Assignment 3

Summary of CNN architecture

```
def print_summary(name,layer_dim,kernel_size):
   if (name == "Conv2D"):
      print(name,"                                                                    ",(None,layer_dim),"
",(layer_dim[0]*(kernel_size*(kernel_size*kernel_size)+1)))
   elif (name == "Maxpooling"):
      print(name,"          ",(None,layer_dim),"             ","0")
   elif (name == "Flatten"):
      print(name,"               ",(None,np.prod(layer_dim)),"                ","0")
   elif(name == "Dense"):
      print(name,"               ",(None,kernel_size),"               ",(kernel_size*(np.prod(layer_dim)+1)))
```

Model Summary

Layer (type)	Output Shape	Param #
Conv2D	(None, (2, 26, 26))	56
Maxpooling	(None, (2, 25, 25))	0
Conv2D	(None, (2, 23, 23))	56
Maxpooling	(None, (2, 22, 22))	0
Flatten	(None, 968)	0
Dense	(None, 50)	48450
Dense	(None, 10)	510

Training dataset with learning rate : 0.001,epoch 20 and mu:1e-6

```
def sgd_momentum(nnet, X_train, y_train, minibatch_size, epoch, learning_rate, mu=1e-6,
            verbose=True, X_test=None, y_test=None, nesterov=True):
   Loss = []
   ValLoss = []
   Epoch_list = []
   Train_Acc = []
   Test_Acc = []
   minibatches = get_minibatches(X_train, y_train, minibatch_size)
   for i in range(epoch):
      loss = 0
      velocity = []
      for param_layer in nnet.params:
         p = [np.zeros_like(param) for param in list(param_layer)]
         velocity.append(p)

      if verbose:
         print("Epoch {0}".format(i + 1))
```

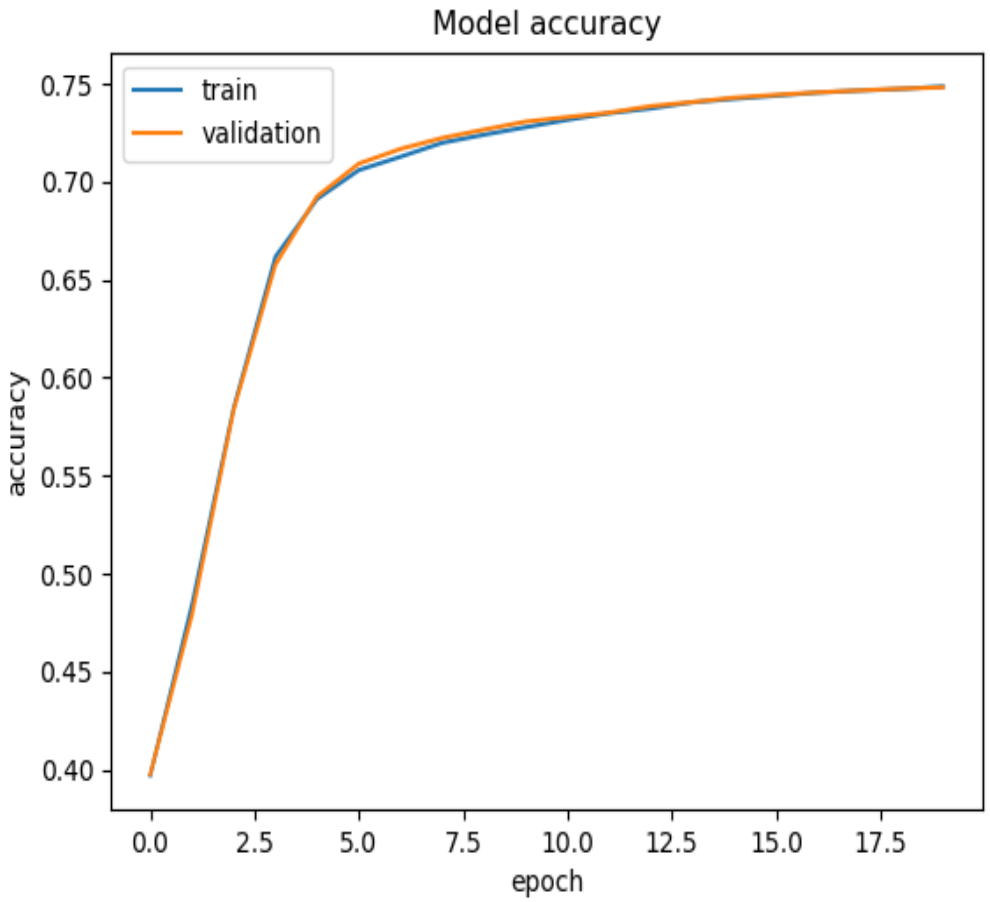
```
        for X_mini, y_mini in minibatches:

            if nesterov:
                for param, ve in zip(nnet.params, velocity):
                    for i in range(len(param)):
                        param[i] += mu * ve[i]

            loss, grads = nnet.train_step(X_mini, y_mini)
            momentum_update(velocity, nnet.params, grads,
                            learning_rate=learning_rate, mu=mu)

        if verbose:
            m_train = X_train.shape[0]
            m_test = X_test.shape[0]
            y_train_pred = np.array([], dtype="int64")
            y_test_pred = np.array([], dtype="int64")
            for i in range(0, m_train, minibatch_size):
                X_tr = X_train[i:i + minibatch_size, :, :, :]
                y_tr = y_train[i:i + minibatch_size, ]
                y_train_pred = np.append(y_train_pred, nnet.predict(X_tr))
            for i in range(0, m_test, minibatch_size):
                X_te = X_test[i:i + minibatch_size, :, :, :]
                y_te = y_test[i:i + minibatch_size, ]
                y_test_pred = np.append(y_test_pred, nnet.predict(X_te))
            _,val_loss = nnet.evaluate(X_test,y_test)
            train_acc = accuracy(y_train, y_train_pred)
            test_acc = accuracy(y_test, y_test_pred)
            print("Loss = {0} | Training Accuracy = {1} | Test Accuracy = {2} | Test loss = {3}".format(
                loss, train_acc, test_acc,val_loss))
            Loss.append(loss)
            Epoch_list.append(epoch)
            Train_Acc.append(train_acc)
            Test_Acc.append(test_acc)
            ValLoss.append(val_loss)
    plot_graph(Loss,Epoch_list,Train_Acc,Test_Acc,ValLoss)
    return nnet

Epoch 1
Loss = 2.046145516391362 | Training Accuracy = 0.39720833333333333 | Test Accuracy =
0.39766666666666667 | Test loss = 1.9126154557705803
Epoch 2
Loss = 1.8611774476174425 | Training Accuracy = 0.4841875 | Test Accuracy =
0.47908333333333336 | Test loss = 1.7203384532077204
Epoch 3
Loss = 1.712488448204334 | Training Accuracy = 0.5845625 | Test Accuracy = 0.5835833333333333
| Test loss = 1.56262759942373
Epoch 4
```

Loss = 1.6055005572085257 | Training Accuracy = 0.6617291666666667 | Test Accuracy = 0.6576666666666666 | Test loss = 1.4452736808104012
Epoch 5
Loss = 1.5216567335893836 | Training Accuracy = 0.6912083333333333 | Test Accuracy = 0.6925833333333333 | Test loss = 1.3549040280474773
Epoch 6
Loss = 1.454580206241941 | Training Accuracy = 0.7059375 | Test Accuracy = 0.70925 | Test loss = 1.283418823191751
Epoch 7
Loss = 1.3996092667976159 | Training Accuracy = 0.71275 | Test Accuracy = 0.71675 | Test loss = 1.2248078505192
Epoch 8
Loss = 1.3531459644769197 | Training Accuracy = 0.719875 | Test Accuracy = 0.7223333333333334 | Test loss = 1.175482212757702
Epoch 9
Loss = 1.3129637287573634 | Training Accuracy = 0.7239791666666666 | Test Accuracy = 0.7265833333333334 | Test loss = 1.1331993838339451
Epoch 10
Loss = 1.27765304760777 | Training Accuracy = 0.7278125 | Test Accuracy = 0.7306666666666667 | Test loss = 1.096432220468313
Epoch 11
Loss = 1.2462657901958614 | Training Accuracy = 0.7315625 | Test Accuracy = 0.733 | Test loss = 1.064133203653139
Epoch 12
Loss = 1.2181205782750786 | Training Accuracy = 0.7349375 | Test Accuracy = 0.73525 | Test loss = 1.035561460134603
Epoch 13
Loss = 1.1927583929759784 | Training Accuracy = 0.7373541666666666 | Test Accuracy = 0.7385833333333334 | Test loss = 1.0102039818181139
Epoch 14
Loss = 1.1698257087396255 | Training Accuracy = 0.7405416666666667 | Test Accuracy = 0.7406666666666667 | Test loss = 0.9876676698394494
Epoch 15
Loss = 1.1491072695519269 | Training Accuracy = 0.7422916666666667 | Test Accuracy = 0.7429166666666667 | Test loss = 0.9676776487751666
Epoch 16
Loss = 1.1305428241351942 | Training Accuracy = 0.743875 | Test Accuracy = 0.7443333333333333 | Test loss = 0.9499544658028247
Epoch 17
Loss = 1.1137998971289875 | Training Accuracy = 0.7454791666666667 | Test Accuracy = 0.7455833333333334 | Test loss = 0.9342542402402927
Epoch 18
Loss = 1.0986048361702705 | Training Accuracy = 0.7465208333333333 | Test Accuracy = 0.7465833333333334 | Test loss = 0.9202863242850735
Epoch 19
Loss = 1.084653635359506 | Training Accuracy = 0.7473541666666667 | Test Accuracy = 0.7475 | Test loss = 0.9077289575841132
Epoch 20

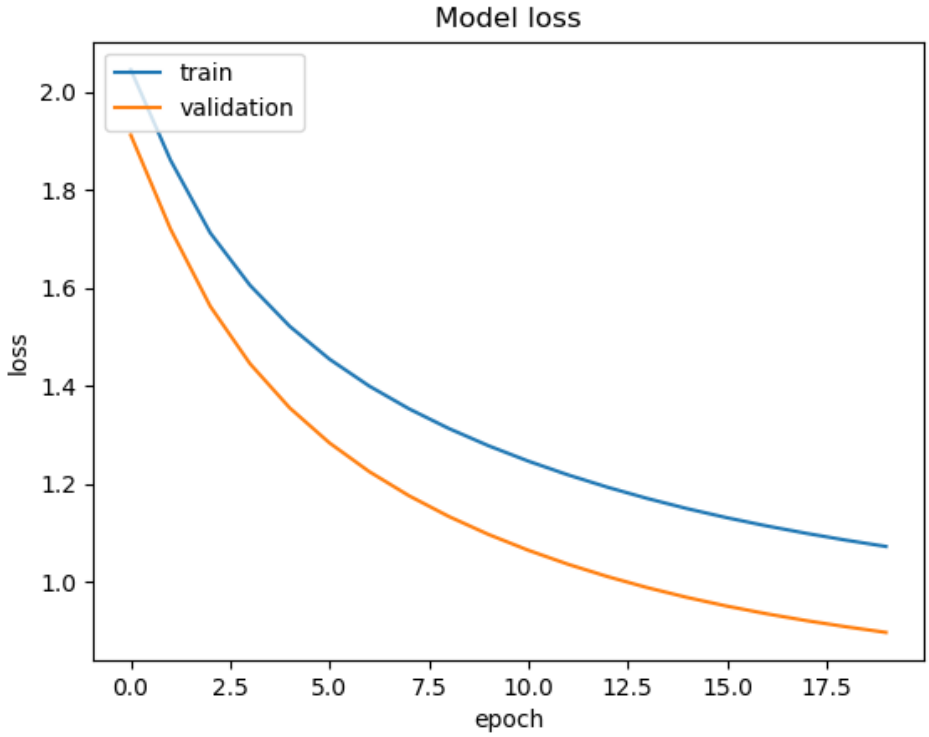
Loss = 1.0718484548076261 | Training Accuracy = 0.7485833333333334 | Test Accuracy = 0.7481666666666666 | Test loss = 0.8962721554839845

Epoch –Accuracy plot for train and validation dataset

```
def plot_graph(loss,epoch,train_acc,test_acc,val_loss)
    plt.plot(train_acc)
    plt.plot(test_acc)
    plt.title('Model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()
    plt.plot(loss)
    plt.plot(val_loss)
    plt.title('Model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()
```



Epoch – Loss plot for train and validation dataset



Classification Report of test data

```
test_pred_y = cnn.predict(test_x)
print(classification_report(test_y, test_pred_y))
cm = confusion_matrix(test_y, test_pred_y)
y_classes =
["Tshirt/top","Trouser","Pullover","Dress","Coat","Sandal","Shirt","Sneaker","Bag","Ankle boot"]
sns.heatmap(cm, annot=True, fmt='d',xticklabels=y_classes, yticklabels=y_classes)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

	Precision	Recall	F1-score	Support
0	0.74	0.77	0.75	1000
1	0.94	0.95	0.94	1000
2	0.58	0.61	0.60	1000
3	0.71	0.84	0.77	1000
4	0.54	0.56	0.55	1000
5	0.83	0.83	0.83	1000

6	0.37	0.27	0.31	1000
7	0.87	0.82	0.84	1000
8	0.87	0.82	0.85	1000
9	0.85	0.93	0.89	1000
Weighted Avg	0.73	0.74	0.73	10000

Confusion Matrix of Test data