Name: Nikhil Arora

Loading Packages

In [1]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from keras.utils import to_categorical
from keras.optimizers import SGD
```

```
Using TensorFlow backend.
```

Importing and Reshaping the Dataset

In [2]:
```
from keras.datasets import fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# reshape dataset to have a single channel
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1))
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1))

# Normalizing the pixel values to be in the range [0, 1]
# by dividing them by 255
x_train = x_train/255.0
x_test = x_test/255.0

# Converting the labels into one-hot vectors
y_train = to_categorical(y_train, num_classes = 10)
y_test = to_categorical(y_test, num_classes = 10)
```

Task 1: Build a neural network without convolutional layers to do the classification task

Step 1: Defining a Model Architecture

In [3]:
```
model_T1 = Sequential()

#Input Layer
model_T1.add(Flatten(input_shape=(28, 28, 1)))

#Hidden Layers
model_T1.add(Dense(128, activation='relu'))
model_T1.add(Dense(64, activation='relu'))

#Output Layer
model_T1.add(Dense(10, activation='softmax'))
```

```
WARNING:tensorflow:From C:\Users\nikhi\anaconda3\envs\NeuralNetwork\lib\si
te-packages\tensorflow_core\python\ops\resource_variable_ops.py:1630: call
ing BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_var
iable_ops) with constraint is deprecated and will be removed in a future v
ersion.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

Step 2: Compiling the model

In [4]:
```
# Defining Optimizer, loss function and evaluation metrics
model_T1.compile(loss='categorical_crossentropy', optimizer='adam', met
```

In [5]:
```
# Model Structure
model_T1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
_________________________________________________________________
```

Step 3: Training the Model

In [6]:
```
# Setting a random state so that comparing the evaluation metrics is po
tf.set_random_seed(1)
np.random.seed(1)

model_T1.fit(x_train, y_train, epochs=10, batch_size=32, validation_dat
```

```
WARNING:tensorflow:From C:\Users\nikhi\AppData\Local\Temp\ipykernel_9356\3
595446061.py:2: The name tf.set_random_seed is deprecated. Please use tf.c
ompat.v1.set_random_seed instead.

WARNING:tensorflow:From C:\Users\nikhi\anaconda3\envs\NeuralNetwork\lib\si
te-packages\keras\backend\tensorflow_backend.py:422: The name tf.global_va
riables is deprecated. Please use tf.compat.v1.global_variables instead.

Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 5s 82us/step - loss: 0.4904
- accuracy: 0.8259 - val_loss: 0.4505 - val_accuracy: 0.8413
Epoch 2/10
60000/60000 [==============================] - 5s 88us/step - loss: 0.3691
- accuracy: 0.8654 - val_loss: 0.3945 - val_accuracy: 0.8534
Epoch 3/10
60000/60000 [==============================] - 5s 81us/step - loss: 0.3340
- accuracy: 0.8770 - val_loss: 0.3775 - val_accuracy: 0.8679
Epoch 4/10
60000/60000 [==============================] - 5s 81us/step - loss: 0.3108
- accuracy: 0.8848 - val_loss: 0.3574 - val_accuracy: 0.8713
Epoch 5/10
60000/60000 [==============================] - 5s 80us/step - loss: 0.2935
- accuracy: 0.8900 - val_loss: 0.3512 - val_accuracy: 0.8673
Epoch 6/10
60000/60000 [==============================] - 5s 79us/step - loss: 0.2771
- accuracy: 0.8977 - val_loss: 0.3364 - val_accuracy: 0.8815
Epoch 7/10
60000/60000 [==============================] - 5s 82us/step - loss: 0.2668
- accuracy: 0.8999 - val_loss: 0.3398 - val_accuracy: 0.8770
Epoch 8/10
60000/60000 [==============================] - 5s 80us/step - loss: 0.2557
- accuracy: 0.9046 - val_loss: 0.3461 - val_accuracy: 0.8737
Epoch 9/10
60000/60000 [==============================] - 5s 85us/step - loss: 0.2450
- accuracy: 0.9086 - val_loss: 0.3459 - val_accuracy: 0.8791
Epoch 10/10
60000/60000 [==============================] - 5s 78us/step - loss: 0.2366
- accuracy: 0.9113 - val_loss: 0.3466 - val_accuracy: 0.8854
```

Out[6]:
```
<keras.callbacks.callbacks.History at 0x17b3c105188>
```

Step 4: Evaluating the model with Testing dataset

In [7]:
```
scores = model_T1.evaluate(x_test, y_test, verbose=1)
print("%s: %.2f%%" % (model_T1.metrics_names[1], scores[1]*100))
```

```
10000/10000 [==============================] - 0s 33us/step
accuracy: 88.54%
```

Step 5: Changing the Model Structure to get better evaluation results

Changing the number of neurons:

In [8]:
```
model_T1 = Sequential()
model_T1.add(Flatten(input_shape=(28, 28, 1)))
model_T1.add(Dense(256, activation='relu'))
model_T1.add(Dense(128, activation='relu'))
model_T1.add(Dense(10, activation='softmax'))

# Defining Optimizer, loss function and evaluation metrics
model_T1.compile(loss='categorical_crossentropy', optimizer='adam', met

# Setting a random state so that comparing the evaluation metrics is po
tf.set_random_seed(1)
np.random.seed(1)
model_T1.fit(x_train, y_train, epochs=10, batch_size=32, validation_dat

scores = model_T1.evaluate(x_test, y_test, verbose=1)
print("%s: %.2f%%" % (model_T1.metrics_names[1], scores[1]*100))
```

```
10000/10000 [==============================] - 1s 51us/step
accuracy: 88.70%
```

Task 2: Build a neural network with the use of convolutional layers

Step 1: Defining Model Architecture

In [9]:
```
model_T2 = Sequential()
#Convolutional Layer 1
model_T2.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,
                    padding='same'))
#Pooling Layer 1
model_T2.add(MaxPooling2D((2, 2)))
#Convolutional Layer 2
model_T2.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
#Pooling Layer 1
model_T2.add(MaxPooling2D((2, 2)))
#Flatten Layer
model_T2.add(Flatten())
#Fully-Connected Layer 1
model_T2.add(Dense(128, activation='relu'))
#Fully-Connected Layer 2 (Output Layer)
model_T2.add(Dense(10, activation='softmax'))
```

```
WARNING:tensorflow:From C:\Users\nikhi\anaconda3\envs\NeuralNetwork\lib\si
te-packages\keras\backend\tensorflow_backend.py:4070: The name tf.nn.max_p
ool is deprecated. Please use tf.nn.max_pool2d instead.
```

Step 2: Compiling the Model

In [10]:
```
# Defining Optimizer, loss function and evaluation metrics
from keras.optimizers import Adam
custom_adam = Adam(lr=0.002)

model_T2.compile(loss='categorical_crossentropy', optimizer=custom_adam
```

In [11]:
```
# Model Structure
model_T2.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               401536
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1290
=================================================================
Total params: 421,642
Trainable params: 421,642
Non-trainable params: 0
_________________________________________________________________
```

Step 3: Training the Model

In [12]:
```
# Setting a random state so that comparing the evaluation metrics is po
tf.set_random_seed(1)
np.random.seed(1)

model_T2.fit(x_train, y_train, epochs=10, batch_size=32, validation_dat
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 58s 959us/step - loss: 0.38
01 - accuracy: 0.8636 - val_loss: 0.3107 - val_accuracy: 0.8880
Epoch 2/10
60000/60000 [==============================] - 58s 969us/step - loss: 0.24
97 - accuracy: 0.9080 - val_loss: 0.2640 - val_accuracy: 0.9023
Epoch 3/10
60000/60000 [==============================] - 58s 972us/step - loss: 0.20
82 - accuracy: 0.9222 - val_loss: 0.2709 - val_accuracy: 0.9051
Epoch 4/10
60000/60000 [==============================] - 59s 991us/step - loss: 0.17
69 - accuracy: 0.9334 - val_loss: 0.2417 - val_accuracy: 0.9097
Epoch 5/10
60000/60000 [==============================] - 58s 966us/step - loss: 0.14
91 - accuracy: 0.9435 - val_loss: 0.2567 - val_accuracy: 0.9156
Epoch 6/10
60000/60000 [==============================] - 59s 979us/step - loss: 0.12
69 - accuracy: 0.9513 - val_loss: 0.2710 - val_accuracy: 0.9087
Epoch 7/10
60000/60000 [==============================] - 58s 967us/step - loss: 0.10
72 - accuracy: 0.9589 - val_loss: 0.2906 - val_accuracy: 0.9142
Epoch 8/10
60000/60000 [==============================] - 59s 985us/step - loss: 0.09
22 - accuracy: 0.9656 - val_loss: 0.3263 - val_accuracy: 0.9092
Epoch 9/10
60000/60000 [==============================] - 59s 983us/step - loss: 0.07
64 - accuracy: 0.9709 - val_loss: 0.3546 - val_accuracy: 0.9145
Epoch 10/10
60000/60000 [==============================] - 59s 983us/step - loss: 0.07
08 - accuracy: 0.9730 - val_loss: 0.4068 - val_accuracy: 0.9114
```

Out[12]:
```
<keras.callbacks.callbacks.History at 0x17b3c5a31c8>
```

Step 4: Evaluating the model with Testing dataset

In [13]:
```
scores = model_T2.evaluate(x_test, y_test, verbose=1)
print("%s: %.2f%%" % (model_T1.metrics_names[1], scores[1]*100))
```

```
10000/10000 [==============================] - 2s 232us/step
accuracy: 91.14%
```

Step 5: Changing the Model Structure to get better evaluation results

Changing the model structure by increasing the number of filters in the convolutional layer to 64 and 128 respectively

In [14]:

```
model_T2 = Sequential()
model_T2.add(Conv2D(64, (3, 3), activation='relu', input_shape=(28, 28,
                    padding='same'))
model_T2.add(MaxPooling2D((2, 2)))
model_T2.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model_T2.add(MaxPooling2D((2, 2)))
model_T2.add(Flatten())
model_T2.add(Dense(128, activation='relu'))
model_T2.add(Dense(10, activation='softmax'))

from keras.optimizers import Adam
custom_adam = Adam(lr=0.002)

model_T2.compile(loss='categorical_crossentropy', optimizer=custom_adam

# Setting a random state so that comparing the evaluation metrics is po
tf.set_random_seed(1)
np.random.seed(1)
model_T2.fit(x_train, y_train, epochs=10, batch_size=32, validation_dat

scores = model_T2.evaluate(x_test, y_test, verbose=1)
print("%s: %.2f%%" % (model_T1.metrics_names[1], scores[1]*100))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 125s 2ms/step - loss: 0.367
1 - accuracy: 0.8677 - val_loss: 0.2994 - val_accuracy: 0.8927
Epoch 2/10
60000/60000 [==============================] - 122s 2ms/step - loss: 0.240
9 - accuracy: 0.9103 - val_loss: 0.2551 - val_accuracy: 0.9085
Epoch 3/10
60000/60000 [==============================] - 126s 2ms/step - loss: 0.197
6 - accuracy: 0.9259 - val_loss: 0.2677 - val_accuracy: 0.9026
Epoch 4/10
60000/60000 [==============================] - 128s 2ms/step - loss: 0.164
6 - accuracy: 0.9387 - val_loss: 0.2484 - val_accuracy: 0.9119
Epoch 5/10
60000/60000 [==============================] - 123s 2ms/step - loss: 0.137
7 - accuracy: 0.9483 - val_loss: 0.2487 - val_accuracy: 0.9123
Epoch 6/10
60000/60000 [==============================] - 127s 2ms/step - loss: 0.114
8 - accuracy: 0.9554 - val_loss: 0.2977 - val_accuracy: 0.9116
Epoch 7/10
60000/60000 [==============================] - 127s 2ms/step - loss: 0.093
4 - accuracy: 0.9647 - val_loss: 0.3319 - val_accuracy: 0.9152
Epoch 8/10
60000/60000 [==============================] - 125s 2ms/step - loss: 0.079
5 - accuracy: 0.9700 - val_loss: 0.3438 - val_accuracy: 0.9174
Epoch 9/10
60000/60000 [==============================] - 123s 2ms/step - loss: 0.068
4 - accuracy: 0.9753 - val_loss: 0.3628 - val_accuracy: 0.9175
Epoch 10/10
60000/60000 [==============================] - 123s 2ms/step - loss: 0.061
6 - accuracy: 0.9770 - val_loss: 0.4392 - val_accuracy: 0.9158
10000/10000 [==============================] - 5s 455us/step
accuracy: 91.58%
```

Task 3: Change the type of optimizer or learning rate that you applied in the previous tasks, and see how these changes can influence model performance

In [15]:

```
#Testing custom sgd with different learning rates
lrate = (0.002, 0.004, 0.006, 0.008)
for lr in lrate:
    epochs = 10
    decay = lr/epochs
    sgd = SGD(lr=lr, momentum=0.7, decay=decay, nesterov=False) #Stocha
    # Compile model
    model_T2.compile(loss='categorical_crossentropy', optimizer=sgd, me
    model_T2.fit(x_train, y_train, epochs=epochs, batch_size=32, valida
    scores = model_T2.evaluate(x_test, y_test, verbose=1)
    print("learning rate: ", lr)
    print("%s: %.2f%%" % (model_T2.metrics_names[1], scores[1]*100))
```

```
10000/10000 [==============================] - 5s 464us/step
learning rate:  0.002
accuracy: 92.65%
10000/10000 [==============================] - 6s 600us/step
learning rate:  0.004
accuracy: 92.74%
10000/10000 [==============================] - 4s 426us/step
learning rate:  0.006
accuracy: 92.73%
10000/10000 [==============================] - 8s 834us/step
learning rate:  0.008
accuracy: 92.80%
```