#LaUISqueQueremos

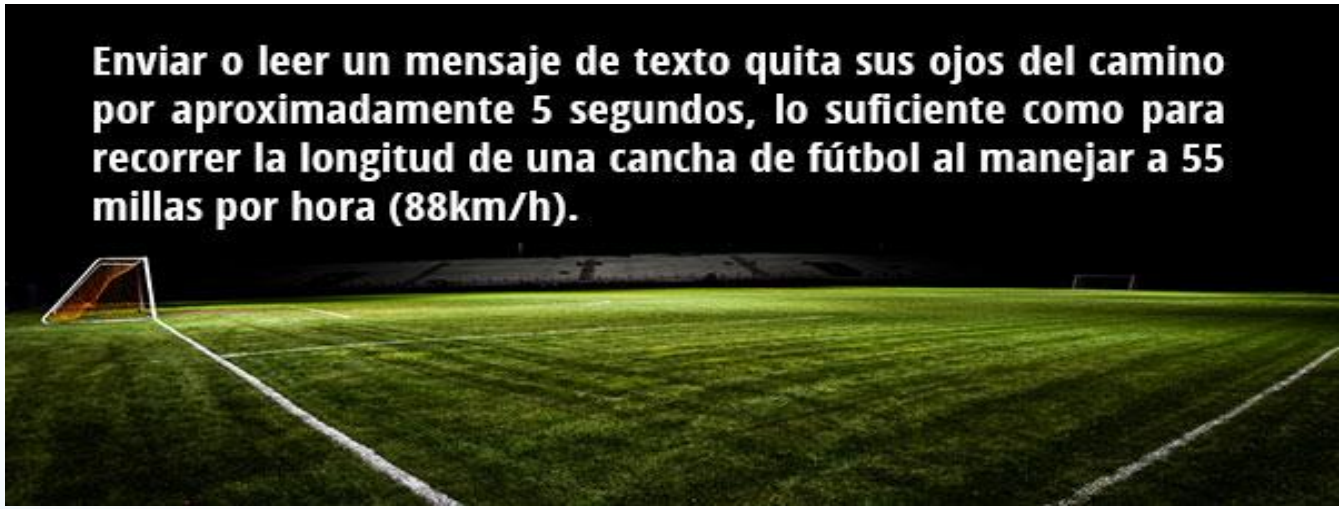
EYES ON THE TRACK

Henry Peña
2150606

Diego Medina
2150011

Motivación:

Prevenir accidentes automovilísticos causados por conductores distraídos, tales como los 85.426 accidentes con heridos y 3.406 con muertos ocurridos en Colombia en 2016. (ANSV 2019)



"una persona al volante en Colombia tiene 4 veces más probabilidades de morir en un accidente de tránsito que un conductor en España o Gran Bretaña" (Revista Motor, 2019).

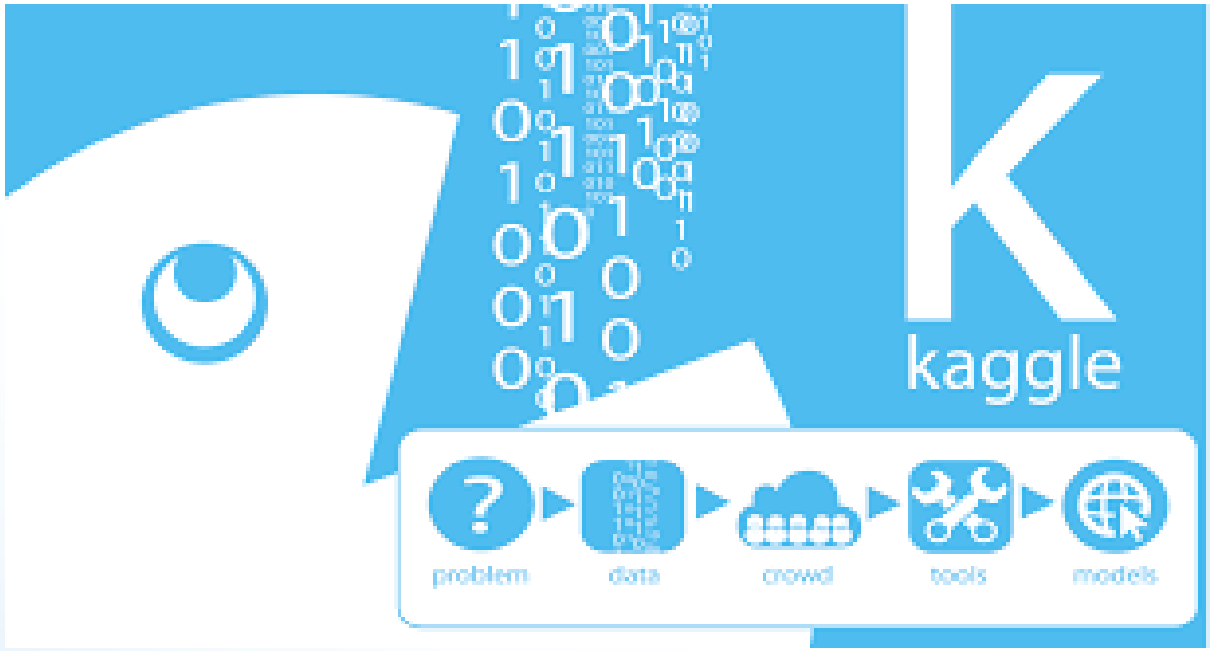
Según **Fesvial, Federación Española de la Seguridad Vial**, el 45% de los accidentes se podrían prevenir si los conductores estuvieran siempre atentos a la carretera.

Dataset:

State Farm Distracted Driver Detection

Can computer vision spot distracted drivers?

$65,000 · 1,440 teams · 3 years ago



El dataset que se usó para este descriptor fue obtenido de un challenge de ***Kaggle*** hecho por la compañía de seguros de automóviles más grande de los Estados Unidos **State Farm Insurance**.

Este dataset contiene 22400 imágenes para entrenamiento de varios conductores en un automóvil, cada cuál tiene asignada una etiqueta de la acción que están realizando (mensajes de texto, comer, hablar por teléfono, maquillaje, alcanzar detrás, etc). Hay 79.700 imágenes sin etiquetar para prueba, de distintos conductores al de entrenamiento.

Las 10 clases a predecir son:

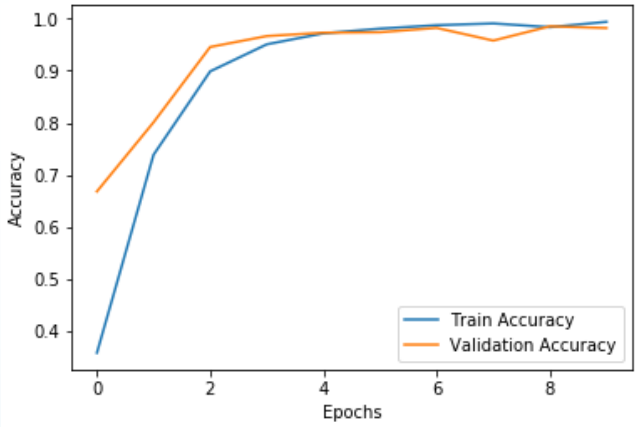
- c0: safe driving
- c1: texting - right
- c2: talking on the phone - right
- c3: texting - left
- c4: talking on the phone - left
- c5: operating the radio
- c6: drinking
- c7: reaching behind
- c8: hair and makeup
- c9: talking to passenger

DNN:

```
model.summary()
```

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 12288)	0
dense_1 (Dense)	(None, 512)	6291968
dense_2 (Dense)	(None, 256)	131328
dense_3 (Dense)	(None, 10)	2570

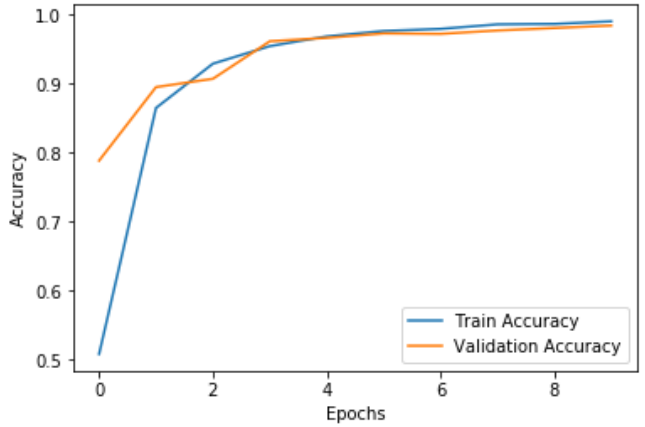
Total params: 6,425,866
Trainable params: 6,425,866
Non-trainable params: 0




```
model5.summary()
```

Layer (type)	Output Shape	Param #
flatten_5 (Flatten)	(None, 12288)	0
dense_8 (Dense)	(None, 10)	122890

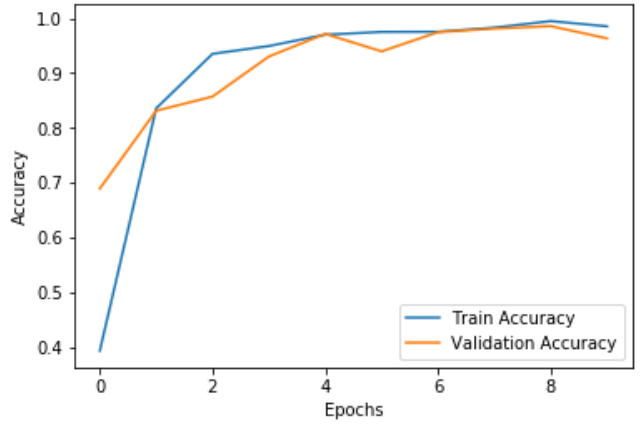
Total params: 122,890
Trainable params: 122,890
Non-trainable params: 0




```
model3.summary()
```

Layer (type)	Output Shape	Param #
flatten_8 (Flatten)	(None, 12288)	0
dense_28 (Dense)	(None, 128)	1572992
dense_29 (Dense)	(None, 256)	33024
dense_30 (Dense)	(None, 512)	131584
dense_31 (Dense)	(None, 1024)	525312

Total params: 2,262,912
Trainable params: 2,262,912
Non-trainable params: 0



BoW:

Data For Dictionary:

```
path_images='../../datasets/driver-detections/train/'
descriptor_extractor = ORB(n_keypoints=200)
array_ORB=[]
size=100 #<1911
for i in tqdm(np.unique(data1["classname"])):
    for j in np.random.choice(data1["img"][data1.classname==i],size):
    #for j in data1["img"][data1.classname==i]:
        temp_image=cv.cvtColor(cv.imread(path_images+i+"/"+j), cv.COLOR_BGR2GRAY)
        descriptor_extractor.detect_and_extract(temp_image)
        if(len(array_ORB)==0):
            array_ORB=descriptor_extractor.descriptors
        else:
            array_ORB=np.vstack((array_ORB,descriptor_extractor.descriptors))
```

```
100%|[redacted]| 10/10 [08:17<00:00, 49.78s/it]
```

Se utilizaron 100 imágenes de cada clase para hacer el diccionario

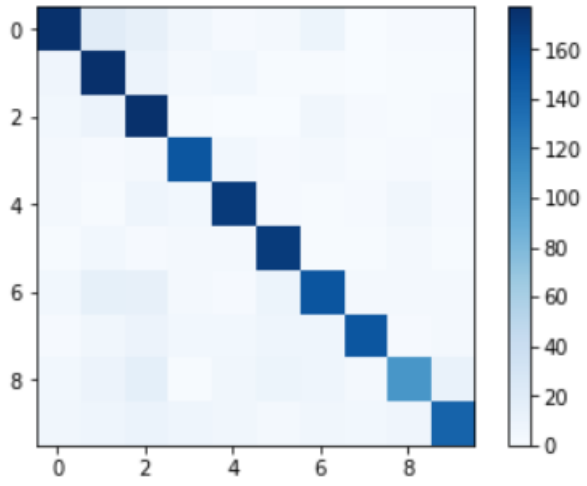
Dictionary Creation:

```
from sklearn.cluster import KMeans
v_words=70
estimator=KMeans(n_clusters=v_words).fit(array_ORB)
dictionary=estimator.cluster_centers_
print dictionary.shape
```

```
(70, 256)
```

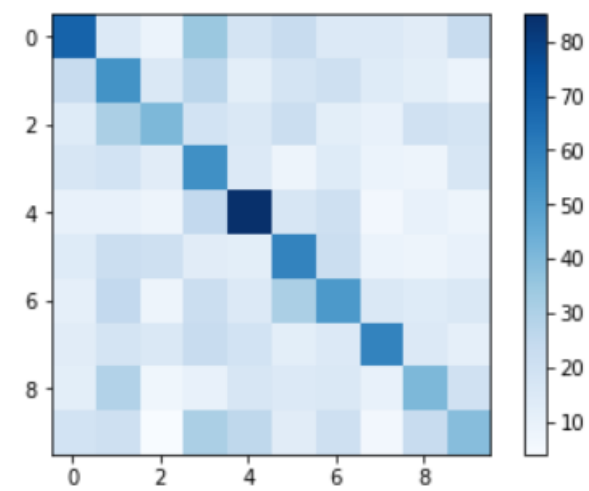
KNN:

0.7855



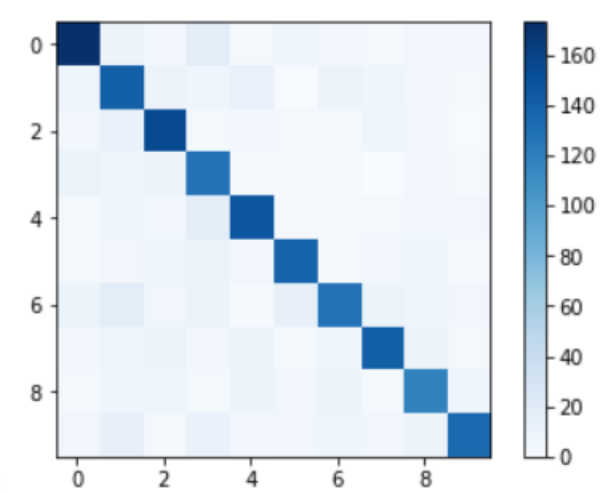
Gaussian:

0.277



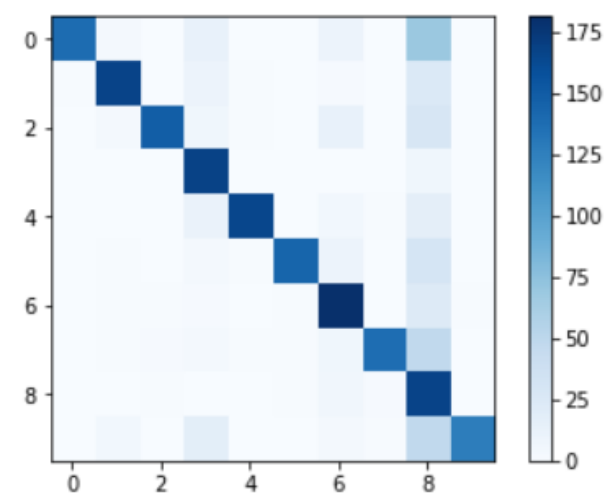
Random Forest:

0.703



SVC (rbf kernel):

0.7725



```
opt = keras.optimizers.SGD(lr=1e-3, momentum=0.9)
.compile(optimizer=opt, loss='sparse_categorical_crossentropy',metrics=['accuracy'])
.fit(X_train,y_train, epochs=10, batch_size=60, verbose=2,validation_data=(X_test, y_test))
```

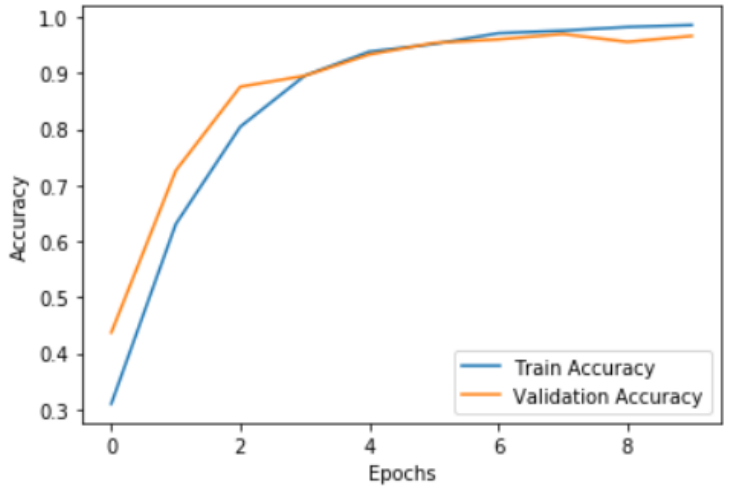
CNN:

Model 1:

```
model_CNN =  keras.models.Sequential()
model_CNN.add(keras.layers.Conv2D(20, (5, 5), activation='relu',
model_CNN.add(keras.layers.MaxPooling2D((2, 2)))
model_CNN.add(keras.layers.Conv2D(50, (5, 5), activation='relu',
model_CNN.add(keras.layers.MaxPooling2D((2, 2)))
model_CNN.add(keras.layers.Flatten())
model_CNN.add(keras.layers.Dense(500, activation='relu', kernel_i
model_CNN.add(keras.layers.Dense(10, activation='softmax'))

model_CNN.summary()
```

```
<matplotlib.legend.Legend at 0x29420190f08>
```

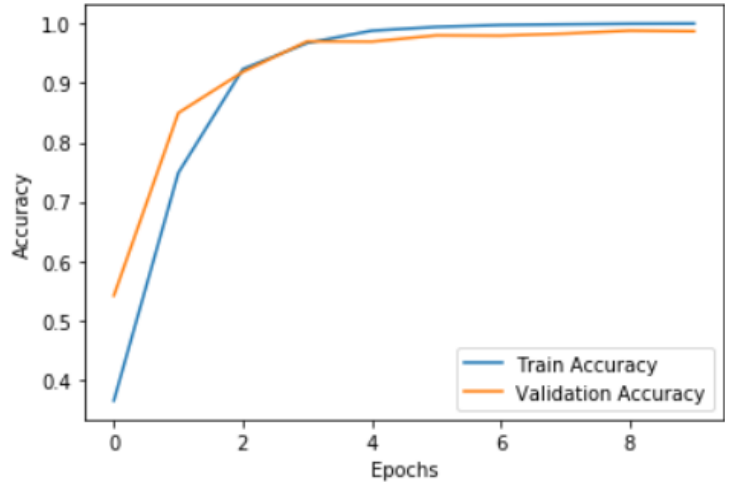


Model 2:

```
model_CNN2 =  keras.models.Sequential()
model_CNN2.add(keras.layers.Conv2D(64, (3, 3), activation='relu',
model_CNN2.add(keras.layers.Conv2D(32, (3, 3), activation='relu',
model_CNN2.add(keras.layers.MaxPooling2D((2, 2)))
model_CNN2.add(keras.layers.Flatten())
model_CNN2.add(keras.layers.Dense(512, activation='relu', kernel_
model_CNN2.add(keras.layers.Dense(256, activation='relu', kernel_
model_CNN2.add(keras.layers.Dense(128, activation='relu', kernel_
model_CNN2.add(keras.layers.Dense(10, activation='softmax'))

model_CNN2.summary()
```

```
<matplotlib.legend.Legend at 0x29420644d88>
```

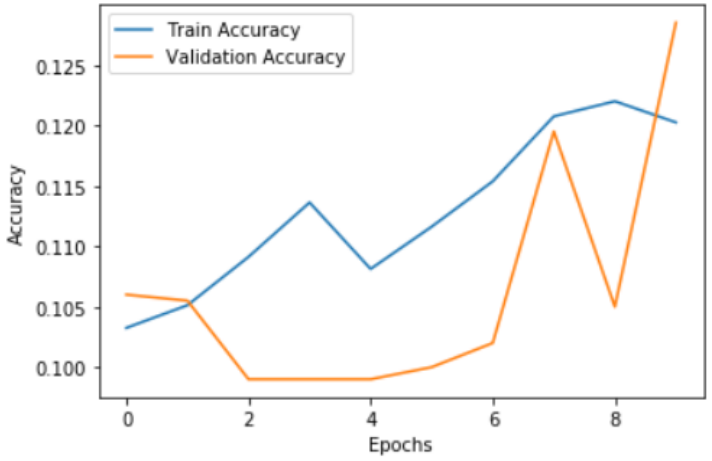


Model 3:

```
model_CNN3 =  keras.models.Sequential()
model_CNN3.add(keras.layers.Conv2D(32, (3, 3), ke
model_CNN3.add(keras.layers.MaxPooling2D((2, 2)))
model_CNN3.add(keras.layers.Dropout(0.5))
model_CNN3.add(keras.layers.Conv2D(64, (3, 3), ke
model_CNN3.add(keras.layers.MaxPooling2D((2, 2)))
model_CNN3.add(keras.layers.Dropout(0.5))
model_CNN3.add(keras.layers.Conv2D(128, (3, 3), k
model_CNN3.add(keras.layers.MaxPooling2D((8, 8)))
model_CNN3.add(keras.layers.Dropout(0.5))
model_CNN3.add(keras.layers.Flatten())
model_CNN3.add(keras.layers.Dense(10, activation=


model_CNN3.summary()
```

```
<matplotlib.legend.Legend at 0x2943c088e48>
```



Modelo Escogido:

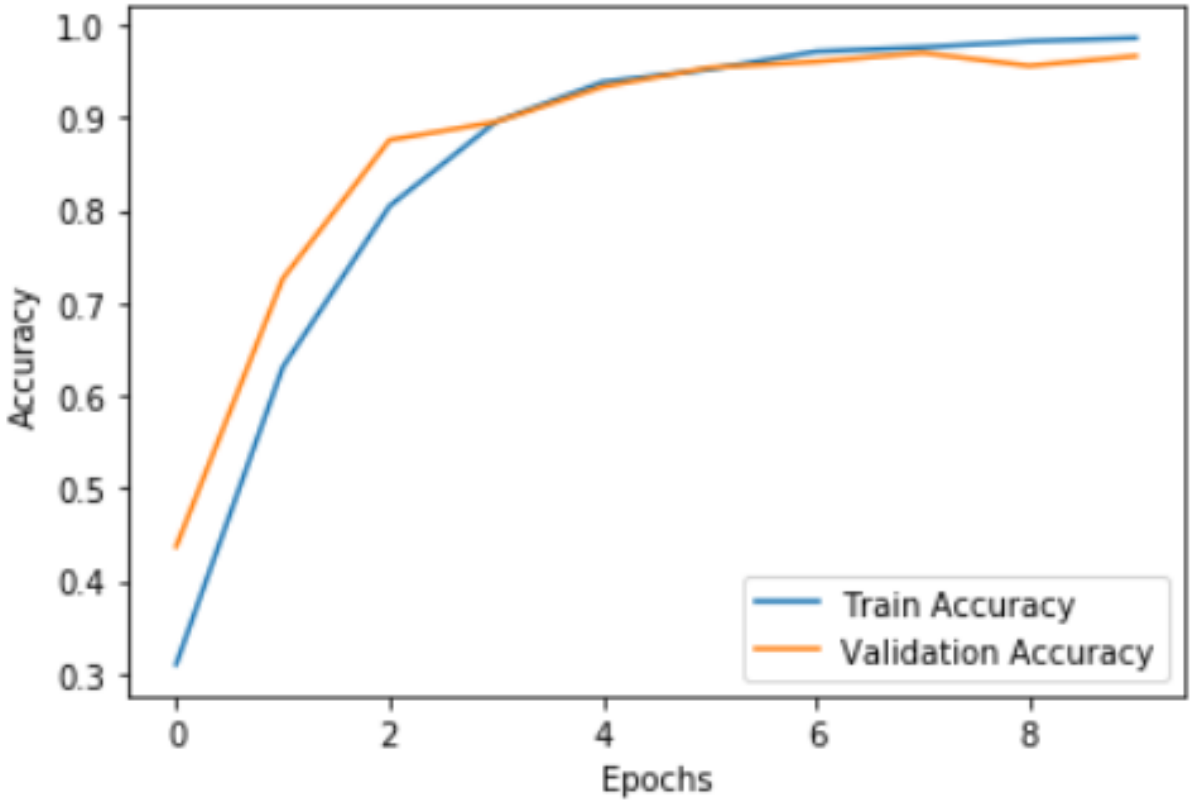
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 60, 60, 20)        1520
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 30, 30, 20)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 26, 26, 50)        25050
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 50)        0
_________________________________________________________________
flatten (Flatten)            (None, 8450)              0
_________________________________________________________________
dense (Dense)                (None, 500)               4225500
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5010
=================================================================
Total params: 4,257,080
Trainable params: 4,257,080
Non-trainable params: 0
```

Entrenamiento:

```
[ ]  history = model_CNN.fit(X_train,y_train, epochs=10, batch_size=160, verbose=2,validation_data=(X_test, y_test))
```

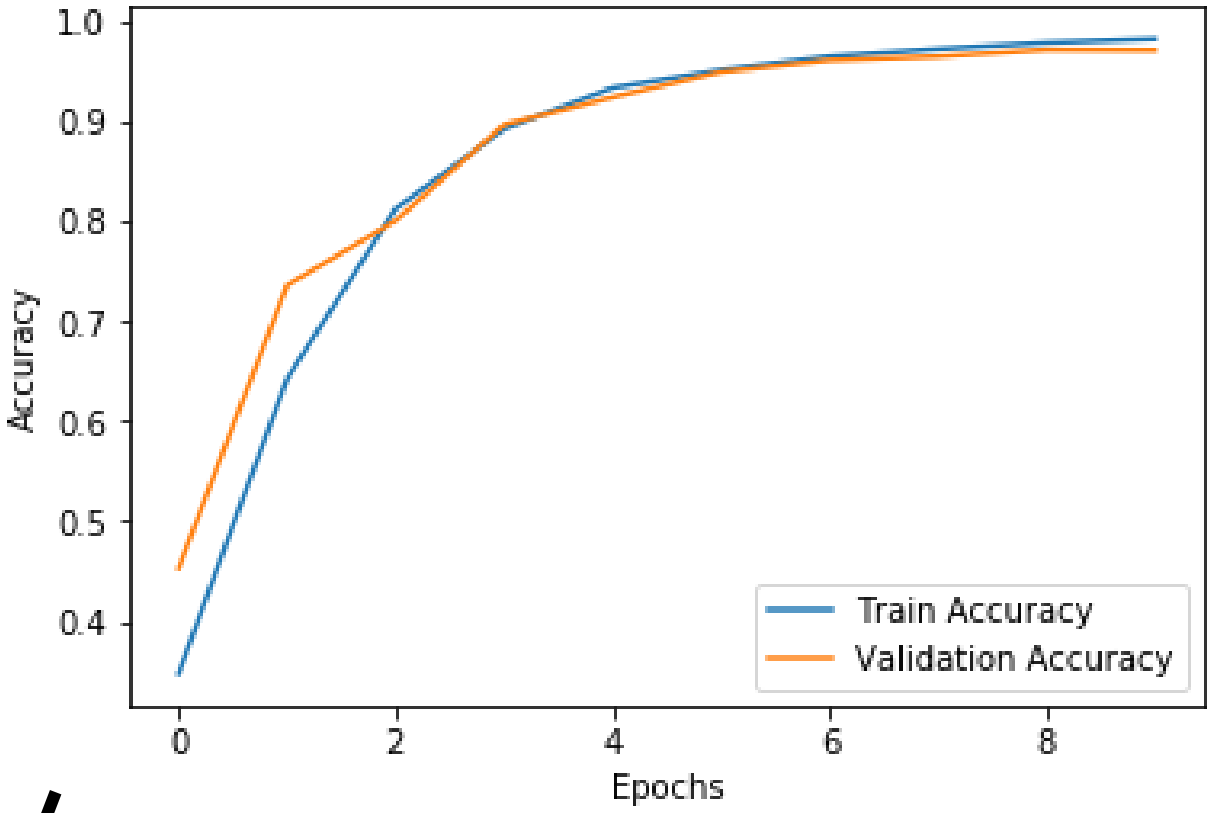
```
Train on 17939 samples, validate on 4485 samples
Epoch 1/10
17939/17939 - 6s - loss: 1.9033 - acc: 0.3476 - val_loss: 1.5848 - val_acc: 0.4531
Epoch 2/10
17939/17939 - 2s - loss: 1.1783 - acc: 0.6422 - val_loss: 0.8958 - val_acc: 0.7358
Epoch 3/10
17939/17939 - 2s - loss: 0.6903 - acc: 0.8123 - val_loss: 0.6564 - val_acc: 0.8004
Epoch 4/10
17939/17939 - 2s - loss: 0.4387 - acc: 0.8922 - val_loss: 0.3983 - val_acc: 0.8965
Epoch 5/10
17939/17939 - 2s - loss: 0.2887 - acc: 0.9334 - val_loss: 0.2954 - val_acc: 0.9240
Epoch 6/10
17939/17939 - 2s - loss: 0.2183 - acc: 0.9513 - val_loss: 0.2125 - val_acc: 0.9494
Epoch 7/10
17939/17939 - 2s - loss: 0.1627 - acc: 0.9648 - val_loss: 0.1710 - val_acc: 0.9610
Epoch 8/10
17939/17939 - 2s - loss: 0.1300 - acc: 0.9722 - val_loss: 0.1440 - val_acc: 0.9648
Epoch 9/10
17939/17939 - 2s - loss: 0.1035 - acc: 0.9784 - val_loss: 0.1173 - val_acc: 0.9715
Epoch 10/10
17939/17939 - 2s - loss: 0.0842 - acc: 0.9823 - val_loss: 0.1127 - val_acc: 0.9710
```

Accuracy/epoch:



8.000 images Train
2.000 images Test

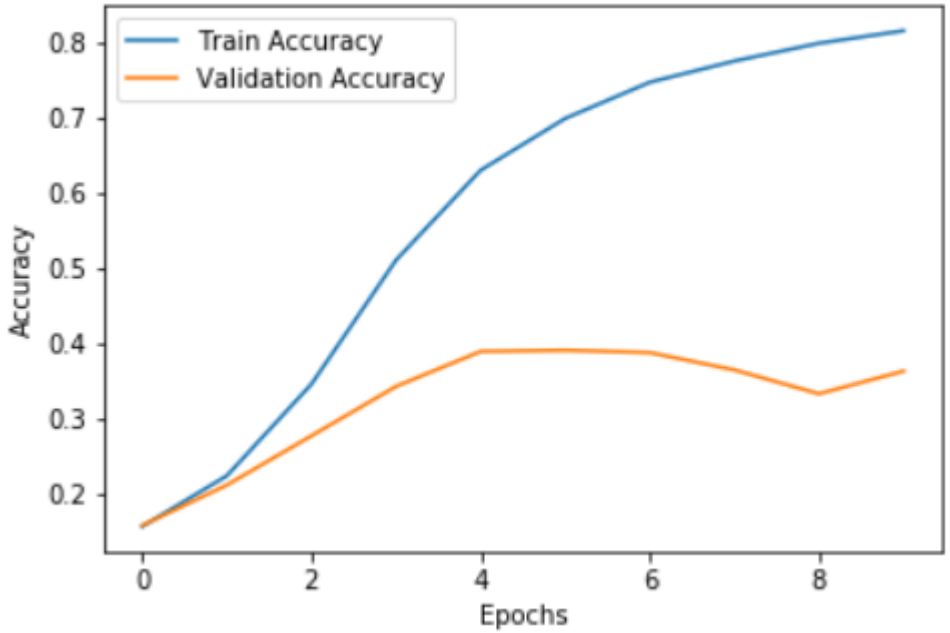
/



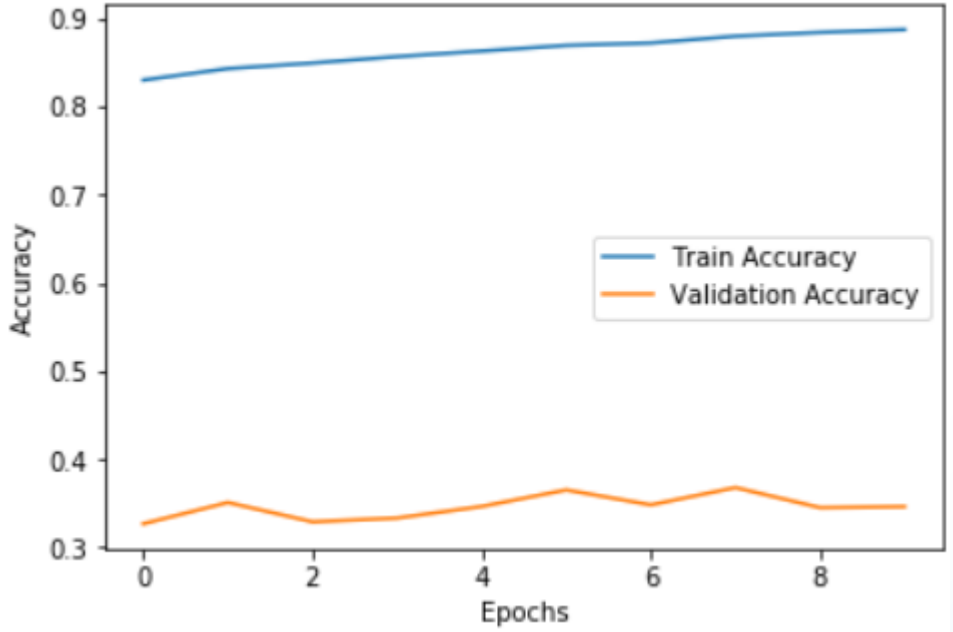
17.939 images Train
4.484 images Test

Transfer Learning (Fully Connected train):

MobileNet:



Vgg19:



Algunos ejemplos:

texting - left

reaching behind

drinking

talking on the phone - right

talking to passenger

Safe Driving

Kaggle Submission:

```
[ ]  predicts=[]
```

```
[ ]  for i in tqdm(range(0,19974)):
       temp_img=cv.resize(cv.cvtColor(cv.imread(path_test_images+'/'+
       temp_predict=model_CNN.predict(np.array([temp_img]))
       predicts.append(np.append(test_images_names[i],temp_predict))
```

```
100%|| 19974/19974 [1:19:43<00:00,  4.06it/s]
```

```
[ ]  for i in tqdm(range(19974, 39948)):
       temp_img=cv.resize(cv.cvtColor(cv.imread(path_test_images+'/'+
       temp_predict=model_CNN.predict(np.array([temp_img]))
       predicts.append(np.append(test_images_names[i],temp_predict))
```

```
100%|| 19974/19974 [1:17:49<00:00,  4.03it/s]
```

```
[ ]  for i in tqdm(range(39948, 59922)):
       temp_img=cv.resize(cv.cvtColor(cv.imread(path_test_images+'/'+
       temp_predict=model_CNN.predict(np.array([temp_img]))
       predicts.append(np.append(test_images_names[i],temp_predict))
```

```
100%|| 19974/19974 [1:12:52<00:00,  4.10it/s]
```

```
[ ]  for i in tqdm(range(59922, 79896)):
       temp_img=cv.resize(cv.cvtColor(cv.imread(path_test_images+'/'+
       temp_predict=model_CNN.predict(np.array([temp_img]))
       predicts.append(np.append(test_images_names[i],temp_predict))
```

```
100%|| 19974/19974 [1:11:08<00:00,  4.78it/s]
```

Wait time	Execution time	Score
0 seconds	1 seconds	17.31954

#LaUISqueQueremos

Universidad Industrial de Santander

www.uis.edu.co

Repositorio del proyecto:

https://github.com/pecons/proyectoCV

Gracias por su atención

1948 - 2018

Patrimonio educativo y cultural de los santandereanos