In [9]:

```
import numpy as np
import matplotlib.pyplot as plt

from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import Adam, RMSprop
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
```

In [ ]:

```
from keras.datasets import mnist
```

In [ ]:

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [ ]:

```
# data pre-processing
# flat the input data
X_train = X_train.reshape(X_train.shape[0], -1) / 255.    # normalize
X_test = X_test.reshape(X_test.shape[0], -1) / 255.       # normalize

# one-hot output data
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

In [ ]:

```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images

datagen.fit(X_train)     # model.fit_generator can fit the model using flow,
                         # this datagen is not flow yet,
                         # it just need to fit the data first to get some of its functionality worki
```

In [36]:

```
model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (28,28,1)))
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))


model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation = "softmax"))


model.summary()
# Define the optimizer
optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)

# Compile the model
model.compile(optimizer = optimizer , loss = "categorical_crossentropy", metrics=["accuracy"])
```

...

In [ ]:

```
# Set a learning rate annealer
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)

# Fit the model
batch_size = 86
model.fit_generator(datagen.flow(X_train, Y_train, batch_size=batch_size),
                    epochs = epochs,
                    validation_data = (X_test,Y_test),
                    steps_per_epoch=X_train.shape[0] // batch_size,
                    callbacks=[learning_rate_reduction])
```

In [ ]:

```
# verbose: Integer. 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch
# epochs is to be understood as "final epoch".
# validation_data will override validation_split

results = model.predict(X_test)
```

examples

cats and dogg

In [2]:

```
import os
PROJECT_FOLDER = os.path.join(os.path.dirname(os.getcwd()), "storage")
os.chdir(PROJECT_FOLDER)
print(os.getcwd())
```

```
D:\PyProjects\Machine-Learning-Tech\storage
```

In [3]:

```
train_path = os.path.join("dogs-vs-cats(smaller)", "train")
valid_path = os.path.join("dogs-vs-cats(smaller)", "valid")
test_path = os.path.join("dogs-vs-cats(smaller)", "test")
```

In [4]:

```
train_batches = ImageDataGenerator().flow_from_directory(train_path,
                                                         target_size=(224, 224),
                                                         classes=["dogs", "cats"], # define which is
                                                         batch_size=10)

valid_batches = ImageDataGenerator().flow_from_directory(valid_path,
                                                         target_size=(224, 224),
                                                         classes=["dogs", "cats"],
                                                         batch_size=4)

test_batches = ImageDataGenerator().flow_from_directory(test_path,
                                                         target_size=(224, 224),
                                                         classes=["dogs", "cats"],
                                                         batch_size=10)
```

```
Found 506 images belonging to 2 classes.
Found 400 images belonging to 2 classes.
Found 440 images belonging to 2 classes.
```

In [5]:

```
def plots(ims, figsize=(12,6), rows=1, interp=False, titles=None):
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims).astype(np.uint8)
        if (ims.shape[-1] != 3):
            ims = ims.transpose((0,2,3,1))
    f = plt.figure(figsize=figsize)
    cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
    for i in range(len(ims)):
        sp = f.add_subplot(rows, cols, i+1)
        sp.axis('Off')
        if titles is not None:
            sp.set_title(titles[i], fontsize=16)
        plt.imshow(ims[i], interpolation=None if interp else 'none')
```

In [6]:

```
imgs, labels = next(train_batches)
plots(imgs, titles=labels)
```

...

-- model --

In [35]:

```
model = Sequential([
    Conv2D(32,
           kernel_size=(3,3),
           strides=(1,1),   # default
           padding="same", # default is "valid", which means no padding
           input_shape=(224, 224, 3),
           activation="relu"),
    Flatten(),
    Dense(2, activation="softmax")
])

model.summary()
```

...

In [13]:

```
model.compile(Adam(lr=.0001),
              loss="categorical_crossentropy",
              metrics=["accuracy"]
             )

model.fit_generator(train_batches,
                    epochs=12,
                    steps_per_epoch=4,

                    validation_data=valid_batches,
                    validation_steps=4)
```

...

evaluate

In [31]:

```
test_imgs, test_labels = next(test_batches)
plots(test_imgs, titles=test_labels)
```

...

In [32]:

```
predictions = model.predict_classes(test_imgs, verbose=0)
# predictions = np.array([to_categorical(p, 2) for p in predictions])

plots(test_imgs, titles=predictions)
```

...

In [33]:

```
predictions = model.predict_generator(test_batches, steps=1, verbose=0)

# print(predictions)
plots(test_imgs, titles=predictions)
```

...

In [ ]:

```
train_batchs.class_indices
```

In [ ]:

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```