Experiment No. 4 : Classification of MNIST Fashion dataset using CNN

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from keras.utils import to_categorical
from matplotlib.pyplot import figure, show
import warnings
import seaborn as sns
warnings.flterwarnings(ignore')
import matplotlib.style as style
from sklearn.model_selection import train_test_split
from keras.layers import Input, Concatenate, concatenate, Dense, Embedding, Dropout,
Conv2D, MaxPooling2 D
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau, History
from keras.layers import Dropout, Flatten, GlobalAveragePooling2D, Activation

from sklearn,preprocessing import LabelEncoder, OneHotEncoder
from keras.preprocessing.image import imageDataGenerator
from sklearn.model_selection import train_test_split
from keras.applications.resnet50 import ResNet50
from keras.callbacks import ReduceLROnPlateau
from keras.callbacks import ModelCheckpoint
from keras.applications.vggl6 import VGG16
from keras.utils import to_categorical
from sklearn.utils import class_weight
from keras.layers.normalization import BatchNormalization
from matplotlib import pyplot as plt
from keras import backend as K
from keras.optimizers import SGD
from keras.models import Model
import seaborn as sns
import numpy as np
import argparse
import time
import glob
import cv2
import numpy
import os
import glob
ímport sys
import os
import json
import pprint
import warnings
warnings.flterwarnings('ignore')

#Load Data
!curl -L -0 https://www.dropbox.com/s/heyqll2my8uwotq/fashionmaistzip
!unzip fashionmnist:zip

#Load training and test data using dataframes from Pandas.
train = pd. read_csv("fashion-mnist_train.csv")
test = pd.read csv("fashion-mnist_test.csv'")
```

```
Img_rows, img_cols = 28, 28
input_shape = (img_rows, img_cols, 1)
X= train.iloc[:,1:]
Y= traln.iloc[:,:1]
X_ test = test.iloc[:, 1:]
Y_test = test.iloc[:,: 1]

#Normalization
X= np.asarray(X).reshape (X.shape [0], img_rows,img_cols, 1)
X_test = np.asarray(X_ test).reshape(X_test.shape [0], img_rows,ímg_cols,1)
X= (255. - X) /255.
X_test = (255. - X_test) / 255.

#Number of classes
classes = len(Y['label'],value _counts())

print("Number of features: ", X.shape[1])
print("Number of train samples: ", Xshape [0])
print("Number of test samples: ", X test.shape [0])
```

```
OUT:
Number of features: 28
Number of train samples: 60000
Number of test samples: 10000
```

```
#Training
```

```
Y_test = to_categorical(Y_test)
Y= to_categorical (Y)
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, stratify=Y, test_size=0.2,
random_state=66)
Irr = ReducelLROn Plateau (monitor='val_loss', factor=0.1, patience=2, verbose=1,
epsilon=1e-3, mode= ‘min')
early_stopping = EarlyStopping(monitor='val loss',patience=5,verbose=0, mode='auto’)
checkpoint = ModelCheckpoint("checkpoint.hdf5", monitor='val_acc', verbose=1,
save_best_only=True, mode='max’)
batch size = 64
epochs = 10


from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier


# define the grid search parameters
batch_size = [16, 32, 64, 80]
epochs = [10, 25, 50]
param_grid = dict (batch_size=batch_size, epochs=epochs)
model = KerasClassifier(build_fn=model_basic, verbose=0)
Grid=GridSearchCV(estimator=model, param_grid=param_grid,n_jobs1, cv=3)
grid_result = grid.fit(X_train, Y_train)


# summarize results
print("Best: %using %s" % (grid_result.best_score, grid_result.bestparams_))
means = grid_result.cv_results ['mean_test_score']
```

```
stds = grid_result.cv_results ['std_test_score']

params =grid_result.cv_results ['params']

for mean, stdev, param in zip(means, stds, params):

print("%f (%f) with: %r" % (mean, stdev, param))


#Model

def model_basic(classes=classes,optimizer='adam'):

kernel_size = (3,3)

dropout = 0.25

pool_size = (2,2)

inputs = Input(shape=(img_rows, img_cols, 1))

y= Conv2D (filters=32, kernel_size=kernel_size,activation='relu',padding='same’) (inputs)

y= MaxPooling2D(pool_size=pool_size,strides=(2,2)) (y)

y= Dropout(dropout)(y)


y= Flatten()(y)


y= Dense(256,activation=’relu')(y)

y= BatchNormalization()(y)

y= Dropout(dropout)(y)

outputs = Dense(classes, activation='softmax’)(y)


model = Model(inputs=inputs, outputs=outputs)

model.compile (optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'|)

return model

basic_model = model_basic()
```

```
import warnings

warnings.filterwarnings('ignore')

history = basic_model.fit(X_train, Y_train, batch size=batch_size, epochs=epochs,

verbose=1, validation_data=(X_val, Y_val)
```

```
OUT:

Train on 48000 samples, validate on 12000 samples

Epoch 1/10

48000/48000 [=======]-82s Zms/step- loss: 0.4390 - acc:.8470 - val_loss: 0.4019val _acc:

0.8618

Epoch 2/10

48000/48000 [=========]-82s 2ms/step - loss: 0.3458 - acc: 0.8776 - val_loss: 0.3046

val_acc: 0.8932

Epoch 3/10

48000/48000 [========] -82s 2ms/step - loss: 0.3110 - acc: 0.8883 - val_loss: 0.2947 -

val_acc: 0.8953

Epoch 4/10

48000/48000 [===]-81s 2ms/step - loss: 0.2935 - acc: 0.8937 - val_loss: 0.2772 -

val_acc: 0.9024

Epoch 5/10

48000/48000 [=========]-85s 2ms/step- loss: 0.2710 - acc: 0.9030 - val_loss: 0.2855-

val_acc: 0.8952

Epoch 6/10[===========| -84s 2ms/step - loss: 0.2592 - acc: 0.9067 - val _loss: 0.2574-

val_acc: 0.9063

Epoch 7/10

48000/48000 [==========]-85s 2ms/step-loss: 0.2450 - acc: 0.9107 - val_loss: 0.2773 -
```

```
val acc: 0.8998

Epoch 8/10

48000/48000 [=======] - 84s 2ms/step - loss: 0.2338 - acc: 0.9147 – val_loss: 0.2833 -

val_acc: 0.8955

Epoch 9/10

48000/48000 [========]-82s 2ms/step - loss: 0.2239 - acc: 0.9174 – val_loss: 0.2553 -

val_acc: 0.9127

Epoch 10/10

48000/48000 [========]-84s 2ms/step - loss: 0.2153 - acc: 0.9207 - val loss: 0.2564 -

val_acc: 0.9123
```

```
score = basic_model.evaluate(X_test,Y_test, verbose=0)

print("Test loss:", score[0])

print("Test accuracy:', score[1])
```

OUT:

```
Test loss: 0.2463475521683693

Test accuracy: 0.9137
```