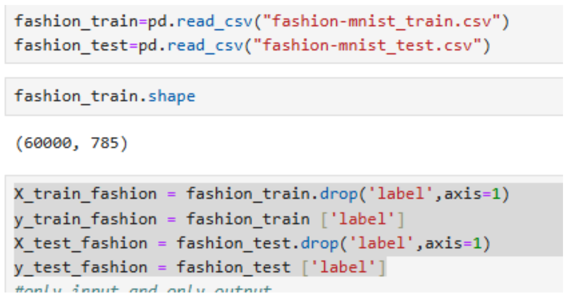


```
fashion_train=pd.read_csv("fashion-mnist_train.csv")
fashion_test=pd.read_csv("fashion-mnist_test.csv")
```

```
fashion_train.shape
```

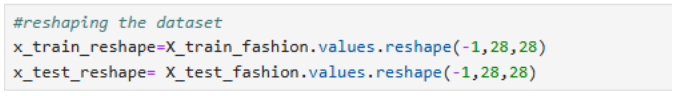
```
(60000, 785)
```

```
X_train_fashion = fashion_train.drop('label',axis=1)
y_train_fashion = fashion_train ['label']
X_test_fashion = fashion_test.drop('label',axis=1)
y_test_fashion = fashion_test ['label']
#only input and only output
```

This code prepares the data for a machine learning model by:

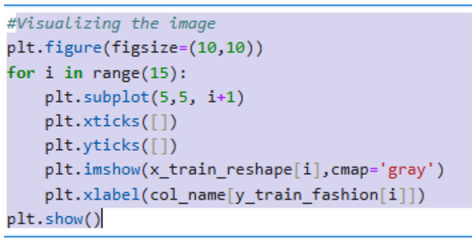
- Separating the training and testing sets.
- Extracting the feature data (images) into X_train_fashion and X_test_fashion.
- Extracting the corresponding labels (classes) into y_train_fashion and y_test_fashion.

This is a common step in machine learning workflows, as it allows the model to learn from the features and predict the labels accurately.



```
#reshaping the dataset
x_train_reshape=X_train_fashion.values.reshape(-1,28,28)
x_test_reshape= X_test_fashion.values.reshape(-1,28,28)
```

This reshaping step is essential for ensuring that your image data is in the correct format for most image-processing and deep learning models



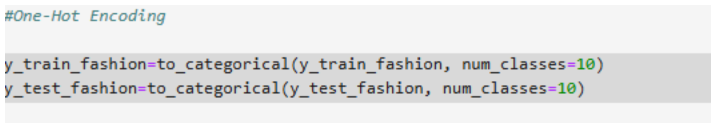
```
#Visualizing the image
plt.figure(figsize=(10,10))
for i in range(15):
    plt.subplot(5,5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_train_reshape[i],cmap='gray')
    plt.xlabel(col_name[y_train_fashion[i]])
plt.show()
```

Explanation:

1. **plt.figure(figsize=(10,10)):** This line creates a new figure window with a size of 10 inches by 10 inches.
2. **for i in range(15)::** This loop iterates 15 times (0 to 14).
3. **plt.subplot(5,5, i+1):** This creates a grid of subplots with 5 rows and 5 columns. The i+1 ensures each image is plotted in a different subplot.
4. **plt.xticks([]) and plt.yticks([]):** These lines hide the x-axis and y-axis ticks for cleaner visualization.
5. **plt.imshow(x_train_reshape[i], cmap='gray'):** This line displays an image from the x_train_reshape data at index i. The cmap='gray' argument specifies that the image should be displayed in grayscale.
6. **plt.xlabel(col_name[y_train_fashion[i]]):** This line retrieves the class label (e.g., "T-shirt", "Trouser") from y_train_fashion at index i and displays it as a label below the image (assuming col_name contains the class labels).
7. **plt.show():** This line displays the entire figure with all the subplots and image labels.

In essence:

This code visualizes 15 images from your x_train_reshape data (assuming it's preprocessed correctly) in a 5x5 grid. Each image is displaye



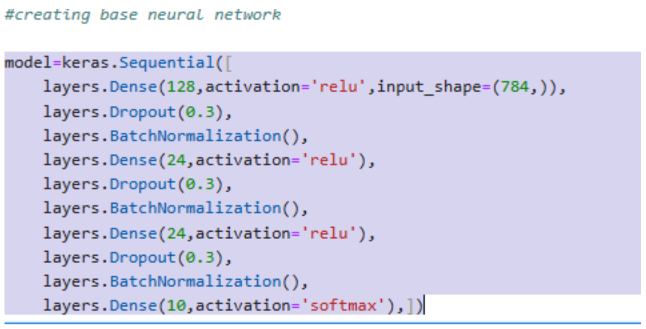
```
#One-Hot Encoding

y_train_fashion=to_categorical(y_train_fashion, num_classes=10)
y_test_fashion=to_categorical(y_test_fashion, num_classes=10)
```

Explanation:

- **One-hot encoding:** This technique transforms the integer labels into a binary vector. For example, if there are 10 classes (as indicated by num_classes=10), and a particular sample belongs to class 3, its label will be represented as:

  [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

```
#creating base neural network

model=keras.Sequential([
    layers.Dense(128,activation='relu',input_shape=(784,)),
    layers.Dropout(0.3),
    layers.BatchNormalization(),
    layers.Dense(24,activation='relu'),
    layers.Dropout(0.3),
    layers.BatchNormalization(),
    layers.Dense(24,activation='relu'),
    layers.Dropout(0.3),
    layers.BatchNormalization(),
    layers.Dense(10,activation='softmax'),])
```

Certainly, let's break down the provided Keras code for creating a sequential neural network model:

1. keras.Sequential([...])

- This line defines a sequential model in Keras. In a sequential model, the layers are stacked one after another in a linear fashion.

2. layers.Dense(128, activation='relu', input_shape=(784,))

- This is the first layer of the network, a **Dense layer**.
  - 128: This specifies the number of neurons in this layer.
  - activation='relu': This defines the activation function as ReLU (Rectified Linear Unit), a common choice for its non-linearity.
  - input_shape=(784,): This specifies the shape of the input data. In this case, each input sample has 784 features. This is crucial for the first layer of the model.

3. layers.Dropout(0.3)

- This layer implements dropout regularization.
  - 0.3: This indicates the dropout rate. During training, 30% of the neurons in the previous layer will be randomly deactivated, preventing overfitting.

4. layers.BatchNormalization()

- This layer normalizes the activations of the previous layer.
    - Batch normalization helps stabilize the training process, improves generalization, and can accelerate convergence.

5-7. layers.Dense(24, activation='relu'), layers.Dropout(0.3), layers.BatchNormalization()

- These lines repeat the pattern of a dense layer with ReLU activation, followed by dropout and batch normalization. This pattern of layers is repeated twice, adding depth to the network.

8. layers.Dense(10, activation='softmax')

- This is the output layer of the network.
    - 10: This specifies the number of output neurons, which corresponds to the number of classes in the classification problem (10 classes in the Fashion MNIST dataset).
    - activation='softmax': This uses the softmax activation function, which produces a probability distribution over the 10 classes. The output of the softmax layer represents the model's predicted probabilities for each class.

In Summary:

This code defines a relatively simple feedforward neural network with three hidden layers. It incorporates dropout and batch normalization to improve the model's generalization ability and training stability.

This type of network is commonly used for image classification tasks, such as the Fashion MNIST dataset.

This code prepares your neural network for training. Here's the breakdown:

This method is used to configure the learning process of a Keras model. It defines how the model will be trained, including the loss function, optimizer, and evaluation metrics.

loss="categorical_crossentropy": This function helps the model understand how far off its predictions are from the actual answers. It's like a guide, telling the model to adjust its guesses to get closer to the truth. This specifies the **loss function** used to measure the error between the model's predictions and the true labels **Categorical Crossentropy:** Since we are dealing with multi-class classification (predicting digits 0-9), categorical crossentropy is an appropriate choice. It measures the difference between the predicted probability distribution and the true one-hot encoded labels.

- The model will try to minimize this loss during training.
- **optimizer="adam":** This is like the model's coach. It helps the model learn from its mistakes and improve its predictions over time. **Adam (Adaptive Moment Estimation):** It is a popular optimization algorithm known for its efficiency and often achieves good results in various deep learning tasks. Adam adapts the learning rate for each parameter during training, which can accelerate convergence.
- **metrics=['accuracy']:** This is how we measure how well the model is doing. It calculates the percentage of correct predictions, giving us a score of how accurate the model is. **Accuracy:** It measures the proportion of correctly classified samples. In this case, it calculates the percentage of images for which the model predicts the correct digit.

Python
```
model.compile(loss="categorical_crossentropy",
        optimizer="adam",
```

metrics=['accuracy'])

This code snippet configures the training process for the neural network model defined earlier. Let's break down each part:

1. model.compile(...):

- This method is used to configure the learning process of a Keras model. It defines how the model will be trained, including the loss function, optimizer, and evaluation metrics.

2. loss="categorical_crossentropy":

- This specifies the **loss function** used to measure the error between the model's predictions and the true labels.
- **Categorical Crossentropy:** Since we are dealing with multi-class classification (predicting digits 0-9), categorical crossentropy is an appropriate choice. It measures the difference between the predicted probability distribution and the true one-hot encoded labels.
- The model will try to minimize this loss during training.

3. optimizer="adam":

- This specifies the **optimizer** used to update the model's weights during training.
- **Adam (Adaptive Moment Estimation):** It is a popular optimization algorithm known for its efficiency and often achieves good results in various deep learning tasks. Adam adapts the learning rate for each parameter during training, which can accelerate convergence.

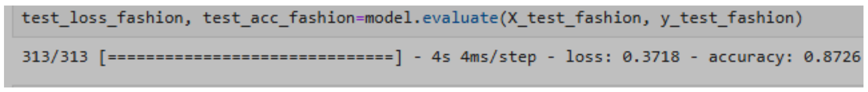
4. metrics=['accuracy']:

- This specifies the **evaluation metric** used to monitor the model's performance during training and testing.
- **Accuracy:** It measures the proportion of correctly classified samples. In this case, it calculates the percentage of images for which the model predicts the correct digit.

In summary:

This code configures the model to use categorical crossentropy as the loss function, the Adam optimizer to update the model's weights, and accuracy as the metric to evaluate its performance. This setup is well-suited for multi-class classification tasks like digit recognition in the MNIST dataset.

This code trains the neural network model on the MNIST training data for 10 epochs(ffn bp), using a batch size of 100 and monitoring its performance on the validation set. The history variable will contain valuable information about the training process, allowing you to analyze the model's learning progress and potentially adjust hyperparameters for further improvement.



```
test_loss_fashion, test_acc_fashion=model.evaluate(X_test_fashion, y_test_fashion)

313/313 [==============================] - 4s 4ms/step - loss: 0.3718 - accuracy: 0.8726
```

-
    - This is the core function call.
    - model: This refers to the trained machine learning model (the neural network you defined previously).

    - X_test_fashion: This represents the input features of the test set.
    - y_test_fashion: This represents the true labels (ground truth) of the test set.
- **test_loss_fashion, test_acc_fashion:**
    - This part unpacks the results returned by model.evaluate().
    - test_loss_fashion: This variable will store the calculated loss (e.g., mean squared error, cross-entropy) on the test set.
    - test_acc_fashion: This variable will store the calculated accuracy on the test set.

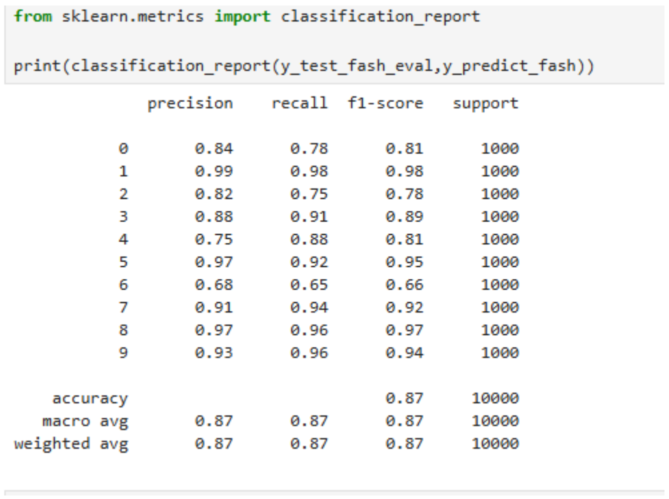
In simpler terms:

This line essentially asks the trained model to make predictions on the X_test_fashion data. It then compares these predictions with the actual labels (y_test_fashion) and calculates:

1. **test_loss_fashion:** How far off the model's predictions are from the true labels.
2. **test_acc_fashion:** The percentage of correct predictions the model made on the test set.

These metrics provide crucial insights into the model's performance on unseen data, helping you understand how well it generalizes beyond the training set.

- **Evaluation is crucial:** Evaluating a model on an independent test set is essential to get an unbiased estimate of its performance and to avoid overfitting (where the model performs well on the training data but poorly on new data).
- **Metrics:** The choice of metrics (loss and accuracy) depends on the specific machine learning task (e.g., regression, classification).



```
from sklearn.metrics import classification_report

print(classification_report(y_test_fash_eval,y_predict_fash))
```

```
              precision    recall  f1-score   support

           0       0.84      0.78      0.81      1000
           1       0.99      0.98      0.98      1000
           2       0.82      0.75      0.78      1000
           3       0.88      0.91      0.89      1000
           4       0.75      0.88      0.81      1000
           5       0.97      0.92      0.95      1000
           6       0.68      0.65      0.66      1000
           7       0.91      0.94      0.92      1000
           8       0.97      0.96      0.97      1000
           9       0.93      0.96      0.94      1000

    accuracy                           0.87     10000
   macro avg       0.87      0.87      0.87     10000
weighted avg       0.87      0.87      0.87     10000
```

Precision:

- **Definition:** Precision measures the proportion of true positive predictions among all positive predictions made by the model.
    - In simpler terms: Out of all the instances the model predicted to belong to a specific class, how many were actually correct?
- **Interpretation:**
    - A high precision indicates that the model is good at identifying true positives and minimizing false positives.
    - For example, if the model predicts that an image is a "dog" with high precision, it means that most of the images it labeled as "dog" are actually dogs.

2. Recall:

- **Definition:** Recall measures the proportion of true positive predictions among all actual positive instances in the dataset.
    - In simpler terms: Out of all the actual instances of a specific class, how many did the model correctly identify?
- **Interpretation:**
    - High recall indicates that the model is good at identifying most of the actual positive instances.
    - For example, if the model has high recall for the "dog" class, it means it can correctly identify most of the dog images in the dataset.

3. F1-score:

- **Definition:** The F1-score is the harmonic mean of precision and recall. It provides a single metric that balances both precision and recall.
- It's particularly useful when you need to consider both false positives and false negatives.
- **Interpretation:**
    - A high F1-score indicates that the model has a good balance between precision and recall.

4. Support:

- **Definition:** Support represents the number of actual occurrences of each class in the dataset.

5. Accuracy:

- **Definition:** Overall accuracy measures the proportion of correctly classified instances across all classes.

6. Macro Average:

- **Definition:** The macro average calculates the average of each metric (precision, recall, F1-score) across all classes and then takes the average of those averages.

7. Weighted Average:

- **Definition:** The weighted average calculates the average of each metric across all classes, but weights each class's contribution based on its support (number of instances).

Interpretation of the Provided Table:

- **Overall Good Performance:** The model generally exhibits good performance across all classes, with accuracy, macro average, and weighted average all around 0.87.
- **Class-Specific Performance:**
    - Classes 1, 5, 8, and 9 show excellent performance with high precision, recall, and F1-scores.
    - Classes 0, 2, 4, and 6 have slightly lower performance, with lower precision and recall values.
    - Class 7 has a lower precision but high recall, suggesting that the model might be prone to false positives for this class.

In Summary:

The classification report provides a comprehensive evaluation of the model's performance, allowing you to:

- Assess overall accuracy.
- Identify classes with strong and weak performance.
- Understand the trade-off between precision and recall for each class.
- Make informed decisions about model improvement, such as addressing class imbalances or focusing on improving performance for specific classes.