Thus the LeNet architecture has been implemented successfully

AD18511 – DEEP LEARNING LABORATORY

DATE:

EX.NO: 9 **IMPLEMENTATION OF ALEXNET ARCHITECTURE.**

AIM:

To implement alexnet architecture for image recognition and classification tasks.

DESCRIPTION:

- The AlexNet CNN architecture won the 2012 ImageNet ILSVRC challenges of deep learning algorithm by a large variance by achieving 17% with top-5 error rate as the second best achieved 26%!
- It was introduced by Alex Krizhevsky (name of founder), The Ilya Sutskever and Geoffrey Hinton are quite similar to LeNet-5, only much bigger and deeper and it was introduced first to stack convolutional layers directly on top of each other models, instead of stacking a pooling layer top of each on CN network convolutional layer.
- AlexNNet has 60 million parameters as AlexNet has total 8 layers, 5 convolutional and 3 fully connected layers.
- AlexNNet is first to execute (ReLUs) Rectified Linear Units as activation functions.
- it was the first CNN architecture that uses GPU to improve the performance.

PROGRAM:

```
# Importing Libraries
import tensorflow as tf
from tensorflow import keras
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam,SGD
from tensorflow.keras.callbacks import TensorBoard

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

import os
import time


train_df = pd.read_csv('/fashion-mnist_train.csv',sep=',')
test_df = pd.read_csv('/fashion-mnist_test.csv', sep = ',')

from google.colab import drive
drive.mount('/content/drive')

train.head (10)
train_data= np.array(train_df, dtype= 'float32')
test_data= np.array(test_df, dtype= 'float32')

x_train = train_data[:,1:]/255
y_train = train_data[:,0]
x_test= test_data[:,1:]/255
y_test=test_data[:,0]
```

# Example of training label content
print(y_train[0], y_train[43], y_train[1923])
print("Minimum value of training labels", y_train.min())
print("Maximum value of training labels", y_train.max())

OUTPUT:
0.0 6.0 2.0
Minimum value of training labels 0.0
Maximum value of training labels 9.0

```
x_train,x_validate,y_train,y_validate = train_test_split(x_train,y_train,test_size = 0.2,random_state = 12345)
x_train.shape, x_validate.shape,y_train.shape,y_validate.shape
```

OUTPUT:
((8000, 784), (2000, 784), (8000,), (2000,))

```
class_names= ['Tshirt', 'Trouser', 'Pullover', 'Dress', 'Coat',
              'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
plt.figure(figsize=(10,10))
for i in range(36):
    plt.subplot(6,6, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i].reshape((28,28)))
    label_index= int(y_train[i])
    plt.title(class_names[label_index])
plt.show()
```

OUTPUT:

a 

```
image_rows = 28
image_cols = 28
batch_size = 4096
image_shape = (image_rows,image_cols,1)

x_train= x_train.reshape(x_train.shape[0], *image_shape)
x_test = x_test.reshape(x_test.shape[0],*image_shape)
x_validate = x_validate.reshape(x_validate.shape[0],*image_shape)
x_train.shape, x_test.shape, x_validate.shape

# Alexnet
model= tf.keras.Sequential([Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation= 'relu', input_shape=
image_shape),
                        BatchNormalization(),
                          MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='same'),
                          Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding='same'),
                          BatchNormalization(),
                          MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='same'),
                          Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu',padding='same'),
                          BatchNormalization(),
                          Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu',padding='same'),
                          BatchNormalization(),
                          Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu',padding='same'),
                          BatchNormalization(),
                          MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='same'),
                          Flatten(),
                          Dense(4096, activation='relu'),
                          Dropout(0.5),
                          Dense(4096, activation='relu'),
                          Dropout(0.5),
                          Dense(10, activation='softmax')
                          ])
model.summary()
```

OUTPUT:

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 5, 5, 96)          11712

 batch_normalization (Batch  (None, 5, 5, 96)          384
 Normalization)

 max_pooling2d (MaxPooling2  (None, 3, 3, 96)          0
 D)

 conv2d_1 (Conv2D)           (None, 3, 3, 256)         614656

 batch_normalization_1 (Bat  (None, 3, 3, 256)         1024
 chNormalization)
```

max_pooling2d_1 (MaxPoolin	(None, 2, 2, 256)	0

```
 g2D)

 conv2d_2 (Conv2D)          (None, 2, 2, 384)        885120

 batch_normalization_2 (Bat  (None, 2, 2, 384)        1536
 chNormalization)

 conv2d_3 (Conv2D)          (None, 2, 2, 384)        1327488

 batch_normalization_3 (Bat  (None, 2, 2, 384)        1536
 chNormalization)

 conv2d_4 (Conv2D)          (None, 2, 2, 256)        884992

 batch_normalization_4 (Bat (None, 2, 2, 256)        1024
 chNormalization)

 max_pooling2d_2 (MaxPoolin (None, 1, 1, 256)        0
 g2D)

 flatten (Flatten)          (None, 256)              0

 dense (Dense)              (None, 4096)             1052672

 dropout (Dropout)          (None, 4096)             0

 dense_1 (Dense)            (None, 4096)             16781312

 dropout_1 (Dropout)        (None, 4096)             0

 dense_2 (Dense)            (None, 10)               40970

=================================================================
Total params: 21604426 (82.41 MB)
Trainable params: 21601674 (82.40 MB)
Non-trainable params: 2752 (10.75 KB)
_________________________________________________________________


from tensorflow.keras.optimizers import SGD

def lr_schedule(epoch):
    lr = 0.01
    if epoch > 50:
        lr *= 0.1
    elif epoch > 75:
        lr *= 0.01
    return lr

sgd = SGD(learning_rate=0.01, momentum=0.9, nesterov=True)
model.compile(loss="sparse_categorical_crossentropy", optimizer=sgd, metrics=["accuracy"])
from tensorflow.keras.callbacks import LearningRateScheduler
lr_scheduler = LearningRateScheduler(lr_schedule)
```

# train!
early_stopping_cb = keras.callbacks.EarlyStopping(monitor='val_loss', patience=30, verbose=1, mode='min')
history= model.fit(x_train, y_train, epochs=5, verbose=1, callbacks=[early_stopping_cb],
        validation_data=(x_validate, y_validate))

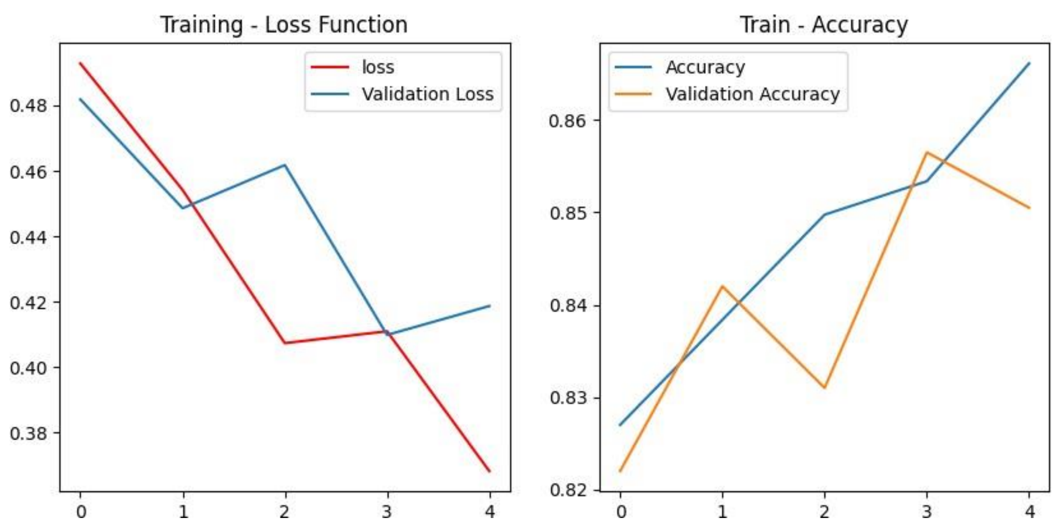
OUTPUT:

Epoch 1/5
250/250 [==============================] - 220s 881ms/step - loss: 0.4928 - accuracy: 0.8270 - val_loss: 0.4818 - val_accuracy: 0.8220
Epoch 2/5
250/250 [==============================] - 218s 872ms/step - loss: 0.4541 - accuracy: 0.8384 - val_loss: 0.4485 - val_accuracy: 0.8420
Epoch 3/5
250/250 [==============================] - 213s 852ms/step - loss: 0.4073 - accuracy: 0.8497 - val_loss: 0.4617 - val_accuracy: 0.8310
Epoch 4/5
250/250 [==============================] - 216s 863ms/step - loss: 0.4110 - accuracy: 0.8534 - val_loss: 0.4099 - val_accuracy: 0.8565
Epoch 5/5
250/250 [==============================] - 222s 890ms/step - loss: 0.3683 - accuracy: 0.8661 - val_loss: 0.4187 - val_accuracy: 0.8505

plt.figure(figsize=(10,10))
plt.subplot(2,2,1)
plt.plot(history.history['loss'], label='loss',color='r')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Training - Loss Function')

plt.subplot(2, 2, 2)
plt.plot(history.history['accuracy'], label='Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Train - Accuracy')

OUTPUT:



```
model_evaluation_results = model.evaluate(x_test, y_test, batch_size=32, verbose=2)
print("The test loss is", model_evaluation_results[0])
print("The test accuracy is", model_evaluation_results[1])
```

OUTPUT:

```
533/533 - 42s - loss: nan - accuracy: 0.8438 - 42s/epoch - 79ms/step
The test loss is nan
The test accuracy is 0.8438380360603333
```

```
# Prediction on test images using model.predict() method
practical_test_images =  x_test[:10]
prediction_probabilites = model.predict(practical_test_images)
prediction_probabilites[:3]
```

OUTPUT:

```
1/1 [============================== = = = = = = = = = = = =  = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = =
 1.62392721e-01, 6.00572117e-03, 3.07703376e-01, 3.86583386e-03,
 8.07231665e-02, 4.55609756e-03],
 [2.13454390e-04, 1.31126042e-04, 2.42721246e-04, 1.57689094e-04,
 2.41081478e-04, 5.40605932e-03, 3.75541451e-04, 3.94529492e-01,
 3.35998135e-04, 5.983666857e-01],
 [1.77102792e-03, 9.13637981e-04, 1.02652036e-01, 3.12519167e-03,
 5.54388940e-01, 6.66437962e-04, 3.32357943e-01, 6.95253024e-04,
 2.90616718e-03, 5.23336872e-04], dtype=float32)
```

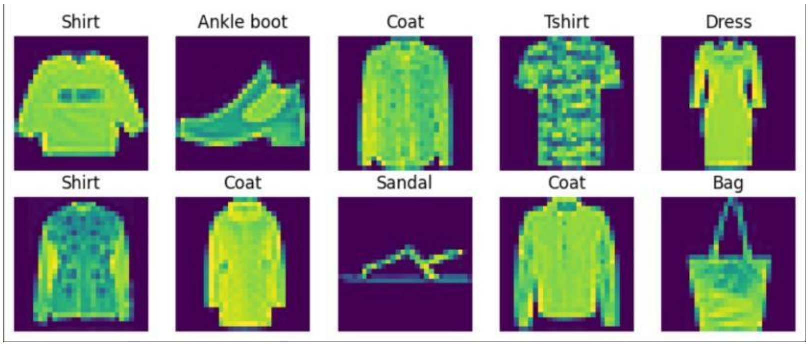
```
# Clean up model prediction using argmax to find the index of the largest probablity
def derive_predicted_classes(prediction_probabilites):
    batch_prediction = []
    for vector in prediction_probabilites:
        batch_prediction.append(np.argmax(vector))
    return batch_prediction

model_prediction = derive_predicted_classes(prediction_probabilites)
model_prediction
```

OUTPUT:

```
[6, 9, 4, 0, 3, 6, 4, 5, 4, 8]
```

```
# Visualise the prediction result
plt.figure(figsize=(10,10))
for i in range(len(practical_test_images)):
    plt.subplot(5,5, i+1)
    plt.axis("off")
    plt.grid(False)
    plt.imshow(practical_test_images[i])
    plt.title(class_names[model_prediction[i]])
plt.show()
```

OUTPUT:



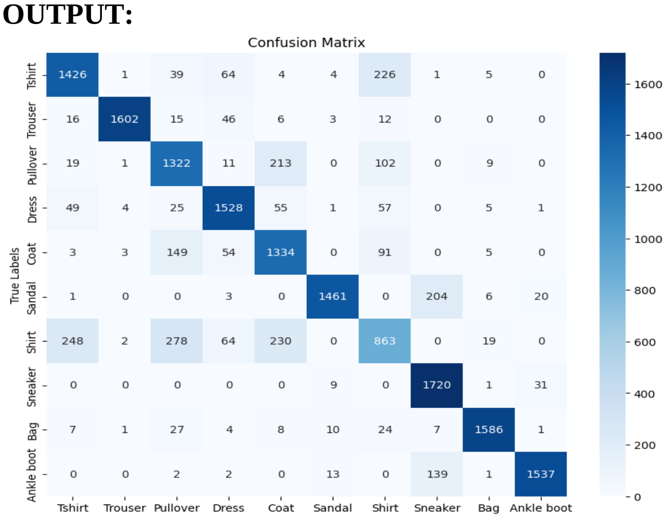

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Reshape x_test to have the correct shape
x_test = x_test.reshape(x_test.shape[0], image_rows, image_cols, 1)

# Calculate accuracy score
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
accuracy = accuracy_score(y_test, y_pred_classes)
print("Accuracy:", accuracy)
```

OUTPUT:

533/533 [==============================] - 41s 78ms/step Accuracy: 0.8438380281690141

```
# Compute confusion matrix
confusion_mtx = confusion_matrix(y_test, y_pred_classes)

# Display confusion matrix as a heatmap
import seaborn as sns
plt.figure(figsize=(10,8))
sns.heatmap(confusion_mtx, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

OUTPUT:



```
# Generate and display classification report
classification_rep = classification_report(y_test, y_pred_classes,
target_names=class_names) print("Classification Report:\n", classification_rep)
```

OUTPUT:

Classification Report:

	precision	recall	f1-score	support
Tshirt	0.81	0.81	0.81	1770
Trouser	0.99	0.94	0.97	1700
Pullover	0.71	0.79	0.75	1677
Dress	0.86	0.89	0.87	1725
Coat	0.72	0.81	0.76	1639
Sandal	0.97	0.86	0.91	1695
Shirt	0.63	0.51	0.56	1704
Sneaker	0.83	0.98	0.90	1761
Bag	0.97	0.95	0.96	1675
Ankle boot	0.97	0.91	0.94	1694
accuracy			0.84	17040
macro avg	0.85	0.84	0.84	17040
weighted avg	0.85	0.84	0.84	17040

RESULT:

Thus the ALexNet architecture has been implemented successfully.