**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МО ЭВМ**

# ОТЧЕТ по лабораторной работе №5

**по дисциплине «Искусственные нейронные сети»**

**Тема: Распознавание объектов на фотографиях**

Студентка гр. 7382 Лящевская А.П.

Преподаватель Жукова Н. А.

Санкт-Петербург
2020

**Цель работы.**

Реализовать классификацию небольших изображений по десяти классам: самолет, автомобиль, птица, кошка, олень, собака, лягушка, лошадь, корабль и грузовик).

**Выполнение работы.**

Изначально была создана сверточная сеть использующая сверточные слои, слои maxpooling и слои разреживания dropout. Количество слоев и их параметры представлены в коде программы. Код программы представлен в приложении А.

На рис. 1-2 представлены результаты выполнения программы. С данной архитектурой сети достигается точность 77%.

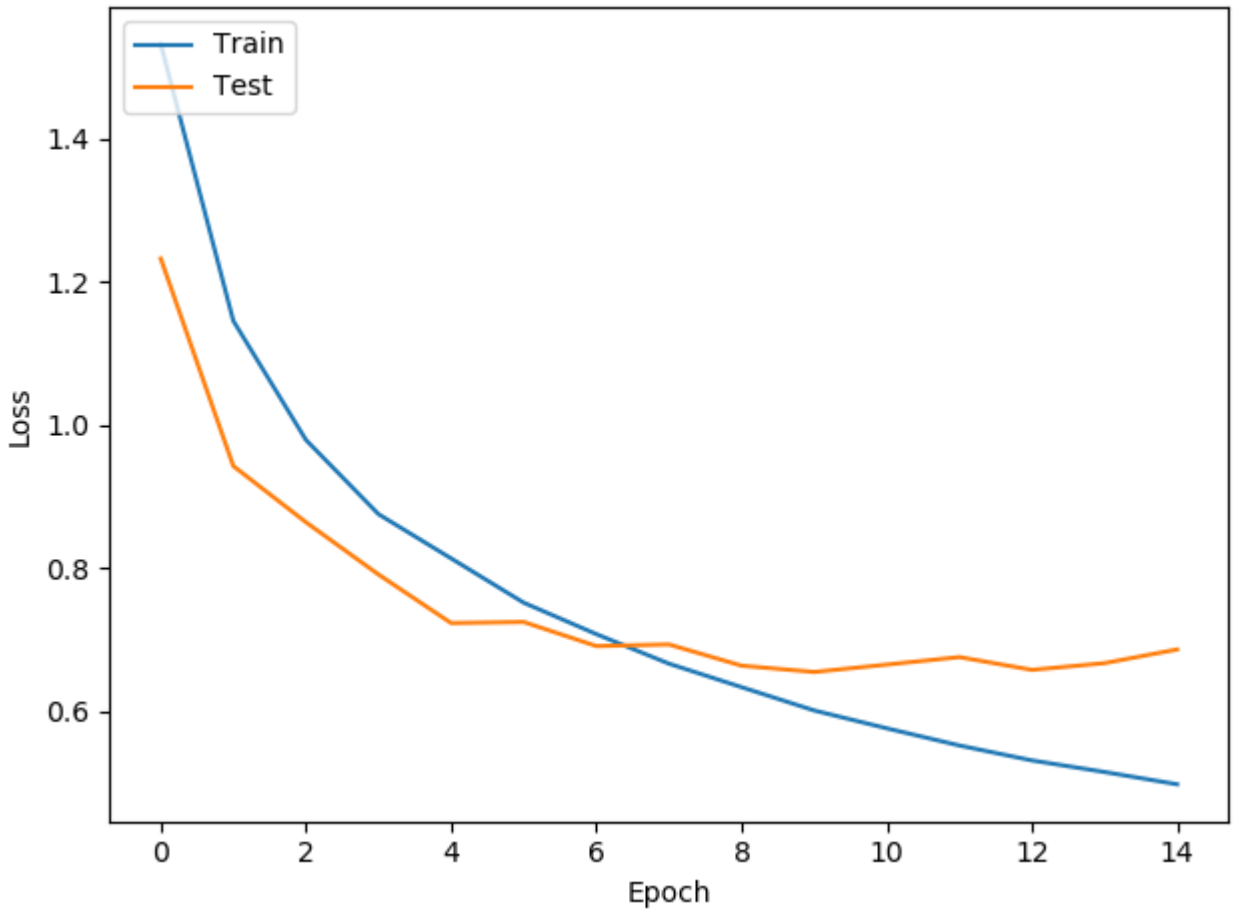


Рисунок 1 – График потерь для базовой модели

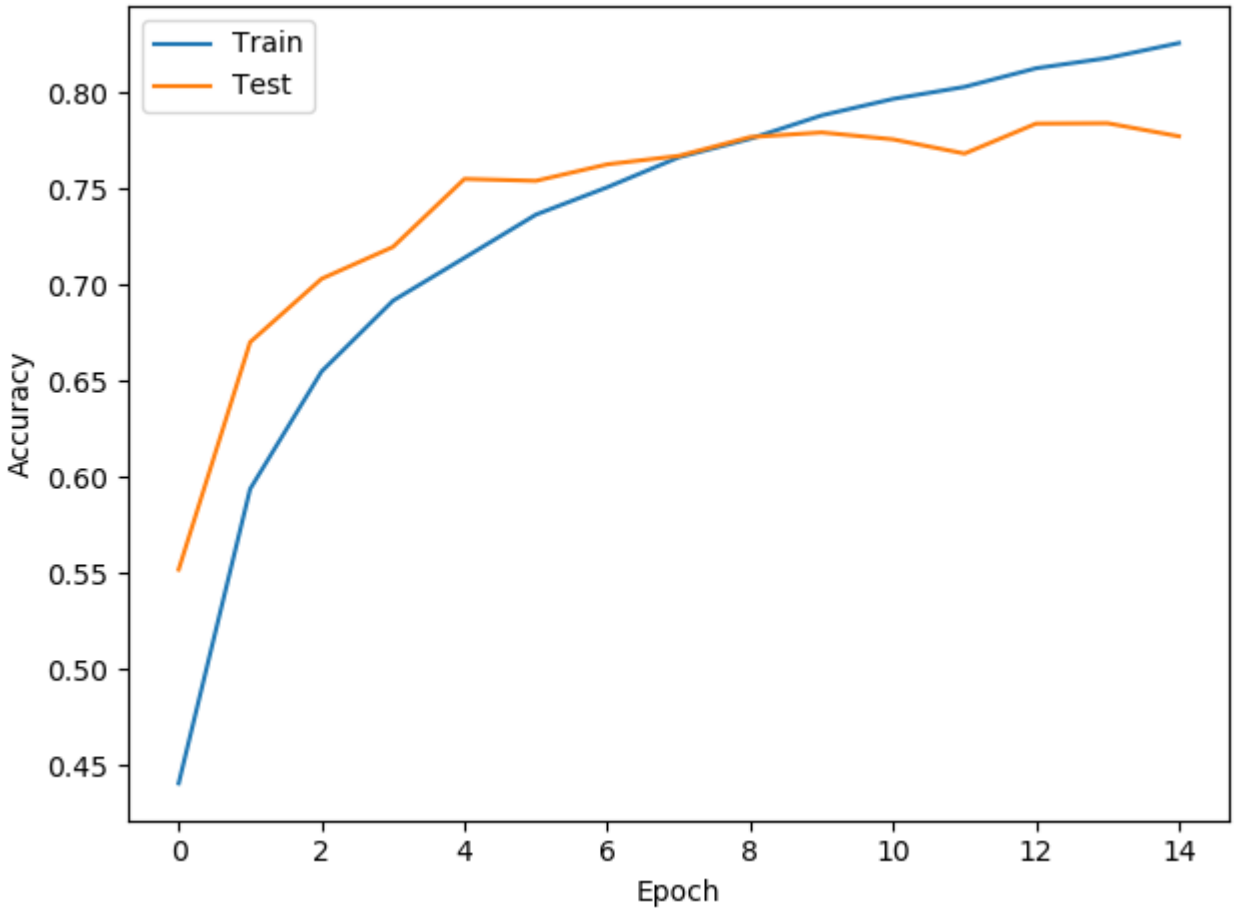


Рисунок 2 – График точности для базовой модели

Далее были убраны слои прореживания dropout, для того чтобы изучить как поведет себя модель без данных слоев. Результаты представлены на рис. 3-4.

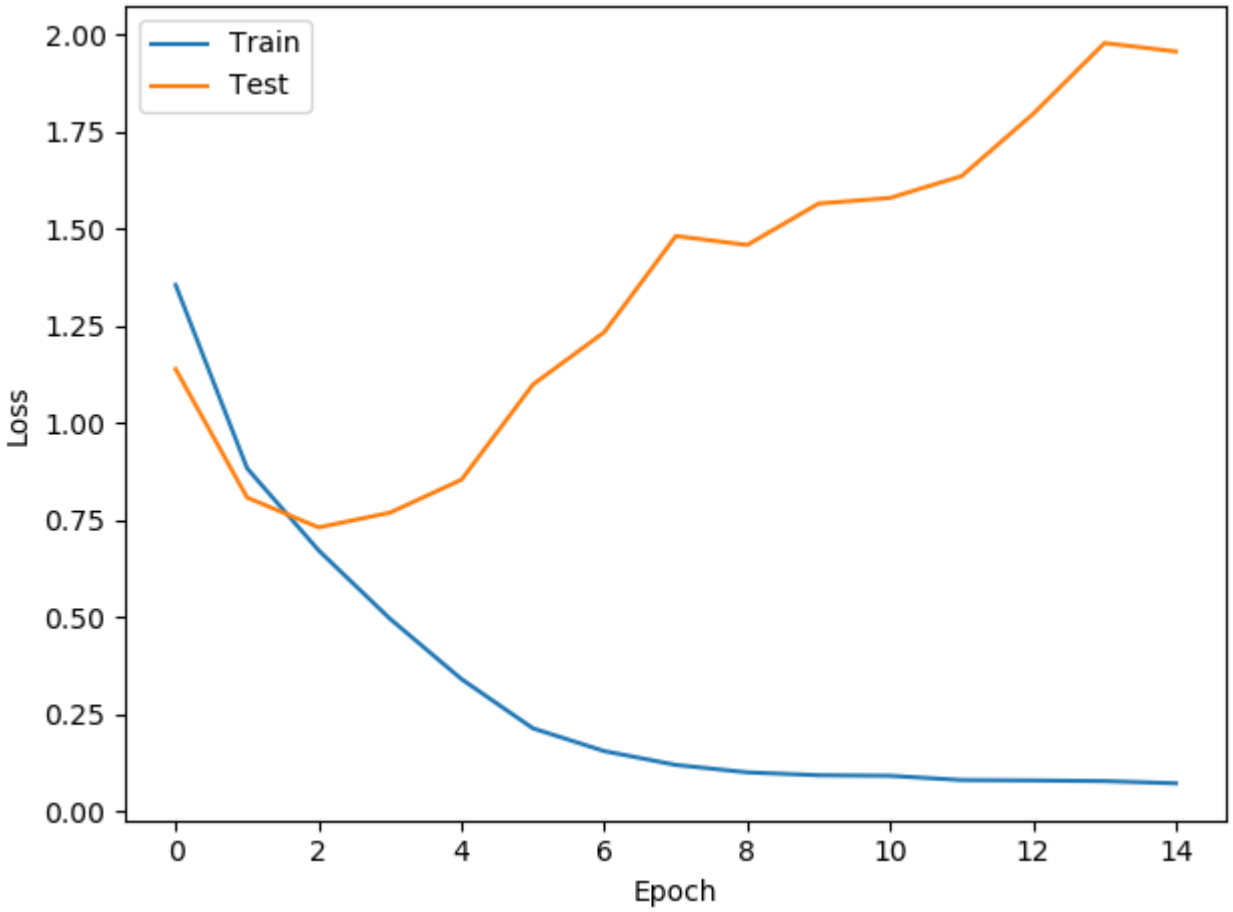


Рисунок 3 – График потерь для модели без dropout слоев

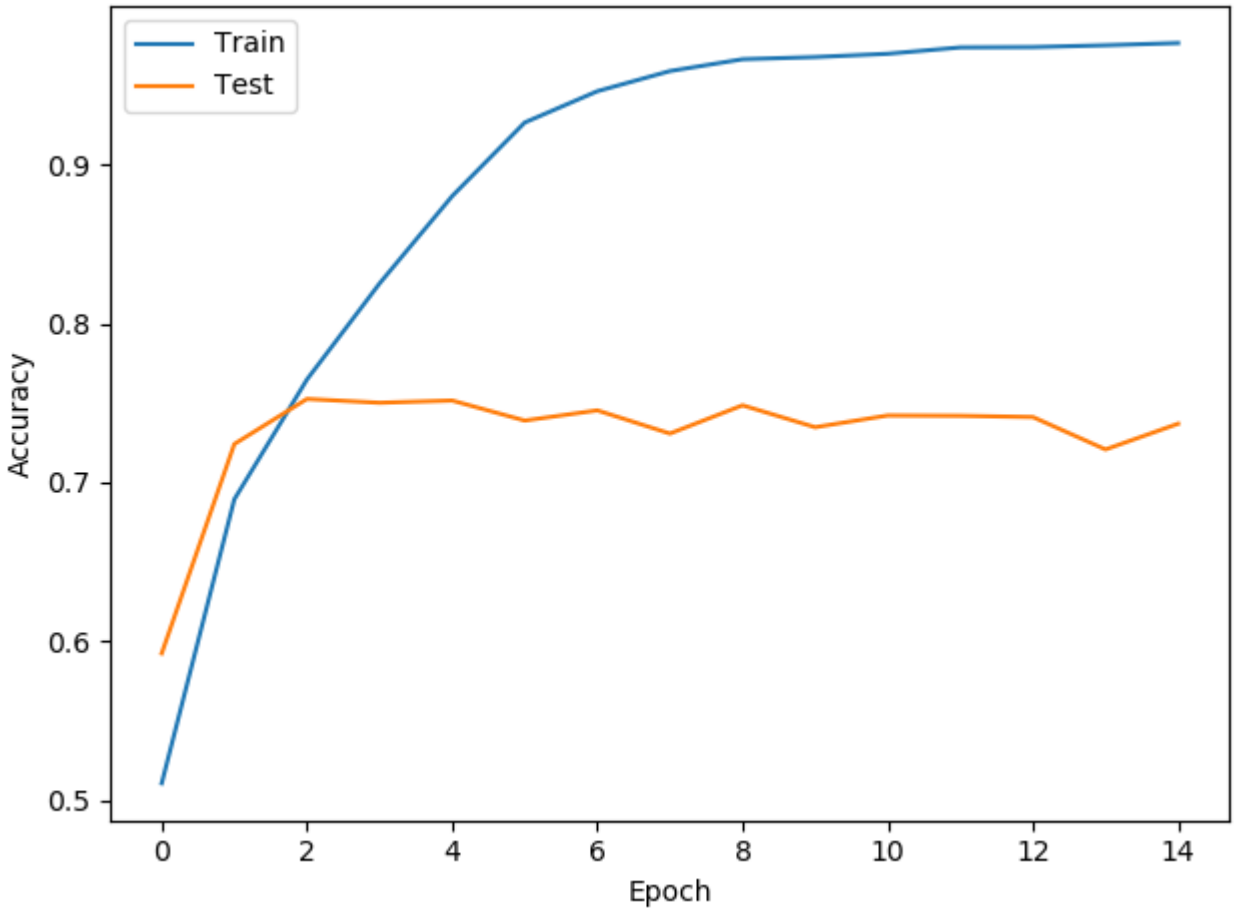


Рисунок 4 – График точности для модели без dropout слоев

Из графиков видно, что после 2 эпохи потери начали расти, а точность перестала повышаться, из чего можно сделать вывод о необходимости слоев прореживания.

Далее была изучена архитектура, с формой ядра свертки имеет (5,5). Результаты представлены на рис. 5-6.

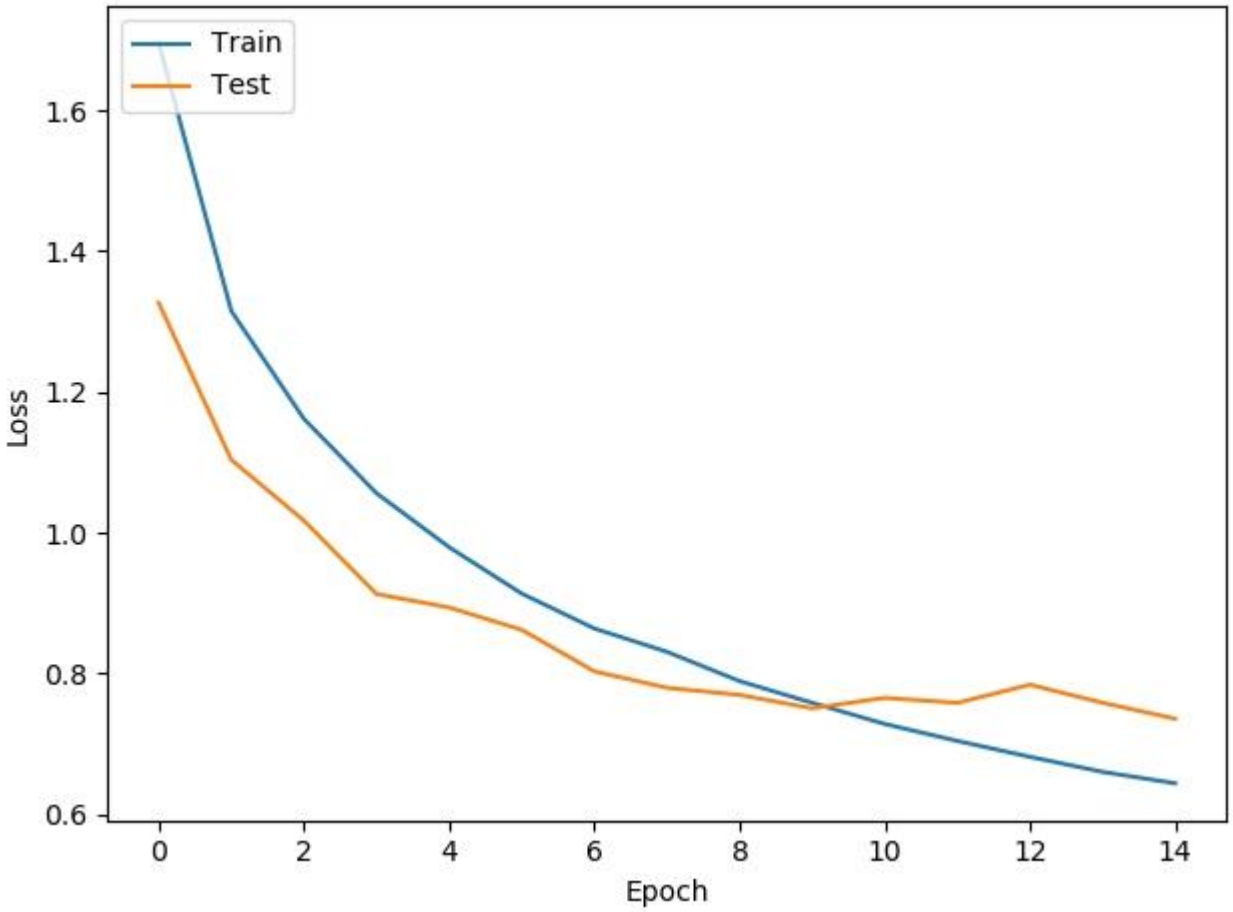


Рисунок 5 – График потерь для модели с размером ядра свертки (5,5)

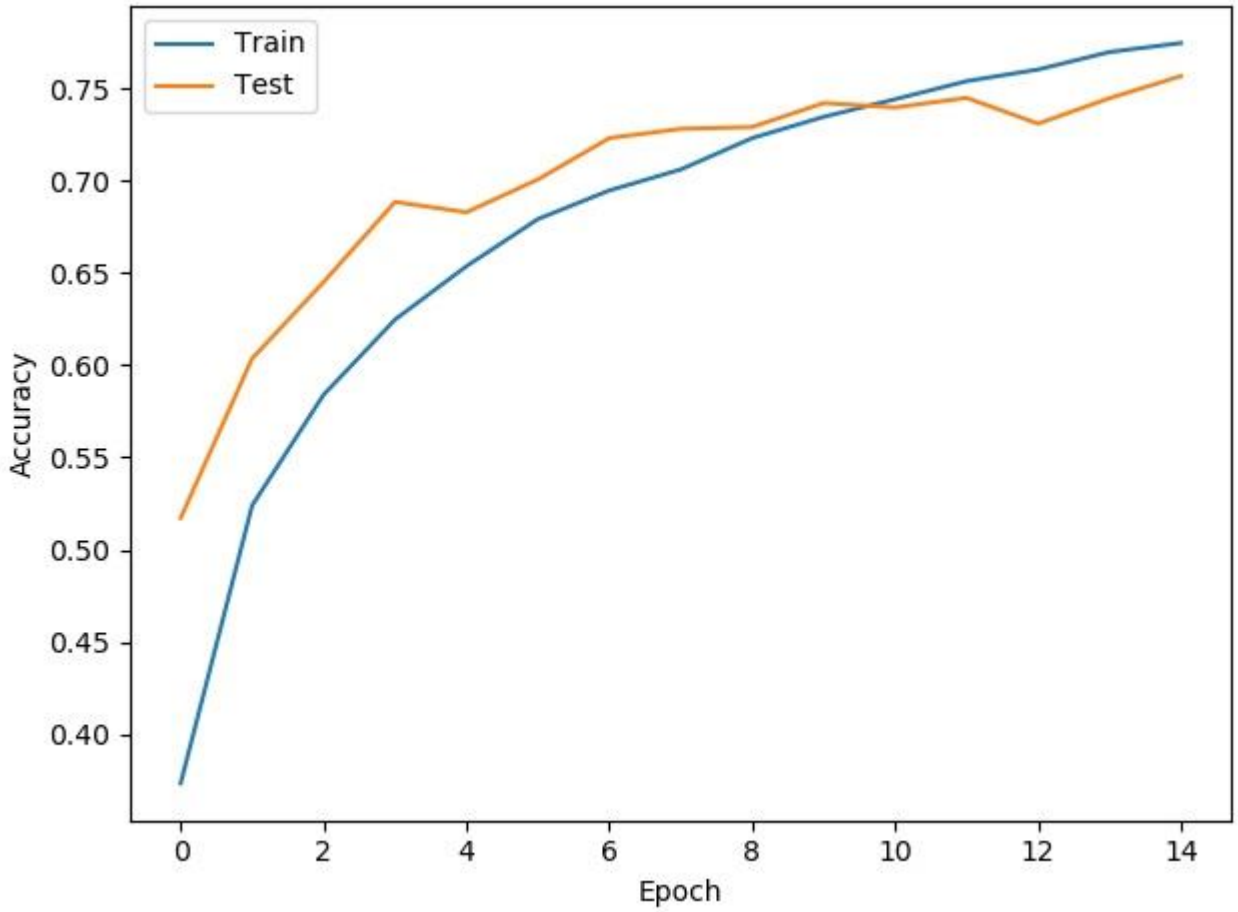


Рисунок 6 – График точности для модели с размером ядра свертки (5,5)

По графикам на рис. 5-6 заметно, что при увеличении размера ядра свертки, падает точность (75%) и возрастает ошибка.

**Выводы.**

В ходе выполнения данной работы была создана сеть для классификации изображений, были более подробно изучены сверточные сети, влияние слоев разреживания и ядра свертки на результаты обучения.

# ПРИЛОЖЕНИЯ ПРИЛОЖЕНИЕ А: КОД ПРОГРАММЫ

```
from keras.datasets import cifar10
from keras.models import Model
from keras.layers import Input, Convolution2D,
MaxPooling2D, Dense, Dropout, Flatten
from keras.utils import np_utils
import numpy as np
import matplotlib.pyplot as plt


batch_size = 32 # in each iteration, we consider
32 training examples at once
num_epochs = 15 # we iterate 200 times over the
entire training set
kernel_size = 5 # we will use 3x3 kernels
throughout
pool_size = 2 # we will use 2x2 pooling throughout
conv_depth_1 = 32 # we will initially have 32
kernels per conv. layer...
conv_depth_2 = 64 # ...switching to 64 after the
first pooling layer
drop_prob_1 = 0.25 # dropout after pooling with
probability 0.25
drop_prob_2 = 0.5 # dropout in the dense layer
with probability 0.5
hidden_size = 512 # the dense layer will have 512
neurons


(X_train, y_train), (X_test, y_test) =
cifar10.load_data() # fetch CIFAR-10 data

num_train, depth, height, width = X_train.shape #
there are 50000 training examples in CIFAR-10
num_test = X_test.shape[0] # there are 10000 test
examples in CIFAR-10
num_classes = np.unique(y_train).shape[0] # there
are 10 image classes

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= np.max(X_train) # Normalise data to [0,
1] range
```

```
X_test /= np.max(X_train) # Normalise data to [0,
1] range

Y_train = np_utils.to_categorical(y_train,
num_classes) # One-hot encode the labels
Y_test = np_utils.to_categorical(y_test,
num_classes) # One-hot encode the labels

inp = Input(shape=(depth, height, width)) # N.B.
depth goes first in Keras


# Conv [32] -> Conv [32] -> Pool (with dropout on
the pooling layer)
conv_1 = Convolution2D(conv_depth_1, kernel_size,
kernel_size, border_mode='same',
activation='relu')(inp)
conv_2 = Convolution2D(conv_depth_1, kernel_size,
kernel_size, border_mode='same',
activation='relu')(conv_1)
pool_1 = MaxPooling2D(pool_size=(pool_size,
pool_size))(conv_2)
drop_1 = Dropout(drop_prob_1)(pool_1)

# Conv [64] -> Conv [64] -> Pool (with dropout on
the pooling layer)
conv_3 = Convolution2D(conv_depth_2, kernel_size,
kernel_size, border_mode='same',
activation='relu')(drop_1)
conv_4 = Convolution2D(conv_depth_2, kernel_size,
kernel_size, border_mode='same',
activation='relu')(conv_3)
pool_2 = MaxPooling2D(pool_size=(pool_size,
pool_size))(conv_4)
drop_2 = Dropout(drop_prob_1)(pool_2)


# Now flatten to 1D, apply Dense -> ReLU (with
dropout) -> softmax
flat = Flatten()(drop_2)
hidden = Dense(hidden_size,
activation='relu')(flat)
drop_3 = Dropout(drop_prob_2)(hidden)
out = Dense(num_classes,
activation='softmax')(drop_3)

model = Model(input=inp, output=out) # To define a
model, just specify its input and output layers
```

```
model.compile(loss='categorical_crossentropy', #
using the cross-entropy loss function
              optimizer='adam', # using the Adam
optimiser
              metrics=['accuracy']) # reporting
the accuracy

H = model.fit(X_train, Y_train, # Train the model
using the training set...
          batch_size=batch_size,
nb_epoch=num_epochs,
          verbose=1, validation_split=0.1) #
...holding out 10% of the data for validation

model.evaluate(X_test, Y_test, verbose=1) #
Evaluate the trained model on the test set!


plt.plot(H.history['accuracy'])
plt.plot(H.history['val_accuracy'])
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

plt.plot(H.history['loss'])
plt.plot(H.history['val_loss'])
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```