Chess Configuration Detection

April 24, 2021

0.1 # Preparing Data

[2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[3]:
```
!ls drive/MyDrive/Chess
```

```
Checkpoint2.h5  train.csv  train.zip  val.csv  val.zip
```

[4]:
```
!unzip drive/MyDrive/Chess/train.zip > /dev/null
!unzip drive/MyDrive/Chess/val.zip > /dev/null
```

[5]:
```
!cp drive/MyDrive/Chess/train.csv ./
!cp drive/MyDrive/Chess/val.csv ./
```

[6]:
```
!pip install chess
```

```
Collecting chess
  Downloading https://files.pythonhosted.org/packages/b6/d4/1dbcc3f009c346
5752e0984fcb6072ab7930cb9de4153b8fb26c1cd01386/chess-1.5.0-py3-none-any.whl
(144kB)
     || 153kB 9.1MB/s
Installing collected packages: chess
Successfully installed chess-1.5.0
```

1 Importing Packages

[7]:
```
import cv2
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf

from tqdm import tqdm
from sklearn.model_selection import train_test_split
```

[8]:
```
train_df = pd.read_csv('train.csv')
train_df
```

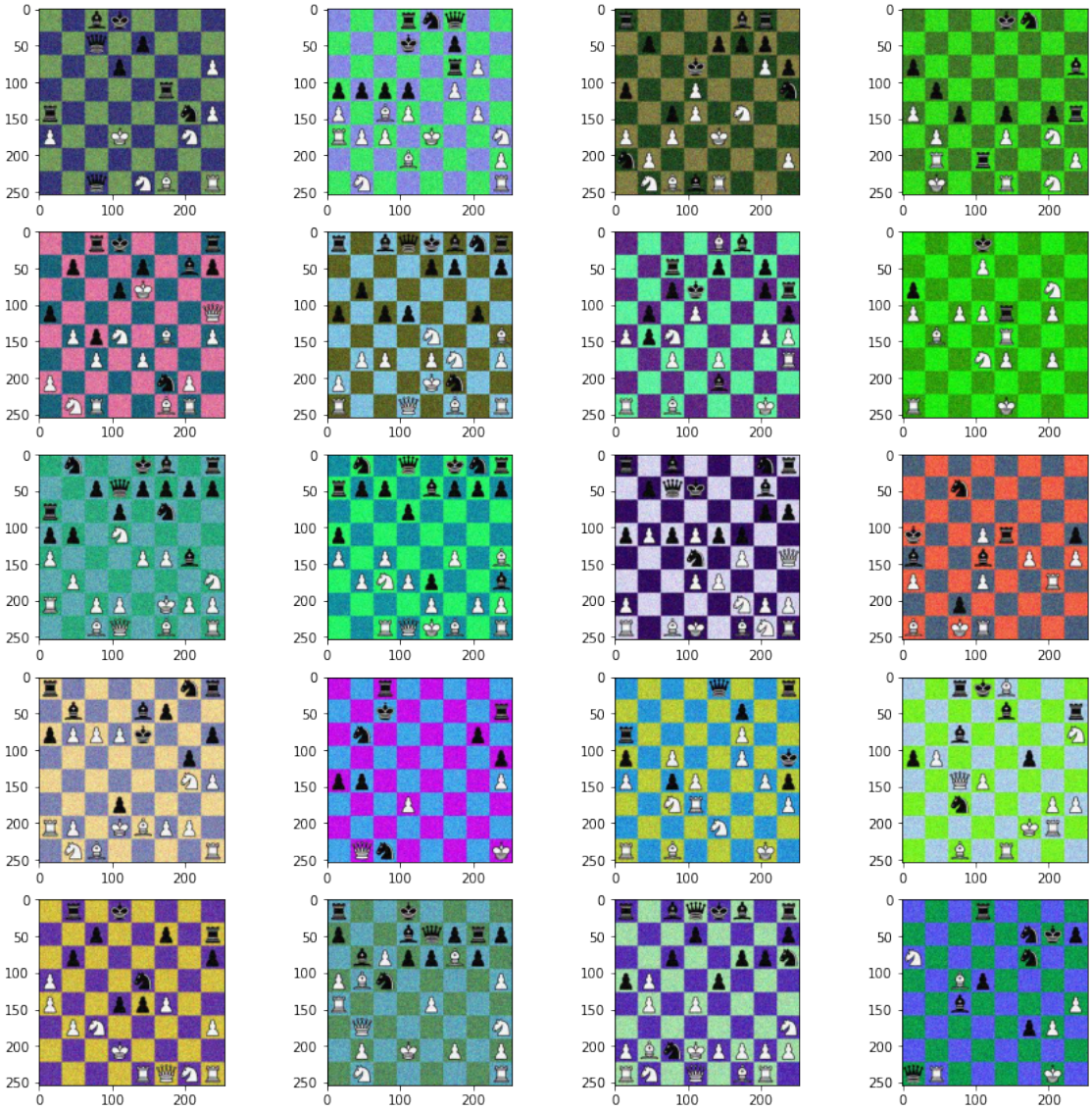
[8]:
```
       ImageID                                              label
0            0  1rbqkb1r/p1p1n1pp/1pn1p3/1P1p1p2/3P4/N3B2P/P1P...
1            1           2bk4/2q1p3/3p3P/5r2/r5nP/P2K2N1/8/2q1NB1R
2            2  3rnq2/3k1p2/5rP1/pppp1P2/P1BP2P1/RPP1K2N/3B3P/...
3            3  r4br1/1p2ppp1/3k2Pp/p2P3n/2pP1N2/P1P1K3/nP5P/1...
4            4     4kn2/8/p6b/1p6/P1p1p1pr/1P2P1N1/1R1r3P/1K2R1N1
...        ...                                                ...
39995    39995                 r7/B6k/2pp1bpP/8/1p6/1P3P2/5N2/5KRB
39996    39996  r5B1/1b2k1b1/4Npp1/p3P3/P2p1P2/1R2P3/2P2KqP/1Q...
39997    39997           6n1/k7/1pP5/1P1pP1n1/p4PBP/R6b/2Kb4/3N2R1
39998    39998  r1b1qk2/3n4/1pp1nb2/p3p1Nr/2p3Pp/PP2P2P/R2PBP1...
39999    39999            1r1n3r/p3k3/P7/2p1P3/1P3p1p/7P/2K2R2/3Q4

[40000 rows x 2 columns]
```

[9]:
```
val_df = pd.read_csv('val.csv')
val_df
```

[9]:
```
      ImageID                                              label
0           0                    7r/2k5/8/8/Pp2P3/RP5r/2R5/K2R1b2
1           1  r1bq1b2/2pppkpr/p6n/5p1p/P1pn1P2/4BNPP/R2KP3/1...
2           2  r1b2br1/p1ppq2p/np1k1pp1/4p1B1/3P2PP/P4P2/1PP1...
3           3  r2r4/pp4bp/4BN1n/1P1pkb2/1n1p2Pp/P1N1Pp2/1BP5/...
4           4  2r3nr/1bppk3/5q1p/p1P1pPp1/1n3P2/1pPK3P/1P2PR2...
...       ...                                                ...
3995     3995              8/1r3pBr/2Pk4/2p4p/5P1R/1P4P1/P4K2/5B2
3996     3996  4rkn1/ppp5/3q4/PN1pppPP/2PP1b2/N1B2r1P/2B3K1/Q...
3997     3997             nk6/5p1r/6Pp/1P1pNK1P/1P3bB1/4p3/3R4/7q
3998     3998  rn3bn1/pp4p1/Pqp1bk2/3pp2r/2B2p2/4P2P/RPPP1PP1...
3999     3999  rnbk2nr/1p4pp/p3pp2/2pp3P/P1P1PNPR/R5b1/1P1P1q...

[4000 rows x 2 columns]
```

1.1 Sample images

[10]:
```
fig=plt.figure(figsize=(15, 15))
columns = 4
rows = 5
for i in range(1, columns*rows +1):
    img = cv2.imread('train/'+str(train_df.iloc[i]['ImageID'])+'.jpg')
    fig.add_subplot(rows, columns, i)
    plt.imshow(img)
plt.show()
```



[11]:
```
train_df.iloc[0].label
```

[11]:
```
'1rbqkb1r/p1p1n1pp/1pn1p3/1P1p1p2/3P4/N3B2P/P1PNPPP1/R2QKB1R'
```

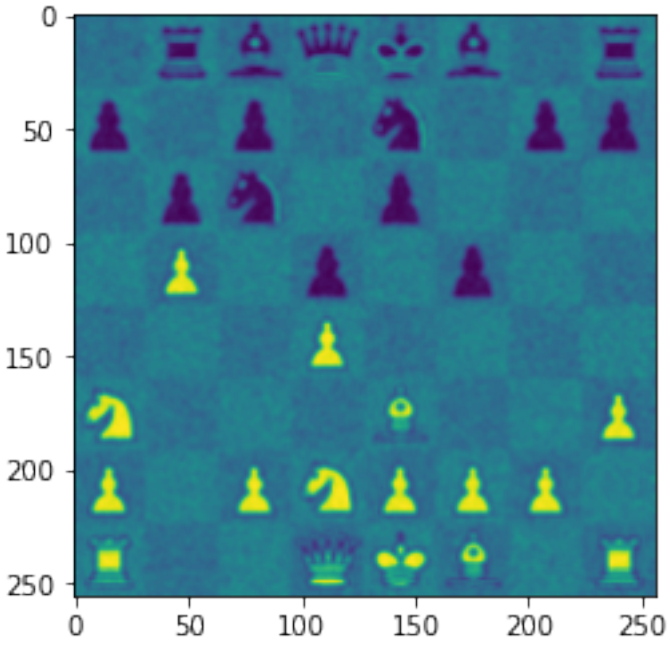
#Parsing FEN notation

[12]:
```
import chess
import numpy as np
```

[13]:
```
img = cv2.imread('train/'+str(train_df.iloc[0]['ImageID'])+'.jpg', cv2.
 ↪IMREAD_GRAYSCALE)
img = cv2.resize(img, (256, 256))
```

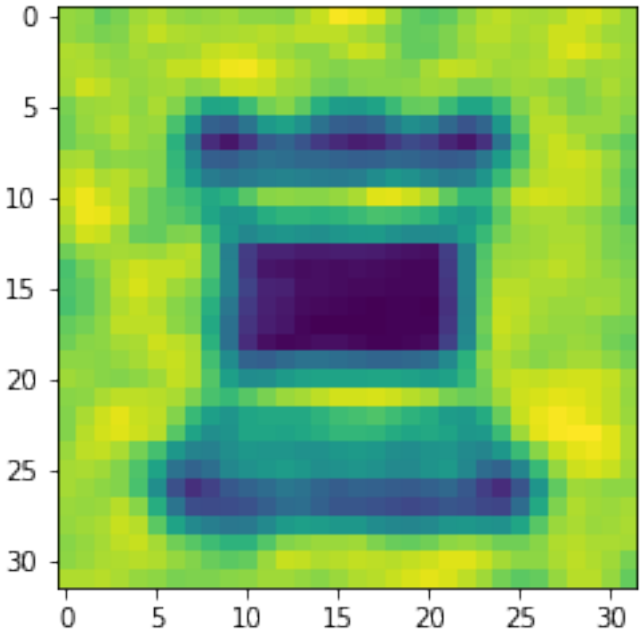
```
img = cv2.GaussianBlur(img,(3,3),cv2.BORDER_DEFAULT)
plt.imshow(img)
```

[13]: <matplotlib.image.AxesImage at 0x7f549f061350>



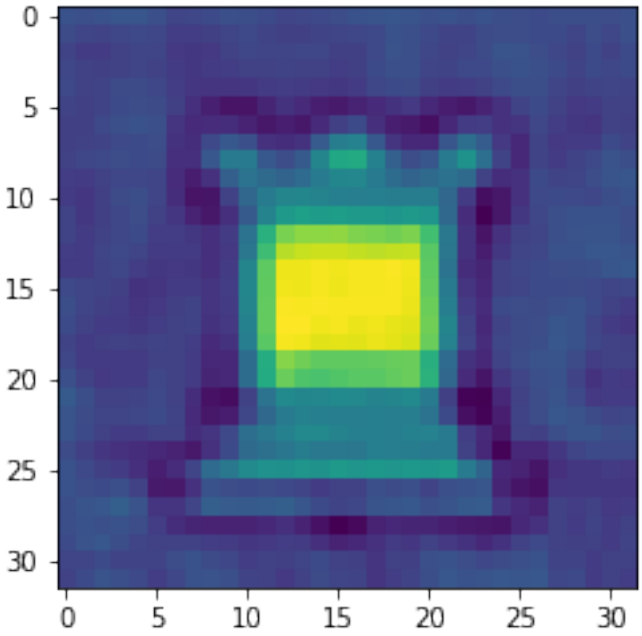
[14]:
```
plt.imshow(img[:32, 32:64])
```

[14]: <matplotlib.image.AxesImage at 0x7f54a1da7910>



[15]:
```
plt.imshow(img[-32:, -32:])
```

[15]: <matplotlib.image.AxesImage at 0x7f54a11f7b90>



[16]:
```
board = chess.Board(train_df.iloc[0].label)
print(board)
```

```
. r b q k b . r
p . p . n . p p
. p n . p . . .
. P . p . p . .
. . . P . . . .
N . . . B . . P
P . P N P P P .
R . . Q K B . R
```

[17]:
```
print(board.piece_at(chess.parse_square('a3')))
```

```
N
```

[18]:
```
for square in chess.SquareSet(chess.BB_ALL):
  print(str(board.piece_at(square)))
```

```
R
None
None
Q
K
B
None
R
P
None
P
N
P
P
P
None
N
None
None
None
B
None
None
P
None
None
None
P
None
```

```
None
None
None
None
P
None
p
None
p
None
None
None
p
n
None
p
None
None
None
p
None
p
None
n
None
p
p
None
r
b
q
k
b
None
r
```

[19]:
```python
def encode_piece(piece: str):
  if piece == 'None':
    return 0
  elif piece == 'K':
    return 1
  elif piece == 'Q':
    return 2
  elif piece == 'R':
    return 3
  elif piece == 'B':
    return 4
```

```
  elif piece == 'N':
    return 5
  elif piece == 'P':
    return 6
  elif piece == 'k':
    return 7
  elif piece == 'q':
    return 8
  elif piece == 'r':
    return 9
  elif piece == 'b':
    return 10
  elif piece == 'n':
    return 11
  else:
    return 12

def preprocess_image(path: str):
  img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
  img = cv2.resize(img ,(256, 256))
  img = cv2.GaussianBlur(img,(3,3),cv2.BORDER_DEFAULT)
  cells = []
  for i in range(256-32, -1, -32):
    for j in range(0, 256-31, 32):
      cells.append(img[i: i+32, j: j+32])
  cells = np.array(cells)
  return cells
```

[20]:
```
preprocess_image('train/'+str(train_df.iloc[0]['ImageID'])+'.jpg').shape
```

[20]:
```
(64, 32, 32)
```

[21]:
```
X = []
y = []

for i in tqdm(range(len(train_df))):
  imgID = str(train_df.iloc[i]['ImageID'])
  img_path = 'train/'+imgID+'.jpg'

  x = preprocess_image(img_path)
  X.extend(x)

  fen = train_df.iloc[i]['label']
  board = chess.Board(fen)
  for square in chess.SquareSet(chess.BB_ALL):
    y.append(encode_piece(str(board.piece_at(square))))
```

```
X = np.array(X)
y = np.array(y)
print('X is: ', X.shape)
print('Y is: ', y.shape)
```

```
100%|| 40000/40000 [01:11<00:00, 563.08it/s]
```

```
X is:  (2560000, 32, 32)
Y is:  (2560000,)
```

[22]:
```
X_test = []
y_test = []

for i in tqdm(range(len(val_df))):
  imgID = str(val_df.iloc[i]['ImageID'])
  img_path = 'val/'+imgID+'.jpg'

  x = preprocess_image(img_path)
  X_test.extend(x)

  fen = val_df.iloc[i]['label']
  board = chess.Board(fen)
  for square in chess.SquareSet(chess.BB_ALL):
    y_test.append(encode_piece(str(board.piece_at(square))))


X_test = np.array(X_test)
y_test = np.array(y_test)
print('X is: ', X_test.shape)
print('Y is: ', y_test.shape)
```

```
100%|| 4000/4000 [00:07<00:00, 566.81it/s]
```

```
X is:  (256000, 32, 32)
Y is:  (256000,)
```

#Undersampling Dataset

[23]:
```
from collections import Counter
from sklearn.datasets import make_classification
from imblearn.under_sampling import RandomUnderSampler
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/externals/six.py:31:
FutureWarning: The module is deprecated in version 0.21 and will be removed in
version 0.23 since we've dropped support for Python 2.7. Please rely on the
official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144:
```

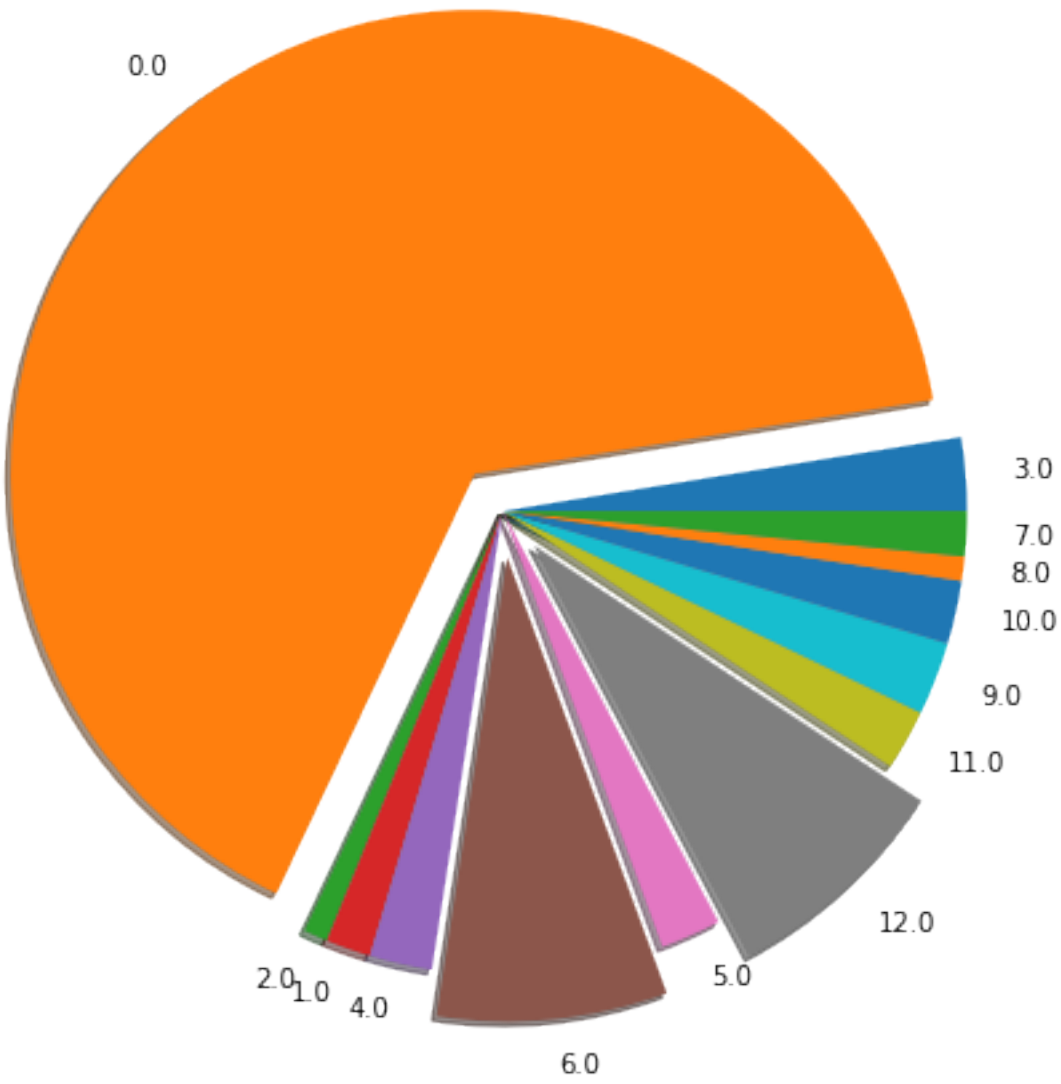
```
FutureWarning: The sklearn.neighbors.base module is  deprecated in version 0.22
and will be removed in version 0.24. The corresponding classes / functions
should instead be imported from sklearn.neighbors. Anything that cannot be
imported from sklearn.neighbors is now part of the private API.
  warnings.warn(message, FutureWarning)
```

[24]:
```
Counter(y)
```

[24]:
```
Counter({0: 1674787,
         1: 40000,
         2: 21592,
         3: 65308,
         4: 55889,
         5: 53949,
         6: 205409,
         7: 40000,
         8: 21698,
         9: 65247,
         10: 56182,
         11: 54384,
         12: 205555})
```

Pie chart of Class Distribution

[25]:
```
counts = Counter(y)
plt.pie([float(v) for v in counts.values()], labels=[float(k) for k in counts],
          autopct=None, radius=2, shadow=True,explode=(0, 0.2, 0, 0, 0, 0.2, 0,␣
 ↪0.2, 0, 0, 0, 0, 0))
plt.show()
```



[26]:
```
X_reshaped = X.reshape(X.shape[0], -1)
print('Reshaped X is: ', X_reshaped.shape)
```

```
Reshaped X is:  (2560000, 1024)
```

[27]:
```
rus = RandomUnderSampler(random_state=42)
X_res, y_res = rus.fit_resample(X_reshaped, y)
print('Resampled dataset shape %s' % Counter(y_res))
X_back = X_res.reshape(X_res.shape[0], 32, 32)
print('X_back is: ', X_back.shape)
print('y is: ', y_res.shape)
```

```
Resampled dataset shape Counter({0: 21592, 1: 21592, 2: 21592, 3: 21592, 4:
21592, 5: 21592, 6: 21592, 7: 21592, 8: 21592, 9: 21592, 10: 21592, 11: 21592,
```

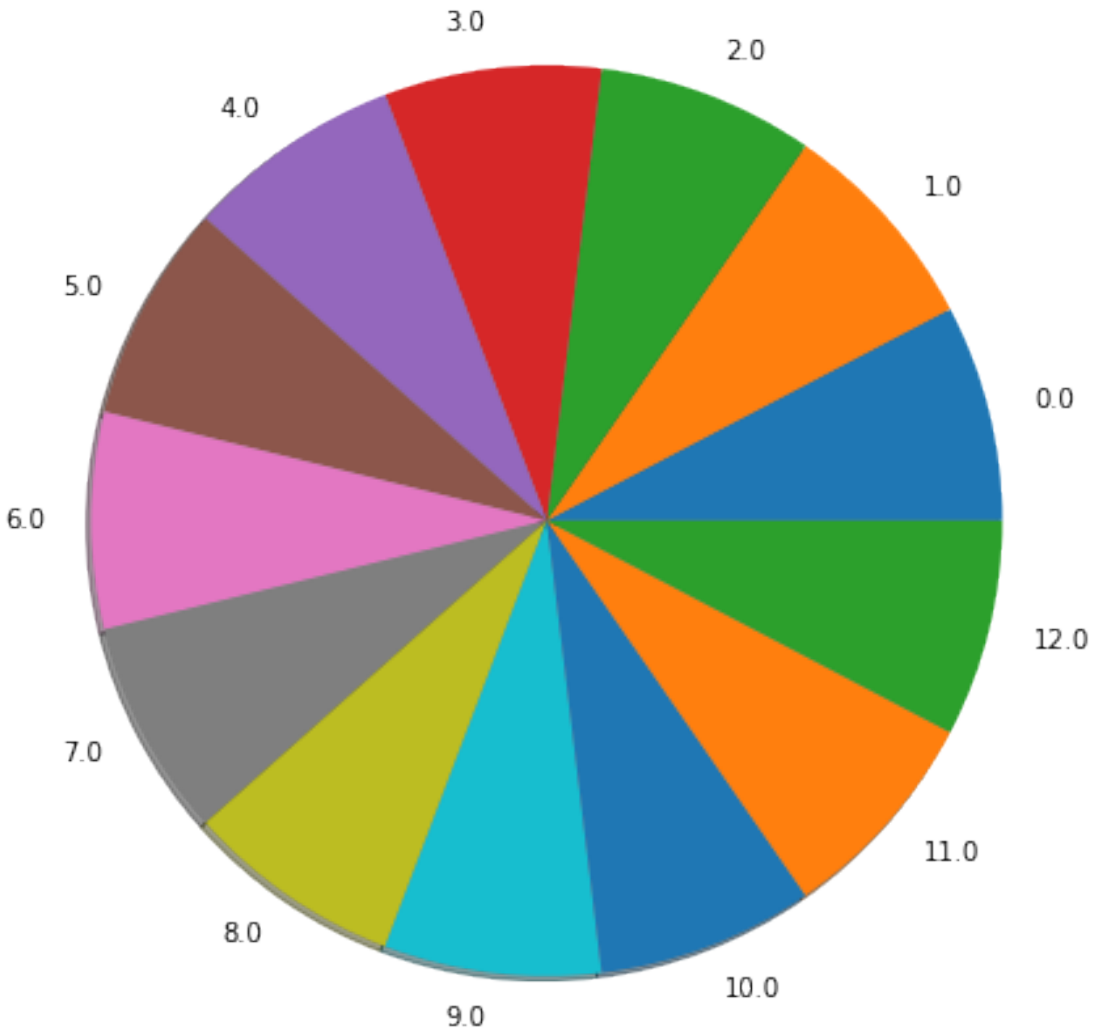
```
12: 21592})
X_back is:  (280696, 32, 32)
y is:  (280696,)

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

[28]:
```
Counter(y_res)
```

[28]:
```
Counter({0: 21592,
         1: 21592,
         2: 21592,
         3: 21592,
         4: 21592,
         5: 21592,
         6: 21592,
         7: 21592,
         8: 21592,
         9: 21592,
         10: 21592,
         11: 21592,
         12: 21592})
```

Pie Chart after under sampling the dataset

[29]:
```
counts = Counter(y_res)
plt.pie([float(v) for v in counts.values()], labels=[float(k) for k in counts],
          autopct=None, radius=2, shadow=True,)
plt.show()
```



[30]:
```
from keras.utils.np_utils import to_categorical
```

[31]:
```
y_res_one_hot = to_categorical(y_res, num_classes=13)
print('Shape of [y_res_one_hot]: ', y_res_one_hot.shape)
y_test_one_hot=to_categorical(y_test,num_classes=13)
print('Shape of [y_test_one_hot]: ', y_test_one_hot.shape)
```

```
Shape of [y_res_one_hot]:  (280696, 13)
Shape of [y_test_one_hot]:  (256000, 13)
```

[32]:
```
X = X.reshape(X.shape[0], 32,32, 1)
```

[33]:
```
y_one_hot = to_categorical(y, num_classes=13)
```

[34]:
```
X_back=X_res.reshape(X_res.shape[0],32,32,1)
X_test=X_test.reshape(X_test.shape[0],32,32,1)
```

[35]:
```
print('Shape of [X_back] is: ', X_back.shape)
```

```
Shape of [X_back] is:  (280696, 32, 32, 1)
```

2 Model

[42]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D,Flatten
```

[43]:
```
from tensorflow.keras.callbacks import EarlyStopping
es = EarlyStopping(monitor='val_loss', patience=3)
```

2.1 Training model

[44]:
```
model=Sequential()
model.add(Conv2D(16, (3,3), padding='same',␣
 ↪activation="relu",input_shape=(32,32,1)))
model.add(MaxPool2D(2,2))
model.add(Conv2D(32, (3,3), padding='same', activation="relu"))
model.add(MaxPool2D(2,2))
model.add(Conv2D(64, (3,3), padding='same', activation="relu"))
model.add(MaxPool2D(2,2))
model.add(Flatten())
model.add(Dense(13,activation='softmax'))
```

[45]:
```
from tensorflow.keras.optimizers import Adam
```

[46]:
```
optimizer=Adam(lr=1e-6)
model.compile(optimizer=optimizer, loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[47]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 32, 32, 16)        160
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 32)        4640
_________________________________________________________________
```

```
max_pooling2d_4 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 64)          18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 13)                13325
=================================================================
Total params: 36,621
Trainable params: 36,621
Non-trainable params: 0
_________________________________________________________________
```

[48]:
```
history = model.fit(X_back,␣
↪y_res_one_hot,validation_data=(X_test,y_test_one_hot), epochs=50,␣
↪batch_size=1024, callbacks=[es])
```

```
Epoch 1/50
275/275 [==============================] - 35s 16ms/step - loss: 25.4085 -
accuracy: 0.0699 - val_loss: 31.8399 - val_accuracy: 0.0181
Epoch 2/50
275/275 [==============================] - 4s 14ms/step - loss: 18.5913 -
accuracy: 0.1264 - val_loss: 25.1880 - val_accuracy: 0.0231
Epoch 3/50
275/275 [==============================] - 4s 14ms/step - loss: 13.8911 -
accuracy: 0.1489 - val_loss: 19.2240 - val_accuracy: 0.0259
Epoch 4/50
275/275 [==============================] - 4s 14ms/step - loss: 10.2435 -
accuracy: 0.1613 - val_loss: 14.2769 - val_accuracy: 0.0323
Epoch 5/50
275/275 [==============================] - 4s 14ms/step - loss: 7.6708 -
accuracy: 0.1816 - val_loss: 9.5259 - val_accuracy: 0.0381
Epoch 6/50
275/275 [==============================] - 4s 14ms/step - loss: 5.3936 -
accuracy: 0.1967 - val_loss: 5.4462 - val_accuracy: 0.0497
Epoch 7/50
275/275 [==============================] - 4s 14ms/step - loss: 3.7391 -
accuracy: 0.2132 - val_loss: 3.4406 - val_accuracy: 0.0785
Epoch 8/50
275/275 [==============================] - 4s 14ms/step - loss: 2.7780 -
accuracy: 0.2838 - val_loss: 2.8170 - val_accuracy: 0.1474
Epoch 9/50
275/275 [==============================] - 4s 14ms/step - loss: 2.2198 -
accuracy: 0.3548 - val_loss: 2.3788 - val_accuracy: 0.1763
Epoch 10/50
275/275 [==============================] - 4s 14ms/step - loss: 1.8202 -
```

```
accuracy: 0.4122 - val_loss: 2.0348 - val_accuracy: 0.2348
Epoch 11/50
275/275 [==============================] - 4s 14ms/step - loss: 1.5122 -
accuracy: 0.4918 - val_loss: 1.7212 - val_accuracy: 0.3474
Epoch 12/50
275/275 [==============================] - 4s 14ms/step - loss: 1.2496 -
accuracy: 0.5860 - val_loss: 1.4455 - val_accuracy: 0.5021
Epoch 13/50
275/275 [==============================] - 4s 14ms/step - loss: 1.0329 -
accuracy: 0.6814 - val_loss: 1.2243 - val_accuracy: 0.6254
Epoch 14/50
275/275 [==============================] - 4s 14ms/step - loss: 0.8535 -
accuracy: 0.7632 - val_loss: 1.0308 - val_accuracy: 0.7454
Epoch 15/50
275/275 [==============================] - 4s 14ms/step - loss: 0.7086 -
accuracy: 0.8264 - val_loss: 0.8828 - val_accuracy: 0.8275
Epoch 16/50
275/275 [==============================] - 4s 14ms/step - loss: 0.5893 -
accuracy: 0.8726 - val_loss: 0.7594 - val_accuracy: 0.8779
Epoch 17/50
275/275 [==============================] - 4s 14ms/step - loss: 0.4936 -
accuracy: 0.9027 - val_loss: 0.6657 - val_accuracy: 0.9044
Epoch 18/50
275/275 [==============================] - 4s 14ms/step - loss: 0.4164 -
accuracy: 0.9232 - val_loss: 0.5924 - val_accuracy: 0.9208
Epoch 19/50
275/275 [==============================] - 4s 14ms/step - loss: 0.3530 -
accuracy: 0.9398 - val_loss: 0.5226 - val_accuracy: 0.9339
Epoch 20/50
275/275 [==============================] - 4s 14ms/step - loss: 0.2998 -
accuracy: 0.9523 - val_loss: 0.4690 - val_accuracy: 0.9430
Epoch 21/50
275/275 [==============================] - 4s 14ms/step - loss: 0.2589 -
accuracy: 0.9611 - val_loss: 0.4193 - val_accuracy: 0.9508
Epoch 22/50
275/275 [==============================] - 4s 14ms/step - loss: 0.2229 -
accuracy: 0.9686 - val_loss: 0.3759 - val_accuracy: 0.9576
Epoch 23/50
275/275 [==============================] - 4s 14ms/step - loss: 0.1932 -
accuracy: 0.9737 - val_loss: 0.3326 - val_accuracy: 0.9642
Epoch 24/50
275/275 [==============================] - 4s 14ms/step - loss: 0.1655 -
accuracy: 0.9798 - val_loss: 0.2958 - val_accuracy: 0.9699
Epoch 25/50
275/275 [==============================] - 4s 14ms/step - loss: 0.1425 -
accuracy: 0.9845 - val_loss: 0.2642 - val_accuracy: 0.9744
Epoch 26/50
275/275 [==============================] - 4s 14ms/step - loss: 0.1222 -
```

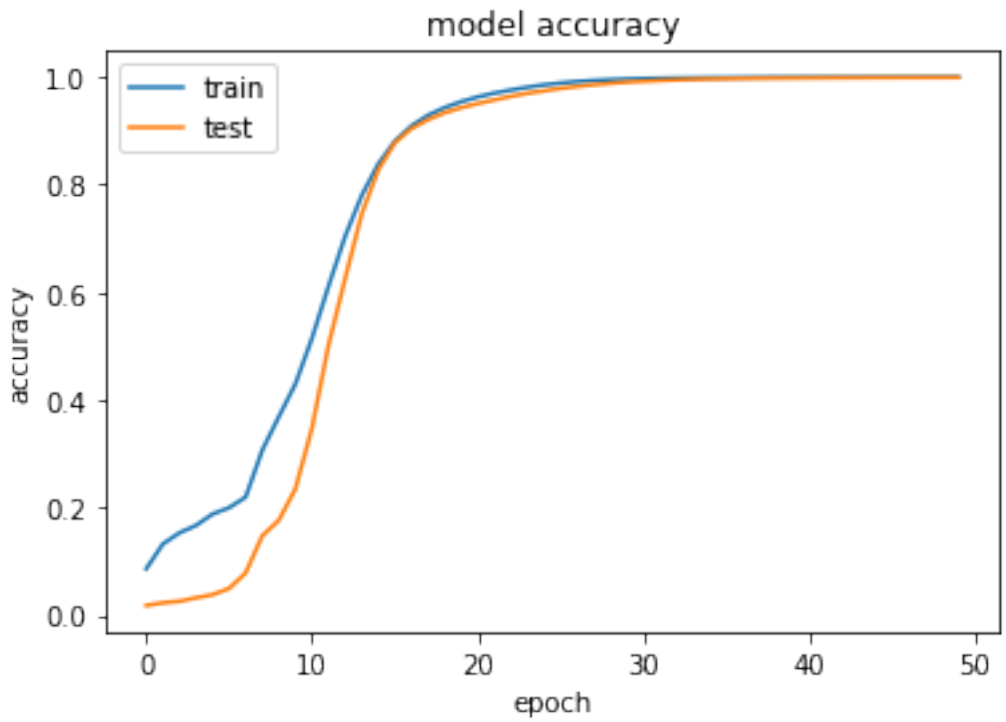
```
accuracy: 0.9882 - val_loss: 0.2361 - val_accuracy: 0.9790
Epoch 27/50
275/275 [==============================] - 4s 14ms/step - loss: 0.1045 -
accuracy: 0.9913 - val_loss: 0.2129 - val_accuracy: 0.9826
Epoch 28/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0908 -
accuracy: 0.9935 - val_loss: 0.1876 - val_accuracy: 0.9858
Epoch 29/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0779 -
accuracy: 0.9951 - val_loss: 0.1682 - val_accuracy: 0.9885
Epoch 30/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0668 -
accuracy: 0.9962 - val_loss: 0.1475 - val_accuracy: 0.9907
Epoch 31/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0580 -
accuracy: 0.9969 - val_loss: 0.1345 - val_accuracy: 0.9924
Epoch 32/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0503 -
accuracy: 0.9977 - val_loss: 0.1174 - val_accuracy: 0.9937
Epoch 33/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0436 -
accuracy: 0.9983 - val_loss: 0.1041 - val_accuracy: 0.9950
Epoch 34/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0384 -
accuracy: 0.9986 - val_loss: 0.0931 - val_accuracy: 0.9959
Epoch 35/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0331 -
accuracy: 0.9989 - val_loss: 0.0818 - val_accuracy: 0.9965
Epoch 36/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0291 -
accuracy: 0.9990 - val_loss: 0.0754 - val_accuracy: 0.9970
Epoch 37/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0252 -
accuracy: 0.9993 - val_loss: 0.0673 - val_accuracy: 0.9975
Epoch 38/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0221 -
accuracy: 0.9994 - val_loss: 0.0605 - val_accuracy: 0.9979
Epoch 39/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0198 -
accuracy: 0.9995 - val_loss: 0.0548 - val_accuracy: 0.9981
Epoch 40/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0175 -
accuracy: 0.9996 - val_loss: 0.0496 - val_accuracy: 0.9983
Epoch 41/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0153 -
accuracy: 0.9996 - val_loss: 0.0439 - val_accuracy: 0.9986
Epoch 42/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0134 -
```

```
accuracy: 0.9997 - val_loss: 0.0405 - val_accuracy: 0.9987
Epoch 43/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0121 -
accuracy: 0.9997 - val_loss: 0.0372 - val_accuracy: 0.9988
Epoch 44/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0108 -
accuracy: 0.9997 - val_loss: 0.0331 - val_accuracy: 0.9989
Epoch 45/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0095 -
accuracy: 0.9998 - val_loss: 0.0301 - val_accuracy: 0.9991
Epoch 46/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0087 -
accuracy: 0.9998 - val_loss: 0.0280 - val_accuracy: 0.9991
Epoch 47/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0076 -
accuracy: 0.9998 - val_loss: 0.0255 - val_accuracy: 0.9992
Epoch 48/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0069 -
accuracy: 0.9998 - val_loss: 0.0229 - val_accuracy: 0.9992
Epoch 49/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0062 -
accuracy: 0.9998 - val_loss: 0.0210 - val_accuracy: 0.9993
Epoch 50/50
275/275 [==============================] - 4s 14ms/step - loss: 0.0055 -
accuracy: 0.9999 - val_loss: 0.0193 - val_accuracy: 0.9994
```

2.1.1 Plotting model learning characteristics

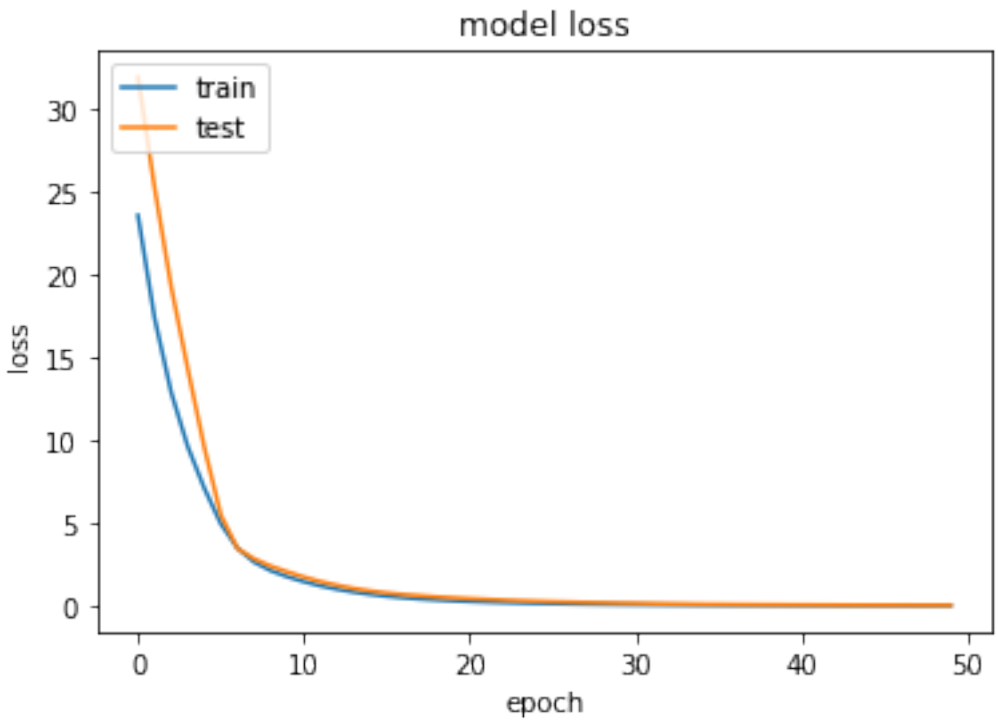
Accuracy Plot

[49]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Loss Plot

[50]:
```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Classification Report

[56]:
```
y_pred=model.predict(X_test)
```

[57]:
```
y_pred=np.argmax(y_pred,axis=1)
```

[58]:
```
y_pred.shape
```

[58]:
```
(256000,)
```

[80]:
```
from sklearn.metrics import classification_report, confusion_matrix
print("Classification report:\n",␣
 ↪classification_report(y_pred=y_pred,y_true=y_test))
```

```
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    167461
           1       1.00      1.00      1.00      4000
           2       1.00      1.00      1.00      2163
           3       1.00      1.00      1.00      6530
           4       1.00      1.00      1.00      5584
           5       1.00      1.00      1.00      5371
           6       1.00      1.00      1.00     20501
```

```
           7       1.00      1.00      1.00      4000
           8       1.00      1.00      1.00      2160
           9       1.00      1.00      1.00      6614
          10       1.00      1.00      1.00      5601
          11       1.00      1.00      1.00      5429
          12       0.99      1.00      1.00     20586

    accuracy                           1.00    256000
   macro avg       1.00      1.00      1.00    256000
weighted avg       1.00      1.00      1.00    256000
```

[60]:
```
model.save('/content/drive/MyDrive/Chess/Checkpoint_Final.h5')
```

3 Generating FEN on test set

3.1 Loading saved model from checkpoint

[61]:
```
model = tf.keras.models.load_model('/content/drive/MyDrive/Chess/
↪Checkpoint_Final.h5')
```

[62]:
```
def decode_piece(piece: int):
  if piece == 0:
    return None
  elif piece == 1:
    return 'K'
  elif piece == 2:
    return 'Q'
  elif piece == 3:
    return 'R'
  elif piece == 4:
    return 'B'
  elif piece == 5:
    return 'N'
  elif piece == 6:
    return 'P'
  elif piece == 7:
    return 'k'
  elif piece == 8:
    return 'q'
  elif piece == 9:
    return 'r'
  elif piece == 10:
    return 'b'
  elif piece == 11:
    return 'n'
```

```
  else:
    return 'p'
```

3.2 Loading Test Data

[63]:
```
X_val = []
y_val = []

for i in tqdm(range(len(val_df))):
  imgID = str(val_df.iloc[i]['ImageID'])
  img_path = 'val/'+imgID+'.jpg'

  x = preprocess_image(img_path)
  X_val.append(x)

  fen = val_df.iloc[i]['label']
  # board = chess.Board(fen)
  # for square in chess.SquareSet(chess.BB_ALL):
  #   y.append(encode_piece(str(board.piece_at(square))))
  y_val.append(fen)

X_val = np.array(X_val)
# y = np.array(y)
print('X is: ', X_val.shape)
print('Y is: ', len(y_val))
```

```
100%|| 4000/4000 [00:05<00:00, 682.06it/s]
```

```
X is:  (4000, 64, 32, 32)
Y is:  4000
```

[64]:
```
def get_fen_image(cells):
  global model
  cells = cells.reshape(-1, 32, 32, 1)
  y_pred = model.predict(cells)
  y_pred = np.argmax(y_pred, axis=1)
  pieces = []
  board = chess.Board()
  board.clear_board()
  for i in y_pred:
    pieces.append(decode_piece(i))
  for counter, square in enumerate(chess.SquareSet(chess.BB_ALL)):
      if pieces[counter] is not None:
        piece = chess.Piece.from_symbol(pieces[counter])
        board.set_piece_at(square, piece)
  return board.fen().split(' ')[0]
```

[70]:
```
predicted_fen_array = []
for example in tqdm(X_val):
  predicted_fen = get_fen_image(example)
  predicted_fen_array.append(predicted_fen)
```

```
100%|| 4000/4000 [01:53<00:00, 35.29it/s]
```

[71]:
```
val_df['PredictedFEN'] = predicted_fen_array
```

[71]:
```

```

[72]:
```
val_df
```

[72]:

	ImageID	...	PredictedFEN
0	0	...	7r/2k5/8/8/Pp2P3/RP5r/2R5/K2R1b2
1	1	...	r1bq1b2/2pppkpr/p6n/5p1p/P1pn1P2/4BNPP/R2KP3/1...
2	2	...	r1b2br1/p1ppq2p/np1k1pp1/4p1B1/3P2PP/P4P2/1PP1...
3	3	...	r2r4/pp4bp/4BN1n/1P1pkb2/1n1p2Pp/P1N1Pp2/1BP5/...
4	4	...	2r3nr/1bppk3/5q1p/p1P1pPp1/1n3P2/1pPK3P/1P2PR2...
...	...	...	...
3995	3995	...	8/1r3pBr/2Pk4/2p4p/5P1R/1P4P1/P4K2/5B2
3996	3996	...	4rkn1/ppp5/3q4/PN1pppPP/2PP1b2/N1B2r1P/2B3K1/Q...
3997	3997	...	nk6/5p1r/6Pp/1P1pNK1P/1P3bB1/4p3/3R4/7q
3998	3998	...	rn3bn1/pp4p1/Pqp1bk2/3pp2r/2B2p2/4P2P/RPPP1PP1...
3999	3999	...	rnbk2nr/1p4pp/p3pp2/2pp3P/P1P1PNPR/R5b1/1P1P1q...

```
[4000 rows x 3 columns]
```

[73]:
```
!pip install jiwer
```

```
Requirement already satisfied: jiwer in /usr/local/lib/python3.7/dist-packages
(2.2.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(from jiwer) (1.19.5)
Requirement already satisfied: python-Levenshtein in
/usr/local/lib/python3.7/dist-packages (from jiwer) (0.12.2)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-
packages (from python-Levenshtein->jiwer) (54.2.0)
```

[74]:
```
import jiwer
```

[75]:
```
print("Word Error Rate[WER]: ", jiwer.wer(list(val_df['label']),␣
 ↪list(val_df['PredictedFEN'])))
```

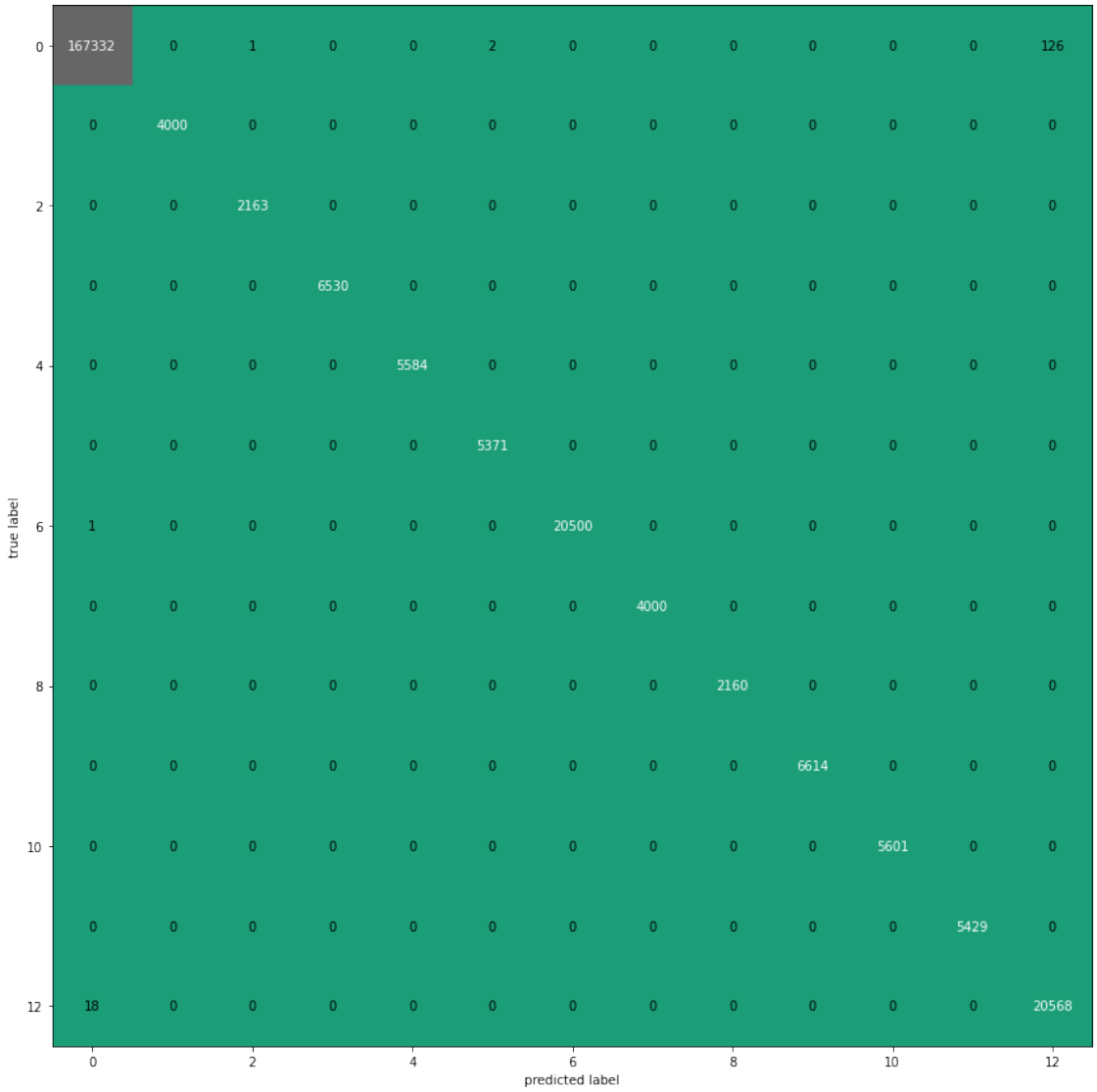
```
Word Error Rate[WER]:  0.01425
```

[ ]:
```
# String1: Sky is blue today
# Strinb2: SKy am red today
```

4 Confusion matrix

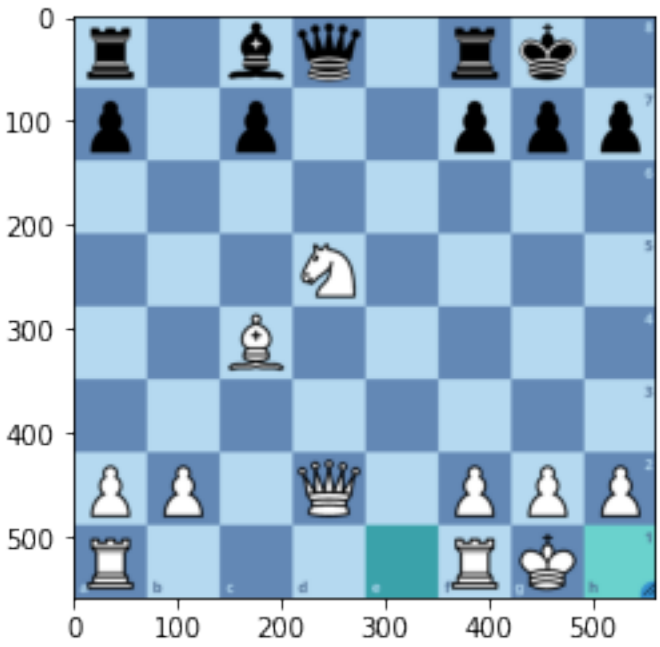
[81]:
```
#confusion matrix
cm = confusion_matrix(y_true=y_test, y_pred=y_pred,␣
↪labels=[0,1,2,3,4,5,6,7,8,9,10,11,12], sample_weight=None, normalize=None)
```

[82]:
```
#confusion matrix plot
from mlxtend.plotting import plot_confusion_matrix

plot_confusion_matrix(cm, figsize=(15,15), cmap=plt.cm.Dark2 )

plt.show()
```



5 Testing with sample image not in either dataset

[76]:
```
# Image generated usign lichess
img = cv2.imread('/content/drive/MyDrive/Chess/test.png', cv2.IMREAD_COLOR)
plt.imshow(img)
plt.show()
```



[77]:
```
print('FEN is: r1bq1rk1/p1p2ppp/8/3N4/2B5/8/PP1Q1PPP/R4RK1')
```

```
FEN is: r1bq1rk1/p1p2ppp/8/3N4/2B5/8/PP1Q1PPP/R4RK1
```

[78]:
```
data = preprocess_image('/content/drive/MyDrive/Chess/test.png')
print("Predicted FEN: ", get_fen_image(data))
```

```
Predicted FEN:  r1bq1rk1/p1p2ppp/8/3N4/2B5/8/PP1Q1PPP/R4RK1
```

[ ]:
```

```