FACULTAD DE INGENIERIA ELECTRÓNICA

Periodo: 2019-2

1. **PROPONENTE** (s):
   Katherin Daniela Melo Rodríguez Código: 2107261
   Edisson David Sastoque Beltrán Código: 2107709

2. **DIRECTOR DEL TRABAJO:** Enrique Estupiñán Escalante.

3. **TÍTULO DEL TRABAJO DIRIGIDO:** Implementación de Deep Learning en carreteras para reconocimiento de tráfico

4. **AREA:** Control y automatización

5. **ANTECEDENTES Y JUSTIFICACIÓN:**
   El desarrollo masivo y en ocasiones desordenado de las ciudades requiere que la infraestructura crezca de igual manera, pero siempre va a tener un retardo y debe modificarse en la medida en que las tecnologías cambien. Específicamente en la infraestructura vial que requieren las ciudades, normalmente saturadas en las zonas centrales y con mejores especificaciones en la periferia se requiere de mecanismos para el control y distribución de los actores tales como vehículos particulares, transporte de carga, transporte público, bicicletas, medios emergentes como patinetas y naturalmente peatones. Con esto en mente se han propuesto diferentes soluciones desde el acercamiento de la automatización y el Internet de las Cosas (IoT, por sus siglas en ingles) sin embargo, dichas soluciones requieren de programas que permitan parametrizar la información contenida en fuentes diversas como cámaras, registro de peajes, sistemas de supervisión de flotillas, llevar a cabo este objetivo. En particular el análisis de los videos de cámaras localizadas en las grandes autopistas es una necesidad imperativa y necesita nuevos desarrollos a partir de los conocidos como lo son la identificación de placas, color, velocidad y posición de los vehículos, como detección de maniobras peligrosas, comportamientos delictivos o servir de soporte a sistemas de seguimiento de flotillas.

6. **OBJETIVOS:**
   General:
   Especificar y diseñar un programa, que permita extraer características propias de los actores viales en las carreteras relacionadas con la identificación inequívoca de vehículos, rutas, tipos de conducción, peatones, motocicletas y entregarlas para que se pueda reconocer y clasificar los actores y sus conductas

   Específicos:
   - Seleccionar una base de datos que contenga información de video de una carretera.
   - Definir criterios para limitar los actores viales en los que se enfoca el trabajo.
   - Definir criterios de clasificación de actores y sus características

- Definir comportamientos a clasificar de los actores seleccionados
- Definir el almacenamiento y políticas de uso de los videos.
- Definir formato de representación de las características por extraer y su almacenamiento.
- Implementar algoritmos para extraer los parámetros de los actores y sus características en un video seleccionado, incluyendo métodos de aprendizaje de máquina (machine learning)

7. METODOLOGÍA:

Inicialmente se realiza una búsqueda bibliográfica para determinar los hitos más importantes en el estado del arte del problema. Luego se evalúa las posibilidades para la obtención de la base de datos de los videos de la vía seleccionada, la disponibilidad, el formato y los parámetros existentes para comparar los algoritmos a implementar; a su vez esto permite definir los criterios en los que se enfoca el proyecto al limitar los actores viales y sus características, así mismos definir los diferentes grupos en los que se clasifican los actores y comportamiento. Luego se procederá a la implementación de algoritmos que permitan extraer las características deseadas por cada actor, para finalmente realizar pruebas sobre que permitan identificar los aspectos positivos y negativos del algoritmo asociados a la correcta extracción de características y sus rangos de funcionamiento.

8. SECUENCIA Y TIPO DE ACTIVIDADES QUE SE DESARROLLARAN:

- Determinación del Estado del Arte
- Seleccionar base de datos que contenga información de video de una carretera.
- Definir criterios para limitar los actores viales en los que se enfoca el trabajo.
- Definir comportamientos a clasificar de los actores seleccionados.
- Definir el almacenamiento y políticas de uso de los videos.
- Definir formato de representación de las características por extraer y su almacenamiento.
- Implementar algoritmos para extraer los parámetros de los actores y sus características en un video seleccionado
- Incluir métodos de aprendizaje de máquina.
- Realizar pruebas en el algoritmo.
- Corrección de fallos y presentación de la versión final.
- Realización del reporte con resultados
- Presentación de los resultados

9. CRONOGRAMA:

Actividad	Tiempo (semanas)															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Determinación del Estado del Arte	X	X	X													

Seleccionar base de datos que contenga información de video de una carretera			X	X												
Definir criterios para limitar los actores viales en los que se enfoca el trabajo.				X	X											
Definir comportamientos a clasificar de los actores seleccionados				X	X	X										
Definir el almacenamiento y políticas de uso de los videos.			X		X											
Definir formato de representación de las características por extraer y su almacenamiento.				X	X											
Implementar algoritmos para extraer los parámetros de los actores y sus características en un video seleccionado					X	X	X	X			X	X				
Incluir métodos de aprendizaje de máquina				X				X	X	X	X					
Realizar pruebas en el algoritmo									X	X	X	X				
Corrección de fallos y presentación de la versión final.											X	X	X	X		
Realización del reporte con resultados		X		X		X		X				X			X	X
Presentación de los resultados																X

10. INSTALACIONES Y EQUIPOS:

Laboratorio de ingeniería Electrónica, equipo procesamiento de video, Matlab, OpenCV

11. COSTO DEL TRABAJO DIRIGIDO:

12. FUENTES DE FINANCIACIÓN:

La Escuela provee acceso a software Matlab, OpenCV y de ser posible el uso de un equipo procesamiento de video. Otra fuente de financiación es personal de los proponentes.

Implementación de algoritmos de visión de máquina para la caracterización de actores viales

Antecedentes y justificación

El desarrollo masivo y en ocasiones desordenado de las ciudades requiere que la infraestructura crezca de igual manera, pero siempre va a tener un retardo y debe modificarse en la medida en que las tecnologías cambien. Específicamente en la infraestructura vial que requieren las ciudades, normalmente saturadas en las zonas centrales y con mejores especificaciones en la periferia se requiere de mecanismos para el control y distribución de los actores tales como vehículos particulares, transporte de carga, transporte público, bicicletas, medios emergentes como patinetas y naturalmente peatones.

Con esto en mente se han propuesto diferentes soluciones desde el acercamiento de la automatización y el Internet de las Cosas (IoT, por sus siglas en inglés) sin embargo, dichas soluciones requieren de programas que permitan parametrizar la información contenida en fuentes diversas como cámaras, registro de peajes, sistemas de supervisión de flotillas, llevar a cabo este objetivo. En particular el análisis de los videos de cámaras localizadas en las grandes autopistas es una necesidad imperativa y necesita nuevos desarrollos a partir de los conocidos como lo son la identificación de placas, color, velocidad y posición de los vehículos, como detección de maniobras peligrosas, comportamientos delictivos o servir de soporte a sistemas de seguimiento de flotillas.

Objetivos

General:

Especificar y diseñar un programa, que permita extraer características propias de los actores viales en las carreteras relacionadas con la identificación inequívoca de vehículos, rutas, tipos de conducción, peatones, motocicletas y entregarlas para que se pueda reconocer y clasificar los actores y sus conductas

Específicos:

- Seleccionar una base de datos que contenga información de video de una carretera.
- Definir criterios para limitar los actores viales en los que se enfoca el trabajo.
- Definir criterios de clasificación de actores y sus características
- Definir comportamientos a clasificar de los actores seleccionados
- Definir el almacenamiento y políticas de uso de los videos.
- Definir formato de representación de las características por extraer y su almacenamiento.
- Implementar algoritmos para extraer los parámetros de los actores y sus características en un video seleccionado, incluyendo métodos de aprendizaje de máquina (machine learning)

Metodología

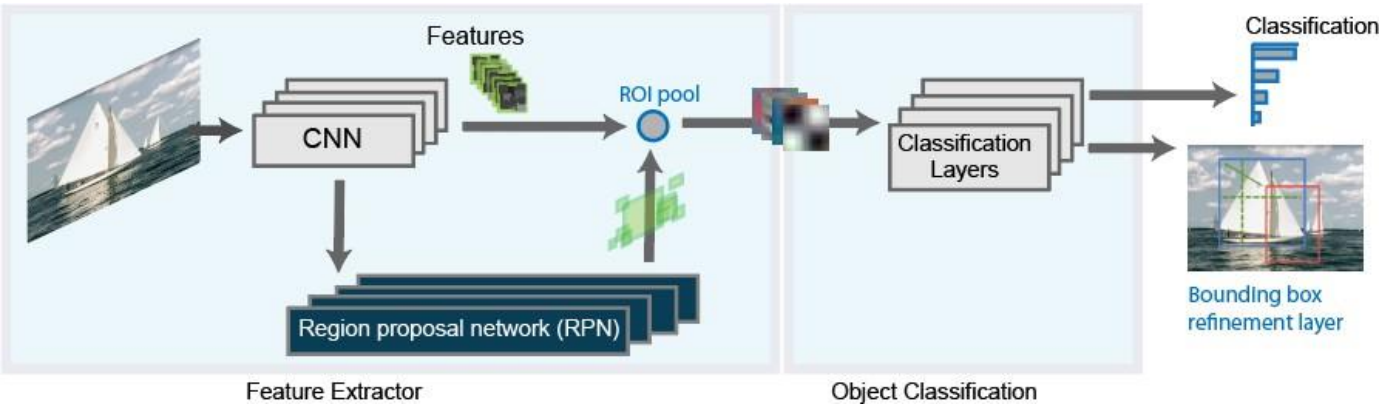
Inicialmente se realiza una búsqueda bibliográfica para determinar los hitos más importantes en el estado del arte del problema. Luego se evalúa las posibilidades para la obtención de la base de datos de los videos de la vía seleccionada, la disponibilidad, el formato y los parámetros existentes para comparar los algoritmos a implementar; a su vez esto permite definir los criterios en los que se enfoca el proyecto al limitar los actores viales y sus características, así mismos definir los diferentes grupos en los que se clasifican los actores y comportamiento. Luego se procederá a la implementación de algoritmos que permitan extraer las características deseadas por cada actor, para finalmente realizar pruebas sobre que permitan identificar los aspectos positivos y negativos del algoritmo asociados a la correcta extracción de características y sus rangos de funcionamiento.

Entrenamiento de Red neuronal convolucional Faster-RCNN para detección de vehículos

Redes Faster RCNN

Las redes neuronales convolucionales son especialmente útiles para la clasificación de imágenes, detección de objetos y tareas de reconocimiento, realizan segmentación de imágenes en el cual se caracterizan los diferentes componentes de estas y así obtienen información de la imagen.

Las CNN se implementan como una serie de capas interconectadas en las que se toma cada imagen y se recorre por medio de recuadros o ventanas más pequeñas y la escanea buscando objetos por medio un algoritmo llamado *Selective Search,* lo que diferencia las redes RCNN de las Faster RCNN es que se agrega una red de propuesta regional (RPN) en la última capa de la RCNN la cual genera múltiples regiones posibles basadas en uniones espaciales de dimensiones fijas llamadas cajas de anclaje como se muestra en la imagen.



Cada propuesta contiene un puntaje (score) para la presencia del objeto en esa región en particular y 4 coordenadas que representan el cuadro delimitador de la región.

Específicamente la red Faster-RCNN encontrada en el software Matlab se compone de las siguientes capas:

- **`ImageInput`**: En la que se ingresa la imagen a la red y se especifica el tamaño de la imagen de entrada, luego se normaliza en este caso restando la media.
- **`Convolution2dLayer:`** Estas capas convolucionales aplican en su entrada un conjunto de filtros. Los filtros se aprenden automáticamente durante el entrenamiento de la red. Convoluciona la entrada moviendo los filtros por toda la extensión de la imagen y luego agrega el *bias*, se puede especificar de a cuentos pixeles ira recorriendo el filtro la imagen y el número de filtros.
- **`ReLu:`** Las capas ReLu realizan una operación en la que todos los valores negativos los vuelve cero
- **`ROIMaxPooling`**: Las capas de agrupación (pooling) disminuyen sus entradas y ayudan a consolidar las características de la imagen. Genera mapas de features de tamaño fijo para cada ROI rectangular dentro del mapa de features de entrada.
- **`FullyConnectedLayer:`** Multiplica la entrada por una matriz de peso obtenida durante el entrenamiento. Se puede especificar el número de salidas que tendrá.

- **`RegionProposalLayer:`** Capa de red de propuesta regional (RPN), se puede especificar el tamaño inicial de las cajas de anclaje.
- **`BoxRegressionLayer:`** Esta capa define las cajas que delimitan los objetos detectados en la imagen.
- **`SoftMax:`** aplica una función sotfmax a la entrada. La función softmax genera un vector que representa las distribuciones de probabilidad de una lista de resultados potenciales.
- **`Classification:`** El número de clases que se obtienen en esta capa es de acuerdo con el tamaño de la salida de la capa anterior

Creación de una red Faster RCNN a partir de una red preentrenada ResNet50

Los pasos que se usan para la creación de una red para detección de vehículos se describen a continuación:

Primero se toma como base una red preentrenada, en este caso ResNet50

```
net = resnet50;
lgraph = layerGraph(net);
```

Se eliminan las 3 últimas capas con el comando `removeLayers`

```
layersToRemove = {
    'fc1000'
    'fc1000_softmax'
    'ClassificationLayer_fc1000'
    };
lgraph = removeLayers(lgraph, layersToRemove);
```

Se especifica el número de clases que la red deberá clasificar y se le suma una más para incluir la clase 'fondo'

```
numClasses = 2;
numClassesPlusBackground = numClasses + 1;
```

Se definen las mismas 3 capas eliminadas de acuerdo con el número de clases especificadas anteriormente

Nota: Cada grupo de clases creadas en conjunto estarán conectadas inicialmente entre ellas según el orden de creación.

```
newLayers = [
    fullyConnectedLayer(numClassesPlusBackground, 'Name', 'rcnnFC')
    softmaxLayer('Name', 'rcnnSoftmax')
    classificationLayer('Name', 'rcnnClassification')
    ];
```

Y se añaden a la red con el comando `addlayers`

```
lgraph = addLayers(lgraph, newLayers);
```

Se conectan estas nuevas capas a la última capa de la red con el comando `connectLayers` y especificando cual capa se conecta y a que capa se conecta

```
lgraph = connectLayers(lgraph, 'avg_pool', 'rcnnFC');
```

Para la ubicación de las cajas que delimitan el objeto encontrado se necesita un numero de salidas igual a 4 por cada clase.

```
numOutputs = 4 * numClasses;
```

Y a partir de este valor se definen dos capas para las cajas, luego se agregan a la red y se conectan a la capa `'avg_pool'`

```
boxRegressionLayers = [
    fullyConnectedLayer(numOutputs,'Name','rcnnBoxFC')
    rcnnBoxRegressionLayer('Name','rcnnBoxDeltas')
    ];
lgraph = addLayers(lgraph, boxRegressionLayers);
lgraph = connectLayers(lgraph,'avg_pool','rcnnBoxFC');
```

Se escoge una capa de extracción de características dependiendo de la red base escogida.

```
featureExtractionLayer = 'activation_40_relu';
```

Se desconectan las capas conectadas a la capa de extracción de características.

```
lgraph = disconnectLayers(lgraph, featureExtractionLayer,'res5a_branch2a');
lgraph = disconnectLayers(lgraph, featureExtractionLayer,'res5a_branch1');
```

Se crea una capa de `'pooling'` y se conecta la capa de extracción de características

```
outputSize = [14 14];
roiPool = roiMaxPooling2dLayer(outputSize,'Name','roiPool');
lgraph = addLayers(lgraph, roiPool);
lgraph = connectLayers(lgraph, featureExtractionLayer,'roiPool/in');
```

Por último, se conectan las capas desconectadas con anterioridad a la capa de `'pooling'`

```
lgraph = connectLayers(lgraph, 'roiPool','res5a_branch2a');
lgraph = connectLayers(lgraph, 'roiPool','res5a_branch1')
```

```
lgraph =
  LayerGraph with properties:

         Layers: [180×1 nnet.cnn.layer.Layer]
    Connections: [195×2 table]
```

Las redes Faster RCNN utilizan una red de propuesta regional (RPN), las cuales diferencian la región 'objeto' de la región 'fondo'. Para añadir esta red se crea una capa de `'regionalProposal'` y se definen un tamaño inicial de cajas de anclaje para las posibles regiones encontradas.

```
anchorBoxes = [
    16 16
    32 16
    16 32
    ];
```

Se crea la capa y se agrega a la red

```
proposalLayer = regionProposalLayer(anchorBoxes,'Name','regionProposal');
lgraph = addLayers(lgraph, proposalLayer);
```

Luego se crea una capa convolucional para la RPN especificando el número de filtros y debajo de esta una capa `'relu'` para la RPN y se conectan a la capa de extracción de características

```
numFilters = 1024;
rpnLayers = [
    convolution2dLayer(3, numFilters,'padding',[1 1],'Name','rpnConv3x3')
    reluLayer('Name','rpnRelu')
    ];
lgraph = addLayers(lgraph, rpnLayers);
lgraph = connectLayers(lgraph, featureExtractionLayer, 'rpnConv3x3');
```

Se crea una segunda capa convolucional para la capa de clasificación de la RPN, esta vez se especifican el número de diferentes tamaños para las regiones y se añade una capa `'sofmax'`, se agregan las capas a la red y se conectan a la `'relu'` de la RPN.

```
numAnchors = size(anchorBoxes,1);
rpnClsLayers = [
    convolution2dLayer(1, numAnchors*2,'Name', 'rpnConv1x1ClsScores')
    rpnSoftmaxLayer('Name', 'rpnSoftmax')
    rpnClassificationLayer('Name','rpnClassification')
    ];
lgraph = addLayers(lgraph, rpnClsLayers);
lgraph = connectLayers(lgraph, 'rpnRelu', 'rpnConv1x1ClsScores');
```

Se crea una capa de `'boxRegression'` para la RPN y se conectan a la capa `'reluRPN'`

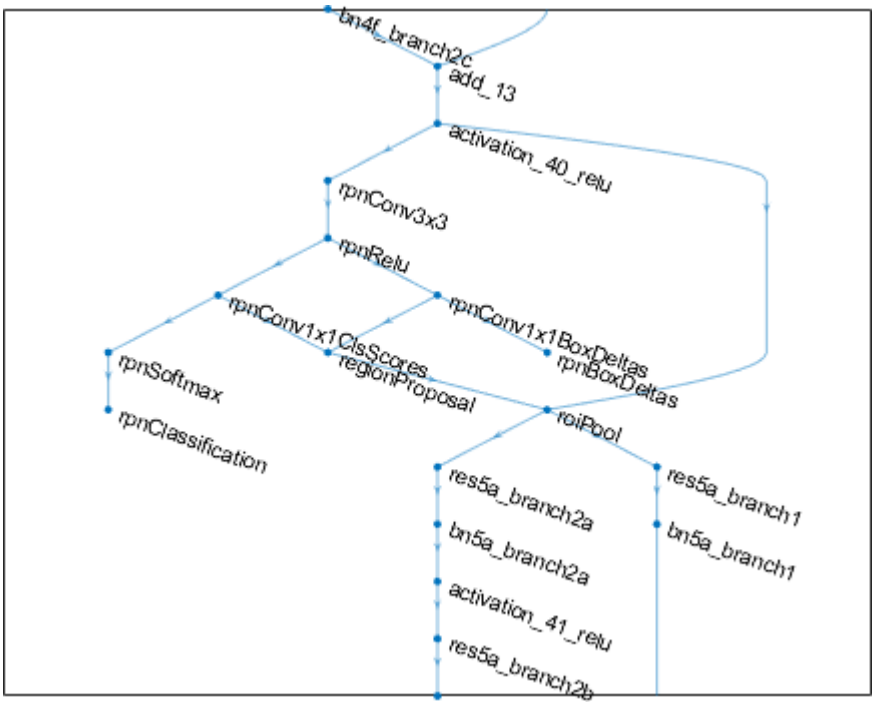
```
rpnRegLayers = [
    convolution2dLayer(1, numAnchors*4, 'Name', 'rpnConv1x1BoxDeltas')
    rcnnBoxRegressionLayer('Name', 'rpnBoxDeltas');
    ];

lgraph = addLayers(lgraph, rpnRegLayers);
lgraph = connectLayers(lgraph, 'rpnRelu', 'rpnConv1x1BoxDeltas');
```

Por último se conectan las salidas de la red RPN a la capa de `'pooling'`

```
lgraph = connectLayers(lgraph, 'rpnConv1x1ClsScores', 'regionProposal/scores');
lgraph = connectLayers(lgraph, 'rpnConv1x1BoxDeltas', 'regionProposal/boxDeltas');
lgraph = connectLayers(lgraph, 'regionProposal', 'roiPool/roi');
```

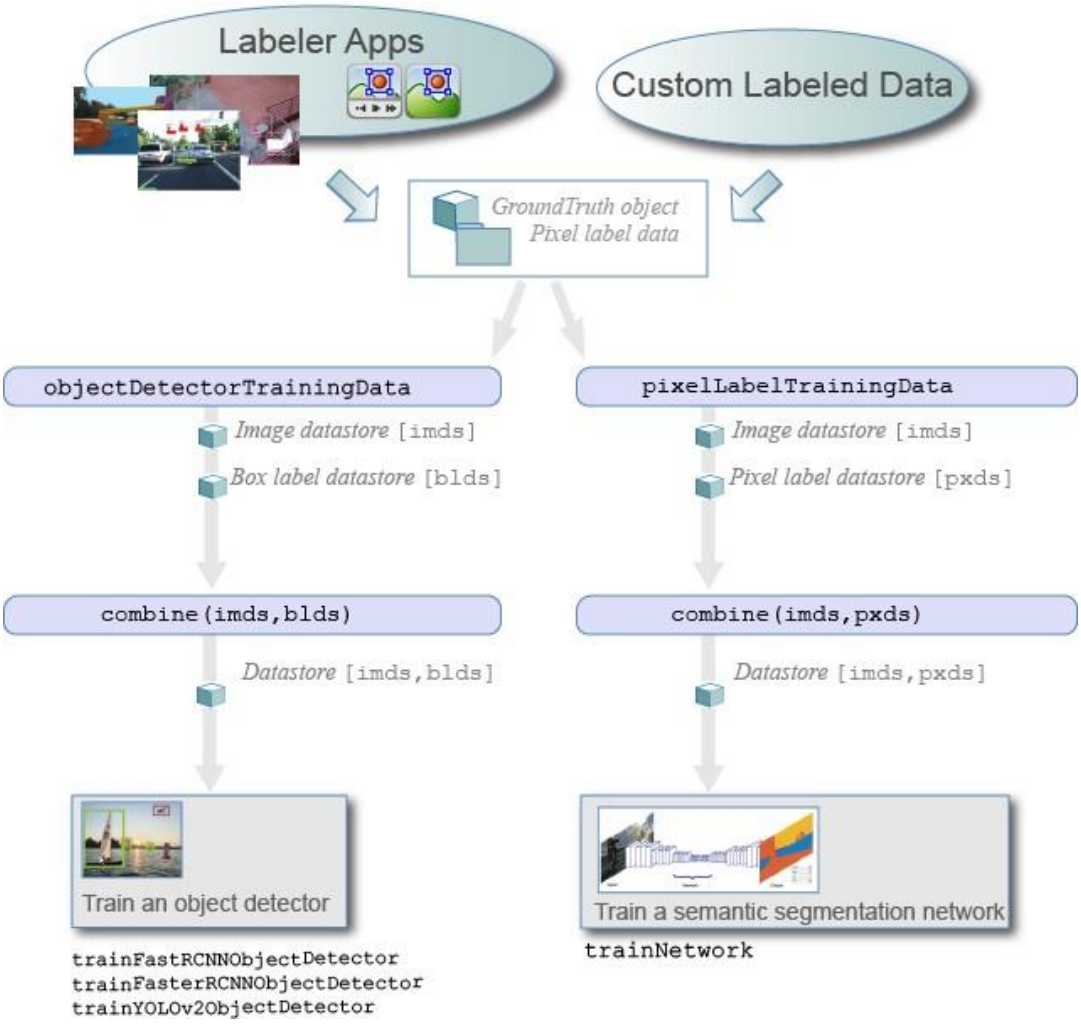
Se muestra un grafica de la red final.

```
figure
plot(lgraph)
ylim([30 42])
```



Entrenamiento de la red

Para la parte del entrenamiento si se quiere utilizar una base de datos propia en necesario agregar un ROI con su respectivo *label* a cada imagen que se use para el entrenamiento como se muestra en la siguiente imagen



Matlab tiene disponible la aplicación *Image Labeler* que permite definir las regiones de interés de cada imagen de manera manual o automática en formato *GroundTruth*. El proceso el etiquetado del Dataset se describe en el apéndice de acondicionamiento de imágenes.

Una vez obtenido el archivo *GroundThruth* podemos sacar de este el path donde se encuentra ubicada la imagen y el ROI con el siguiente comando

```
trainingData =  objectDetectorTrainingData(gTruth);
```

La base de datos se organiza de manera aleatoria para el entrenamiento del detector. se toma un 70% del Dataset para el entrenamiento y el 30% restante para evaluar el funcionamiento de la red

```
rng(0)
shuffledIdx = randperm(height(trainingData));
idx = floor(0.7 * height(trainingData));
trainingDataTbl = trainingData(shuffledIdx(1:idx),:);
testDataTbl = trainingData(shuffledIdx(idx+1:end),:);
```

Luego se crea un datastore tanto de las imágenes como de los recuadros del área de interés para luego combinarlas.

```
imds = imageDatastore(trainingData.imageFilename);
blds = boxLabelDatastore(trainingData(:,2:end));
ds = combine(imds, blds);
```

Se definen las opciones para el entrenamiento

```
options = trainingOptions('sgdm', ...
    'MiniBatchSize', 20, ...
    'InitialLearnRate', 1e-3, ...
    'MaxEpochs', 7, ...
    'VerboseFrequency', 50, ...
    'CheckpointPath', tempdir);
```

Se especifican los valores de las siguientes variables:

- `Minibatchsize:` número de imágenes usadas en cada iteración.
- `InitialLearnRate:` la taza de muestreo inicial.
- `MaxEpochs:` número de ciclos completos de entrenamiento durante todo el proceso
- `VerboseFrecuency:` sirve para ver el progreso del entrenamiento en la ventana de comandos y especificar cada cuantas iteraciones se muestra el avance
- `CheckpointPath:` la dirección en la que se guarda el progreso del entrenamiento

Hay que tener en cuenta que la función de entrenamiento a usar con la Fast-RCNN no soporta las siguientes opciones de entrenamiento

- `training-progress` para la opción `Plots`
- `ValidationData`, `ValidationFrequency` y `ValidationPatience`
- `OutputFcn`

Finalmente se entrena el detector

```
detector = trainFasterRCNNObjectDetector(trainingData, lgraph, options, ...
    'NegativeOverlapRange',[0 0.3], ...
    'PositiveOverlapRange',[0.6 1]);
```

```
Training a Faster R-CNN Object Detector for the following object classes:

* Pickup
* Truck

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single CPU.
|=========================================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     Loss     |   Accuracy   |     RMSE     |      Rate       |
|=========================================================================================================|
|       1 |           1 |       00:00:08 |      26.6481 |       56.72% |         1.33 |          0.0010 |
|       5 |          50 |       00:07:34 |       1.8205 |       84.47% |         0.27 |          0.0010 |
|       7 |          70 |       00:10:36 |       1.2332 |       85.43% |         0.22 |          0.0010 |
|=========================================================================================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 207 training images...done.
Training on single CPU.
|=========================================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     Loss     |   Accuracy   |     RMSE     |      Rate       |
|=========================================================================================================|
|       1 |           1 |       00:00:13 |      23.4805 |       37.21% |         0.85 |          0.0010 |
|       5 |          50 |       00:11:54 |       5.8929 |       68.52% |         0.44 |          0.0010 |
|       7 |          70 |       00:16:42 |       5.6243 |       82.72% |         0.44 |          0.0010 |
|=========================================================================================================|

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single CPU.
|=========================================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     Loss     |   Accuracy   |     RMSE     |      Rate       |
|=========================================================================================================|
|       1 |           1 |       00:00:06 |       6.0538 |       83.04% |         0.55 |          0.0010 |
|       5 |          50 |       00:05:52 |       2.5921 |       85.47% |         0.34 |          0.0010 |
|       7 |          70 |       00:08:15 |       2.1766 |       85.03% |         0.31 |          0.0010 |
|=========================================================================================================|

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 207 training images...done.

Training on single CPU.
|=========================================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     Loss     |   Accuracy   |     RMSE     |      Rate       |
|=========================================================================================================|
|       1 |           1 |       00:00:10 |       9.6498 |       66.39% |         0.48 |          0.0010 |
|       5 |          50 |       00:09:14 |       6.1215 |       84.29% |         0.36 |          0.0010 |
|       7 |          70 |       00:12:57 |       2.6445 |       87.14% |         0.23 |          0.0010 |
|=========================================================================================================|

Detector training complete
```

Pruebas y Evaluación de la red

Se realizaron 4 pruebas para entrenar la red con 4 Datasets diferentes,

Para la evaluación de esta red se toman 3 factores:

- Accurancy (exactitud): Porcentaje de predicciones correctas.
- Recall (recuperación): Porcentaje de casos positivos capturados
- Precision (precisión): Porcentaje de predicciones positivas correctas.

Para obtener estos datos de cada prueba se utiliza el siguiente código:

Primero se crea una tabla para guardar los resultados

```
p=size(testDataTbl);
results=table('Size',[p(1),2],...
    'VariableTypes',{'cell','cell'},...
    'VariableNames',{'Boxes','Scores'});
```

Luego se corre el detector con el Dataset de prueba, y se guardan los resultados en la tabla

```
for i=1:p(1)
    I=imread(testDataTbl.imageFilename{i});
    [bboxes, scores]=detect(detector,I);
    results.Boxes{i}=bboxes;
    results.Scores{i}=scores;
end
```

Por ultimo de toman los datos

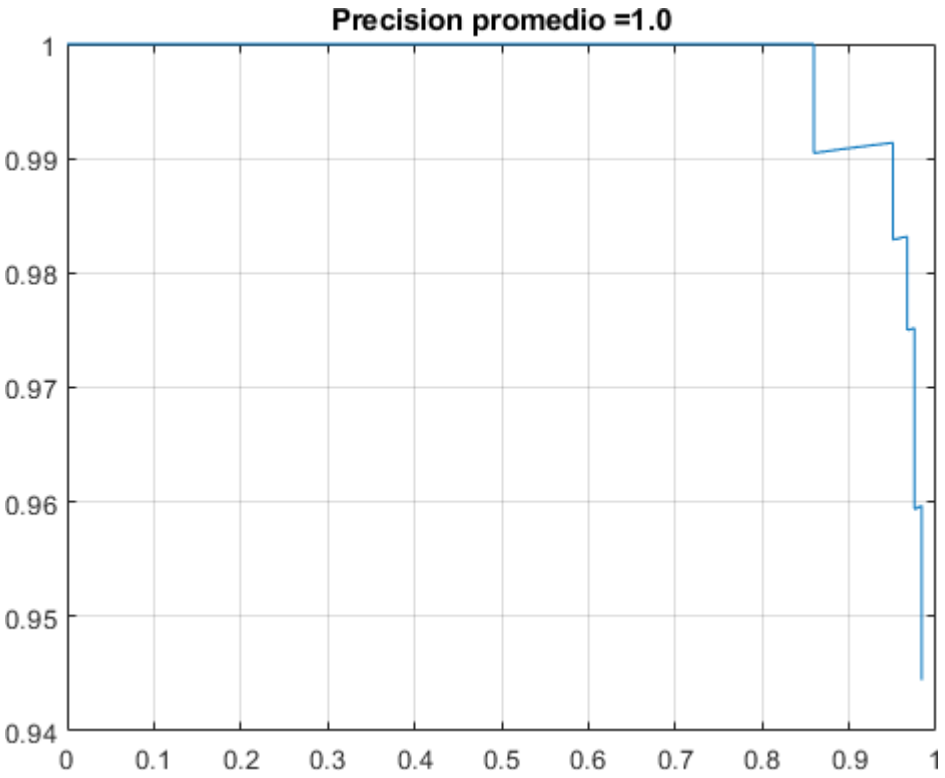
```
[ap,recall,precision]=evaluateDetectionPrecision(results,testDataTbl(:,2));
```

Se toman estos datos para cada prueba y también se evalúa su funcionamiento con 3 imágenes diferentes a las del Dataset.

Prueba 1: Dataset de vehículos de Matlab

Para la primera prueba se utilizó el Dataset incluido en la carpeta de Matlab 'fasterRCNNVehicleTrainingData.mat' que contiene 295 imágenes, para este Dataset no fue necesario agregar las ROI ya que este ya las incluía.

```
figure
plot(recall,precision)
grid on
title(sprintf('Precision promedio =%.1f',ap))
```



Se comprueba su funcionamiento cargando una imagen cualquiera de un vehículo

```
img = imread('prueba1.jpg');
```

Luego se corre el detector sobre la imagen y se muestran los resultados.

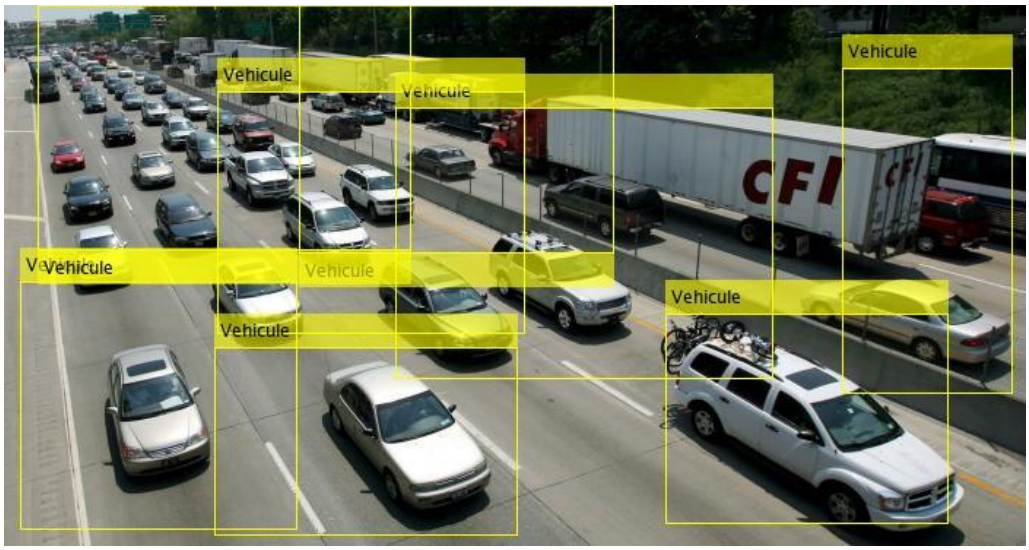
```
[bbox, score, label] = detect(detector,img)
```

```
bbox =
    27     5   205   156
score = single

1.0000
label = categorical
     Vehicule
```

```
detectedImg = insertObjectAnnotation(img,'Rectangle',bbox,cellstr(label));
figure
imshow(detectedImg)
```


```
img = imread('prueba2.jpg');
[bbox, score, label] = detect(detector,img);
detectedImg = insertObjectAnnotation(img,'Rectangle',bbox,cellstr(label));
figure
imshow(detectedImg)
```

```
img = imread('prueba3.jpg');
[bbox, score, label] = detect(detector,img);
detectedImg = insertObjectAnnotation(img,'Rectangle',bbox,cellstr(label));
figure
imshow(detectedImg)
```



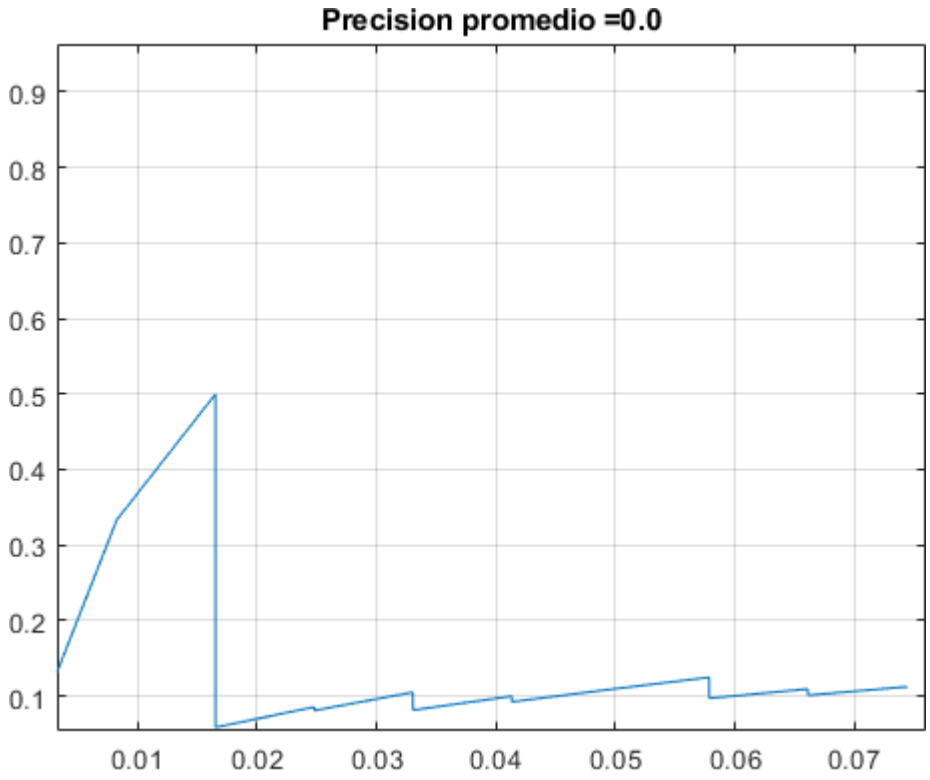
Prueba 2: Dataset carros

El segundo Dataset que se utilizó contiene 402 imágenes de carros.

Las imágenes de este Dataset venían de diferentes tamaños lo que hacía más lento el entrenamiento de la red, así que se realizó un código para igualar su tamaño, dicho código esta descrito en el anexo de acondicionamiento de imágenes.

A su vez para este Dataset fue necesario agregar la ROI con el programa *ImageLabeler.*

```
figure
plot(recall,precision)
grid on
title(sprintf('Precision promedio =%.1f',ap))
```

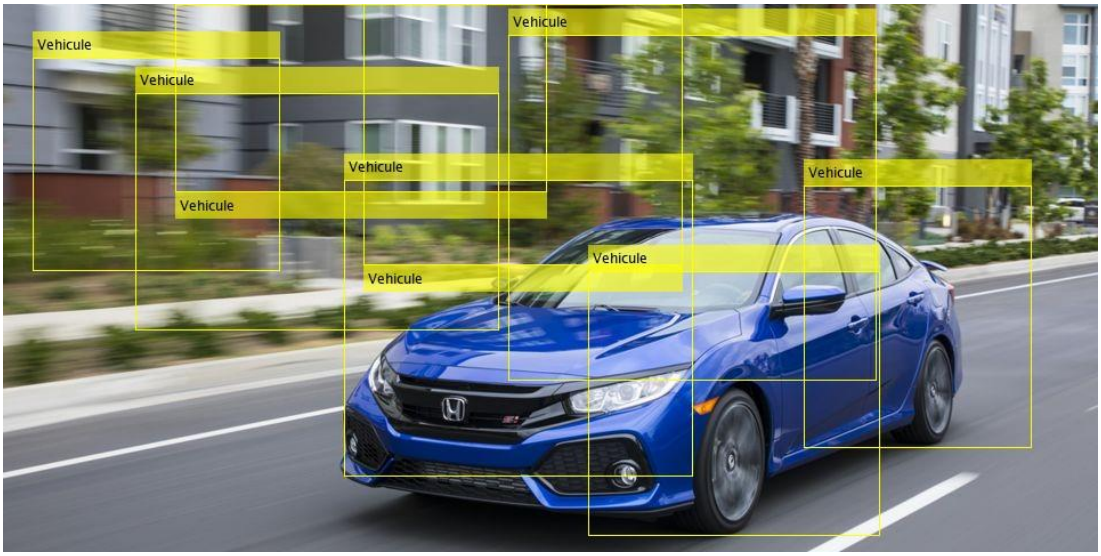



```
img = imread('prueba1.jpg');
[bbox, score, label] = detect(detector,img)
```

```
bbox =
   300   155   306   260
   152     1   326   165
   444    28   323   303
   117    79   319   208
   317     1   280   229
   703   160   200   230
   514   235   256   232
    27    48   217   187
score =
8×1 single column vector
    0.8920
    0.6198
    0.8249
    0.8384
    0.7855
    0.9029
    0.9294
    0.5954
label = 8×1 categorical array
     Vehicule
     Vehicule
     Vehicule
     Vehicule
     Vehicule
     Vehicule
     Vehicule
```

```
Vehicule
```

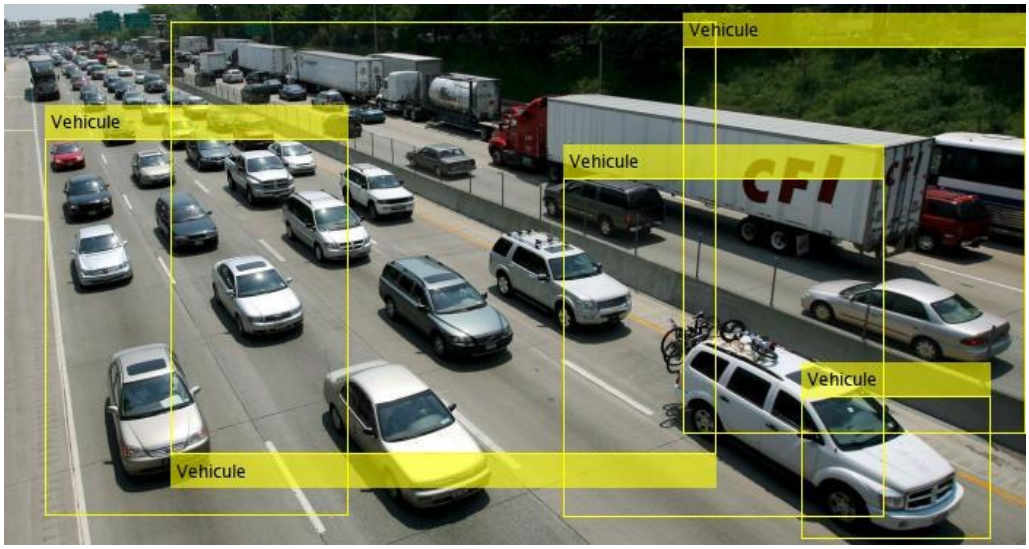
```
detectedImg = insertObjectAnnotation(img,'Rectangle',bbox,cellstr(label));
figure
imshow(detectedImg)
```




```
img = imread('prueba2.jpg');
[bbox, score, label] = detect(detector,img);
detectedImg = insertObjectAnnotation(img,'Rectangle',bbox,cellstr(label));
figure
imshow(detectedImg)
```


```
img = imread('prueba3.jpg');
[bbox, score, label] = detect(detector,img);
detectedImg = insertObjectAnnotation(img,'Rectangle',bbox,cellstr(label));
figure
```

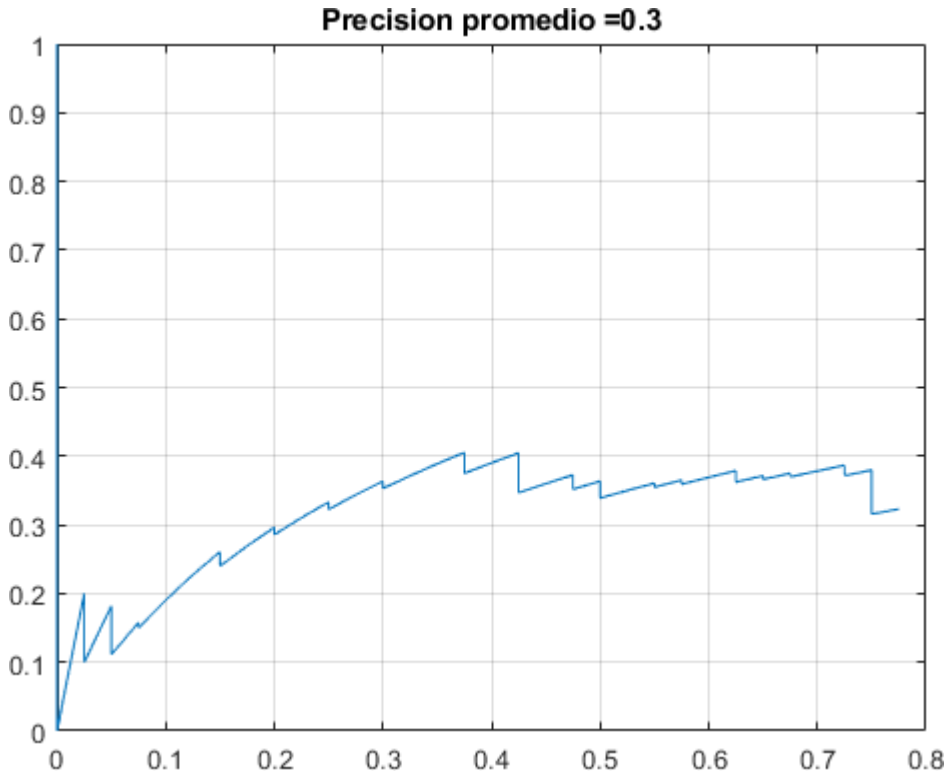
```
imshow(detectedImg)
```



Prueba 3: Dataset truck y pickup

El tercer Dataset que se utilizó contiene 300 imágenes de camiones y pickup, 150 cada clase.

```
figure
plot(recall,precision)
grid on
title(sprintf('Precision promedio =%.1f',ap))
```




```
img = imread('prueba1.jpg');
[bbox, score, label] = detect(detector,img)
```

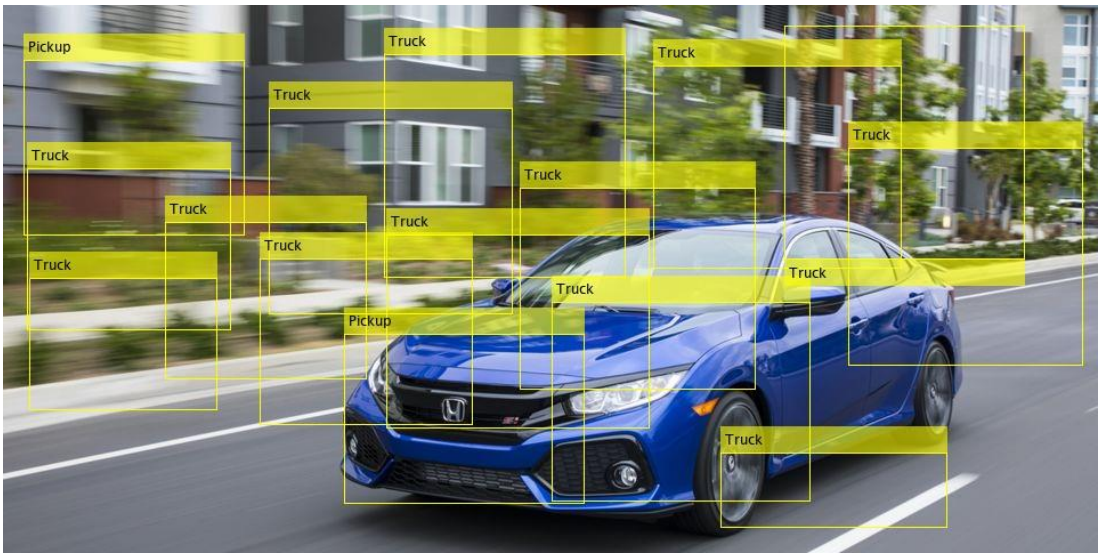
```
bbox =
   234    91   214   181
   571    54   218   178
   742   126   206   191
   482   261   227   175
   143   191   177   138
   454   161   207   177
    19    49   194   154
   300   289   211   149
    22   144   179   142
    24   240   165   116
     ⋮
```

```
score =
15×1 single column vector
    0.5908
    0.5301
    0.6033
    0.5516
    0.5891
    0.5312
    0.5449
    0.5771
    0.5197
    0.6032
     ⋮
```

```
label = 15×1 categorical array
     Truck
```

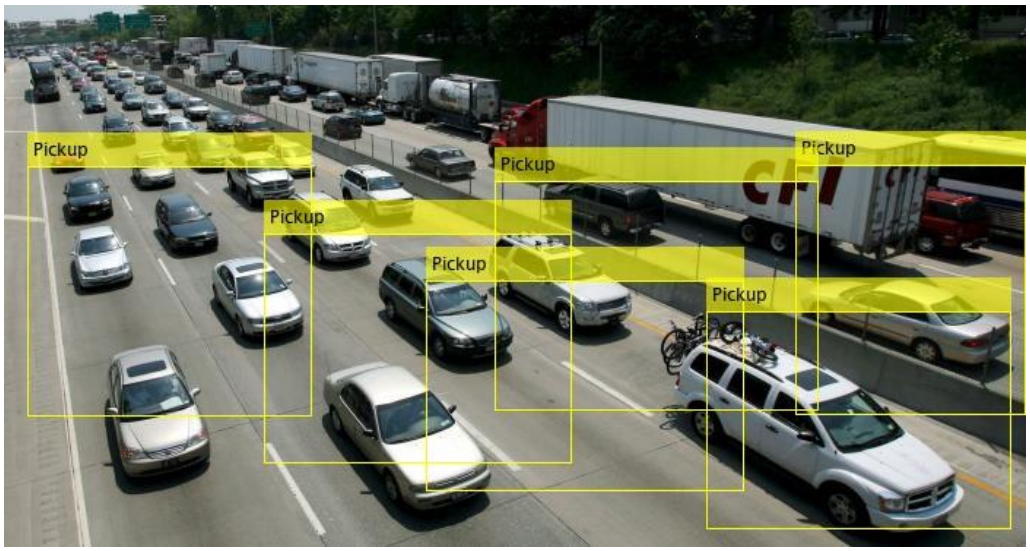
```
Truck
Truck
Truck
Truck
Truck
Pickup
Pickup
Truck
Truck
Truck
Truck
Truck
Truck
Truck
```

```
detectedImg = insertObjectAnnotation(img,'Rectangle',bbox,cellstr(label));
figure
imshow(detectedImg)
```




```
img = imread('prueba2.jpg');
[bbox, score, label] = detect(detector,img);
detectedImg = insertObjectAnnotation(img,'Rectangle',bbox,cellstr(label));
figure
imshow(detectedImg)
```


```
img = imread('prueba3.jpg');
[bbox, score, label] = detect(detector,img);
detectedImg = insertObjectAnnotation(img,'Rectangle',bbox,cellstr(label));
figure
imshow(detectedImg)
```



Prueba 4: Dataset bus y carro

El cuarto Dataset que se utilizó contiene 936 imágenes de un video de carretera en el que transitan buses y carros.

No fue posible entrenar la red con este Dataset por las siguientes razones:

- En un principio se pensó que podía ser la gran cantidad de imágenes del Dataset por lo que se optó por reducir el número a 300 imágenes, ya que la red ya había sido capaz de entrenarse con un Dataset de este tamaño.
- Las imágenes del Dataset eran de 960x540 pixeles, con este tamaño de imágenes el programa colapsaba en el primer paso y no era posible continuar con el entrenamiento, por eso decidimos ajustar el tamaño a 350x350 pixeles.
- Con el nuevo ajuste la red lograba pasar del primer paso de entrenamiento, sin embargo como cada imagen contiene más de 8 etiquetas, la memoria no era suficiente para completarlo y se decidió reducir el número de etiquetas a máximo 5 por imagen, este arreglo tampoco funciono y la memoria seguía siendo insuficiente.

Análisis de resultados Faster RCNN

El primer Dataset está conformado por imágenes de carreteras en la que se encontraba por lo general un solo vehículo, con una buena resolución, y de tamaño 228x128 pixeles. Entre los vehículos no se encontraban camiones ni buses. Esta prueba fue la que obtuvo mejores resultados por el tipo de imágenes usadas. Como se puede ver en la imagen de prueba del camión, la red reconoce la cabina del camión como un vehículo, pero no reconoce la caja de carga de este ya que en el data set no se encontraban imágenes de estos.

En la imagen de prueba en la que se encuentran varios vehículos se puede ver como detecta mejor los carros más grandes y a medida que el tamaño de los carros disminuye deja de detectarlos, en esta imagen no detecto el camión ni su cabina.

En las pruebas 2 y 3 se utilizaron imágenes de muy poca resolución en las que no se podía distinguir bien el fondo de los bordes del vehículo, además estos Dataset contienen imágenes que enfocan un solo vehículo sin casi nada de fondo por lo que la red no tuvo éxito el detectar los vehículos y clasificarlos.

En la última prueba, aunque fuera un dataset más acorde a la red, se pudo evidenciar que la red ocupa mucha memoria dependiendo de la cantidad de etiquetas puestas en cada imagen. y el dataset necesita de ciertas especificaciones para que se pueda entrenar.

La red toma un tiempo de 45 minutos aproximadamente de entrenamiento con 300 imágenes.

Al aumentar la tasa inicial de muestro a un numero 10 veces más grande el entrenamiento falla.

Recomendaciones Faster RCNN

Se recomienda usar un Dataset de 300 imágenes de un tamaño cercano a 227x227 pixeles en las que haya una gran proporción de fondo a comparación del objeto a detectar, preferiblemente de buena resolución que permita diferenciar el fondo y los bordes de los objetos y con no más de 2 etiquetas por imagen.

Para que la red pueda tomar más de una imagen por iteración en el entrenamiento es necesario que todas las imágenes sean del mismo tamaño, al aumentar el número de imágenes por iteración se reduce considerablemente el tiempo de entrenamiento.

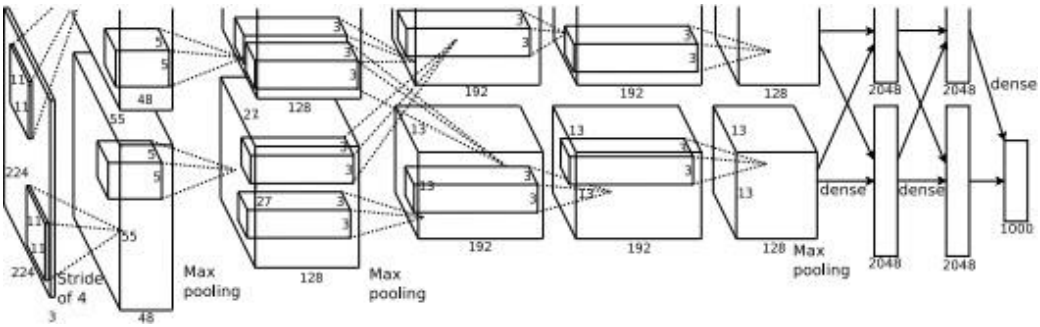
Si se contara con mayor memoria se podría utilizar un Dataset con más imágenes y mayor número de etiquetas por imagen

Alexnet

Alexnet [1] es una red neuronal convolucional capaz de entrenarse con altas cantidades de imágenes, posee una arquitectura de ocho capas y marcó una diferencia en campo de las redes neuronales principalmente gracias a que:

- Utiliza unidades lineales rectificadas (ReLU) las cuales implementan una función de activación lineal que le permite entrenarse mucho más rápido que aquellas redes que presentan funciones exponenciales y tangenciales las cuales requieren más tiempo y espacio de procesamiento.
- Permite realizar el entrenamiento con dos GPUs en paralelo, lo cual también reduce drásticamente el tiempo de convergencia de la red.
- Alexnet agrupa y sobrepone salidas de grupos de neuronas vecinas, lo cual permite reducir el error y hacer más díficil su sobreajuste (exceso de entrenamiento).

la arquitectura de Alexnet según su paper oficial [1] se resume en la siguiente imagen:



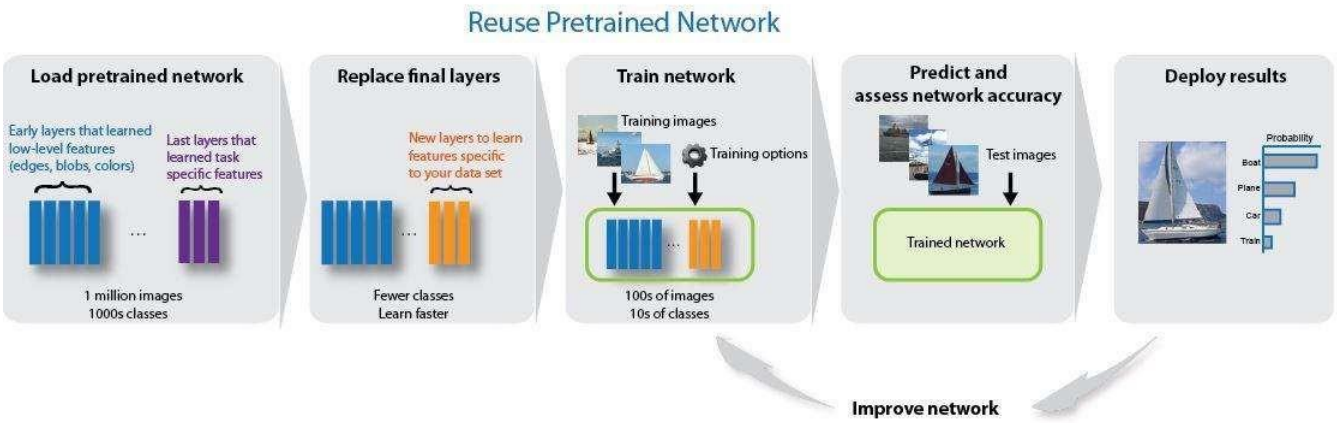
Esta arquitectura le permitió a Alexnet entrenarse con ImageNet un dataset con más de mil clases y 1.2 millones de imágenes y presentar un error de apenas 18% ante una validación de 150.000 imágenes con 1000 clases distintas. La arquitectura Alexnet ha tenido éxito a tal punto de convertirse en una de las más utilizadas en el campo de la inteligencia artificial.

[1] https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Uso y acondicionamiento de red pre entrenada Alexnet

Alexnet es una red neuronal pre entrenada con millones de imágenes, sin embargo, es necesario realizar un proceso de acondicionamiento para los fines de nuestro proyecto, a continuación se explica el método gráfico hallado en (Mathworks.com, s.f.).

Este método se lleva a cabo realizando los siguientes pasos:



Cargar red pre entrenada

Cargar la red se lleva a cabo ejecutando el comando:

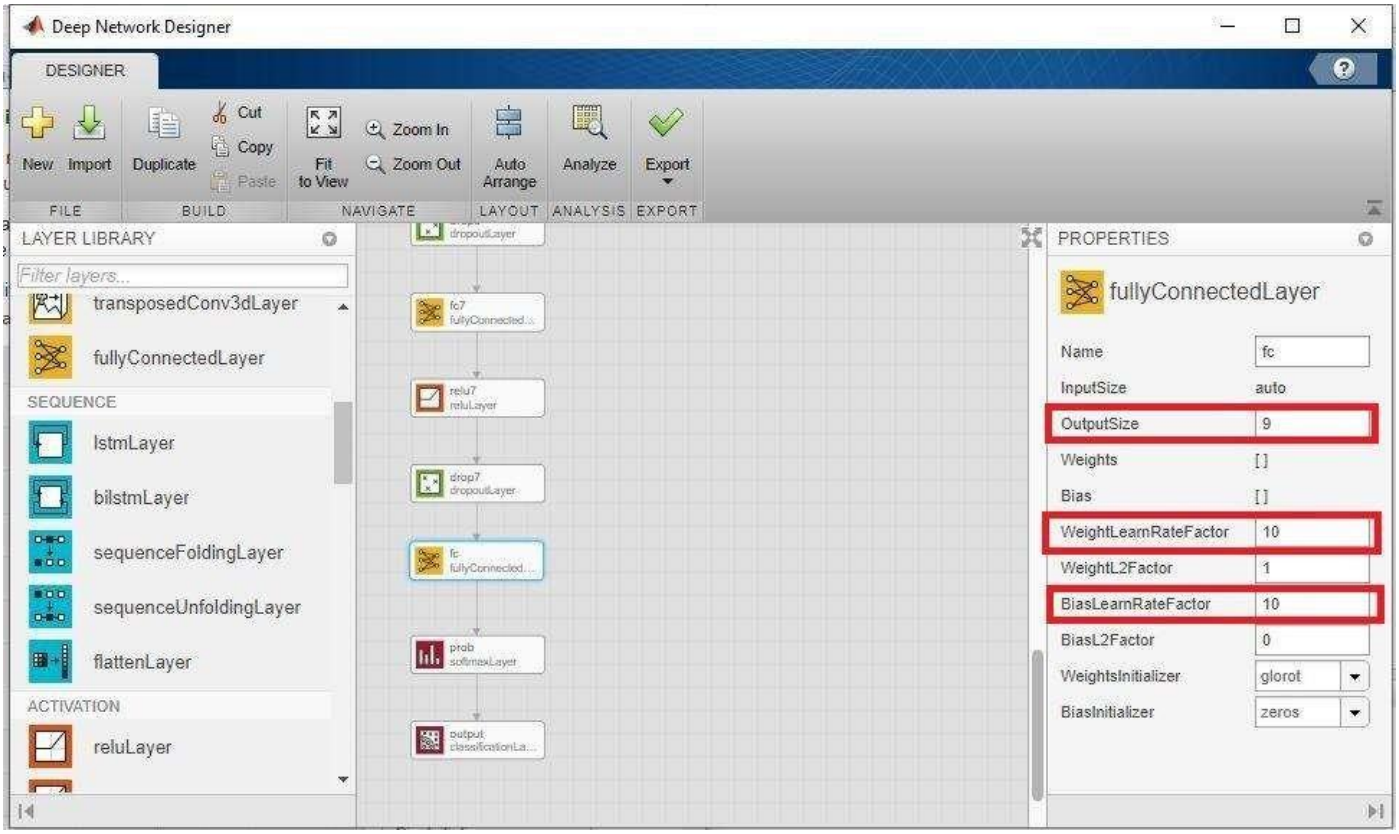
```
% Net=alexnet
```

Reemplazar las capas finales

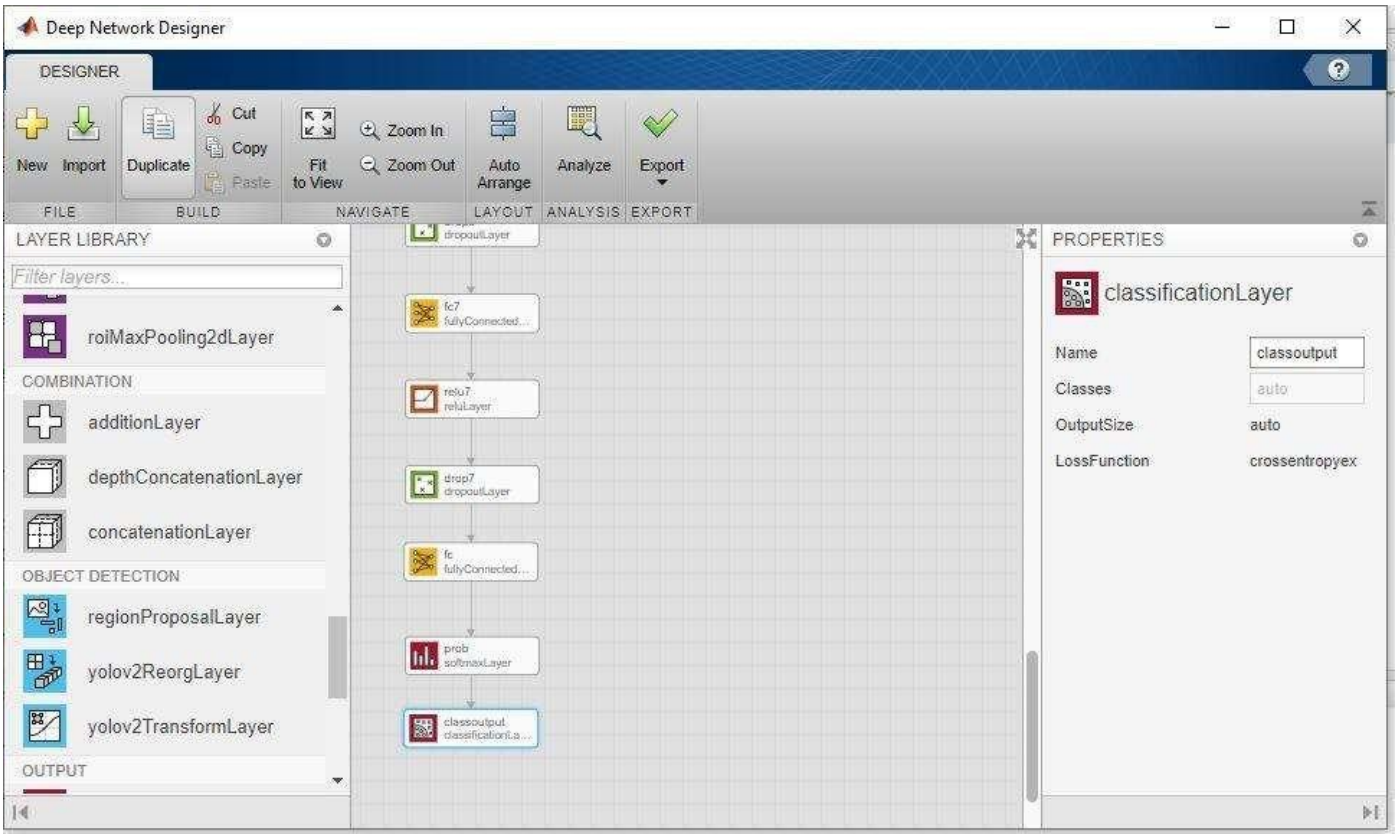
Para esto se debe abrir el deepnetworkdesigner

```
% deepNetworkDesigner
```

una vez dentro de la app se debe importar la red, también es necesario cambiar las capas que se muestran a continuación:



La capa que estaba en esta posición es reemplazada por una *FullyConnectedLayer* en la cual se puede observar que se han cambiado sus parámetros de aprendizaje y su número de salidas, en esto caso usaremos 9 tipos distintos de cehículos, razón por la cual OutputSize=9.



La última capa también debe ser reemplazada por una *ClassificationLayer* la cual no tiene ningún

cambio adicional.

Al terminar los cambios se debe hacer clic en "exportar" lo cual dejará la nueva red (Layers_1) en nuestro workspace.

Entrenar la red

El siguiente paso consiste en entrenar la nueva red, lo primero que se debe hacer es escoger el dataset que vamos a utilizar y crear un almacén de datos con el mismo

```
imds = imageDatastore('train3','IncludeSubfolders',true,'LabelSource','foldernames');
```

Hecho esto procederemos a dividir nuestro Dataset entre las imágenes de validación y las de entrenamiento:

```
[imdsTrain,imdsValidation] = splitEachLabel(imds,0.8,'randomized');
```

en este caso el 0.8 quiere decir que el 80% de las imágenes son de entrenamiendo y el 20% restante son imágenes de validación.

Cada red neuronal tiene un tamaño de imagen preterminado de entrada, por esta razón es necesario acondicionar todas las imágenes a un tamaño fijo (Alexnet =227*227)

```
augimdsTrain = augmentedImageDatastore([227 227],imdsTrain);
augimdsValidation = augmentedImageDatastore([227 227],imdsValidation);
```

Con esto hemos finalizado el acondicionamiento de nuestro dataset, ahora debemos especificar las opciones de entrenamiento:

```
miniBatchSize = 10;
```

Especifica el número de imágenes que se usará por cada iteración

%valFrequency = floor(numel(augimdsTrain.Files)/miniBatchSize);

Define una frecuencia de validación, en este caso es una apróximación al número total de iteraciones (imágenes/minimumBatch). Después procedemos a crear el archivo de opciones.

```
options = trainingOptions('sgdm', ...
    'MiniBatchSize',miniBatchSize, ...
    'MaxEpochs',5, ...
    'InitialLearnRate',3e-4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',augimdsValidation, ...
    'ValidationFrequency',valFrequency, ...
    'Verbose',false, ...
    'Plots','training-progress');
```

Donde

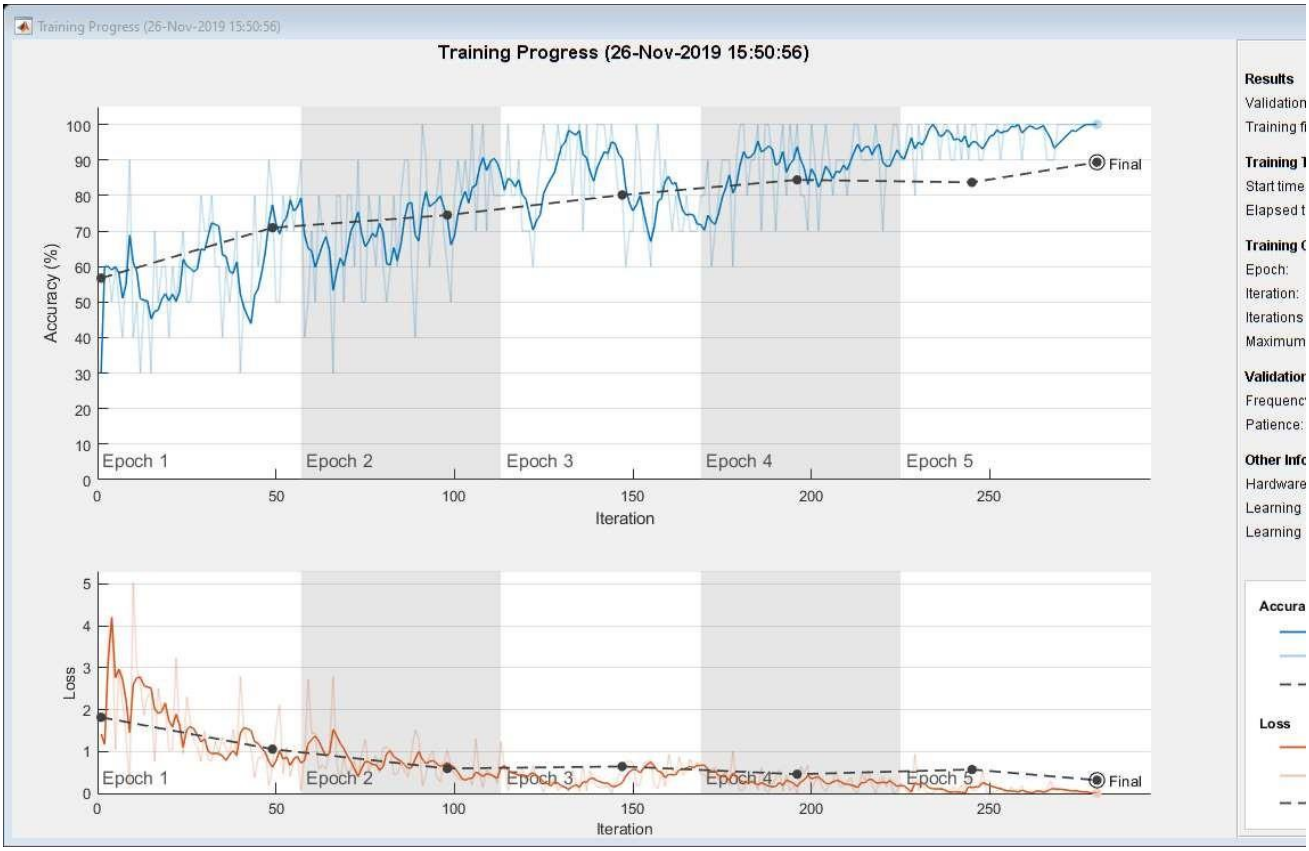
'*InitialLearnRate*' especifica la tasa inicial de aprendizaje.

'MaxEpoch' define la cantidad de veces que la base de datos pasará por la red.

'Shuffle' indica que se mezclarán las imágenes en cada época.

Una vez hemos definido el archivo de opciones procedemos a realizar el entrenamiento de la red

```
trainedNet = trainNetwork(augimdsTrain,layers_1,options);
```



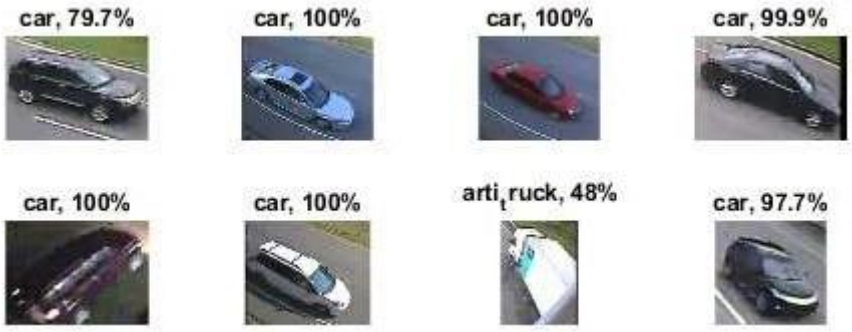
Este proceso tomará un tiempo, el cual se describe más adelante. Hecho esto se procede a realizar una validación de la red clasificando las imágenes de validación y calculando el promedio de acierto de la red

```
[YPred,probs] = classify(trainedNet,augimdsValidation);
accuracy = mean(YPred == imdsValidation.Labels)
```

```
accuracy = 0.8936
```

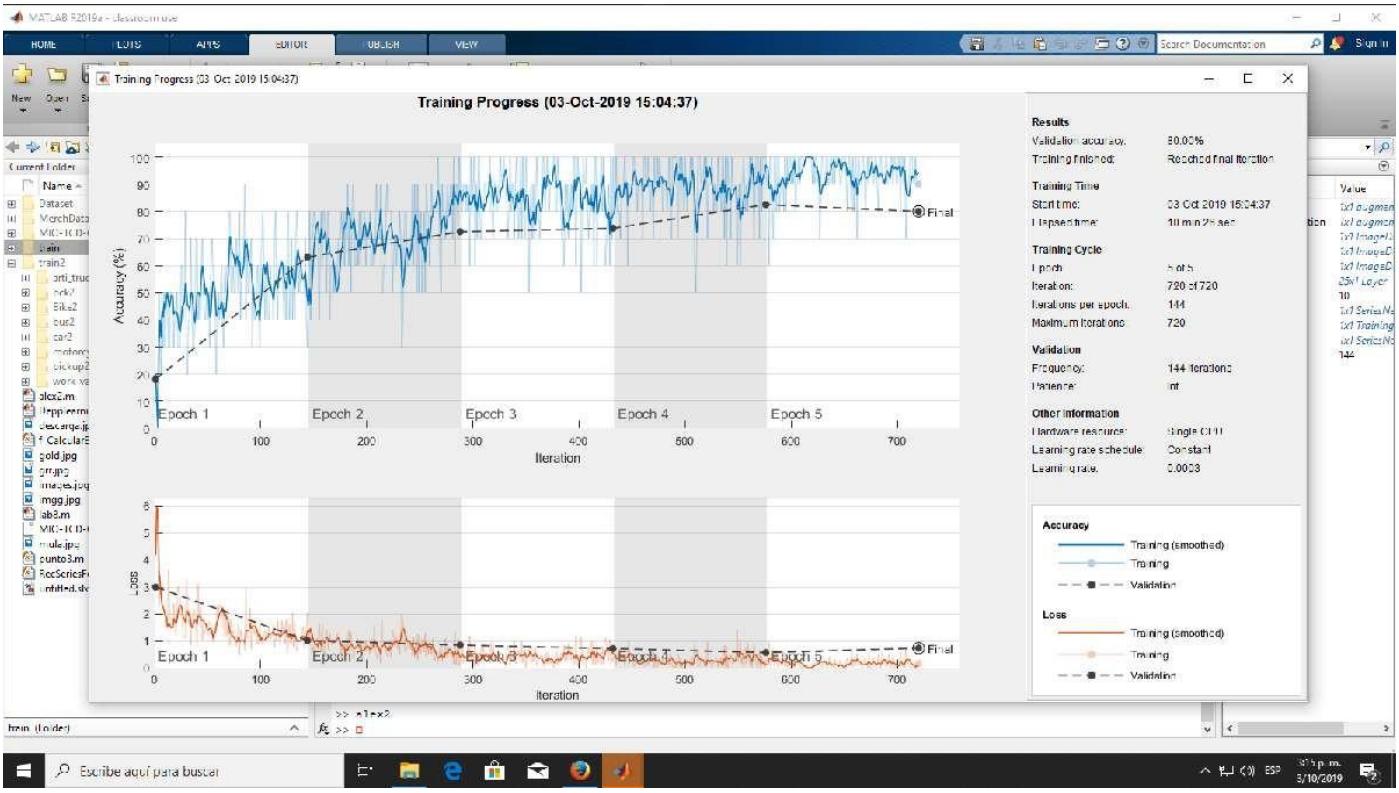
Teniendo estos valores graficamos una clasificación de muestra

```
idx = randperm(numel(imdsValidation.Files),8);
figure
for i = 1:8
subplot(4,4,i)
I = readimage(imdsValidation,idx(i));
imshow(I)
label = YPred(idx(i));
title(string(label) +", "+ num2str(100*max(probs(idx(i),:)),3) +"%");
end
```



Los resultados de llevar a cabo este proceso con Datasets de diferentes tamaños fueron los siguientes:

1) Con 200 imágenes por carpeta (1800 imágenes) se utilizó el 80% de imágenes ára entrenar la red y el 20% para validarla, esto da como resultado 1800*0.8=1440 imágenes de entrenamiento, lo cual corresponde a 1440imágenes*5épocas/10imágenes por iteración= 720 iteraciones que se llevaron a cabo en 10 minutos y 25 segundos.



con una precisión de 81.5%.

Gráfico de imágenes de validación:

1) Con 400 imágenes por carpeta (3600 imágenes) se utilizó el 90% de imágenes ára entrenar la red y el 10% para validarla, esto da como resultado 3600*0.9=3240 imágenes de entrenamiento, lo cual corresponde a 3240imágenes*5épocas/10imágenes por iteración= 1620 iteraciones que se llevaron a cabo en 22 minutos y 26 segundos.

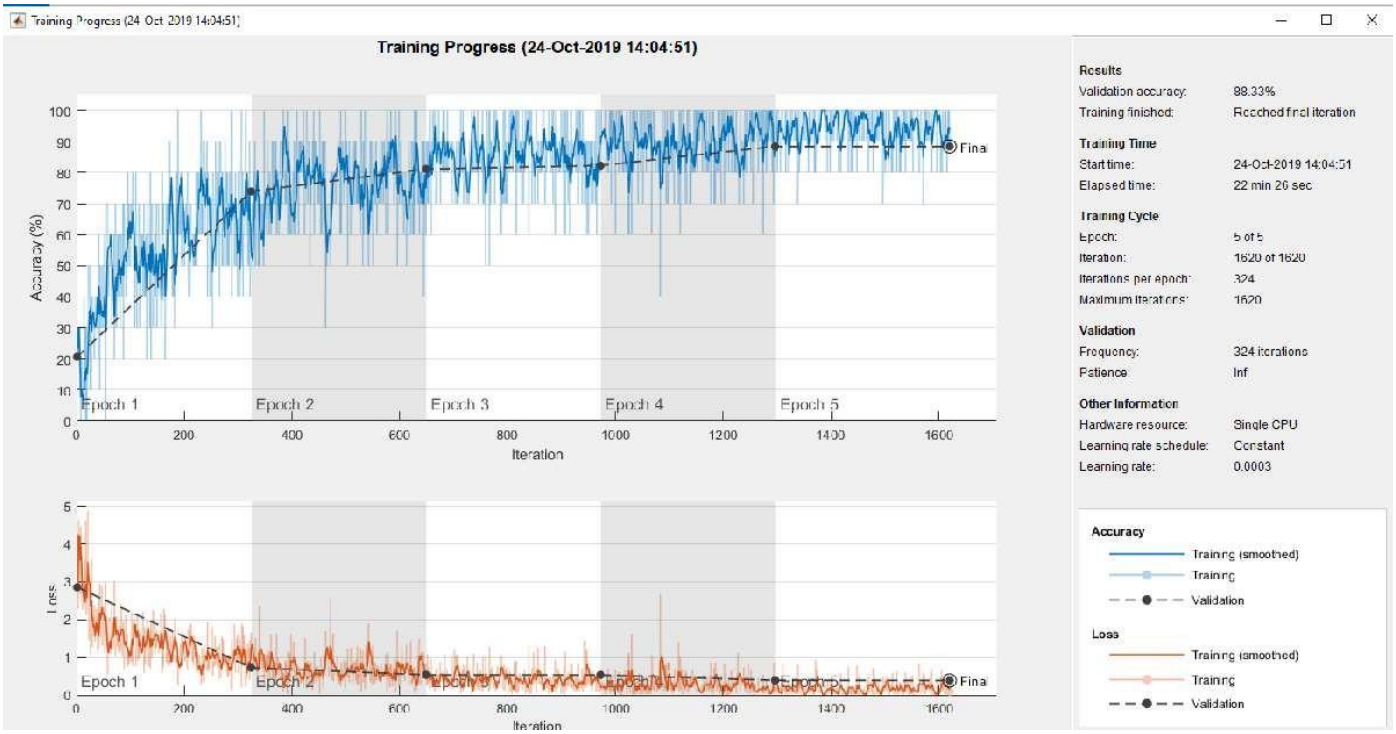
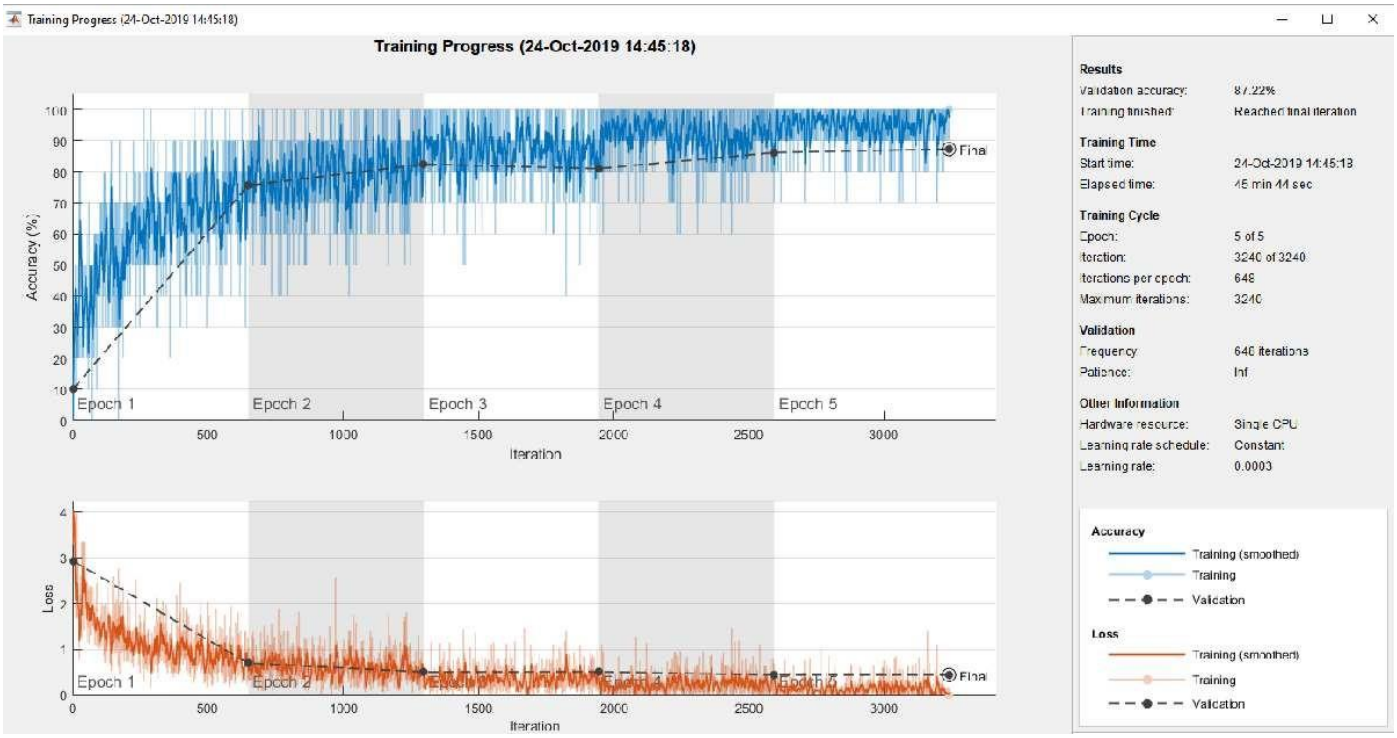


Gráfico de imágenes de validación:

1) Con 800 imágenes por carpeta (7200 imágenes) se utilizó el 90% de imágenes ára entrenar la red y el 10% para validarla, esto da como resultado 7200*0.9=6480 imágenes de entrenamiento, lo cual corresponde a 6480imágenes*5épocas/10imágenes por iteración= 3240 iteraciones que se llevaron a cabo en 45 minutos y 44 segundos.



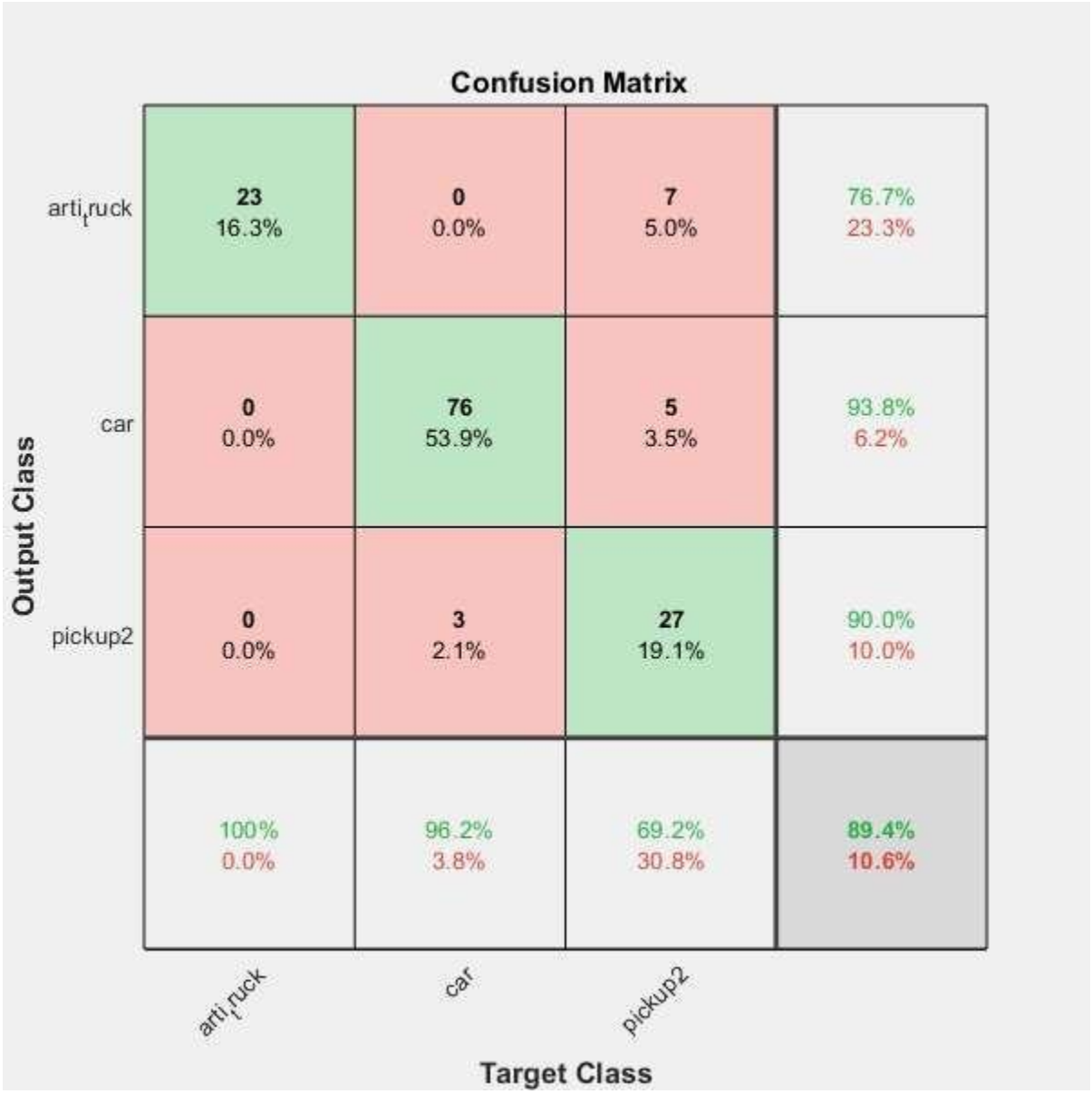
con una precisión de 87.22%.

Gráfico de imágenes de validación:

Se realizó una última prueba con 703 archivos y 3 categorías la cual obtuvo como resultado

con un 91% de precisión. Por otra parte se imprimió la matriz de confusión

```
plotconfusion(classify(trainedNet,augimdsValidation),imdsValidation.Labels)
```

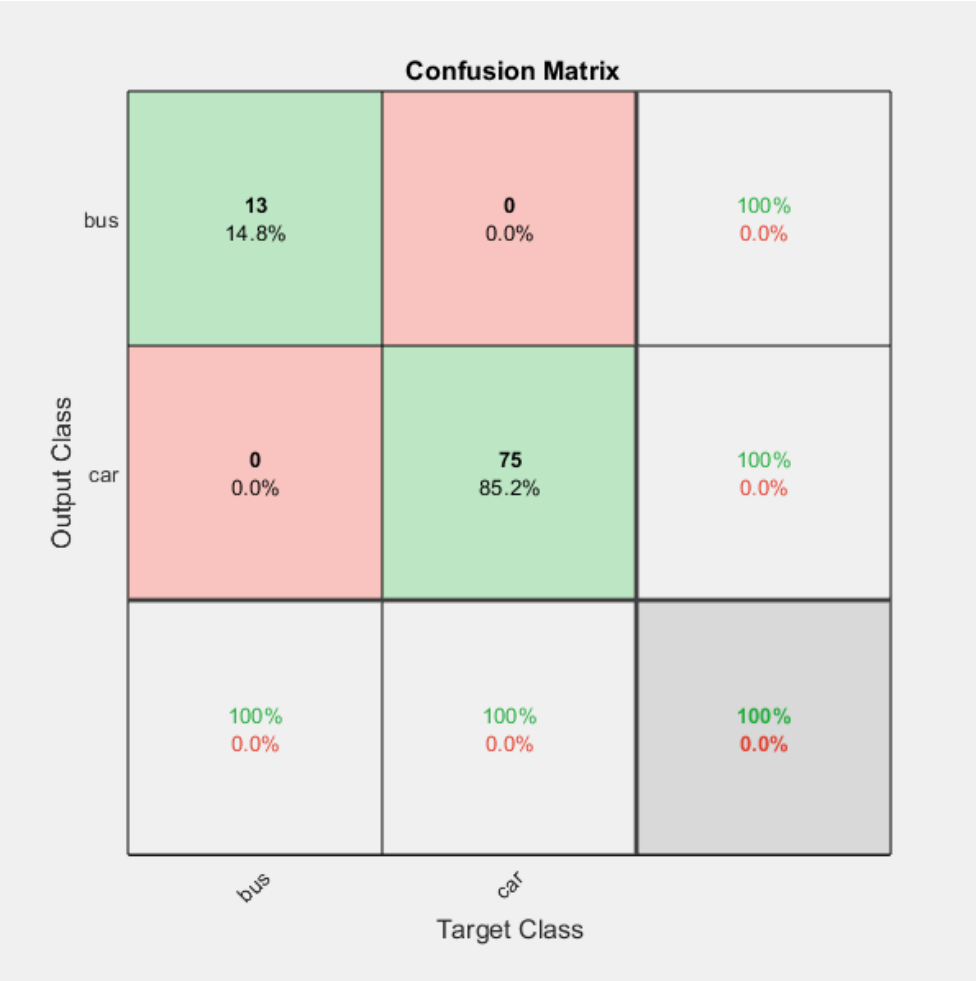

Confusion Matrix
artiₜruck
car
pickup2
Output Class
23
16.3%
0
0.0%
7
5.0%
76.7%
23.3%
0
0.0%
76
53.9%
5
3.5%
93.8%
6.2%
0
0.0%
3
2.1%
27
19.1%
90.0%
10.0%
100%
0.0%
96.2%
3.8%
69.2%
30.8%
89.4%
10.6%
artiₜruck
car
pickup2
Target Class

Análisis de resultados Alexnet

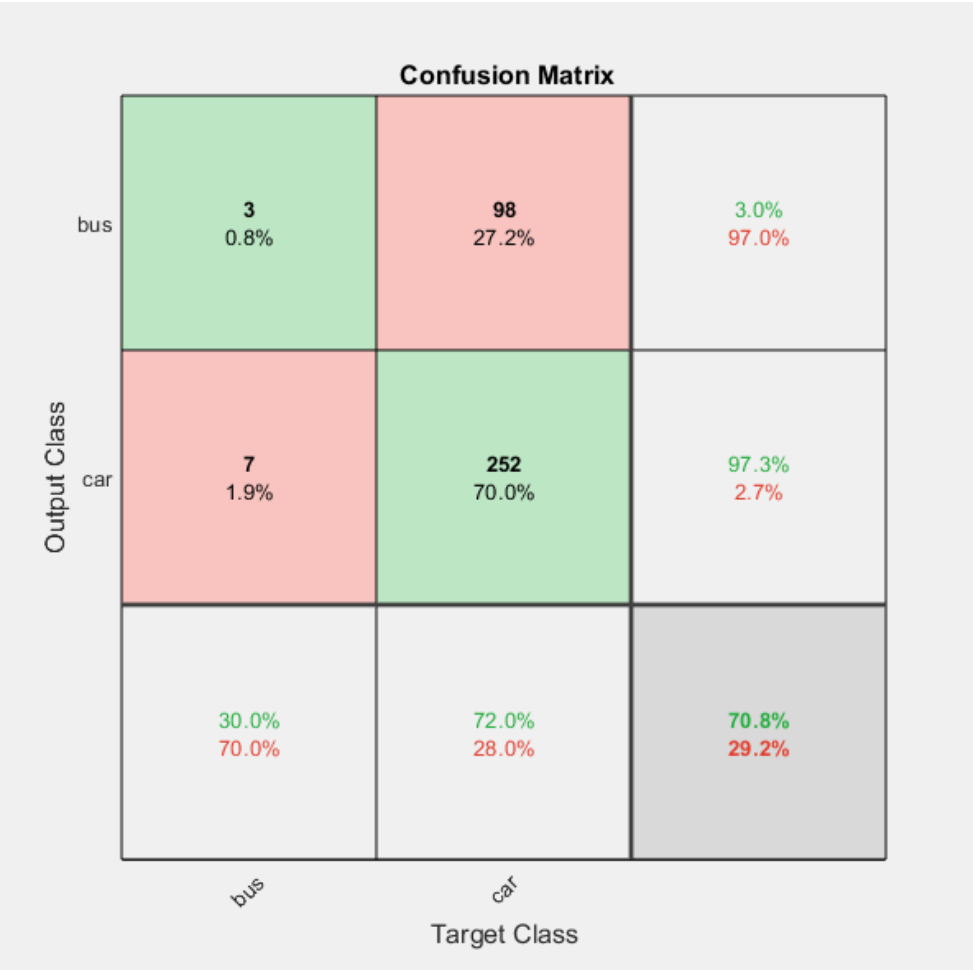
El entrenamiento de la red bajo distintos conjuntos y número de imágenes se resume en la siguiente tabla:

N° imágenes/Nº clases	Tiempo por iteración	Precisión	Tiempo total
1800 /9	0,86 seg	81,5%	10 min 25 seg
3600 /9	0,83 seg		22 min 26 seg
7200 /9	0,85 seg	87,22%	45min 44 seg
703 /3		91%	

Adicionalmente se realizó el entrenamiento de la red con un dataset que contiene más de una categoría por imagen (utilizando la función de recortar descrita anteriormente), este entrenamiento se dio con dos diferentes categorías y arrojó las siguientes matrices de confusión:



Cuando las imágenes de validación provienen del mismo dataset de entrenamiento



Cuando las imágenes de validación provienen de un dataset distinto al de entrenamiento

Comparación de resultados:

Se realizó el entrenamiento de ambas redes con el mismo Dataset que contenía dos categorías con una sola etiqueta cada imagen, dichas categorías eran *Arti Truck* y *Pickup.*

Los resultados fueron los siguientes:

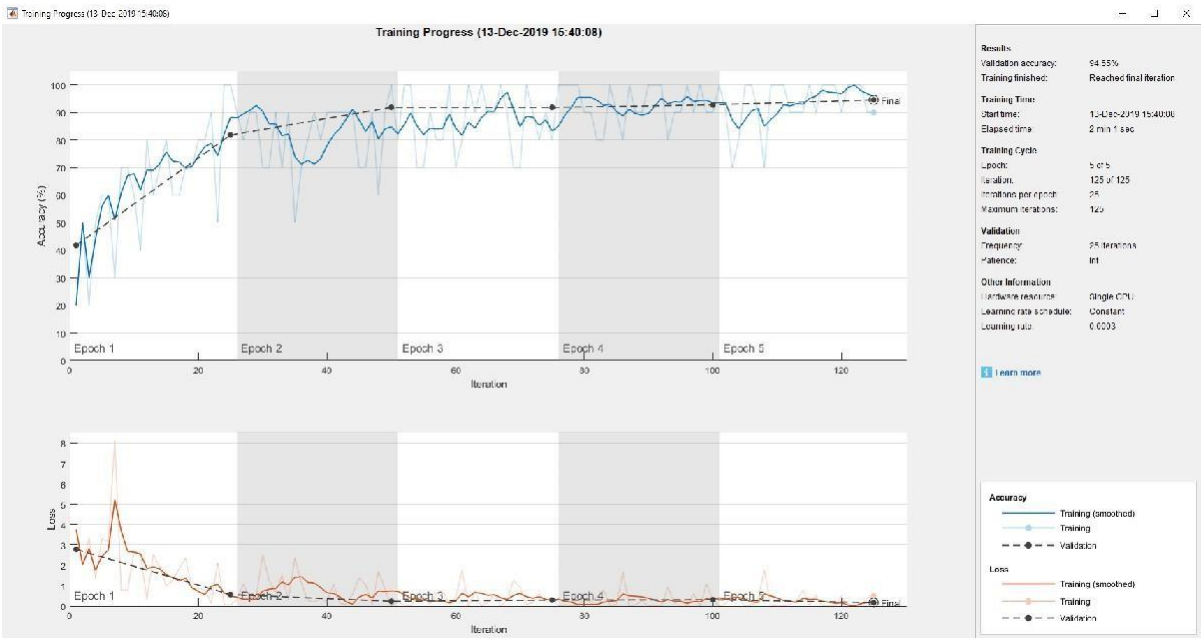
Tiempo de entrenamiento:

- ResNet50

```
Training a Faster R-CNN Object Detector for the following object classes:

* Pickup
* Truck

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single CPU.
|=========================================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     Loss     |   Accuracy   |     RMSE     |       Rate      |
|=========================================================================================================|
|       1 |           1 |       00:00:08 |      26.6481 |       56.72% |         1.33 |          0.0010 |
|       5 |          50 |       00:07:34 |       1.8205 |       84.47% |         0.27 |          0.0010 |
|       7 |          70 |       00:10:36 |       1.2332 |       85.43% |         0.22 |          0.0010 |
|=========================================================================================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 207 training images...done.

Training on single CPU.
|=========================================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     Loss     |   Accuracy   |     RMSE     |       Rate      |
|=========================================================================================================|
|       1 |           1 |       00:00:13 |      23.4805 |       37.21% |         0.85 |          0.0010 |
|       5 |          50 |       00:11:54 |       5.8929 |       68.52% |         0.44 |          0.0010 |
|       7 |          70 |       00:16:42 |       5.6243 |       82.72% |         0.44 |          0.0010 |
|=========================================================================================================|

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single CPU.
|=========================================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     Loss     |   Accuracy   |     RMSE     |       Rate      |
|=========================================================================================================|
|       1 |           1 |       00:00:05 |       6.0538 |       83.04% |         0.55 |          0.0010 |
|       5 |          50 |       00:05:52 |       2.5921 |       85.47% |         0.34 |          0.0010 |
|       7 |          70 |       00:08:15 |       2.1766 |       85.03% |         0.31 |          0.0010 |
|=========================================================================================================|

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 207 training images...done.

Training on single CPU.
|=========================================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     Loss     |   Accuracy   |     RMSE     |       Rate      |
|=========================================================================================================|
|       1 |           1 |       00:00:10 |       9.6498 |       66.39% |         0.48 |          0.0010 |
|       5 |          50 |       00:09:14 |       6.1215 |       84.29% |         0.36 |          0.0010 |
|       7 |          70 |       00:12:57 |       2.6445 |       87.14% |         0.23 |          0.0010 |
|=========================================================================================================|

Detector training complete
```

46 minutos.

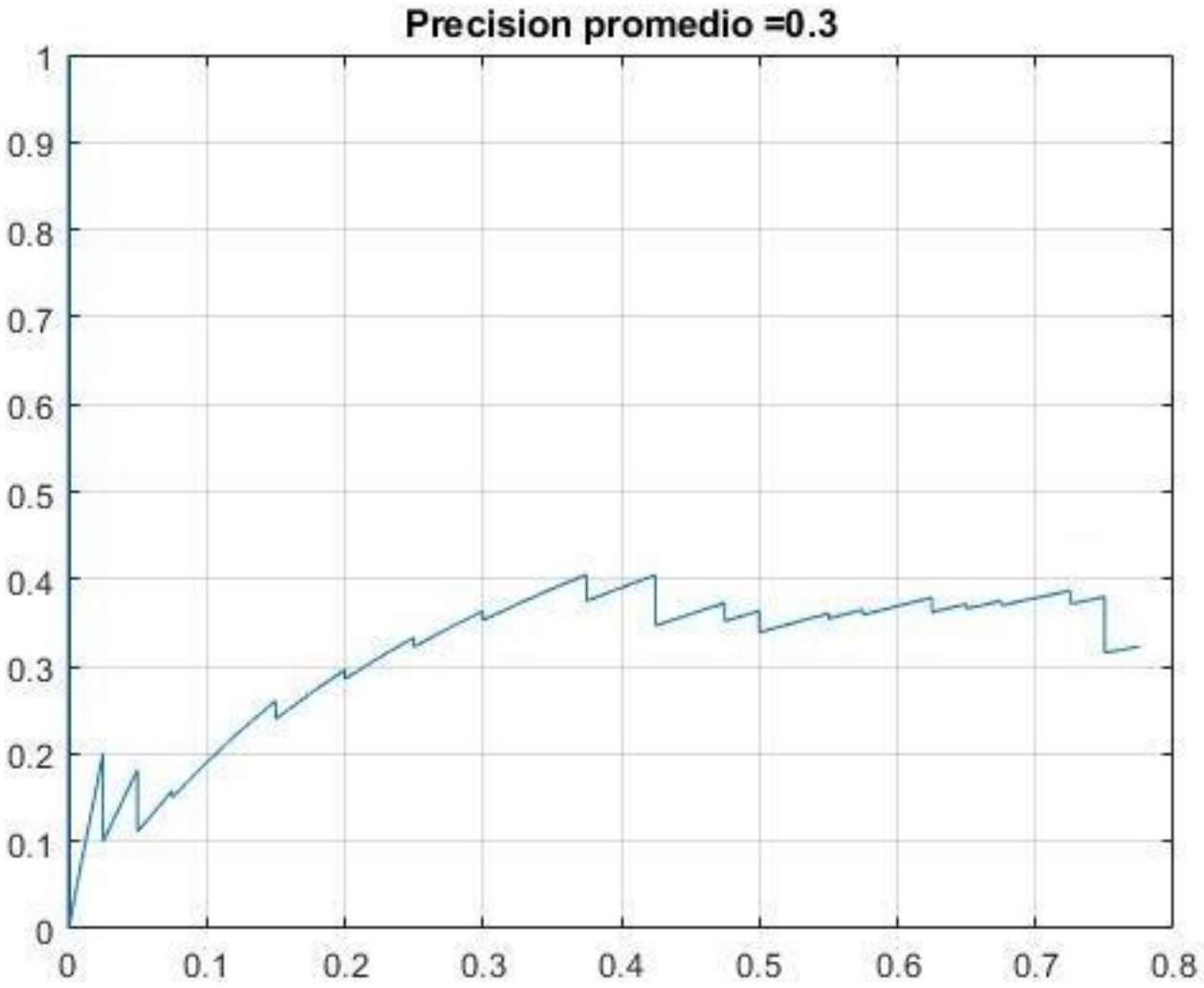
- Alexnet



2 minutos.

Precisión:

- ResNet50



30%

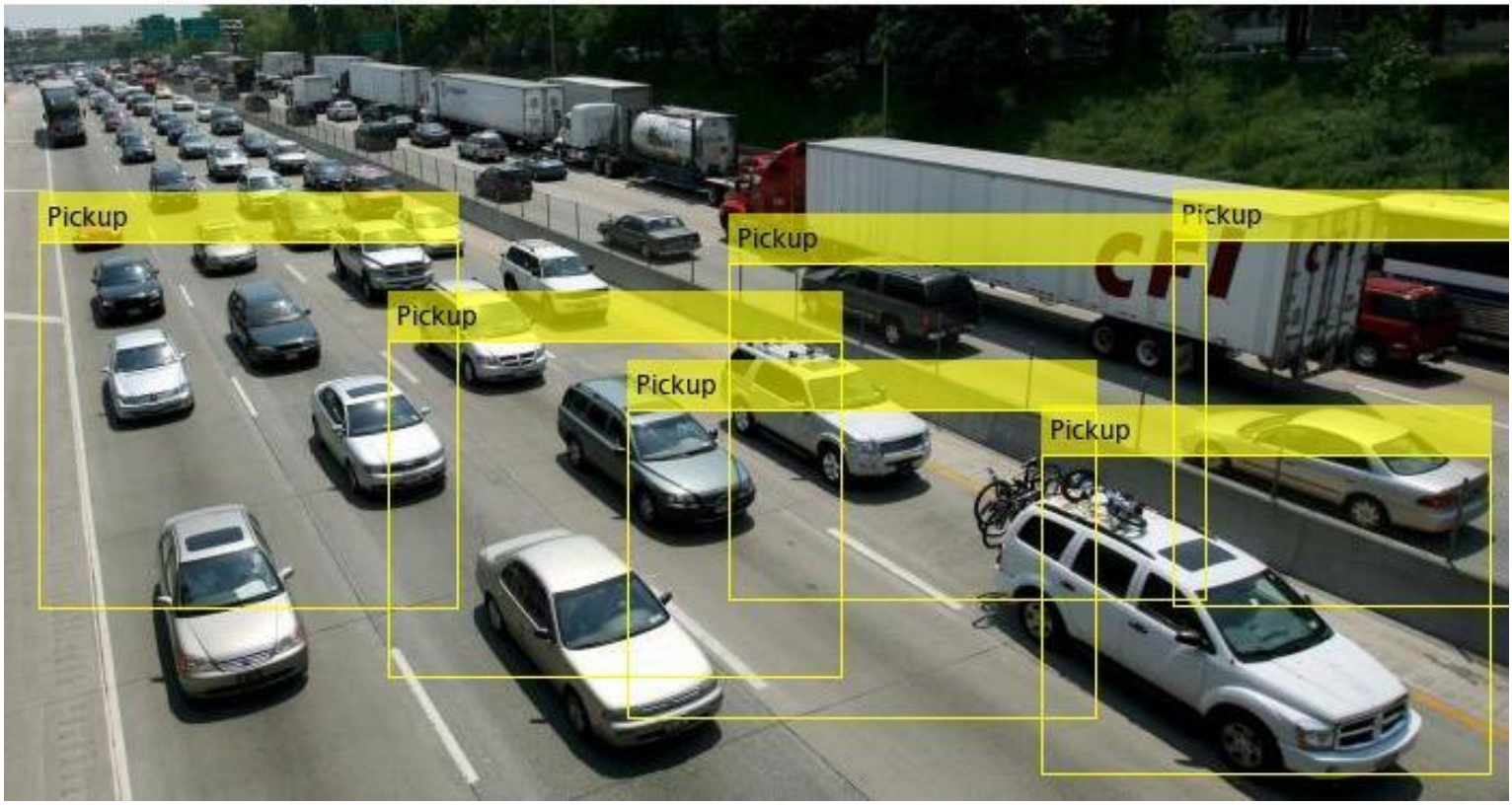
- Alexnet:

```
>> accuracy

accuracy =

    0.9455
```

94.5%

Resultados:

- ResNet50



- Alexnet

pickup2, 100%

arti$_t$ruck, 100%

arti$_t$ruck, 87%

arti$_t$ruck, 100%

pickup2, 80.5%

arti$_t$ruck, 100%

pickup2, 67.6%

pickup2, 100%

Conclusiones:

La red Faster RCNN funciona como un detector de objetos en imágenes, por lo cual puede trabajar con imágenes que contengan más de un objeto en ellas, a diferencia de Alexnet, sin embrago Faster RCNN toma mucho más tiempo para entrenarse que Alexnet y necesita de un mayor acondicionamiento para las imágenes a usar, además de mayor memoria, lo cual la hace un poco más complicada para trabajar.

La arquitectura de Alexnet presenta los errores más bajos en la actualidad, sin embargo, debido a su alta cantidad de parámetros se encuentra propensa al sobreajuste. Por esta razón es recomendable realizar el entrenamiento de esta red con Datasets que posean gran variedad de imágenes de una misma categoría (imágenes aleatorias) en lugar de Datasets que presenten la misma imagen en diferente posición (archivos de video).

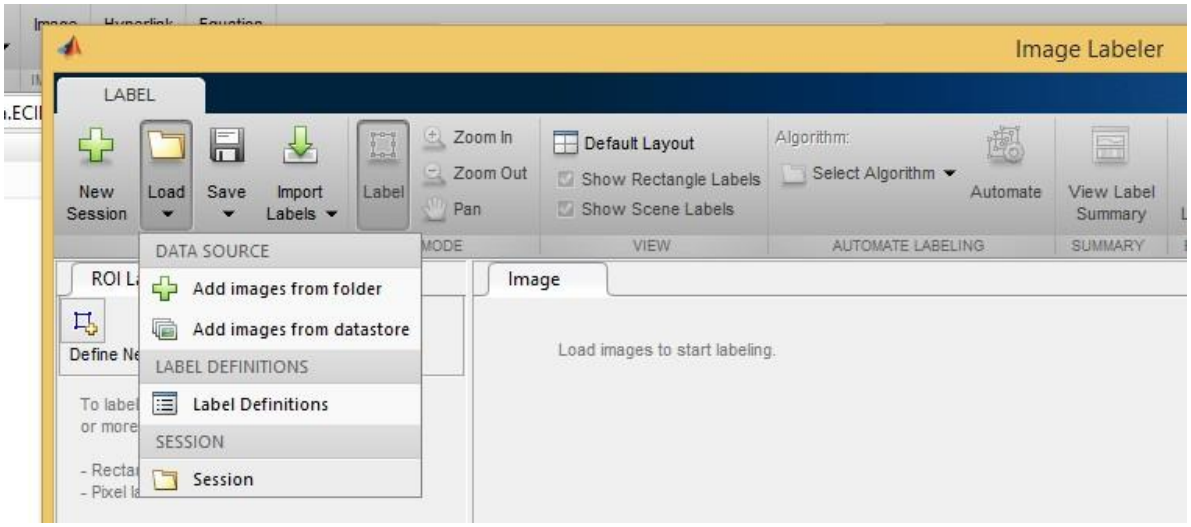
Se hace hincapié en la necesidad de entrenar Alexnet con una sola categoría por imagen ya que la estructura de la red así lo sugiere, por esta razón para su utilización es conveniente apoyarse primero en redes como ResNet50 capaces de hacer reconocimiento de diferentes objetos sobre una misma imagen

Aunque Faster RCNN no arrojo buenos resultados cuando se entrenó para detectar más de una categoría, esto puede deberse al que el Dataset usado no fue el correcto para la red.

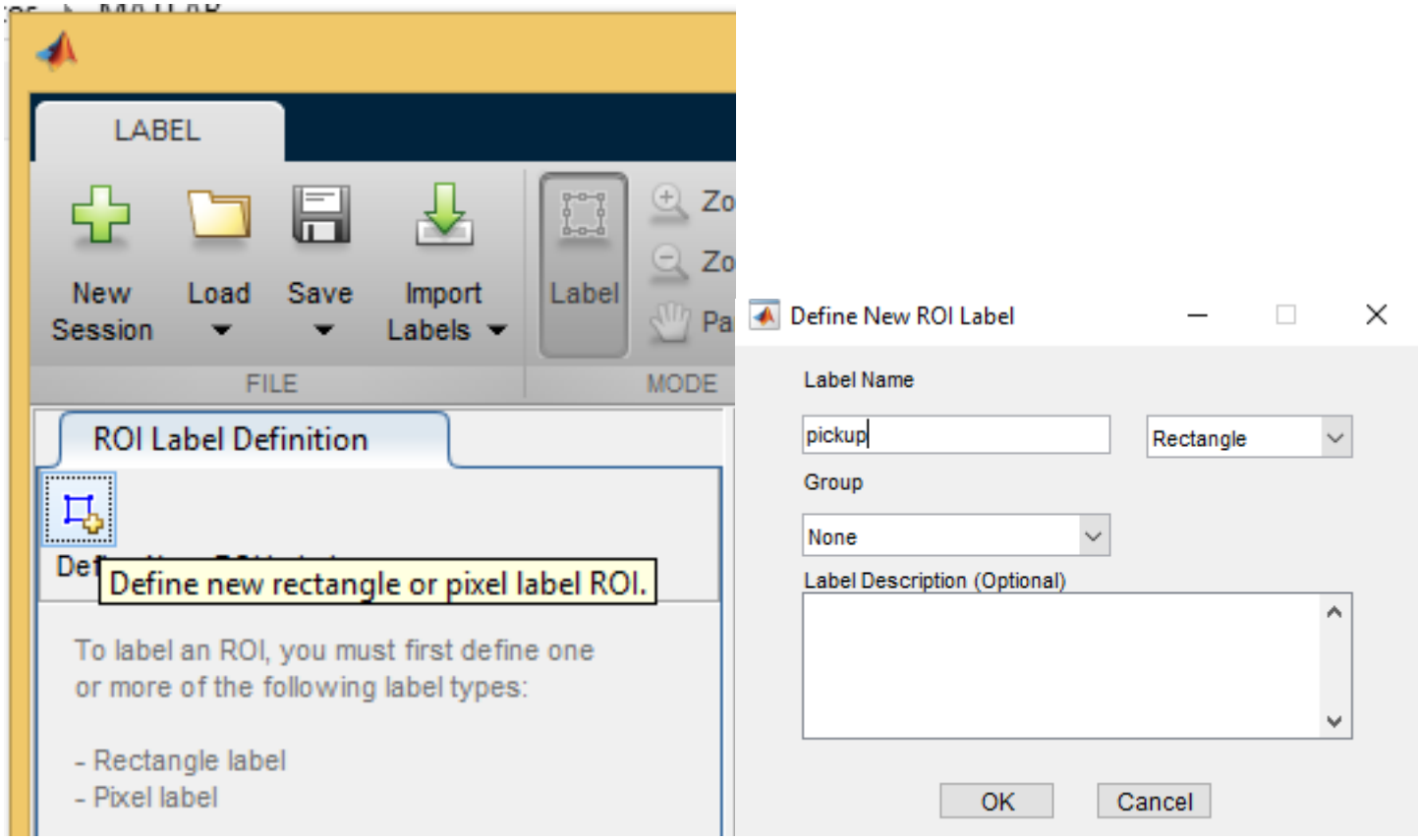
Acondicionamiento de imágenes

Para llevar a cabo el entrenamiento de la red ResNet50 se realizó el etiquetado de diferentes imágenes provenientes de distintos Datasets mediante la herramienta "*Image labeler*" tal y como se muestra a continuación:

1. Ejecutar Image Labeler desde la barra de aplicaciones o mediante el comando "imageLabeler".
2. Seleccionar y cargar las imágenes a etiquetar

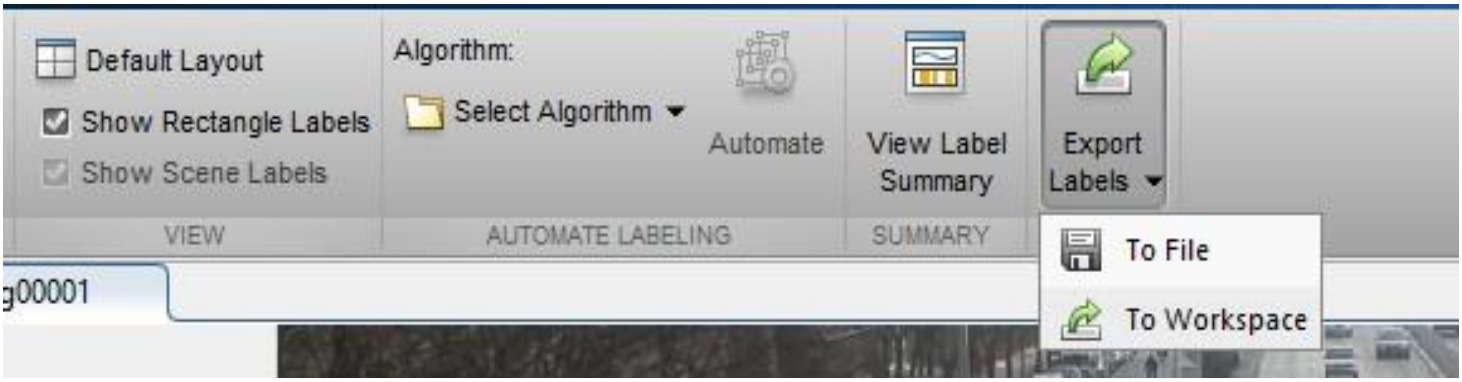


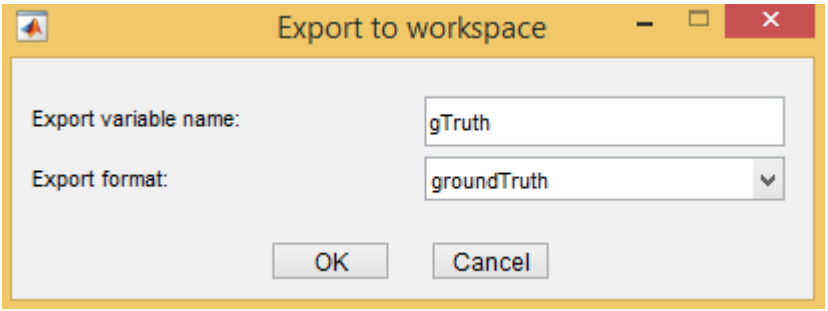
3. Definir el nombre de las etiquetas, en este caso se utilizarán "truck" y "pickup" como categorías.



4. Proceder al etiquetado del Dataset

5. Una vez se encuentren etiquetadas todas las imágenes se procederá a exportar las etiquetas al *workspace*, lo cual generará como resultado un archivo gTruth.





El archivo gTruth (Ahora localizado en el Workspace) constará de las siguientes partes:

gTruth

1x1 groundTruth

Property	Value
DataSource	*1x1 groundTruthData...*
LabelDefinitions	*2x4 table*
LabelData	*936x2 table*

- Data Source:

gTruth.DataSource.Source

	1	2	3	4
1	C:\Users\ELECTRONICA.ECINUT\Documents\MATLAB\train3\pickup2\00000001.jpg			
2	C:\Users\ELECTRONICA.ECINUT\Documents\MATLAB\train3\pickup2\00000035.jpg			
3	C:\Users\ELECTRONICA.ECINUT\Documents\MATLAB\train3\pickup2\00000047.jpg			
4	C:\Users\ELECTRONICA.ECINUT\Documents\MATLAB\train3\pickup2\00000060.jpg			
5	C:\Users\ELECTRONICA.ECINUT\Documents\MATLAB\train3\pickup2\00000063.jpg			

Contiene la ruta de cada imagen dentro del PC.

- Label Definitions:

gTruth.LabelDefinitions

	1 Name	2 Type	3 Group	4 Description	5
1	'pickup'	'0'	'None'	''	
2	'truck'	'0'	'None'	''	
3					
4					

Es un elemento de tipo tabla en el cual se encuentran las definiciones, el tipo y la descripción de cada etiqueta.

- Label Data:

gTruth | gTruth.LabelDefinitions | gTruth.LabelData

gTruth.LabelData

	1 pickup	2 truck	3	4	5
1	[16,49,358,2...	[]			
2	[21,11,101,68]	[]			
3	[14,39,357,2...	[]			
4	[6,6,119,62]	[]			
5	[36,43,341,2...	[]			

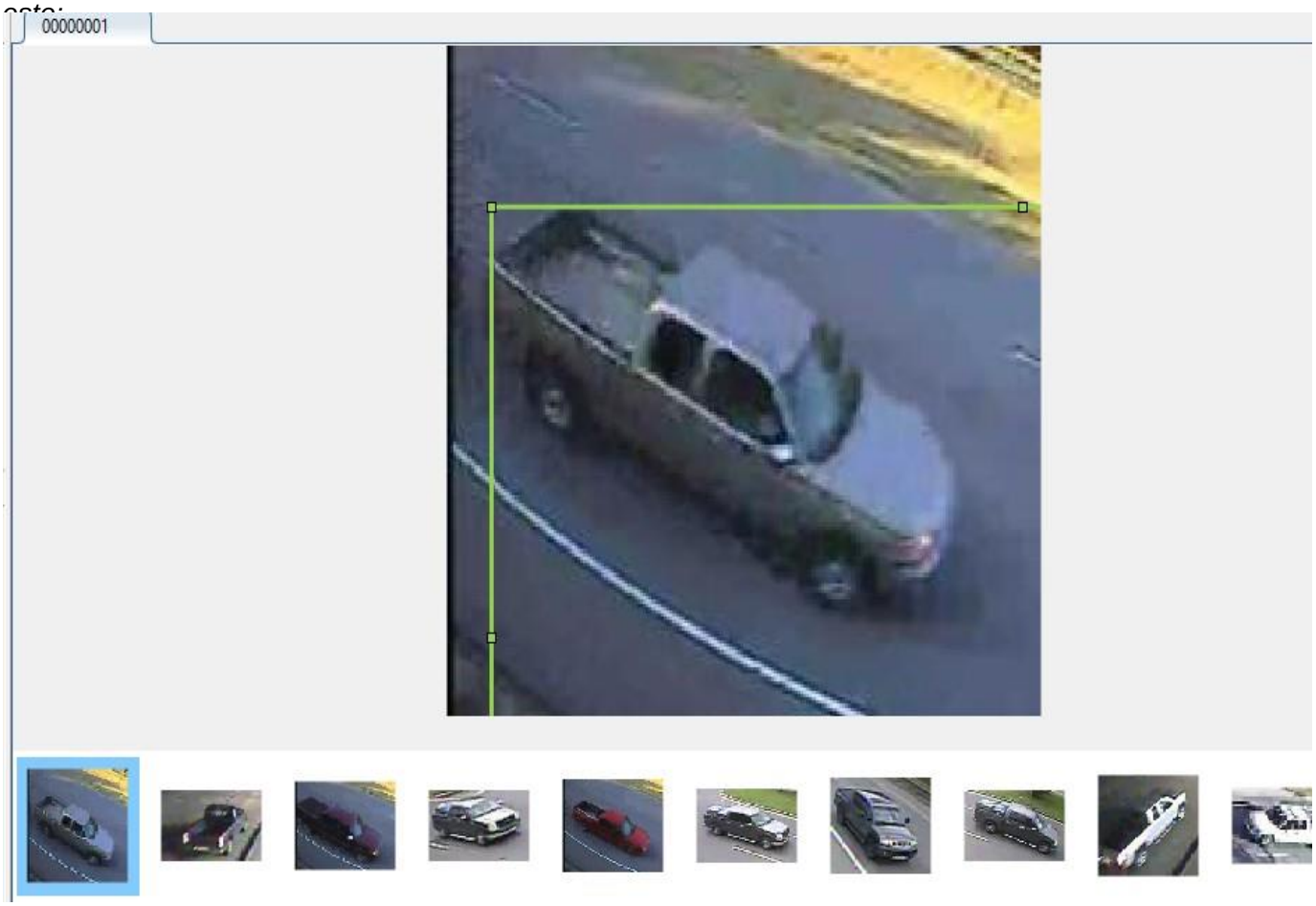
Contiene un vector o arreglo de vectores (si se presenta más de una etiqueta del mismo tipo por imagen) con la información de las etiquetas, cada vector cuenta con cuatro datos distribuidos en el siguiente orden:

- Coordenada del píxel en X de la esquina superior izquierda.
- Coordenada del píxel en Y de la esquina superior izquierda.
- Ancho en píxeles.
- Alto en píxeles.

Conociendo la estructura del elemento gTruth nos permitimos realizar algunas modificaciones:

Cambiar el tamaño de las imágenes (incluyendo etiquetas):

Algunas veces fue necesario ajustar el tamaño de las imágenes para entrenar la red ResNet50, pero llevar a cabo únicamente la implementación de la función *"Resize" nos llevaría a un resultado como esto:*



Por esta razón es necesario llevar a cabo el ajuste de cada vector de etiquetas, dicho ajuste se logró mediante la implementación del siguiente código:

El primer paso consiste en extraer en una tabla la información del archivo gTruth:

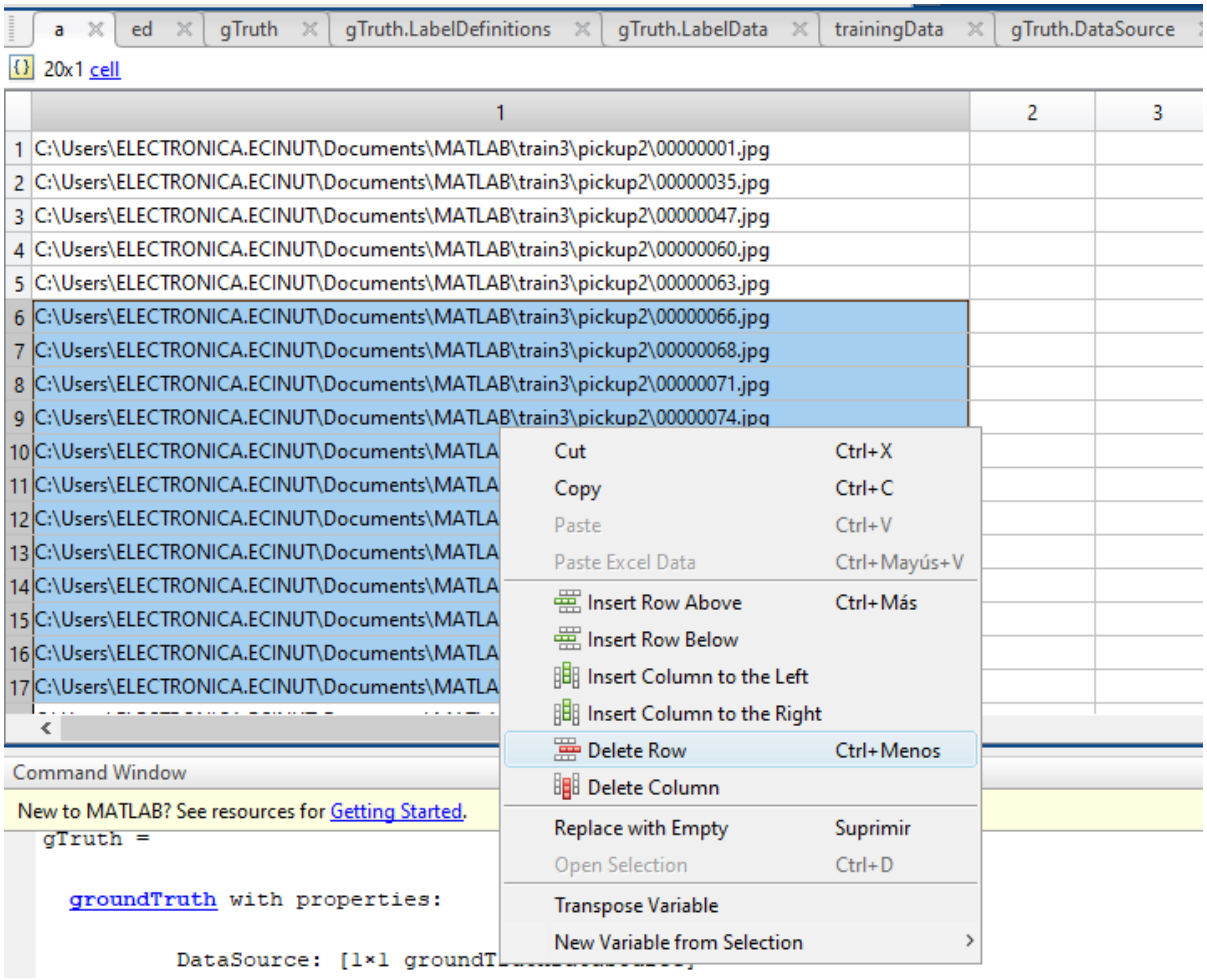
```
trainingData =  objectDetectorTrainingData(gTruth);
```

la cual tiene como resultado:

	1 imageFilename	2 pickup	3 truck	4
1	'C:\Users\ELECTRONICA.ECINUT\Documents\MATLAB\train3\pickup2\00000001.jpg'	[16,49,358,2...	[]	
2	'C:\Users\ELECTRONICA.ECINUT\Documents\MATLAB\train3\pickup2\00000035.jpg'	[21,11,101,68]	[]	
3	'C:\Users\ELECTRONICA.ECINUT\Documents\MATLAB\train3\pickup2\00000047.jpg'	[14,39,357,2...	[]	
4	'C:\Users\ELECTRONICA.ECINUT\Documents\MATLAB\train3\pickup2\00000060.jpg'	[6,6,119,62]	[]	
5	'C:\Users\ELECTRONICA.ECINUT\Documents\MATLAB\train3\pickup2\00000063.jpg'	[36,43,341,2...	[]	

Con esta información podemos realizar el ajuste de las imágenes, en este ejemplo se hará para las primeras 5 imágenes, por esta razón es necesario crear un nuevo Dataset con las primeras cinco rutas:

```
a=gTruth.DataSource.Source;
```



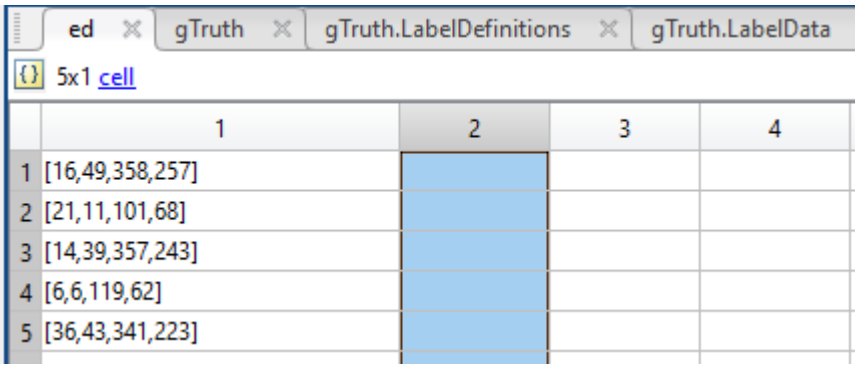
se eliminan los elementos restantes para construir un nuevo Data Source con las imágenes necesarias:

```
new_DataSource=groundTruthDataSource(a);
```

Hecho esto procedemos a ajustar el tamaño de las imágenes

```
for i=1:5                          % Cantidad de imágenes a ajustar.
    a=trainingData(i,1);           % Extracción de la ruta del i-ésimo elemento del dataset.
    a=table2array(a);              % Conversión de tabla a vector.
    a=char(a);                     % Formato necesario para la función imread.
    b=imread(a);                   % Lectura de la i-ésima imagen.
    [x y z]=size(b);               % Lectura de tamaño para posterior escalización.
    b=imresize(b,[227 227]);       % Ajuste de la imagen al tamaño deseado.
    imwrite(b,a);                  % Escritura de la nueva imagen en el destino.
    a=trainingData(i,2);           % Ruta del i-ésimo vector de datos (primera etiqueta = pickup).
    a=table2array(a);              % Conversión de tabla a vector.
    a=a{1,1};                      % Extracción de los datos para su respectiva operación.
    if a~=0                        % Validación de vector de datos no vacío.
        a(1)=a(1)*(227/y);         % Escalización de los datos de etiqueta.
        a(3)=a(3)*(227/y);
        a(2)=a(2)*(227/x);
        a(4)=a(4)*(227/x);
        ed(i,1)=num2cell(a,2);     % Almacenamiento de nuevo vector de etiqueta.
    end
    a=trainingData(i,3);           % Ruta del i-ésimo vector de datos (segunda etiqueta = truck).
    a=table2array(a);              % Conversión de tabla a vector.
    a=a{1,1};                      % Extracción de los datos para su respectiva operación.
    if a~=0                        % Validación de vector de datos no vacío.
        a(2)=a(1)*(227/y);         % Escalización de los datos de etiqueta.
        a(3)=a(3)*(227/y);
        a(2)=a(2)*(227/x);
        a(4)=a(4)*(227/x);
        ed(i,2)=num2cell(a,2);
    end
end
```

Hecho esto obtendremos un arreglo como el siguiente:



	1	2	3	4
1	[16,49,358,257]			
2	[21,11,101,68]			
3	[14,39,357,243]			
4	[6,6,119,62]			
5	[36,43,341,223]			

El cual debemos convertir a tabla mediante:

```
ed=cell2table(ed);
```

ed | gTruth | gTruth.LabelDefir

5x1 table

	1 ed	2
1	16 49 358 257	
2	21 11 101 68	
3	14 39 357 243	
4	6 6 119 62	
5	36 43 341 223	

ahora es necesario cambiar los nombres de las columnas para que coincidan con las Label Definitions:

ed | gTruth | gTruth.LabelDefiniti

5x2 table

	1 pickup	2 truck
1	16 49 358 257	[]
2	21 11 101 68	[]
3	14 39 357 243	[]
4	6 6 119 62	[]
5	36 43 341 223	[]
6		

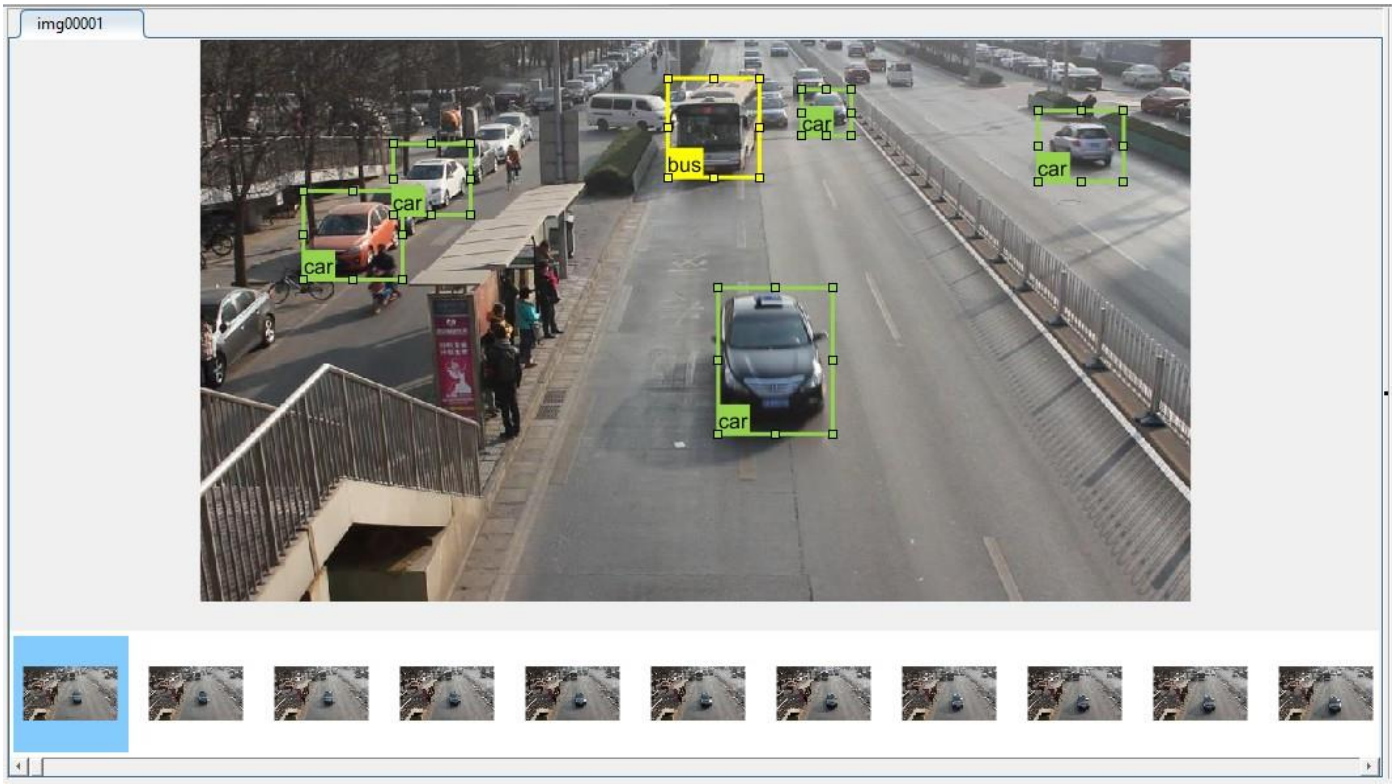
hecho esto podemos crear un nuevo archivo gTruth con los nuevos datos:

```
new=groundTruth(new_DataSource,gTruth.LabelDefinitions,ed);
```

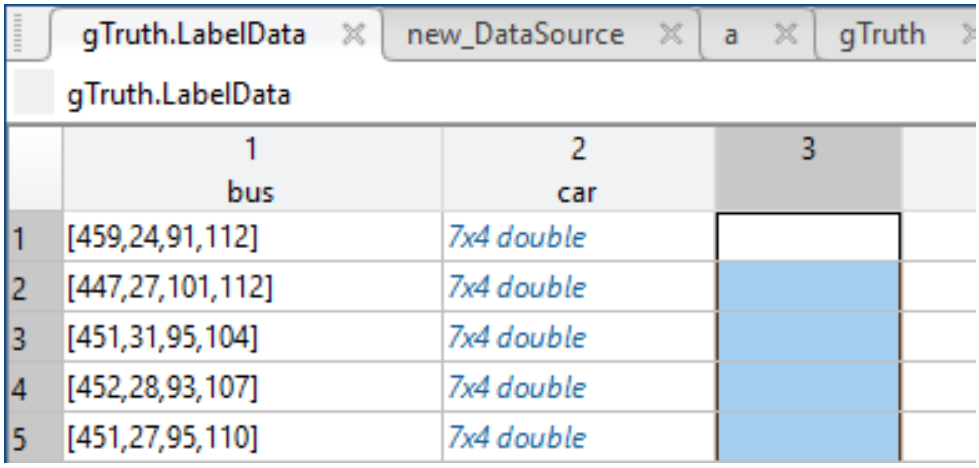
Al importar nuestro nuevo archivo gTruth en el *ImageLabeler* aparecerá un resultado como este:

Cambiar el tamaño de las imágenes (con más de una etiqueta por imagen):

Para ajustar aquellas imágenes que contienen más de una etiqueta como la que se muestra a continuación



es necesario llevar a cabo un proceso similar, pero con algunas modificaciones, ya que la tabla de Label Data luce parecida a esto



gTruth.LabelData | new_DataSource | a | gTruth

gTruth.LabelData

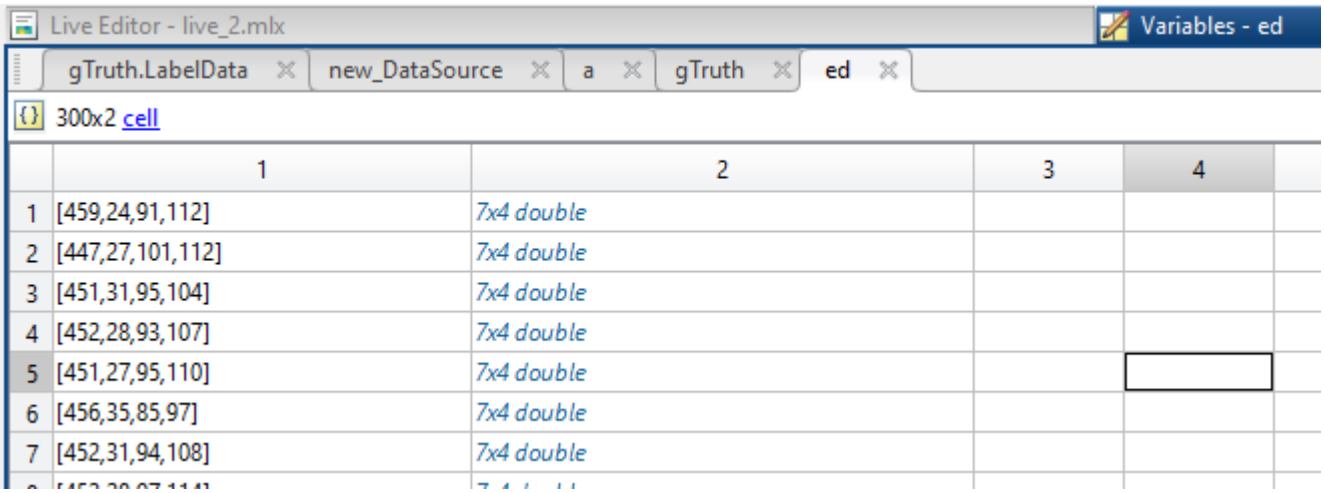
	1 bus	2 car	3
1	[459,24,91,112]	7x4 double	
2	[447,27,101,112]	7x4 double	
3	[451,31,95,104]	7x4 double	
4	[452,28,93,107]	7x4 double	
5	[451,27,95,110]	7x4 double	

debido a que cada etiqueta posee un vector, aquellas imágenes que posean más de una etiqueta del mismo tipo tendrán un arreglo de vectores, por esta razón es necesario realizar algunos ajustes que se describen a continuación:

```
for i=1:300
    a=f(i,1);
    a=table2array(a);
    a=char(a);
    b=imread(a);
    [x y z]=size(b);
    b=imresize(b,[350 350]);
    imwrite(b,a);
    a=f(i,2);
    a=table2array(a);
    a=a{1,1};
    if a~=0
        s=size(a);                  % Extraer tamaño del arreglo de vectores.
        if s(1)==1                  % Si es un sólo vector se opera directamente.
            a(1)=a(1)*(350/y);
            a(3)=a(3)*(350/y);
            a(2)=a(2)*(350/x);
            a(4)=a(4)*(350/x);
            ed(i,1)=num2cell(a,2);
        elseif s(1)>1               % Para varios vectores es necesario recorrer fila por fila.
            for j=1:s(1);
                a(j,1)=a(j,1)*(350/y);
                a(j,3)=a(j,3)*(350/y);
                a(j,2)=a(j,2)*(350/x);
                a(j,4)=a(j,4)*(350/x);
            end
            m=num2cell(a,2);        % Convertir números independientes en arreglo.
            m=cell2mat(m);          % Convertir arreglo a matriz para su respectivo almacenamiento
            ed{i,1}=m;              % Almacenar arreglo de vectores
        end
```

```
    end
    a=f(i,3);
    a=table2array(a);
    a=a{1,1};
    if a~=0
        s=size(a);
        for j=1:s(1);
            a(j,1)=a(j,1)*(350/y);
            a(j,3)=a(j,3)*(350/y);
            a(j,2)=a(j,2)*(350/x);
            a(j,4)=a(j,4)*(350/x);

        end
        m=num2cell(a,2);
        m=cell2mat(m);
        ed{i,2}=m;
    end
end
```

La ejecución de este código nos dejará como resultado un arreglo como el siguiente:

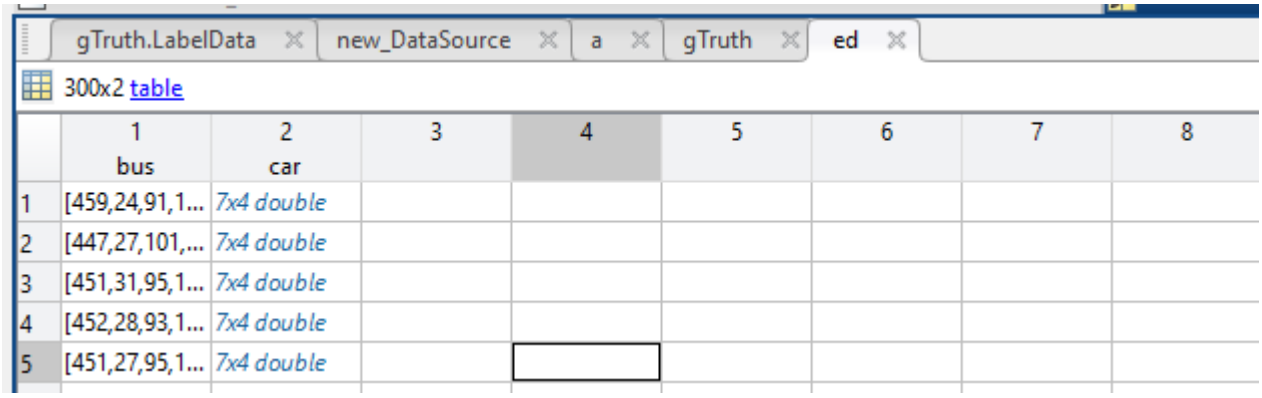


	1	2	3	4
1	[459,24,91,112]	7x4 double		
2	[447,27,101,112]	7x4 double		
3	[451,31,95,104]	7x4 double		
4	[452,28,93,107]	7x4 double		
5	[451,27,95,110]	7x4 double		
6	[456,35,85,97]	7x4 double		
7	[452,31,94,108]	7x4 double		

procedemos a convertir el arreglo en tabla

```
ed=array2table(ed);
```

y a cambiar los valores de las columnas para que coincidan con la definición de las etiquetas



gTruth.LabelData | new_DataSource | a | gTruth | ed

300x2 table

	1 bus	2 car	3	4	5	6	7	8
1	[459,24,91,1...	7x4 double						
2	[447,27,101,...	7x4 double						
3	[451,31,95,1...	7x4 double						
4	[452,28,93,1...	7x4 double						
5	[451,27,95,1...	7x4 double						

Con esto podremos construir un nuevo archivo gTruth con las imágenes ajustadas.

Recorte de imágenes para Alexnet

Debido a que la red Alexnet requiere el ingreso de imágenes que contengan un único objeto a detectar, es necesario realizar un recorte sobre las etiquetas del archivo gTruth para utilizar imágenes con varias etiquetas sobre la red Alexnet, esto se llevó a cabo mediante el siguiente código:

```
a=gTruth.DataSource.Source;  % Extraer ruta del Data Source.
c=gTruth.LabelData;          % Extraer definición de etiquetas.
t=0;                         % Contador para imagen.
for i=1:300
b=a(i);                      % Extraer ruta de la i-ésima imagen.
y=a(i+1);                    % Extraer ruta de la siguiente imagen.
b=char(b);
y=char(y);
I=imread(b);                 % Leer imágenes.
I2=imread(y);
d=c(i,2);                    % Extraer vector de datos de etiqueta de la i-ésima imagen.
p=c(i+1,2);                  % Extraer vector de datos de etiqueta de la siguiente imagen.
d=table2cell(d);             % Convertir de tabla a arreglo para extraer arreglo de vectores.
d=d{1};                      % Leer arreglo de vectores (matriz de datos).
p=table2cell(p);
p=p{1};
s=size(d);                   % Leer tamaño de la i-ésima imagen.
l=size(p);                   % Leer tamaño de la siguiente imagen.
for pp= s(1):11              % Rellenar ambas matrices con cero para que tengan el mismo tamaño.
    d(pp+1,1)=0;
end
for pp= l(1):11
    p(pp+1,1)=0;
end
uu=d-p;                      % Realizar resta de ambas matrices para comparar sus diferencias.

for j=1:s(1)
if(uu(j,1)>1)                % Evaluar si las imágenes son distintas (desplazamiento de la etiqueta).
t=t+1;
e=d(j,:);                    % Tomar vector de datos de etiqueta (mapa).
X2=imcrop(I,e);              % Recortar imagen sobre el mapa de la etiqueta.
h=sprintf('img%d.jpg',t);    % Dar formato al nombre de la imagen para almacenarla.
imwrite(X2,h);               % Almacenar imagen,
```

```
end
end
end
```

El resultado de ejecutar el código es el siguiente: