In [18]:

```python
from numpy.random import seed
seed(1)
import tensorflow
tensorflow.random.set_seed(2)
```

In [19]:

```python
#Import Stuff here

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

import tensorflow.keras as keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten, GlobalAveragePooling2D, Depthw
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd
import numpy as np
from keras.models import load_model
from tensorflow.keras.utils import plot_model
```

In [20]:

```python
#creates a Image Generator from the directories and scales the images by dividing by 255. Also changes the images

def load_data(train_folder = 'data/train/'):

    datagen = ImageDataGenerator(rescale = 1/255)

    train_it = datagen.flow_from_directory (train_folder, target_size = (256, 256),
                                        class_mode = 'categorical', color_mode="rgb", batch_size=64, seed = 1

    val_it = datagen.flow_from_directory ('data/validation/', target_size = (256, 256),
                                        class_mode = 'categorical', color_mode="rgb", batch_size=64, seed = 1

    test_it = datagen.flow_from_directory ('data/test/', target_size = (256, 256),
                                        class_mode = 'categorical', color_mode="rgb", batch_size=64, seed = 1

    return train_it, val_it, test_it

train_it, val_it, test_it = load_data('resized/')
```

```
Found 13600 images belonging to 17 classes.
Found 1700 images belonging to 17 classes.
Found 1700 images belonging to 17 classes.
```

In [21]:

```python
batchX, batchy = train_it.next()
print('Batch shape=%s, min=%.3f, max=%.3f' % (batchX.shape, batchX.min(), batchX.max()))
```

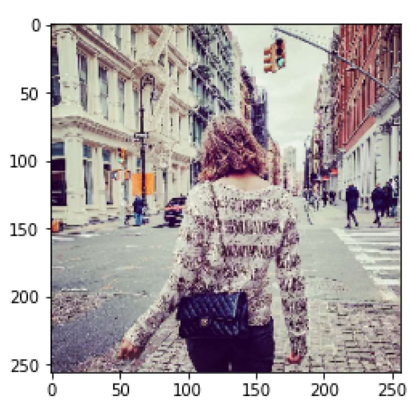
```
Batch shape=(64, 256, 256, 3), min=0.000, max=1.000
```

In [22]:

```python
plt.imshow(batchX[0])
```

Out[22]:

```
<matplotlib.image.AxesImage at 0x210a731f2b0>
```



In [23]:
```python
batchy[0]
```

Out[23]:
```
array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
      dtype=float32)
```

In [24]:
```python
train_it.class_indices
```

Out[24]:
```
{'1977': 0,
 'Amaro': 1,
 'Brannan': 2,
 'Clarendon': 3,
 'Gingham': 4,
 'He-Fe': 5,
 'Hudson': 6,
 'Lo-Fi': 7,
 'Mayfair': 8,
 'Nashville': 9,
 'Original': 10,
 'Perpetua': 11,
 'Sutro': 12,
 'Toaster': 13,
 'Valencia': 14,
 'Willow': 15,
 'X-ProII': 16}
```

In [36]:
```python
def build_model():
    # TODO: build the model,


    model = Sequential()
    model.add(BatchNormalization()) #added


    model.add(Conv2D(64, kernel_size= 3, activation = 'relu', padding='same'))
    model.add(MaxPooling2D (pool_size= 2))


    model.add(Conv2D(128, kernel_size= 3, activation = 'relu', padding='same'))
    model.add(MaxPooling2D (pool_size= 2))


    model.add(Conv2D(256, kernel_size= 3, activation = 'relu',  padding='same'))
    model.add(MaxPooling2D (pool_size= 2))


    model.add(Conv2D(512, kernel_size= 3, activation = 'relu',  padding='same'))
    model.add(MaxPooling2D (pool_size= 2))

    model.add(Conv2D(256, kernel_size= 3, activation = 'relu',  padding='same'))
    model.add(MaxPooling2D (pool_size= 2))

    model.add(Conv2D(128, kernel_size= 3, activation = 'relu',  padding='same'))
    model.add(Flatten ())


    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.2))

    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.2))

    model.add(Dense(17, activation='softmax'))

    return model

model = build_model()
```

In [38]:
```python
def compile_model(model):

    model.compile(optimizer = keras.optimizers.Adam(learning_rate = 0.001),
              loss=keras.losses.CategoricalCrossentropy(),
              metrics=['accuracy'])
```

```
    return model

def train_model(model, train_it, val_it):

    callback = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights= True)

    history = model.fit(train_it, epochs = 50, steps_per_epoch = 107, verbose = 2,
                        validation_data= val_it, validation_steps = 27, callbacks=[callback])
    return model, history


model = compile_model(model)

model, history = train_model(model, train_it, val_it)

print (model.summary())
```

```
Epoch 1/50
107/107 - 58s - loss: 3.0177 - accuracy: 0.1948 - val_loss: 2.0001 - val_accuracy: 0.3347 - 58s/epoch - 543ms/ste
p
Epoch 2/50
107/107 - 57s - loss: 2.0548 - accuracy: 0.3211 - val_loss: 2.1032 - val_accuracy: 0.3100 - 57s/epoch - 530ms/ste
p
Epoch 3/50
107/107 - 55s - loss: 1.7165 - accuracy: 0.4372 - val_loss: 1.4881 - val_accuracy: 0.5218 - 55s/epoch - 516ms/ste
p
Epoch 4/50
107/107 - 53s - loss: 1.4340 - accuracy: 0.5220 - val_loss: 1.2355 - val_accuracy: 0.5929 - 53s/epoch - 498ms/ste
p
Epoch 5/50
107/107 - 54s - loss: 1.2348 - accuracy: 0.5835 - val_loss: 1.1253 - val_accuracy: 0.6324 - 54s/epoch - 502ms/ste
p
Epoch 6/50
107/107 - 56s - loss: 1.2161 - accuracy: 0.5951 - val_loss: 1.3149 - val_accuracy: 0.5735 - 56s/epoch - 522ms/ste
p
Epoch 7/50
107/107 - 55s - loss: 1.1794 - accuracy: 0.6121 - val_loss: 0.9472 - val_accuracy: 0.6600 - 55s/epoch - 516ms/ste
p
Epoch 8/50
107/107 - 53s - loss: 0.9580 - accuracy: 0.6759 - val_loss: 0.9371 - val_accuracy: 0.6994 - 53s/epoch - 498ms/ste
p
Epoch 9/50
107/107 - 54s - loss: 0.8610 - accuracy: 0.7042 - val_loss: 0.8793 - val_accuracy: 0.7118 - 54s/epoch - 502ms/ste
p
Epoch 10/50
107/107 - 54s - loss: 0.8712 - accuracy: 0.7011 - val_loss: 0.7920 - val_accuracy: 0.7371 - 54s/epoch - 505ms/ste
p
Epoch 11/50
107/107 - 53s - loss: 0.8743 - accuracy: 0.7041 - val_loss: 0.8964 - val_accuracy: 0.6947 - 53s/epoch - 498ms/ste
p
Epoch 12/50
107/107 - 53s - loss: 0.7472 - accuracy: 0.7452 - val_loss: 0.7231 - val_accuracy: 0.7547 - 53s/epoch - 499ms/ste
p
Epoch 13/50
107/107 - 54s - loss: 0.7119 - accuracy: 0.7535 - val_loss: 0.9303 - val_accuracy: 0.7271 - 54s/epoch - 504ms/ste
p
Epoch 14/50
107/107 - 54s - loss: 0.7983 - accuracy: 0.7363 - val_loss: 0.7511 - val_accuracy: 0.7682 - 54s/epoch - 504ms/ste
p
Epoch 15/50
107/107 - 53s - loss: 0.6867 - accuracy: 0.7661 - val_loss: 0.6890 - val_accuracy: 0.7835 - 53s/epoch - 499ms/ste
p
Epoch 16/50
107/107 - 54s - loss: 0.7063 - accuracy: 0.7629 - val_loss: 0.7786 - val_accuracy: 0.7612 - 54s/epoch - 501ms/ste
p
Epoch 17/50
107/107 - 54s - loss: 0.5829 - accuracy: 0.7921 - val_loss: 0.8011 - val_accuracy: 0.7447 - 54s/epoch - 501ms/ste
p
Epoch 18/50
107/107 - 53s - loss: 0.5414 - accuracy: 0.8169 - val_loss: 0.6599 - val_accuracy: 0.8006 - 53s/epoch - 499ms/ste
p
Epoch 19/50
107/107 - 54s - loss: 0.5988 - accuracy: 0.7984 - val_loss: 0.5923 - val_accuracy: 0.8112 - 54s/epoch - 501ms/ste
```

```
p
Epoch 20/50
107/107 - 53s - loss: 0.5126 - accuracy: 0.8250 - val_loss: 0.8631 - val_accuracy: 0.7400 - 53s/epoch - 499ms/ste
p
Epoch 21/50
107/107 - 53s - loss: 0.4770 - accuracy: 0.8311 - val_loss: 0.5434 - val_accuracy: 0.8294 - 53s/epoch - 495ms/ste
p
Epoch 22/50
107/107 - 53s - loss: 0.5178 - accuracy: 0.8324 - val_loss: 0.6929 - val_accuracy: 0.8147 - 53s/epoch - 494ms/ste
p
Epoch 23/50
107/107 - 53s - loss: 0.4603 - accuracy: 0.8411 - val_loss: 0.8407 - val_accuracy: 0.7506 - 53s/epoch - 493ms/ste
p
Epoch 24/50
107/107 - 52s - loss: 0.4510 - accuracy: 0.8471 - val_loss: 0.6981 - val_accuracy: 0.8082 - 52s/epoch - 489ms/ste
p
Epoch 25/50
107/107 - 53s - loss: 0.4311 - accuracy: 0.8535 - val_loss: 0.6690 - val_accuracy: 0.8118 - 53s/epoch - 496ms/ste
p
Epoch 26/50
107/107 - 53s - loss: 0.3329 - accuracy: 0.8854 - val_loss: 0.5588 - val_accuracy: 0.8412 - 53s/epoch - 498ms/ste
p
Epoch 27/50
107/107 - 56s - loss: 0.3568 - accuracy: 0.8791 - val_loss: 0.7595 - val_accuracy: 0.8071 - 56s/epoch - 520ms/ste
p
Epoch 28/50
107/107 - 54s - loss: 0.3948 - accuracy: 0.8706 - val_loss: 0.8900 - val_accuracy: 0.7624 - 54s/epoch - 504ms/ste
p
Epoch 29/50
107/107 - 54s - loss: 0.4540 - accuracy: 0.8517 - val_loss: 0.6114 - val_accuracy: 0.8312 - 54s/epoch - 502ms/ste
p
Epoch 30/50
107/107 - 54s - loss: 0.2831 - accuracy: 0.9052 - val_loss: 0.6073 - val_accuracy: 0.8235 - 54s/epoch - 503ms/ste
p
Epoch 31/50
107/107 - 54s - loss: 0.2736 - accuracy: 0.9077 - val_loss: 0.7255 - val_accuracy: 0.8276 - 54s/epoch - 501ms/ste
p
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 batch_normalization_1 (Batc  (None, None, None, 3)    12
 hNormalization)

 conv2d_12 (Conv2D)          (None, None, None, 64)    1792

 max_pooling2d_10 (MaxPoolin  (None, None, None, 64)   0
 g2D)

 conv2d_13 (Conv2D)          (None, None, None, 128)   73856

 max_pooling2d_11 (MaxPoolin  (None, None, None, 128)  0
 g2D)

 conv2d_14 (Conv2D)          (None, None, None, 256)   295168

 max_pooling2d_12 (MaxPoolin  (None, None, None, 256)  0
 g2D)

 conv2d_15 (Conv2D)          (None, None, None, 512)   1180160

 max_pooling2d_13 (MaxPoolin  (None, None, None, 512)  0
 g2D)

 conv2d_16 (Conv2D)          (None, None, None, 256)   1179904

 max_pooling2d_14 (MaxPoolin  (None, None, None, 256)  0
 g2D)

 conv2d_17 (Conv2D)          (None, None, None, 128)   295040

 flatten_2 (Flatten)         (None, None)              0

 dense_6 (Dense)             (None, 128)               1048704
```

```
 dropout_4 (Dropout)         (None, 128)               0

 dense_7 (Dense)             (None, 64)                8256

 dropout_5 (Dropout)         (None, 64)                0

 dense_8 (Dense)             (None, 17)                1105

=================================================================
Total params: 4,083,997
Trainable params: 4,083,991
Non-trainable params: 6
_________________________________________________________________
None
```

In [39]:

```python
def eval_model(model, test_it):
    # TODO: evaluate the model

    test_loss, test_accuracy = model.evaluate (test_it, steps = 27)

    return test_loss,  test_accuracy

test_loss, test_accuracy = eval_model(model, test_it)
```

```
27/27 [==============================] - 18s 658ms/step - loss: 0.5691 - accuracy: 0.8441
```

In [40]:

```python
test_it.reset()
preds = model.predict(test_it, steps = 27)
```

In [41]:

```python
y_predict = np.argmax(preds,axis=1)
```

In [42]:

```python
print(classification_report(test_it.classes, y_predict, target_names=test_it.class_indices))
```

```
              precision    recall  f1-score   support

        1977       0.91      0.91      0.91       100
       Amaro       0.72      0.83      0.77       100
     Brannan       0.98      0.95      0.96       100
   Clarendon       0.71      0.70      0.70       100
     Gingham       0.86      0.83      0.84       100
       He-Fe       0.77      0.89      0.83       100
      Hudson       0.84      0.92      0.88       100
       Lo-Fi       0.53      0.71      0.61       100
     Mayfair       0.72      0.77      0.74       100
   Nashville       0.98      0.98      0.98       100
    Original       0.69      0.48      0.56       100
    Perpetua       0.94      0.87      0.90       100
       Sutro       0.98      0.94      0.96       100
     Toaster       1.00      0.92      0.96       100
    Valencia       0.89      0.71      0.79       100
      Willow       0.99      1.00      1.00       100
     X-ProII       0.99      0.94      0.96       100

    accuracy                           0.84      1700
   macro avg       0.85      0.84      0.84      1700
weighted avg       0.85      0.84      0.84      1700
```

In [43]:

```python
error_count = 0

for x in range (0,len(y_predict)):
    if y_predict[x] != test_it.classes[x]:
        error_count += 1

error_count
```

Out[43]:

```
265
```

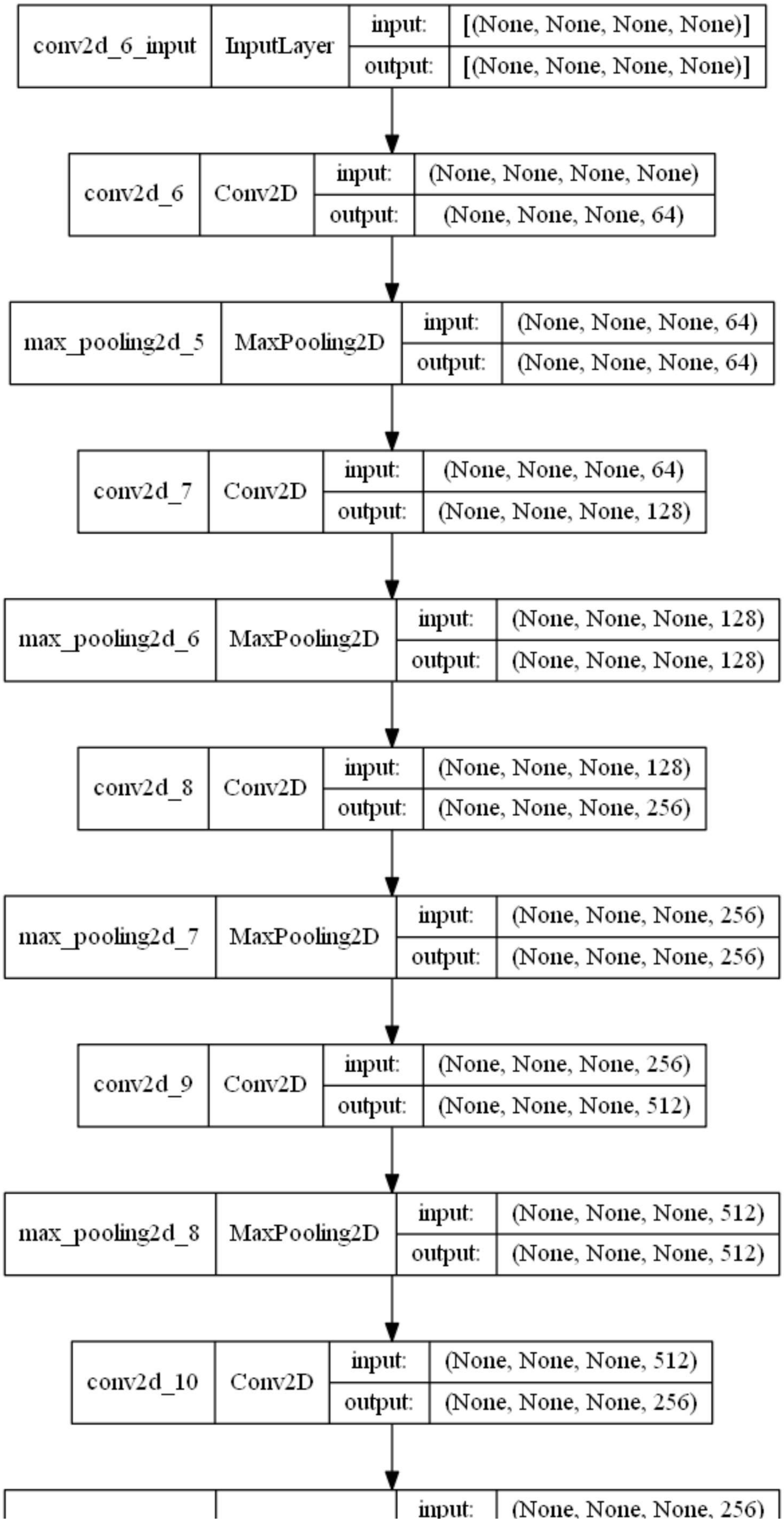
In [45]:

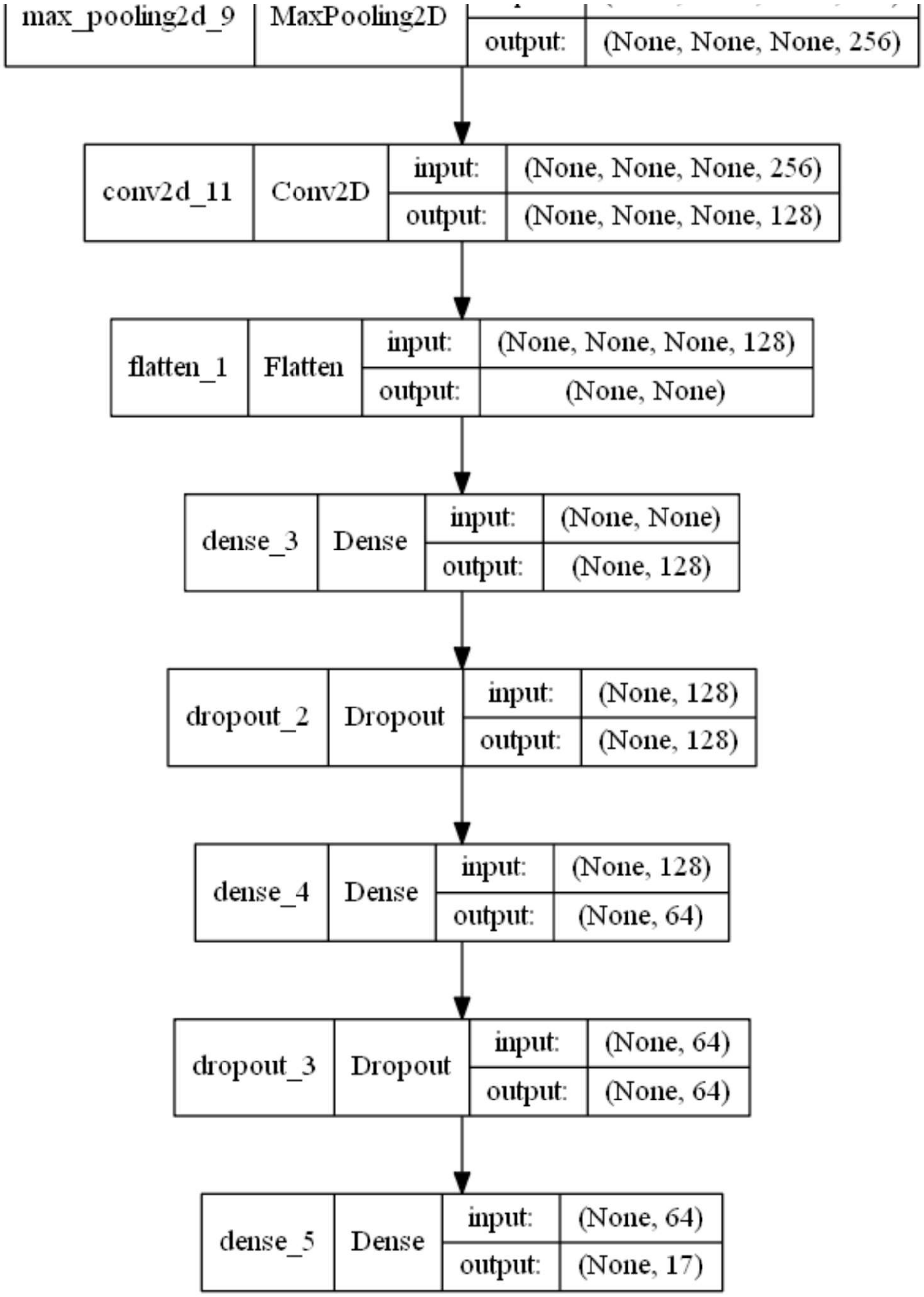
```python
model.save('CNN_Augmented_84_final.h5', overwrite=True,
```

```
    include_optimizer=True)
```

In [33]:
```
plot_model (model, 'model.png', show_shapes=True)
```

Out[33]:




max_pooling2d_9
MaxPooling2D
output:
(None, None, None, 256)
conv2d_11
Conv2D
input:
(None, None, None, 256)
output:
(None, None, None, 128)
flatten_1
Flatten
input:
(None, None, None, 128)
output:
(None, None)
dense_3
Dense
input:
(None, None)
output:
(None, 128)
dropout_2
Dropout
input:
(None, 128)
output:
(None, 128)
dense_4
Dense
input:
(None, 128)
output:
(None, 64)
dropout_3
Dropout
input:
(None, 64)
output:
(None, 64)
dense_5
Dense
input:
(None, 64)
output:
(None, 17)

In [ ]: