Statistical Learning for Data Science Final Project

Fall 2019

Hunter Kempf

Description

The goal of this project is to identify and explore interesting aspects of data in a real-world context, provide extensive explanations about each step, and report on your analysis of the results. The final project will involve communicating results of your analysis and technical results in a group presentation that effectively translates your work. You will work in groups of 3-4 students for the group presentation component and turn in an individual write-up component and a short reflection assignment.

The project will consist of the following tasks:

- Analyzing data and selecting the model best suited to solve the problem.
- Implementing and optimizing algorithms discussed in this course.
- Using Python to perform computations and analyze the results.
- Completing a slide deck for the group presentation.
- Effectively communicating methodology and results during the group presentation.

Project Problem - Classification

Given numerous images, create an image recognition algorithm to accurately identify images and explore how these images are related. Identify and discuss interesting aspects of the data. Optimize an algorithm to correctly identify images with the lowest possible error rate.

You will use the CIFAR-10 image dataset (https://www.cs.toronto.edu/~kriz/cifar.html (https://www.cs.toronto.edu/~kriz/cifar.html)), a widely used dataset for machine learning and image recognition. The data consists of 10 classes of images: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Each of the images are 32x32 color pixels with 6,000 images per class.
The original problem included 50,000 images for training and 10,000 for testing. You do not need to use the entire dataset. For this project, you should formulate a binary classification that you want to solve with this dataset. You will pick two classes of images for this problem. For example, you may choose to build a model that classifies images as either cats or dogs, or you might relabel the dataset to classify the images as objects or animals.

Although the goal is to build a good classifier, you are not expected to create the most accurate classifier possible. Instead you will be evaluated on your methodology: how you defined the problem, the process by which you solved the problem (exploratory data analysis, preprocessing, modeling and optimization) and how well you clearly communicate your analysis (group presentation and individual write-up).

As a data scientist you should be able to turn an ambiguous problem into a practical one problem that you can solve. For this reason, you are not required to use all of the data as it is. Feel free to use a subset of the original data, or use different sized training and test sets, but be able to defend the choices you make in defining the problem. It is better to have an effective solution to your problem, even if it isn't the optimal one.
You are to work individually on and turn in a written report summarizing the results of your project and your code. The actual work can and should be done as a group. You can work together to use the same code, analyses, results and plots. But this write-up describing the work you did should be in your own words.

The report should be:

- 3–10 pages in length (not including graphs)
- In 12-point font, double spaced, with 1-inch margins
- Saved as PDF

Your individual write-up should consist of the following sections:

1. **Define the Questions**
   A. Formulate a binary classification problem using the CIFAR-10 dataset.
2. **Get the Data**
   A. Describe the steps you used to prepare the data to answer the questions for this classification problem.
   B. Explain how you split the data for analysis and evaluation, and why you did it that way.
   C. Did you make any changes to the data or labels?
3. **Explore the Data**
   A. Perform exploratory data analysis (EDA) to first understand the dataset and hypothesis what features or methods might be useful (e.g. what features of the images are most similar or different?)
   B. Summarize your findings from the EDA.
4. **Preprocessing**
   A. Defend any preprocessing and explain why you chose those particular method(s).
   B. How did you handle the color images?
   C. If you used dimension reduction methods, how/why did you optimize it the way you did (i.e. number of components used, include scree plot where possible)?
5. **Initial Model**
   A. Choose an unoptimized models discussed in this course to build a classifier, and explain why you chose them. This should be a flexible model that you can optimize for better performance. This can be a Random Forest, Gradient Boosted Decision Trees, or a Neural Network.
   B. Evaluate the performance of the initial model using the appropriate metrics discussed in this course, and explain why you chose these.
   C. Justify what this model was selected. What are the advantages and disadvantages for this model compared to others?
6. **Model Optimization**
   A. Optimize the initial model to get better performance metrics.
   B. Discuss the methods used to optimize your model. How did you choose the tuning parameters?
7. **Figures**
   A. Include figures to compare the performance of the optimized model and base model.

In [1]:
```
#Load packages and get the CIFAR-10 data.
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# TensorFlow / Keras functions
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import cifar10
```

In [2]:
```
import datetime
# Load the TensorBoard notebook extension
%load_ext tensorboard
# Clear any logs from previous runs
!rm -rf ./logs/

log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, h
istogram_freq=1)
```

In [ ]:

In [3]:
```
import os
os.getcwd()
```

Out[3]:
```
'/Users/hk720x/Developer/MSDS/Stats Learning'
```

In [4]:
```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

In [ ]:

EDA

In [5]:
```
print("Feature Data Shape:",np.shape(X_train))
```

```
Feature Data Shape: (50000, 32, 32, 3)
```

Data Structure: 50,000 images, 32x32 pixels per image, 3 colors (RGB)

```
In [6]: print("Feature Data Min:",np.min(X_train))
        print("Feature Data Max:",np.max(X_train))
        print("Feature Data Mean:",np.mean(X_train))
```

```
Feature Data Min: 0
Feature Data Max: 255
Feature Data Mean: 120.70756512369792
```

Data Characteristics: Pixel Values are 0-255

```
In [7]: print("Target Data Shape:",np.shape(y_train))
```

```
Target Data Shape: (50000, 1)
```

```
In [8]: np.unique(y_train)
```

```
Out[8]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

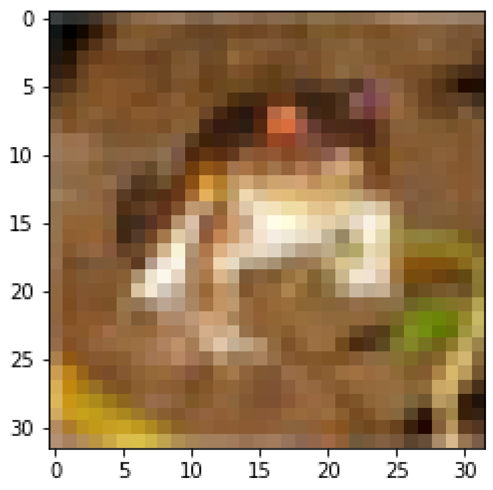
Target Data Structure is 50,000 values between 0-9 representing 10 different image classes

- 0 $\rightarrow$ airplane
- 1 $\rightarrow$ automobile
- 2 $\rightarrow$ bird
- 3 $\rightarrow$ cat
- 4 $\rightarrow$ deer
- 5 $\rightarrow$ dog
- 6 $\rightarrow$ frog
- 7 $\rightarrow$ horse
- 8 $\rightarrow$ ship
- 9 $\rightarrow$ truck

Example Image

In [9]:
```
plt.imshow(X_train[0])
```

Out[9]:
```
<matplotlib.image.AxesImage at 0x1a3c676dd8>
```



y value = 6 → frog

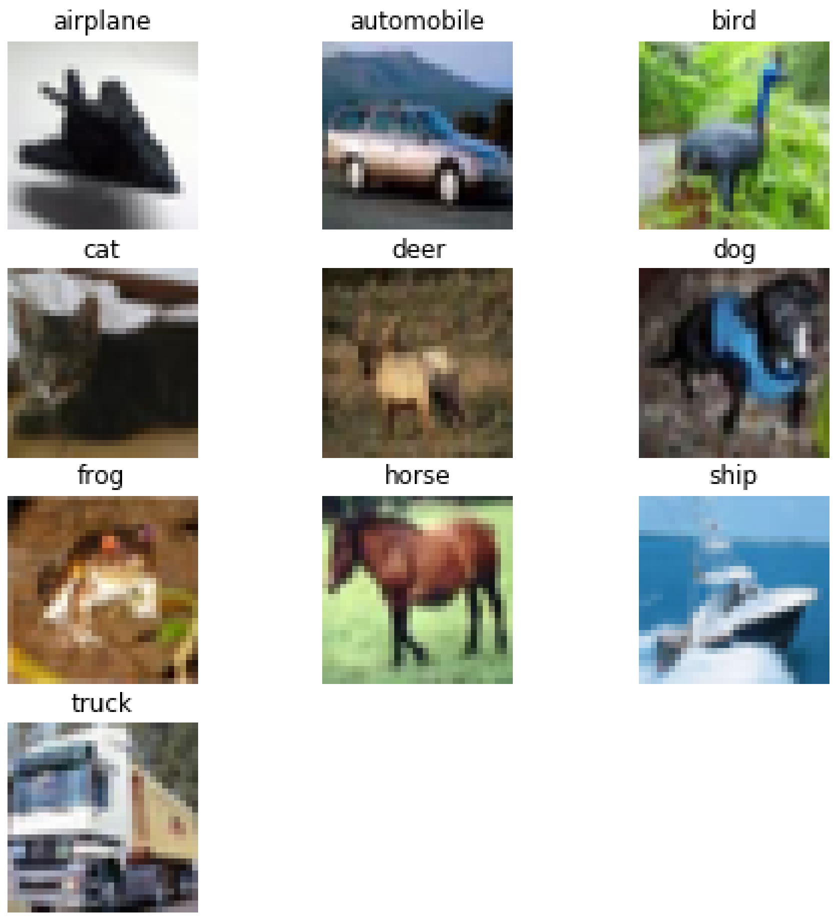
In [10]:
```
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog'
, 'horse', 'ship', 'truck']
```

In [11]:
```
def first_image_index_of_yval(yval):
    img_index = np.where(y_train == yval)[0][0]
    return img_index
```

In [12]:
```
first_image_index_of_yval(0)
```

Out[12]:
```
29
```

In [13]:
```
fig=plt.figure(figsize=(8, 8))
columns = 3
rows = 4
for i in range(1, 10 +1):
    index = first_image_index_of_yval(i-1)
    img = X_train[index]
    fig.add_subplot(rows, columns, i)
    plt.title(labels[i-1])
    plt.axis('off')
    plt.imshow(img)
plt.show()
```



In [14]:
```
# labels: ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'fro
g', 'horse', 'ship', 'truck']
```

Pick Two Classes

In our case we chose to determine the differences between planes and automobiles

In [15]:
```
# classify airplanes (0) and automobiles (1)
img_index = np.where((y_train == 0) | (y_train == 1))[0]
img_index_test = np.where((y_test == 0) | (y_test == 1))[0]
#img_index1 = np.where()[0]

X_train = X_train[img_index]
X_test = X_test[img_index_test]

y_train = y_train[img_index]
y_test = y_test[img_index_test]

print("Y Unique Values",np.unique(y_train),np.unique(y_test))
print("Y Shape",np.shape(y_train),np.shape(y_test))
```

```
Y Unique Values [0 1] [0 1]
Y Shape (10000, 1) (2000, 1)
```

Split Train and Validate

80% / 20% split of the train data into train and validation sets to prevent overfitting.

In [16]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test
_size=.2, stratify=y_train, random_state = 1842)
```

In [17]:
```
print("X Shape: ")
print("Train: ",np.shape(X_train))
print("Validate: ",np.shape(X_val))
print("Test: ",np.shape(X_test))
print("Y Shape: ")
print("Train: ",np.shape(y_train))
print("Validate: ",np.shape(y_val))
print("Test: ",np.shape(y_test))
```

```
X Shape:
Train:  (8000, 32, 32, 3)
Validate:  (2000, 32, 32, 3)
Test:  (2000, 32, 32, 3)
Y Shape:
Train:  (8000, 1)
Validate:  (2000, 1)
Test:  (2000, 1)
```

In [18]:
```
print("=========== Train ================")
print("Train Feature Data Min:",np.min(X_train))
print("Train Feature Data Max:",np.max(X_train))
print("Train Feature Data Mean:",round(np.mean(X_train),2))
print("Train Feature Data Type:",X_train.dtype)
print("=========== Validate ================")
print("Val Feature Data Min:",np.min(X_val))
print("Val Feature Data Max:",np.max(X_val))
print("Val Feature Data Mean:",round(np.mean(X_val),2))
print("Val Feature Data Type:",X_val.dtype)
print("=========== Test ================")
print("Test Feature Data Min:",np.min(X_test))
print("Test Feature Data Max:",np.max(X_test))
print("Test Feature Data Mean:",round(np.mean(X_test),2))
print("Test Feature Data Type:",X_test.dtype)
```

```
=========== Train ================
Train Feature Data Min: 0
Train Feature Data Max: 255
Train Feature Data Mean: 129.58
Train Feature Data Type: uint8
=========== Validate ================
Val Feature Data Min: 0
Val Feature Data Max: 255
Val Feature Data Mean: 129.34
Val Feature Data Type: uint8
=========== Test ================
Test Feature Data Min: 0
Test Feature Data Max: 255
Test Feature Data Mean: 131.13
Test Feature Data Type: uint8
```

Normalize Data

Neural Nets Generally perform better when the input values are normalized to a min of 0 and a max of 1 because of the activation functions.

In [19]:
```
X_train_norm = X_train/255
X_val_norm = X_val/255
X_test_norm = X_test/255
```

In [20]:
```
print("=========== Train =================")
print("Train Feature Data Min:",np.min(X_train_norm))
print("Train Feature Data Max:",np.max(X_train_norm))
print("Train Feature Data Mean:",round(np.mean(X_train_norm),2))
print("=========== Validate =================")
print("Val Feature Data Min:",np.min(X_val_norm))
print("Val Feature Data Max:",np.max(X_val_norm))
print("Val Feature Data Mean:",round(np.mean(X_val_norm),2))
print("=========== TEST =================")
print("Test Feature Data Min:",np.min(X_test_norm))
print("Test Feature Data Max:",np.max(X_test_norm))
print("Test Feature Data Mean:",round(np.mean(X_test_norm),2))
```

```
=========== Train =================
Train Feature Data Min: 0.0
Train Feature Data Max: 1.0
Train Feature Data Mean: 0.51
=========== Validate =================
Val Feature Data Min: 0.0
Val Feature Data Max: 1.0
Val Feature Data Mean: 0.51
=========== TEST =================
Test Feature Data Min: 0.0
Test Feature Data Max: 1.0
Test Feature Data Mean: 0.51
```

Model Code

- Base Model
- Model with out Feature regularization
- Model with Feature regularization

In [21]:
```
# Clear any logs from previous runs you may or may not want to do this
!rm -rf ./logs/
%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

In [22]:
```
from tensorflow.keras import Sequential
from tensorflow.keras.regularizers import l1, l2
from tensorflow.keras.layers import Dense, Dropout, LeakyReLU, Conv2D, M
axPooling2D, Flatten, LeakyReLU
from tensorflow.keras.callbacks import EarlyStopping, TensorBoard
kernelSize = 8
windowSize1 = (2,2)
windowSize2 = (2,2)

model1 = Sequential()
model1.add(Conv2D(kernelSize, windowSize1, padding='same', activation='r
elu',
                 input_shape=(32, 32, 3)))
model1.add(MaxPooling2D(pool_size=windowSize2))
model1.add(Flatten())
model1.add(Dense(32, activation='relu'))
model1.add(Dense(1, activation='sigmoid'))

model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['a
ccuracy'])


runName = 'logs/Base_model'
tensorboard = TensorBoard(log_dir=runName)

model1.fit(X_train, y_train, epochs=10, batch_size=100, validation_data=
(X_val, y_val), callbacks=[tensorboard])
```

```
Train on 8000 samples, validate on 2000 samples
Epoch 1/10
8000/8000 [==============================] - 1s 179us/sample - loss: 5.
2314 - accuracy: 0.7182 - val_loss: 0.7680 - val_accuracy: 0.7945
Epoch 2/10
8000/8000 [==============================] - 1s 103us/sample - loss: 0.
6476 - accuracy: 0.8131 - val_loss: 0.5627 - val_accuracy: 0.8420
Epoch 3/10
8000/8000 [==============================] - 1s 103us/sample - loss: 0.
4631 - accuracy: 0.8482 - val_loss: 0.4210 - val_accuracy: 0.8555
Epoch 4/10
8000/8000 [==============================] - 1s 103us/sample - loss: 0.
3804 - accuracy: 0.8631 - val_loss: 0.3880 - val_accuracy: 0.8555
Epoch 5/10
8000/8000 [==============================] - 1s 103us/sample - loss: 0.
3235 - accuracy: 0.8815 - val_loss: 0.3607 - val_accuracy: 0.8785
Epoch 6/10
8000/8000 [==============================] - 1s 100us/sample - loss: 0.
2702 - accuracy: 0.8975 - val_loss: 0.3748 - val_accuracy: 0.8745
Epoch 7/10
8000/8000 [==============================] - 1s 101us/sample - loss: 0.
2541 - accuracy: 0.9007 - val_loss: 0.3063 - val_accuracy: 0.8825
Epoch 8/10
8000/8000 [==============================] - 1s 100us/sample - loss: 0.
2240 - accuracy: 0.9089 - val_loss: 0.3112 - val_accuracy: 0.8825
Epoch 9/10
8000/8000 [==============================] - 1s 102us/sample - loss: 0.
2226 - accuracy: 0.9137 - val_loss: 0.2995 - val_accuracy: 0.8890
Epoch 10/10
8000/8000 [==============================] - 1s 106us/sample - loss: 0.
1806 - accuracy: 0.9273 - val_loss: 0.3029 - val_accuracy: 0.8895
```

```
Out[22]: <tensorflow.python.keras.callbacks.History at 0x1a50bcf4a8>
```

Model without Normalization of X values

In [23]:
```
# set tensorflow seed
tf.random.set_seed(1842)
kernelSize = 32
windowSize = (3,3)
windowSize2 = (2,2)
l2Decay = 0.0001
leakyRelu = LeakyReLU(0.001)

modelFinal = Sequential()

modelFinal.add(Conv2D(32, windowSize, padding='same', activation=leakyRe
lu, kernel_regularizer=l2(l2Decay), input_shape=(32,32,3)))
modelFinal.add(Conv2D(32, windowSize, padding='same', activation=leakyRe
lu, kernel_regularizer=l2(l2Decay)))
modelFinal.add(MaxPooling2D(pool_size=windowSize))
modelFinal.add(Dropout(0.2))

modelFinal.add(Conv2D(64, windowSize, padding='same', activation=leakyRe
lu, kernel_regularizer=l2(l2Decay)))
modelFinal.add(Conv2D(64, windowSize, padding='same', activation=leakyRe
lu, kernel_regularizer=l2(l2Decay)))
modelFinal.add(MaxPooling2D(pool_size=windowSize))
modelFinal.add(Dropout(0.2))

modelFinal.add(Conv2D(128, windowSize, padding='same', activation=leakyR
elu, kernel_regularizer=l2(l2Decay)))
modelFinal.add(Conv2D(128, windowSize, padding='same', activation=leakyR
elu, kernel_regularizer=l2(l2Decay)))
modelFinal.add(MaxPooling2D(pool_size=windowSize))
modelFinal.add(Dropout(0.2))
modelFinal.add(Flatten())

# Dense layers (add layers to account for dropout layers)
modelFinal.add(Dense(64, activation='relu'))
modelFinal.add(Dense(32, activation='relu'))
modelFinal.add(Dense(16, activation='relu'))
modelFinal.add(Dense(1, activation='sigmoid'))

modelFinal.summary()

modelFinal.compile(optimizer='adam', loss='binary_crossentropy', metrics
=['accuracy'])

runName = 'FinalProject_Final'
tensorboard = TensorBoard(log_dir='logs/'+runName)

modelFinal.fit(X_train, y_train, epochs=15, batch_size=50, validation_da
ta=(X_val, y_val),
         callbacks=[tensorboard])
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        9248      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 32)        0         
_________________________________________________________________
dropout (Dropout)            (None, 10, 10, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 64)        18496     
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 64)        36928     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 64)          0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 3, 3, 64)          0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 128)         73856     
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 3, 3, 128)         147584    
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 1, 1, 128)         0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 1, 1, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256      
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080      
_________________________________________________________________
dense_4 (Dense)              (None, 16)                528       
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 17        
=================================================================
Total params: 297,889
Trainable params: 297,889
Non-trainable params: 0
_________________________________________________________________
Train on 8000 samples, validate on 2000 samples
Epoch 1/15
8000/8000 [==============================] - 15s 2ms/sample - loss: 0.5
770 - accuracy: 0.7483 - val_loss: 0.3367 - val_accuracy: 0.8725
Epoch 2/15
8000/8000 [==============================] - 14s 2ms/sample - loss: 0.3
272 - accuracy: 0.8823 - val_loss: 0.2421 - val_accuracy: 0.9295
Epoch 3/15
8000/8000 [==============================] - 15s 2ms/sample - loss: 0.2
543 - accuracy: 0.9168 - val_loss: 0.1963 - val_accuracy: 0.9400
Epoch 4/15
8000/8000 [==============================] - 14s 2ms/sample - loss: 0.2
084 - accuracy: 0.9346 - val_loss: 0.1974 - val_accuracy: 0.9390
Epoch 5/15
8000/8000 [==============================] - 15s 2ms/sample - loss: 0.2
```

```
053 - accuracy: 0.9342 - val_loss: 0.2683 - val_accuracy: 0.9245
Epoch 6/15
8000/8000 [==============================] - 14s 2ms/sample - loss: 0.1
755 - accuracy: 0.9469 - val_loss: 0.2172 - val_accuracy: 0.9425
Epoch 7/15
8000/8000 [==============================] - 14s 2ms/sample - loss: 0.1
521 - accuracy: 0.9554 - val_loss: 0.1614 - val_accuracy: 0.9540
Epoch 8/15
8000/8000 [==============================] - 14s 2ms/sample - loss: 0.1
350 - accuracy: 0.9653 - val_loss: 0.1545 - val_accuracy: 0.9590
Epoch 9/15
8000/8000 [==============================] - 14s 2ms/sample - loss: 0.1
353 - accuracy: 0.9620 - val_loss: 0.1706 - val_accuracy: 0.9565
Epoch 10/15
8000/8000 [==============================] - 14s 2ms/sample - loss: 0.1
259 - accuracy: 0.9646 - val_loss: 0.2060 - val_accuracy: 0.9560
Epoch 11/15
8000/8000 [==============================] - 14s 2ms/sample - loss: 0.1
166 - accuracy: 0.9703 - val_loss: 0.1514 - val_accuracy: 0.9660
Epoch 12/15
8000/8000 [==============================] - 14s 2ms/sample - loss: 0.1
106 - accuracy: 0.9735 - val_loss: 0.1606 - val_accuracy: 0.9665
Epoch 13/15
8000/8000 [==============================] - 15s 2ms/sample - loss: 0.1
018 - accuracy: 0.9758 - val_loss: 0.1524 - val_accuracy: 0.9655
Epoch 14/15
8000/8000 [==============================] - 14s 2ms/sample - loss: 0.1
016 - accuracy: 0.9762 - val_loss: 0.1495 - val_accuracy: 0.9720
Epoch 15/15
8000/8000 [==============================] - 14s 2ms/sample - loss: 0.0
948 - accuracy: 0.9795 - val_loss: 0.1476 - val_accuracy: 0.9700
```

Out[23]:
```
<tensorflow.python.keras.callbacks.History at 0x1a38e3a4e0>
```

Model Analysis

In [24]:
```
scores = modelFinal.evaluate(X_test, y_test, verbose=0)
print("========Scores on Test Set========")
print("    * Accuracy: ",scores[1])
print("    * Loss: ",round(scores[0],7))
```

```
========Scores on Test Set========
    * Accuracy:  0.97
    * Loss:  0.1536703
```

In [25]:
```
print(np.shape(X_test))
print(X_test.dtype)
X_test_32 = tf.cast(X_test, tf.float32)
y_test_preds = modelFinal.predict_classes(X_test_32)
y_test_preds_8 = y_test_preds.astype(int)
class_names = np.array(['Airplanes','Cars'])
```

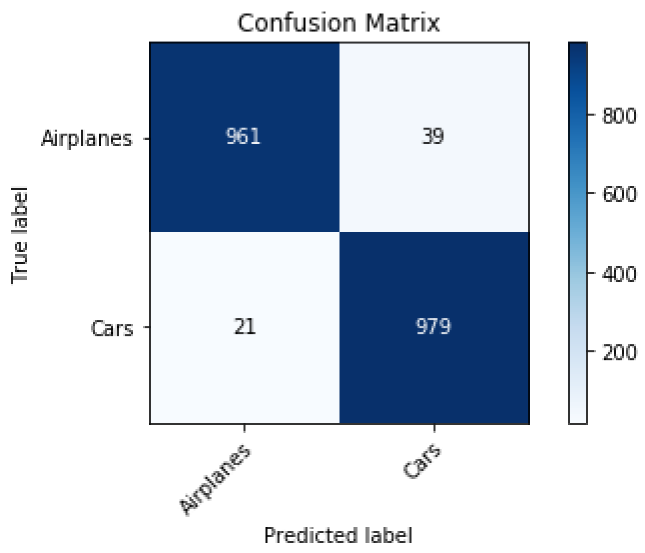
```
(2000, 32, 32, 3)
uint8
```

In [26]:
```
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels

def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax
```

In [27]:
```
# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, y_test_preds_8, classes=class_names,
                      title='Confusion Matrix')
```

```
Confusion matrix, without normalization
[[961  39]
 [ 21 979]]
```

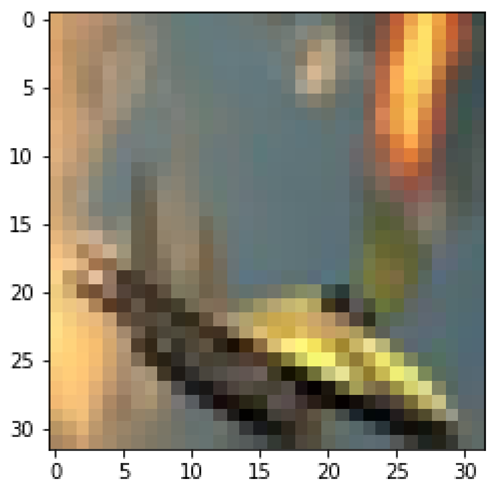
Out[27]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3a82e860>
```



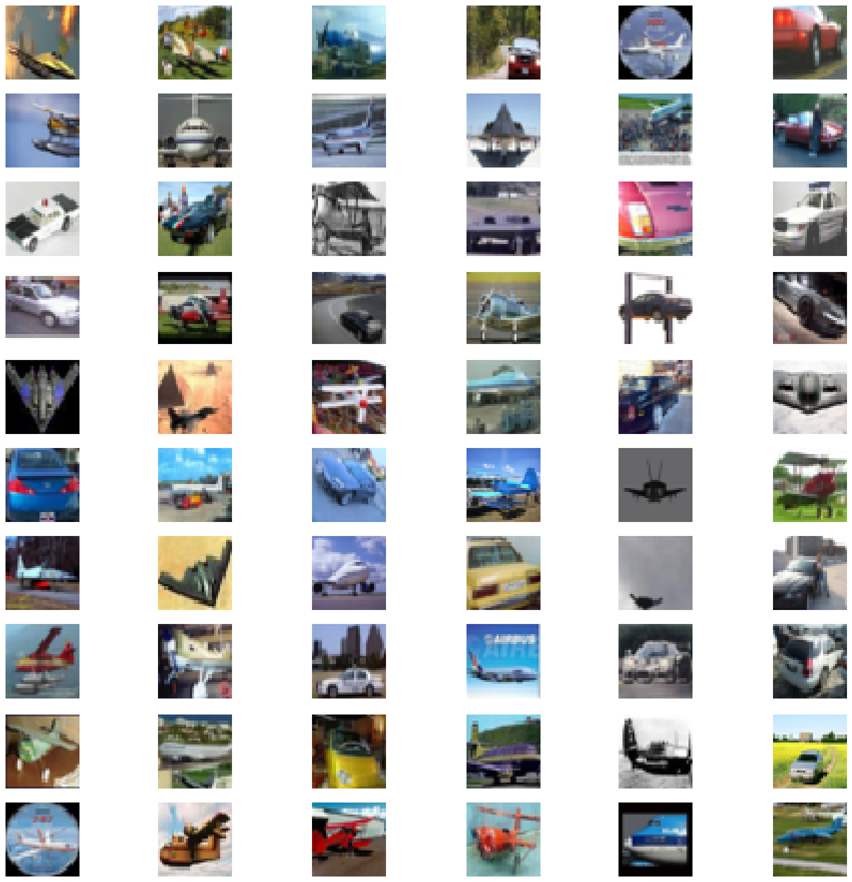
In [28]:
```
d = {'actual': y_test.ravel(), 'pred': y_test_preds_8.ravel()}

df = pd.DataFrame(data=d)
#df = df.assign(c=0 if (df.actual-df.pred) == 0 else 1)
Missclassified = np.where((df.actual-df.pred) == 0, 0, 1)
```

In [29]:
```
img_index_missclass = np.where(Missclassified == 1)[0].ravel()
plt.imshow(X_test[img_index_missclass[0]])
print(type(img_index_missclass))
```

```
<class 'numpy.ndarray'>
```

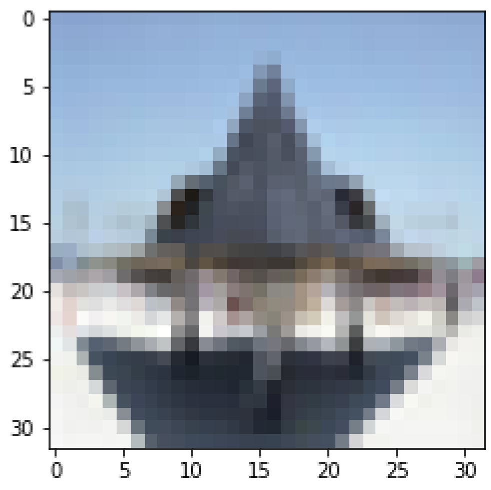


In [30]:
```
fig=plt.figure(figsize=(8, 8))
columns = 6
rows = 10
for i in range(1, columns*rows +1):
    index = img_index_missclass[i-1]
    img = X_test[index]
    fig.add_subplot(rows, columns, i)
    plt.axis('off')
    plt.imshow(img)
plt.show()
```



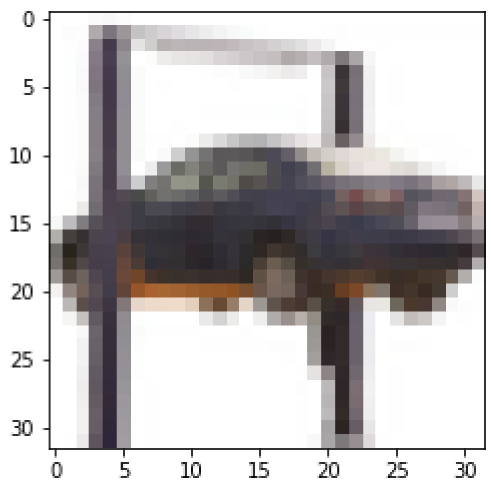
```
In [31]: plt.imshow(X_test[img_index_missclass[9]])
```

```
Out[31]: <matplotlib.image.AxesImage at 0x1a3026bba8>
```

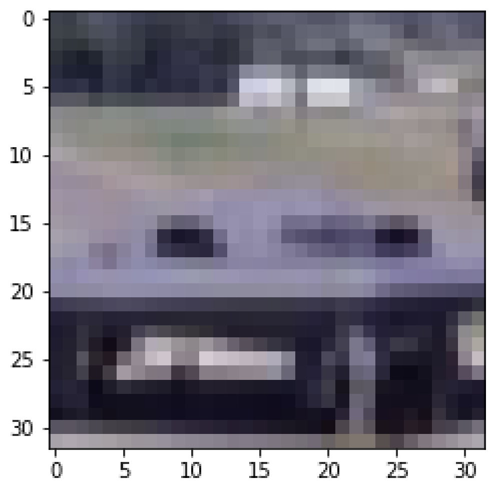


```
In [32]: plt.imshow(X_test[img_index_missclass[22]])
```

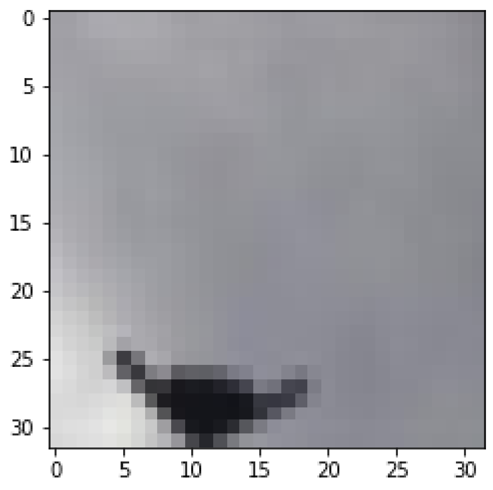
```
Out[32]: <matplotlib.image.AxesImage at 0x1a303532b0>
```



```
In [33]: plt.imshow(X_test[img_index_missclass[15]])

Out[33]: <matplotlib.image.AxesImage at 0x1a304b27f0>
```



```
In [34]: plt.imshow(X_test[img_index_missclass[40]])

Out[34]: <matplotlib.image.AxesImage at 0x1a30518048>
```



Model with Normalization of X values

In [35]:
```
tf.random.set_seed(1842)
kernelSize = 32
windowSize = (3,3)
windowSize2 = (2,2)
l2Decay = 0.0001
leakyRelu = LeakyReLU(0.001)

modelFinal = Sequential()

modelFinal.add(Conv2D(32, windowSize, padding='same', activation='relu',
kernel_regularizer=l2(l2Decay), input_shape=(32,32,3)))
#modelFinal.add(Conv2D(32, windowSize, padding='same', activation='rel
u', kernel_regularizer=l2(l2Decay)))
modelFinal.add(MaxPooling2D(pool_size=windowSize))
modelFinal.add(Dropout(0.2))

#modelFinal.add(Conv2D(64, windowSize, padding='same', activation='rel
u', kernel_regularizer=l2(l2Decay)))
#modelFinal.add(Conv2D(64, windowSize, padding='same', activation='rel
u', kernel_regularizer=l2(l2Decay)))
#modelFinal.add(MaxPooling2D(pool_size=windowSize))
#modelFinal.add(Dropout(0.2))

#modelFinal.add(Conv2D(128, windowSize, padding='same', activation='rel
u', kernel_regularizer=l2(l2Decay)))
#modelFinal.add(Conv2D(128, windowSize, padding='same', activation='rel
u', kernel_regularizer=l2(l2Decay)))
#modelFinal.add(MaxPooling2D(pool_size=windowSize))
#modelFinal.add(Dropout(0.2))
modelFinal.add(Flatten())

# Dense layers (add layers to account for dropout layers)
#modelFinal.add(Dense(64, activation='relu'))
#modelFinal.add(Dense(32, activation='relu'))
#modelFinal.add(Dense(16, activation='relu'))
modelFinal.add(Dense(1, activation='sigmoid'))

modelFinal.summary()

modelFinal.compile(optimizer='adam', loss='binary_crossentropy', metrics
=['accuracy'])

runName = 'FinalProject_Final'+ '_NormalizedFeatures'
tensorboard = TensorBoard(log_dir='logs/'+runName)

modelFinal.fit(X_train_norm, y_train, epochs=15, batch_size=50, validati
on_data=(X_test_norm, y_test),
         callbacks=[tensorboard])
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_7 (Conv2D)            (None, 32, 32, 32)        896       
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 10, 10, 32)        0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 10, 10, 32)        0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 3200)              0         
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 3201      
=================================================================
Total params: 4,097
Trainable params: 4,097
Non-trainable params: 0
_________________________________________________________________
Train on 8000 samples, validate on 2000 samples
Epoch 1/15
8000/8000 [==============================] - 3s 342us/sample - loss: 0.
4673 - accuracy: 0.7897 - val_loss: 0.3381 - val_accuracy: 0.8635
Epoch 2/15
8000/8000 [==============================] - 2s 285us/sample - loss: 0.
3447 - accuracy: 0.8536 - val_loss: 0.2935 - val_accuracy: 0.8820
Epoch 3/15
8000/8000 [==============================] - 2s 297us/sample - loss: 0.
3062 - accuracy: 0.8691 - val_loss: 0.2664 - val_accuracy: 0.8880
Epoch 4/15
8000/8000 [==============================] - 2s 301us/sample - loss: 0.
2775 - accuracy: 0.8834 - val_loss: 0.2490 - val_accuracy: 0.8970
Epoch 5/15
8000/8000 [==============================] - 2s 290us/sample - loss: 0.
2612 - accuracy: 0.8917 - val_loss: 0.2359 - val_accuracy: 0.9025
Epoch 6/15
8000/8000 [==============================] - 2s 292us/sample - loss: 0.
2505 - accuracy: 0.8984 - val_loss: 0.2206 - val_accuracy: 0.9105
Epoch 7/15
8000/8000 [==============================] - 2s 295us/sample - loss: 0.
2329 - accuracy: 0.9078 - val_loss: 0.2117 - val_accuracy: 0.9140
Epoch 8/15
8000/8000 [==============================] - 2s 298us/sample - loss: 0.
2241 - accuracy: 0.9119 - val_loss: 0.2039 - val_accuracy: 0.9185
Epoch 9/15
8000/8000 [==============================] - 2s 300us/sample - loss: 0.
2194 - accuracy: 0.9130 - val_loss: 0.2005 - val_accuracy: 0.9185
Epoch 10/15
8000/8000 [==============================] - 2s 284us/sample - loss: 0.
2122 - accuracy: 0.9165 - val_loss: 0.2001 - val_accuracy: 0.9215
Epoch 11/15
8000/8000 [==============================] - 2s 271us/sample - loss: 0.
2045 - accuracy: 0.9166 - val_loss: 0.1907 - val_accuracy: 0.9240
Epoch 12/15
8000/8000 [==============================] - 2s 276us/sample - loss: 0.
1991 - accuracy: 0.9237 - val_loss: 0.1885 - val_accuracy: 0.9275
Epoch 13/15
8000/8000 [==============================] - 2s 275us/sample - loss: 0.
```

```
1972 - accuracy: 0.9187 - val_loss: 0.1860 - val_accuracy: 0.9290
Epoch 14/15
8000/8000 [==============================] - 2s 277us/sample - loss: 0.
1932 - accuracy: 0.9225 - val_loss: 0.1863 - val_accuracy: 0.9265
Epoch 15/15
8000/8000 [==============================] - 2s 278us/sample - loss: 0.
1863 - accuracy: 0.9268 - val_loss: 0.1806 - val_accuracy: 0.9280
```

```
Out[35]: <tensorflow.python.keras.callbacks.History at 0x1a30788ac8>
```

Tensorboard Graphs

Use the below cell to launch tensorboard and view the training graphs

In [36]:
```
%tensorboard --logdir='logs/' --host localhost --port 8087
```

Index of /

Name	Size	Date Modified
.DocumentRevisions-V100/		11/4/19, 8:37:01 AM
.fseventsd/		11/5/19, 7:36:49 PM
.PKInstallSandboxManager-SystemSoftware/		10/31/19, 11:58:42 AM
.Spotlight-V100/		6/3/19, 11:17:40 AM
.Trashes/		6/29/16, 9:46:55 AM
.vol/		2/25/19, 10:49:22 PM
anaconda3/		11/5/19, 7:50:28 PM
Applications/		11/4/19, 5:52:11 PM
bin/		7/30/19, 1:59:21 PM
cores/		2/25/19, 10:49:19 PM
dev/		11/4/19, 8:36:50 AM
etc/		11/5/19, 7:35:36 PM
home/		11/4/19, 7:26:30 PM
Library/		8/20/19, 11:03:41 AM
net/		11/4/19, 7:26:30 PM
Network/		11/4/19, 8:37:01 AM
opt/		8/20/19, 11:03:40 AM
private/		6/3/19, 4:56:29 PM
Quarantine/		6/3/19, 4:48:16 PM
sbin/		7/30/19, 1:59:21 PM
System/		5/4/19, 12:21:03 AM
tmp/		11/5/19, 7:40:02 PM
Users/		6/3/19, 5:07:49 PM
usr/		5/4/19, 12:14:37 AM
var/		6/3/19, 4:47:25 PM
Volumes/		11/4/19, 12:45:49 PM
.DS_Store	14.0 kB	11/4/19, 6:12:15 PM
.file	0 B	2/25/19, 10:50:14 PM
.OSInstallerMessages	591 B	7/30/19, 2:00:28 PM
installer.failurerequests	313 B	2/24/19, 9:18:53 PM

In [ ]:

In [ ]: