In [1]:
```
# set tf 1.x for colab
%tensorflow_version 1.x
```

```
UsageError: Line magic function `%tensorflow_version` not found.
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore', category=DeprecationWarning)
warnings.filterwarnings('ignore', category=FutureWarning)
```

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**:

Import stuff

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

In [4]:
```
# !!! remember to clear session/graph if you rebuild your graph to avoid out-of-memory errors !!!
```

In [5]:
```
download_utils.link_all_keras_resources()
```

In [6]:
```
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.

1.14.0
2.3.1
```

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

In [8]:
```
# token expires every 30 min
COURSERA_TOKEN = "elDwxJj9sVxfld8n"
COURSERA_EMAIL = "lxwvictor@gmail.com"
```

Load dataset

In [9]:

```
from keras.datasets import cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

In [10]:

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

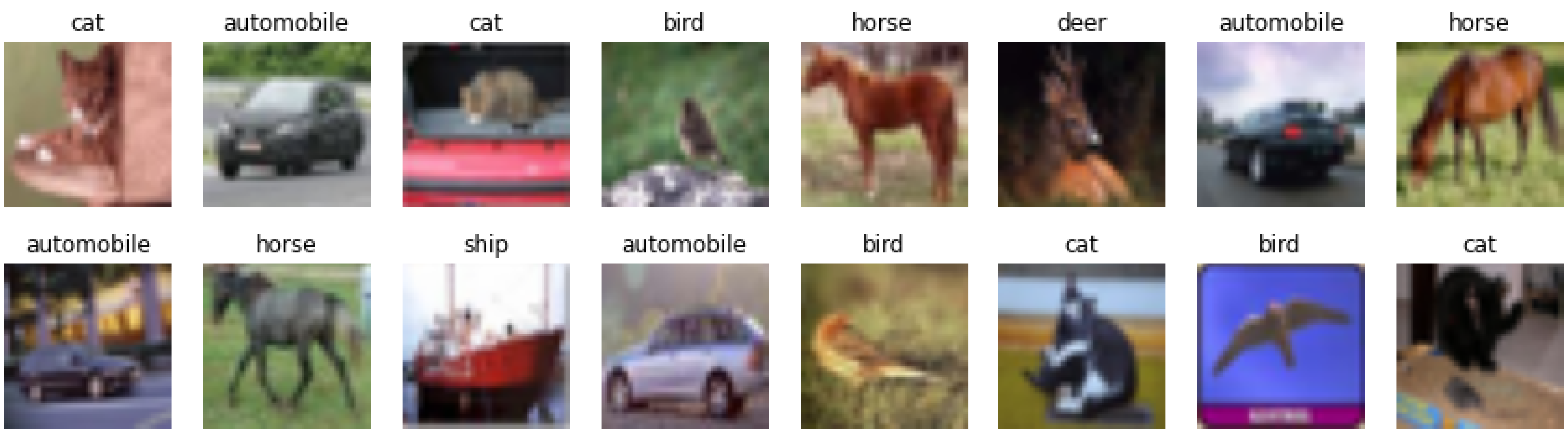
```
Train samples: (50000, 32, 32, 3) (50000, 1)
Test samples: (10000, 32, 32, 3) (10000, 1)
```

In [11]:

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

In [12]:

```
# 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()
```



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 [13]:

```
# normalize inputs
### YOUR CODE HERE
x_train2 = x_train/255 - 0.5
### YOUR CODE HERE
x_test2 = x_test/255 - 0.5

# convert class labels to one-hot encoded, should have shape (?, NUM_CLASSES)
### YOUR CODE HERE
y_train2 = keras.utils.to_categorical(y_train, 10)
### YOUR CODE HERE
y_test2 = keras.utils.to_categorical(y_test, 10)
```

Define CNN architecture

In [14]:

```
# 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 (https://keras.io/layers/convolutional/#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 (https://keras.io/layers/pooling/#maxpooling2d) - performs 2D max pooling.
- Flatten (https://keras.io/layers/core/#flatten) - flattens the input, does not affect the batch size.
- Dense (https://keras.io/layers/core/#dense) - fully-connected layer.
- Activation (https://keras.io/layers/core/#activation) - applies an activation function.
- LeakyReLU (https://keras.io/layers/advanced-activations/#leakyrelu) - applies leaky relu activation.
- Dropout (https://keras.io/layers/core/#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:

```
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):

```
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**).

In [15]:

```
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(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(0.25))
    model.add(Flatten())
    model.add(Dense(256))
    model.add(LeakyReLU(0.1))
    model.add(Dropout(0.25))
    model.add(Dense(10))
    model.add(Activation('softmax'))

    return model
```

In [16]:
```
# describe model
s = reset_tf_session()  # clear default graph
model = make_model()
model.summary()
```

```
WARNING:tensorflow:From ..\keras_utils.py:68: The name tf.get_default_session is deprecated. Please use tf.compat.v
1.get_default_session instead.

WARNING:tensorflow:From ..\keras_utils.py:75: The name tf.ConfigProto is deprecated. Please use tf.compat.v1.ConfigP
roto instead.

WARNING:tensorflow:From ..\keras_utils.py:77: The name tf.InteractiveSession is deprecated. Please use tf.compat.v1.
InteractiveSession instead.

WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.p
y:4070: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 16)        448       
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 32, 32, 16)        0         
```

In [17]:
```
## GRADED PART, DO NOT CHANGE!
# Number of model parameters
grader.set_answer("7W4tu", grading_utils.model_total_params(model))
```

In [18]:
```
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
You used an invalid email or your token may have expired. Please make sure you have entered all fields correctly. Try g
enerating a new token if the issue still persists.
```

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 [19]:
```
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 simultaneously)
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 [20]:
```
# 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 [21]:

```
%%time
# 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
)
```

```
WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.py:4
22: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

Learning rate: 0.005

Epoch 1/10
```

loss: 1.3007; accuracy: 0.4351; val_loss: 0.9630; … 1564/? [01:17<00:00, 20.21it/s]

```
Model saved in cifar.000.hdf5
Learning rate: 0.0045

Epoch 2/10
```

loss: 0.8946; accuracy: 0.6697; val_loss: 0.7938; … 1564/? [01:00<00:00, 25.64it/s]

```
Model saved in cifar.001.hdf5
Learning rate: 0.00405

Epoch 3/10
```

loss: 0.7399; accuracy: 0.7398; val_loss: 0.7303; … 1564/? [00:45<00:00, 34.27it/s]

```
Model saved in cifar.002.hdf5
Learning rate: 0.003645

Epoch 4/10
```

loss: 0.6450; accuracy: 0.7757; val_loss: 0.6835; … 1564/? [00:30<00:00, 51.45it/s]

```
Model saved in cifar.003.hdf5
Learning rate: 0.0032805

Epoch 5/10
```

loss: 0.5631; accuracy: 0.8071; val_loss: 0.6801; … 1564/? [02:14<00:00, 11.64it/s]

```
Model saved in cifar.004.hdf5
Learning rate: 0.00295245

Epoch 6/10
```

loss: 0.5077; accuracy: 0.8228; val_loss: 0.6367; … 1564/? [00:30<00:00, 51.68it/s]

```
Model saved in cifar.005.hdf5
Learning rate: 0.002657205

Epoch 7/10
```

loss: 0.4554; accuracy: 0.8401; val_loss: 0.6342; … 1564/? [01:43<00:00, 15.08it/s]

```
Model saved in cifar.006.hdf5
```

```
Learning rate: 0.0023914846

Epoch 8/10
```

loss: 0.4118; accuracy: 0.8563; val_loss: 0.6662; … 1564/? [01:28<00:00, 17.63it/s]

```
Model saved in cifar.007.hdf5
Learning rate: 0.002152336

Epoch 9/10
```

loss: 0.3770; accuracy: 0.8674; val_loss: 0.6303; … 1564/? [00:15<00:00, 102.23it/s]

```
Model saved in cifar.008.hdf5
Learning rate: 0.0019371024

Epoch 10/10
```

loss: 0.3444; accuracy: 0.8814; val_loss: 0.6614; … 1564/? [00:58<00:00, 26.61it/s]

```
Model saved in cifar.009.hdf5
Wall time: 2min 33s
```

Out[21]:
```
<keras.callbacks.callbacks.History at 0x26905c17f48>
```

In [22]:
```python
# save weights to file
model.save_weights("weights.h5")
```

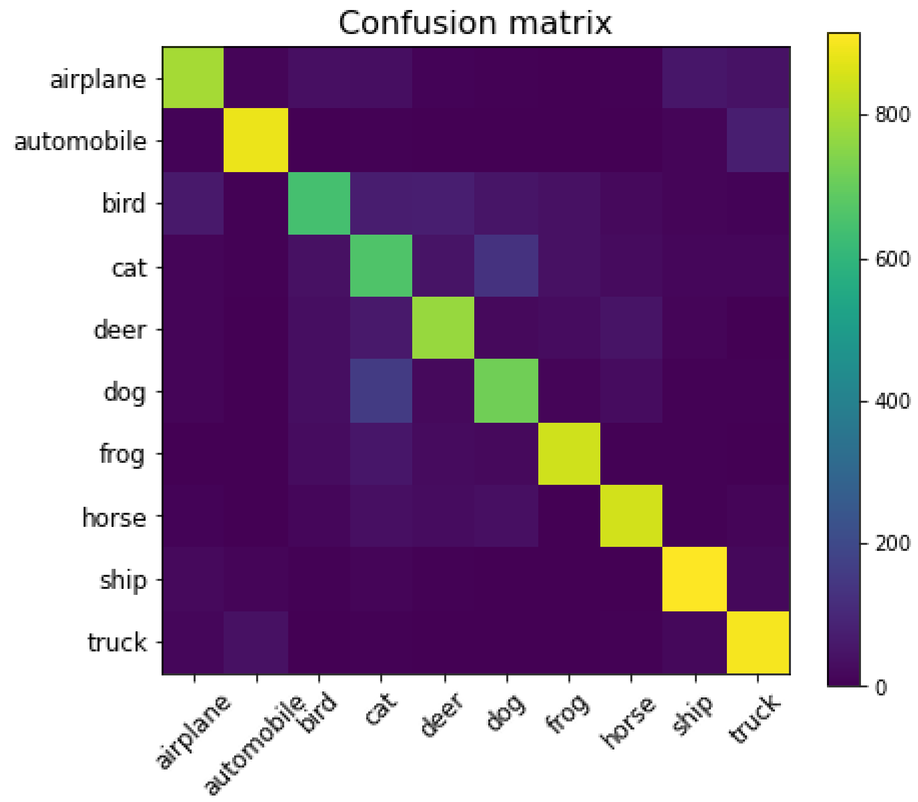
In [23]:
```python
# load weights from file (can call without model.fit)
model.load_weights("weights.h5")
```

Evaluate model

In [24]:
```python
# 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)
```

In [25]:

```
# 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: 0.7993
```

In [26]:

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

In [27]:

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

```
You used an invalid email or your token may have expired. Please make sure you have entered all fields correctly. Try g
enerating a new token if the issue still persists.
```

In [28]:

```
# 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()
```

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 [29]:

```
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 [30]:

```
# all weights we have
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
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 [31]:

```python
def find_maximum_stimuli(layer_name, is_conv, filter_index, model, iterations=20, step=1., verbose=True):

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

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

        # 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 [32]:

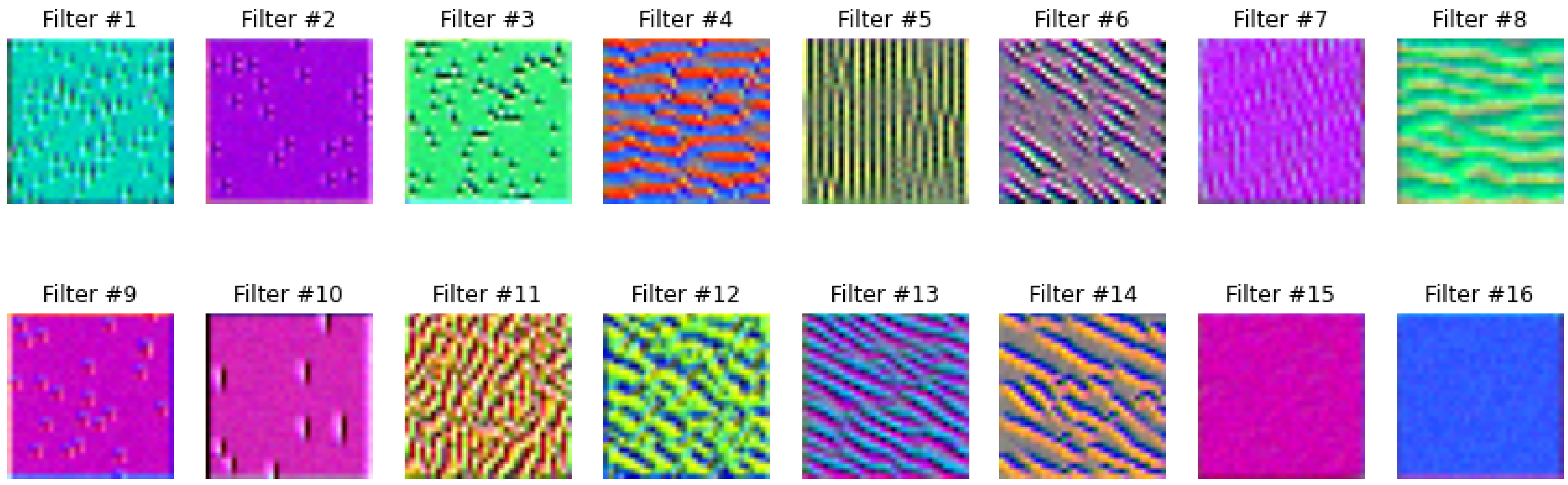
```python
# sample maximum stimuli
def plot_filters_stimuli(layer_name, is_conv, model, iterations=20, step=1., verbose=False):
    cols = 8
    rows = 2
    filter_index = 0
    max_filter_index = list(filter(lambda x: x.name == layer_name, model.layers))[0].output.shape.as_list()[-1] - 1
    fig = 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_index, model,
                                                         iterations, step, verbose=verbose)
                    filter_index += 1
                if loss > 0:
                    ax.imshow(stimuli)
                    ax.set_title("Filter #{}".format(filter_index))
    plt.show()
```

In [33]:

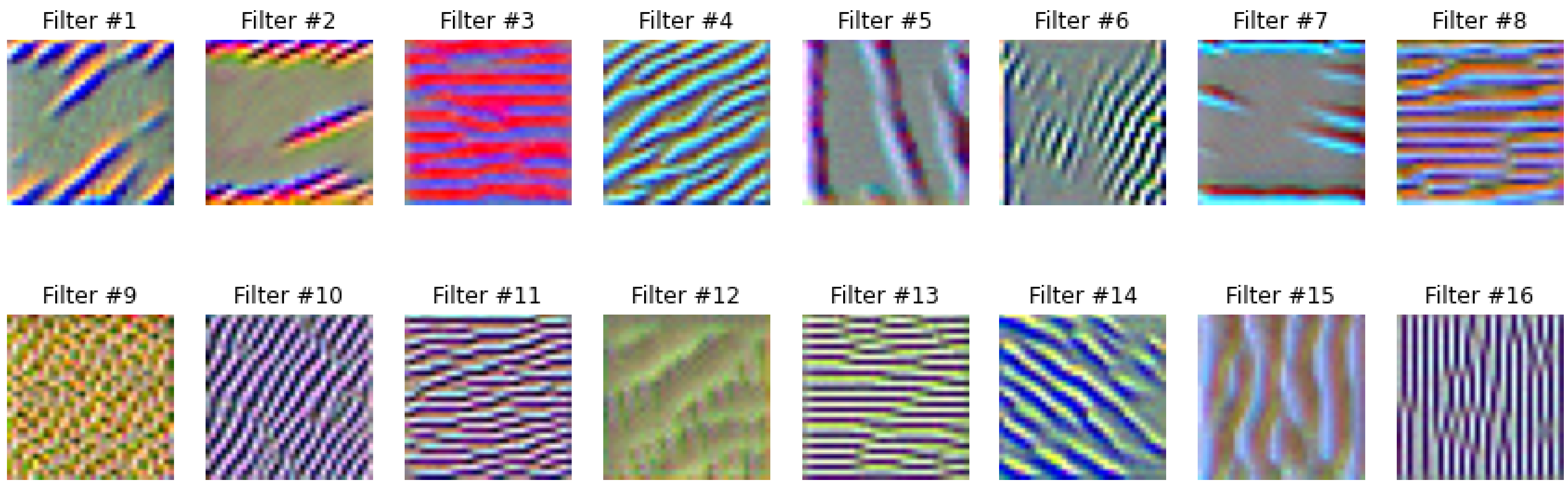
```python
# maximum stimuli for convolutional neurons
conv_activation_layers = []
for layer in model.layers:
    if isinstance(layer, LeakyReLU):
        prev_layer = layer._inbound_nodes[0].inbound_layers[0]
        if isinstance(prev_layer, Conv2D):
            conv_activation_layers.append(layer)

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

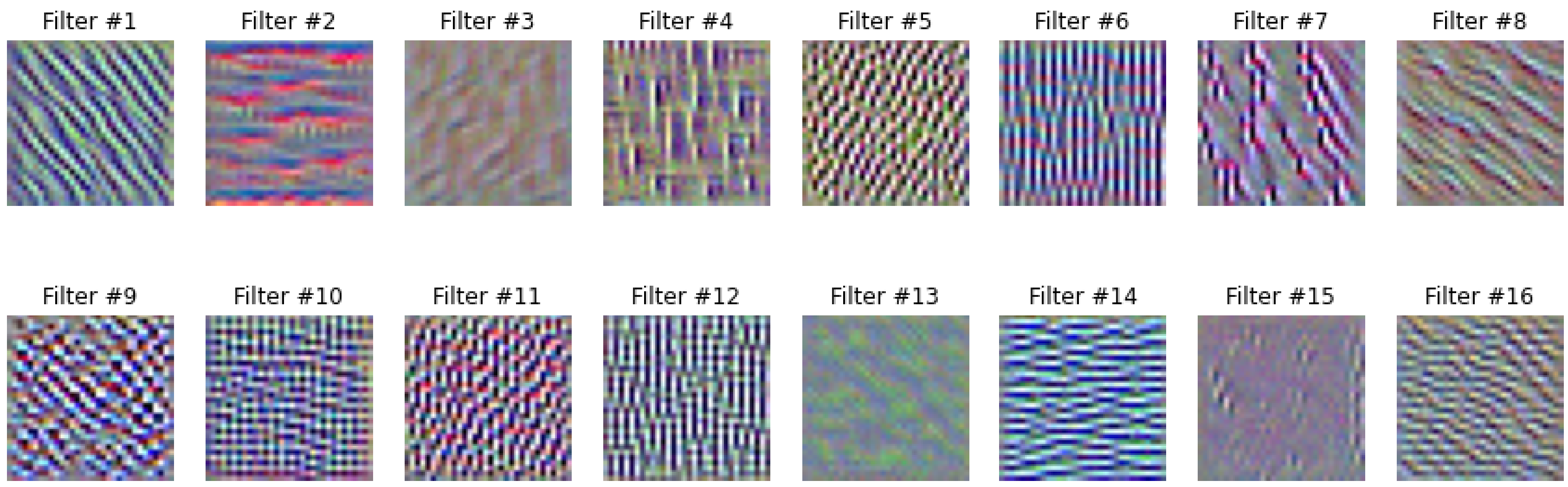
```
leaky_re_lu_1
```



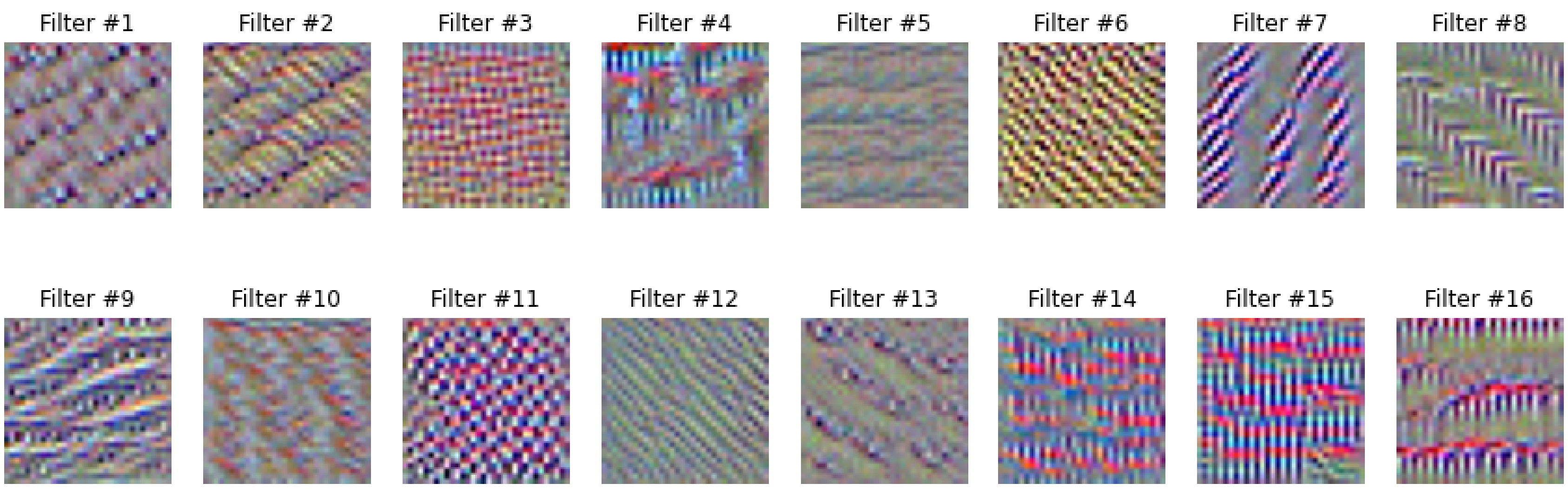

```
leaky_re_lu_2
```




```
leaky_re_lu_3
```

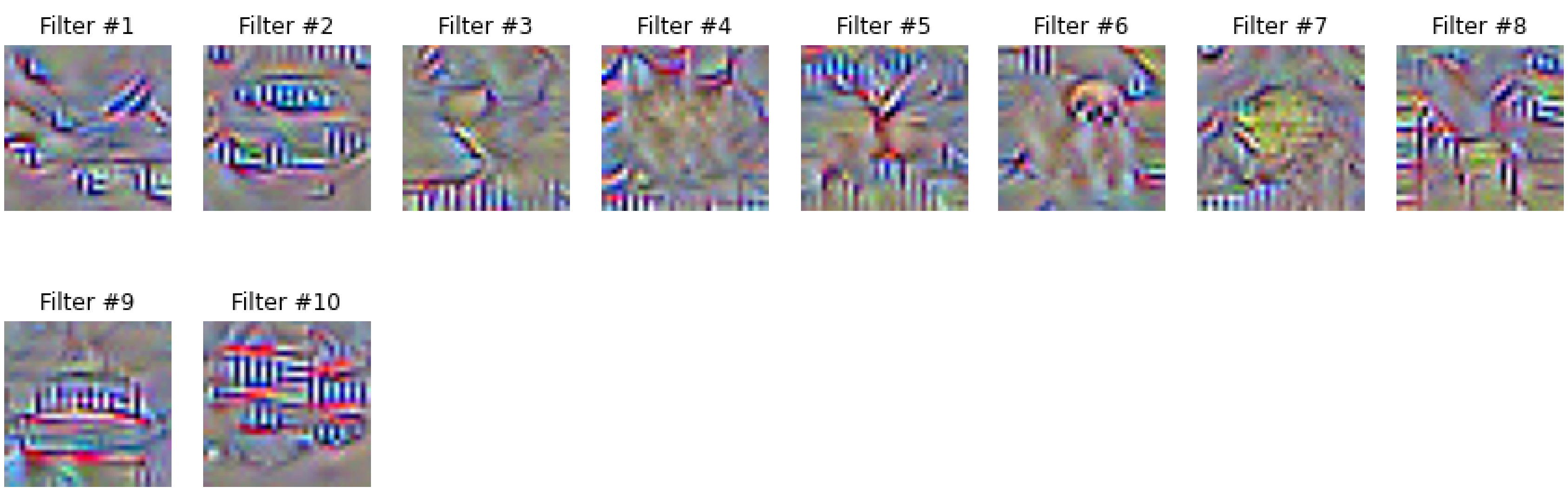


leaky_re_lu_4



In [34]:

```
# 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 [35]:

```
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 [36]:

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

In [36]:

```
# 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