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

ОТЧЕТ

по индивидуальному заданию
по дисциплине «Искусственные нейронные сети»
Тема: Классификация животных

Студент гр. 7383 _____ Лосев М.Л.

Студент гр. 7383 _____ Кирсанов А.Я.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург
2020

Цель.

Классифицировать животных по их описанию. Добиться максимально возможной точности.

Задачи.

- Реализовать эффективную модель
- Обучить её наиболее эффективным способом

Описание датасета.

Датасет булевых значений. Включает в себя информацию о 101 животном из зоопарка, которые относятся к 7 классам.

Свойства:

1. animal name: Unique for each instance
2. hair: Boolean
3. feathers: Boolean
4. eggs: Boolean
5. milk: Boolean
6. airborne: Boolean
7. aquatic: Boolean
8. predator: Boolean
9. toothed: Boolean
10. backbone: Boolean
11. breathes: Boolean
12. venomous: Boolean
13. fins: Boolean
14. legs: Numeric (set of values: {0,2,4,5,6,8})
15. tail: Boolean
16. domestic: Boolean
17. catsize: Boolean
18. type: Numeric (integer values in range [1,7])

Задача заключается в классификации животного по его описанию.

Выполнение работы.

Рассмотрим пять архитектур:

```
# basic model #1
model1 = Sequential()
model1.add(Dense(16, activation='relu', input_shape=(16, )))
model1.add(Dense(7, activation='softmax'))

# basic with dropout layer #2
model2 = Sequential()
model2.add(Dense(16, activation='relu', input_shape=(16, )))
model2.add(Dropout(0.25))
model2.add(Dense(7, activation='softmax'))

# basic with extra dense layer #3
model3 = Sequential()
model3.add(Dense(16, activation='relu', input_shape=(16, )))
model3.add(Dense(32, activation='relu'))
model3.add(Dense(7, activation='softmax'))

# basic + dropout + extra dense layer #4
model4 = Sequential()
model4.add(Dense(16, activation='relu', input_shape=(16, )))
model4.add(Dropout(0.25))
model4.add(Dense(32, activation='relu'))
model4.add(Dropout(0.25))
model4.add(Dense(7, activation='softmax'))

# basic + dropouts + 2 extra dense layers #5
model5 = Sequential()
```

```
model5.add(Dense(16, activation='relu', input_shape=(16, )))
model5.add(Dense(32, activation='relu'))
model5.add(Dropout(0.25))
model5.add(Dense(32, activation='relu'))
model5.add(Dropout(0.25))
model5.add(Dense(7, activation='softmax'))
```

Так как датасет небольшой, можно пробовать много архитектур, так как модели быстро обучаются.

Для каждой модели приведены графики обучения

```
# basic model #0
```

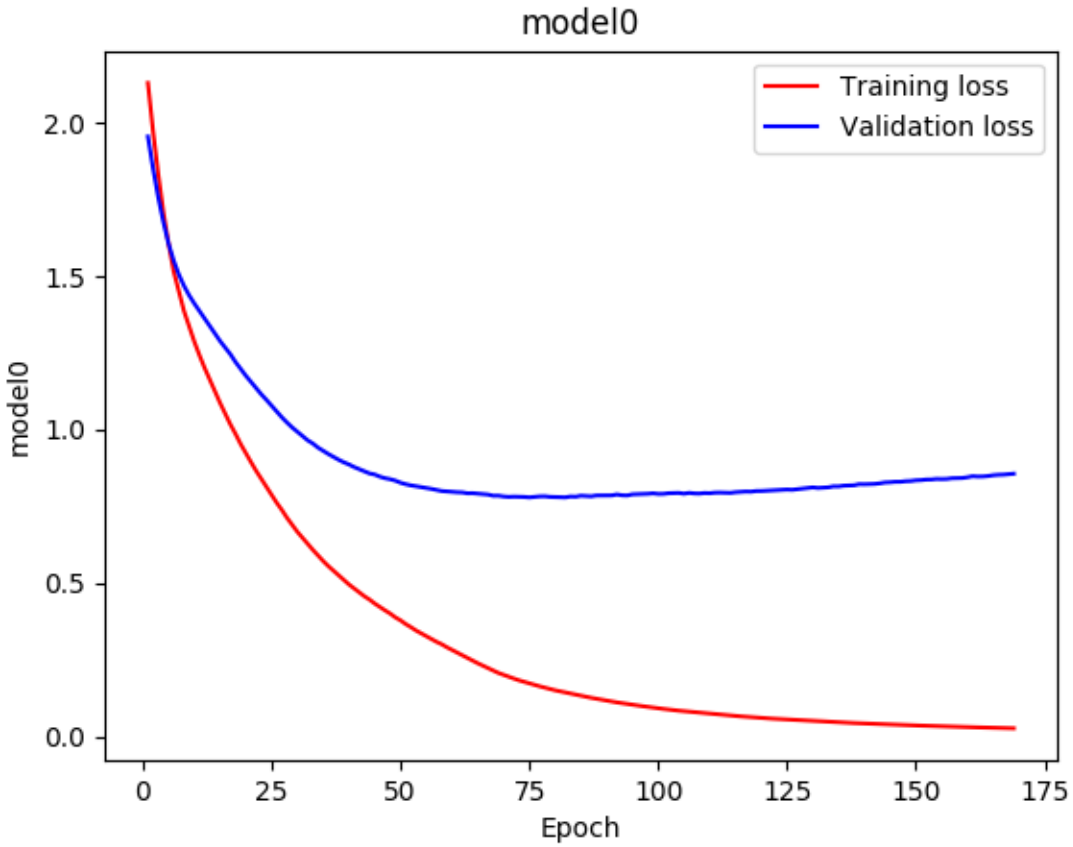

Рисунок 1. Ошибка первой модели

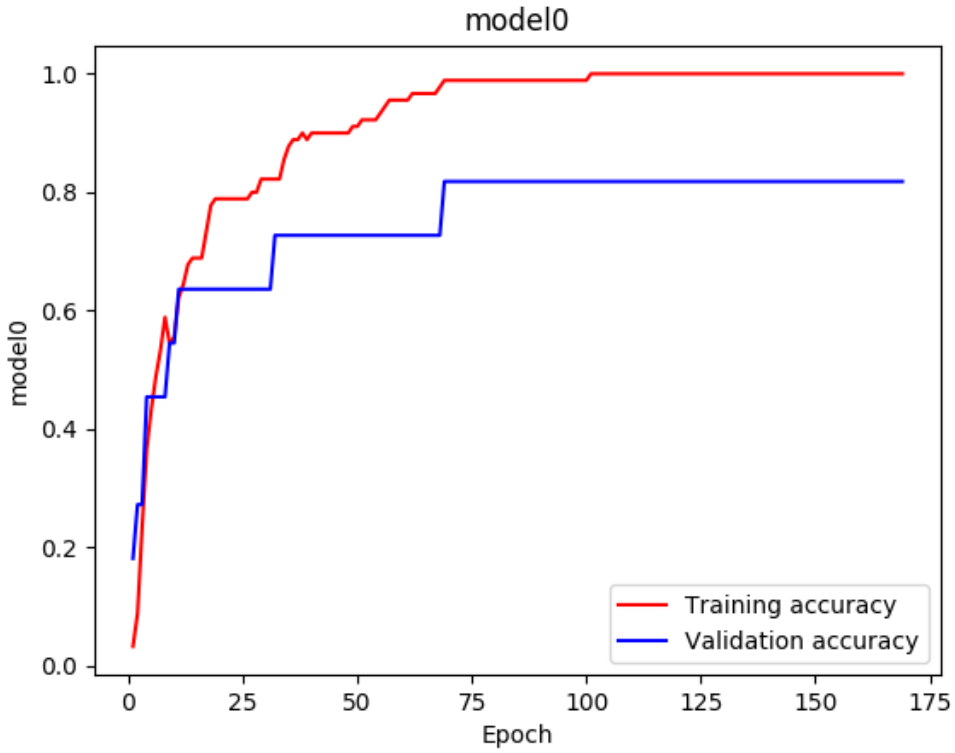

Рисунок 2. Точность первой модели

basic with dropout layer #1

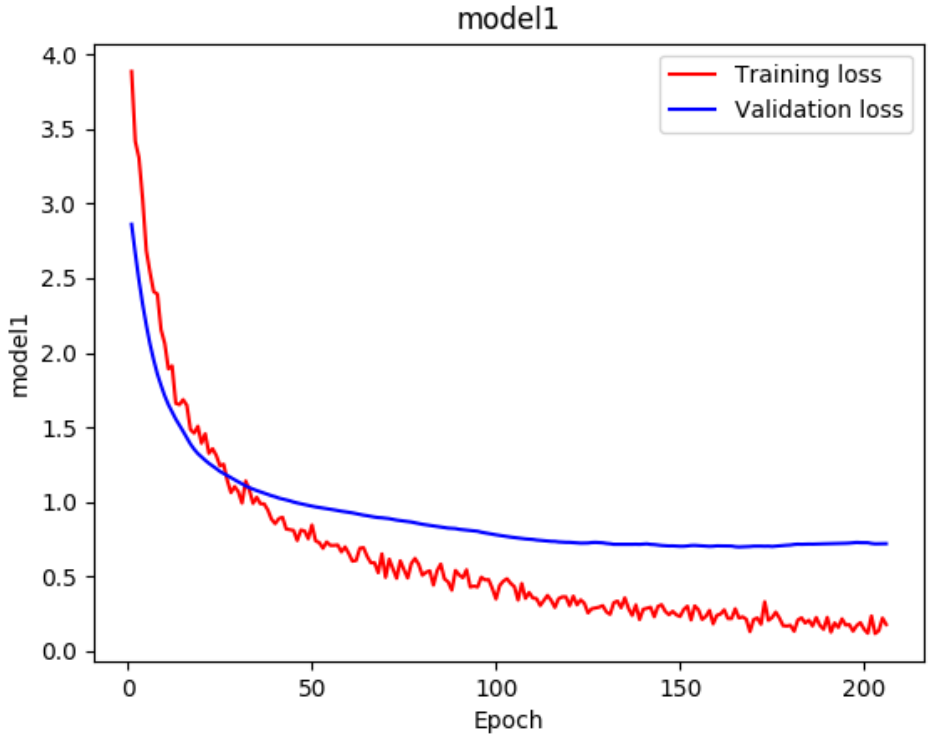

Рисунок 3. Ошибка для второй модели

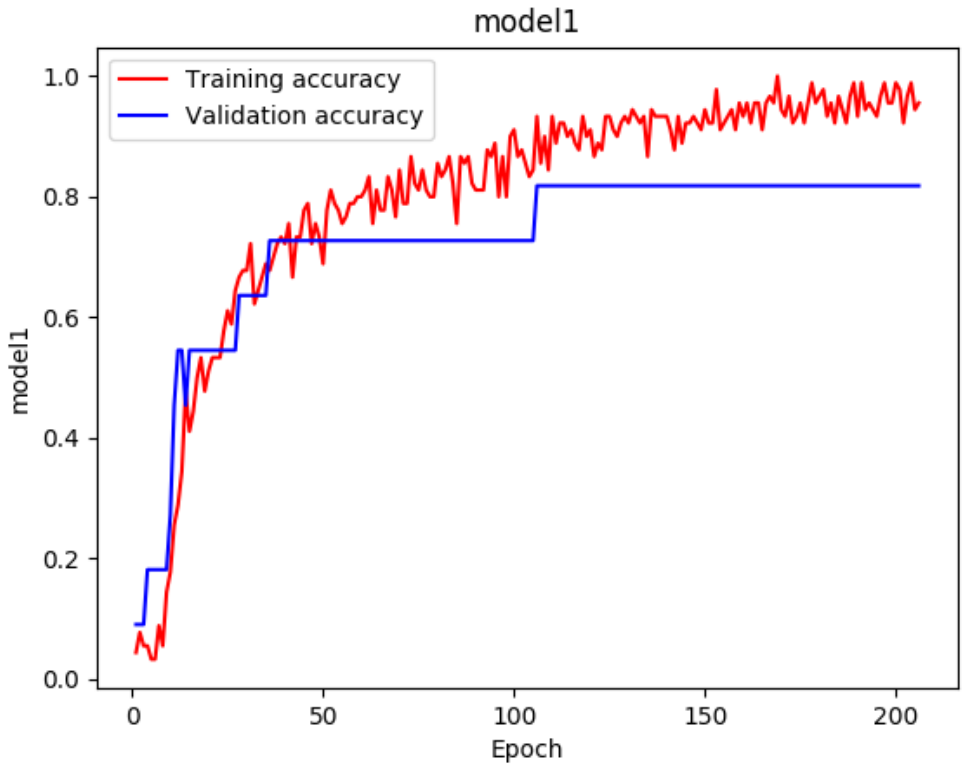

Рисунок 4. Точность второй модели

basic with extra dense layer #2

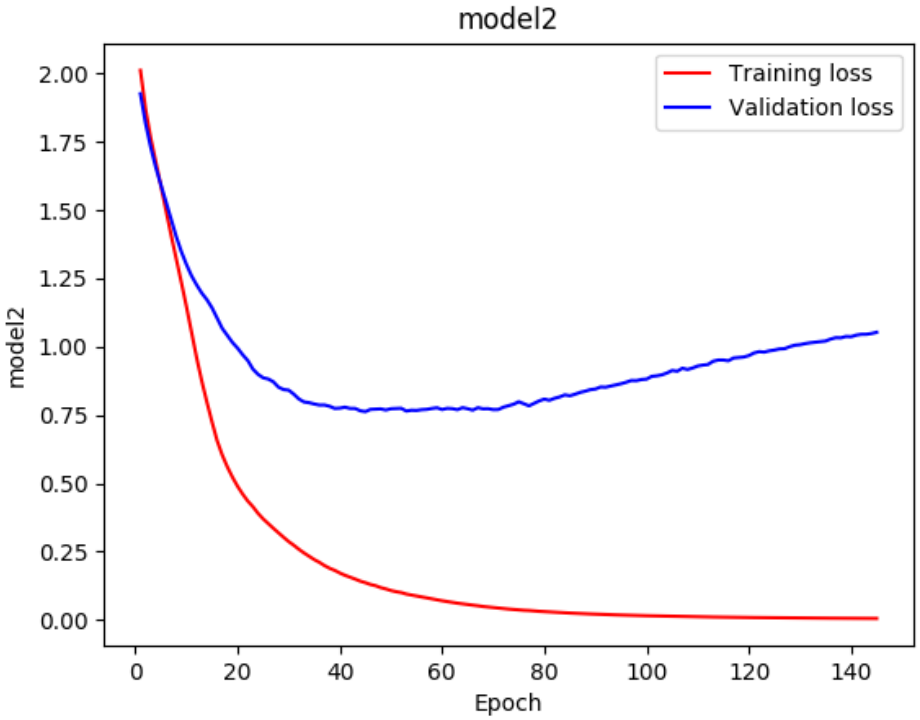

Рисунок 5. Ошибка для третьей модели

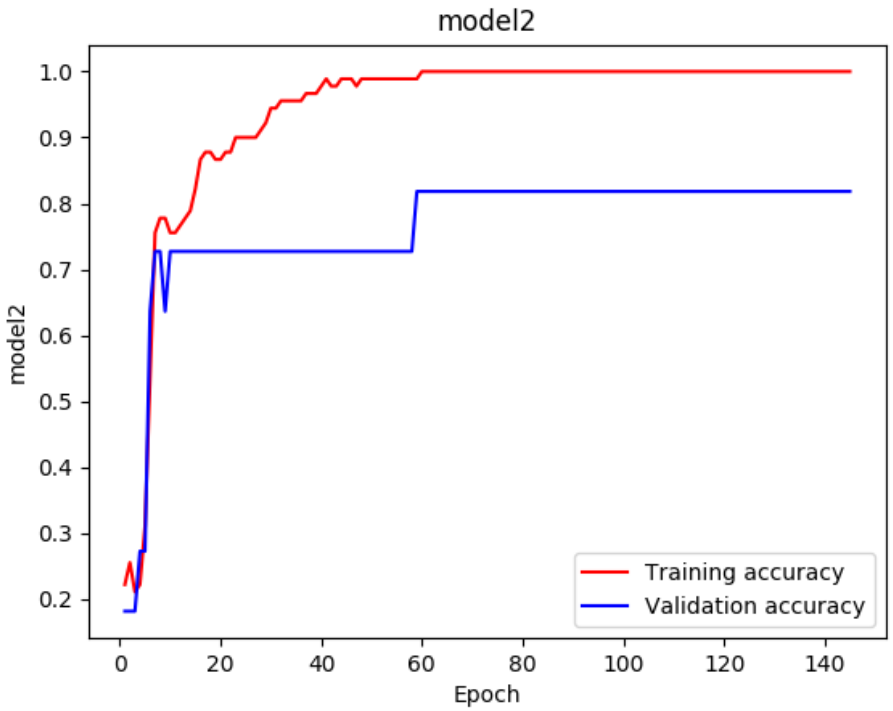

Рисунок 6. Точность третьей модели

basic + dropout + extra dense layer #3

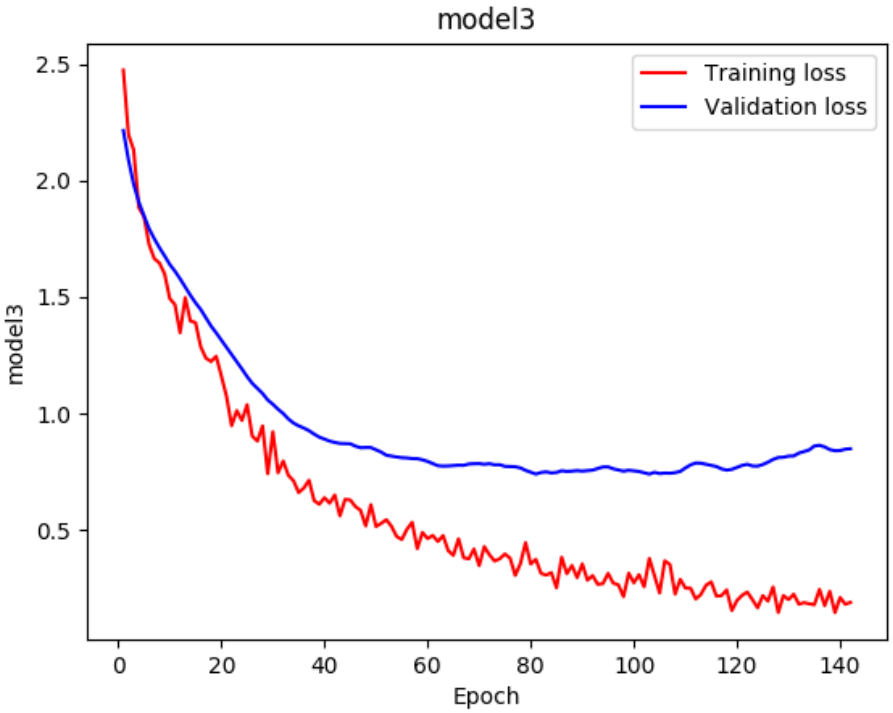

Рисунок 7. Ошибка четвертой модели

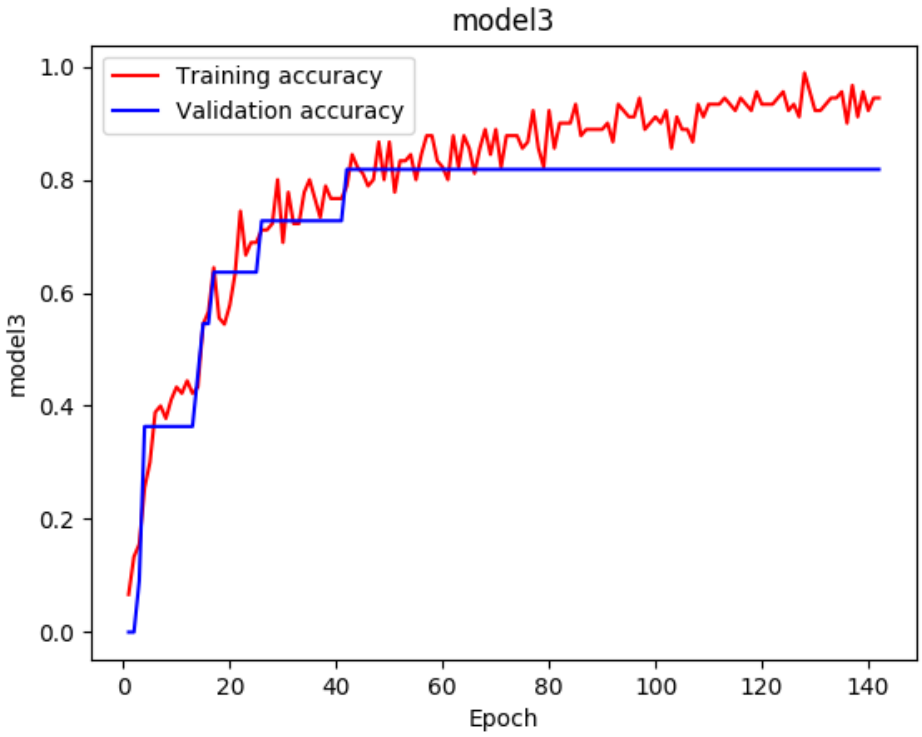

Рисунок 8. Точность четвертой модели

basic + dropouts + 2 extra dense layers #4

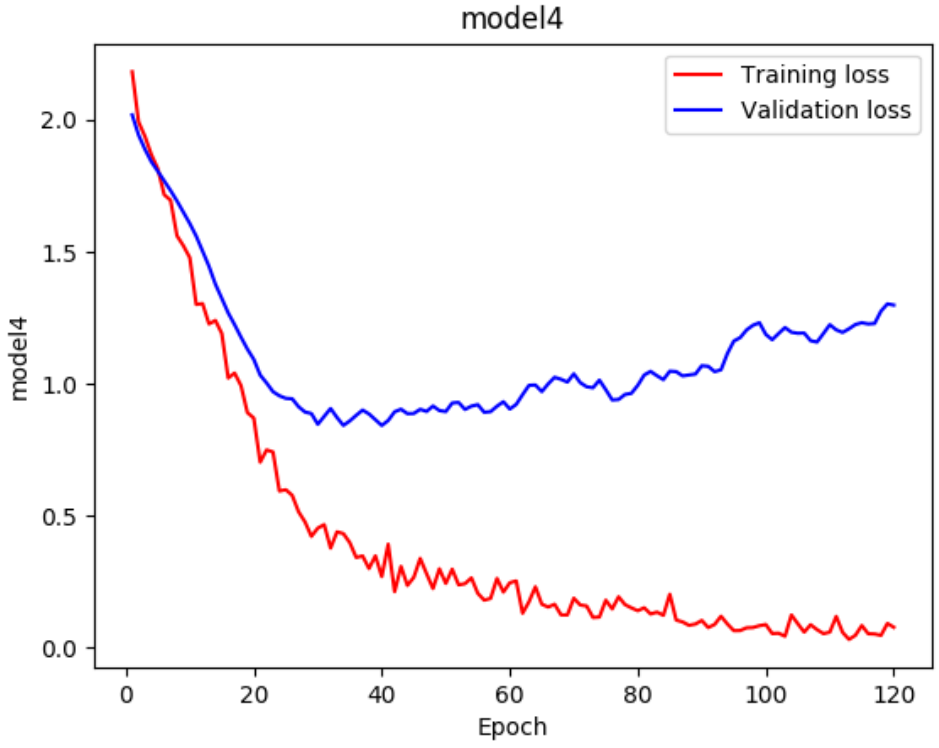

Рисунок 9. Ошибка пятой модели

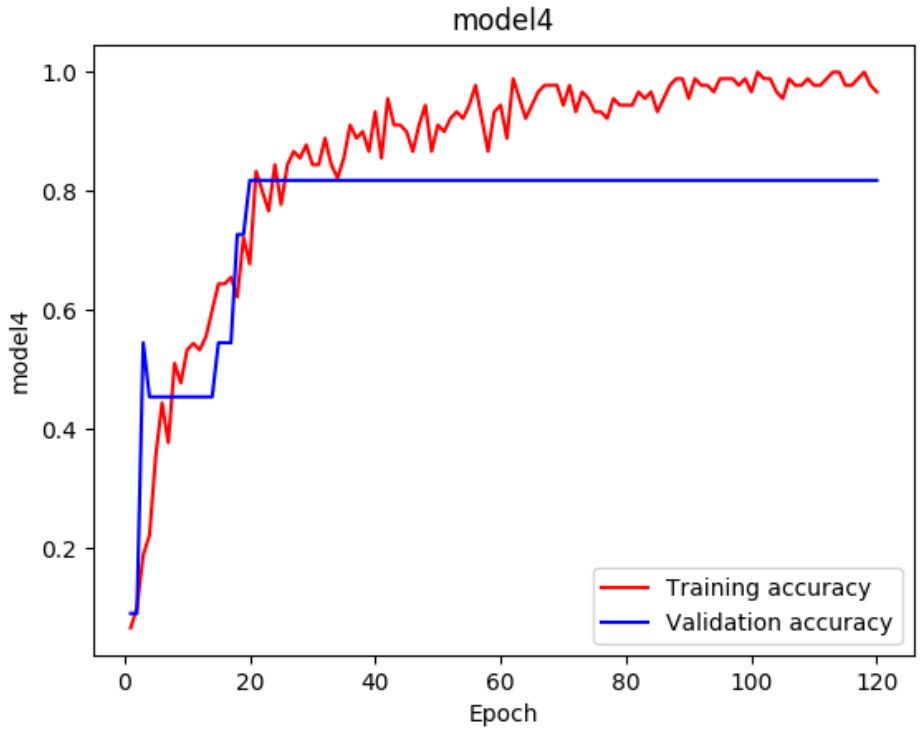

Рисунок 10. Точность пятой модели

Точность моделей на тестовых данных почти одинакова и не превышает 0.82 (выберем модель с двумя полносвязными слоями и одним слоем Dropout как лучгую). С одной стороны, это довольно низкая точность. Но с другой стороны, датасет очень маленький, поэтому трудно добиться лучшего результата.

Можно было бы попробовать разные функции активации, но известно, что функциями relu можно аппроксимировать любую другую функцию (вопрос лишь в количестве слоев, необходимом для этого). Но увеличение количества слоев по сравнению с базовой моделью не дало прироста точности, а значит дело, скорее всего, не в функции активации.

Попробуем использовать разные оптимизаторы. Может быть, это позволит улучшить точность. Ничего страшного в переборе в этом случае, опять же, нет, потому что датасет маленький, а обучение очень быстрое.

Были рассмотрены результаты, полученные с использованием оптимизаторов Adam, Adadelta, RMSprop и SGD и разных параметров обучения. Ниже приведены наилучшие результаты для каждого оптимизатора (при самых

удачных параметрах обучения).

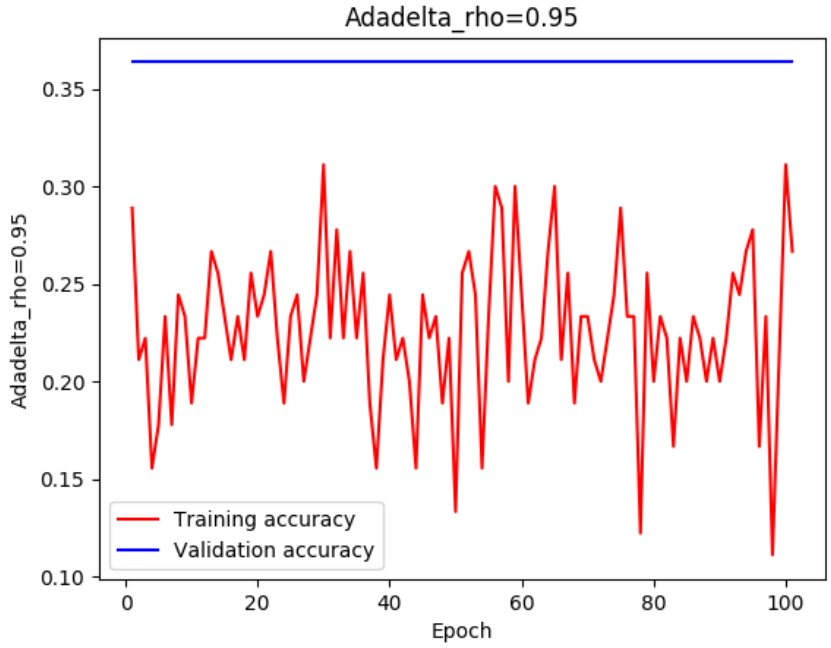

Рисунок 11. Точность для Adadelta

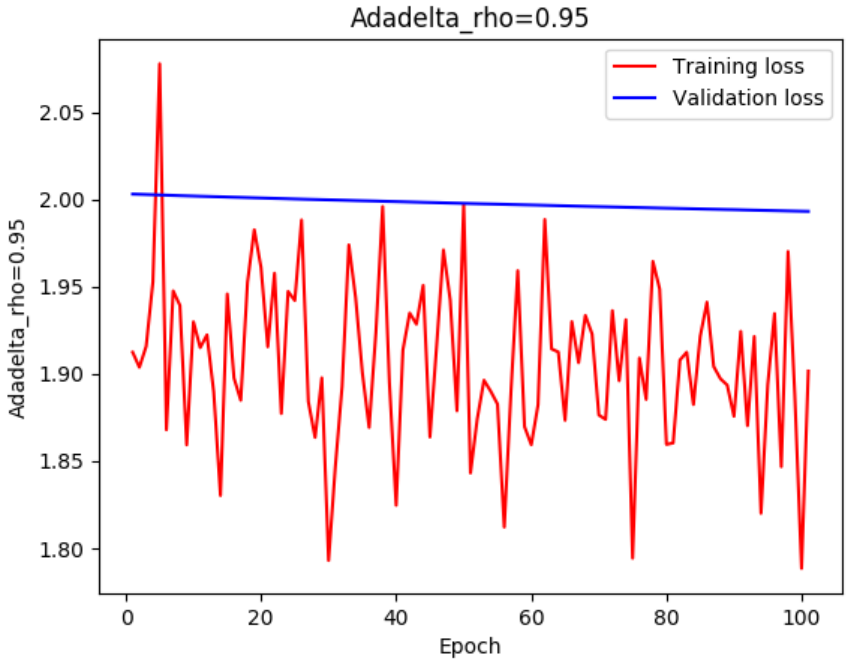

Рисунок 12. Ошибка для Adadelta

Рисунок 13. Точность для Adam

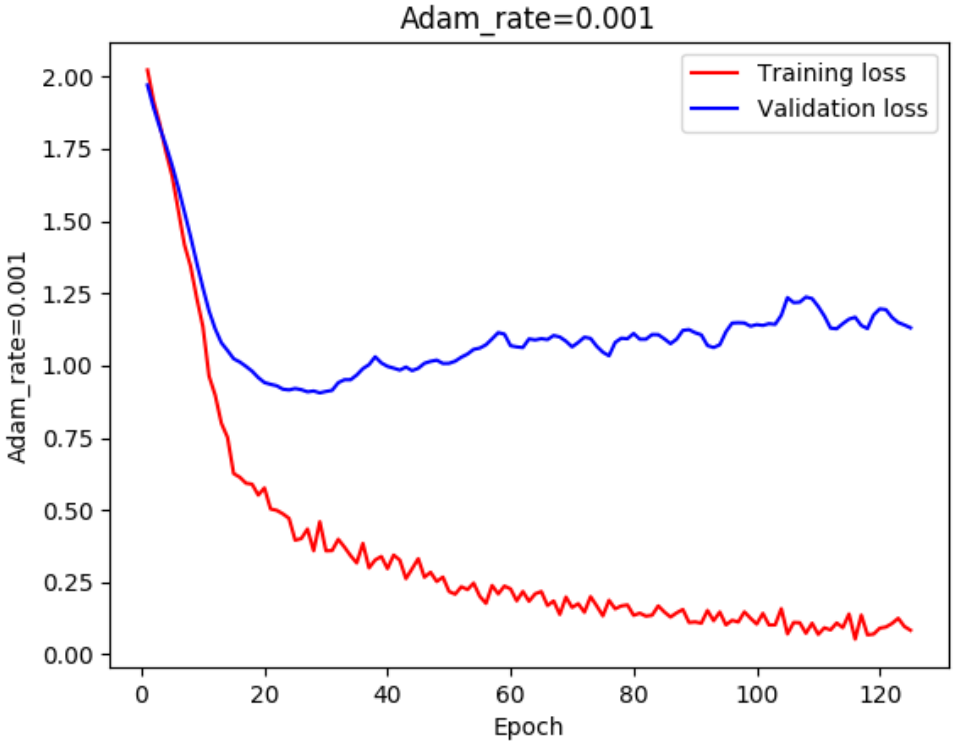

Рисунок 14. Обшибка для Adam

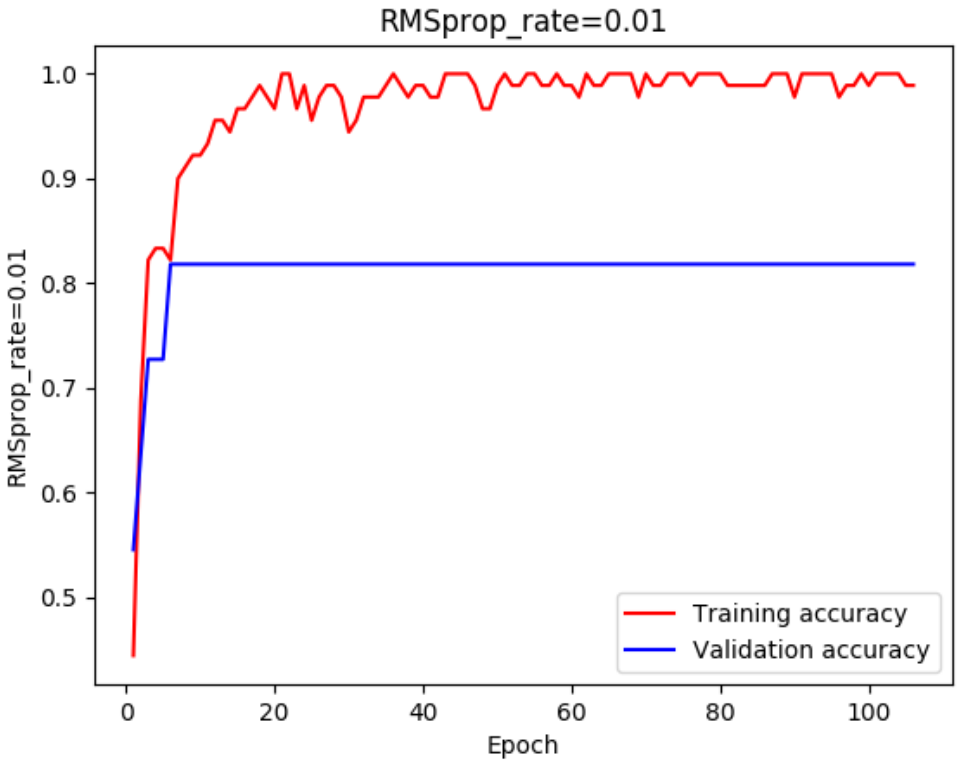

Рисунок 15. Точность для RMSprop

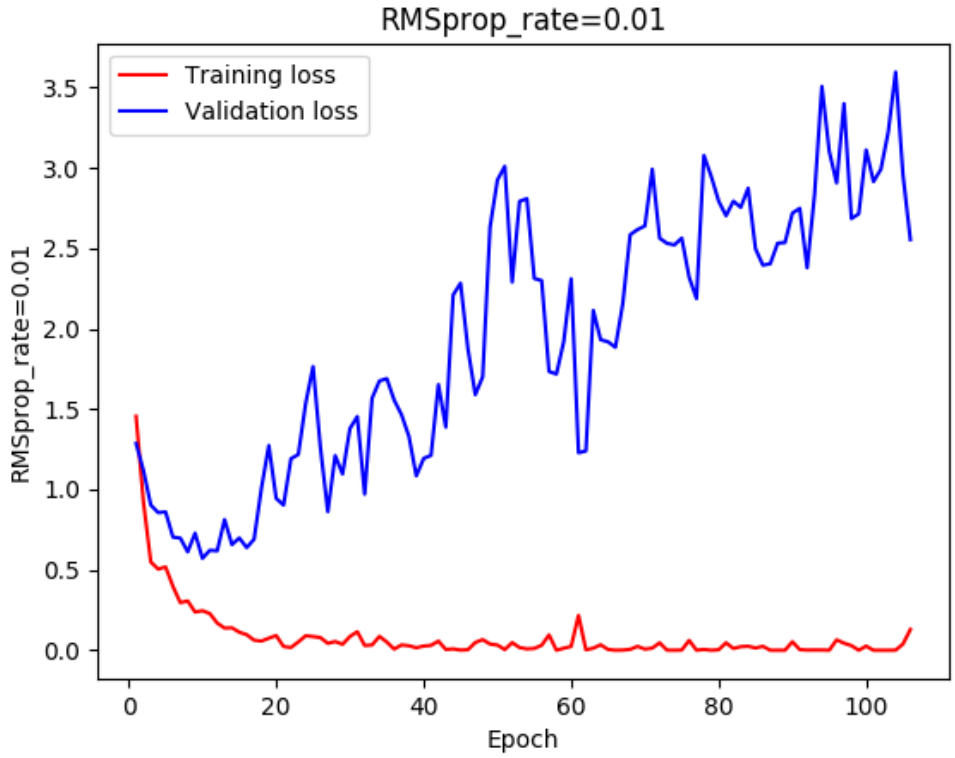

Рисунок 16. Ошбика для RMSprop

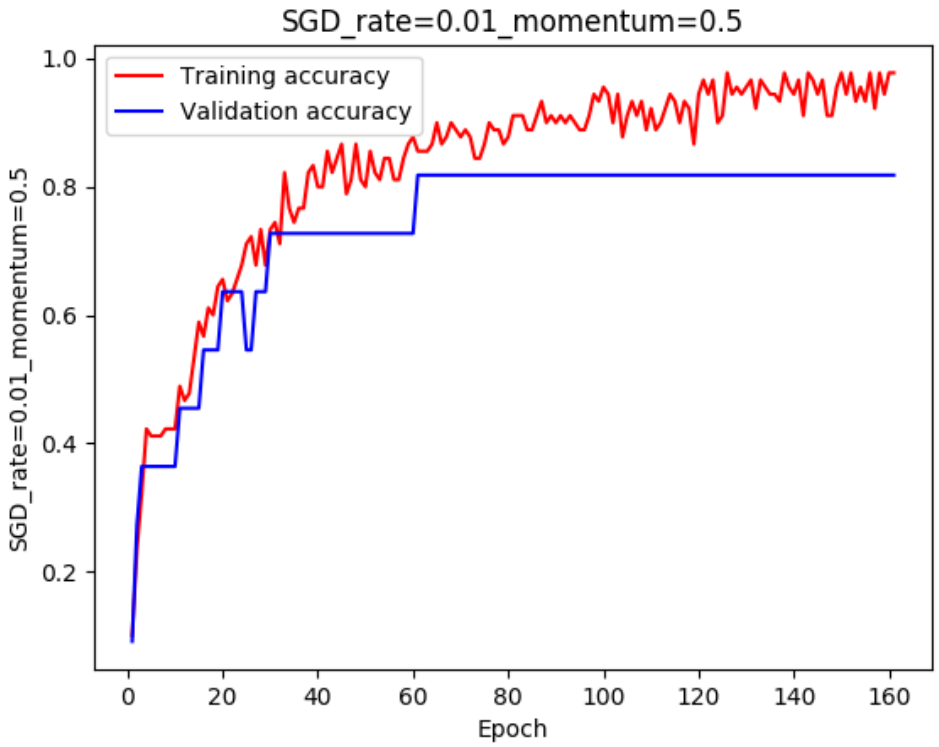

Рисунок 17. Точность для SGD

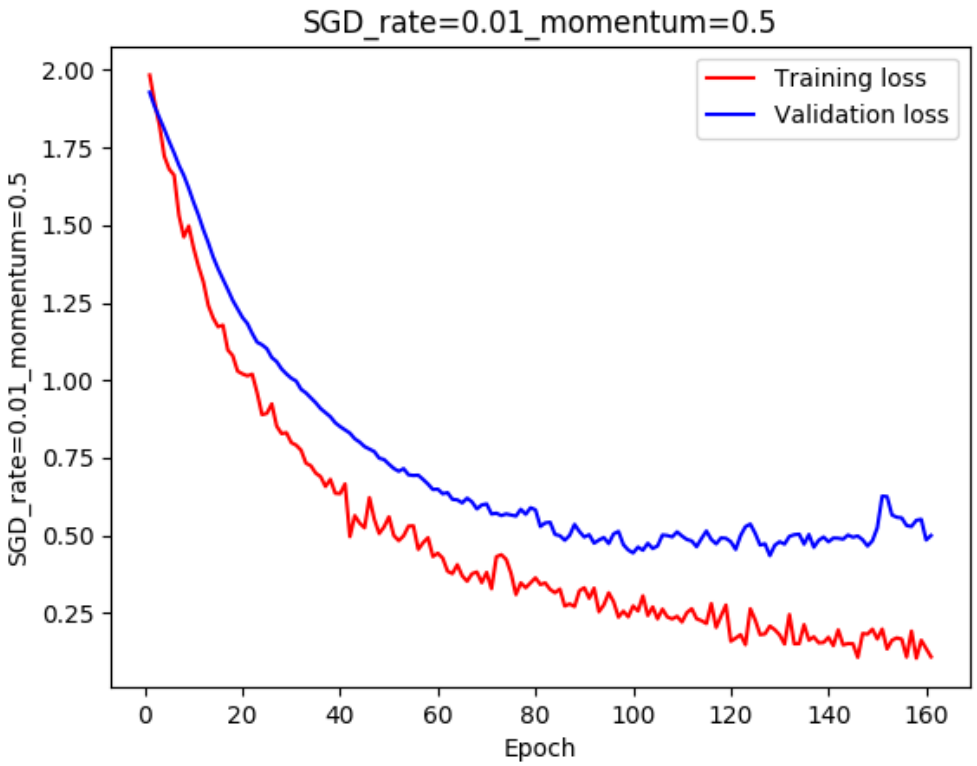

Рисунок 18. Ошибка для SGD

Лучше всего подошел SGD, потому что точность, которой с ним можно

достигнуть, не меньше, чем для других оптимизаторов, а ошибка не растет в процессе обучения только для него.

Вывод.

Были рассмотрены некоторые архитектуры моделей и выбрана наилучшая. Результаты, достигнутые с использованием моделей с разными архитектурами, оказались почти одинаковы, но лучше всего подошла самая простая модель, состоящая только из входного и выходного слоев. Были рассмотрены некоторые оптимизаторы при различных параметрах обучения и выбран лучший. Им оказался SGD. Но несмотря на эти меры достигнутая точность не превысила 0.82 для тестовых данных, хотя для обучающих данных и удавалось достичь точности 1. Это можно объяснить размером датасета: в нем описано всего 101 животное. Этого мало, чтобы хорошо обучить модель.

Приложение А

```
import tensorflow as tf
import pandas
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Activation, Flatten, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import EarlyStopping
from PIL import Image
import numpy as np

# builds 4 models with different architectures
def get_models():
    # basic model
    model1 = Sequential()
    model1.add(Dense(16, activation='relu', input_shape=(16, )))
    model1.add(Dense(7, activation='softmax'))

    # basic with dropout layer
    model2 = Sequential()
    model2.add(Dense(16, activation='relu', input_shape=(16, )))
    model2.add(Dropout(0.25))
    model2.add(Dense(7, activation='softmax'))

    # basic with extra  dense layer
    model3 = Sequential()
    model3.add(Dense(16, activation='relu', input_shape=(16, )))
    model3.add(Dense(32, activation='relu'))
    model3.add(Dense(7, activation='softmax'))

    # basic + dropout + extra dense layer
    model4 = Sequential()
    model4.add(Dense(16, activation='relu', input_shape=(16, )))
    model4.add(Dropout(0.25))
    model4.add(Dense(32, activation='relu'))
    model4.add(Dropout(0.25))
    model4.add(Dense(7, activation='softmax'))

     # basic + dropouts + 2 extra dense layers
    model5 = Sequential()
    model5.add(Dense(16, activation='relu', input_shape=(16, )))
    model5.add(Dense(32, activation='relu'))
    model5.add(Dropout(0.25))
    model5.add(Dense(32, activation='relu'))
    model5.add(Dropout(0.25))
    model5.add(Dense(7, activation='softmax'))

    return [model1, model2, model3, model4, model5]

def build_model():
    model = Sequential()
    model.add(Dense(16, activation='relu', input_shape=(16, )))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(7, activation='softmax'))
    return model
```

```
def load_data(filename):
    dataframe = pandas.read_csv(filename, header=None)
    dataset = dataframe.values
    X = dataset[:,1:17].astype(float)   # animal data
    Y = dataset[:,17].astype(float)     # animal types
    return np.array(X), np.array(Y)

def split_data(X, Y, val_split):
    assert(len(X) == len(Y))
    border = int(np.floor(len(X) * (1 - val_split)))
    training_data = X[0:border]
    testing_data = X[border:]
    training_targets = Y[0:border]
    testing_targets = Y[border:]
    return (training_data, training_targets), (testing_data, testing_targets)

def save_plot(label, history):
    picdir = './pics/'
    plt.clf()
    arg = range(1, len(history['loss']) + 1)

    # save accuracy
    plt.clf()
    plt.plot(arg, history['acc'], 'r', label='Training accuracy')
    plt.plot(arg, history['val_acc'], 'b', label='Validation accuracy')
    plt.title(label)
    plt.xlabel('Epoch')
    plt.ylabel(label)
    plt.legend()
    plt.savefig(picdir + label + '_accuracy' + '.png')

    # save loss
    plt.clf()
    plt.plot(arg, history['loss'], 'r', label='Training loss')
    plt.plot(arg, history['val_loss'], 'b', label='Validation loss')
    plt.title(label)
    plt.xlabel('Epoch')
    plt.ylabel(label)
    plt.legend()
    plt.savefig(picdir + label + '_loss' + '.png')

def explore_optimizer(optimizer, label):
    model = build_model()
    model.compile(optimizer=optimizer,                  loss='categorical_crossentropy',
metrics=['accuracy'])
    H = model.fit(training_data, training_targets, epochs=750, batch_size=10,
            validation_data=(testing_data, testing_targets),
            callbacks=[EarlyStopping(monitor='val_loss', mode='min', patience = 100),
                       EarlyStopping(monitor='val_acc', mode='max', patience = 100)])
    save_plot(label, H.history)

def prepare_targets(targets):
    targets = [t - 1 for t in targets] # cuz labels are ints of range [1:7]
    return to_categorical(targets)

def try_some_optimizers():
    rate = [0.1, 0.01, 0.001]
    momentum = [0, 0.1, 0.5, 0.95]
    rho = [0, 0.9, 0.95, 0.99, 10]
    for r in rate:
        for m in momentum:
```

```
            explore_optimizer(optimizers.SGD(learning_rate=r,              momentum=m),
'SGD_rate='+str(r)+'_momentum='+str(m))
        explore_optimizer(optimizers.Adam(learning_rate=r), 'Adam_rate='+str(r))
        # It is recommended to leave the parameters of this optimizer at their default
values
        # (except the learning rate, which can be freely tuned).
        explore_optimizer(optimizers.RMSprop(learning_rate=r), 'RMSprop_rate='+str(r))
    for r in rho:
        # Initial learning rate is recommended to leave at the default value.
        explore_optimizer(optimizers.Adadelta(rho=r), 'Adadelta_rho='+str(r))

def try_some_models(models):
    for i, model in enumerate(models):
        model.compile(optimizer='adam',                loss='categorical_crossentropy',
metrics=['accuracy'])
        H = model.fit(training_data, training_targets, epochs=750, batch_size=10,
            validation_data=(testing_data, testing_targets),
            callbacks=[EarlyStopping(monitor='val_loss', mode='min', patience = 100),
                       EarlyStopping(monitor='val_acc', mode='max', patience = 100)])
        save_plot('model'+str(i), H.history)


# load and prepare the data
X, Y = load_data("zoo.data")
Y = prepare_targets(Y)
(training_data, training_targets), (testing_data, testing_targets) = split_data(X, Y,
0.1)

# try some models in order to choose the best one
models = get_models()
try_some_models(models)

# get the model chosen and try some optimizers in order to choose the one that fits best
model = build_model()
try_some_optimizers()
```