week3_task1_first_cnn_cifar10_clean

January 3, 2021

1 Your first CNN on CIFAR-10

In this task you will: * define your first CNN architecture for CIFAR-10 dataset * train it from scratch * visualize learnt filters

CIFAR-10 dataset contains 32x32 color images from 10 classes: **airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck**:

2 Import stuff

```
In [3]: import sys
        sys.path.append("..")
        import grading
        import download_utils
```

```
In [ ]: # !!! remember to clear session/graph if you rebuild your graph to avoid out-of-memory e
```

```
In [2]: download_utils.link_all_keras_resources()
```

```
In [4]: import tensorflow as tf
        import keras
        from keras import backend as K
        import numpy as np
        %matplotlib inline
        import matplotlib.pyplot as plt
        print(tf.__version__)
        print(keras.__version__)
        import grading_utils
        import keras_utils
        from keras_utils import reset_tf_session
```

```
Using TensorFlow backend.


1.2.1
2.0.6
```

3 Fill in your Coursera token and email

To successfully submit your answers to our grader, please fill in your Coursera submission token and email

```
In [5]: grader = grading.Grader(assignment_key="s1B1I5DuEeeyLAqI7dCYkg",
                                all_parts=["7W4tu", "nQOsg", "96eco"])
```

```
In [6]: # token expires every 30 min
        COURSERA_TOKEN = '5xddopyBxSg6v513'        ### YOUR TOKEN HERE
        COURSERA_EMAIL = 'knowtech94@gmail.com'    ### YOUR EMAIL HERE
```

4 Load dataset

```
In [20]: from keras.datasets import cifar10
         (x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

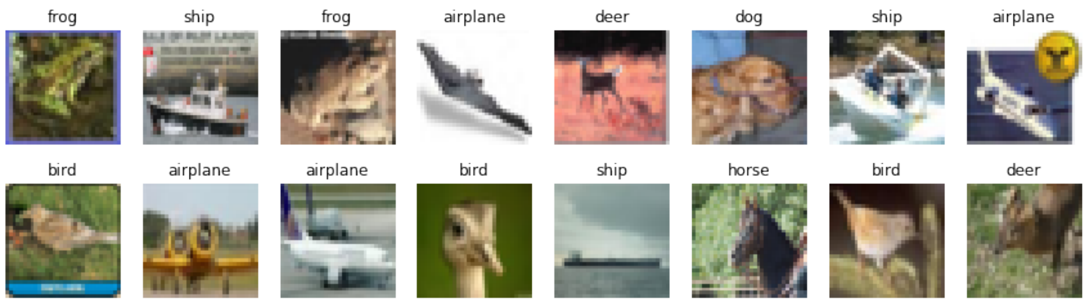
```
Downloading data from http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
169803776/170498071 [============================>.] - ETA: 0s
```

```
In [21]: print("Train samples:", x_train.shape, y_train.shape)
         print("Test samples:", x_test.shape, y_test.shape)
```

```
Train samples: (50000, 32, 32, 3) (50000, 1)
Test samples: (10000, 32, 32, 3) (10000, 1)
```

```
In [22]: NUM_CLASSES = 10
         cifar10_classes = ["airplane", "automobile", "bird", "cat", "deer",
                            "dog", "frog", "horse", "ship", "truck"]
```

```
In [23]: # show random images from train
         cols = 8
         rows = 2
         fig = plt.figure(figsize=(2 * cols - 1, 2.5 * rows - 1))
         for i in range(cols):
             for j in range(rows):
                 random_index = np.random.randint(0, len(y_train))
                 ax = fig.add_subplot(rows, cols, i * rows + j + 1)
                 ax.grid('off')
                 ax.axis('off')
                 ax.imshow(x_train[random_index, :])
                 ax.set_title(cifar10_classes[y_train[random_index, 0]])
         plt.show()
```



5 Prepare data

We need to normalize inputs like this:

$$x_{norm} = \frac{x}{255} - 0.5$$

We need to convert class labels to one-hot encoded vectors. Use **keras.utils.to_categorical**.

```
In [24]: # normalize inputs
         x_train2 = x_train/255 - 0.5          ### YOUR CODE HERE
         x_test2 = x_test/255 - 0.5            ### YOUR CODE HERE

         # convert class labels to one-hot encoded, should have shape (?, NUM_CLASSES)
         y_train2 = keras.utils.to_categorical(y_train)       ### YOUR CODE HERE
         y_test2 = keras.utils.to_categorical(y_test)         ### YOUR CODE HERE
```

6 Define CNN architecture

```
In [10]: # import necessary building blocks
         from keras.models import Sequential
         from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout
         from keras.layers.advanced_activations import LeakyReLU
```

Convolutional networks are built from several types of layers: - Conv2D - performs convolution: - **filters**: number of output channels; - **kernel_size**: an integer or tuple/list of 2 integers, specifying the width and height of the 2D convolution window; - **padding**: padding="same" adds zero padding to the input, so that the output has the same width and height, padding='valid' performs convolution only in locations where kernel and the input fully overlap; - **activation**: "relu", "tanh", etc. - **input_shape**: shape of input. - MaxPooling2D - performs 2D max pooling. - Flatten - flattens the input, does not affect the batch size. - Dense - fully-connected layer. - Activation - applies an activation function. - LeakyReLU - applies leaky relu activation. - Dropout - applies dropout.

You need to define a model which takes **(None, 32, 32, 3)** input and predicts **(None, 10)** output with probabilities for all classes. **None** in shapes stands for batch dimension.

Simple feed-forward networks in Keras can be defined in the following way:

```python
model = Sequential()  # start feed-forward model definition
model.add(Conv2D(..., input_shape=(32, 32, 3)))  # first layer needs to define "input_shape"

...  # here comes a bunch of convolutional, pooling and dropout layers

model.add(Dense(NUM_CLASSES))  # the last layer with neuron for each class
model.add(Activation("softmax"))  # output probabilities
```

Stack **4** convolutional layers with kernel size **(3, 3)** with growing number of filters **(16, 32, 32, 64)**, use "same" padding.

Add **2x2** pooling layer after every 2 convolutional layers (conv-conv-pool scheme).

Use **LeakyReLU** activation with recommended parameter **0.1** for all layers that need it (after convolutional and dense layers):

```python
model.add(LeakyReLU(0.1))
```

Add a dense layer with **256** neurons and a second dense layer with **10** neurons for classes. Remember to use **Flatten** layer before first dense layer to reshape input volume into a flat vector!

Add **Dropout** after every pooling layer (**0.25**) and between dense layers (**0.5**).

```python
In [31]: def make_model():
             """
             Define your model architecture here.
             Returns `Sequential` model.
             """
             model = Sequential()

             ### YOUR CODE HERE
             model.add(Conv2D(filters = 16, kernel_size = (3, 3), padding = 'same',
                                    input_shape = (32, 32, 3)))
             model.add(LeakyReLU(0.1))
             model.add(Conv2D(filters = 32, kernel_size = (3, 3), padding = 'same'))
             model.add(LeakyReLU(0.1))
             model.add(MaxPooling2D(pool_size = (2, 2)))
             model.add(Dropout(rate = 0.25))
             ##
             model.add(Conv2D(filters = 32, kernel_size = (3, 3), padding = 'same'))
             model.add(LeakyReLU(0.1))
             model.add(Conv2D(filters = 64, kernel_size = (3, 3), padding = 'same'))
             model.add(LeakyReLU(0.1))
             model.add(MaxPooling2D(pool_size = (2, 2)))
             model.add(Dropout(rate = 0.25))
             ##
             model.add(Flatten())
             model.add(Dense(units = 256))
             model.add(LeakyReLU(0.1))
             model.add(Dropout(rate = 0.5))
             model.add(Dense(units = NUM_CLASSES))
```

```
        model.add(Activation('softmax'))
        return model
```

In [32]:
```
# describe model
s = reset_tf_session()  # clear default graph
model = make_model()
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 16)        448
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        4640
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 32, 32, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 32)        9248
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 16, 16, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               1048832
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0
=================================================================
Total params: 1,084,234
```

```
Trainable params: 1,084,234
Non-trainable params: 0
_________________________________________________________________
```

In [33]:
```
## GRADED PART, DO NOT CHANGE!
# Number of model parameters
grader.set_answer("7W4tu", grading_utils.model_total_params(model))
```

In [34]:
```
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

7 Train model

Training of your model can take approx. 4-8 minutes per epoch.

During training you should observe the decrease in reported loss on training and validation.

If the loss on training is not decreasing with epochs you should revise your model definition and learning rate.

In [18]:
```
INIT_LR = 5e-3  # initial learning rate
BATCH_SIZE = 32
EPOCHS = 10

s = reset_tf_session()  # clear default graph
# don't call K.set_learning_phase() !!! (otherwise will enable dropout in train/test si
model = make_model()  # define our model

# prepare model for fitting (loss, optimizer, etc)
model.compile(
    loss='categorical_crossentropy',  # we train 10-way classification
    optimizer=keras.optimizers.adamax(lr=INIT_LR),  # for SGD
    metrics=['accuracy']  # report accuracy during training
)

# scheduler of learning rate (decay with epochs)
def lr_scheduler(epoch):
    return INIT_LR * 0.9 ** epoch

# callback for printing of actual learning rate used by optimizer
class LrHistory(keras.callbacks.Callback):
    def on_epoch_begin(self, epoch, logs={}):
        print("Learning rate:", K.get_value(model.optimizer.lr))
```

Training takes approximately **1.5 hours**. You're aiming for ~0.80 validation accuracy.

In [19]:
```
# we will save model checkpoints to continue training in case of kernel death
model_filename = 'cifar.{0:03d}.hdf5'
last_finished_epoch = None

#### uncomment below to continue training from model checkpoint
#### fill `last_finished_epoch` with your latest finished epoch
# from keras.models import load_model
# s = reset_tf_session()
# last_finished_epoch = 7
# model = load_model(model_filename.format(last_finished_epoch))
```

In [20]:
```
# fit model
model.fit(
    x_train2, y_train2,  # prepared data
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    callbacks=[keras.callbacks.LearningRateScheduler(lr_scheduler),
               LrHistory(),
               keras_utils.TqdmProgressCallback(),
               keras_utils.ModelSaveCallback(model_filename)],
    validation_data=(x_test2, y_test2),
    shuffle=True,
    verbose=0,
    initial_epoch=last_finished_epoch or 0
)
```

```
Learning rate: 0.005

Epoch 1/10


A Jupyter Widget


50000/|/loss: 1.3324; acc: 0.5227: 100%|| 50000/50000 [11:02<00:00, 82.03it/s]
Model saved in cifar.000.hdf5
Learning rate: 0.0045

Epoch 2/10


A Jupyter Widget


50000/|/loss: 0.9377; acc: 0.6698: 100%|| 50000/50000 [11:23<00:00, 74.41it/s]
Model saved in cifar.001.hdf5
Learning rate: 0.00405

Epoch 3/10
```

```
A Jupyter Widget


50000/|/loss: 0.8120; acc: 0.7154: 100%|| 50000/50000 [11:12<00:00, 79.76it/s]
Model saved in cifar.002.hdf5
Learning rate: 0.003645

Epoch 4/10


A Jupyter Widget


50000/|/loss: 0.7241; acc: 0.7451: 100%|| 50000/50000 [11:02<00:00, 77.60it/s]
Model saved in cifar.003.hdf5
Learning rate: 0.0032805

Epoch 5/10


A Jupyter Widget


50000/|/loss: 0.6679; acc: 0.7659: 100%|| 50000/50000 [11:02<00:00, 74.98it/s]
Model saved in cifar.004.hdf5
Learning rate: 0.00295245

Epoch 6/10


A Jupyter Widget


50000/|/loss: 0.6161; acc: 0.7829: 100%|| 50000/50000 [11:02<00:00, 75.73it/s]
Model saved in cifar.005.hdf5
Learning rate: 0.00265721

Epoch 7/10


A Jupyter Widget


50000/|/loss: 0.5805; acc: 0.7950: 100%|| 50000/50000 [10:52<00:00, 79.73it/s]
Model saved in cifar.006.hdf5
Learning rate: 0.00239148

Epoch 8/10
```

```
A Jupyter Widget
```

```
50000/|/loss: 0.5393; acc: 0.8115: 100%|| 50000/50000 [10:41<00:00, 68.31it/s]
Model saved in cifar.007.hdf5
Learning rate: 0.00215234

Epoch 9/10
```

```
A Jupyter Widget
```

```
50000/|/loss: 0.5124; acc: 0.8184: 100%|| 50000/50000 [11:02<00:00, 78.27it/s]
Model saved in cifar.008.hdf5
Learning rate: 0.0019371

Epoch 10/10
```

```
A Jupyter Widget
```

```
50000/|/loss: 0.4831; acc: 0.8284: 100%|| 50000/50000 [11:22<00:00, 66.30it/s]
Model saved in cifar.009.hdf5
```

```
Out[20]: <keras.callbacks.History at 0x7f5580304278>
```

```
In [21]: # save weights to file
         model.save_weights("weights.h5")
```

```
In [35]: # load weights from file (can call without model.fit)
         model.load_weights("weights.h5")
```

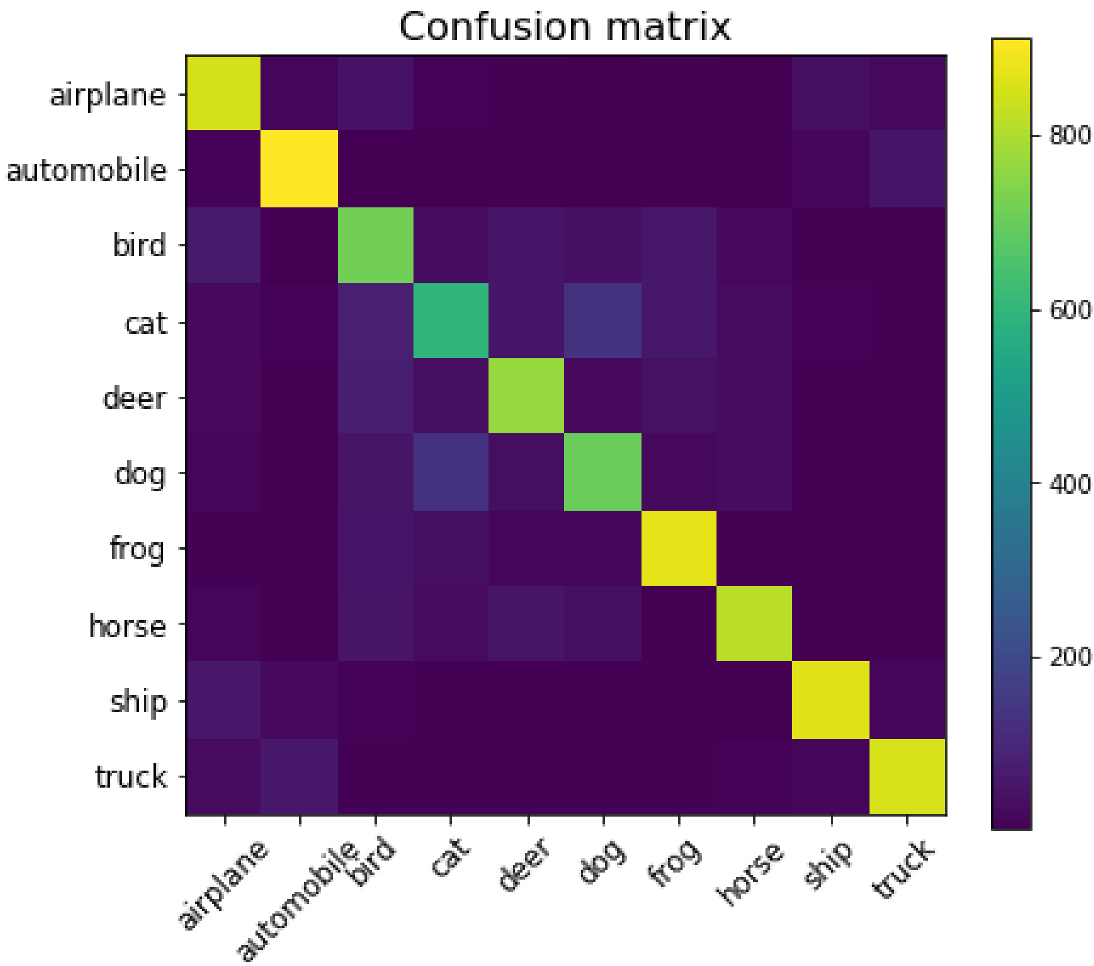
8 Evaluate model

```
In [36]: # make test predictions
         y_pred_test = model.predict_proba(x_test2)
         y_pred_test_classes = np.argmax(y_pred_test, axis=1)
         y_pred_test_max_probas = np.max(y_pred_test, axis=1)
```

```
10000/10000 [==============================] - 41s
```

```
In [37]: # confusion matrix and accuracy
         from sklearn.metrics import confusion_matrix, accuracy_score
         plt.figure(figsize=(7, 6))
         plt.title('Confusion matrix', fontsize=16)
```

```
plt.imshow(confusion_matrix(y_test, y_pred_test_classes))
plt.xticks(np.arange(10), cifar10_classes, rotation=45, fontsize=12)
plt.yticks(np.arange(10), cifar10_classes, fontsize=12)
plt.colorbar()
plt.show()
print("Test accuracy:", accuracy_score(y_test, y_pred_test_classes))
```



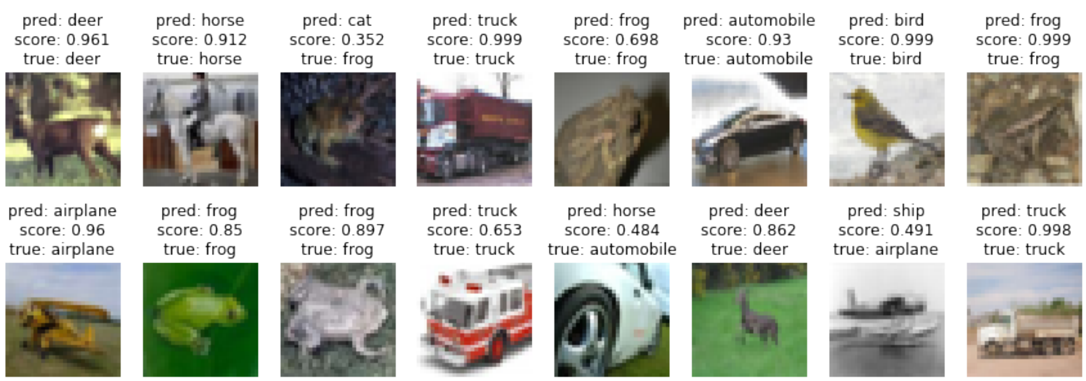

```
Test accuracy: 0.7967
```

In [38]:
```
## GRADED PART, DO NOT CHANGE!
# Accuracy on validation data
grader.set_answer("nQOsg", accuracy_score(y_test, y_pred_test_classes))
```

In [39]:
```
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

In [29]:
```
# inspect preditions
cols = 8
rows = 2
fig = plt.figure(figsize=(2 * cols - 1, 3 * rows - 1))
for i in range(cols):
    for j in range(rows):
        random_index = np.random.randint(0, len(y_test))
        ax = fig.add_subplot(rows, cols, i * rows + j + 1)
        ax.grid('off')
        ax.axis('off')
        ax.imshow(x_test[random_index, :])
        pred_label = cifar10_classes[y_pred_test_classes[random_index]]
        pred_proba = y_pred_test_max_probas[random_index]
        true_label = cifar10_classes[y_test[random_index, 0]]
        ax.set_title("pred: {}\nscore: {:.3}\ntrue: {}".format(
               pred_label, pred_proba, true_label
        ))
plt.show()
```



9 Visualize maximum stimuli

We want to find input images that provide maximum activations for particular layers of our network.

We will find those maximum stimuli via gradient ascent in image space.

For that task we load our model weights, calculate the layer output gradient with respect to image input and shift input image in that direction.

In [15]:
```
s = reset_tf_session()  # clear default graph
K.set_learning_phase(0)  # disable dropout
model = make_model()
model.load_weights("weights.h5")  # that were saved after model.fit
```

In [16]:
```
# all weights we have
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 16)        448
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        4640
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 32, 32, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 32)        9248
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 16, 16, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               1048832
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0
=================================================================
Total params: 1,084,234
Trainable params: 1,084,234
Non-trainable params: 0
_________________________________________________________________
```

In [17]:
```
def find_maximum_stimuli(layer_name, is_conv, filter_index, model, iterations=20, step=
```

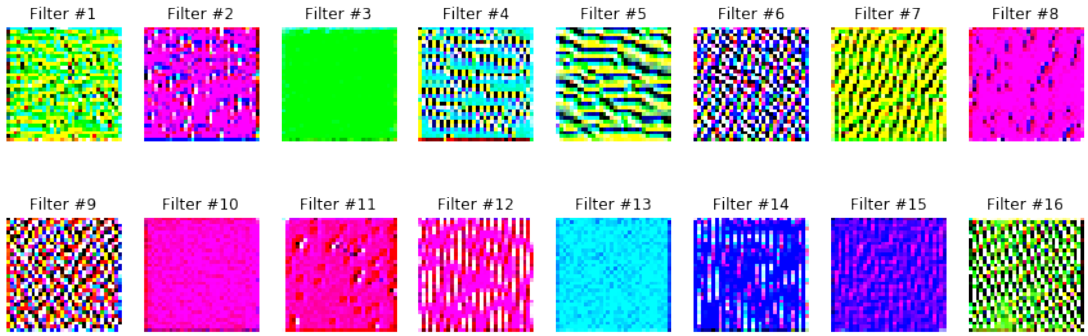
```
def image_values_to_rgb(x):
    # normalize x: center on 0 (np.mean(x_train2)), ensure std is 0.25 (np.std(x_tr
    # so that it looks like a normalized image input for our network
    x = (x - np.mean(x_train2))/np.std(x_train2)     ### YOUR CODE HERE

    # do reverse normalization to RGB values: x = (x_norm + 0.5) * 255
    x = (x + 0.5) * 255      ### YOUR CODE HERE

    # clip values to [0, 255] and convert to bytes
    x = np.clip(x, 0, 255).astype('uint8')
    return x

# this is the placeholder for the input image
input_img = model.input
img_width, img_height = input_img.shape.as_list()[1:3]

# find the layer output by name
layer_output = list(filter(lambda x: x.name == layer_name, model.layers))[0].output

# we build a loss function that maximizes the activation
# of the filter_index filter of the layer considered
if is_conv:
    # mean over feature map values for convolutional layer
    loss = K.mean(layer_output[:, :, :, filter_index])
else:
    loss = K.mean(layer_output[:, filter_index])

# we compute the gradient of the loss wrt input image
grads = K.gradients(loss, input_img)[0]  # [0] because of the batch dimension!

# normalization trick: we normalize the gradient
grads = grads / (K.sqrt(K.sum(K.square(grads))) + 1e-10)

# this function returns the loss and grads given the input picture
iterate = K.function([input_img], [loss, grads])

# we start from a gray image with some random noise
input_img_data = np.random.random((1, img_width, img_height, 3))
input_img_data = (input_img_data - 0.5) * (0.1 if is_conv else 0.001)

# we run gradient ascent
for i in range(iterations):
    loss_value, grads_value = iterate([input_img_data])
    input_img_data += grads_value * step
    if verbose:
        print('Current loss value:', loss_value)
```

```
        # decode the resulting input image
        img = image_values_to_rgb(input_img_data[0])

        return img, loss_value
```

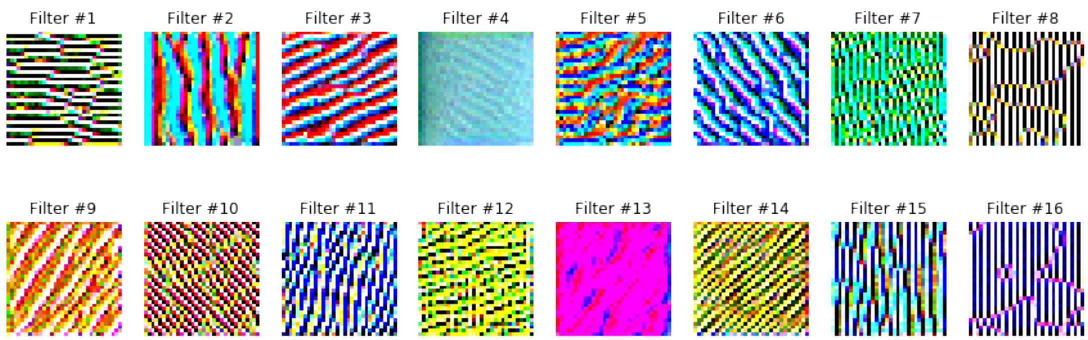
In [25]:
```
# sample maximum stimuli
def plot_filters_stimuli(layer_name, is_conv, model, iterations=20, step=1., verbose=Fa
    cols = 8
    rows = 2
    filter_index = 0
    max_filter_index = list(filter(lambda x: x.name == layer_name, model.layers))[0].ou
    fig = plt.figure(figsize=(2 * cols - 1, 3 * rows - 1))
    for i in range(cols):
        for j in range(rows):
            if filter_index <= max_filter_index:
                ax = fig.add_subplot(rows, cols, i * rows + j + 1)
                ax.grid('off')
                ax.axis('off')
                loss = -1e20
                while loss < 0 and filter_index <= max_filter_index:
                    stimuli, loss = find_maximum_stimuli(layer_name, is_conv, filter_in
                                                         iterations, step, verbose=verb
                    filter_index += 1
                if loss > 0:
                    ax.imshow(stimuli)
                    ax.set_title("Filter #{}".format(filter_index))
    plt.show()
```

In [26]:
```
# maximum stimuli for convolutional neurons
conv_activation_layers = []
for layer in model.layers:
    if isinstance(layer, LeakyReLU):
        prev_layer = layer.inbound_nodes[0].inbound_layers[0]
        if isinstance(prev_layer, Conv2D):
            conv_activation_layers.append(layer)

for layer in conv_activation_layers:
    print(layer.name)
    plot_filters_stimuli(layer_name=layer.name, is_conv=True, model=model)
```

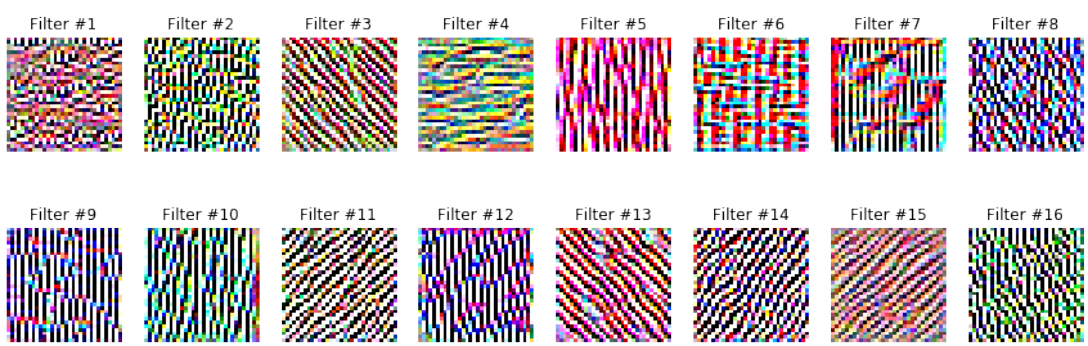
```
leaky_re_lu_1
```



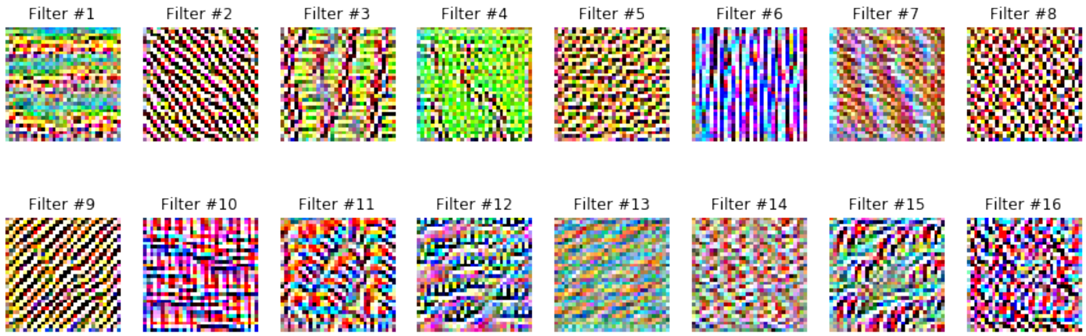
`leaky_re_lu_2`



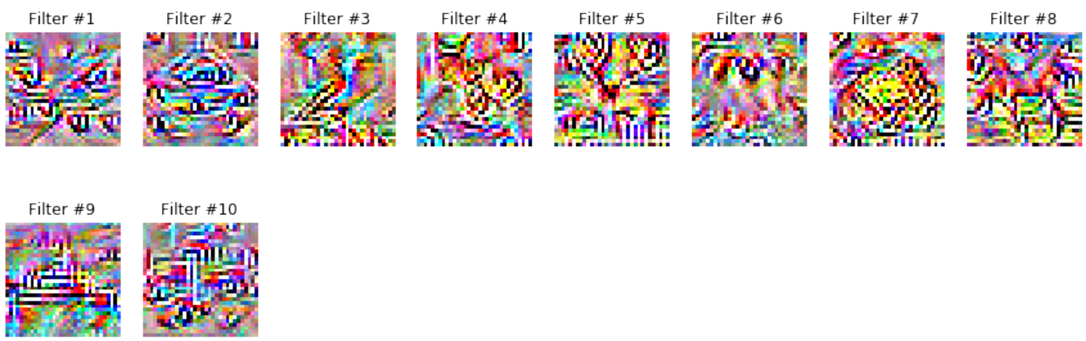
`leaky_re_lu_3`



leaky_re_lu_4



In [27]:
```
# maximum stimuli for last dense layer
last_dense_layer = list(filter(lambda x: isinstance(x, Dense), model.layers))[-1]
plot_filters_stimuli(layer_name=last_dense_layer.name, is_conv=False,
                     iterations=200, step=0.1, model=model)
```



In [28]:
```
def maximum_stimuli_test_for_grader():
    layer = list(filter(lambda x: isinstance(x, Dense), model.layers))[-1]
    output_index = 7
    stimuli, loss = find_maximum_stimuli(
        layer_name=layer.name,
        is_conv=False,
        filter_index=output_index,
        model=model,
        verbose=False
    )
    return model.predict_proba(stimuli[np.newaxis, :])[0, output_index]
```

```
In [29]: ## GRADED PART, DO NOT CHANGE!
         # Maximum stimuli test
         grader.set_answer("96eco", maximum_stimuli_test_for_grader())
```

```
1/1 [==============================] - 0s
```

```
In [30]: # you can make submission with answers so far to check yourself at this stage
         grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

That's it! Congratulations!

What you've done: - defined CNN architecture - trained your model - evaluated your model - visualised learnt filters