lab2_bashtovyi

December 1, 2021

[59]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from skimage.io import imread_collection, imshow
from sklearn.model_selection import train_test_split


from keras.models import Sequential
from keras.layers import InputLayer, Conv2D, MaxPooling2D, Flatten, Dropout,␣
↪Dense
from keras.optimizers import adam
```

Lets import our data.

Since image shape vary, we will use padd_image function to transform images into the shape (300, 300, 3).

We will exclude grayscale images from the dataset.

[68]:
```python
i = -1
def padd_image(img):
        global i
        i += 1
        old_image_height, old_image_width, channels = img.shape

        # create new image of desired size and color (blue) for padding
        new_image_width = 300
        new_image_height = 300
        color = (0,0,0)
        result = np.full((new_image_height,new_image_width, channels), color,␣
↪dtype=np.uint8)

        # compute center offset
        x_center = (new_image_width - old_image_width) // 2
        y_center = (new_image_height - old_image_height) // 2
```

```
        # copy img image into center of result image
        result[y_center:y_center+old_image_height,
               x_center:x_center+old_image_width] = img

        return result

n_imgs = 545
n_labels = 9
input_shape = (300, 300, 3)
X = np.zeros((n_imgs, 300, 300, 3), dtype=np.int32)
Y = np.zeros(n_imgs)

# the data, split between train and test sets
beaver = imread_collection('/Users/andrei/Desktop/KNUProjects/Masters/Neironki/
 ↪master_labs/101_ObjectCategories/beaver/*.jpg')
print('beaver')
for images in beaver:
    if len(images.shape) == 3:
        res = padd_image(images)
        X[i] = res
        Y[i] = 0
        #Y[i] = [1, 0, 0, 0, 0, 0, 0, 0, 0]

cellphone = imread_collection('/Users/andrei/Desktop/KNUProjects/Masters/
 ↪Neironki/master_labs/101_ObjectCategories/cellphone/*.jpg')
print('cellphone')
for images in cellphone:
    if len(images.shape) == 3:
        res = padd_image(images)
        X[i] = res
        Y[i] = 1
        #Y[i] = [0, 1, 0, 0, 0, 0, 0, 0, 0]

dollar_bill = imread_collection('/Users/andrei/Desktop/KNUProjects/Masters/
 ↪Neironki/master_labs/101_ObjectCategories/dollar_bill/*.jpg')
print('dollar_bill')
for images in dollar_bill:
    if len(images.shape) == 3:
        res = padd_image(images)
        X[i] = res
        Y[i] = 2
        #Y[i] = [0, 1, 0, 0, 0, 0, 0, 0, 0]

garfield = imread_collection('/Users/andrei/Desktop/KNUProjects/Masters/
 ↪Neironki/master_labs/101_ObjectCategories/garfield/*.jpg')
print('garfield')
for images in garfield:
```

```
    if len(images.shape) == 3:
        res = padd_image(images)
        res = padd_image(images)
        X[i] = res
        Y[i] = 3
        #Y[i] = [0, 1, 0, 0, 0, 0, 0, 0, 0]

kangaroo = imread_collection('/Users/andrei/Desktop/KNUProjects/Masters/
 ↪Neironki/master_labs/101_ObjectCategories/kangaroo/*.jpg')
print('kangaroo')
for images in kangaroo:
    if len(images.shape) == 3:
        res = padd_image(images)
        X[i] = res
        Y[i] = 4
        #Y[i] = [0, 1, 0, 0, 0, 0, 0, 0, 0]

minaret = imread_collection('/Users/andrei/Desktop/KNUProjects/Masters/Neironki/
 ↪master_labs/101_ObjectCategories/minaret/*.jpg')
print('minaret')
for images in minaret:
    if len(images.shape) == 3:
        res = padd_image(images)
        X[i] = res
        Y[i] = 5
        #Y[i] = [0, 1, 0, 0, 0, 0, 0, 0, 0]

rhino = imread_collection('/Users/andrei/Desktop/KNUProjects/Masters/Neironki/
 ↪master_labs/101_ObjectCategories/rhino/*.jpg')
print('rhino')
for images in rhino:
    if len(images.shape) == 3:
        res = padd_image(images)
        X[i] = res
        Y[i] = 6
        #Y[i] = [0, 1, 0, 0, 0, 0, 0, 0, 0]

stegosaurus = imread_collection('/Users/andrei/Desktop/KNUProjects/Masters/
 ↪Neironki/master_labs/101_ObjectCategories/stegosaurus/*.jpg')
print('stegosaurus')
for images in stegosaurus:
    if len(images.shape) == 3:
        res = padd_image(images)
        X[i] = res
        Y[i] = 7
        #Y[i] = [0, 1, 0, 0, 0, 0, 0, 0, 0]
```

```
windsor_chair = imread_collection('/Users/andrei/Desktop/KNUProjects/Masters/
↪Neironki/master_labs/101_ObjectCategories/windsor_chair/*.jpg')
print('windsor_chair')
for images in windsor_chair:
    if len(images.shape) == 3:
        res = padd_image(images)
        X[i] = res
        Y[i] = 8
        #Y[i] = [0, 1, 0, 0, 0, 0, 0, 0, 0]
```

```
beaver
cellphone
dollar_bill
garfield
kangaroo
minaret
rhino
stegosaurus
windsor_chair
```

We obtained dataset X and set of coresponding labels Y.

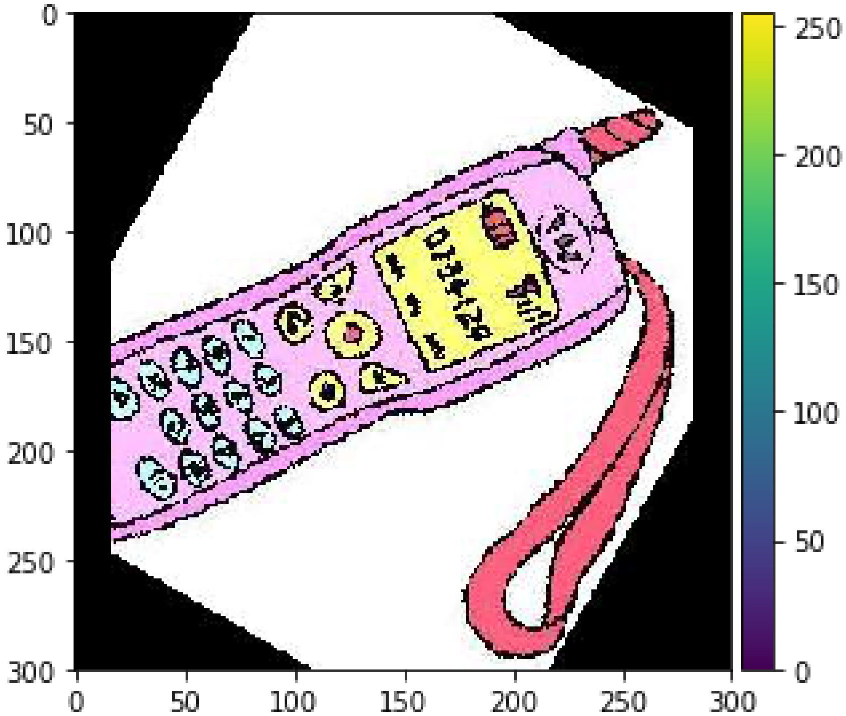
Labels corespond to the following indexes: 0 - beaver, 1 - cellphone, 2 - dollar_bill, 3 - garfield, 4 - kangaroo, 5 - minaret, 6 - rhino, 7 - stegosaurus, 8 - windsor_chair.

Lets show an example of the (X, Y) pair.

[69]:
```
imshow(X[54])
Y[54]
```

```
/Users/andrei/anaconda3/envs/nlp_course/lib/python3.7/site-
packages/skimage/io/_plugins/matplotlib_plugin.py:150: UserWarning: Low image
data range; displaying image with stretched contrast.
  lo, hi, cmap = _get_display_range(image)
```

[69]:
```
1.0
```



Lets normalize pixel values to the range (0, 1).

Lets transform numerical values of 0, 1, 2, 3 .. 8 into the arrays of label probabilities. Label 0 will be [1, 0, 0, 0 .., 0] etc.

Lets split our data to the train and test sets.

[70]:
```
X = X/255.
Y = pd.get_dummies(Y)
Y_cols = list(Y.columns)
Y = Y.values

#split between train and test sets
x_train, x_test, y_train, y_test = train_test_split(X, Y)
```

Lets define, train and test dense model.

[71]:
```
def dense(input_shape, n_labels):
    model = Sequential()
    model.add(Flatten(input_shape=(300, 300, 3)))
    model.add(Dense(32, kernel_initializer="normal", activation="relu"))
    model.add(Dense(64, kernel_initializer="normal", activation="relu"))
    model.add(Dense(128, kernel_initializer="normal", activation="relu"))
    model.add(Dense(64, kernel_initializer="normal", activation="relu"))
```

```
    model.add(Dense(n_labels, kernel_initializer="normal",␣
 ↪activation="softmax"))
    model.compile(optimizer=adam(lr = 0.001, decay=1e-6),
        loss="categorical_crossentropy", metrics=["accuracy"])
    model.summary()
    return model
```

[72]:
```
model_d = dense(input_shape, n_labels)
batch_size = 64
epochs = 50

model_d.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,␣
 ↪validation_split=0.2)
```

```
Model: "sequential_23"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_20 (Flatten)         (None, 270000)            0
_________________________________________________________________
dense_83 (Dense)             (None, 32)                8640032
_________________________________________________________________
dense_84 (Dense)             (None, 64)                2112
_________________________________________________________________
dense_85 (Dense)             (None, 128)               8320
_________________________________________________________________
dense_86 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_87 (Dense)             (None, 9)                 585
=================================================================
Total params: 8,659,305
Trainable params: 8,659,305
Non-trainable params: 0
_________________________________________________________________
Train on 326 samples, validate on 82 samples
Epoch 1/50
326/326 [==============================] - 2s 6ms/step - loss: 2.1897 -
accuracy: 0.2270 - val_loss: 2.0524 - val_accuracy: 0.3049
Epoch 2/50
326/326 [==============================] - 1s 3ms/step - loss: 1.8410 -
accuracy: 0.4049 - val_loss: 1.6473 - val_accuracy: 0.4634
Epoch 3/50
326/326 [==============================] - 1s 3ms/step - loss: 1.5506 -
accuracy: 0.4540 - val_loss: 1.4134 - val_accuracy: 0.6098
Epoch 4/50
326/326 [==============================] - 1s 3ms/step - loss: 1.4449 -
accuracy: 0.4632 - val_loss: 1.3767 - val_accuracy: 0.5854
```

```
Epoch 5/50
326/326 [==============================] - 1s 3ms/step - loss: 1.2828 -
accuracy: 0.5276 - val_loss: 1.3157 - val_accuracy: 0.6585
Epoch 6/50
326/326 [==============================] - 1s 3ms/step - loss: 1.1975 -
accuracy: 0.6012 - val_loss: 1.3297 - val_accuracy: 0.6585
Epoch 7/50
326/326 [==============================] - 1s 3ms/step - loss: 1.0967 -
accuracy: 0.6074 - val_loss: 1.1114 - val_accuracy: 0.6463
Epoch 8/50
326/326 [==============================] - 1s 3ms/step - loss: 0.9940 -
accuracy: 0.6564 - val_loss: 1.2329 - val_accuracy: 0.6829
Epoch 9/50
326/326 [==============================] - 1s 3ms/step - loss: 0.9516 -
accuracy: 0.7117 - val_loss: 1.1941 - val_accuracy: 0.6463
Epoch 10/50
326/326 [==============================] - 1s 3ms/step - loss: 0.9633 -
accuracy: 0.6963 - val_loss: 1.1041 - val_accuracy: 0.7073
Epoch 11/50
326/326 [==============================] - 1s 3ms/step - loss: 0.9203 -
accuracy: 0.7025 - val_loss: 1.1128 - val_accuracy: 0.6829
Epoch 12/50
326/326 [==============================] - 1s 3ms/step - loss: 0.8988 -
accuracy: 0.6994 - val_loss: 1.2942 - val_accuracy: 0.6220
Epoch 13/50
326/326 [==============================] - 1s 3ms/step - loss: 0.8035 -
accuracy: 0.7147 - val_loss: 1.0586 - val_accuracy: 0.6829
Epoch 14/50
326/326 [==============================] - 1s 3ms/step - loss: 0.7573 -
accuracy: 0.7730 - val_loss: 1.1549 - val_accuracy: 0.6707
Epoch 15/50
326/326 [==============================] - 1s 3ms/step - loss: 0.8358 -
accuracy: 0.7178 - val_loss: 1.1245 - val_accuracy: 0.6951
Epoch 16/50
326/326 [==============================] - 1s 3ms/step - loss: 0.7214 -
accuracy: 0.7638 - val_loss: 1.1004 - val_accuracy: 0.7195
Epoch 17/50
326/326 [==============================] - 1s 3ms/step - loss: 0.6539 -
accuracy: 0.7883 - val_loss: 1.1730 - val_accuracy: 0.7073
Epoch 18/50
326/326 [==============================] - 1s 3ms/step - loss: 0.6251 -
accuracy: 0.8221 - val_loss: 1.0358 - val_accuracy: 0.6829
Epoch 19/50
326/326 [==============================] - 1s 3ms/step - loss: 0.5514 -
accuracy: 0.8650 - val_loss: 1.0932 - val_accuracy: 0.7195
Epoch 20/50
326/326 [==============================] - 1s 3ms/step - loss: 0.5231 -
accuracy: 0.8528 - val_loss: 1.1581 - val_accuracy: 0.6951
```

```
Epoch 21/50
326/326 [==============================] - 1s 3ms/step - loss: 0.4887 -
accuracy: 0.8558 - val_loss: 1.1925 - val_accuracy: 0.6951
Epoch 22/50
326/326 [==============================] - 1s 3ms/step - loss: 0.7269 -
accuracy: 0.7945 - val_loss: 1.4542 - val_accuracy: 0.5488
Epoch 23/50
326/326 [==============================] - 1s 3ms/step - loss: 0.6019 -
accuracy: 0.8558 - val_loss: 1.3978 - val_accuracy: 0.6463
Epoch 24/50
326/326 [==============================] - 1s 3ms/step - loss: 0.4490 -
accuracy: 0.8834 - val_loss: 0.9982 - val_accuracy: 0.7195
Epoch 25/50
326/326 [==============================] - 1s 3ms/step - loss: 0.4028 -
accuracy: 0.9018 - val_loss: 1.4091 - val_accuracy: 0.6585
Epoch 26/50
326/326 [==============================] - 1s 3ms/step - loss: 0.4723 -
accuracy: 0.8589 - val_loss: 1.1632 - val_accuracy: 0.6829
Epoch 27/50
326/326 [==============================] - 1s 3ms/step - loss: 0.4798 -
accuracy: 0.8742 - val_loss: 1.1145 - val_accuracy: 0.7561
Epoch 28/50
326/326 [==============================] - 1s 3ms/step - loss: 0.3447 -
accuracy: 0.9049 - val_loss: 1.2566 - val_accuracy: 0.6829
Epoch 29/50
326/326 [==============================] - 1s 3ms/step - loss: 0.3107 -
accuracy: 0.9264 - val_loss: 1.0415 - val_accuracy: 0.7317
Epoch 30/50
326/326 [==============================] - 1s 3ms/step - loss: 0.2839 -
accuracy: 0.9049 - val_loss: 1.5046 - val_accuracy: 0.6463
Epoch 31/50
326/326 [==============================] - 1s 3ms/step - loss: 0.5697 -
accuracy: 0.8466 - val_loss: 1.3183 - val_accuracy: 0.7073
Epoch 32/50
326/326 [==============================] - 1s 3ms/step - loss: 0.5324 -
accuracy: 0.8313 - val_loss: 1.1631 - val_accuracy: 0.6463
Epoch 33/50
326/326 [==============================] - 1s 3ms/step - loss: 0.3904 -
accuracy: 0.8896 - val_loss: 1.3740 - val_accuracy: 0.6707
Epoch 34/50
326/326 [==============================] - 1s 3ms/step - loss: 0.3358 -
accuracy: 0.9141 - val_loss: 0.8830 - val_accuracy: 0.7439
Epoch 35/50
326/326 [==============================] - 1s 3ms/step - loss: 0.2772 -
accuracy: 0.9141 - val_loss: 1.0838 - val_accuracy: 0.7317
Epoch 36/50
326/326 [==============================] - 1s 3ms/step - loss: 0.2327 -
accuracy: 0.9356 - val_loss: 1.2318 - val_accuracy: 0.7317
```

```
Epoch 37/50
326/326 [==============================] - 1s 3ms/step - loss: 0.2537 -
accuracy: 0.9356 - val_loss: 1.0582 - val_accuracy: 0.7683
Epoch 38/50
326/326 [==============================] - 1s 3ms/step - loss: 0.2726 -
accuracy: 0.9080 - val_loss: 0.8764 - val_accuracy: 0.7317
Epoch 39/50
326/326 [==============================] - 1s 3ms/step - loss: 0.1914 -
accuracy: 0.9540 - val_loss: 1.1486 - val_accuracy: 0.7561
Epoch 40/50
326/326 [==============================] - 1s 3ms/step - loss: 0.1392 -
accuracy: 0.9663 - val_loss: 1.0232 - val_accuracy: 0.7561
Epoch 41/50
326/326 [==============================] - 1s 3ms/step - loss: 0.1325 -
accuracy: 0.9632 - val_loss: 1.2658 - val_accuracy: 0.7195
Epoch 42/50
326/326 [==============================] - 1s 3ms/step - loss: 0.1228 -
accuracy: 0.9571 - val_loss: 1.0559 - val_accuracy: 0.7561
Epoch 43/50
326/326 [==============================] - 1s 3ms/step - loss: 0.0907 -
accuracy: 0.9816 - val_loss: 1.4280 - val_accuracy: 0.7195
Epoch 44/50
326/326 [==============================] - 1s 3ms/step - loss: 0.0913 -
accuracy: 0.9724 - val_loss: 1.1149 - val_accuracy: 0.7561
Epoch 45/50
326/326 [==============================] - 1s 3ms/step - loss: 0.0644 -
accuracy: 0.9816 - val_loss: 1.0595 - val_accuracy: 0.7317
Epoch 46/50
326/326 [==============================] - 1s 3ms/step - loss: 0.0493 -
accuracy: 0.9877 - val_loss: 1.2852 - val_accuracy: 0.7439
Epoch 47/50
326/326 [==============================] - 1s 3ms/step - loss: 0.0323 -
accuracy: 0.9969 - val_loss: 1.0482 - val_accuracy: 0.7561
Epoch 48/50
326/326 [==============================] - 1s 3ms/step - loss: 0.0319 -
accuracy: 0.9939 - val_loss: 1.0892 - val_accuracy: 0.7439
Epoch 49/50
326/326 [==============================] - 1s 3ms/step - loss: 0.0259 -
accuracy: 0.9969 - val_loss: 1.3150 - val_accuracy: 0.7317
Epoch 50/50
326/326 [==============================] - 1s 3ms/step - loss: 0.0179 -
accuracy: 1.0000 - val_loss: 1.2029 - val_accuracy: 0.7439
```

```
[72]: <keras.callbacks.callbacks.History at 0x7f95066bb790>
```

Model with 8,659,305 parametrs and dense layers showed 70% test accuracy

[73]:
```
score = model_d.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 1.7012835040579748
Test accuracy: 0.7007299065589905
```

Lets define, train and test convolutional model.

[84]:
```
def cnn(input_shape, n_labels):
    model = Sequential()
    model.add(InputLayer(input_shape))
    model.add(Conv2D(64, kernel_size=(3, 3), activation="relu"))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(64, kernel_size=(3, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(n_labels, activation="softmax"))
    model.compile(optimizer=adam(lr = 0.001, decay=1e-6),
        loss="categorical_crossentropy", metrics=["accuracy"])
    model.summary()
    return model
```

[85]:
```
model = cnn(input_shape, n_labels)
batch_size = 128
epochs = 15

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,␣
 ↪validation_split=0.1)
```

```
Model: "sequential_28"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 298, 298, 64)      1792
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 149, 149, 64)      0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 147, 147, 64)      36928
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 73, 73, 64)        0
_________________________________________________________________
flatten_25 (Flatten)         (None, 341056)            0
_________________________________________________________________
dropout_11 (Dropout)         (None, 341056)            0
_________________________________________________________________
dense_92 (Dense)             (None, 9)                 3069513
```

```
=================================================================
Total params: 3,108,233
Trainable params: 3,108,233
Non-trainable params: 0
_________________________________________________________________
Train on 367 samples, validate on 41 samples
Epoch 1/15
367/367 [==============================] - 37s 101ms/step - loss: 5.9520 -
accuracy: 0.2098 - val_loss: 4.0458 - val_accuracy: 0.4390
Epoch 2/15
367/367 [==============================] - 27s 73ms/step - loss: 4.0500 -
accuracy: 0.2670 - val_loss: 3.1665 - val_accuracy: 0.3902
Epoch 3/15
367/367 [==============================] - 24s 67ms/step - loss: 2.4383 -
accuracy: 0.4469 - val_loss: 2.7795 - val_accuracy: 0.5122
Epoch 4/15
367/367 [==============================] - 26s 71ms/step - loss: 1.5441 -
accuracy: 0.5913 - val_loss: 1.8777 - val_accuracy: 0.6098
Epoch 5/15
367/367 [==============================] - 26s 72ms/step - loss: 0.9620 -
accuracy: 0.6785 - val_loss: 1.3454 - val_accuracy: 0.6341
Epoch 6/15
367/367 [==============================] - 25s 68ms/step - loss: 0.8609 -
accuracy: 0.6839 - val_loss: 0.8850 - val_accuracy: 0.6829
Epoch 7/15
367/367 [==============================] - 27s 75ms/step - loss: 0.5712 -
accuracy: 0.8665 - val_loss: 1.0238 - val_accuracy: 0.7317
Epoch 8/15
367/367 [==============================] - 25s 69ms/step - loss: 0.4446 -
accuracy: 0.9074 - val_loss: 1.2733 - val_accuracy: 0.7073
Epoch 9/15
367/367 [==============================] - 27s 72ms/step - loss: 0.3705 -
accuracy: 0.8910 - val_loss: 1.0410 - val_accuracy: 0.7561
Epoch 10/15
367/367 [==============================] - 36s 97ms/step - loss: 0.2525 -
accuracy: 0.9401 - val_loss: 0.8574 - val_accuracy: 0.7317
Epoch 11/15
367/367 [==============================] - 26s 71ms/step - loss: 0.2017 -
accuracy: 0.9537 - val_loss: 0.7203 - val_accuracy: 0.7561
Epoch 12/15
367/367 [==============================] - 25s 68ms/step - loss: 0.1531 -
accuracy: 0.9728 - val_loss: 0.7343 - val_accuracy: 0.7073
Epoch 13/15
367/367 [==============================] - 24s 67ms/step - loss: 0.1342 -
accuracy: 0.9700 - val_loss: 0.7275 - val_accuracy: 0.7073
Epoch 14/15
367/367 [==============================] - 39s 105ms/step - loss: 0.1014 -
accuracy: 0.9755 - val_loss: 0.7091 - val_accuracy: 0.7805
```

```
Epoch 15/15
367/367 [==============================] - 27s 73ms/step - loss: 0.0807 -
accuracy: 0.9837 - val_loss: 0.7146 - val_accuracy: 0.7317
```

[85]:
```
<keras.callbacks.callbacks.History at 0x7f9028ddcc10>
```

Model with 3,108,2335 parametrs and convolutional layers showed 74% test accuracy

[86]:
```python
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.9328046523741562
Test accuracy: 0.7372262477874756
```

Conclusion:

Considering the dataset we used, our simple models shown good results. 545 images is not enough to train a network that will be able to perfectly classify 9 different objects. Moreover, some of the images from the “cellphone” and “minaret” were rotated, which reflects on the models accuracy as well.

Still, from the results it is obvious, that convolutional networks are better for the image clasification tasks then dense networks.