1. Setup

Download the Fashion-MNIST dataset (https://github.com/zalandoresearch/fashion-mnist). Normalize the data such that pixel values are floats in [0, 1], and use the normalized data for all of the following questions. Use the train-test split provided by Fashion-MNIST.

In [89]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import numpy as np
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
import seaborn as sns
np.random.seed(33)
```

In [76]:
```
# upload data and split to training/testing sets
(x_train , y_train), (x_test , y_test) = datasets.fashion_mnist.load_data()

# convert pixel values to floats
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# normalize
x_train /= 255.0
x_test /= 255.0

# confirm the normalization
print("Minimum pixel value after normalization:", x_test.min())
print("Maximum pixel value after normalization:", x_test.max())
```

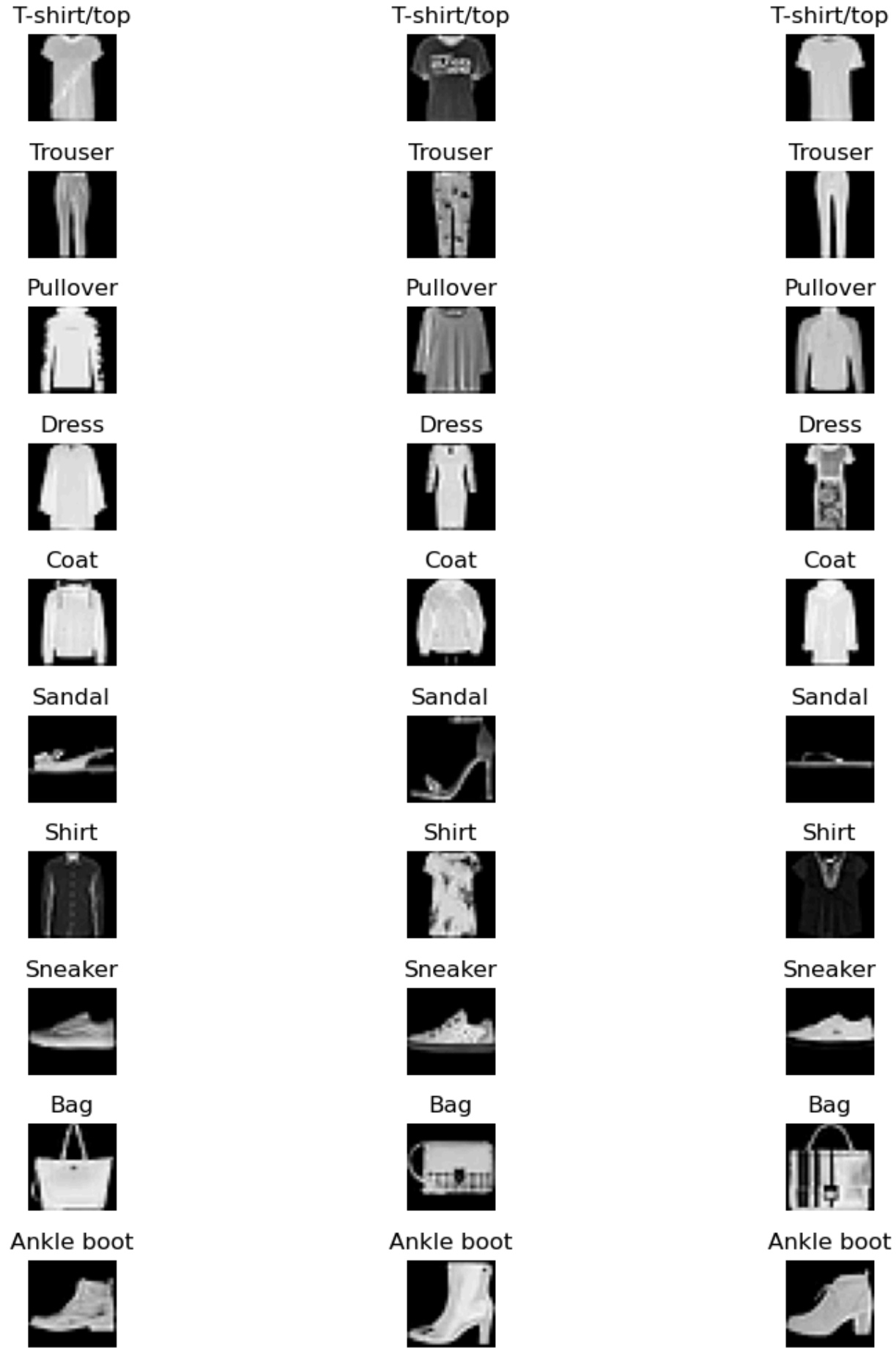
```
Minimum pixel value after normalization: 0.0
Maximum pixel value after normalization: 1.0
```

2. Exploratory Data Analysis

1. Plot a random sample of 3 images from each class in the training set.

In [66]:
```
# define the class names
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal'
# count number of classes
num_classes = len(class_names)
# prep dimensions of the plot
fig, axes = plt.subplots(num_classes, 3, figsize=(10, 10))
# iterate through each class
for i, class_name in enumerate(class_names):
    class_indices = np.where(y_train == i)[0]
    random_indices = np.random.choice(class_indices, 3, replace=False)

    # plot random samples from the each class
    for j, idx in enumerate(random_indices):
        axes[i, j].imshow(x_train[idx], cmap='gray')
        axes[i, j].set_title(class_name)
```

```
        axes[i, j].axis('off')
# formatting specifications
plt.tight_layout()
plt.show()
```



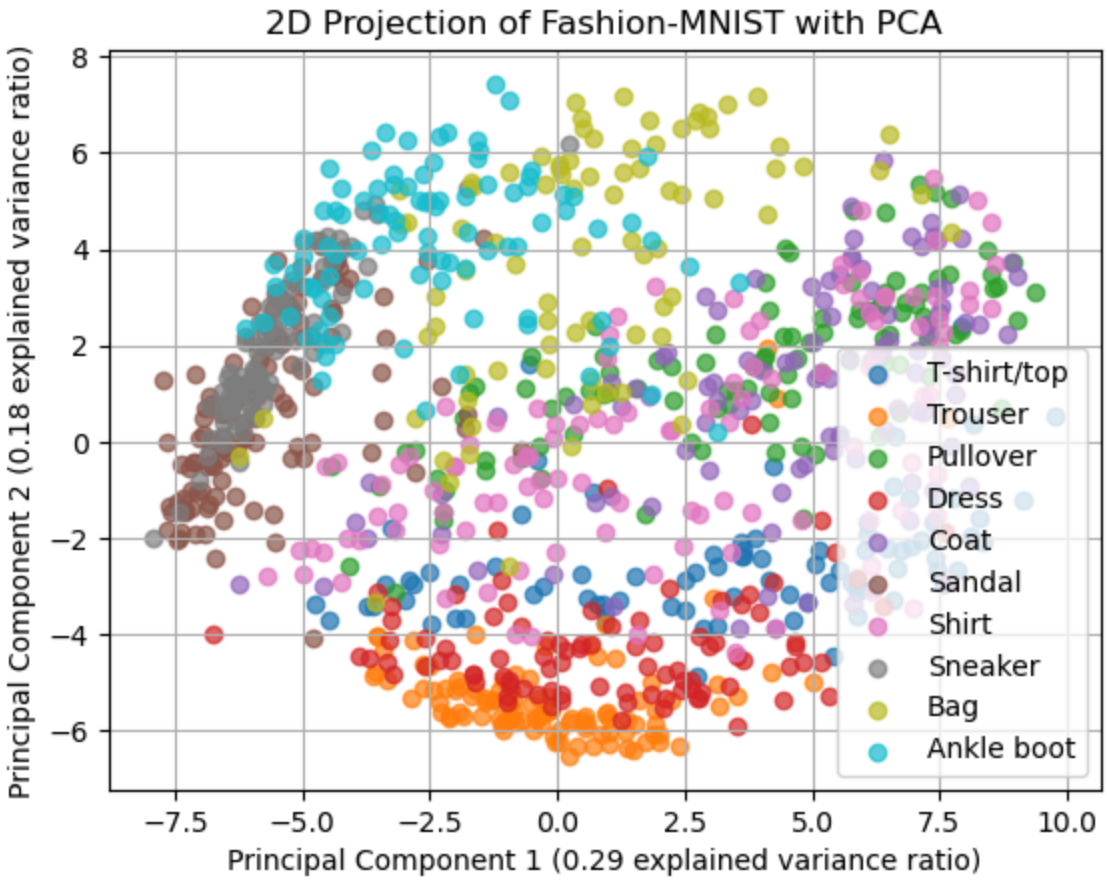
2. Fit a PCA model to the training data.

3. Plot a random sample of 1000 examples from the training split in the 2D space defined by the first two principal components. Color the points by their ground truth class.

4. Include the explained variance ratio of the first two principal components on the axis labels.

In [44]:
```
# fit the PCA to the training data
x_train_flat = x_train.reshape(x_train.shape[0], -1)
pca = PCA()
x_train_pca = pca.fit_transform(x_train_flat)
# extract the explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_

# randomly sample 1000 examples
random_indices = np.random.choice(x_train_pca.shape[0], 1000, replace=False)
sample_data = x_train_pca[random_indices]
sample_labels = y_train[random_indices]
```

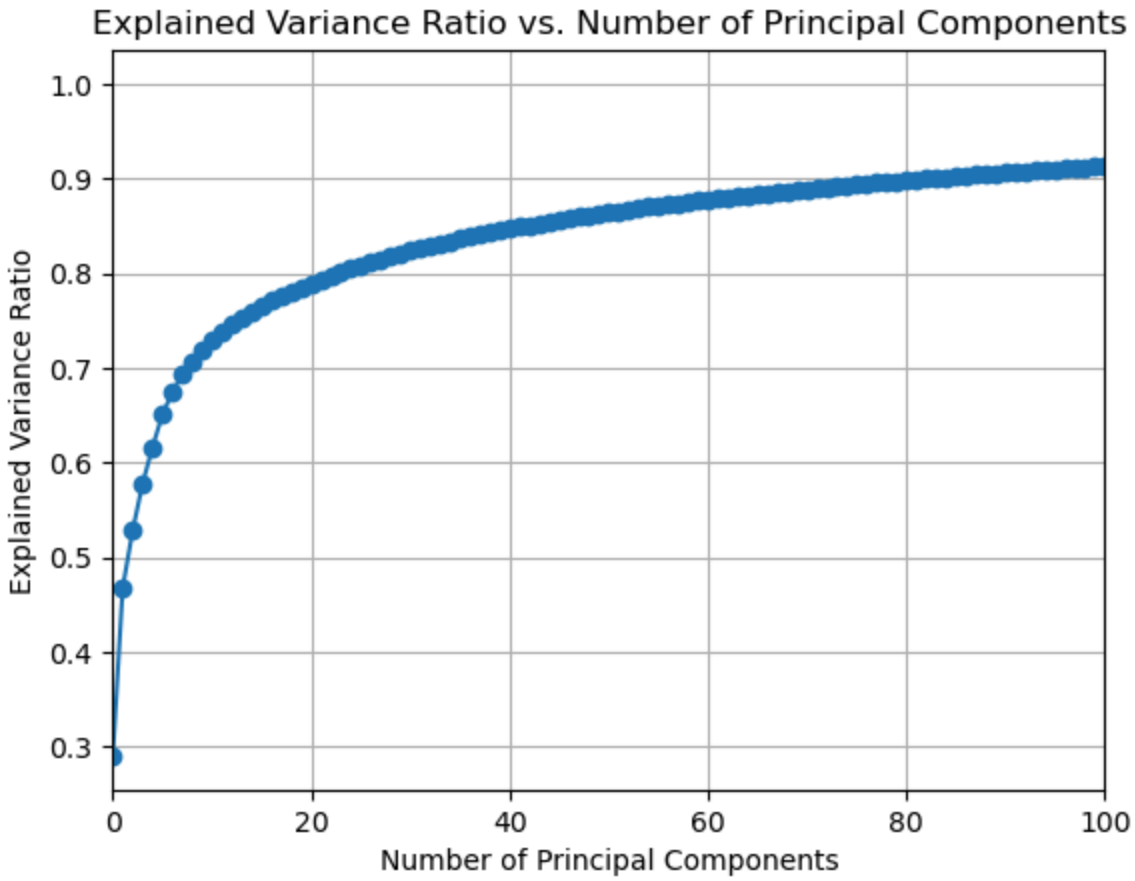
In [45]:
```
# plot the data points in 2D space defined by the first two principal componen
for label in np.unique(sample_labels):
    plt.scatter(sample_data[sample_labels == label, 0],
                sample_data[sample_labels == label, 1],
                label=class_names[label],
                alpha=0.7)
plt.title('2D Projection of Fashion-MNIST with PCA')
plt.xlabel(f'Principal Component 1 ({explained_variance_ratio[0]:.2f} explaine
plt.ylabel(f'Principal Component 2 ({explained_variance_ratio[1]:.2f} explaine
plt.legend()
plt.grid(True)
```



5. Create a line plot showing explained variance ratio as a function of the number of principal components.

In [46]:
```
# plot the explained variance ratio as a function of the # of principal compone
plt.plot(np.cumsum(explained_variance_ratio), marker='o', linestyle='-')
plt.xlabel('Number of Principal Components')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio vs. Number of Principal Components')
plt.grid(True)
# added limit to # of principal components after initially observing the graph
plt.xlim(0,100)
```

Out[46]:
```
(0.0, 100.0)
```



3. Clustering

K-means:

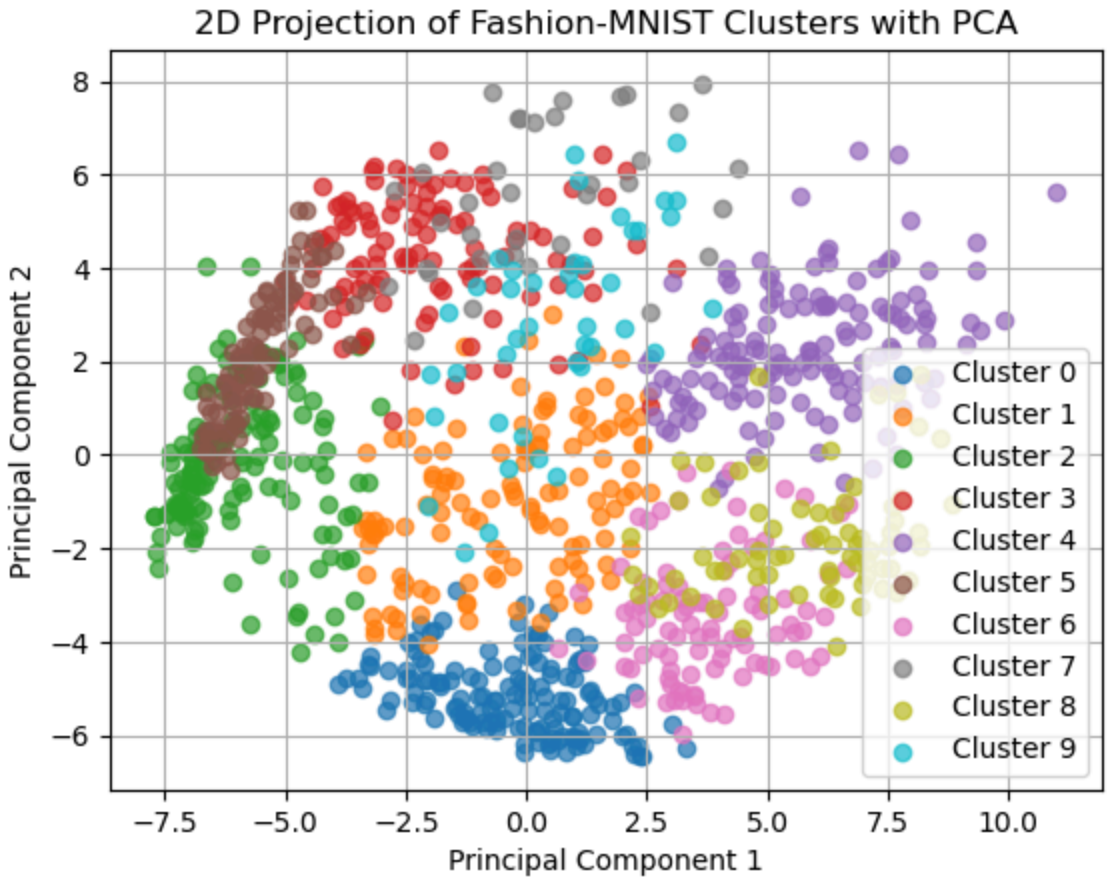
1. Fit a K-means model to the training data with k=10 clusters. Use your plot of the explained variance ratio (the last step in the previous section) to select a number of principal components to use for clustering. Justify your choice. Use the principal components you selected as the input to the K-means model.

In [53]:
```
pca_selected = PCA(n_components = 9)
x_train_pca = pca_selected.fit_transform(x_train_flat)

# fit k-means model with 9 principal components
kmeans = KMeans(n_clusters=10, random_state=33, n_init = 10)
kmeans.fit(x_train_pca)

cluster_labels = kmeans.labels_
cluster_counts = np.bincount(cluster_labels)

print("Number of Samples in Each Cluster:", cluster_counts)
```

```
Number of Samples in Each Cluster: [6727 9629 9177 2546 4380 2842 7273 7705 22
90 7431]
```

2. Plot a random sample of 1000 examples from the training split in the 2D space defined by the first two principal components. Color the points by their cluster assignment.

In [62]:
```
# randomly sample 1000 examples
random_indices = np.random.choice(x_train_pca.shape[0], 1000, replace=False)
sample_data = x_train_pca[random_indices]
sample_labels = y_train[random_indices]

kmeans.fit(sample_data)

cluster_labels = kmeans.labels_
cluster_counts = np.bincount(cluster_labels)

# plot the data points in 2D space defined by the first two principal componen

for label in np.unique(cluster_labels):
    plt.scatter(sample_data[cluster_labels == label, 0],
                sample_data[cluster_labels == label, 1],
                label=f'Cluster {label}',
                alpha=0.7)
plt.title('2D Projection of Fashion-MNIST Clusters with PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid(True)
plt.show()
```

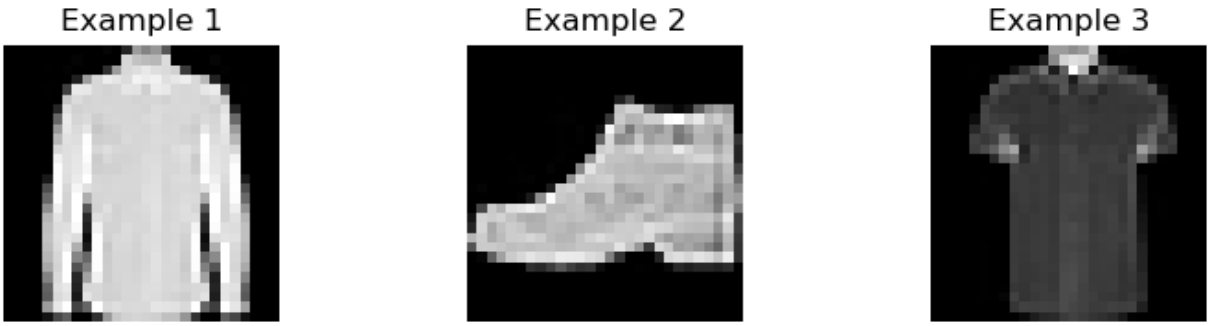


3. Show 3 examples from each cluster.

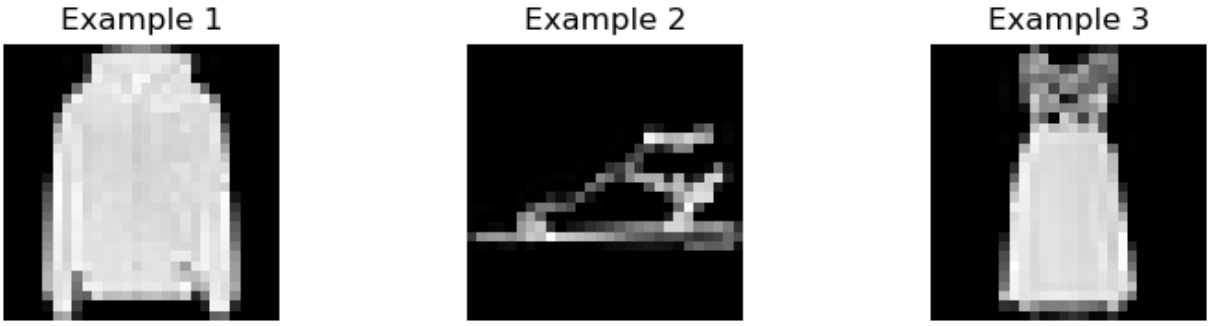
In [71]:
```
# iterate through each cluster, randomly select 3 examples, and print them as
for label in np.unique(cluster_labels):
    cluster_indices = np.where(cluster_labels == label)[0]
    random_indices = np.random.choice(cluster_indices, 3, replace=False)
```

```
    print(f"Cluster {label}:")

    # Create a single figure with three subplots in the same row
    fig, axes = plt.subplots(1, 3, figsize=(8, 2))
    for i, idx in enumerate(random_indices):
        axes[i].imshow(x_train[idx], cmap='gray')
        axes[i].set_title(f"Example {i+1}")
        axes[i].axis('off')
    plt.tight_layout()
    plt.show()
```

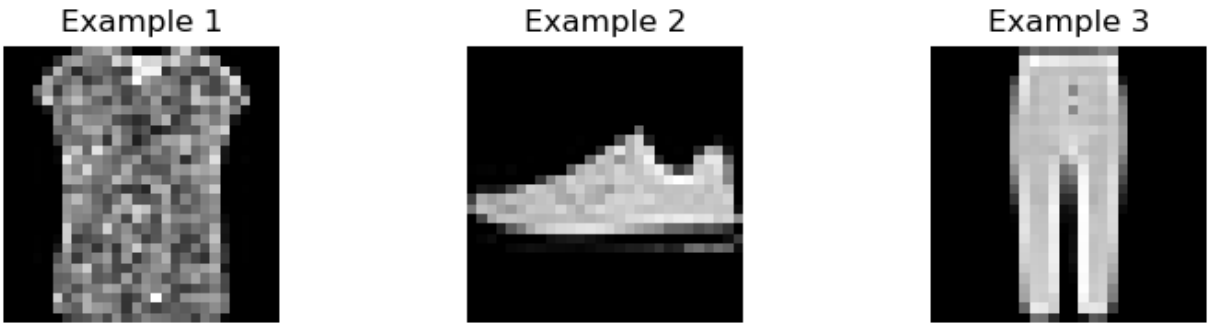
Cluster 0:



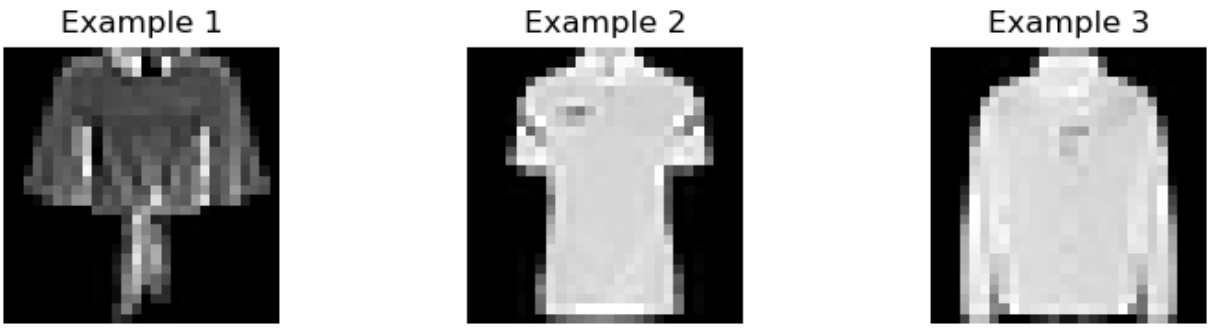
Cluster 1:



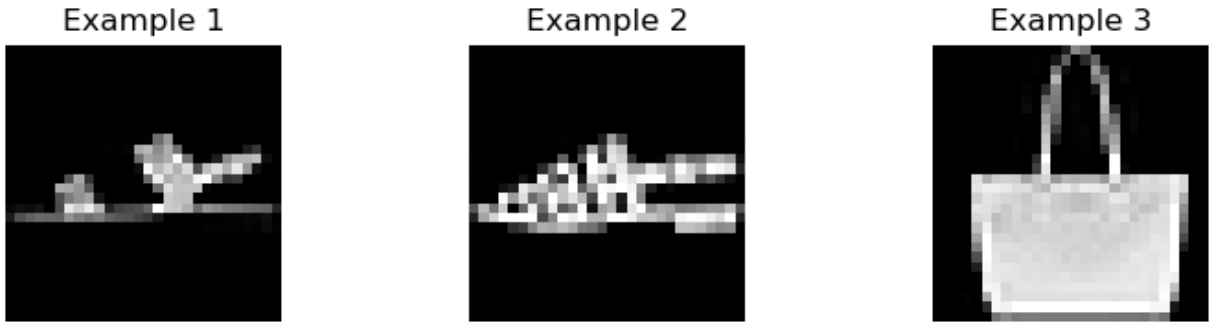
Cluster 2:



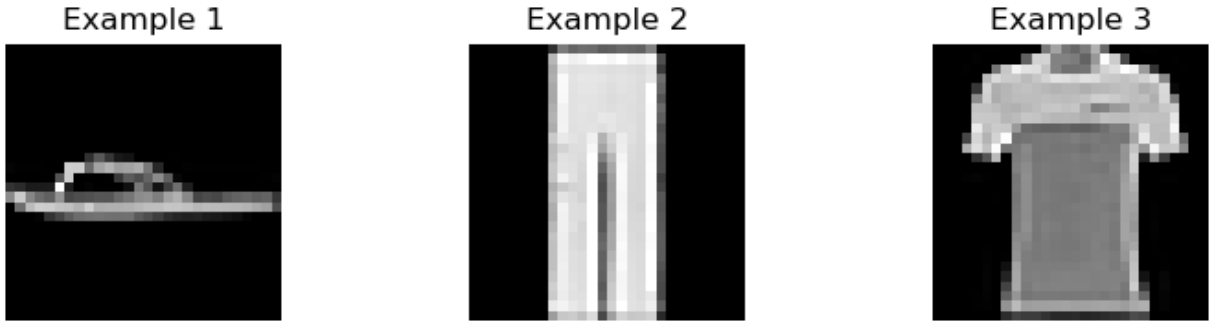
Cluster 3:



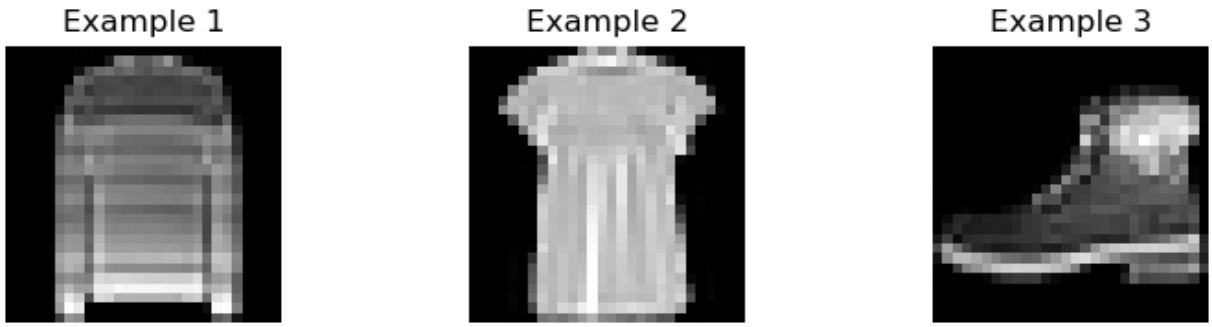
Cluster 4:



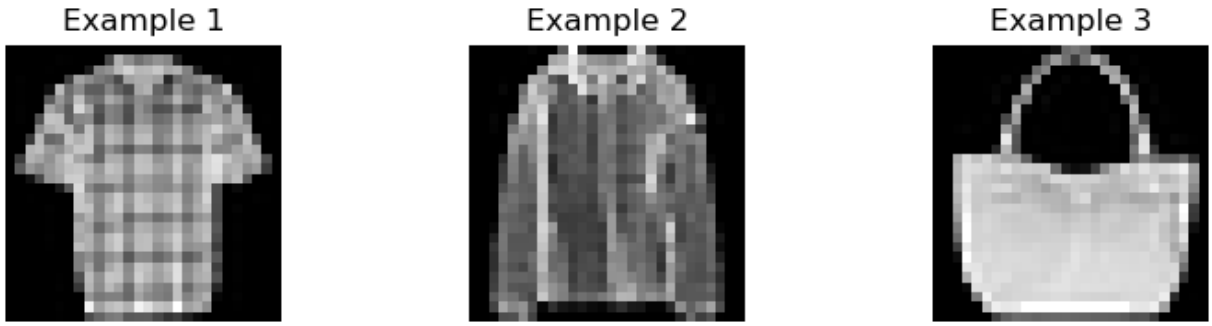
Cluster 5:



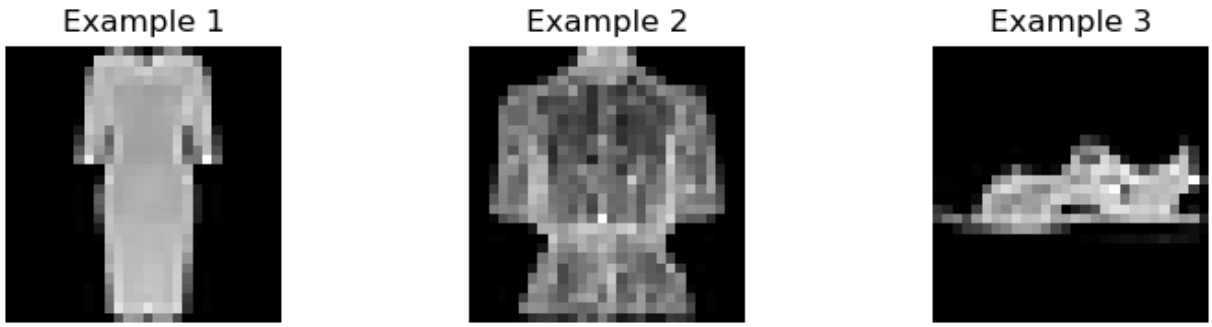
Cluster 6:



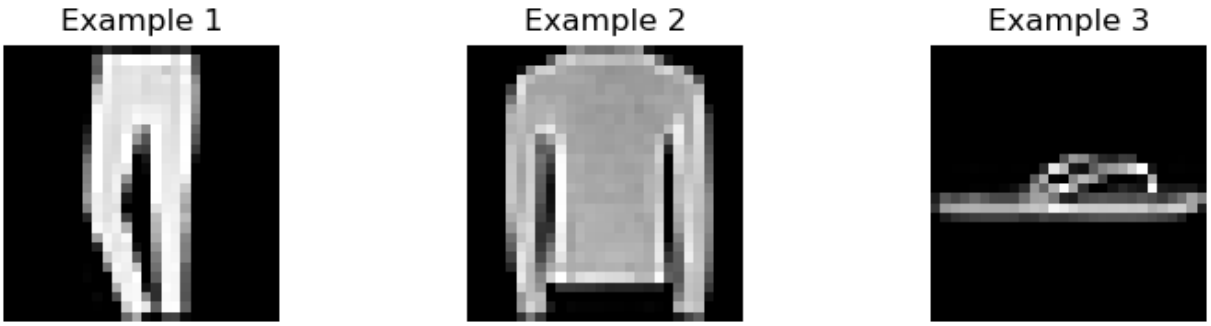
Cluster 7:



Cluster 8:



Cluster 9:



4. Comment on the quality of the clustering.

By printing 3 examples from each cluster, we can see that the Kmeans clustering was not sufficient in categorizing the images into their appropriate class. Within the same cluster, we see pants, shirts, and shoes, none of which belong in the same class. Based on the scatterplot from before, we could presume that this might be the case based on the amount of overlap between classes. Points of different colors had nearly zero separation, so these results are not surprising.

CNN:

Training:

Train a convolutional neural network on the training data with the following architecture:

- Layer 1: 2D convolutional layer, 28 filters, 3x3 window size, ReLU activation
- Layer 2: 2x2 max pooling
- Layer 3: 2D convolutional layer, 56 filters, 3x3 window size, ReLU activation
- Layer 4: fully-connected layer, 56 nodes, ReLU activation
- Layer 5: fully-connected layer, 10 nodes, softmax activation

Use the Adam optimizer, 32 observations per batch, and categorical cross-entropy loss. Use the train and test splits provided by Fashion-MNIST. Use the last 12000 samples of the training data as a validation set. Train for 10 epochs. You are free to reshape the data between layers if needed.

In [82]:
```
# include channel dimension (for convolutional layers)
train_x = np.expand_dims(x_train, axis=-1)
test_x = np.expand_dims(x_test, axis=-1)

# one-hot encode the labels
train_y = to_categorical(y_train)
test_y = to_categorical(y_test)

# validation set of the last 12000 samples
val_x = train_x[-12000:]
val_y = train_y[-12000:]
train_x = train_x[:-12000]
train_y = train_y[:-12000]
```

```
# the CNN architecture
model = models.Sequential([
    layers.Conv2D(28, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(56, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(56, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# compile
model.compile(optimizer='adam',loss='categorical_crossentropy', metrics=['accu

# Train the model
training = model.fit(train_x, train_y, epochs=10, batch_size=32, validation_da
```

```
Epoch 1/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.4424 - ac
curacy: 0.8396 - val_loss: 0.3259 - val_accuracy: 0.8845
Epoch 2/10
1500/1500 [==============================] - 22s 14ms/step - loss: 0.2880 - ac
curacy: 0.8953 - val_loss: 0.2765 - val_accuracy: 0.8987
Epoch 3/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.2380 - ac
curacy: 0.9136 - val_loss: 0.2515 - val_accuracy: 0.9087
Epoch 4/10
1500/1500 [==============================] - 21s 14ms/step - loss: 0.2039 - ac
curacy: 0.9255 - val_loss: 0.2817 - val_accuracy: 0.9013
Epoch 5/10
1500/1500 [==============================] - 20s 13ms/step - loss: 0.1750 - ac
curacy: 0.9349 - val_loss: 0.2418 - val_accuracy: 0.9144
Epoch 6/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.1489 - ac
curacy: 0.9461 - val_loss: 0.2269 - val_accuracy: 0.9180
Epoch 7/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.1262 - ac
curacy: 0.9540 - val_loss: 0.2468 - val_accuracy: 0.9156
Epoch 8/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.1081 - ac
curacy: 0.9611 - val_loss: 0.2673 - val_accuracy: 0.9121
Epoch 9/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.0874 - ac
curacy: 0.9680 - val_loss: 0.2818 - val_accuracy: 0.9176
Epoch 10/10
1500/1500 [==============================] - 19s 12ms/step - loss: 0.0752 - ac
curacy: 0.9724 - val_loss: 0.3448 - val_accuracy: 0.9111
```

Evaluation

After training your network:

1. Print the number of trainable parameters in the model

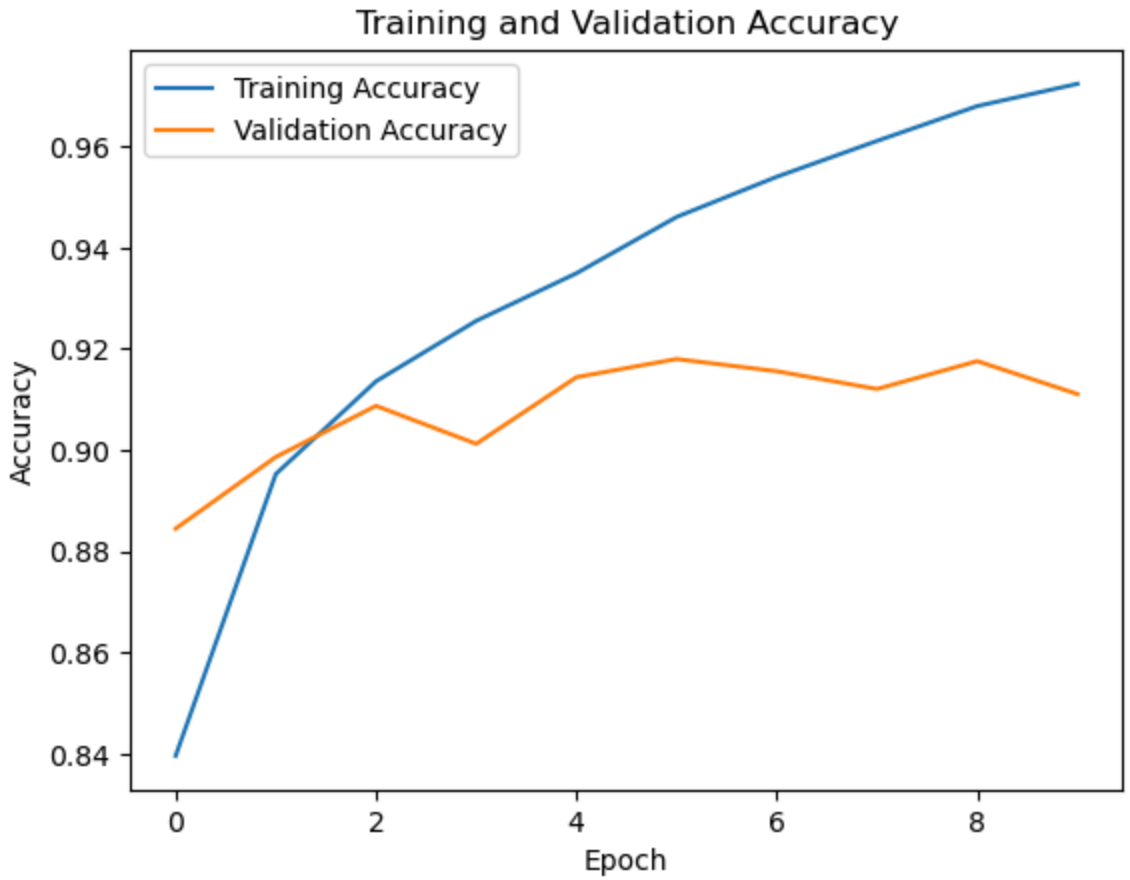
In [115…

```
# extract the number of trainable parameters
trainable_params = model.count_params()
print("Number of trainable parameters:", trainable_params)
```

```
Number of trainable parameters: 394530
```

1. Evaluate training and validation accuracy at the end of each epoch, and plot them as line plots on the same set of axes.

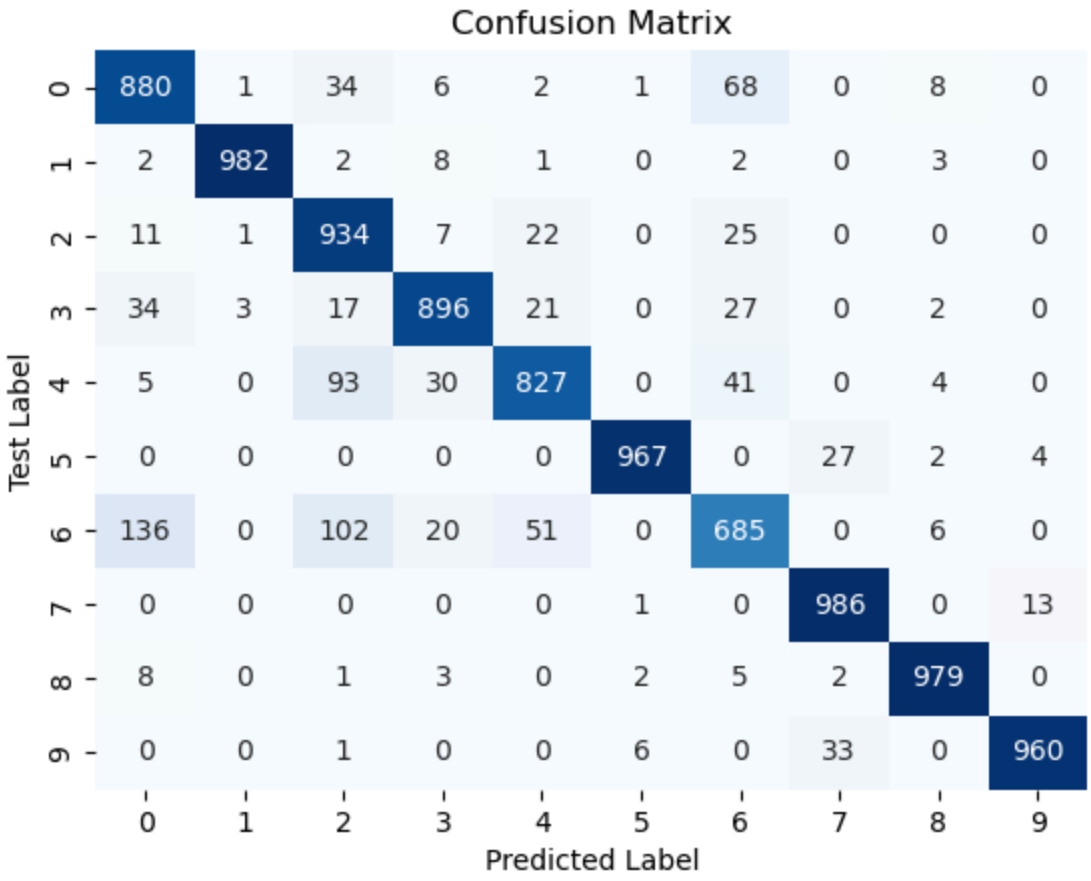
In [116…
```
# plot training and validation accuracy
plt.plot(training.history['accuracy'], label='Training Accuracy')
plt.plot(training.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()
```



3. Show a confusion matrix for the model predictions on the test set.

In [106…
```
# get model predictions
pred = model.predict(test_x)

pred_labels = np.argmax(pred, axis=1)
test_labels = np.argmax(test_y, axis=1)
# create confusion matrix
conf_mat = confusion_matrix(test_labels, pred_labels)
# plot confusion matrix
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('Test Label')
plt.title('Confusion Matrix')
plt.show()
```

```
313/313 [==============================] - 1s 2ms/step
```

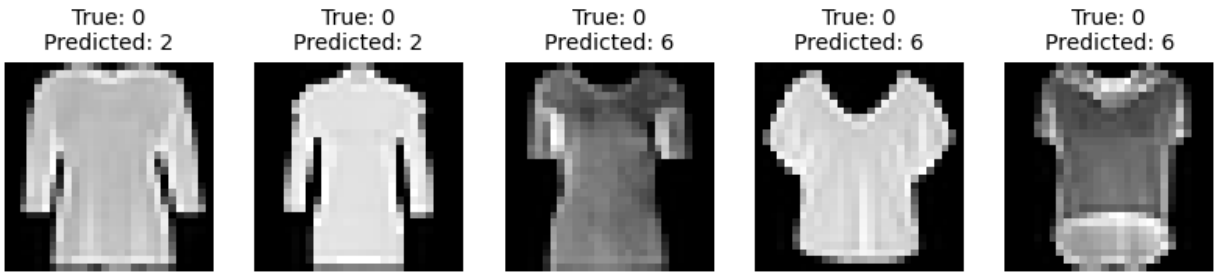


4. For each class, show misclassified examples from the test set.

In [114…

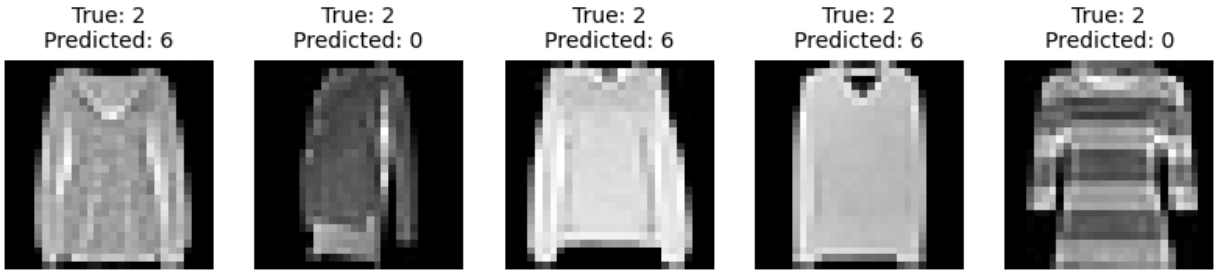
```
# misclassified examples
misclassified_idx = np.where(pred_labels != test_labels)[0]

# dictionary to store misclassified examples for each class
misclassified_ex = {i: [] for i in range(10)}

# iterate over misclassified examples and store them
for idx in misclassified_idx:
    test_label = test_labels[idx]
    pred_label = pred_labels[idx]
    misclassified_ex[test_label].append(idx)

# show misclassified examples for each class
for test_label, misclassified_idxs in misclassified_ex.items():
    plt.figure(figsize=(10, 3))
    plt.suptitle(f'Misclassified Examples for Class {test_label}', fontsize=16
    for i, idx in enumerate(misclassified_idxs[:5]):
        plt.subplot(1, 5, i+1)
        plt.imshow(test_x[idx].reshape(28, 28), cmap='gray')
        plt.title(f'True: {test_label}\nPredicted: {pred_labels[idx]}', fontsi
        plt.axis('off')
    plt.show()
```

Misclassified Examples for Class 0

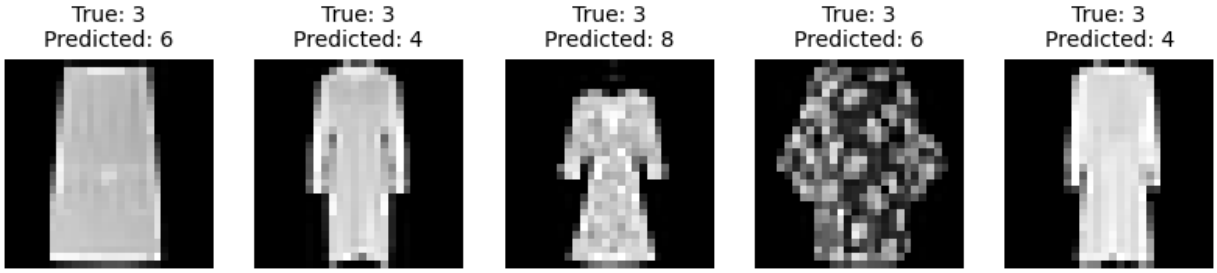


Misclassified Examples for Class 1

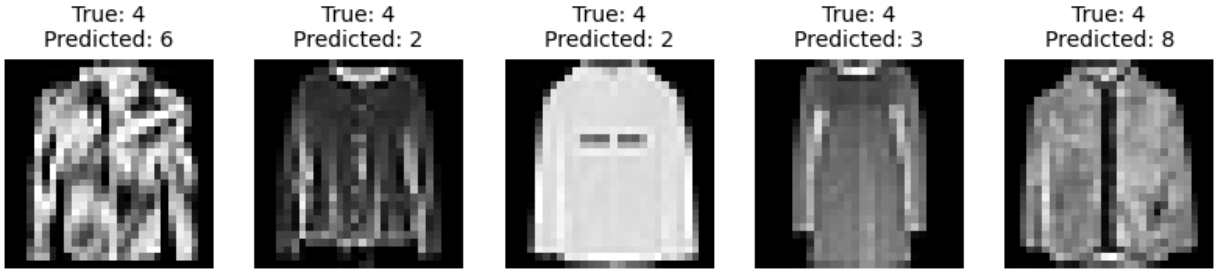
Misclassified Examples for Class 2

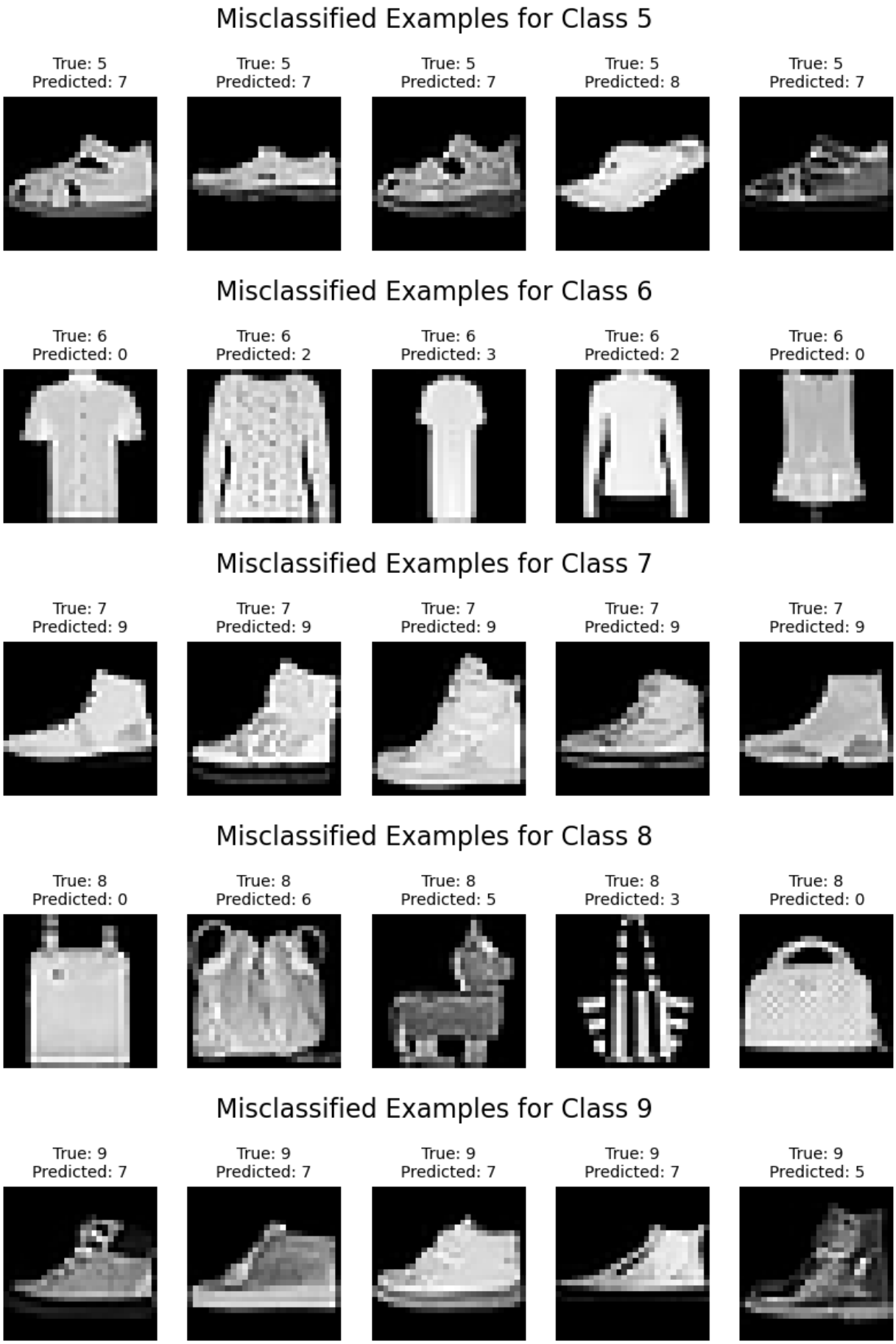


Misclassified Examples for Class 3



Misclassified Examples for Class 4





5. Comment on any other observations about the model performance.

The CNN's model performance is significantly better than the K-means clustering in this scenario. The Kmeans was practically random, and when we looked at samples from the clusters it was evident that the algorithm failed to appropriately classify the images. On the other hand, the CNN worked its way up until a nearly 0.98 accuracy level, meaning it was very accurate in classifying the images. Furthermore, after looking at some of the misclassified examples, we can understand why the model made those mistakes based on how similar some of the groups look. There are no misclassifications that are obviously incorrect, and this further proves the accuracy and appropriateness of this model.