CS 512

Project Report

Real Time Facial Emotion Detection in Live Videos Feed

By
Mahima Agarwal (A20383052)
Sudipta Swarnakar (A20377210)

Introduction:

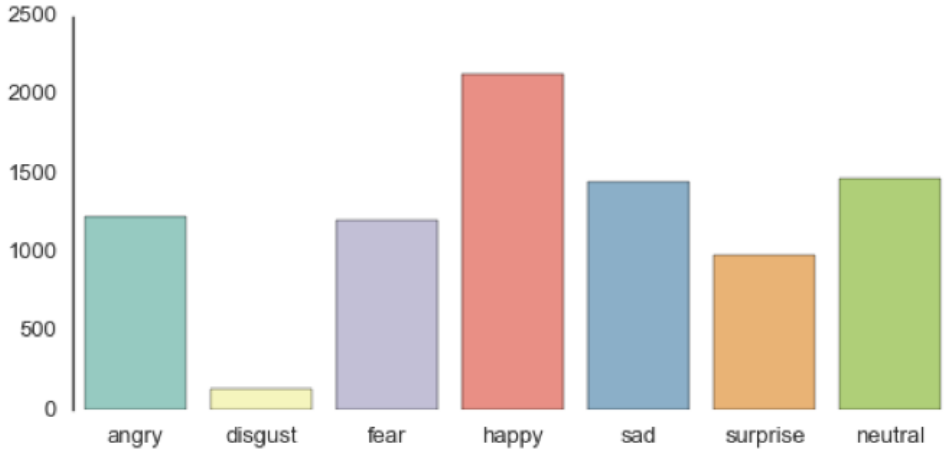
Facial emotion recognition is one of the most important cognitive functions that our brain performs efficiently. Human facial expressions can be easily classified into 7 basic emotions: happy, sad, surprise, fear, anger, disgust, and neutral. Facial emotion is an important cue for sensing human emotion and intention in people. Facial expression analysis has attracted significant attention in the computer vision community during the past decade, since it lies in the intersection of many important applications in a wide variety of domains including judicial systems, interactive games, online/remote education, entertainment, and other intelligent systems. The difficulty of this problem lies in its interdisciplinary nature. The problem involves two important tasks in emotion recognition which are given as,

i) Feature Extraction
ii) Classification

Recent discoveries and GPU Computational power have made it possible to efficiently train Deep neural networks with multiple layers of hidden units, allowing for more hierarchical, incrementally abstracted representations. Humans are well-trained in reading the emotions of others, in fact, at just 14 months old, babies can already tell the difference between happy and sad. **But can computers do a better job than us in accessing emotional states?** To answer the question, we designed a deep learning neural network that gives machines the ability to make inferences about our emotional states. In other words, we give them eyes to see what we can see.

The Datasets:

The dataset we used in this project is from a Kaggle Facial Expression Recognition Challenge few years back (FER2013). It comprises a total of 35887 pre-cropped, 48-by-48-pixel grayscale images of faces each labeled with one of the 7 emotion classes: anger, disgust, fear, happiness, sadness, surprise, and neutral. Out of 35887 images, 28,709 images (48x48 pixels) are for training set whereas 3,589 images have been used for after-training validation.



An overview of FER2013 Dataset

Data Vizualization:

FER-2013 data. Each column consists of faces with
the same expression: starting from the leftmost columns: Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral.

FER-2013 data Table:

Label	Training set	Testing set
Angry	3995	491
Disgust	436	55

Fear	4097	528
Happy	7215	879
Sad	4830	594
Surprise	3171	416
Neutral	4965	626

DATA PREPROCESSING

In [6]:
```
df = pd.read_csv("fer2013.csv")
df.head()
```

Out[6]:

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
4	6	4 0 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training

In [7]:
```
df.shape
```

Out[7]:
```
(35887, 3)
```

In [8]:
```
df["Usage"].value_counts()
```

Out[8]:
```
Training       28709
PrivateTest     3589
PublicTest      3589
Name: Usage, dtype: int64
```

In [9]:
```
train = df[["emotion", "pixels"]][df["Usage"] == "Training"]
train.isnull().sum()
```

Out[9]:
```
emotion    0
pixels     0
dtype: int64
```

In [10]:
```
train['pixels'] = train['pixels'].apply(lambda im: np.fromstring(im, sep=' '))
x_train = np.vstack(train['pixels'].values)
y_train = np.array(train["emotion"])
x_train.shape, y_train.shape
```

Out[10]:
```
((28709, 2304), (28709,))
```

In [11]:
```
public_test_df = df[["emotion", "pixels"]][df["Usage"]=="PublicTest"]
```

In [12]:
```
public_test_df["pixels"] = public_test_df["pixels"].apply(lambda im: np.fromstring(im, sep=' '))
x_test = np.vstack(public_test_df["pixels"].values)
y_test = np.array(public_test_df["emotion"])
```

In [13]:
```
x_train = x_train.reshape(-1, 48, 48, 1)
x_test = x_test.reshape(-1, 48, 48, 1)
x_train.shape, x_test.shape
```

Out[13]:
```
((28709, 48, 48, 1), (3589, 48, 48, 1))
```

In [14]:
```
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
y_train.shape, y_test.shape
```

Out[14]:
```
((28709, 7), (3589, 7))
```

In [15]:
```
import seaborn as sns
plt.figure(0, figsize=(12,6))

for i in range(1, 13):
    plt.subplot(3,4,i)
    plt.imshow(x_train[i, :, :, 0], cmap="gray")

plt.tight_layout()
plt.show()
```

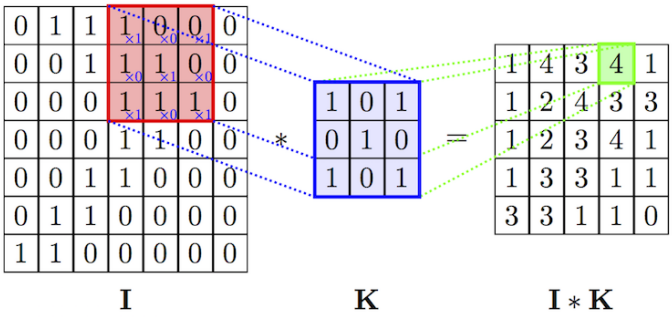
SOLUTION APPROACH-

The Deep Learning Model (Convolution Neural Networks):

Deep learning is a popular technique used in computer vision. We chose convolutional neural network (CNN) layers as building blocks to create our model architecture. A CNN architecture (see Figure 2) is formed by stacking distinct layers that transform the input volume into an output volume using a differentiable function.

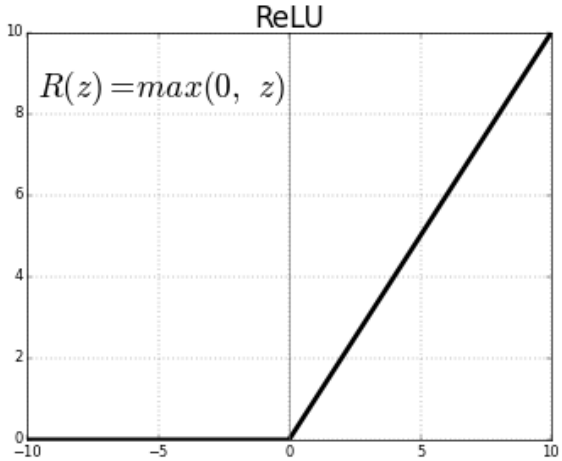
Convolutional layer

This layer is the core component of a CNN. Its parameters are a set of learnable filters or kernels. The filters have a receptive field which extends thorough the input. In the forward pass, every filter is convolved across the height and width of the input and the dot product between the filter entries are computed producing a 2-d activation map corresponding to it. Hence the CNN learns filters which activate when they see a specific type of feature in the input. Then the activation maps are stacked along the depth dimension to form the complete output of the convolution layer. An entry in the output volume can be interpreted as a neuron which works on a small input region and shares layer parameters with neurons in the same activation map.



ReLU (Rectified Linear Units) layer

ReLU is a neuron layer that uses the non-saturating activation function. It increases the nonlinearity of the decision function as well as that of the network. Though other activation functions like tanh and sigmoid can be used but we used ReLU with each Convolution layer, since it makes training faster without overfitting to the training data.



Pooling Layer

The pooling layer allows a number of features to be diminished and non-overlapped. It reduces spatial resolution and thus naturally decreases the importance of exactly where a feature was found, just keeping the rough location. It is an important step when building CNNs as adding more convolutional layers can greatly affect computational time. We used a popular pooling method called MaxPooling2D that uses (2, 2) windows across the feature map only keeping the maximum pixel value. The pooled pixels form an image with dimention reduced by 4.

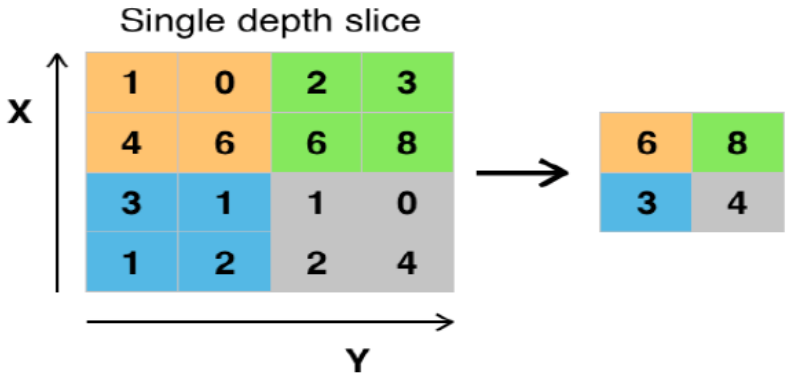
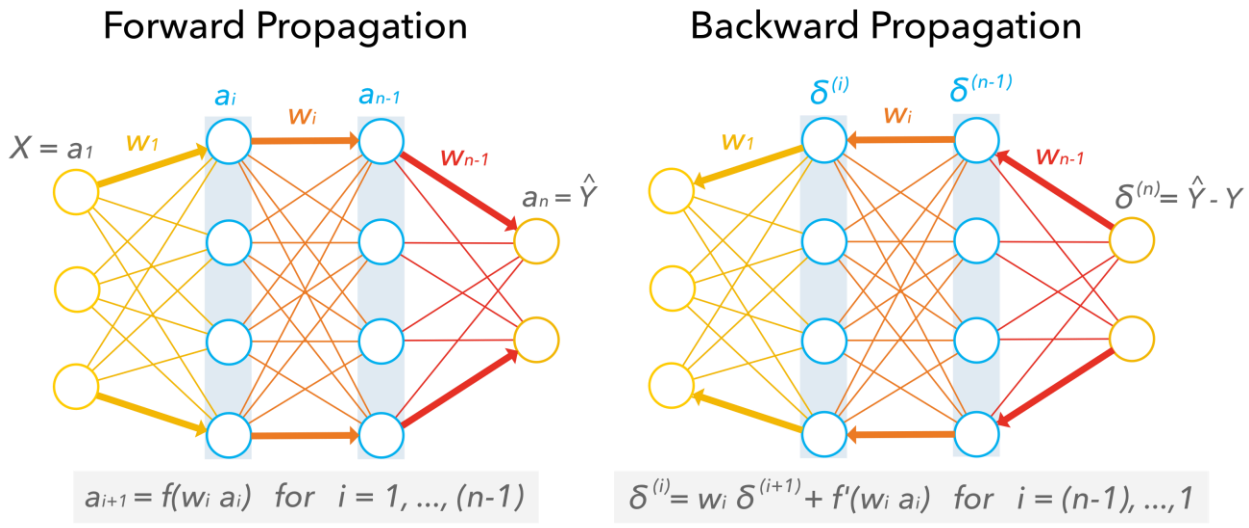


Figure 3. Max pooling with a 2x2 filter and stride = 2

Fully Connected layer

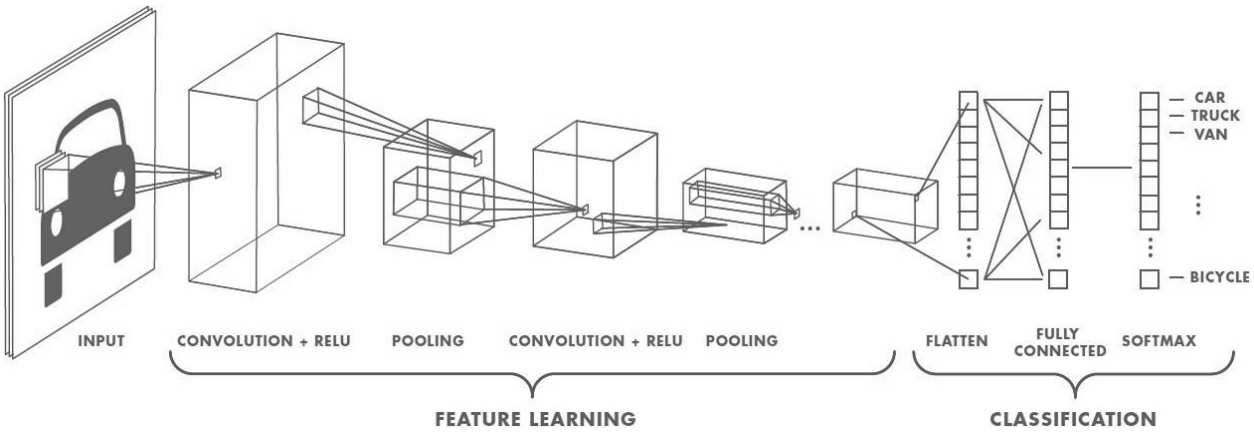
The high-level reasoning is done using fully connected layers in a neural network. Neurons in this layer are connected to all activations in the previous layer. The activations of these neurons are computed using a matrix multiplication and then a bias offset. This layer takes a large number of input features and transform features through layers connected with trainable weights.The weights are trained by forward propagation of training data then backward propagation of its errors. **Back propagation** starts from evaluating the difference between prediction and true value, and back calculates the weight adjustment needed to every layer before. We can control the training speed and the complexity of the architecture by tuning the hyper-parameters, such as **learning rate** and **network density**. As we feed in more data, the network is able to gradually make adjustments until errors are minimized.



Essentially, the more layers/nodes we add to the network the better it can pick up signals. As good as it may sound, the model also becomes increasingly prone to overfitting the training data. One method to prevent overfitting and generalize on unseen data is to apply **dropout**. Dropout randomly selects a portion (for our model it's 20%) of nodes to set their weights to zero during training. This method can effectively control the model's sensitivity to noise during training while maintaining the necessary complexity of the architecture.

Output Layer

Instead of using sigmoid activation function, we used Softmax at the output layer. This output presents itself as a probability for each emotion class. Therefore, the model is able to show the detail probability composition of the emotions in the face.



Simple CNN Architecture (AlexNet)

More Complex Models

After doing the training for the first time we realized that the simple CNN architecture(**AlexNet**) with an input, three convolution layers, one dense layer, and an output layer is too simple to detect the subtle details in facial expressions. This is where deep learning comes in. Given the pattern complexity of facial expressions, it is necessary to build with a deeper architecture in order to identify subtle signals. So we used two different deeper CNN architectures like **VGGNet** and **GoogleNet** to train our Deep Learning model and found that **VGGNet** performed better than **GoogleNet** architecture for emotion detection.

Training Model on AWS GPU Instance Using Caffe with Tensorflow backend :

As the above mentioned models like **VGGNet** and **GoogleNet** are computational heavy so it's really hard to train it on traditional CPUs. So we setup g2.2xlarge Deep Learning AMI (Amazon Linux) cluster on AWS EC2 which offers NVIDIA GRID K520 GPUs having 3072 CUDA Cores with 4.7 TFLOPS performance. On AWS instance, we used Keras with Tensorflow backend to train our Deep Learning model.

Amazon Web Services **Deep Learning Instance:**

Deep Learning AMI (Amazon Linux) Version 1.0 - ami-77eb3a0f

Select

Amazon Linux

Free tier eligible

Deep Learning AMI with Conda-based virtual environments for Apache MXNet, TensorFlow, Caffe2, PyTorch, Theano, CNTK and Keras

64-bit

Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

▾ AMI Details

Deep Learning AMI (Amazon Linux) Version 1.0 - ami-77eb3a0f

Free tier eligible

Deep Learning AMI with Conda-based virtual environments for Apache MXNet, TensorFlow, Caffe2, PyTorch, Theano, CNTK and Keras

Root Device Type: ebs Virtualization type: hvm

▾ Instance Type

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
g2.2xlarge	26	8	15	1 x 60	Yes	High

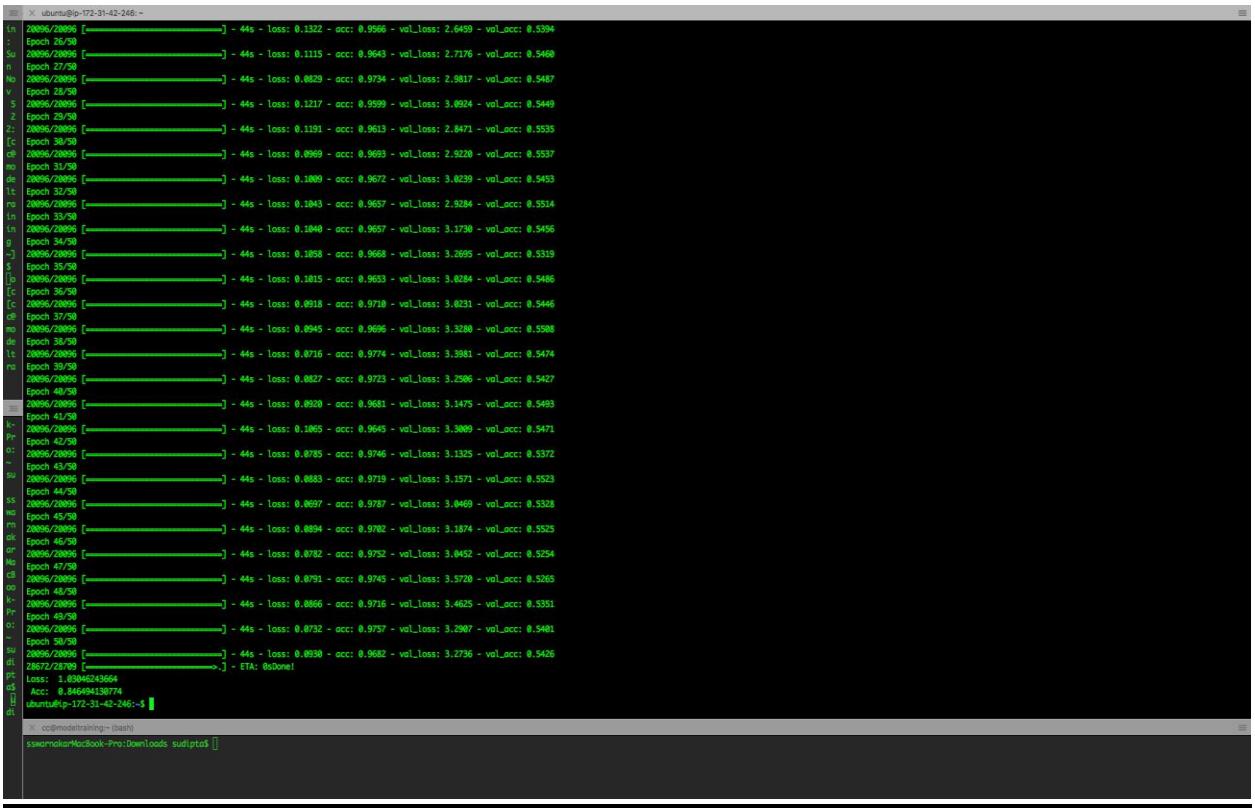
▾ Security Groups

NVIDIA GeForce GRID Processor

GPUs	CUDA Cores	Memory Size	Memory Perf	Shader Perf	TDP
Dual	3,072	8GB	320 GB/sec	4.7 TFLOPS	250W

```
ubuntu@ip-172-31-42-246:~$ python EmotionDetectionONNModel.py
Using TensorFlow backend.
X_train shape:  (28709, 48, 48, 1)
y_train shape:  (28709, 7)
img size:  48 1
batch size:  128
nb_epoch:  50
EmotionDetectionONNModel.py:55: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(32, (3, 3), padding="same", activation="relu", input_shape=(48, 48, 1...)`
  model.add(Conv2D(32, 3, 3, border_mode='same', activation='relu',input_shape=(48,48,1)))
EmotionDetectionONNModel.py:56: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(32, (3, 3), padding="same", activation="relu")`
  model.add(Conv2D(32, 3, 3, border_mode='same', activation='relu'))
EmotionDetectionONNModel.py:57: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(32, (3, 3), padding="same", activation="relu")`
  model.add(Conv2D(32, 3, 3, border_mode='same', activation='relu'))
EmotionDetectionONNModel.py:60: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(64, (3, 3), padding="same", activation="relu")`
  model.add(Conv2D(64, 3, 3, border_mode='same', activation='relu'))
EmotionDetectionONNModel.py:61: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(64, (3, 3), padding="same", activation="relu")`
  model.add(Conv2D(64, 3, 3, border_mode='same', activation='relu'))
EmotionDetectionONNModel.py:62: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(64, (3, 3), padding="same", activation="relu")`
  model.add(Conv2D(64, 3, 3, border_mode='same', activation='relu'))
EmotionDetectionONNModel.py:65: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(128, (3, 3), padding="same", activation="relu")`
  model.add(Conv2D(128, 3, 3, border_mode='same', activation='relu'))
EmotionDetectionONNModel.py:66: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(128, (3, 3), padding="same", activation="relu")`
  model.add(Conv2D(128, 3, 3, border_mode='same', activation='relu'))
EmotionDetectionONNModel.py:67: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(128, (3, 3), padding="same", activation="relu")`
  model.add(Conv2D(128, 3, 3, border_mode='same', activation='relu'))
Training....
/usr/local/lib/python2.7/dist-packages/keras-2.0.8-py2.7.egg/keras/models.py:848: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
Train on 28096 samples, validate on 8613 samples
Epoch 1/50
2017-11-05 22:04:39.260758: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.1 instructions, but these are available on your machine and could speed up CPU computations.
2017-11-05 22:04:39.260803: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.2 instructions, but these are available on your machine and could speed up CPU computations.
2017-11-05 22:04:39.260819: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX instructions, but these are available on your machine and could speed up CPU computations.
2017-11-05 22:04:40.494982: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:893] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2017-11-05 22:04:40.495265: I tensorflow/core/common_runtime/gpu/gpu_device.cc:955] Found device 0 with properties:
name: GRID K520
major: 3 minor: 0 memoryClockRate (GHz) 0.797
pciBusID 0000:00:03.0
Total memory: 3.94GiB
Free memory: 3.91GiB
2017-11-05 22:04:40.495296: I tensorflow/core/common_runtime/gpu/gpu_device.cc:976] DMA: 0
2017-11-05 22:04:40.495310: I tensorflow/core/common_runtime/gpu/gpu_device.cc:986] 0:   Y
2017-11-05 22:04:40.495331: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1045] Creating TensorFlow device (/gpu:0) -> (device: 0, name: GRID K520, pci bus id: 0000:00:03.0)
28096/28096 [==============================] - 106s - loss: 1.8172 - acc: 0.2794 - val_loss: 1.6177 - val_acc: 0.3697
Epoch 2/50
28096/28096 [==============================] - 44s - loss: 1.5520 - acc: 0.4003 - val_loss: 1.4503 - val_acc: 0.4376
Epoch 3/50
28096/28096 [==============================] - 44s - loss: 1.4118 - acc: 0.4596 - val_loss: 1.3576 - val_acc: 0.4708
Epoch 4/50
28096/28096 [==============================] - 44s - loss: 1.2988 - acc: 0.5030 - val_loss: 1.2816 - val_acc: 0.5114
Epoch 5/50
28096/28096 [==============================] - 44s - loss: 1.2005 - acc: 0.5408 - val_loss: 1.2246 - val_acc: 0.5403
Epoch 6/50
28096/28096 [==============================] - 44s - loss: 1.1062 - acc: 0.5844 - val_loss: 1.2042 - val_acc: 0.5439
Epoch 7/50
28096/28096 [==============================] - 44s - loss: 1.0291 - acc: 0.6136 - val_loss: 1.2118 - val_acc: 0.5436
Epoch 8/50
28096/28096 [==============================] - 44s - loss: 0.9292 - acc: 0.6530 - val_loss: 1.2858 - val_acc: 0.5491
Epoch 9/50
18688/28096 [==================>...] - ETA: 2s - loss: 0.8230 - acc: 0.6936
```

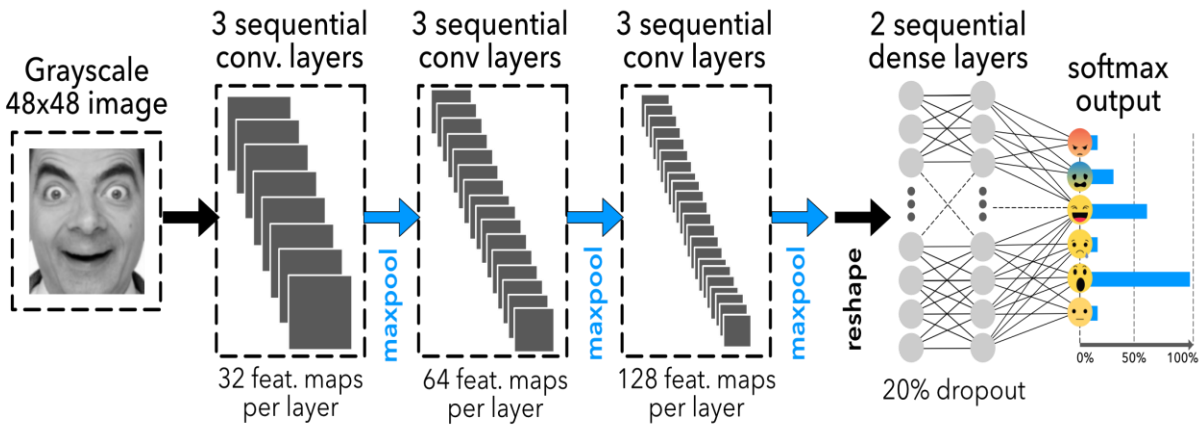
```
cc@modeltraining:~ (bash)
aswarnakarMacBook-Pro:Downloads sudipta$
```



From the above screenshots, it’s quite clear that our training accuracy is almost 97%

VGGNet Architecture:

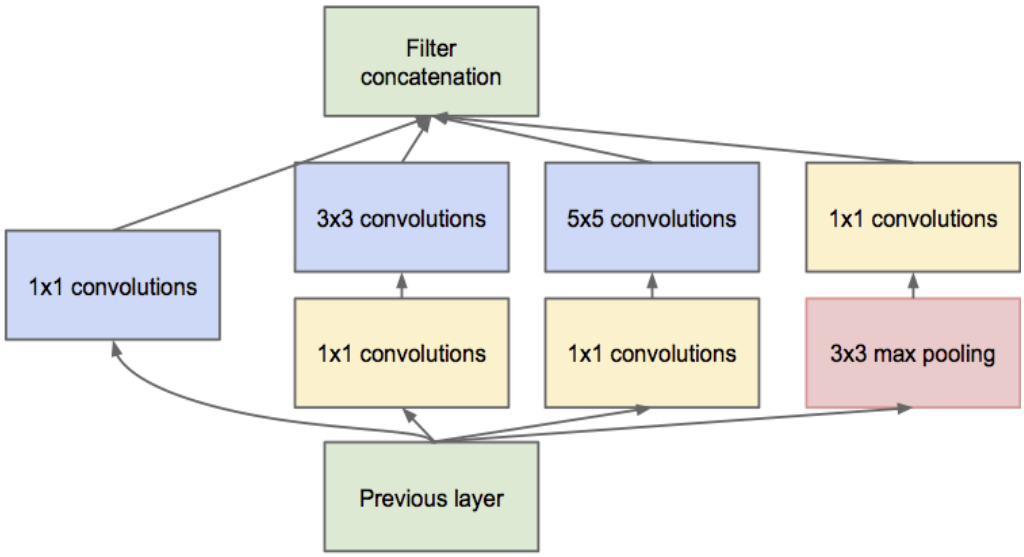
For VGGNet, our final model architecture ended up having 9 layers deep in convolution with one max-pooling after every three convolution layers as seen in below figure.



Source Code for VGGNet Model:

```
model = Sequential()
model.add(Conv2D(32, 3, 3,border_mode='same',activation='relu',input_shape=(48,48,1)))
model.add(Conv2D(32, 3, 3, border_mode='same', activation='relu'))
model.add(Conv2D(32, 3, 3, border_mode='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, 3, 3, border_mode='same', activation='relu'))
model.add(Conv2D(64, 3, 3, border_mode='same', activation='relu'))
model.add(Conv2D(64, 3, 3, border_mode='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, 3, 3, border_mode='same', activation='relu'))
model.add(Conv2D(128, 3, 3, border_mode='same', activation='relu'))
model.add(Conv2D(128, 3, 3, border_mode='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(7, activation='softmax'))

# optimizer:
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print 'Training....'
model.fit(x_train, y_train, nb_epoch=nb_epoch, batch_size=batch_size,validation_split=0.3, shuffle=True, verbose=1)

#Model result:
loss_and_metrics = model.evaluate(x_train, y_train, batch_size=batch_size, verbose=1)
print 'Done!'
print 'Loss: ', loss_and_metrics[0]
print ' Acc: ', loss_and_metrics[1]
```

GoogleNet Architecture:



Source Code for GoogleNet:

```
# model architecture:

model = Graph()
model.add_input(name='n00', input_shape=(1,48,48))

# layer 1
model.add_node(Convolution2D(64,1,1, activation='relu'), name='n11', input='n00')
model.add_node(Flatten(), name='n11_f', input='n11')
model.add_node(Convolution2D(96,1,1, activation='relu'), name='n12', input='n00')
model.add_node(Convolution2D(16,1,1, activation='relu'), name='n13', input='n00')
model.add_node(MaxPooling2D((3,3),strides=(2,2)), name='n14', input='n00')

# layer 2
model.add_node(Convolution2D(128,3,3, activation='relu'), name='n22', input='n12')
model.add_node(Flatten(), name='n22_f', input='n22')
model.add_node(Convolution2D(32,5,5, activation='relu'), name='n23', input='n13')
model.add_node(Flatten(), name='n23_f', input='n23')
model.add_node(Convolution2D(32,1,1, activation='relu'), name='n24', input='n14')
model.add_node(Flatten(), name='n24_f', input='n24')

# output layer
model.add_node(Dense(1024, activation='relu'), name='layer4',
        inputs=['n11_f', 'n22_f', 'n23_f', 'n24_f'], merge_mode='concat')
model.add_node(Dense(10, activation='softmax'), name='layer5', input='layer4')
model.add_output(name='output1',input='layer5')

print 'Training....'
model.compile(loss={'output1':'categorical_crossentropy'}, optimizer='adam',metrics=['accuracy'])
model.fit({'n00':x_train, 'output1':y_train}, nb_epoch=nb_epoch, batch_size=batch_size,validation_split=0.3, shuffle=True,
verbose=1)

#Model result:
loss_and_metrics = model.evaluate(x_train, y_train, batch_size=batch_size, verbose=1)
print 'Loss: ', loss_and_metrics[0]
print ' Acc: ', loss_and_metrics[1]
```

For each CNN Model, It took us 45 minutes each with the above AWS configuration to completely train the model with a mini-batch size of 128 and # epochs of 50 and save the weights afterwards.

Testing:

Once the model has been trained, we focused on how to test the trained model on real time video feed. Testing phase was crucial for us due to the fact that our model has never seen this data and has to perform on it. To feed testing data to our model, we have written a script in OpenCV and Python which will extract frame for each second from a live video feed. After extracting frame for each second, we are using Haar-Cascade face detection classifier to detect faces in each frame and then we are feeding this face to our trained model for classifying different type of Emotions. Below we have described in details on how we are detecting faces using Haar-Cascade classifier using OpenCV.

FACE DETECTION CLASSIFIERS:

A computer program that decides whether an image is a positive image (face image) or negative image (non-face image) is called a **classifier**. A classifier is trained on hundreds of thousands of face and non-face images to learn how to classify a new image correctly. OpenCV provides us with two pre-trained and ready to be used for face detection classifiers :

i) Haar-Cascade Classifier
ii) LBP Classifier

Four our project we have used Haar-Cascade Classifier for detecting faces. As this is a pre-trained classifier in OpenCV so their learned knowledge files also come bundled with OpenCV (haarcascade_frontalface_alt.xml). The Haar classifier is a machine learning based approach, an algorithm created by Paul Viola and Michael Jones; which was trained on many positive images (with faces) and negatives images (without faces).

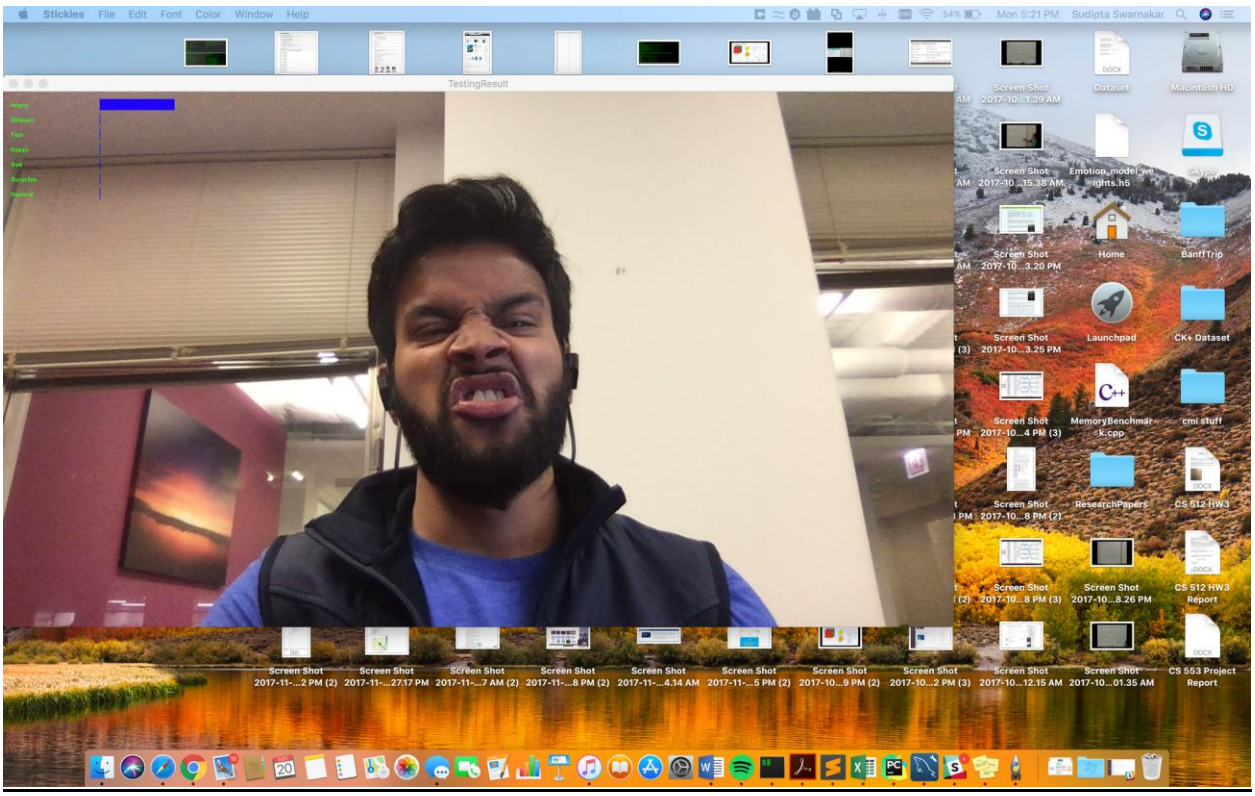
Source Code for Testing the Trained Model on Live Video Feed:

```
import cv2
import keras
from scipy.misc import imresize
import numpy as np


EMOTIONS =['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise','Neutral']

cascadeClassifier = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
height = width = 20


def detect_face(image):
  faces = cascadeClassifier.detectMultiScale(image, 1.3, 5)
  for (x,y,w,h) in faces:
    image = image[y:y+h, x:x+w]
    return True,image
  return False,image


cap = cv2.VideoCapture(0)

modelWeights = keras.models.load_model('emotion_weightsGoogleNet.h5')

while cap.isOpened():
   ret, img = cap.read()
   if ret:
```

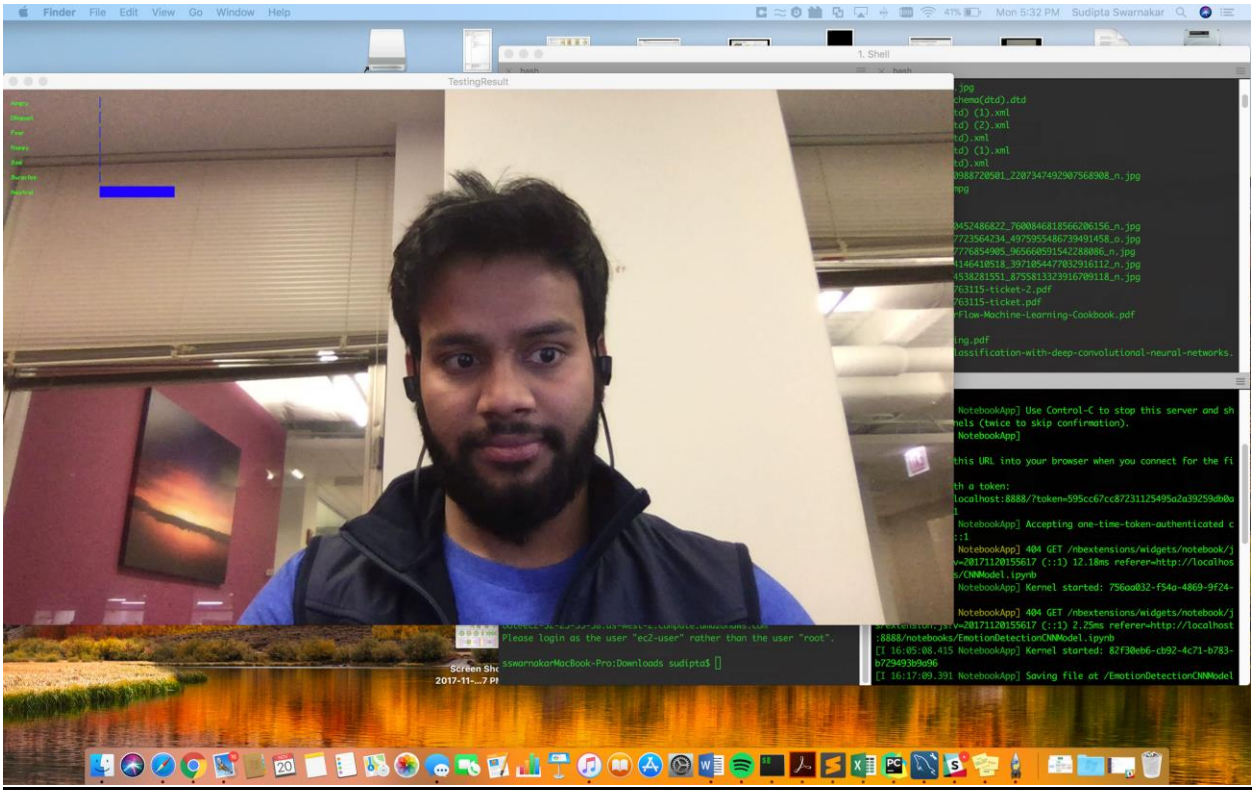
```
    x=[]
    gray = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
    flag,face = detect_face(gray)
    if flag:
      gray = imresize(face, [height, width], 'bilinear')
      gray = np.dstack((gray,) * 3)
      x.append(gray)
      x = np.asarray(x)
      print x.shape
      result=modelWeights.predict( x, batch_size=8, verbose=0)
      for index,emotion in enumerate(EMOTIONS):
        cv2.putText(img, emotion, (10,index * 20 + 20), cv2.FONT_HERSHEY_PLAIN, 0.5, (0, 255, 0), 1);
        cv2.rectangle(img, (130, index * 20 + 10), (130 + int(result[0][index] * 100), (index + 1) * 20 + 4), (255, 0, 0), -1)
        print(result)

    cv2.imshow('TestingResult',img)
    if cv2.waitKey(1) &0xff==ord('q'):
      break
else:
  print "Can't open camera"

cv2.destroyAllWindows()
```

Output Result:

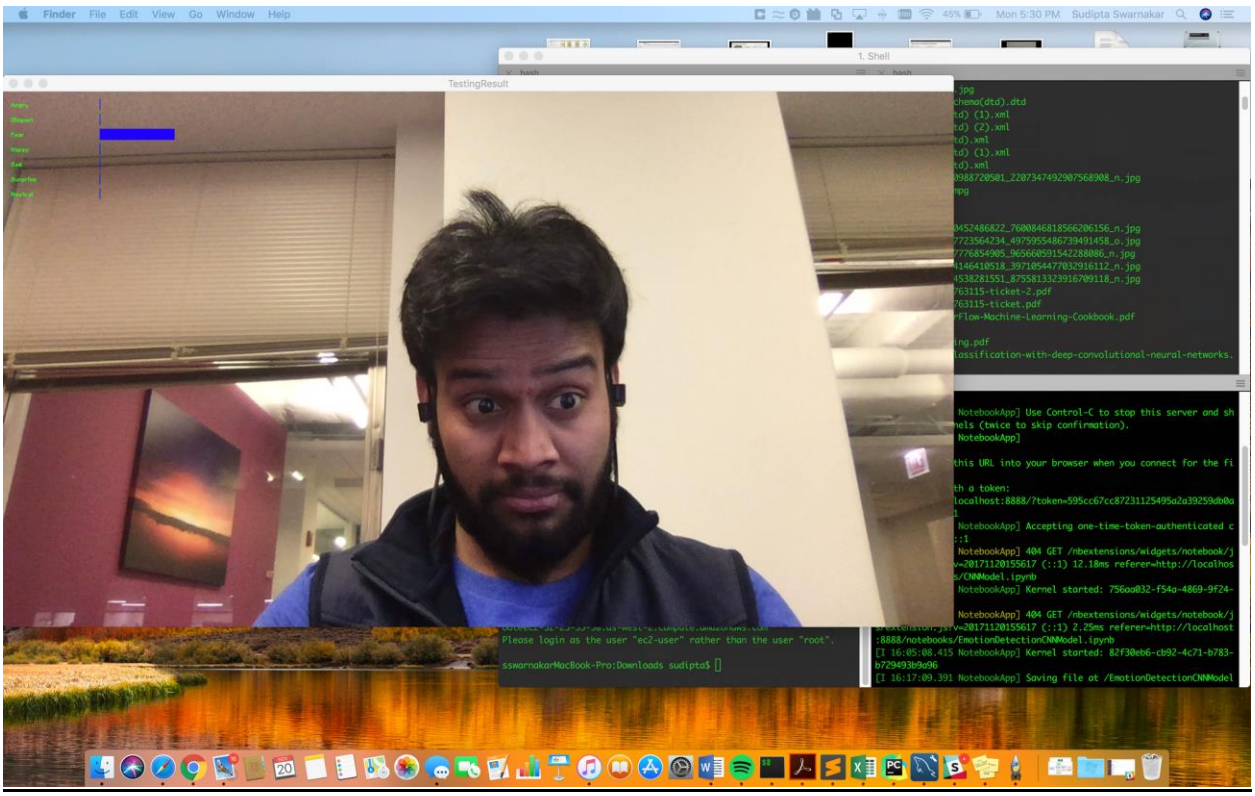
ANGRY



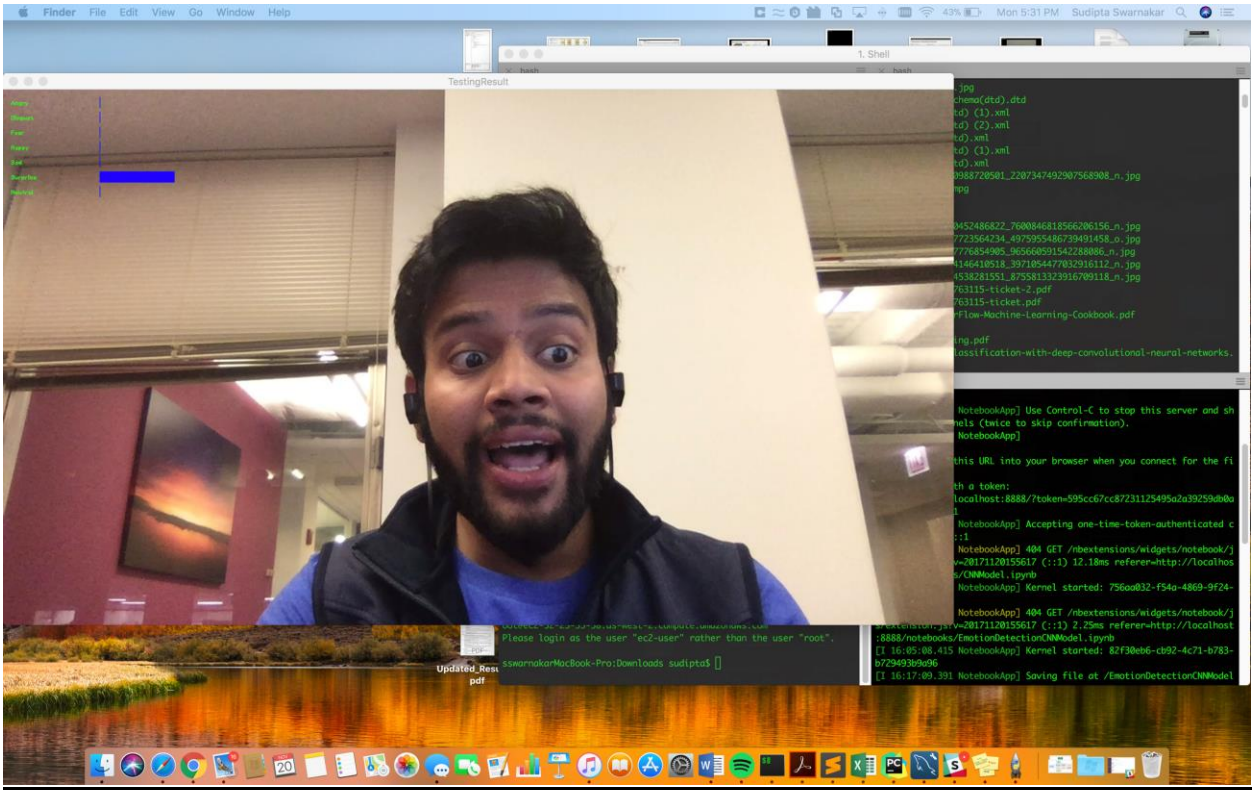
Neutral:



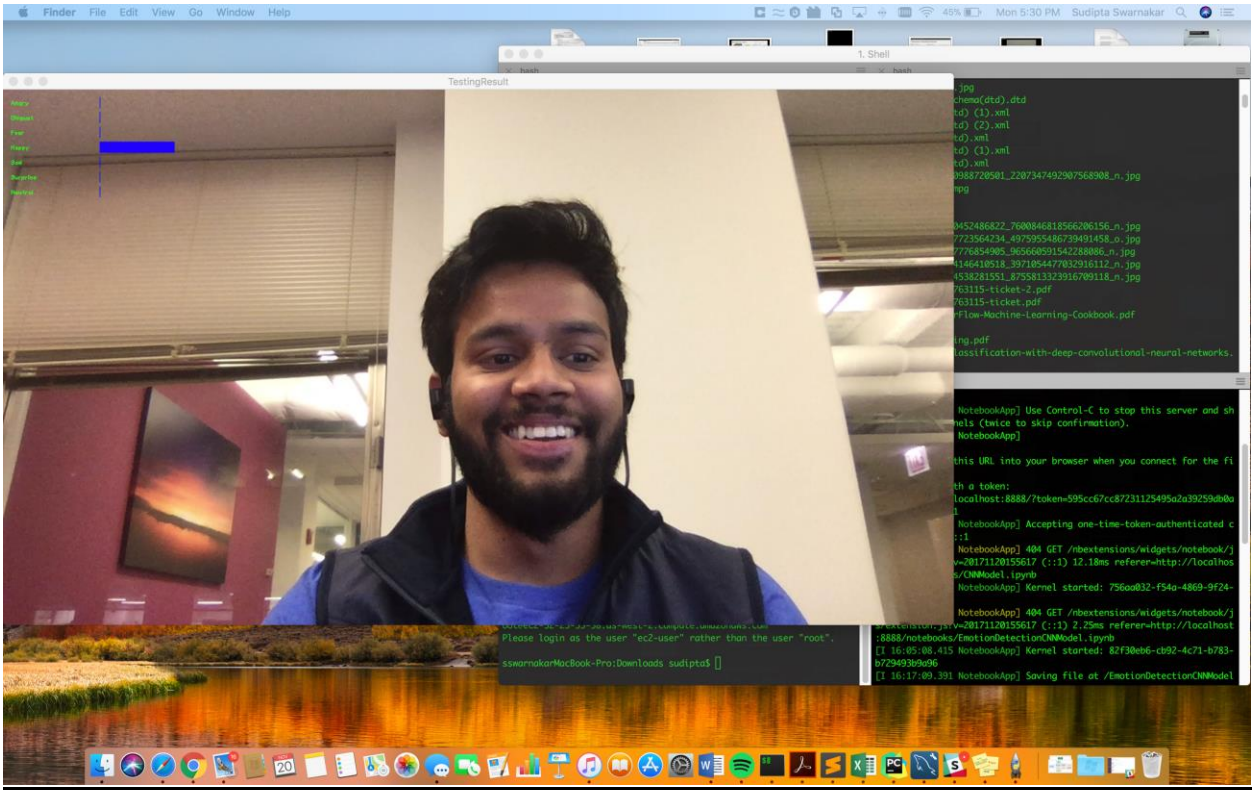
Fear:



Surprise:



Happy:



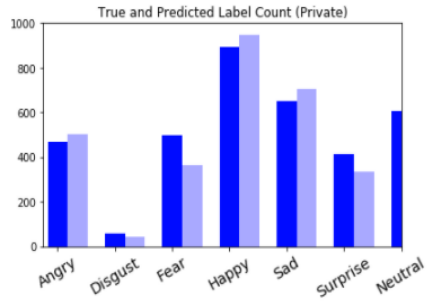
Performance:

VGGNet Performance

In [19]:
```
# prediction and true labels
y_prob = model.predict(x_test, batch_size=32, verbose=0)
y_pred = [np.argmax(prob) for prob in y_prob]
y_true = [np.argmax(true) for true in y_test]
```

In [20]:
```
labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise','Neutral']
```

In [21]:
```
def plot_distribution(y_true, y_pred):
    ind = np.arange(1.5,8,1)
    width = 0.35
    fig, ax = plt.subplots()
    true = ax.bar(ind, np.bincount(y_true), width, color="blue", alpha=1.0)
    pred = ax.bar(ind + width, np.bincount(y_pred), width, color="blue", alpha=0.3)
    ax.set_xticks(np.arange(1.5,8,1))
    ax.set_xticklabels(labels, rotation=30, fontsize=14)
    ax.set_xlim([1.25, 7.5])
    ax.set_ylim([0, 1000])
    ax.set_title('True and Predicted Label Count (Private)')
    plt.tight_layout()
    plt.show()

plot_distribution(y_true, y_pred)
```

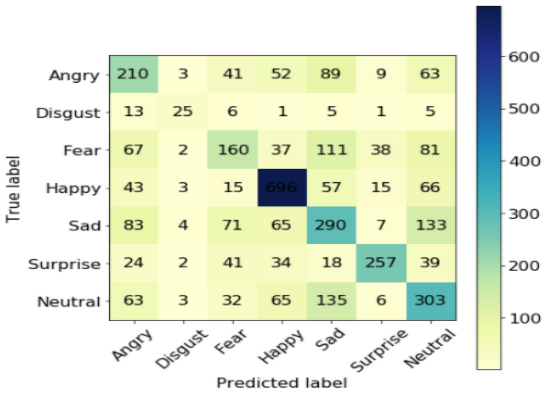


True Vs. Predicted Count on Validation Set

Confusion Matrix

In [20]:
```
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(y_true, y_pred, cmap=plt.cm.Blues):
    cm = confusion_matrix(y_true, y_pred)
    fig = plt.figure(figsize=(7,7))
    matplotlib.rcParams.update({'font.size': 16})
    ax  = fig.add_subplot(111)
    matrix = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    fig.colorbar(matrix)
    for i in range(0,7):
        for j in range(0,7):
            ax.text(j,i,cm[i,j],va='center', ha='center')
    # ax.set_title('Confusion Matrix')
    ticks = np.arange(len(labels))
    ax.set_xticks(ticks)
    ax.set_xticklabels(labels, rotation=45)
    ax.set_yticks(ticks)
    ax.set_yticklabels(labels)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

In [21]:
```
plot_confusion_matrix(y_true, y_pred, cmap=plt.cm.YlGnBu)
```



Classification Metrics

In [22]:
```
def class_precision(y_true, y_pred, emotion):
    cm = confusion_matrix(y_true, y_pred)
    i = [i for i, label in enumerate(labels) if label == emotion][0]
    col = [cm[j,i] for j in range(0,7)]
    return float(col[i])/sum(col)

def class_recall(y_true, y_pred, emotion):
    cm = confusion_matrix(y_true, y_pred)
    i = [i for i, label in enumerate(labels) if label == emotion][0]
    row = [cm[i,j] for j in range(0,7)]
    return float(row[i])/sum(row)

def class_accuracy(y_true, y_pred, emotion):
    cm = confusion_matrix(y_true, y_pred)
    i = [i for i, label in enumerate(labels) if label == emotion][0]
    tp = cm[i,i]
    fn = sum([cm[i,j] for j in range(0,7) if j != i])
    fp = sum([cm[j,i] for j in range(0,7) if j != i])
    tn = sum([cm[i,j] for j in range(0,7) for i in range(0,7)]) -(tp+fp+fn)
    return float(tp + tn)/sum([tp, fn, fp, tn])
```

In [24]:
```
# private test set
for emotion in labels:
    print emotion.upper()
    print '    accuracy = {}'.format(class_accuracy(y_true, y_pred, emotion))
    print '   preision = {}'.format(class_precision(y_true, y_pred, emotion))
    print 'recall = {}\n'.format(class_recall(y_true, y_pred, emotion))
```

```
ANGRY
    accuracy = 0.846753970465
   preision = 0.417495029821
recall = 0.449678800857

DISGUST
    accuracy = 0.986625801059
   preision = 0.595238095238
recall = 0.446428571429

FEAR
    accuracy = 0.848983003622
   preision = 0.437158469945
recall = 0.322580645161

HAPPY
    accuracy = 0.873780997492
   preision = 0.732631578947
recall = 0.777653631285

SAD
    accuracy = 0.783226525495
   preision = 0.41134751773
recall = 0.444104134763

SURPRISE
    accuracy = 0.934800780162
   preision = 0.771771771772
recall = 0.619277108434

NEUTRAL
    accuracy = 0.807467261076
   preision = 0.439130434783
recall = 0.499176276771
```

In [26]:
```
from sklearn.metrics import classification_report
print classification_report(y_true, y_pred, target_names=labels)
```

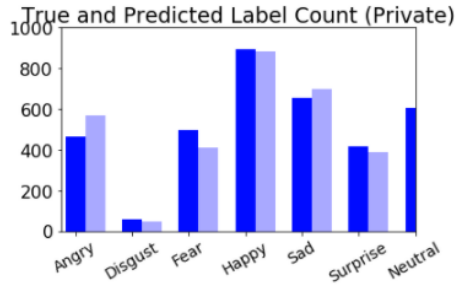
```
             precision    recall  f1-score   support

      Angry       0.42      0.45      0.43       467
    Disgust       0.60      0.45      0.51        56
       Fear       0.44      0.32      0.37       496
      Happy       0.73      0.78      0.75       895
        Sad       0.41      0.44      0.43       653
   Surprise       0.77      0.62      0.69       415
    Neutral       0.44      0.50      0.47       607

avg / total       0.55      0.54      0.54      3589
```

GoogleNet Performance:

In [29]:
```
# prediction and true labels
y_prob = model.predict(x_test, batch_size=32, verbose=0)
y_pred = [np.argmax(prob) for prob in y_prob]
y_true = [np.argmax(true) for true in y_test]
```

In [30]:
```
labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise','Neutral']
```

In [31]:
```
def plot_distribution(y_true, y_pred):
    ind = np.arange(1.5,8,1)
    width = 0.35
    fig, ax = plt.subplots()
    true = ax.bar(ind, np.bincount(y_true), width, color="blue", alpha=1.0)
    pred = ax.bar(ind + width, np.bincount(y_pred), width, color="blue", alpha=0.3)
    ax.set_xticks(np.arange(1.5,8,1))
    ax.set_xticklabels(labels, rotation=30, fontsize=14)
    ax.set_xlim([1.25, 7.5])
    ax.set_ylim([0, 1000])
    ax.set_title('True and Predicted Label Count (Private)')
    plt.tight_layout()
    plt.show()

plot_distribution(y_true, y_pred)
```

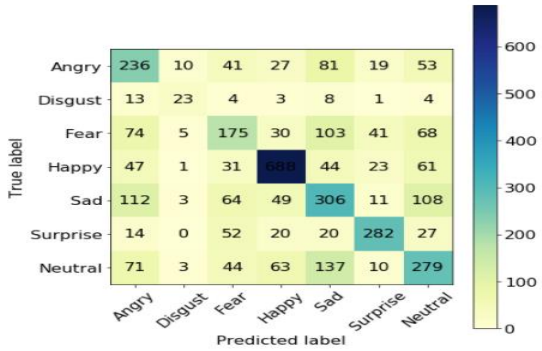


True Vs. Predicted Count on Validation Set

Confusion Matrix

In [32]:
```
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(y_true, y_pred, cmap=plt.cm.Blues):
    cm = confusion_matrix(y_true, y_pred)
    fig = plt.figure(figsize=(7,7))
    matplotlib.rcParams.update({'font.size': 16})
    ax  = fig.add_subplot(111)
    matrix = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    fig.colorbar(matrix)
    for i in range(0,7):
        for j in range(0,7):
            ax.text(j,i,cm[i,j],va='center', ha='center')
    # ax.set_title('Confusion Matrix')
    ticks = np.arange(len(labels))
    ax.set_xticks(ticks)
    ax.set_xticklabels(labels, rotation=45)
    ax.set_yticks(ticks)
    ax.set_yticklabels(labels)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

In [33]:
```
plot_confusion_matrix(y_true, y_pred, cmap=plt.cm.YlGnBu)
```



Classification Metrics

In [34]:
```
def class_precision(y_true, y_pred, emotion):
    cm = confusion_matrix(y_true, y_pred)
    i = [i for i, label in enumerate(labels) if label == emotion][0]
    col = [cm[j,i] for j in range(0,7)]
    return float(col[i])/sum(col)

def class_recall(y_true, y_pred, emotion):
    cm = confusion_matrix(y_true, y_pred)
    i = [i for i, label in enumerate(labels) if label == emotion][0]
    row = [cm[i,j] for j in range(0,7)]
    return float(row[i])/sum(row)

def class_accuracy(y_true, y_pred, emotion):
    cm = confusion_matrix(y_true, y_pred)
    i = [i for i, label in enumerate(labels) if label == emotion][0]
    tp = cm[i,i]
    fn = sum([cm[i,j] for j in range(0,7) if j != i])
    fp = sum([cm[j,i] for j in range(0,7) if j != i])
    tn = sum([cm[i,j] for j in range(0,7) for i in range(0,7)]) -(tp+fp+fn)
    return float(tp + tn)/sum([tp, fn, fp, tn])
```

In [35]:
```
# private test set
for emotion in labels:
    print emotion.upper()
    print '    accuracy = {}'.format(class_accuracy(y_true, y_pred, emotion))
    print '   preision = {}'.format(class_precision(y_true, y_pred, emotion))
    print 'recall = {}\n'.format(class_recall(y_true, y_pred, emotion))
```

```
ANGRY
    accuracy = 0.84341042073
   preision = 0.416225749559
recall = 0.505353319058

DISGUST
    accuracy = 0.984675397047
   preision = 0.511111111111
recall = 0.410714285714

FEAR
    accuracy = 0.844803566453
   preision = 0.425790754258
recall = 0.352822580645

HAPPY
    accuracy = 0.888826971301
   preision = 0.781818181818
recall = 0.768715083799

SAD
    accuracy = 0.79381443299
   preision = 0.437768240343
recall = 0.468606431853

SURPRISE
    accuracy = 0.933686263583
   preision = 0.728682170543
recall = 0.679518072289

NEUTRAL
    accuracy = 0.819169685149
   preision = 0.465
recall = 0.459637561779
```

```
from sklearn.metrics import classification_report
print classification_report(y_true, y_pred, target_names=labels)
```

```
             precision    recall  f1-score   support

      Angry       0.42      0.51      0.46       467
    Disgust       0.51      0.41      0.46        56
       Fear       0.43      0.35      0.39       496
      Happy       0.78      0.77      0.78       895
        Sad       0.44      0.47      0.45       653
   Surprise       0.73      0.68      0.70       415
    Neutral       0.47      0.46      0.46       607

avg / total       0.56      0.55      0.56      3589
```

Comparison - VGGNet Vs. GoogleNet:

So from the above results of trained VGGNet and GoogleNet models, it's clear that Precision and Recall are better for GoogleNet over VGGNet. Although we have used more complex architecture for VGGNet but simple GoogleNet is performing better for out dataset.

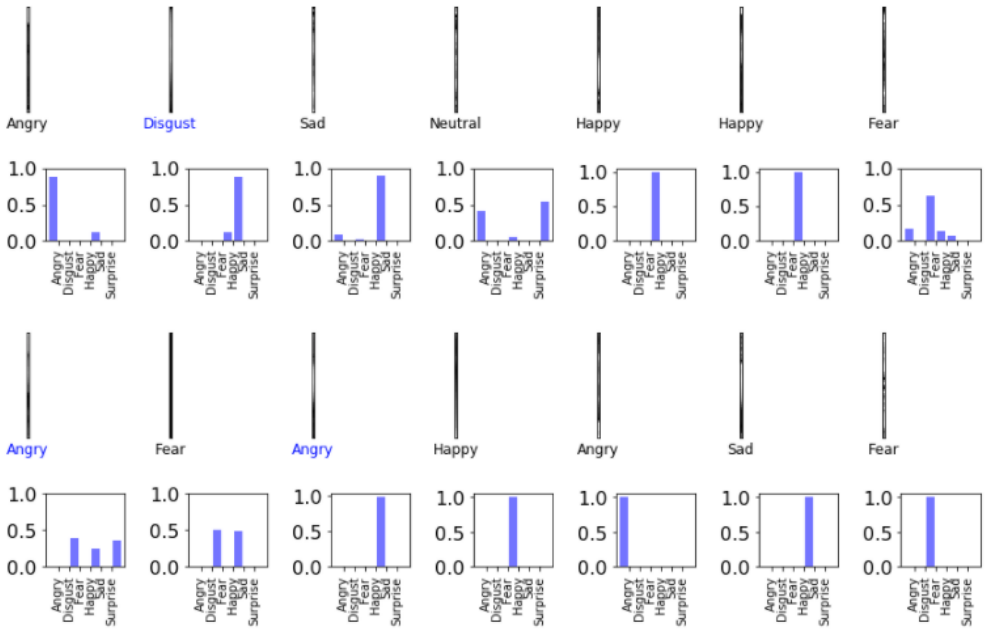
Model Performance on Validation Set

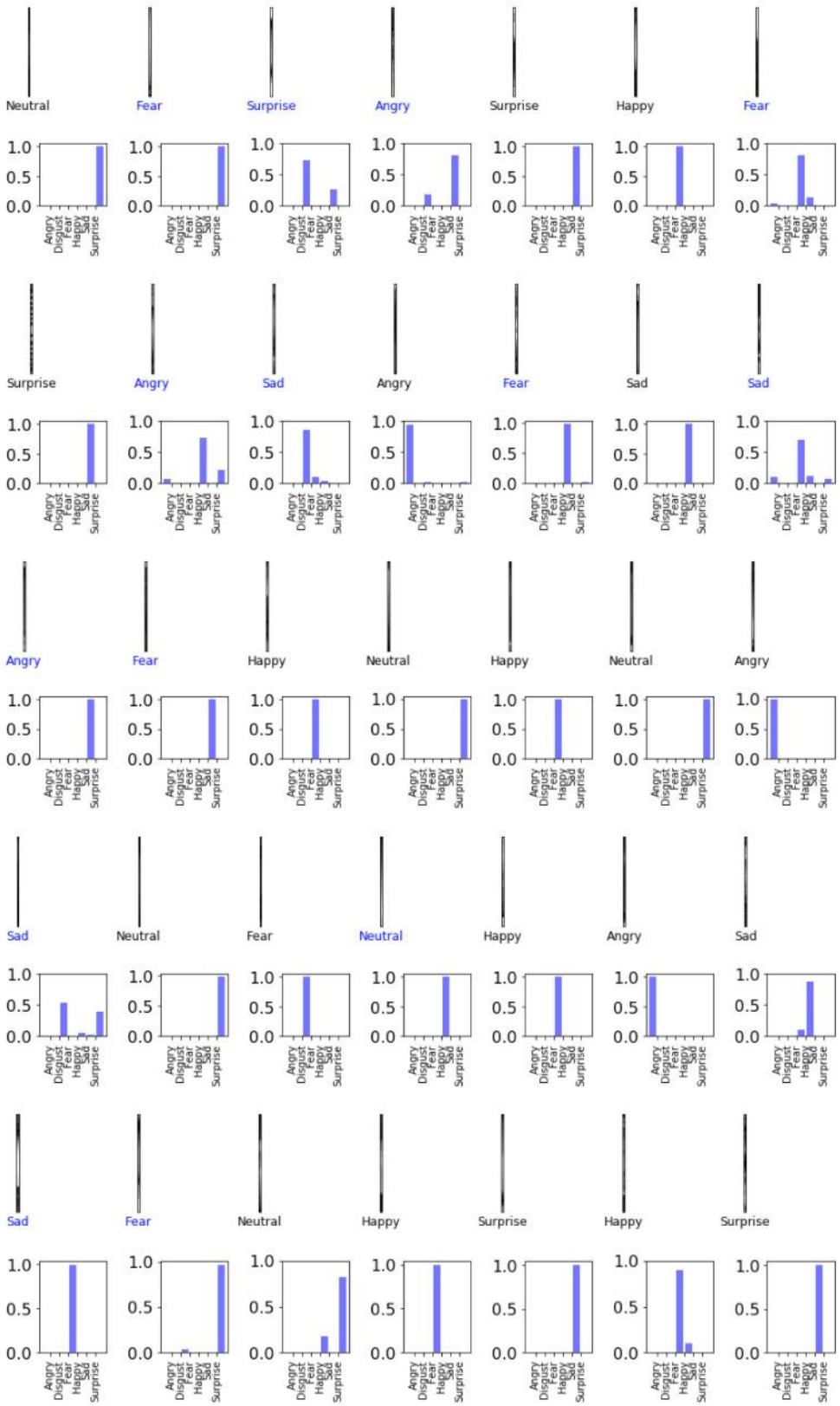
In [27]:
```
def plot_subjects(start, end, y_pred, y_true, title=False):
    fig = plt.figure(figsize=(12,12))
    emotion = {0:'Angry', 1:'Disgust', 2:'Fear', 3:'Happy', 4:'Sad', 5:'Surprise',6:'Neutral'}
    for i in range(start, end+1):
        input_img = x_test[i:(i+1),:,:,:]
        ax = fig.add_subplot(7,7,i+1)
        ax.imshow(input_img[0,0,:,:], cmap=matplotlib.cm.gray)
        plt.xticks(np.array([]))
        plt.yticks(np.array([]))
        if y_pred[i] != y_true[i]:
            plt.xlabel(emotion[y_true[i]], color='blue',fontsize=12)
        else:
            plt.xlabel(emotion[y_true[i]], fontsize=12)
        if title:
            plt.title(emotion[y_pred[i]], color='blue')
        plt.tight_layout()
    plt.show()
```

In [28]:
```
def plot_probs(start,end, y_prob):
    fig = plt.figure(figsize=(12,12))
    for i in range(start, end+1):
        input_img = x_test[i:(i+1),:,:,:]
        ax = fig.add_subplot(7,7,i+1)
        ax.bar(np.arange(0,7), y_prob[i], color='blue',alpha=0.5)
        ax.set_xticks(np.arange(0.5,6.5,1))
        labels =['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise','Neutral']
        ax.set_xticklabels(labels, rotation=90, fontsize=10)
        ax.set_yticks(np.arange(0.0,1.1,0.5))
        plt.tight_layout()
    plt.show()
```

In [29]:
```
def plot_subjects_with_probs(start, end, y_prob):
    iter = (end - start)/7
    for i in np.arange(0,iter):
        plot_subjects(i*7,(i+1)*7-1, y_pred, y_true, title=False)
        plot_probs(i*7,(i+1)*7-1, y_prob)
```

In [30]:
```
plot_subjects_with_probs(0,49, y_prob)
```




Neutral
Fear
Surprise
Angry
Surprise
Happy
Fear
Surprise
Angry
Sad
Angry
Fear
Sad
Sad
Angry
Fear
Happy
Neutral
Happy
Neutral
Angry
Sad
Neutral
Fear
Neutral
Happy
Angry
Sad
Sad
Fear
Neutral
Happy
Surprise
Happy
Surprise
1.0
0.5
0.0
Angry
Disgust
Fear
Happy
Sad
Surprise

Technical Prerequisite:

- GNU/Linux host capable of running Keras and Tensorflow (CentOS 7 suggested)
- Nvidia GPU supporting CUDA 8 suggested
- Python 2.7 Suggested
- OpenCV

Execution:

For VGGNet Model Training Please Execute (on CUDA 8 enabled Cluster),

python EmotionDetectionCNNModel.py

For GoogleNet Model Training Please Execute (on CUDA 8 enabled Cluster),

python GoogleNet.py

For Testing on Live Video Feed Please Execute,

python testing.py

Conclusion:

With the help of CNN using deep learning, we trained our model with VGGNet and GoogleNet for same dataset(FER2013). This dataset has pre-cropped images. We got validation accuracy of 60.55% for GoogleNet which is higher than VGGNet one(54.2%).
Finally tested the live video feed with the GoogleNet trained model and got accurate results. We used Haar-Cascade Classifier to detect the faces in live video feed, it's OpenCV provided.

Application:

Through facial emotion recognition, we are able to measure the effects that content and services have on the audience/users through an easy and low-cost procedure. For example, retailers may use these metrics to evaluate customer interest. Healthcare providers can provide better service by using additional information about patients' emotional state during treatment. Entertainment producers can monitor audience engagement in events to consistently create desired content.

Future Recommendations

1. Test with more subjects and expressions.
2. Data for Disgust was very less in comparison to other expressions, it can be merged with sad or angry to get more accurate results.