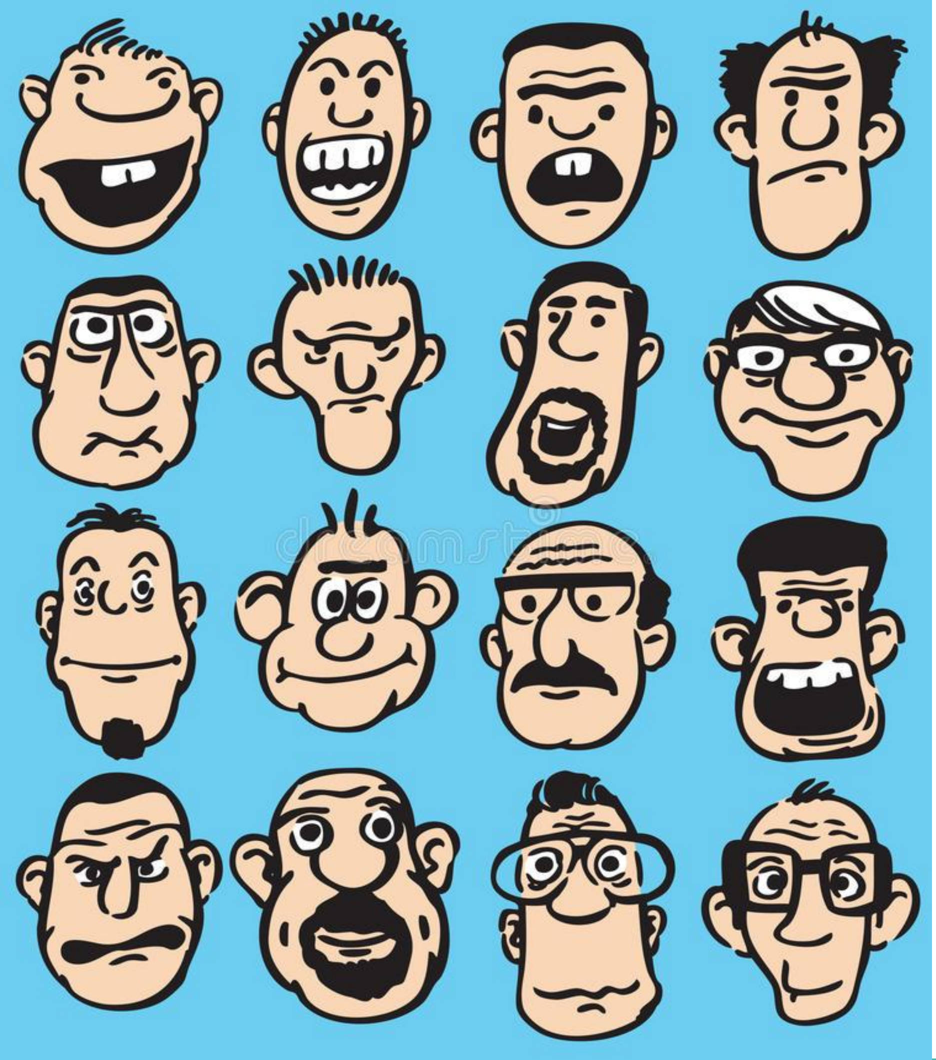
<matplotlib.axes._subplots.AxesSubplot at 0x1a23203c18>



Preprocessing of Dataset

- **Resizing:** reshape the pictures into 32x32 pixel size
  - Reduce input data sizes while change resolution
  - Restricted data as square, so rectangular pictures are stretched/compressed
- **Channel Matching:** transfer pictures into 3-channel RGB
- **Input Format:**
  - Logistic & Random Forest: 13690x3072 numpy array of uint8s. Each row of the array stores a 32x32 colour image. The first 1024 entries contain the red channel values, the next 1024 the green, and the final 1024 the blue. The image is stored in row-major order, so that the first 32 entries of the array are the red channel values of the first row of the image.
  - ConvNet Models: 4D numpy array of dimension (sample_size, 32,32,3), sample_size = 500. Here we used the first 500 data from our dataset, as the computational cost is high especially for our laptops.





3. DIVE INTO

MODEL DESIGN

Model Overview

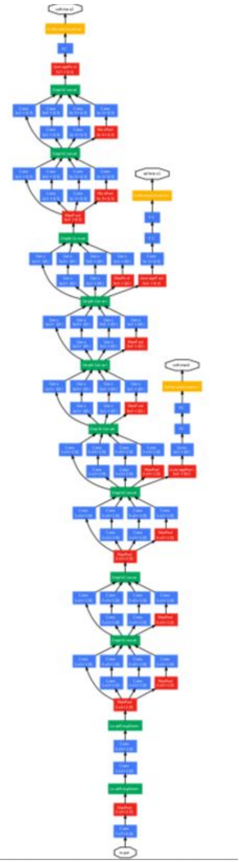
- Logistic regression
  - *model = LogisticRegression(solver = 'lbfgs', multi_class = 'multinomial', penalty='l2', C = 2e10, max_iter = 1e7)*
- KNN
- Decision Tree
- Random Forest
- CNN with different architectures
  - VGG (VGG11, VGG13, VGG19)
  - GoogLeNet
  - SEResNet (SEResNet18, SEResNet34)
  - InceptionV3
  - MobileNet

Model Design - GoogLeNet

22 Layer Network

- **Inception architecture**: how an optimal local sparse structure in a convolutional vision network can be approximated and covered by readily available dense components.
- Judiciously applying **dimension reductions** and **projections** wherever the computational requirements would increase too much otherwise — rectified linear activation.
- Allow for increasing the number of the units at each stage significantly **without an uncontrolled blow-up in computational complexity**.

type
convolution
max pool
convolution
max pool
inception (3a)
inception (3b)
max pool
inception (4a)
inception (4b)
inception (4c)
inception (4d)
inception (4e)
max pool
inception (5a)
inception (5b)
avg pool
dropout (40%)
linear
softmax

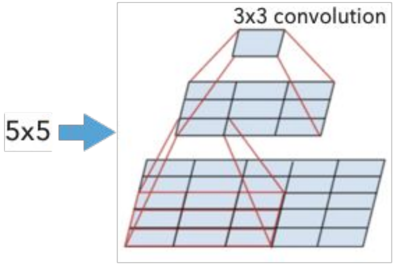


Model Design - Inception V3

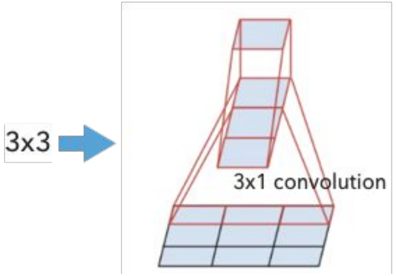
42 Layer Network

❖ **Factorizing Convolutions with Larger Filter Size**

- smaller

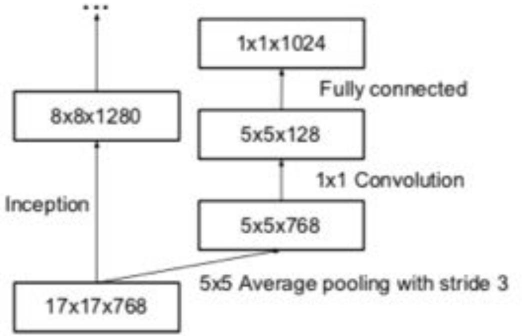


- asymmetric

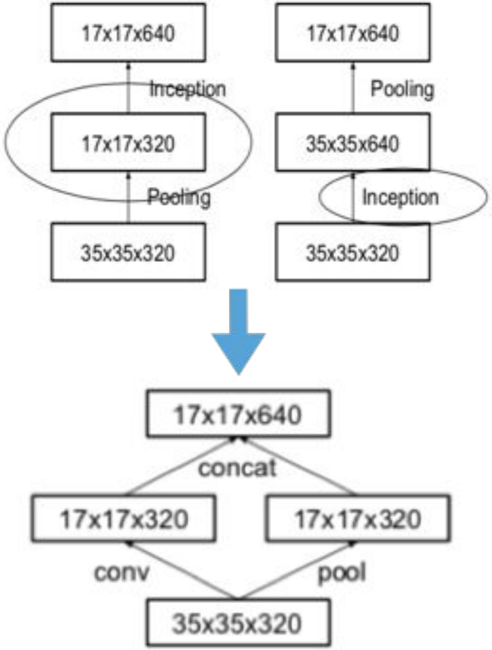


❖ **Utility of Auxiliary Classifiers**

- promote stable learning and convergence
- accuracy reaches slightly higher plateau by the end of training
- regularizer



❖ **Efficient Grid Size Reduction**

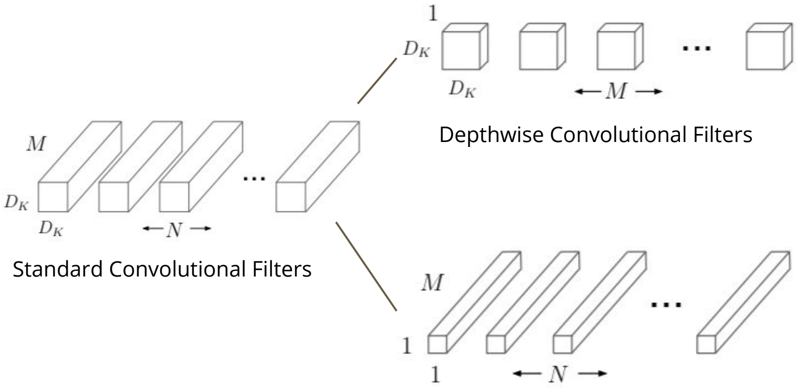


Model Design - Inception V3

42 Layer Network

- Label Smoothing Regularization(LSR)
  - Consider cross entropy: $\ell = -\sum_{k=1}^{K} \log(p(k))q(k)$. Minimizing the loss may cause overfitting and reduces the model adaptability.
  - Introducing $q'(k|x) = (1-\epsilon)\delta_{k,y} + \epsilon u(k)$, which is a mixture of original ground-truth distribution and fixed distribution $u(k)$ with assigned weights.
  - $$H(q',p) = -\sum_{k=1}^{K} \log p(k) q'(k) = (1-\epsilon)H(q,p) + \epsilon H(u,p)$$
- Good Performance on Low Resolution Input
Receptive Field Size	Top-1 Accuracy (single frame)
79×79	75.2%
151×151	76.4%
299×299	76.6%

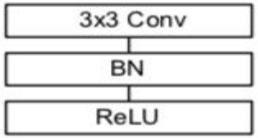
Model Design - Mobile Net



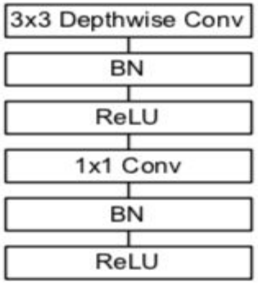
Depthwise Convolutional Filters

Standard Convolutional Filters

Pointwise Convolutional Filters



Standard Convolutional Layer

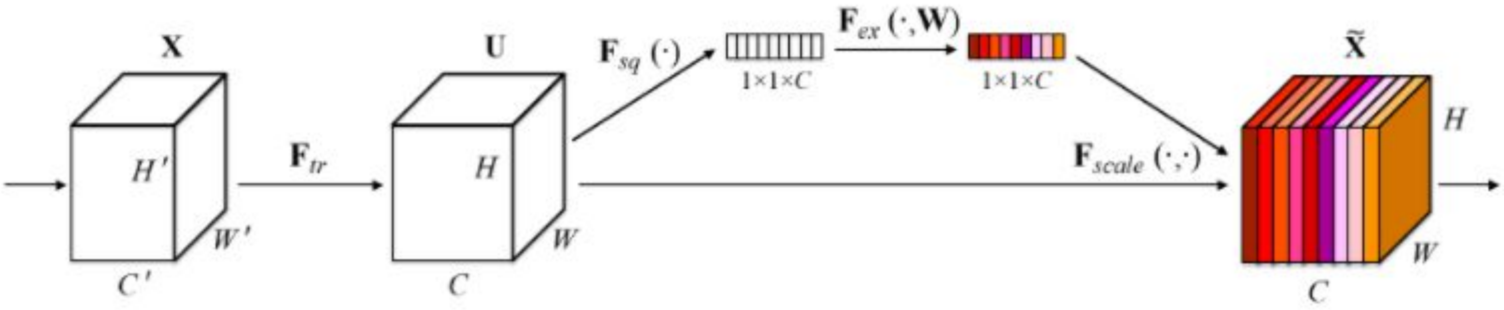


Depthwise Seperable Layer

- Built on **depthwise separable convolutions**, except first layer which is full convolution
- MobileNets use both batchnorm and ReLU nonlinearities for both layers.
- **Small architecture** and **low latency**
- **Width multiplier**: $\alpha \in (0, 1]$
  - give thinner models
  - reduce computational cost
  - reduce the number of parameters quadratically(α^2)
- **Resolution Multiplier**: $\rho \in (0, 1]$
  - reducing representation
  - reducing computational cost by ρ^2

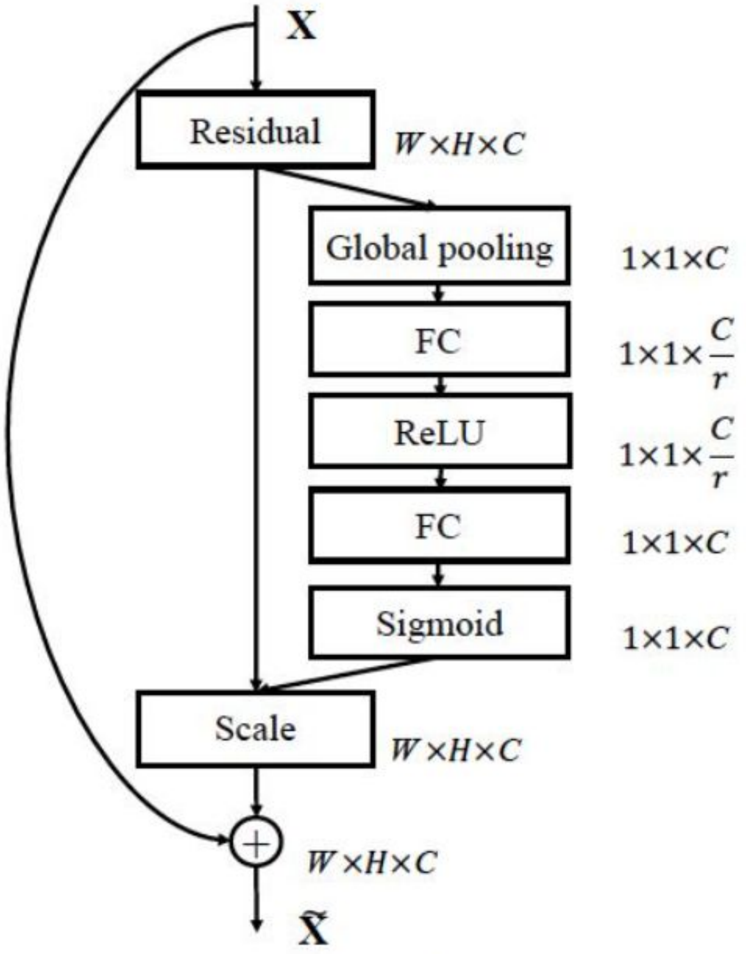
Model Design - SENet

General SENet

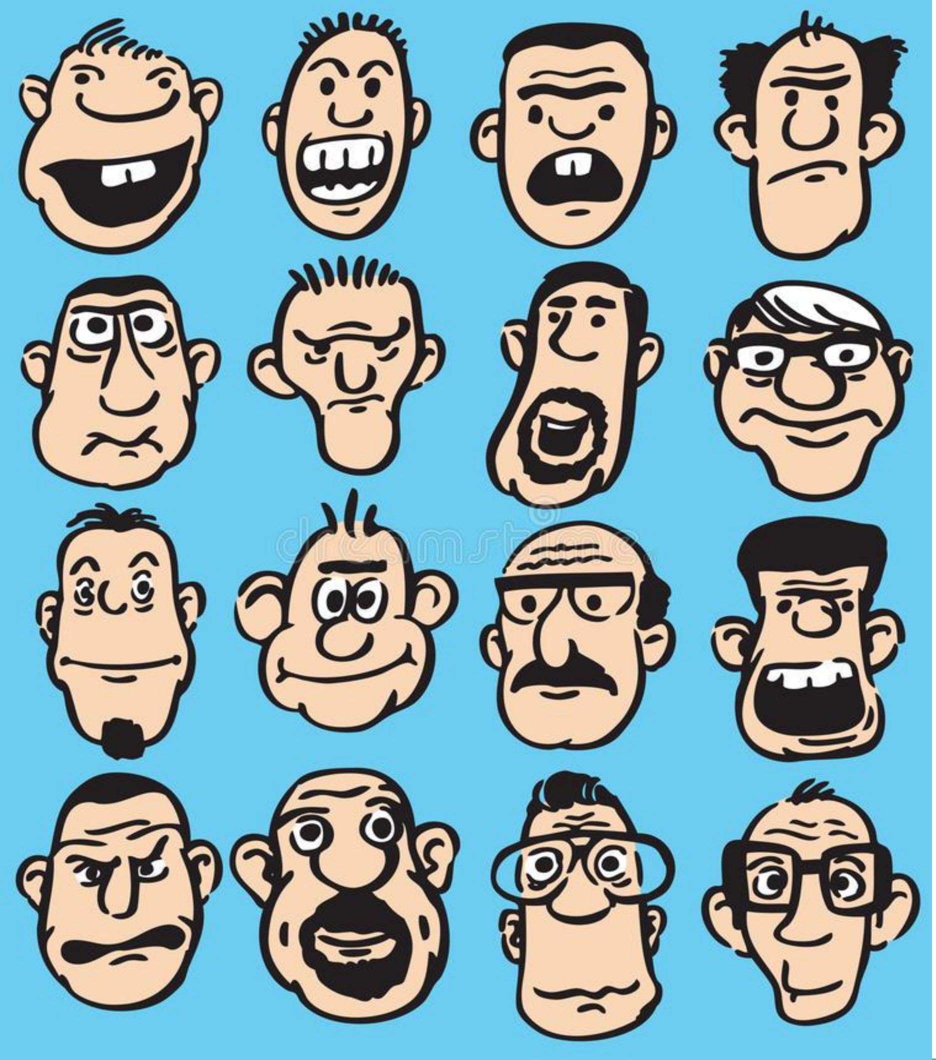


Model Design - SEResnet

Key idea: Give each channel a weight



SE-ResNet Module



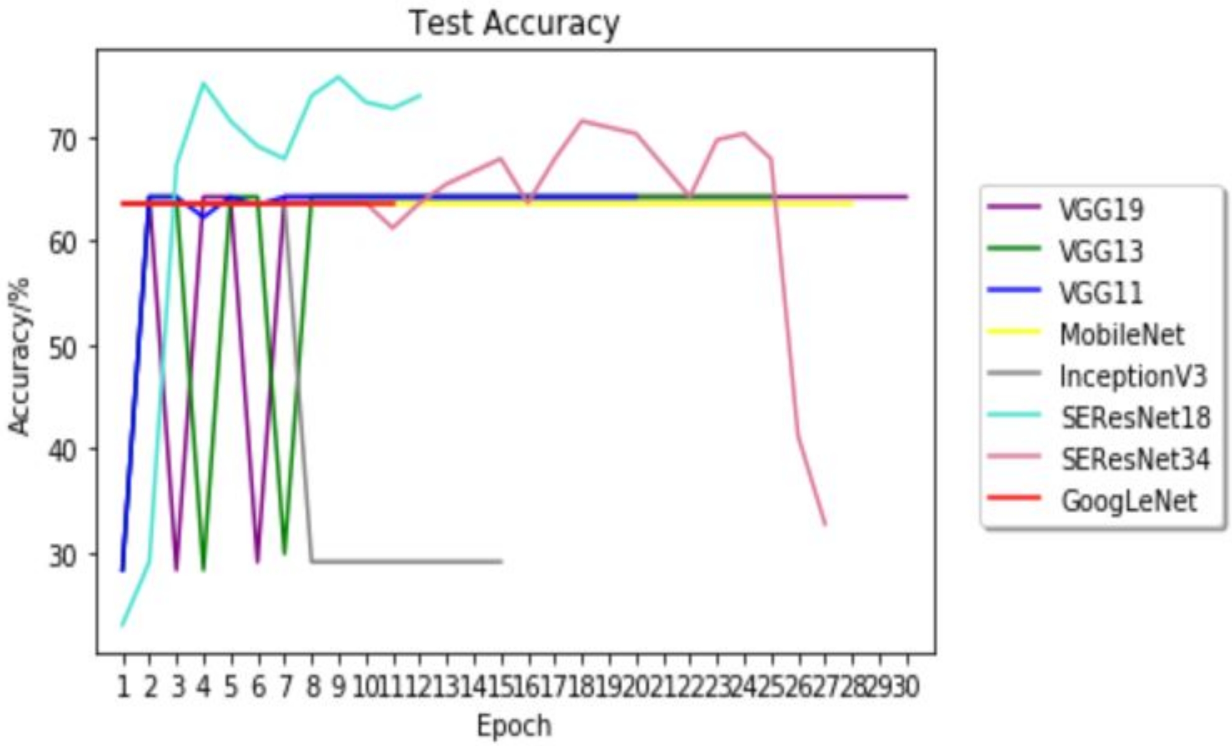
4. CONTINUE WITH

RESULTS & DISCUSSIONS

Training Phase - ConvNets

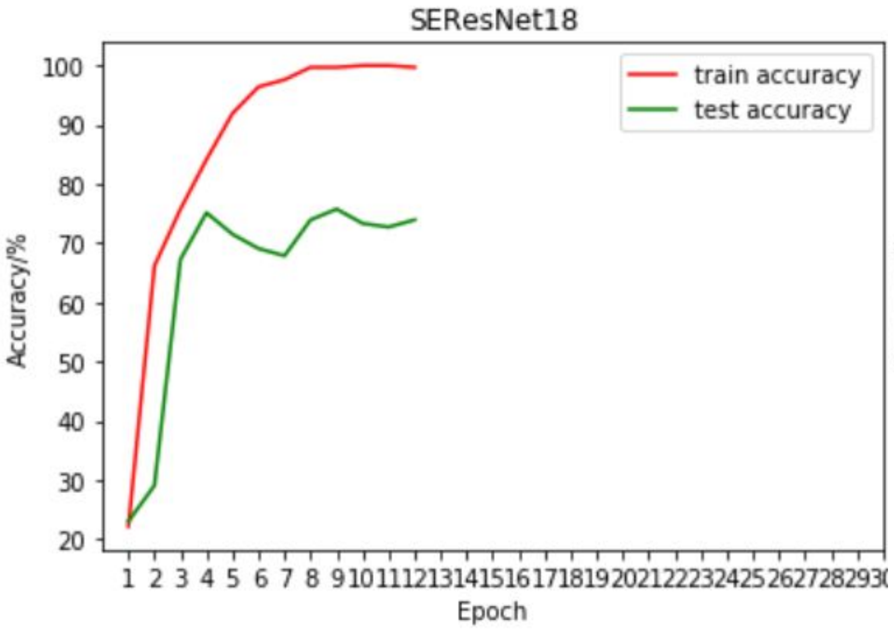
- Use the first 500 labeled images with each 32x32, RGB channel.
- Train test split ratio: 2:1.
- Train for 30 epoches, learning rate = 0.001, batch size = 100.
- **Early stopping:** when test loss increases for 3 consecutive epoches.
- **Loss function:** Cross Entropy Loss
  - *loss_object = tf.keras.losses.SparseCategoricalCrossentropy()*
- **Learning rate update:** Adam
  - *optimizer = tf.keras.optimizers.Adam(args.lr)*

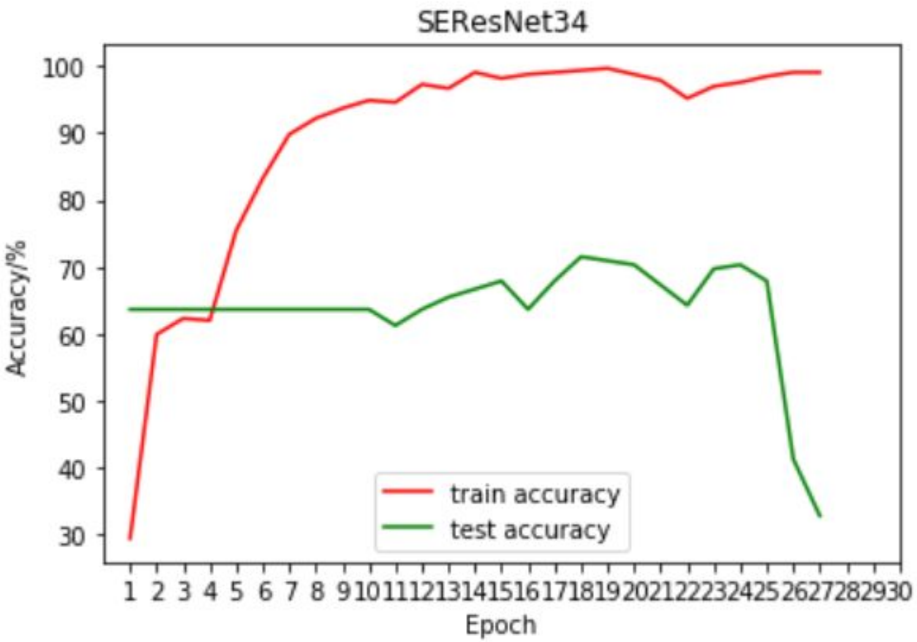
Results and discussions



Model	Best Test Accuracy
Logistic Regression	72.2%
KNN	61.0%
Random Forest	71.5%
Decision Tree	60.1%
VGG11,13,18	64.2%
GoogLeNet	63.6%
InceptionV3	63.6%
MobileNet	63.6%
SEResNet18	75.8%
SEResNet34	71.5%

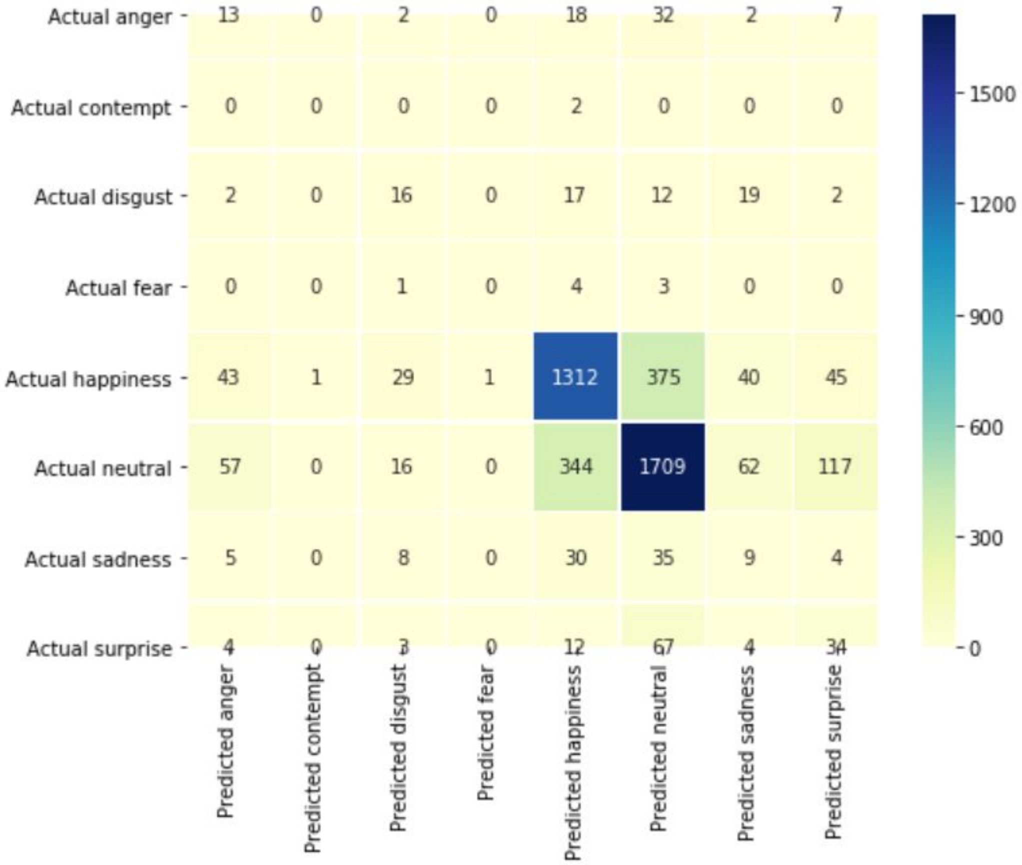
Results and discussions





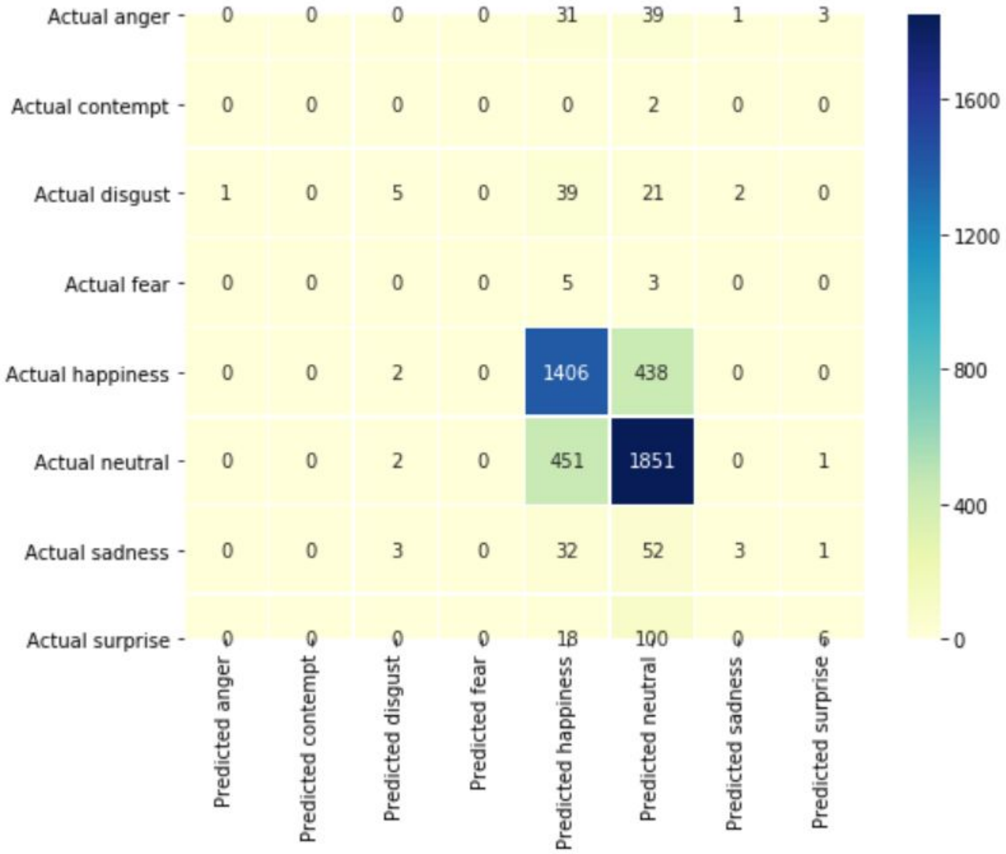
Results and discussions

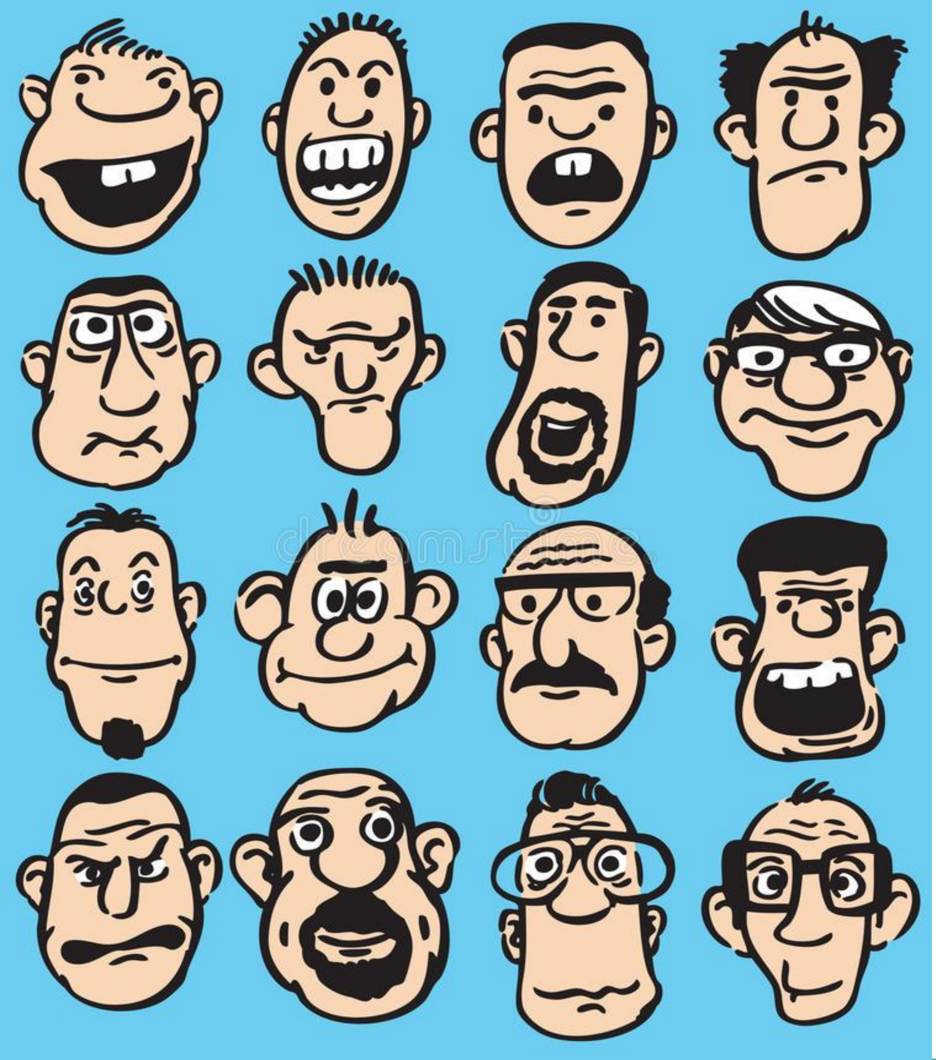
Logistic regression



Results and discussions

Random Forest





5. ABOUT

FUTURE WORKS

Future Works

- Dataset not even
  - Solution: Use Bootstrap
- Dataset labels are similar
  - Solution: Design more labels and train with unambiguous images

THANK YOU!