Exp No: 3 BUILD A CONVOLUTIONAL NEURAL NETWORK

Aim:

To build a simple convolutional neural network with Keras/TensorFlow. Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Program:

# To load the mnist data from keras.datasets import fashion\_mnist

from tensorflow.keras.models import Sequential

# importing various types of hidden layers from tensorflow.keras.layers import Conv2D, MaxPooling2D,\

Dense, Flatten

# Adam optimizer for better LR and less loss from tensorflow.keras.optimizers import Adam import matplotlib.pyplot as plt import numpy as np

# Split the data into training and testing

(trainX, trainy), (testX, testy) = fashion\_mnist.load\_data()

# Print the dimensions of the dataset print('Train: X = ', trainX.shape) print('Test: X = ', testX.shape)

def model\_arch():

models = Sequential()

# We are learning 64

# filters with a kernal size of 5x5

models.add(Conv2D(64, (5, 5),

padding="same", activation="relu", input\_shape=(28, 28, 1)))

# Max pooling will reduce the # size with a kernal size of 2x2 models.add(MaxPooling2D(pool\_size=(2, 2))) models.add(Conv2D(128, (5, 5), padding="same",

activation="relu"))

models.add(MaxPooling2D(pool\_size=(2, 2))) models.add(Conv2D(256, (5, 5), padding="same",

activation="relu"))

models.add(MaxPooling2D(pool\_size=(2, 2)))

# Once the convolutional and pooling

# operations are done the layer

# is flattened and fully connected layers

# are added models.add(Flatten()) models.add(Dense(256, activation="relu")) # Finally as there are total 10

# classes to be added a FCC layer of

# 10 is created with a softmax activation

# function models.add(Dense(10, activation="softmax"))

return models

model = model\_arch()

model.compile(optimizer=Adam(learning\_rate=1e-3),

loss='sparse\_categorical\_crossentropy', metrics=['sparse\_categorical\_accuracy'])

model.summary()

history = model.fit( trainX.astype(np.float32), trainy.astype(np.float32),

epochs=5, steps\_per\_epoch=50, validation\_split=0.33

)

# Accuracy vs Epoch plot plt.plot(history.history['sparse\_categorical\_accuracy']) plt.plot(history.history['val\_sparse\_categorical\_accuracy']) plt.title('Model Accuracy') plt.ylabel('Accuracy') plt.xlabel('epoch') plt.legend(['train', 'val'], loc='upper left')

plt.show()

# Loss vs Epoch plot plt.plot(history.history['loss']) plt.plot(history.history['val\_loss']) plt.title('Model Accuracy') plt.ylabel('loss') plt.xlabel('epoch')

plt.legend(['train', 'val'], loc='upper left')

plt.show()

# There are 10 output labels for the # Fashion MNIST dataset

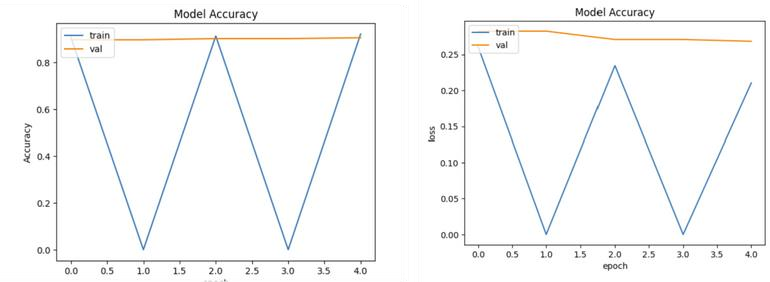
labels = ['t\_shirt', 'trouser', 'pullover',

'dress', 'coat', 'sandal', 'shirt',

'sneaker', 'bag', 'ankle\_boots']

# Make a prediction predictions = model.predict(testX[:1]) label = labels[np.argmax(predictions)] print(label) plt.imshow(testX[:1][0]) plt.show()

Output:



Result:

CNN has been successfully built using the provided resources.