In [1]:
```
# Importing Necessary Libraries
import tensorflow as tf
from tensorflow.keras import models
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout, BatchNormalization
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
```

In [2]:
```
# Data Preprocessing
data = pd.read_csv('/Volumes/Samsung/Datasets/state-farm-distracted-driver-detection/driver_imgs_list.csv')
img = cv2.imread('/Volumes/Samsung/Datasets/state-farm-distracted-driver-detection/imgs/train/c0/img_327.jpg')


def get_im_cv2(path, img_rows, img_cols):
    img = cv2.imread(path, 0)
    resized = cv2.resize(img, (img_cols, img_rows))
    return resized


X = data[['img']]
y = data[['classname']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)

image_train_arr = []
for i in range(len(X_train)):
    path = "/Volumes/Samsung/Datasets/state-farm-distracted-driver-detection/imgs/train/{}/{}".format(
        y_train.iloc[i, 0],
        X_train.iloc[i, 0])
    resized = get_im_cv2(path, 32, 32)
    image_train_arr.append(resized)

image_test_arr = []
for i in range(len(X_test)):
    path = "/Volumes/Samsung/Datasets/state-farm-distracted-driver-detection/imgs/train/{}/{}".format(y_test.iloc[i, 0],
                                                                                                      X_test.iloc[i, 0])
    resized = get_im_cv2(path, 32, 32)
    image_test_arr.append(resized)

image_train_arr = np.array(image_train_arr)
image_test_arr = np.array(image_test_arr)

image_train_arr = image_train_arr / 255
image_test_arr = image_test_arr / 255
```

In [3]:
```
# Converting Classname from c0, c1 ... c9 to 0, 1 .... 9
labelencoder = LabelEncoder()
y_train['new-col'] = labelencoder.fit_transform(y_train)
y_test['new-col'] = labelencoder.fit_transform(y_test)
```

```
/Users/sagarkav/Library/Python/3.8/lib/python/site-packages/sklearn/preprocessing/_label.py:115: DataConversionWarning: A column-vector y was passed when a 1d array w
as expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [4]:
```
# CNN Layers
model = models.Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 1)))
model.add(BatchNormalization())
model.add(MaxPool2D(2, 2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(2, 2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(10))
model.summary()
```

```
2021-11-17 00:26:37.594338: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN)
to use the following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        320

 batch_normalization (BatchN  (None, 30, 30, 32)       128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 batch_normalization_1 (Batc  (None, 13, 13, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

 batch_normalization_2 (Batc  (None, 4, 4, 64)         256
 hNormalization)

 dropout (Dropout)           (None, 4, 4, 64)          0

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 batch_normalization_3 (Batc  (None, 64)               256
 hNormalization)

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 122,890
Trainable params: 122,442
Non-trainable params: 448
_________________________________________________________________
```

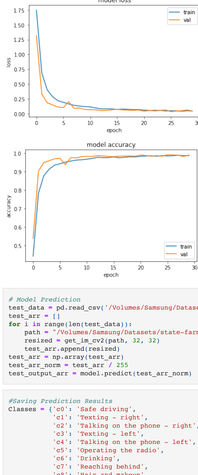
In [5]:
```
# Training Model
model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
history = model.fit(image_train_arr, y_train['new-col'], epochs=30, validation_data=(image_test_arr, y_test['new-col']))
print('\nModel evaluation after 30 epochs: ', model.evaluate(image_test_arr, y_test['new-col'], verbose=2))
```

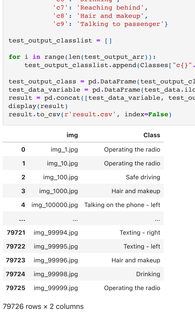
```
Epoch 1/30
470/470 [==============================] - 29s 60ms/step - loss: 1.7467 - accuracy: 0.4410 - val_loss: 1.3149 - val_accuracy: 0.5368
Epoch 2/30
470/470 [==============================] - 28s 60ms/step - loss: 0.6843 - accuracy: 0.7825 - val_loss: 0.3298 - val_accuracy: 0.9015
Epoch 3/30
470/470 [==============================] - 30s 64ms/step - loss: 0.4022 - accuracy: 0.8775 - val_loss: 0.1832 - val_accuracy: 0.9482
Epoch 4/30
470/470 [==============================] - 27s 57ms/step - loss: 0.2858 - accuracy: 0.9125 - val_loss: 0.1463 - val_accuracy: 0.9581
Epoch 5/30
470/470 [==============================] - 24s 51ms/step - loss: 0.2184 - accuracy: 0.9340 - val_loss: 0.1110 - val_accuracy: 0.9686
Epoch 6/30
470/470 [==============================] - 24s 51ms/step - loss: 0.1899 - accuracy: 0.9420 - val_loss: 0.1015 - val_accuracy: 0.9714
Epoch 7/30
470/470 [==============================] - 24s 51ms/step - loss: 0.1631 - accuracy: 0.9503 - val_loss: 0.2016 - val_accuracy: 0.9369
Epoch 8/30
470/470 [==============================] - 26s 56ms/step - loss: 0.1400 - accuracy: 0.9561 - val_loss: 0.0873 - val_accuracy: 0.9747
Epoch 9/30
470/470 [==============================] - 25s 52ms/step - loss: 0.1278 - accuracy: 0.9614 - val_loss: 0.0959 - val_accuracy: 0.9732
Epoch 10/30
470/470 [==============================] - 25s 54ms/step - loss: 0.1184 - accuracy: 0.9633 - val_loss: 0.0702 - val_accuracy: 0.9803
Epoch 11/30
470/470 [==============================] - 28s 59ms/step - loss: 0.1132 - accuracy: 0.9644 - val_loss: 0.0649 - val_accuracy: 0.9818
Epoch 12/30
470/470 [==============================] - 27s 58ms/step - loss: 0.0963 - accuracy: 0.9705 - val_loss: 0.0652 - val_accuracy: 0.9811
Epoch 13/30
470/470 [==============================] - 24s 50ms/step - loss: 0.0797 - accuracy: 0.9767 - val_loss: 0.0515 - val_accuracy: 0.9858
Epoch 14/30
470/470 [==============================] - 24s 51ms/step - loss: 0.0809 - accuracy: 0.9748 - val_loss: 0.0538 - val_accuracy: 0.9841
Epoch 15/30
470/470 [==============================] - 23s 50ms/step - loss: 0.0800 - accuracy: 0.9753 - val_loss: 0.0644 - val_accuracy: 0.9811
Epoch 16/30
470/470 [==============================] - 22s 48ms/step - loss: 0.0697 - accuracy: 0.9788 - val_loss: 0.0732 - val_accuracy: 0.9803
Epoch 17/30
470/470 [==============================] - 22s 48ms/step - loss: 0.0776 - accuracy: 0.9743 - val_loss: 0.0623 - val_accuracy: 0.9823
Epoch 18/30
470/470 [==============================] - 22s 48ms/step - loss: 0.0722 - accuracy: 0.9766 - val_loss: 0.0580 - val_accuracy: 0.9835
Epoch 19/30
470/470 [==============================] - 23s 48ms/step - loss: 0.0675 - accuracy: 0.9796 - val_loss: 0.0548 - val_accuracy: 0.9842
Epoch 20/30
470/470 [==============================] - 23s 48ms/step - loss: 0.0686 - accuracy: 0.9788 - val_loss: 0.0613 - val_accuracy: 0.9824
Epoch 21/30
470/470 [==============================] - 22s 48ms/step - loss: 0.0581 - accuracy: 0.9820 - val_loss: 0.0410 - val_accuracy: 0.9889
Epoch 22/30
470/470 [==============================] - 22s 48ms/step - loss: 0.0544 - accuracy: 0.9836 - val_loss: 0.0523 - val_accuracy: 0.9861
Epoch 23/30
470/470 [==============================] - 23s 48ms/step - loss: 0.0468 - accuracy: 0.9848 - val_loss: 0.0581 - val_accuracy: 0.9827
Epoch 24/30
470/470 [==============================] - 22s 48ms/step - loss: 0.0546 - accuracy: 0.9823 - val_loss: 0.0538 - val_accuracy: 0.9854
Epoch 25/30
470/470 [==============================] - 23s 49ms/step - loss: 0.0445 - accuracy: 0.9859 - val_loss: 0.0618 - val_accuracy: 0.9827
Epoch 26/30
470/470 [==============================] - 22s 48ms/step - loss: 0.0481 - accuracy: 0.9837 - val_loss: 0.0336 - val_accuracy: 0.9904
Epoch 27/30
470/470 [==============================] - 22s 47ms/step - loss: 0.0466 - accuracy: 0.9851 - val_loss: 0.0432 - val_accuracy: 0.9896
Epoch 28/30
470/470 [==============================] - 21s 46ms/step - loss: 0.0410 - accuracy: 0.9876 - val_loss: 0.0481 - val_accuracy: 0.9878
Epoch 29/30
470/470 [==============================] - 25s 53ms/step - loss: 0.0474 - accuracy: 0.9855 - val_loss: 0.0613 - val_accuracy: 0.9812
Epoch 30/30
470/470 [==============================] - 25s 53ms/step - loss: 0.0438 - accuracy: 0.9860 - val_loss: 0.0436 - val_accuracy: 0.9888
232/232 - 3s - loss: 0.0436 - accuracy: 0.9888 - 3s/epoch - 11ms/step

Model evaluation after 30 epochs:  [0.04355091229081154, 0.9887837767601013]
```

In [6]:
```
# Plotting Loss and Accuracy
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='lower right')
plt.show()
```





In [7]:
```
# Model Prediction
test_data = pd.read_csv('/Volumes/Samsung/Datasets/state-farm-distracted-driver-detection/sample_submission.csv')
test_arr = []
for i in range(len(test_data)):
    path = "/Volumes/Samsung/Datasets/state-farm-distracted-driver-detection/imgs/test/{}".format(test_data.iloc[i, 0])
    resized = get_im_cv2(path, 32, 32)
    test_arr.append(resized)
test_arr = np.array(test_arr)
test_arr_norm = test_arr / 255
test_output_arr = model.predict(test_arr_norm)
```

In [8]:
```
#Saving Prediction Results
Classes = {'c0': 'Safe driving',
           'c1': 'Texting - right',
           'c2': 'Talking on the phone - right',
           'c3': 'Texting - left',
           'c4': 'Talking on the phone - left',
           'c5': 'Operating the radio',
           'c6': 'Drinking',
           'c7': 'Reaching behind',
           'c8': 'Hair and makeup',
           'c9': 'Talking to passenger'}

test_output_classlist = []

for i in range(len(test_output_arr)):
    test_output_classlist.append(Classes["c{}".format(np.argmax(test_output_arr[i]))])

test_output_class = pd.DataFrame(test_output_classlist, columns=['Class'])
test_data_variable = pd.DataFrame(test_data.iloc[:, 0])
result = pd.concat([test_data_variable, test_output_class], axis=1)
display(result)
result.to_csv(r'result.csv', index=False)
```

	img	Class
0	img_1.jpg	Operating the radio
1	img_10.jpg	Operating the radio
2	img_100.jpg	Safe driving
3	img_1000.jpg	Hair and makeup
4	img_100000.jpg	Talking on the phone - left
...	...	...
79721	img_99994.jpg	Texting - right
79722	img_99995.jpg	Texting - left
79723	img_99996.jpg	Hair and makeup
79724	img_99998.jpg	Drinking
79725	img_99999.jpg	Operating the radio

79726 rows × 2 columns