[CM4]

From Model 2, we take the encoding of second FCC layer and use that for this exercise. Since there are 50 neurons in this layer, the dimension of data at this layer is 50. We reduce that to 2 dimensions using PCA for the purpose of data visualization.

In [ ]:
```
loader2= build_dataloader(data = x_train_data, label = y_train_data, batc
input_embedding = []
input_label = []
for x,y in zip(loader2[0],loader2[1]):
  x = x.float()
  x = x.cuda()
  y = y.numpy()
  z = model2.forward2(x)
  input_embedding.append(z.cpu().numpy()[0])
  input_label.append(y)
```

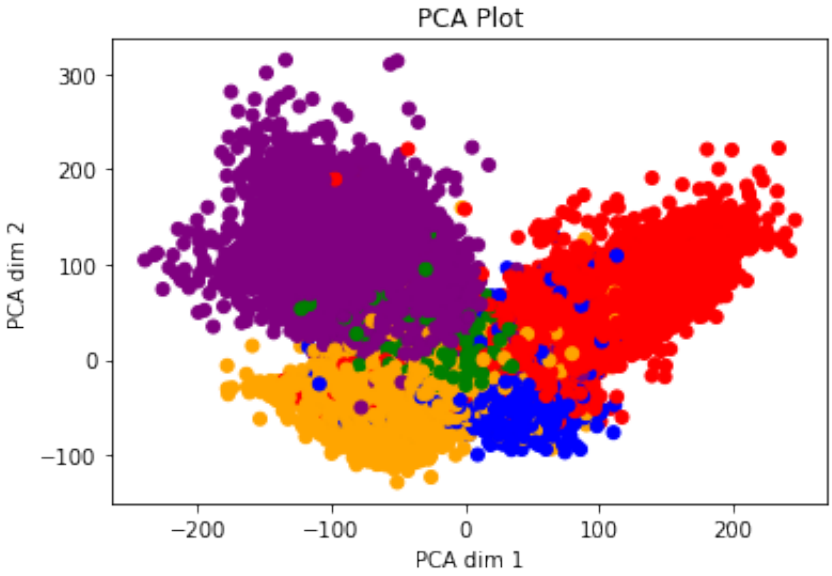
In [ ]:
```
ip_emb = []
ip_label = []
for ip,y in zip(input_embedding, input_label):
  if not(np.isnan(ip).any()):
    ip_emb.append(ip)
    ip_label.append(y)
```

In [ ]:
```
ip_label = np.array(ip_label)
ip_emb = np.array(ip_emb)
```

PCA

In [ ]:
```
embedded_ip = np.array(ip_emb)
embedded_label = np.array(ip_label)
import numpy as np
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(embedded_ip)
x_2d = pca.transform(embedded_ip)
colors = ['red','green','blue','purple', 'orange']
plt.scatter(x_2d[:,0], x_2d[:,1],c=ip_label, cmap=matplotlib.colors.Liste
plt.title("PCA Plot")
plt.xlabel("PCA dim 1")
plt.ylabel("PCA dim 2")
```

Out[ ]:
```
Text(0, 0.5, 'PCA dim 2')
```



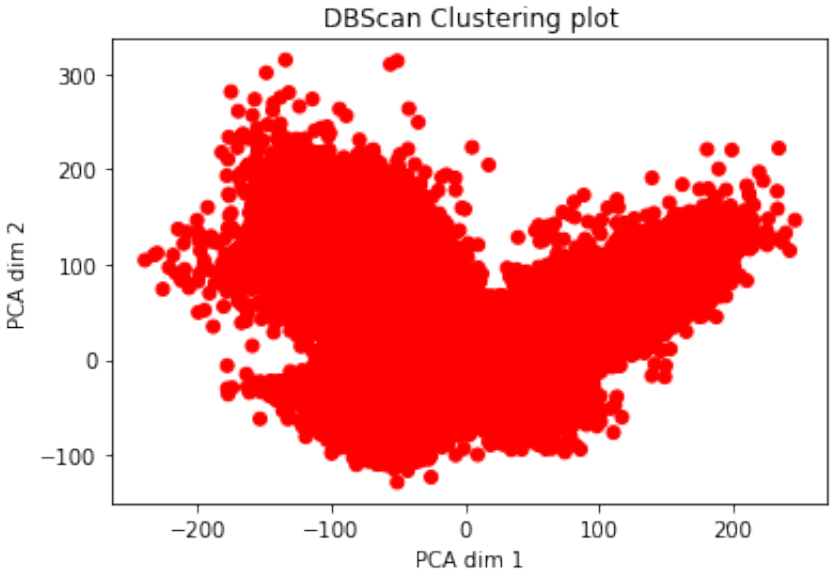
From the plot, it is evident that at this stage, the data is fairly clustered based on their class. One can note that the clusters being formed are of sperical in nature. This actually shows that this data can effectively be clustered by algorithms like k-means clustering.

DBSCAN

In [ ]:
```
from sklearn.cluster import DBSCAN
clustering_labels = DBSCAN(eps=4, min_samples=15).fit_predict(embedded_ip
```

In [ ]:
```
plt.scatter(x_2d[:,0], x_2d[:,1],c=clustering_labels, cmap=matplotlib.col
plt.title("DBScan Clustering plot")
plt.xlabel("PCA dim 1")
plt.ylabel("PCA dim 2")
```

Out[ ]:
```
Text(0, 0.5, 'PCA dim 2')
```



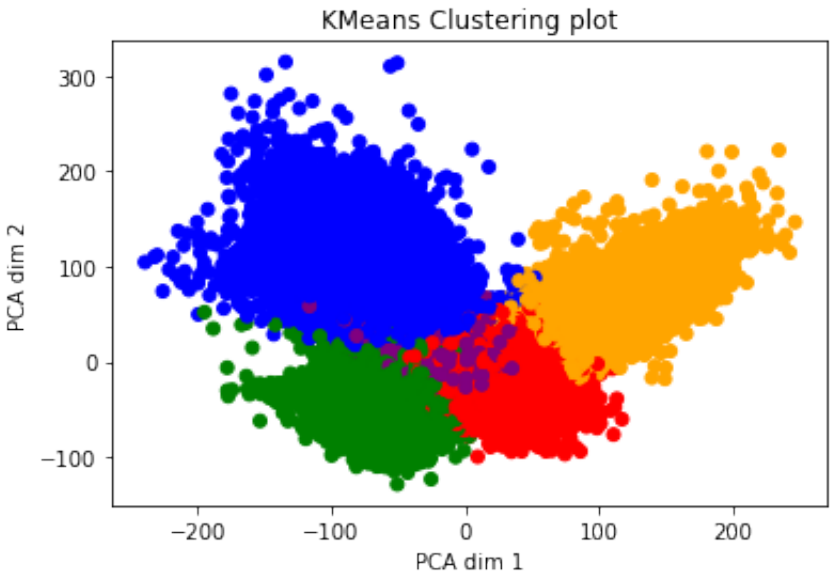
DBScan is a clustering algorithm that is very good at forming non-sperical shaped clusters (clusters of unusual shape). This algorithm depends mainly on density. Since our original dataset forms sperical cluster at the point where we take the encoded data, and since the points appear to be of similar density, we are getting poor performance.

KMeans

In [ ]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=5, random_state=0).fit(ip_emb)
```

In [ ]:
```
plt.scatter(x_2d[:,0], x_2d[:,1],c=kmeans.labels_, cmap=matplotlib.colors
plt.title("KMeans Clustering plot")
plt.xlabel("PCA dim 1")
plt.ylabel("PCA dim 2")
```

Out[ ]:
```
Text(0, 0.5, 'PCA dim 2')
```



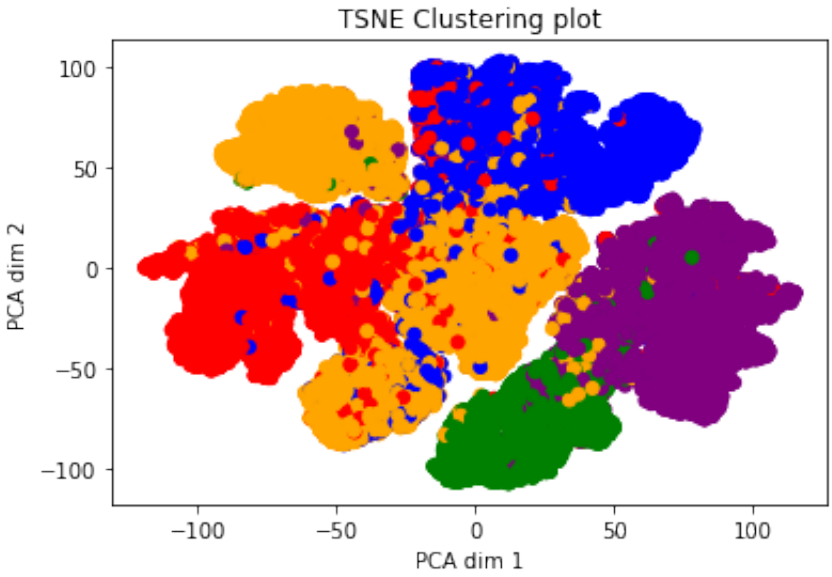
Since the original data at the point where we have taken forms sperical clusters, k-means clustering has decently clustered the points into its classes.

TSNE

In [ ]:
```
from sklearn.manifold import TSNE
X_embedded = TSNE(n_components=2, learning_rate='auto',init='random').fit
```

In [ ]:
```
plt.scatter(X_embedded[:,0], X_embedded[:,1],c=ip_label, cmap=matplotlib.
plt.title("TSNE Clustering plot")
plt.xlabel("PCA dim 1")
plt.ylabel("PCA dim 2")
```

Out[ ]:
```
Text(0, 0.5, 'PCA dim 2')
```

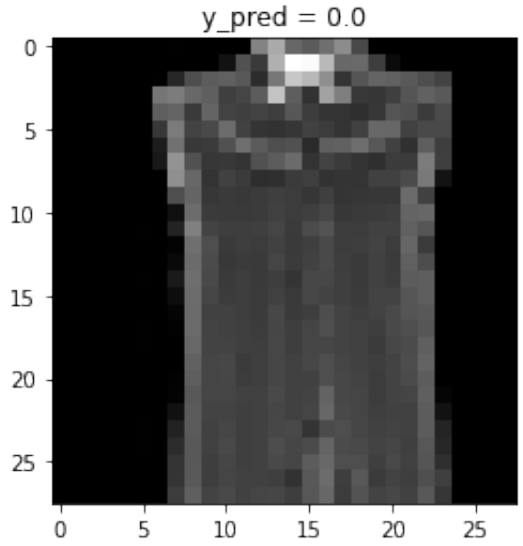


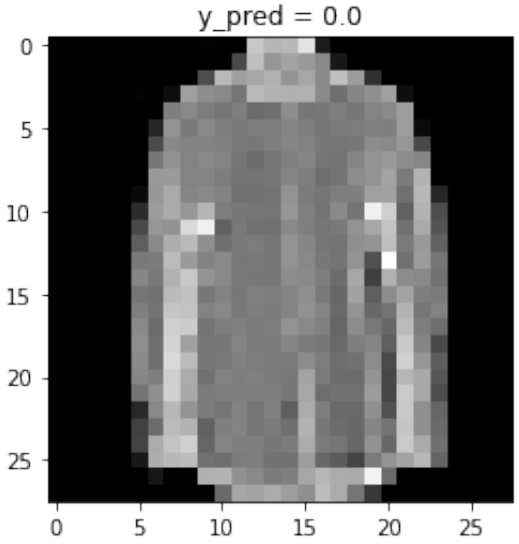
TSNE is mainly used to effectively visualize data of higher dimension. We can see that clusters are well separated and the degree of separation is better than that of PCA.

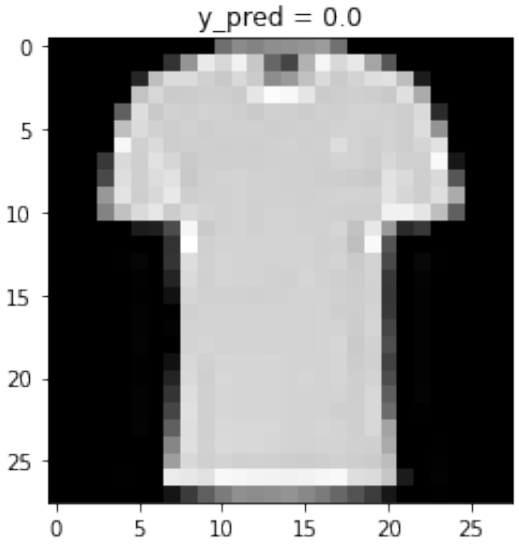
From the clustering, we print out some of the labels to figure out the meaning behind the labels.

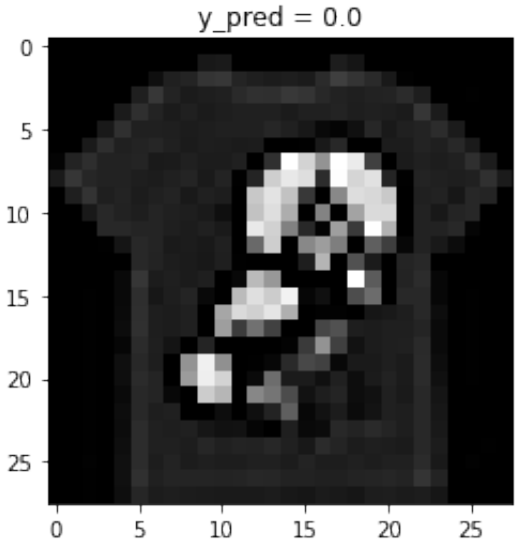
In [ ]:
```
print("Label 0 images")
for i in range(5):
  show_data_2(pred_0.values[i,0:-1], pred_0.values[i,-1])
```

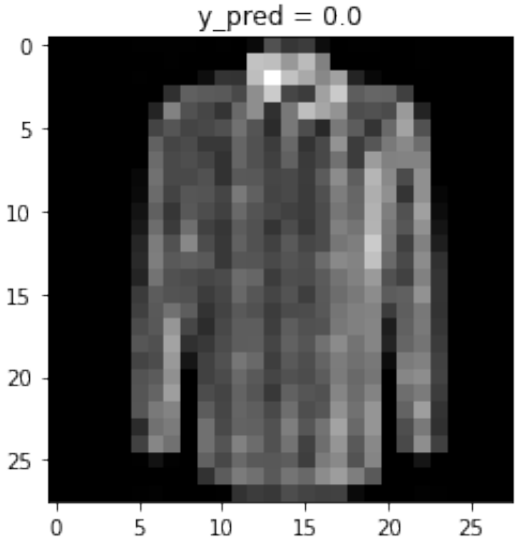
```
Label 0 images
```




y_pred = 0.0
0
5
10
15
20
25
0
5
10
15
20
25

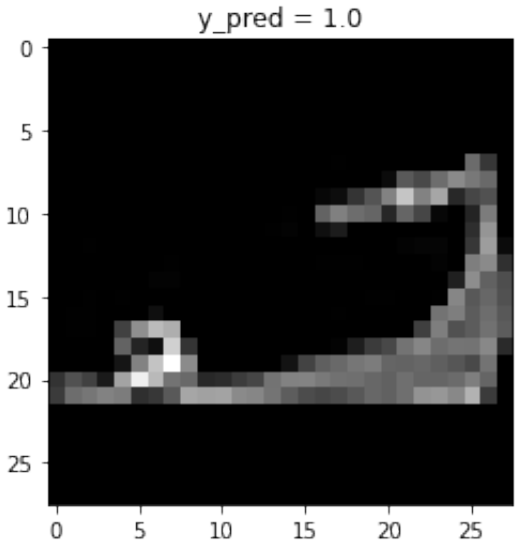

y_pred = 0.0
0
5
10
15
20
25
0
5
10
15
20
25

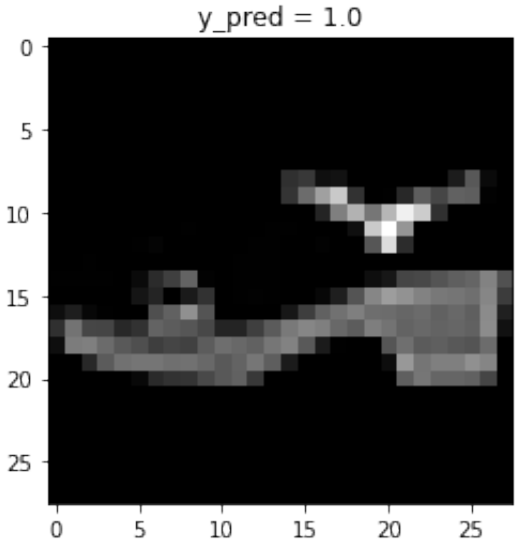

y_pred = 0.0
0
5
10
15
20
25
0
5
10
15
20
25

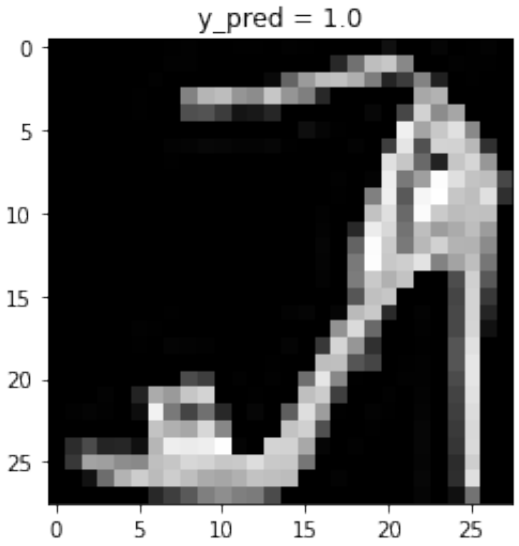


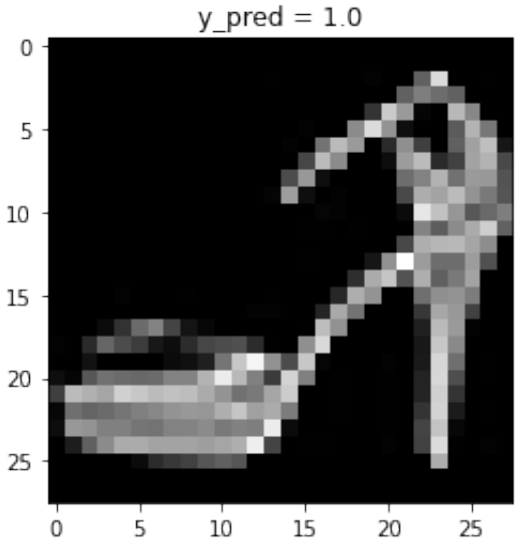
In [ ]:
```
print("Label 1 images")
for i in range(5):
  show_data_2(pred_1.values[i,0:-1], pred_1.values[i,-1])
```

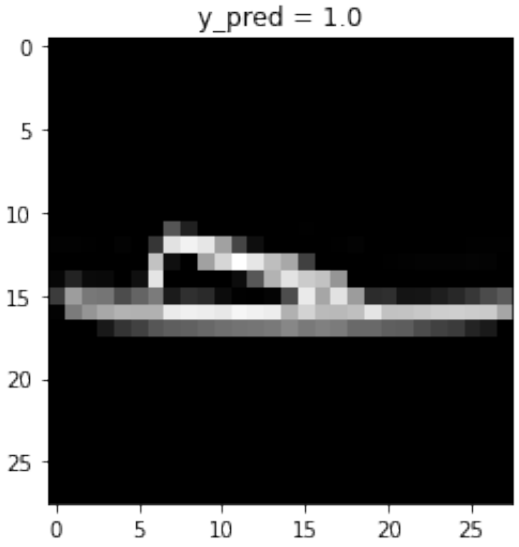
```
Label 1 images
```





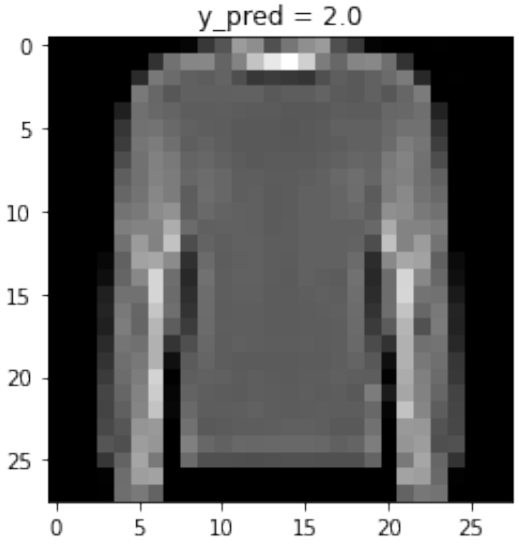


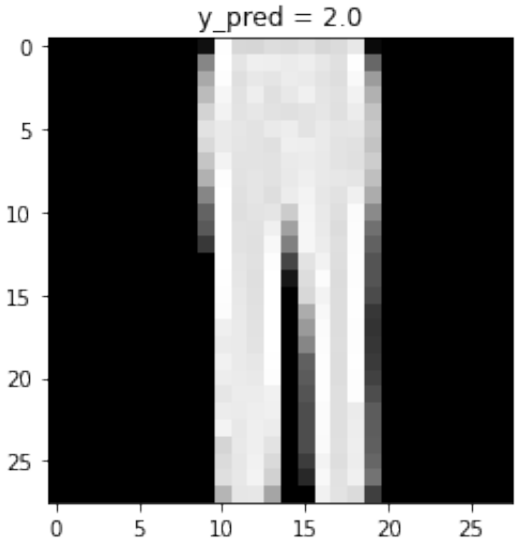


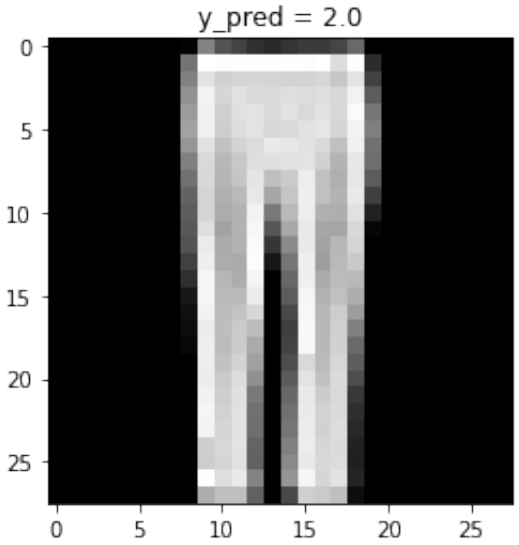


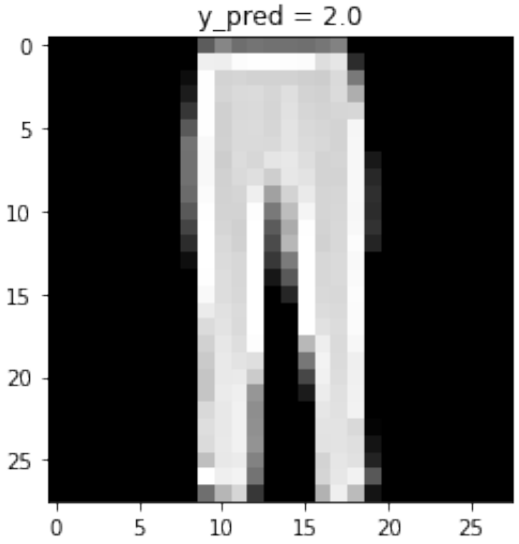
In [ ]:
```
print("Label 2 images")
for i in range(5):
  show_data_2(pred_2.values[i,0:-1], pred_2.values[i,-1])
```

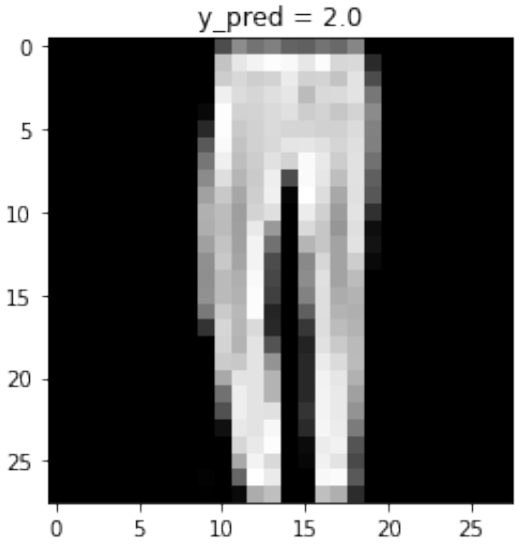
```
Label 2 images
```


y_pred = 2.0


y_pred = 2.0

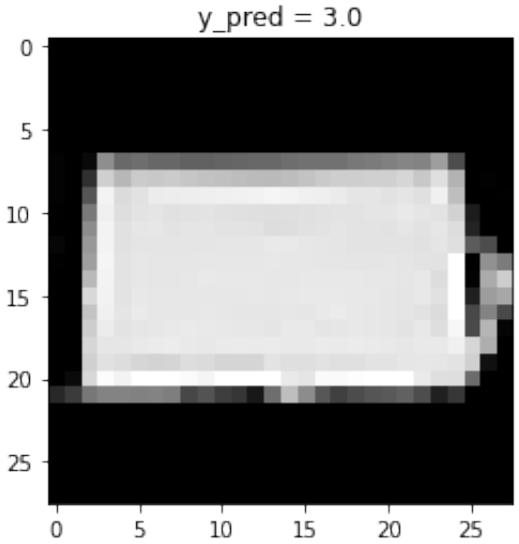

y_pred = 2.0

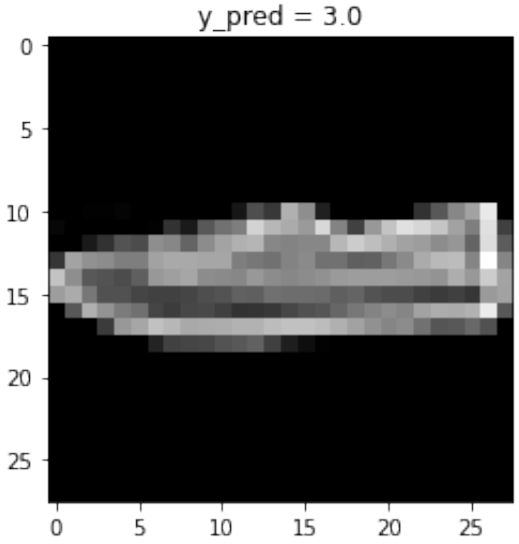


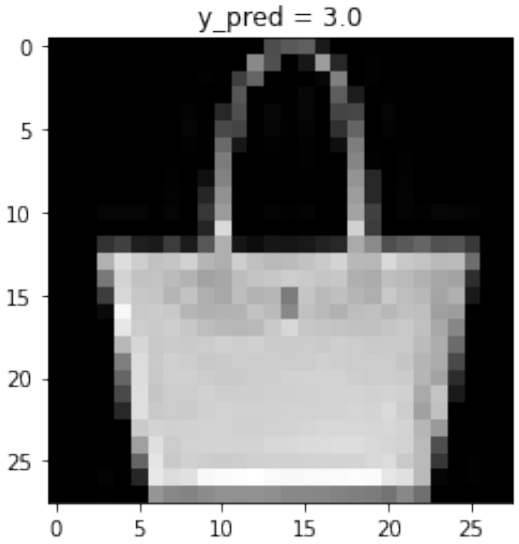


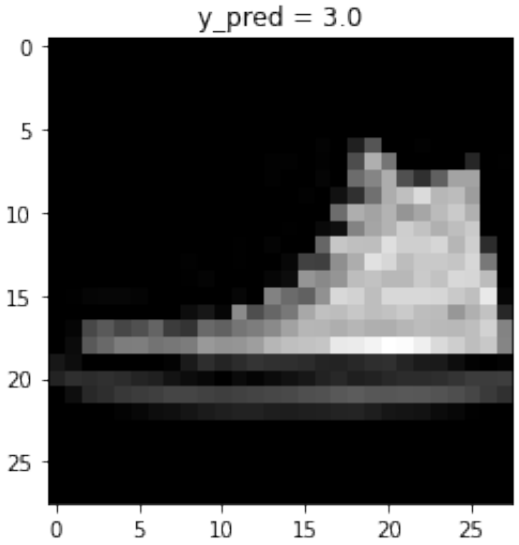
In [ ]:
```
print("Label 3 images")
for i in range(5):
  show_data_2(pred_3.values[i,0:-1], pred_3.values[i,-1])
```

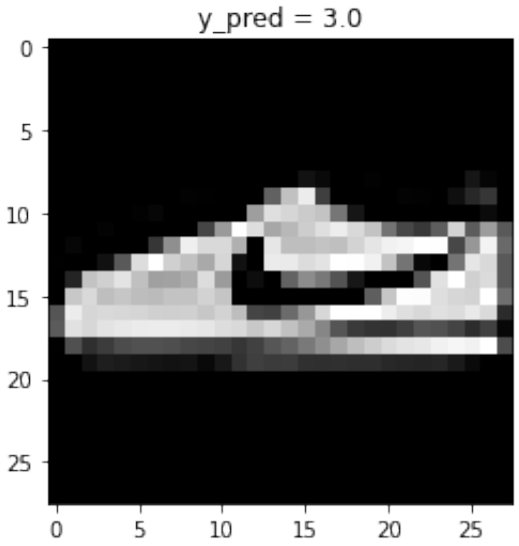
```
Label 3 images
```




y_pred = 3.0
0
5
10
15
20
25
0
5
10
15
20
25

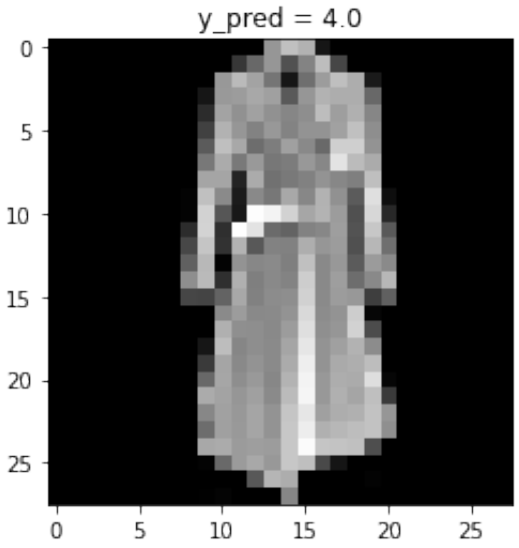

y_pred = 3.0
0
5
10
15
20
25
0
5
10
15
20
25

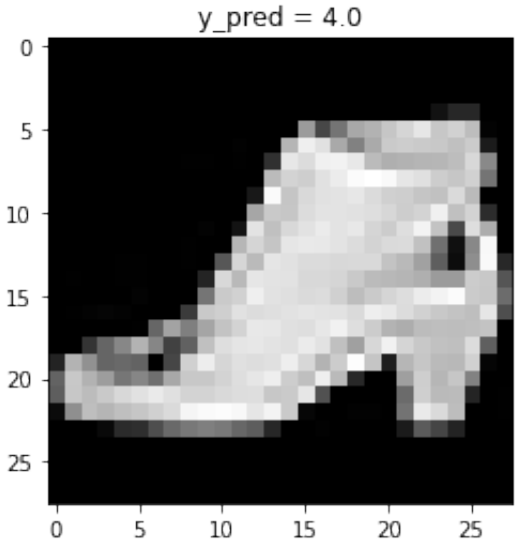

y_pred = 3.0
0
5
10
15
20
25
0
5
10
15
20
25

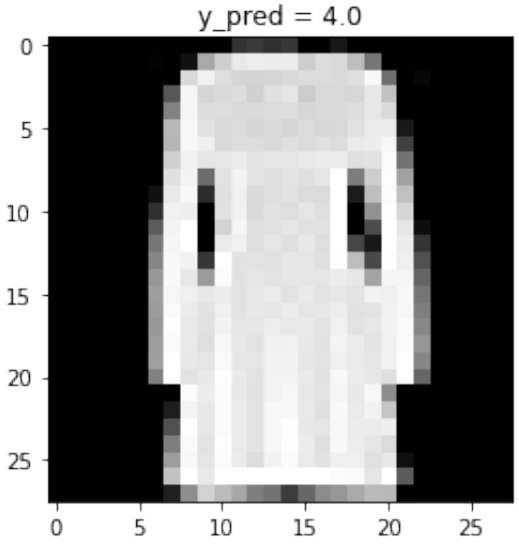


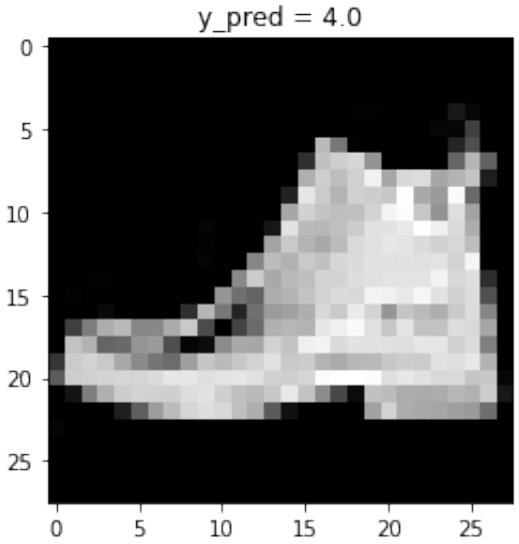
In [ ]:
```
print("Label 4 images")
for i in range(5):
  show_data_2(pred_4.values[i,0:-1], pred_4.values[i,-1])
```

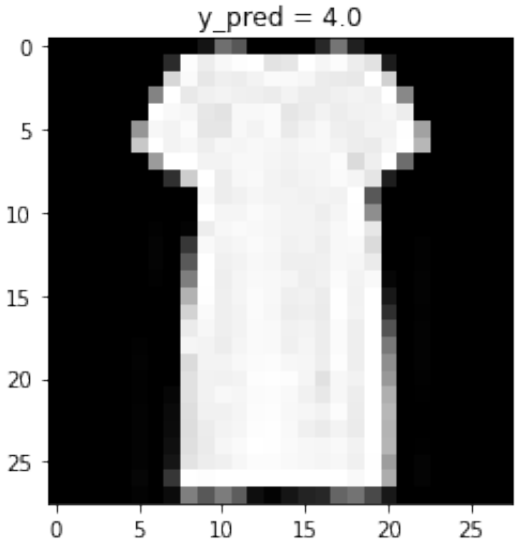
```
Label 4 images
```






y_pred = 4.0


y_pred = 4.0


y_pred = 4.0

Group 0 - Tops, T-Shirts, Pullovers and Coats
Group 1 - Sandles and Heels
Group 2 - Trousers, One Shirt that is incorrectly classified
Group 3 - Clutch,Bags, Shoes, Sneakers
Group 4 - Boots and Dresses

Clustering does not utlize labels however it groups similar types of objects together to a certain extent. We can see similar behaviour here.

REFERENCES:

https://pytorch.org/docs/stable/generated/torch.nn.functional.one_hot.html
https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html
https://pytorch.org/docs/stable/generated/torch.optim.Adam.html
https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html
https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html