CIFAR-10 CNN Image Classification - Data Augmentation

The problem of automatically classifying photographs of objects is difficult because of the near infinite number of permutations of objects, positions, lighting and so on. A standard computer vision and deep learning dataset for this problem was developed by the Canadian Institute for Advanced Research (CIFAR).

The CIFAR-10 dataset consists of 60,000 photos divided into 10 classes. Classes include common objects such as airplanes, automobiles, birds, cats and so on. The dataset is split in a standard way, where 50,000 images are used for training a model and the remaining 10,000 for evaluating its performance. The photos are in color with red, green and blue components, but are small measuring 32 by 32 pixel squares.

State-of-the-art models have achieved 96% classification accuracy, with very good performance considered above 90% and human performance on the problem is at 94%. This is my attempt to use a relatively small amount of data samples along with data augmentation to achieve good results; hopefully.

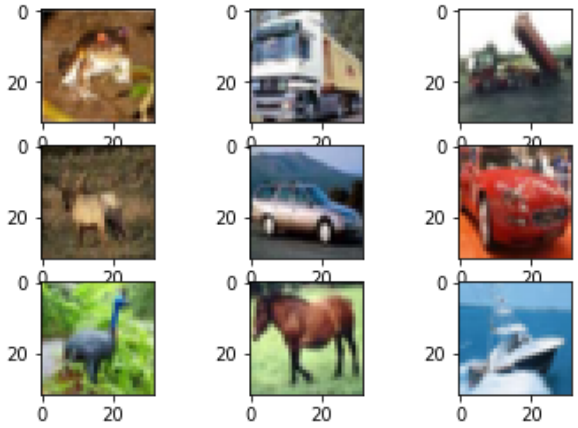
In [1]:
```
# imports
import numpy as np
from keras.datasets import cifar10
from matplotlib import pyplot as plt
from PIL import Image as im
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.utils import np_utils
from keras.preprocessing.image import ImageDataGenerator
```

```
/home/brent/Dev/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the s
econd argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `n
p.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [2]:
```
# set random seed
seed = 7
np.random.seed(seed)
```

In [3]:
```
# load dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

In [4]:
```
# check out images
for i in range(0,9):
    plt.subplot(330 + 1 + i)
    plt.imshow(im.fromarray(X_train[i], 'RGB'))
plt.show()
```



Data Augmented Model

In [5]:
```
# review input data
print("Input training shape:", X_train.shape)
print("Input test shape:", X_test.shape)
print("Output training shape:", y_train.shape)
print("Output test shape:", y_test.shape)
```

```
Input training shape: (50000, 32, 32, 3)
Input test shape: (10000, 32, 32, 3)
Output training shape: (50000, 1)
Output test shape: (10000, 1)
```

In [6]:
```
# one-hot encode the output vectors to convert them from a class number
# to a binary matrix representation of a class number
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
print("Number of output classes:", num_classes)
```

```
Number of output classes: 10
```

In [7]:
```
# validate OH encoding
print("Training output shape:", y_train.shape)
print("Test output shape:", y_test.shape)
print(y_test[3])
```

```
Training output shape: (50000, 10)
Test output shape: (10000, 10)
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

In [8]:
```
# define the CNN model
def cnn_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), data_format="channels_last", activation='relu', padd
ing='same'))
    model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(num_classes, activation='softmax'))
    return model
```

In [9]:
```
# create the cnn model
model = cnn_model()
nb_epochs = 150
lrate = 0.01
decay = lrate / nb_epochs
sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['acc'])
model.summary()
```

```
_________________________________________________________________
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, 16, 16, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        18496     
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        36928     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         73856     
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 128)         147584    
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 2048)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              2098176   
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                10250     
=================================================================
Total params: 2,395,434
Trainable params: 2,395,434
Non-trainable params: 0
_________________________________________________________________
```

In [10]:
```
# normalize and augment the input training and validation data
batch_size = 20
training_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True)

training_datagen.fit(X_train)
X_test = X_test / 255

train_steps = len(X_train) / batch_size
val_steps = len(X_test) / batch_size
```

In [11]:
```
# fit the model
history = model.fit_generator(training_datagen.flow(X_train, y_train, batch_size=batch_size), epochs=nb_epoch
s, steps_per_epoch=train_steps, validation_data=(X_test, y_test), validation_steps=val_steps).history
```

```
WARNING:tensorflow:Variable *= will be deprecated. Use variable.assign_mul if you want assignment to the vari
able value or 'x = x * y' if you want a new python Tensor object.
Epoch 1/150
2500/2500 [==============================] - 187s 75ms/step - loss: 1.9292 - acc: 0.2825 - val_loss: 1.6275 -
val_acc: 0.4145
Epoch 2/150
2500/2500 [==============================] - 185s 74ms/step - loss: 1.5798 - acc: 0.4175 - val_loss: 1.3002 -
val_acc: 0.5287
Epoch 3/150
2500/2500 [==============================] - 185s 74ms/step - loss: 1.3892 - acc: 0.4941 - val_loss: 1.1804 -
val_acc: 0.5801
Epoch 4/150
2500/2500 [==============================] - 183s 73ms/step - loss: 1.2531 - acc: 0.5514 - val_loss: 0.9929 -
val_acc: 0.6534
Epoch 5/150
2500/2500 [==============================] - 182s 73ms/step - loss: 1.1461 - acc: 0.5912 - val_loss: 0.9884 -
val_acc: 0.6501
Epoch 6/150
2500/2500 [==============================] - 183s 73ms/step - loss: 1.0737 - acc: 0.6179 - val_loss: 0.9135 -
val_acc: 0.6772
Epoch 7/150
2500/2500 [==============================] - 184s 73ms/step - loss: 1.0144 - acc: 0.6416 - val_loss: 0.8859 -
val_acc: 0.6830
Epoch 8/150
2500/2500 [==============================] - 183s 73ms/step - loss: 0.9581 - acc: 0.6614 - val_loss: 0.7921 -
val_acc: 0.7180
Epoch 9/150
2500/2500 [==============================] - 184s 73ms/step - loss: 0.9134 - acc: 0.6773 - val_loss: 0.8012 -
val_acc: 0.7191
Epoch 10/150
2500/2500 [==============================] - 183s 73ms/step - loss: 0.8761 - acc: 0.6912 - val_loss: 0.7939 -
val_acc: 0.7250
Epoch 11/150
2500/2500 [==============================] - 184s 73ms/step - loss: 0.8382 - acc: 0.7076 - val_loss: 0.7030 -
val_acc: 0.7552
Epoch 12/150
2500/2500 [==============================] - 179s 72ms/step - loss: 0.8073 - acc: 0.7171 - val_loss: 0.7313 -
val_acc: 0.7494
Epoch 13/150
2500/2500 [==============================] - 179s 72ms/step - loss: 0.7833 - acc: 0.7264 - val_loss: 0.6633 -
val_acc: 0.7730
Epoch 14/150
2500/2500 [==============================] - 179s 72ms/step - loss: 0.7593 - acc: 0.7361 - val_loss: 0.6235 -
val_acc: 0.7834
Epoch 15/150
2500/2500 [==============================] - 179s 72ms/step - loss: 0.7355 - acc: 0.7443 - val_loss: 0.6229 -
```

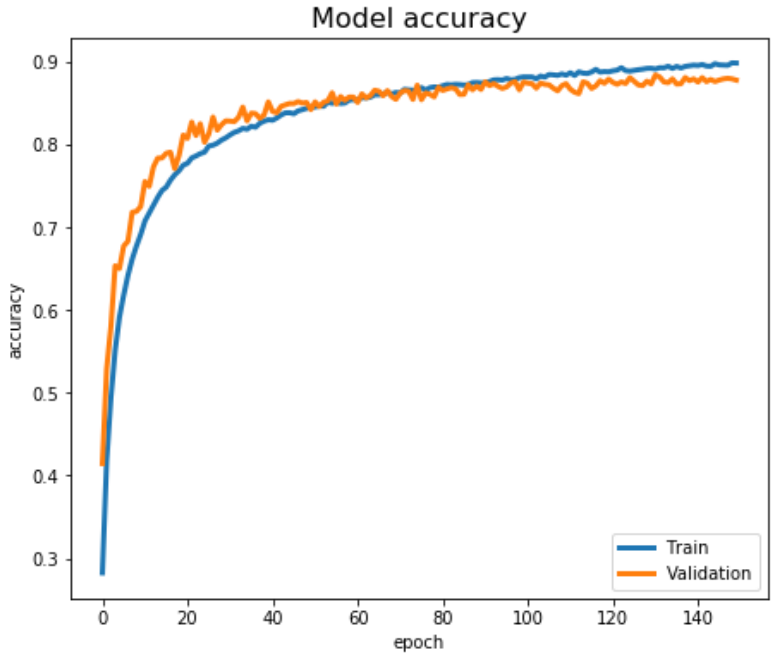
In [12]:
```
# evaluate model
score = model.evaluate(X_test, y_test)
print("Accuracy: %.2f%%" % (score[1]*100))
```

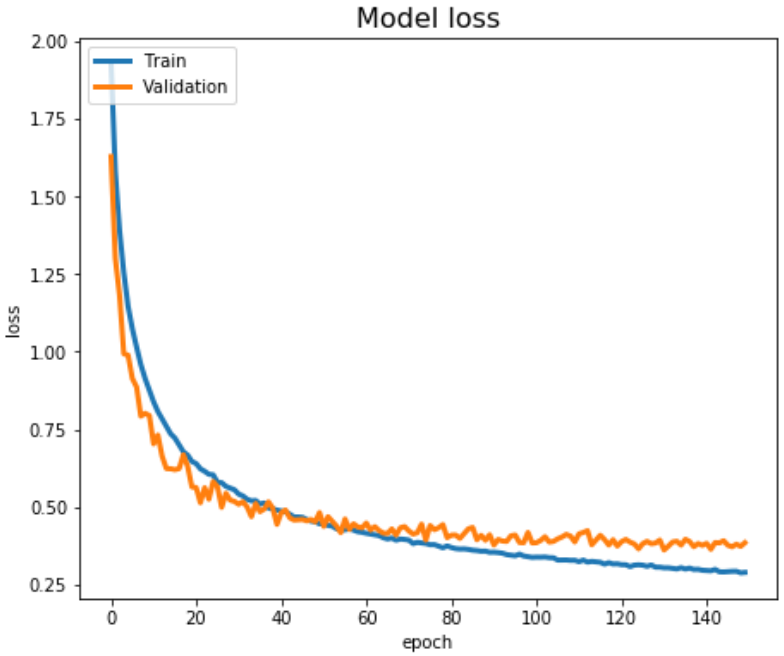
```
10000/10000 [==============================] - 9s 915us/step
Accuracy: 87.77%
```

In [13]:
```
# plot the model loss and accuracy
fig, (axis1, axis2) = plt.subplots(nrows=1, ncols=2, figsize=(16,6))

# summarize history for accuracy
axis1.plot(history['acc'], label='Train', linewidth=3)
axis1.plot(history['val_acc'], label='Validation', linewidth=3)
axis1.set_title('Model accuracy', fontsize=16)
axis1.set_ylabel('accuracy')
axis1.set_xlabel('epoch')
axis1.legend(loc='lower right')

# summarize history for loss
axis2.plot(history['loss'], label='Train', linewidth=3)
axis2.plot(history['val_loss'], label='Validation', linewidth=3)
axis2.set_title('Model loss', fontsize=16)
axis2.set_ylabel('loss')
axis2.set_xlabel('epoch')
axis2.legend(loc='upper left')
plt.show()
```





In [ ]: