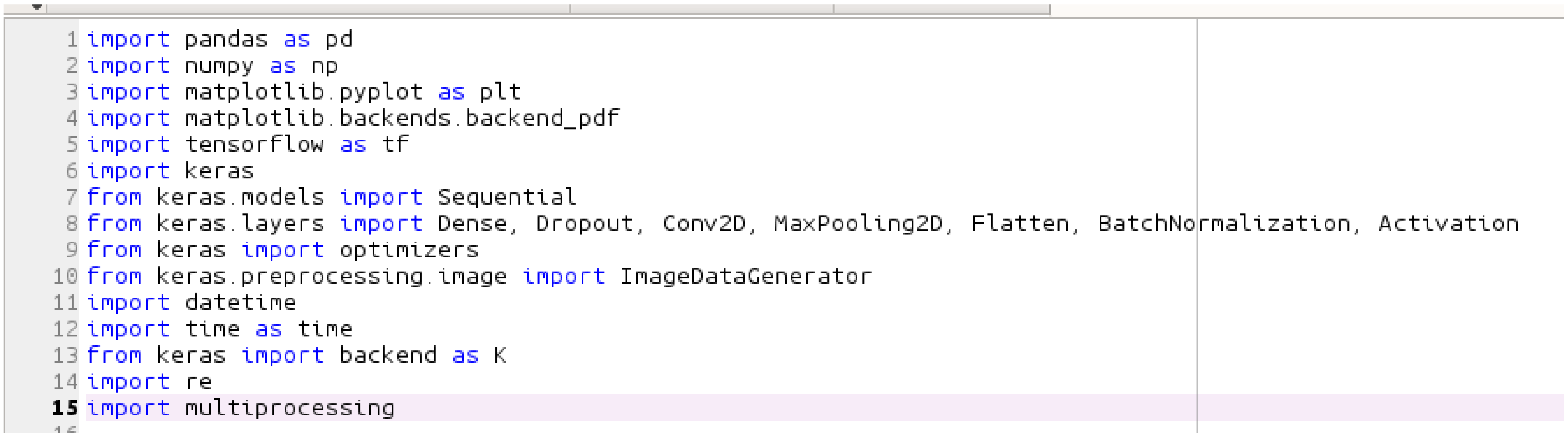
# Predict the Characters in The Simposons

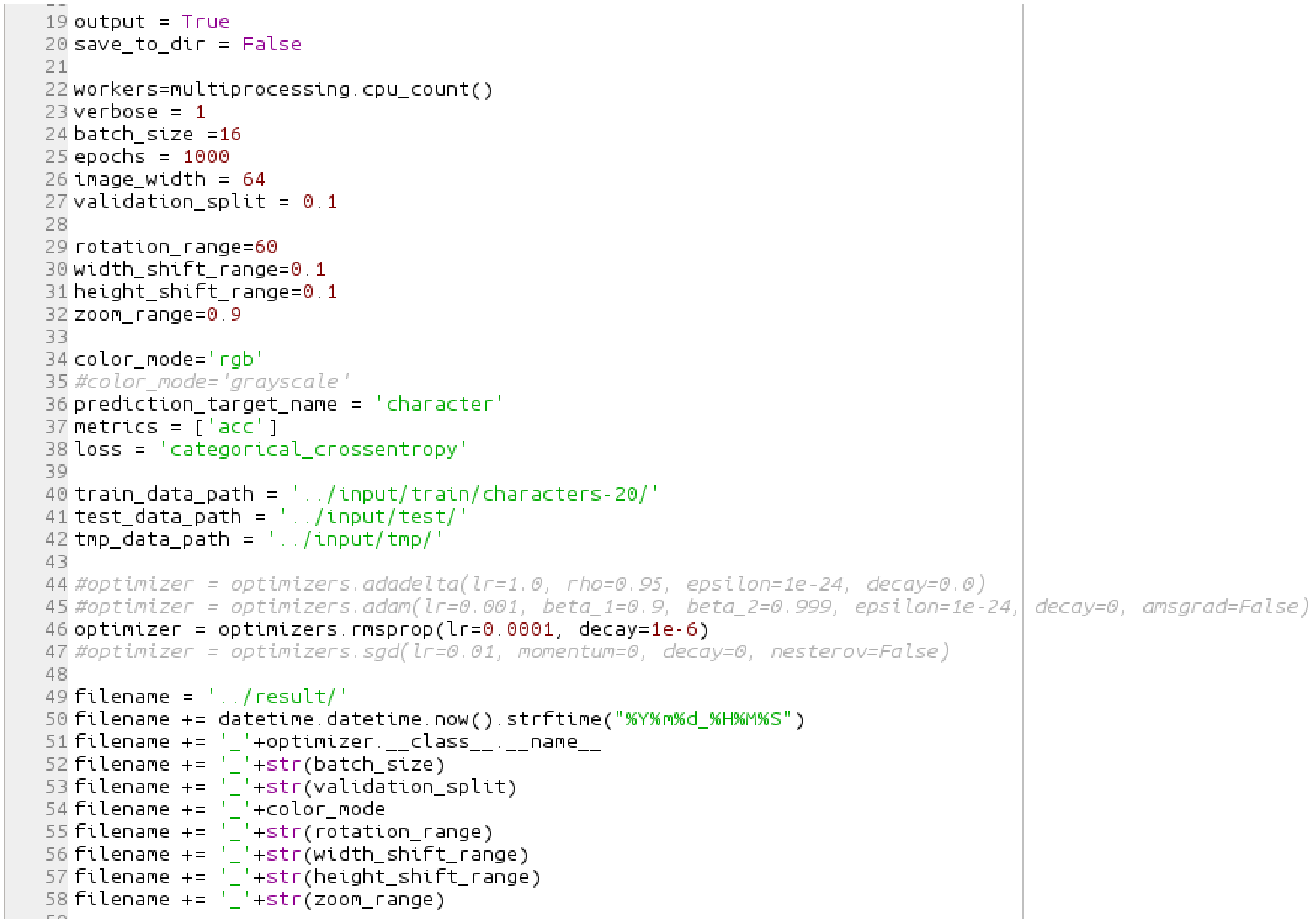
電子三甲  105360046  蕭賢傑

# import



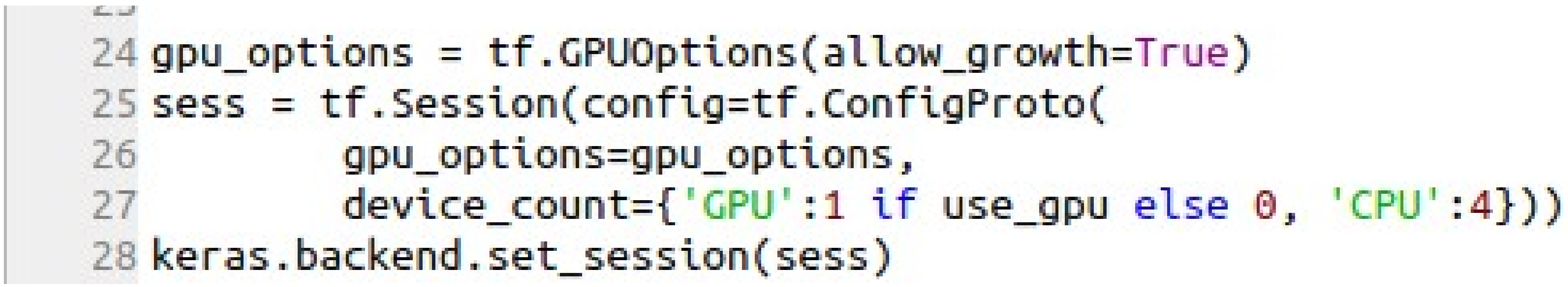
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.backends.backend_pdf
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten, BatchNormalization, Activation
from keras import optimizers
from keras.preprocessing.image import ImageDataGenerator
import datetime
import time as time
from keras import backend as K
import re
import multiprocessing
```

# Set parameter



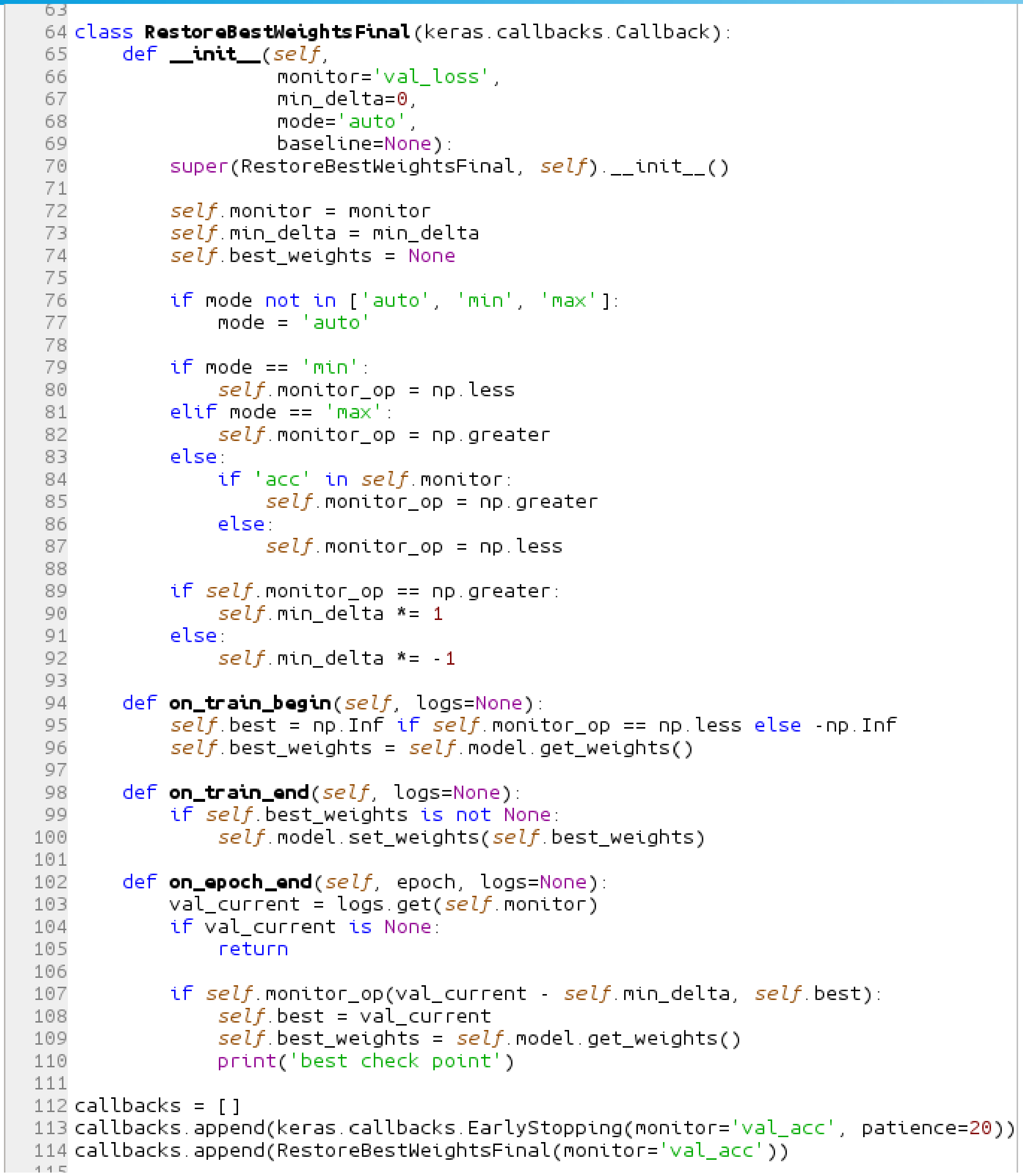
```python
output = True
save_to_dir = False

workers=multiprocessing.cpu_count()
verbose = 1
batch_size =16
epochs = 1000
image_width = 64
validation_split = 0.1

rotation_range=60
width_shift_range=0.1
height_shift_range=0.1
zoom_range=0.9

color_mode='rgb'
#color_mode='grayscale'
prediction_target_name = 'character'
metrics = ['acc']
loss = 'categorical_crossentropy'

train_data_path = '../input/train/characters-20/'
test_data_path = '../input/test/'
tmp_data_path = '../input/tmp/'

#optimizer = optimizers.adadelta(lr=1.0, rho=0.95, epsilon=1e-24, decay=0.0)
#optimizer = optimizers.adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-24, decay=0, amsgrad=False)
optimizer = optimizers.rmsprop(lr=0.0001, decay=1e-6)
#optimizer = optimizers.sgd(lr=0.01, momentum=0, decay=0, nesterov=False)

filename = '../result/'
filename += datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
filename += '_'+optimizer.__class__.__name__
filename += '_'+str(batch_size)
filename += '_'+str(validation_split)
filename += '_'+color_mode
filename += '_'+str(rotation_range)
filename += '_'+str(width_shift_range)
filename += '_'+str(height_shift_range)
filename += '_'+str(zoom_range)
```

## Setup device



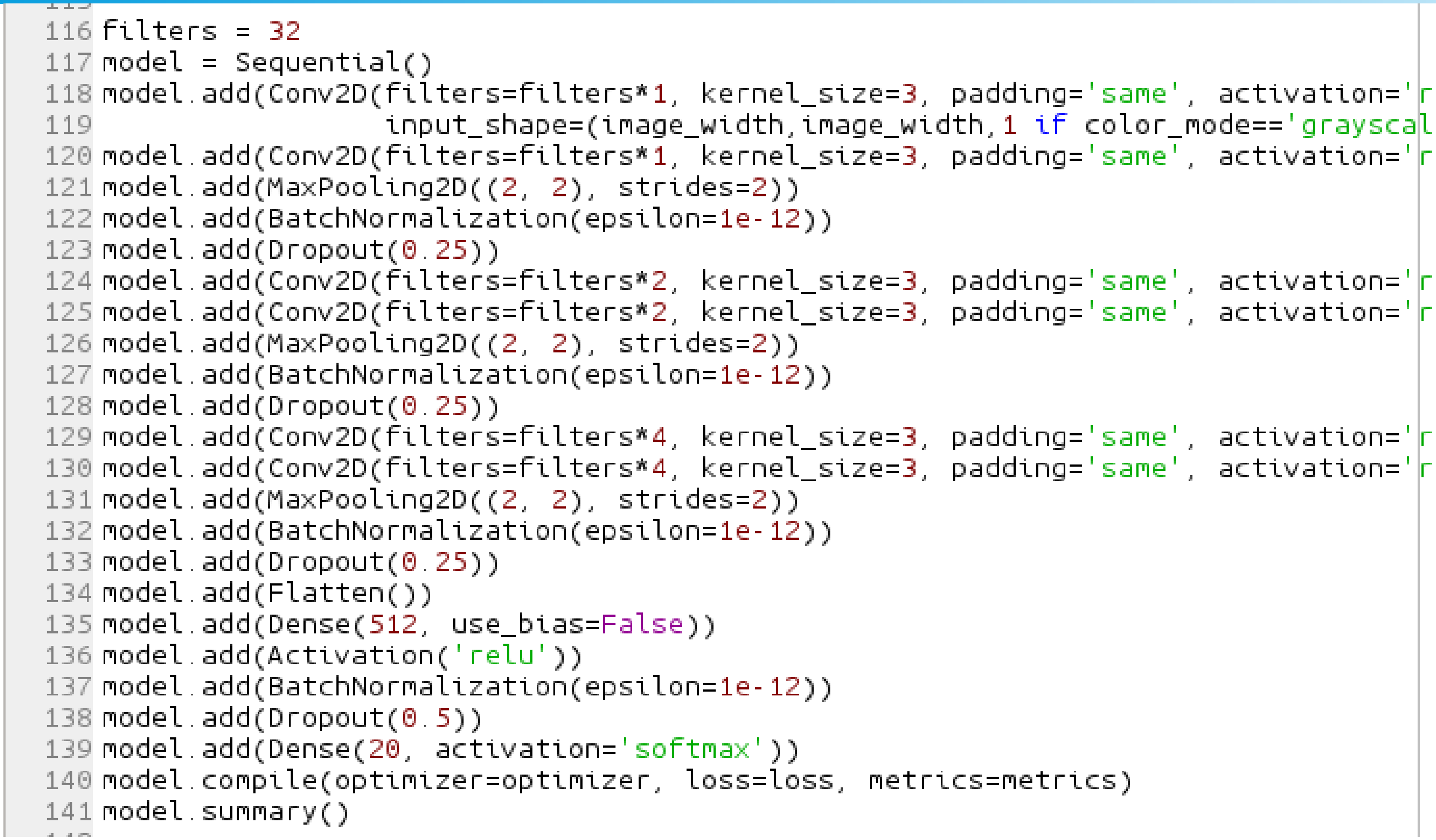
```
gpu_options = tf.GPUOptions(allow_growth=True)
sess = tf.Session(config=tf.ConfigProto(
        gpu_options=gpu_options,
        device_count={'GPU':1 if use_gpu else 0, 'CPU':4}))
keras.backend.set_session(sess)
```

# Callbacks



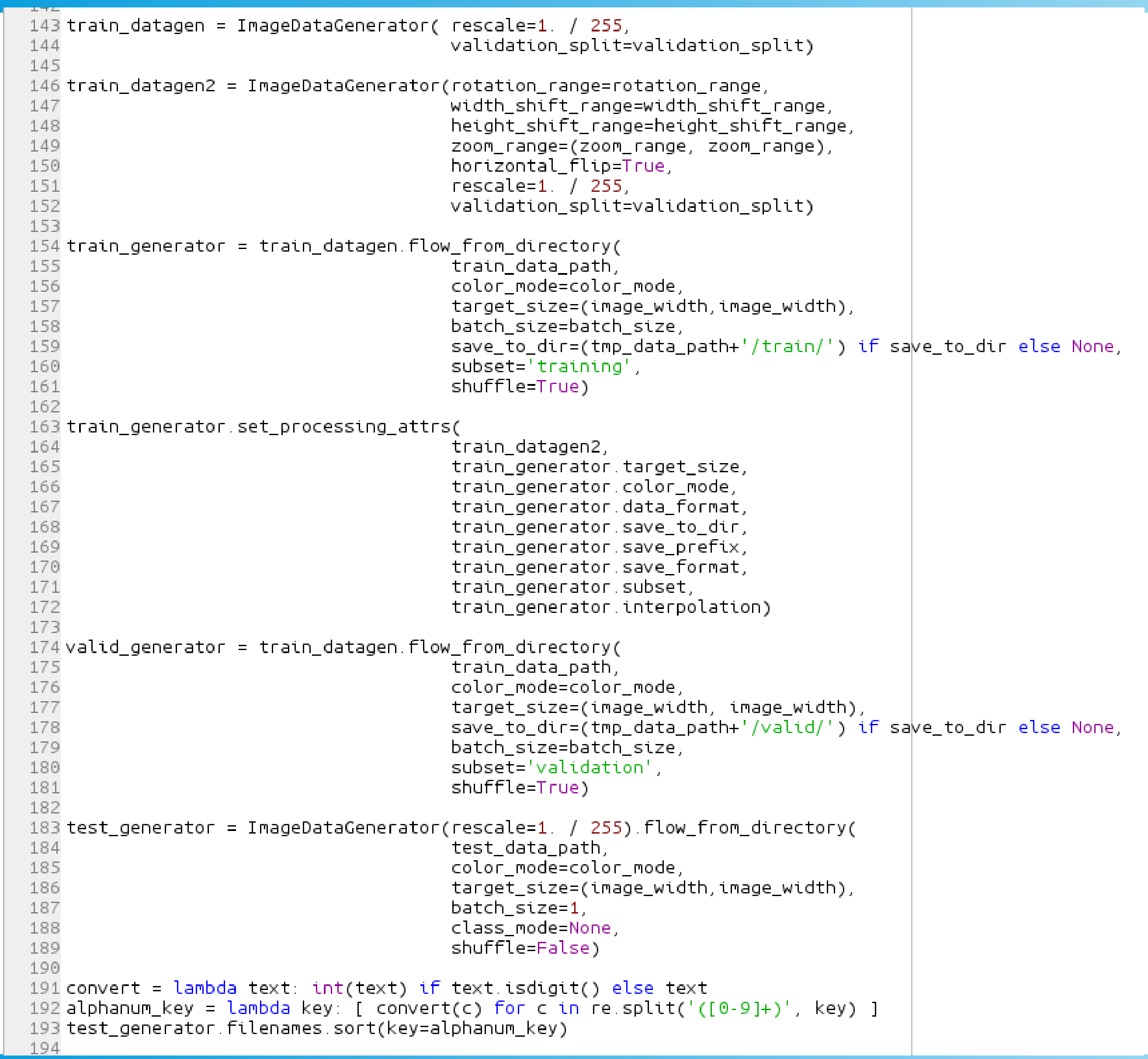
```python
class RestoreBestWeightsFinal(keras.callbacks.Callback):
    def __init__(self,
                 monitor='val_loss',
                 min_delta=0,
                 mode='auto',
                 baseline=None):
        super(RestoreBestWeightsFinal, self).__init__()

        self.monitor = monitor
        self.min_delta = min_delta
        self.best_weights = None

        if mode not in ['auto', 'min', 'max']:
            mode = 'auto'

        if mode == 'min':
            self.monitor_op = np.less
        elif mode == 'max':
            self.monitor_op = np.greater
        else:
            if 'acc' in self.monitor:
                self.monitor_op = np.greater
            else:
                self.monitor_op = np.less

        if self.monitor_op == np.greater:
            self.min_delta *= 1
        else:
            self.min_delta *= -1

    def on_train_begin(self, logs=None):
        self.best = np.Inf if self.monitor_op == np.less else -np.Inf
        self.best_weights = self.model.get_weights()

    def on_train_end(self, logs=None):
        if self.best_weights is not None:
            self.model.set_weights(self.best_weights)

    def on_epoch_end(self, epoch, logs=None):
        val_current = logs.get(self.monitor)
        if val_current is None:
            return

        if self.monitor_op(val_current - self.min_delta, self.best):
            self.best = val_current
            self.best_weights = self.model.get_weights()
            print('best check point')

callbacks = []
callbacks.append(keras.callbacks.EarlyStopping(monitor='val_acc', patience=20))
callbacks.append(RestoreBestWeightsFinal(monitor='val_acc'))
```

# Model



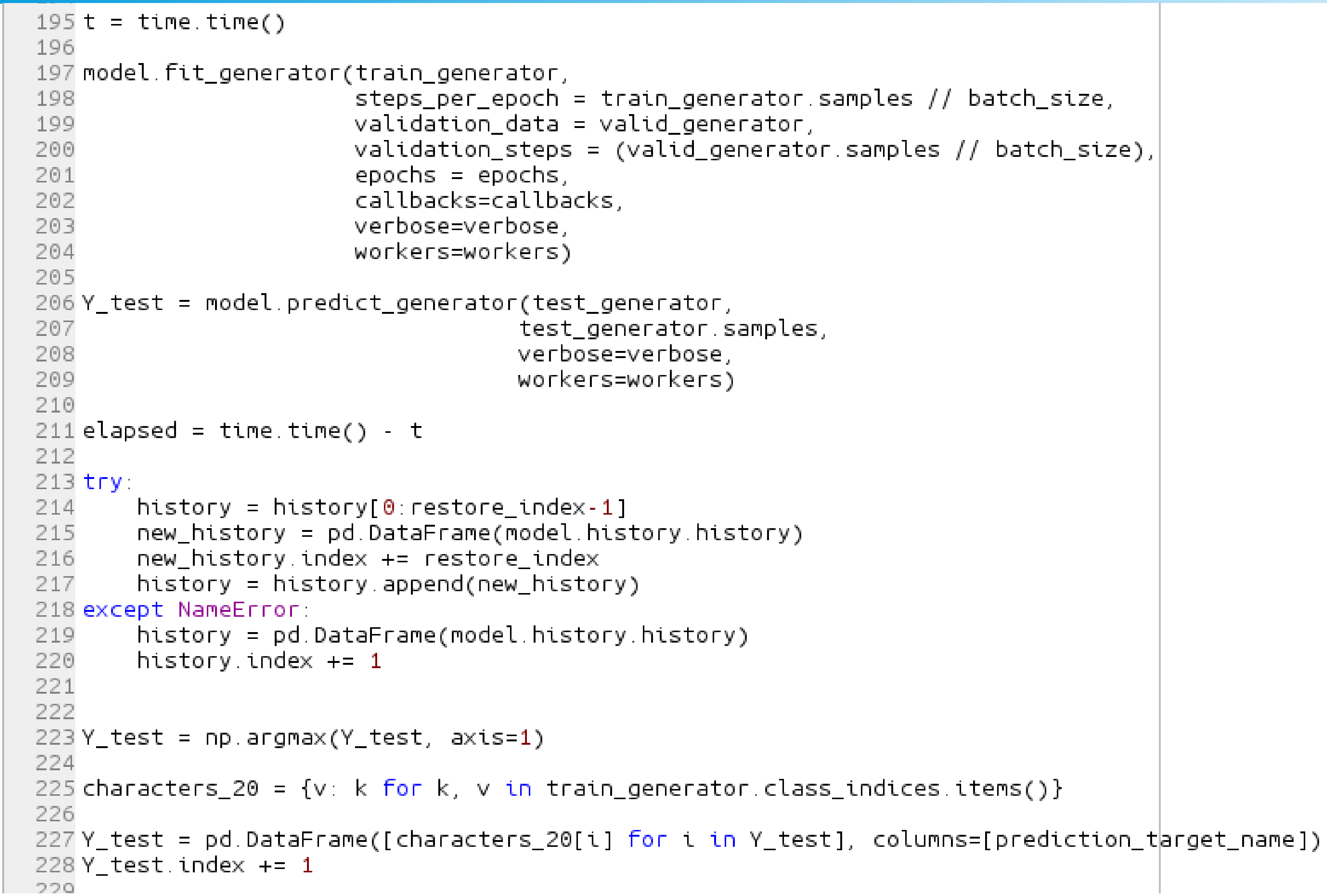
```python
filters = 32
model = Sequential()
model.add(Conv2D(filters=filters*1, kernel_size=3, padding='same', activation='relu',
                 input_shape=(image_width,image_width,1 if color_mode=='grayscale' else 3)))
model.add(Conv2D(filters=filters*1, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling2D((2, 2), strides=2))
model.add(BatchNormalization(epsilon=1e-12))
model.add(Dropout(0.25))
model.add(Conv2D(filters=filters*2, kernel_size=3, padding='same', activation='relu'))
model.add(Conv2D(filters=filters*2, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling2D((2, 2), strides=2))
model.add(BatchNormalization(epsilon=1e-12))
model.add(Dropout(0.25))
model.add(Conv2D(filters=filters*4, kernel_size=3, padding='same', activation='relu'))
model.add(Conv2D(filters=filters*4, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling2D((2, 2), strides=2))
model.add(BatchNormalization(epsilon=1e-12))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512, use_bias=False))
model.add(Activation('relu'))
model.add(BatchNormalization(epsilon=1e-12))
model.add(Dropout(0.5))
model.add(Dense(20, activation='softmax'))
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
model.summary()
```

# Flow data



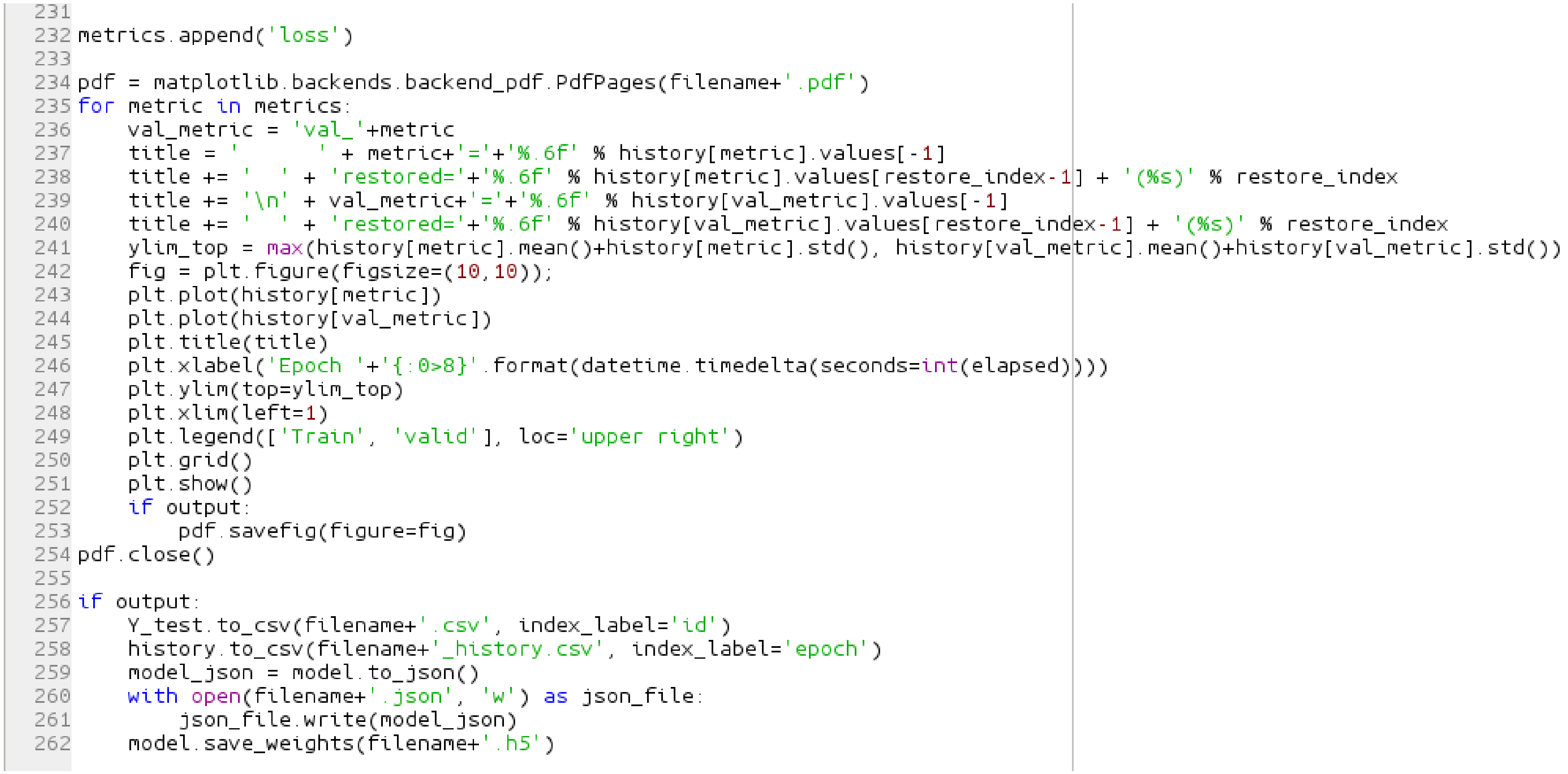
```
train_datagen = ImageDataGenerator( rescale=1. / 255,
                                    validation_split=validation_split)

train_datagen2 = ImageDataGenerator(rotation_range=rotation_range,
                                    width_shift_range=width_shift_range,
                                    height_shift_range=height_shift_range,
                                    zoom_range=(zoom_range, zoom_range),
                                    horizontal_flip=True,
                                    rescale=1. / 255,
                                    validation_split=validation_split)

train_generator = train_datagen.flow_from_directory(
                                    train_data_path,
                                    color_mode=color_mode,
                                    target_size=(image_width,image_width),
                                    batch_size=batch_size,
                                    save_to_dir=(tmp_data_path+'/train/') if save_to_dir else None,
                                    subset='training',
                                    shuffle=True)

train_generator.set_processing_attrs(
                                    train_datagen2,
                                    train_generator.target_size,
                                    train_generator.color_mode,
                                    train_generator.data_format,
                                    train_generator.save_to_dir,
                                    train_generator.save_prefix,
                                    train_generator.save_format,
                                    train_generator.subset,
                                    train_generator.interpolation)

valid_generator = train_datagen.flow_from_directory(
                                    train_data_path,
                                    color_mode=color_mode,
                                    target_size=(image_width, image_width),
                                    save_to_dir=(tmp_data_path+'/valid/') if save_to_dir else None,
                                    batch_size=batch_size,
                                    subset='validation',
                                    shuffle=True)

test_generator = ImageDataGenerator(rescale=1. / 255).flow_from_directory(
                                    test_data_path,
                                    color_mode=color_mode,
                                    target_size=(image_width,image_width),
                                    batch_size=1,
                                    class_mode=None,
                                    shuffle=False)

convert = lambda text: int(text) if text.isdigit() else text
alphanum_key = lambda key: [ convert(c) for c in re.split('([0-9]+)', key) ]
test_generator.filenames.sort(key=alphanum_key)

```

# Fit model and predict



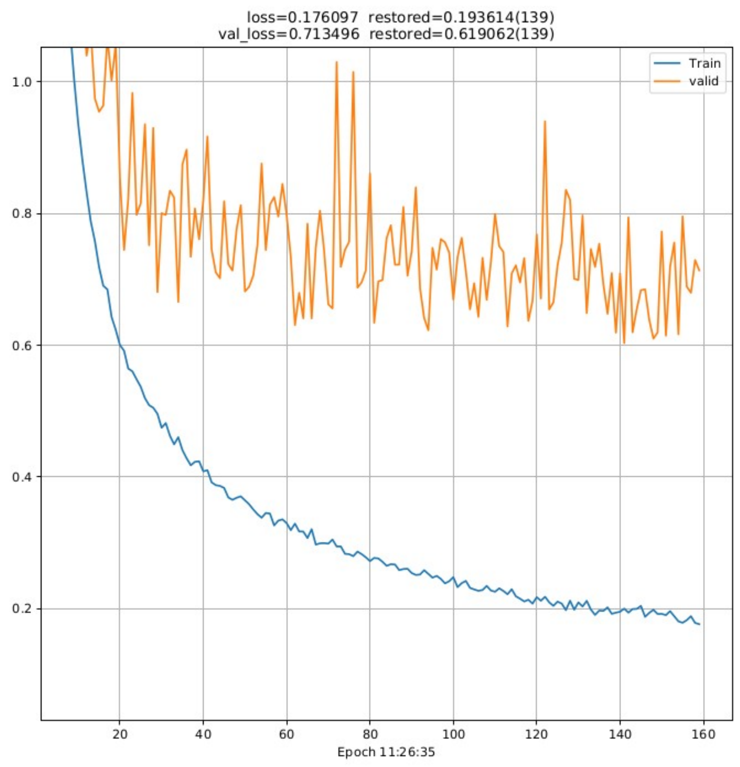
```python
t = time.time()

model.fit_generator(train_generator,
                    steps_per_epoch = train_generator.samples // batch_size,
                    validation_data = valid_generator,
                    validation_steps = (valid_generator.samples // batch_size),
                    epochs = epochs,
                    callbacks=callbacks,
                    verbose=verbose,
                    workers=workers)

Y_test = model.predict_generator(test_generator,
                                 test_generator.samples,
                                 verbose=verbose,
                                 workers=workers)

elapsed = time.time() - t

try:
    history = history[0:restore_index-1]
    new_history = pd.DataFrame(model.history.history)
    new_history.index += restore_index
    history = history.append(new_history)
except NameError:
    history = pd.DataFrame(model.history.history)
    history.index += 1


Y_test = np.argmax(Y_test, axis=1)

characters_20 = {v: k for k, v in train_generator.class_indices.items()}

Y_test = pd.DataFrame([characters_20[i] for i in Y_test], columns=[prediction_target_name])
Y_test.index += 1
```

# Output result

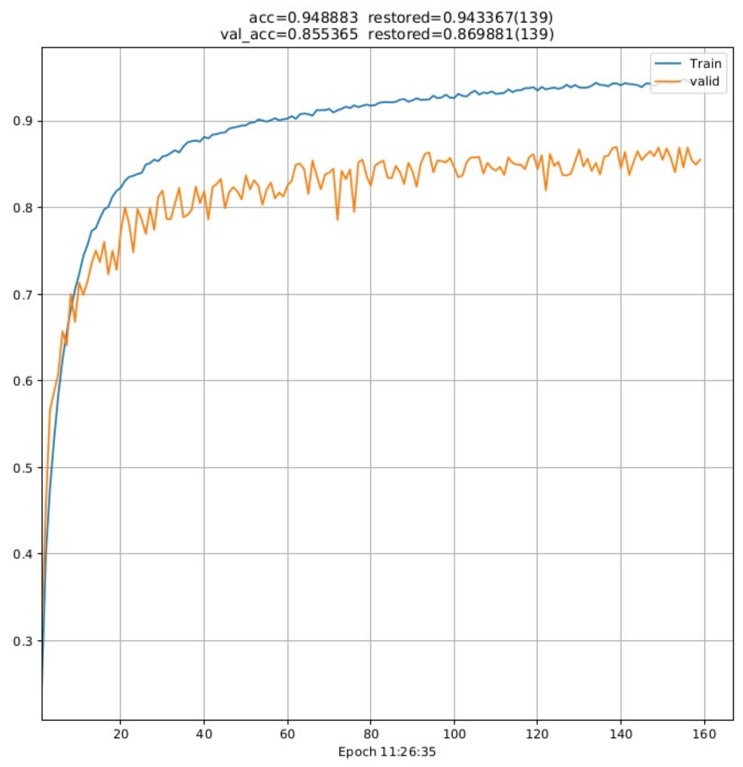


```python

metrics.append('loss')

pdf = matplotlib.backends.backend_pdf.PdfPages(filename+'.pdf')
for metric in metrics:
    val_metric = 'val_'+metric
    title = '       ' + metric+'='+'%.6f' % history[metric].values[-1]
    title += '   ' + 'restored='+'%.6f' % history[metric].values[restore_index-1] + '(%s)' % restore_index
    title += '\n' + val_metric+'='+'%.6f' % history[val_metric].values[-1]
    title += '   ' + 'restored='+'%.6f' % history[val_metric].values[restore_index-1] + '(%s)' % restore_index
    ylim_top = max(history[metric].mean()+history[metric].std(), history[val_metric].mean()+history[val_metric].std())
    fig = plt.figure(figsize=(10,10));
    plt.plot(history[metric])
    plt.plot(history[val_metric])
    plt.title(title)
    plt.xlabel('Epoch '+'{:0>8}'.format(datetime.timedelta(seconds=int(elapsed))))
    plt.ylim(top=ylim_top)
    plt.xlim(left=1)
    plt.legend(['Train', 'valid'], loc='upper right')
    plt.grid()
    plt.show()
    if output:
        pdf.savefig(figure=fig)
pdf.close()

if output:
    Y_test.to_csv(filename+'.csv', index_label='id')
    history.to_csv(filename+'_history.csv', index_label='epoch')
    model_json = model.to_json()
    with open(filename+'.json', 'w') as json_file:
        json_file.write(model_json)
    model.save_weights(filename+'.h5')
```

# History



Loss



acc

# 改進方式

1. 將較無法辨識的圖片刪除
2. 加入 Dropout, BatchNormalization 等
3. 嘗試使用不同的 Optimizer