final project

- neodoggy-

about

fruit tart

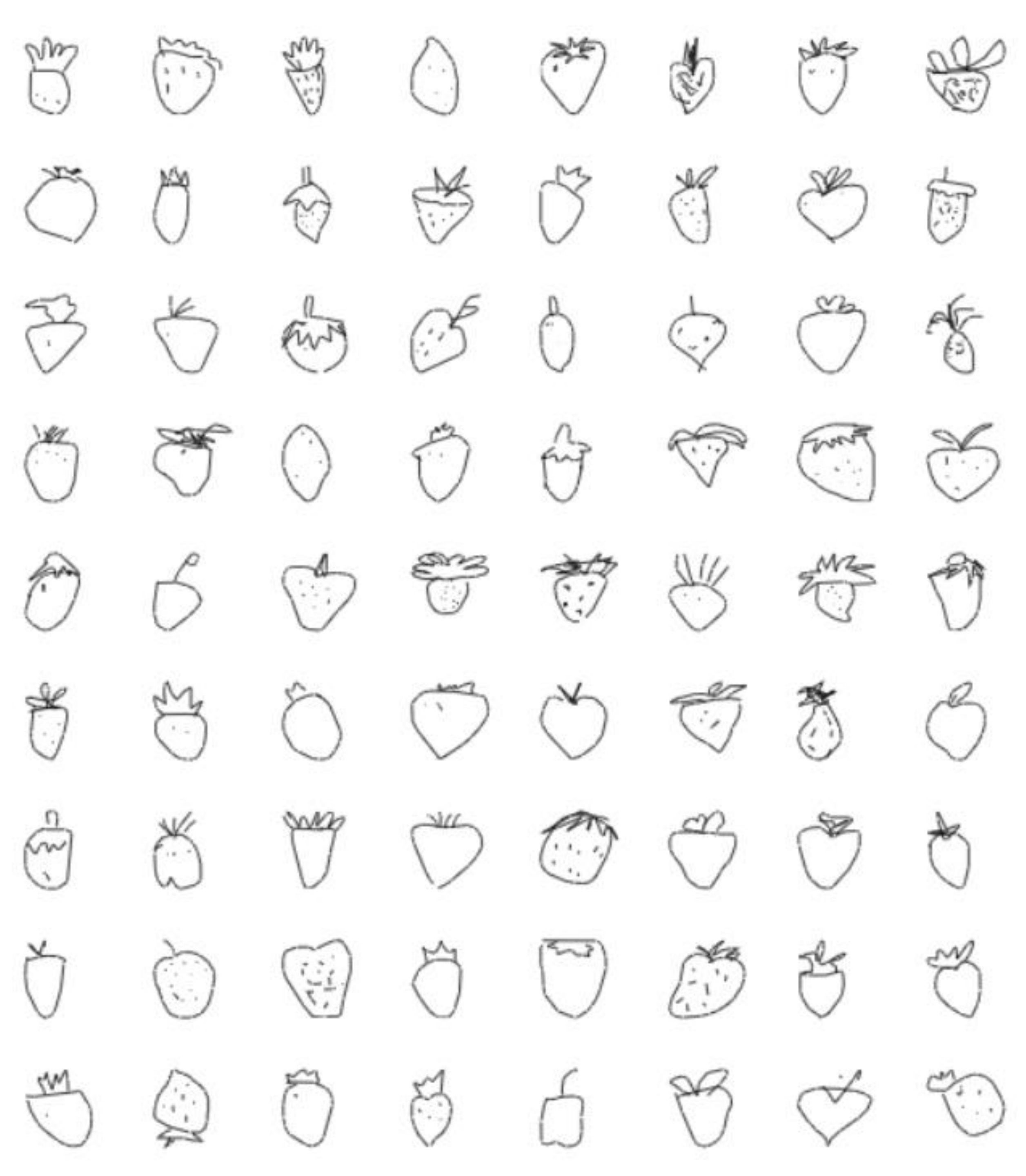
I love fruits

FRUIT AUTOMATON

how

Image Splitter

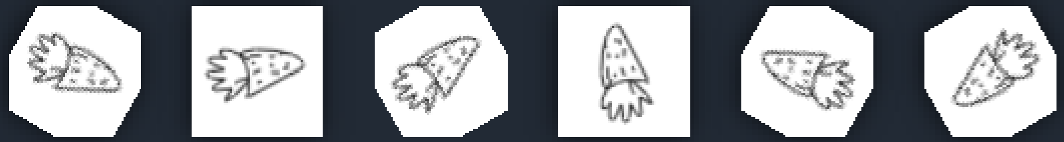
split the image into small squares that we found on web



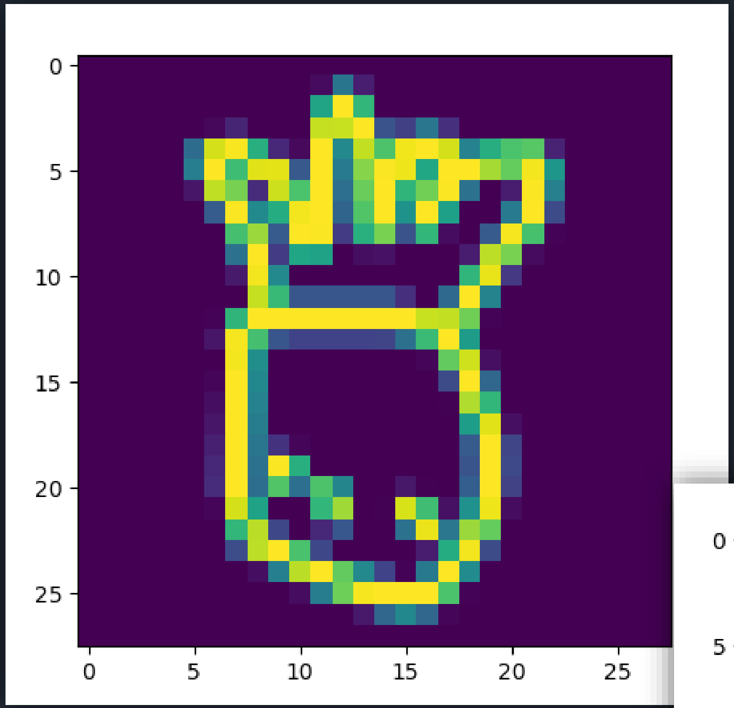
how

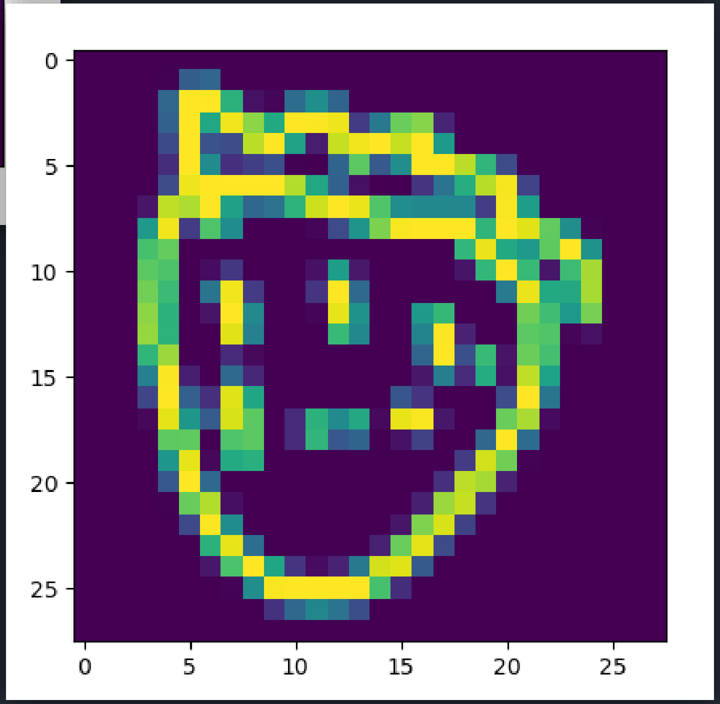
Image Data Augmentation

spin the images



how





Array

transform images into numpy arrays

code

Image Splitter

split the image into small squares that we found on web

```
from PIL import Image
import os

def crop(infile,height,width):
    im = Image.open(infile)
    imgwidth, imgheight = im.size
    for i in range(imgheight//height):
        for j in range(imgwidth//width):
            box = (j*width, i*height, (j+1)*width, (i+1)*height)
            yield im.crop(box)

if __name__=='__main__':
    infile='./ww.png'
    height=81
    width= 79
    start_num= 1
    for k,piece in enumerate(crop(infile,height,width),start_num):
        img=Image.new('RGB', (height,width), 255)
        img.paste(piece)
        path=os.path.join('/tmp',"IMG-%s.png" % k)
        img.save(path)
```

code

Image Data Augmentation

using PIL

```
from PIL import Image

colorImage  = Image.open("./3.png")

r = colorImage.rotate(60)
r.save('./img%s-60.png')
r = colorImage.rotate(90)
r.save('./img%s-90.png')
r = colorImage.rotate(120)
r.save('./img%s-120.png')
r = colorImage.rotate(180)
r.save('./img%s-180.png')
r = colorImage.rotate(240)
r.save('./img%s-240.png')
r = colorImage.rotate(300)
r.save('./img%s-300.png')
```

code

```
import cv2
import matplotlib.pyplot as plt
im = cv2.imread("./tmp.png",cv2.IMREAD_GRAYSCALE)
im=~im
plt.imshow(im)
plt.show()
```

Array

using opencv to tun it into numpy array

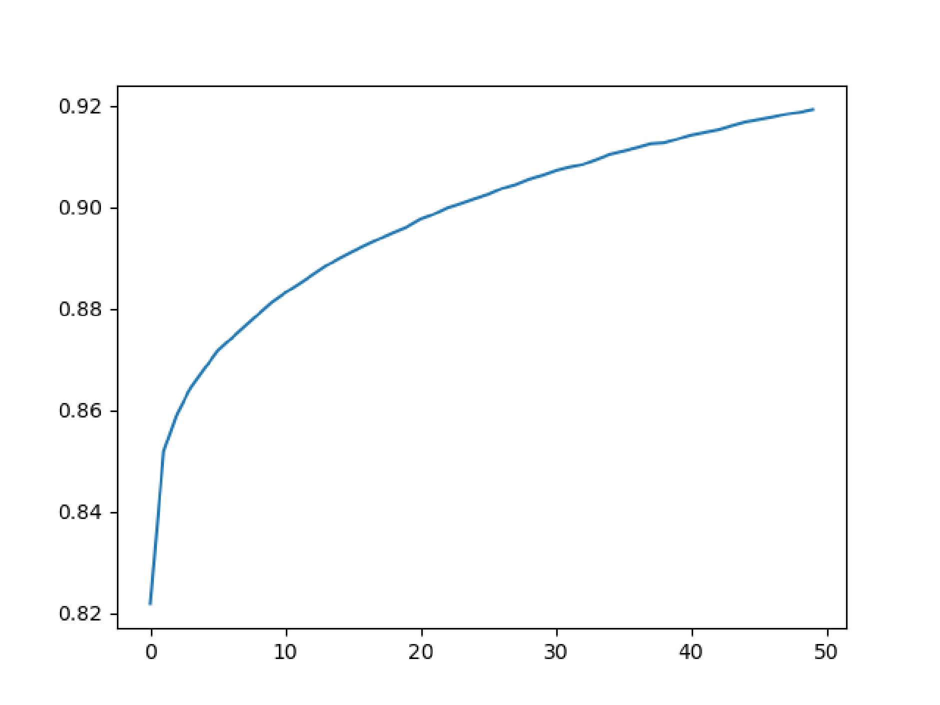
then save it to a .npy file

also doing some grayscale tricks

```
from numpy import genfromtxt
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
import pydot
#import files and label
path="./numpydatasets"
apple=np.load(f'{path}/apple.npy')
ay=np.full(len(apple),1)
banana=np.load(f'{path}/banana.npy')
by=np.full(len(banana),2)
blackberry=np.load(f'{path}/blackberry.npy')
blay=np.full(len(blackberry),3)
blueberry=np.load(f'{path}/blueberry.npy')
bluy=np.full(len(blueberry),4)
grapes=np.load(f'{path}/grapes.npy')
gy=np.full(len(grapes),5)
pear=np.load(f'{path}/pear.npy')
py=np.full(len(pear),6)
strawberry=np.load(f'{path}/strawberry.npy')
sty=np.full(len(strawberry),7)
#training dataset
Dx=np.concatenate((apple[:-4000],banana[:-4000]))
Dy=np.concatenate((ay[:-4000],by[:-4000]))
Dx=np.concatenate((Dx,blackberry[:-4000]))
Dy=np.concatenate((Dy,blay[:-4000]))
Dx=np.concatenate((Dx,blueberry[:-4000]))
Dy=np.concatenate((Dy,bluy[:-4000]))
Dx=np.concatenate((Dx,grapes[:-4000]))
Dy=np.concatenate((Dy,gy[:-4000]))
Dx=np.concatenate((Dx,pear[:-4000]))
Dy=np.concatenate((Dy,py[:-4000]))
Dx=np.concatenate((Dx,strawberry[:-4000]))
Dy=np.concatenate((Dy,sty[:-4000]))
#testing dataset
Tx=np.concatenate((apple[-4000:],banana[-4000:]))
Ty=np.concatenate((ay[-4000:],by[-4000:]))
Tx=np.concatenate((Tx,blackberry[-4000:]))
Ty=np.concatenate((Ty,blay[-4000:]))
Tx=np.concatenate((Tx,blueberry[-4000:]))
Ty=np.concatenate((Ty,bluy[-4000:]))
Tx=np.concatenate((Tx,grapes[-4000:]))
Ty=np.concatenate((Ty,gy[-4000:]))
Tx=np.concatenate((Tx,pear[-4000:]))
Ty=np.concatenate((Ty,py[-4000:]))
Tx=np.concatenate((Tx,strawberry[-4000:]))
Ty=np.concatenate((Ty,sty[-4000:]))

np.save('traindataX',Dx)
np.save('traindataY',Dy)
np.save('testdataX',Tx)
np.save('testdataY',Ty)
```

main code




```python
from numpy import genfromtxt
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
import pydot

Dx=np.load('./datasets/traindataX.npy')
Dy=np.load('./datasets/traindataY.npy')
Tx=np.load('./datasets/testdataX.npy')
Ty=np.load('./datasets/testdataY.npy')

Dx=Dx/255
Tx=Tx/255

Dy=np_utils.to_categorical(Dy,8)
Ty=np_utils.to_categorical(Ty,8)

model=Sequential()
model.add(Dense(input_dim=28*28,units=256,activation='relu'))
model.add(Dense(units=128,activation='relu'))
model.add(Dense(units=8,activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
train_history=model.fit(x=Dx,
                        y=Dy,
                        validation_split=0.2,
                        epochs=50,
                        batch_size=600,
                        verbose=2)
#plt.plot(train_history.history['loss'])
plt.plot(train_history.history['accuracy'])
plt.show()
model.evaluate(Tx,Ty,batch_size=50)
prediction=model.predict_classes(Tx)
print(prediction[:10])
```

final output

Final tested accuracy

we got a 0.7

```
Epoch 1/50
2021-06-30 09:28:29.417059: W tensorflow/core/framework/cpu_allocator_impl.cc:80] Allocation of
674365440 exceeds 10% of free system memory.
1434/1434 - 19s - loss: 0.4918 - accuracy: 0.8219 - val_loss: 5.7229 - val_accuracy: 0.3707
Epoch 2/50
1434/1434 - 9s - loss: 0.4089 - accuracy: 0.8519 - val_loss: 5.9972 - val_accuracy: 0.3688
Epoch 3/50
1434/1434 - 9s - loss: 0.3872 - accuracy: 0.8593 - val_loss: 6.7891 - val_accuracy: 0.3707
Epoch 4/50
1434/1434 - 8s - loss: 0.3721 - accuracy: 0.8644 - val_loss: 7.4124 - val_accuracy: 0.3729
Epoch 5/50
1434/1434 - 8s - loss: 0.3611 - accuracy: 0.8681 - val_loss: 7.7996 - val_accuracy: 0.3744
Epoch 6/50
1434/1434 - 8s - loss: 0.3516 - accuracy: 0.8717 - val_loss: 8.5521 - val_accuracy: 0.3747
Epoch 7/50
1434/1434 - 8s - loss: 0.3434 - accuracy: 0.8741 - val_loss: 9.2795 - val_accuracy: 0.3642
Epoch 8/50
1434/1434 - 8s - loss: 0.3362 - accuracy: 0.8765 - val_loss: 9.3587 - val_accuracy: 0.3752
Epoch 9/50
1434/1434 - 8s - loss: 0.3298 - accuracy: 0.8789 - val_loss: 10.1078 - val_accuracy: 0.3691
Epoch 10/50
1434/1434 - 9s - loss: 0.3232 - accuracy: 0.8812 - val_loss: 10.8384 - val_accuracy: 0.3750
Epoch 11/50
1434/1434 - 9s - loss: 0.3179 - accuracy: 0.8831 - val_loss: 11.7805 - val_accuracy: 0.3808
Epoch 12/50
1434/1434 - 8s - loss: 0.3126 - accuracy: 0.8848 - val_loss: 12.5187 - val_accuracy: 0.3763
Epoch 13/50
1434/1434 - 8s - loss: 0.3080 - accuracy: 0.8866 - val_loss: 13.2254 - val_accuracy: 0.3839
Epoch 14/50
1434/1434 - 8s - loss: 0.3031 - accuracy: 0.8884 - val_loss: 13.6483 - val_accuracy: 0.3741
Epoch 15/50
1434/1434 - 8s - loss: 0.2986 - accuracy: 0.8899 - val_loss: 14.2617 - val_accuracy: 0.3707
Epoch 16/50
1434/1434 - 9s - loss: 0.2948 - accuracy: 0.8913 - val_loss: 14.8381 - val_accuracy: 0.3666
Epoch 17/50
1434/1434 - 10s - loss: 0.2906 - accuracy: 0.8926 - val_loss: 14.6691 - val_accuracy: 0.3693
Epoch 18/50
1434/1434 - 10s - loss: 0.2874 - accuracy: 0.8938 - val_loss: 15.7088 - val_accuracy: 0.3666
Epoch 19/50
1434/1434 - 10s - loss: 0.2834 - accuracy: 0.8950 - val_loss: 15.9644 - val_accuracy: 0.3687
Epoch 20/50
1434/1434 - 10s - loss: 0.2802 - accuracy: 0.8961 - val_loss: 16.2650 - val_accuracy: 0.3697
Epoch 21/50
1434/1434 - 10s - loss: 0.2766 - accuracy: 0.8976 - val_loss: 16.4266 - val_accuracy: 0.3728
Epoch 22/50
1434/1434 - 10s - loss: 0.2737 - accuracy: 0.8986 - val_loss: 16.5820 - val_accuracy: 0.3747
Epoch 23/50
1434/1434 - 10s - loss: 0.2702 - accuracy: 0.8999 - val_loss: 18.1298 - val_accuracy: 0.3767
Epoch 24/50
1434/1434 - 10s - loss: 0.2677 - accuracy: 0.9007 - val_loss: 16.9166 - val_accuracy: 0.3752
Epoch 25/50
1434/1434 - 10s - loss: 0.2650 - accuracy: 0.9016 - val_loss: 18.3619 - val_accuracy: 0.3696
Epoch 26/50
1434/1434 - 10s - loss: 0.2620 - accuracy: 0.9025 - val_loss: 19.3841 - val_accuracy: 0.3703
Epoch 27/50
1434/1434 - 9s - loss: 0.2594 - accuracy: 0.9036 - val_loss: 18.0540 - val_accuracy: 0.3640
Epoch 28/50
1434/1434 - 10s - loss: 0.2569 - accuracy: 0.9044 - val_loss: 19.1290 - val_accuracy: 0.3701
Epoch 29/50
1434/1434 - 10s - loss: 0.2545 - accuracy: 0.9055 - val_loss: 19.6109 - val_accuracy: 0.3639
Epoch 30/50
1434/1434 - 8s - loss: 0.2519 - accuracy: 0.9062 - val_loss: 19.4441 - val_accuracy: 0.3666
Epoch 31/50
1434/1434 - 10s - loss: 0.2500 - accuracy: 0.9072 - val_loss: 19.9860 - val_accuracy: 0.3733
Epoch 32/50
1434/1434 - 10s - loss: 0.2478 - accuracy: 0.9078 - val_loss: 19.6027 - val_accuracy: 0.3738
Epoch 33/50
1434/1434 - 9s - loss: 0.2456 - accuracy: 0.9084 - val_loss: 20.4177 - val_accuracy: 0.3703
Epoch 34/50
1434/1434 - 9s - loss: 0.2435 - accuracy: 0.9094 - val_loss: 22.0064 - val_accuracy: 0.3754
Epoch 35/50
1434/1434 - 8s - loss: 0.2411 - accuracy: 0.9104 - val_loss: 21.4604 - val_accuracy: 0.3779
Epoch 36/50
1434/1434 - 9s - loss: 0.2395 - accuracy: 0.9110 - val_loss: 21.5386 - val_accuracy: 0.3672
Epoch 37/50
1434/1434 - 8s - loss: 0.2373 - accuracy: 0.9117 - val_loss: 21.6748 - val_accuracy: 0.3615
Epoch 38/50
1434/1434 - 9s - loss: 0.2357 - accuracy: 0.9125 - val_loss: 22.6208 - val_accuracy: 0.3714
Epoch 39/50
1434/1434 - 9s - loss: 0.2340 - accuracy: 0.9126 - val_loss: 22.7600 - val_accuracy: 0.3622
Epoch 40/50
1434/1434 - 9s - loss: 0.2320 - accuracy: 0.9134 - val_loss: 23.7085 - val_accuracy: 0.3624
Epoch 41/50
1434/1434 - 9s - loss: 0.2303 - accuracy: 0.9142 - val_loss: 23.8094 - val_accuracy: 0.3709
Epoch 42/50
1434/1434 - 9s - loss: 0.2285 - accuracy: 0.9147 - val_loss: 24.1740 - val_accuracy: 0.3711
Epoch 43/50
1434/1434 - 10s - loss: 0.2274 - accuracy: 0.9152 - val_loss: 24.8494 - val_accuracy: 0.3645
Epoch 44/50
1434/1434 - 10s - loss: 0.2255 - accuracy: 0.9160 - val_loss: 23.7945 - val_accuracy: 0.3687
Epoch 45/50
1434/1434 - 10s - loss: 0.2238 - accuracy: 0.9167 - val_loss: 25.7254 - val_accuracy: 0.3748
Epoch 46/50
1434/1434 - 10s - loss: 0.2224 - accuracy: 0.9172 - val_loss: 25.2353 - val_accuracy: 0.3703
Epoch 47/50
1434/1434 - 11s - loss: 0.2208 - accuracy: 0.9177 - val_loss: 26.0976 - val_accuracy: 0.3696
Epoch 48/50
1434/1434 - 11s - loss: 0.2196 - accuracy: 0.9183 - val_loss: 25.3889 - val_accuracy: 0.3666
Epoch 49/50
1434/1434 - 13s - loss: 0.2181 - accuracy: 0.9187 - val_loss: 26.7714 - val_accuracy: 0.3682
Epoch 50/50
1434/1434 - 13s - loss: 0.2165 - accuracy: 0.9192 - val_loss: 26.7316 - val_accuracy: 0.3623
560/560 [==============================] - 1s 2ms/step - loss: 7.4309 - accuracy: 0.6988
/usr/local/lib/python3.8/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning:
`model.predict_classes()` is deprecated and will be removed after 2021-01-01. Please use instead:*
`np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it
uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model
does binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
[1 1 1 4 1 1 1 1 1 1]
```

links

Github

https://github.com/NeoDoggy/ai_project