Assignment 6.2

October 12, 2020

1 Assignment 6.2

Brandon Sams

06Oct2020

1.1 Part A:

Using section 5.2 in Deep Learning with Python as a guide, create a ConvNet model that classifies images CIFAR10 small images classification dataset. Do not use dropout or data-augmentation in this part. Save the model, predictions, metrics, and validation plots in the dsc650/assignments/assignment06/results directory. If you are using JupyterHub, you can include those plots in your Jupyter notebook.

1.2 Downloading the data

[1]:
```
from keras.datasets import cifar10
import matplotlib.pyplot as plt
```

[2]:
```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

[3]:
```
print(f'Training Data: x = {x_train.shape}, y = {y_train.shape}')
print(f'Testing Data:  x = {x_test.shape}, y = {y_test.shape}')
```

```
Training Data: x = (50000, 32, 32, 3), y = (50000, 1)
Testing Data:  x = (10000, 32, 32, 3), y = (10000, 1)
```

[4]:
```
# plot first few images
for i in range(9):
    # define subplot
    plt.subplot(330 + 1 + i)
    # plot raw pixel data
    plt.imshow(x_train[i])
# show the figure
plt.show()
```

[5]:
```
from keras.utils import to_categorical
y_test = to_categorical(y_test)
y_train = to_categorical(y_train)

print(f'Training Data: x = {x_train.shape}, y = {y_train.shape}')
print(f'Testing Data:  x = {x_test.shape}, y = {y_test.shape}')
```

```
Training Data: x = (50000, 32, 32, 3), y = (50000, 10)
Testing Data:  x = (10000, 32, 32, 3), y = (10000, 10)
```

1.3 Instantiating a convnet

[6]:
```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
```

```
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(10, activation='sigmoid'))
```

[7]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 3,457,738
Trainable params: 3,457,738
Non-trainable params: 0
_________________________________________________________________
```

[8]:
```
x_train.shape
```

[8]:
```
(50000, 32, 32, 3)
```

1.4 Configuring a model for training

[9]:
```
from keras import optimizers

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

1.5 Preprocess the data

[11]:
```
import skimage.transform
new_shape = (150,150,3)
x_train = [skimage.transform.resize(image, new_shape) for image in x_train]
x_test = [skimage.transform.resize(image, new_shape) for image in x_test]
```

[12]:
```
import numpy as np
x_test = np.stack(x_test, axis=0)
x_train = np.stack(x_train, axis=0)
```

[13]:
```
x_test.shape
```

[13]:
```
(10000, 150, 150, 3)
```

[14]:
```
x_train.shape
```

[14]:
```
(50000, 150, 150, 3)
```

[15]:
```
from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(rescale=1./255)
history = model.fit(
        datagen.flow(x_train, y_train, batch_size=20),
        steps_per_epoch=len(x_train) / 20,
        epochs=20,
        validation_data=datagen.flow(x_test, y_test, batch_size=20),
        validation_steps=len(x_test) / 20)
```

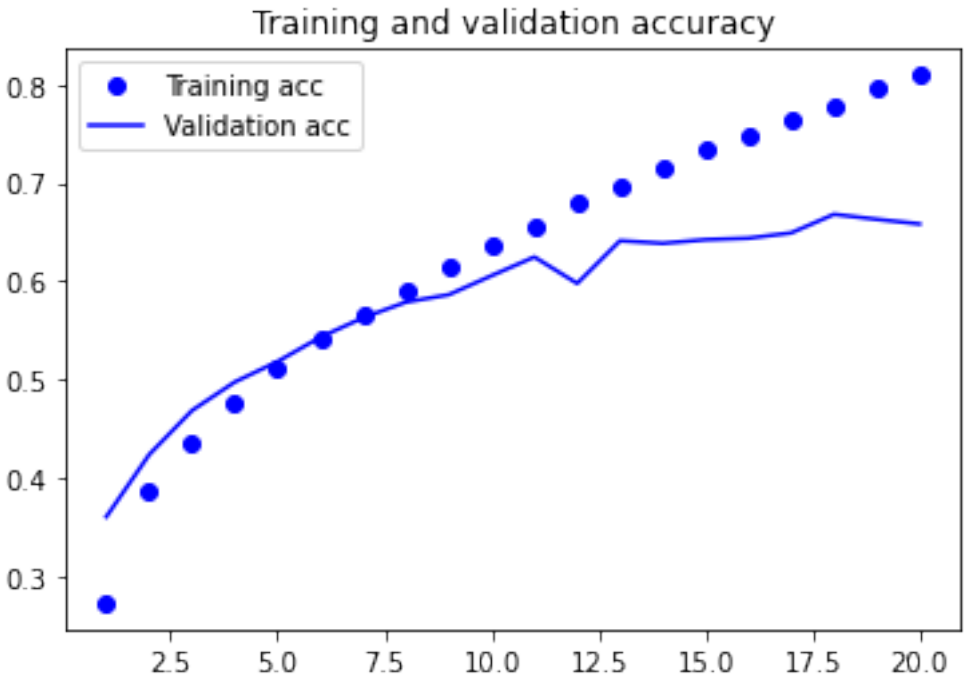
```
Epoch 1/20
2500/2500 [==============================] - 477s 191ms/step - loss: 2.0174 -
acc: 0.2724 - val_loss: 1.7926 - val_acc: 0.3601
Epoch 2/20
2500/2500 [==============================] - 455s 182ms/step - loss: 1.7199 -
acc: 0.3865 - val_loss: 1.6171 - val_acc: 0.4233
Epoch 3/20
2500/2500 [==============================] - 457s 183ms/step - loss: 1.5729 -
acc: 0.4365 - val_loss: 1.4889 - val_acc: 0.4682
Epoch 4/20
2500/2500 [==============================] - 450s 180ms/step - loss: 1.4751 -
acc: 0.4768 - val_loss: 1.4253 - val_acc: 0.4972
Epoch 5/20
2500/2500 [==============================] - 454s 182ms/step - loss: 1.3856 -
acc: 0.5111 - val_loss: 1.3704 - val_acc: 0.5180
Epoch 6/20
2500/2500 [==============================] - 444s 178ms/step - loss: 1.3055 -
acc: 0.5418 - val_loss: 1.2894 - val_acc: 0.5428
Epoch 7/20
```

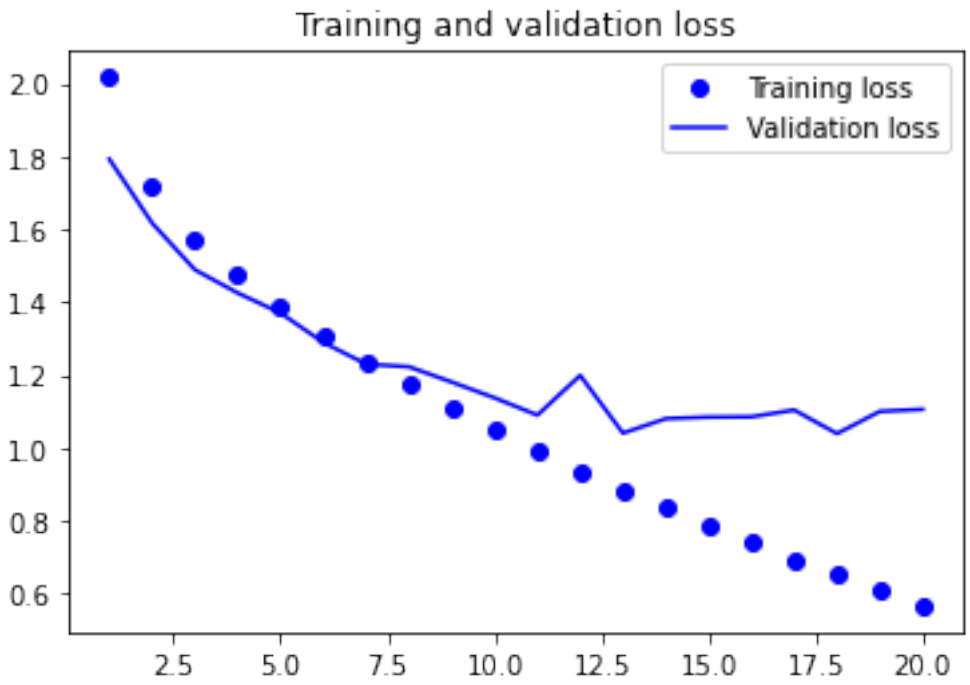
```
2500/2500 [==============================] - 441s 176ms/step - loss: 1.2362 -
acc: 0.5665 - val_loss: 1.2300 - val_acc: 0.5626
Epoch 8/20
2500/2500 [==============================] - 442s 177ms/step - loss: 1.1716 -
acc: 0.5908 - val_loss: 1.2228 - val_acc: 0.5785
Epoch 9/20
2500/2500 [==============================] - 438s 175ms/step - loss: 1.1071 -
acc: 0.6140 - val_loss: 1.1808 - val_acc: 0.5860
Epoch 10/20
2500/2500 [==============================] - 429s 172ms/step - loss: 1.0471 -
acc: 0.6362 - val_loss: 1.1376 - val_acc: 0.6054
Epoch 11/20
2500/2500 [==============================] - 430s 172ms/step - loss: 0.9924 -
acc: 0.6565 - val_loss: 1.0901 - val_acc: 0.6244
Epoch 12/20
2500/2500 [==============================] - 430s 172ms/step - loss: 0.9366 -
acc: 0.6786 - val_loss: 1.1991 - val_acc: 0.5975
Epoch 13/20
2500/2500 [==============================] - 431s 173ms/step - loss: 0.8837 -
acc: 0.6971 - val_loss: 1.0408 - val_acc: 0.6409
Epoch 14/20
2500/2500 [==============================] - 432s 173ms/step - loss: 0.8352 -
acc: 0.7143 - val_loss: 1.0805 - val_acc: 0.6383
Epoch 15/20
2500/2500 [==============================] - 434s 173ms/step - loss: 0.7839 -
acc: 0.7336 - val_loss: 1.0847 - val_acc: 0.6420
Epoch 16/20
2500/2500 [==============================] - 431s 173ms/step - loss: 0.7402 -
acc: 0.7470 - val_loss: 1.0854 - val_acc: 0.6435
Epoch 17/20
2500/2500 [==============================] - 435s 174ms/step - loss: 0.6930 -
acc: 0.7653 - val_loss: 1.1041 - val_acc: 0.6489
Epoch 18/20
2500/2500 [==============================] - 431s 172ms/step - loss: 0.6508 -
acc: 0.7787 - val_loss: 1.0394 - val_acc: 0.6678
Epoch 19/20
2500/2500 [==============================] - 430s 172ms/step - loss: 0.6103 -
acc: 0.7953 - val_loss: 1.1003 - val_acc: 0.6627
Epoch 20/20
2500/2500 [==============================] - 432s 173ms/step - loss: 0.5667 -
acc: 0.8094 - val_loss: 1.1061 - val_acc: 0.6580
```

[16]:
```
model.save('./results/model_6_2_a.h5')
```

1.6 Loss and Accuracy during training

[17]:
```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





1.7 Part B

Using section 5.2 in Deep Learning with Python as a guide, create a ConvNet model that classifies images CIFAR10 small images classification dataset. This time includes dropout and data-augmentation. Save the model, predictions, metrics, and validation plots in the dsc650/assignments/assignment06/results directory. If you are using JupyterHub, you can include those plots in your Jupyter notebook.

1.8 Instantiating a convnet

[27]:
```
from keras import layers
from keras import models
#from keras.optimizers import SGD

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
```

```
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(10, activation='sigmoid'))

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              #optimizer=SGD(lr=0.01, momentum=0.9),
              metrics=['acc'])

model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_24 (Conv2D)           (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 7, 7, 128)         0
_________________________________________________________________
flatten_6 (Flatten)          (None, 6272)              0
_________________________________________________________________
dropout_5 (Dropout)          (None, 6272)              0
_________________________________________________________________
dense_12 (Dense)             (None, 512)               3211776
_________________________________________________________________
dense_13 (Dense)             (None, 10)                5130
=================================================================
Total params: 3,457,738
Trainable params: 3,457,738
Non-trainable params: 0
_________________________________________________________________
```

[28]:
```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
```

```
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(
    rescale=1./255)

history = model.fit(
        train_datagen.flow(x_train, y_train, batch_size=20),
        steps_per_epoch=len(x_train) / 20,
        epochs=20,
        validation_data=test_datagen.flow(x_test, y_test, batch_size=20),
        validation_steps=len(x_test) / 20)
```

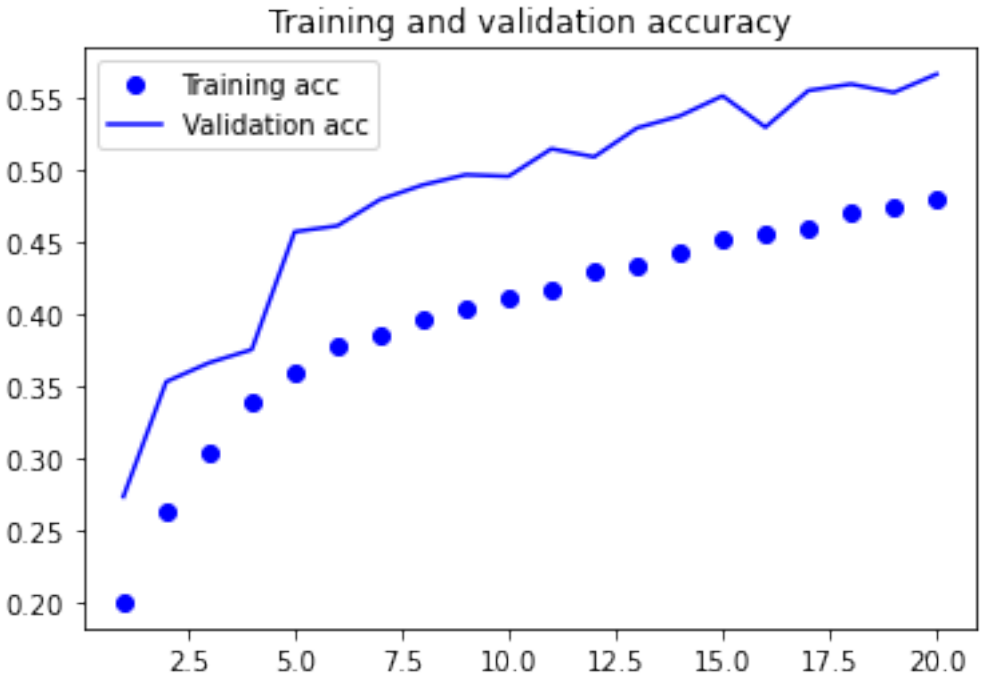
```
Epoch 1/20
2500/2500 [==============================] - 448s 179ms/step - loss: 2.1492 -
acc: 0.2011 - val_loss: 1.9773 - val_acc: 0.2738
Epoch 2/20
2500/2500 [==============================] - 447s 179ms/step - loss: 2.0053 -
acc: 0.2628 - val_loss: 1.8135 - val_acc: 0.3532
Epoch 3/20
2500/2500 [==============================] - 448s 179ms/step - loss: 1.9008 -
acc: 0.3042 - val_loss: 1.7580 - val_acc: 0.3663
Epoch 4/20
2500/2500 [==============================] - 447s 179ms/step - loss: 1.8185 -
acc: 0.3391 - val_loss: 1.7575 - val_acc: 0.3754
Epoch 5/20
2500/2500 [==============================] - 461s 184ms/step - loss: 1.7672 -
acc: 0.3592 - val_loss: 1.5125 - val_acc: 0.4568
Epoch 6/20
2500/2500 [==============================] - 541s 217ms/step - loss: 1.7263 -
acc: 0.3772 - val_loss: 1.5047 - val_acc: 0.4609
Epoch 7/20
2500/2500 [==============================] - 493s 197ms/step - loss: 1.6963 -
acc: 0.3851 - val_loss: 1.4505 - val_acc: 0.4791
Epoch 8/20
2500/2500 [==============================] - 448s 179ms/step - loss: 1.6700 -
acc: 0.3971 - val_loss: 1.4171 - val_acc: 0.4892
Epoch 9/20
2500/2500 [==============================] - 443s 177ms/step - loss: 1.6438 -
acc: 0.4047 - val_loss: 1.4014 - val_acc: 0.4962
Epoch 10/20
2500/2500 [==============================] - 444s 178ms/step - loss: 1.6256 -
```

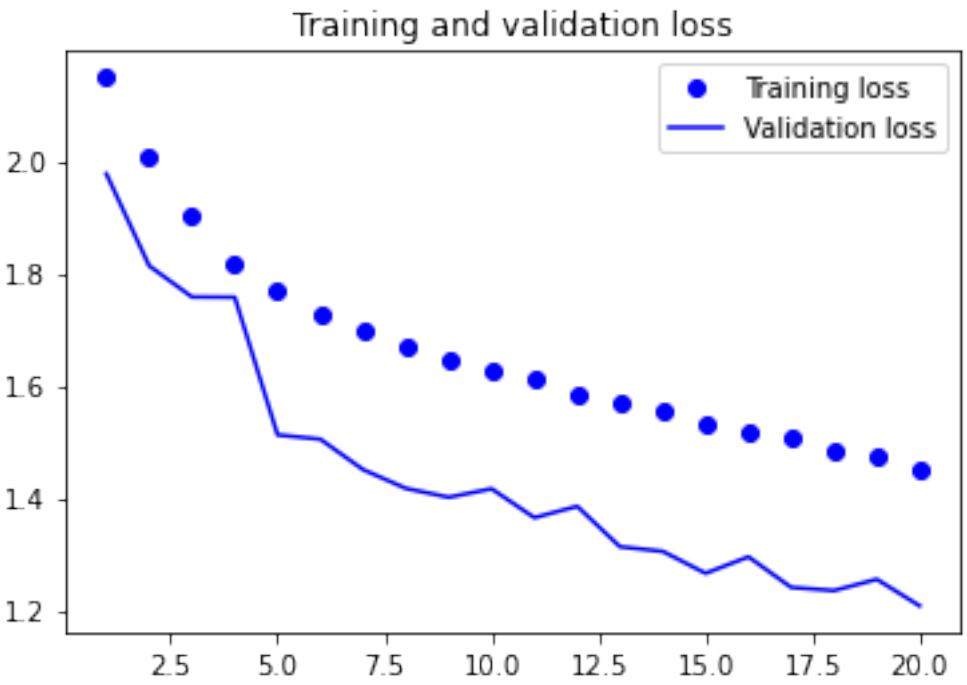
```
acc: 0.4103 - val_loss: 1.4165 - val_acc: 0.4951
Epoch 11/20
2500/2500 [==============================] - 458s 183ms/step - loss: 1.6108 -
acc: 0.4170 - val_loss: 1.3651 - val_acc: 0.5141
Epoch 12/20
2500/2500 [==============================] - 446s 178ms/step - loss: 1.5860 -
acc: 0.4291 - val_loss: 1.3853 - val_acc: 0.5085
Epoch 13/20
2500/2500 [==============================] - 443s 177ms/step - loss: 1.5703 -
acc: 0.4337 - val_loss: 1.3133 - val_acc: 0.5285
Epoch 14/20
2500/2500 [==============================] - 447s 179ms/step - loss: 1.5529 -
acc: 0.4419 - val_loss: 1.3051 - val_acc: 0.5368
Epoch 15/20
2500/2500 [==============================] - 444s 178ms/step - loss: 1.5326 -
acc: 0.4512 - val_loss: 1.2660 - val_acc: 0.5507
Epoch 16/20
2500/2500 [==============================] - 446s 178ms/step - loss: 1.5179 -
acc: 0.4551 - val_loss: 1.2954 - val_acc: 0.5290
Epoch 17/20
2500/2500 [==============================] - 446s 178ms/step - loss: 1.5063 -
acc: 0.4600 - val_loss: 1.2409 - val_acc: 0.5543
Epoch 18/20
2500/2500 [==============================] - 445s 178ms/step - loss: 1.4844 -
acc: 0.4695 - val_loss: 1.2353 - val_acc: 0.5588
Epoch 19/20
2500/2500 [==============================] - 448s 179ms/step - loss: 1.4755 -
acc: 0.4733 - val_loss: 1.2554 - val_acc: 0.5530
Epoch 20/20
2500/2500 [==============================] - 447s 179ms/step - loss: 1.4522 -
acc: 0.4799 - val_loss: 1.2081 - val_acc: 0.5657
```

[29]:
```
model.save('./results/model_6_2_b.h5')
```

[30]:
```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
```

```
plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```




Training and validation loss
Training loss
Validation loss
2.0
1.8
1.6
1.4
1.2
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

[ ]:

[ ]: