In [ ]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
PATH = '/kaggle/input/fashionmnist/fashion-mnist_'
```

In [ ]:

```
import pandas as pd
from collections import Counter
import cv2
import pprint
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.python import keras
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Flatten, Conv2D, Dropout, MaxPooling2D
from keras.utils.vis_utils import plot_model
from sklearn.ensemble import GradientBoostingClassifier
import pickle
```

In [ ]:

```
train = pd.read_csv(PATH+'train.csv')
X_train = train.drop('label', axis=1)
y_train = train['label']
print(f'Features shape:  {X_train.shape}')
print(f'Labels shape:  {y_train.shape}')
```

In [ ]:

```
test = pd.read_csv(PATH+'test.csv')
X_test = test.drop('label', axis=1)
y_test = test['label']
print(f'Features shape:  {X_test.shape}')
print(f'Labels shape:  {y_test.shape}')
```

In [ ]:

```
Counter(y_train)
```

In [ ]:

```
print(f'NAs in Features: {X_train.isna().sum().sum()}')
print(f'NAs in Labels: {y_train.isna().sum().sum()}')
```

In [ ]:

```
plt.imshow(X_train.values.reshape(-1, 28, 28)[0])
```

In [ ]:

```
# Nifty function to convert class number to Human readable class name
def get_name_from_class(class_number):
    classes = {
        '0': 'T-shirt/top',
        '1': 'Trouser',
        '2': 'Pullover',
        '3': 'Dress',
        '4': 'Coat',
        '5': 'Sandal',
        '6': 'Shirt',
        '7': 'Sneaker',
        '8': 'Bag',
        '9': 'Ankle boot'
    }
    return classes[str(class_number)]
```

In [ ]:

```
plt.figure(figsize = (20,10))
for image in range(20):
    plt.subplot(4, 5, image+1)
    plt.imshow(X_train.values.reshape(-1, 28, 28)[image])
    plt.title(get_name_from_class(y_train[image]))
    plt.axis('off')
```

In [ ]:

```
# Normalizing to values between 0 and 1
X_train = X_train / 255.0
X_test = X_test / 255.0
X_train.head(n=25)
```

In [ ]:

```
# First I will implement a Gradient Boosting Algorithm
model0 = GradientBoostingClassifier(verbose=1)
model0.fit(X_train, y_train)
```

In [ ]:

```
filename = 'model0.sav'
pickle.dump(model0, open(filename, 'wb'))
```

In [ ]:

```
tmp = model0.predict(X_test)
```

In [ ]:

```
# Reshape X
X_train = X_train.values.reshape(-1, 28, 28, 1)
X_test = X_test.values.reshape(-1, 28, 28, 1)
print(f'X_train Shape: {X_train.shape}')
print(f'X_test Shape: {X_test.shape}')
```

In [ ]:

```
# One Hot Encoding for Labels
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
print(f'y_train Shape: {y_train.shape}')
print(f'y_test Shape: {y_test.shape}')
```

In [ ]:

```
# Split Train dataset into training and validation
```

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, rand
om_state=123)
print(f'X_train Shape: {X_train.shape}')
print(f'X_val Shape: {X_val.shape}')
print(f'y_train Shape: {y_train.shape}')
print(f'y_val Shape: {y_val.shape}')
```

In [ ]:

```
# Model
model = Sequential()
# Add convolution 2D
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 kernel_initializer='he_normal',
                 input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64,
                 kernel_size=(3, 3),
                 activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))


model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer='adam',
              metrics=['accuracy'])
```

In [ ]:

```
model.summary()
```

In [ ]:

```
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

In [ ]:

```
train_model = model.fit(X_train, y_train,
                  batch_size=100,
                  epochs=60,
                  verbose=1,
                  validation_data=(X_val, y_val))
```

In [ ]:

```
plt.plot(train_model.history['accuracy'])
plt.plot(train_model.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

In [ ]:

```
plt.plot(train_model.history['loss'])
plt.plot(train_model.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

In [ ]:

```
# Model
```

```
model2 = Sequential()
# Add convolution 2D
model2.add(Conv2D(64, kernel_size=(2, 2),
                  activation='relu',
                  kernel_initializer='he_normal',
                  input_shape=(28, 28, 1)))
model2.add(MaxPooling2D((2, 2)))
model2.add(Dropout(0.33))
model2.add(Conv2D(32,
                  kernel_size=(2, 2),
                  activation='relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.33))
model2.add(Conv2D(32,
                  kernel_size=(2, 2),
                  activation='relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.33))
model2.add(Conv2D(128, (2, 2), activation='relu'))
model2.add(Flatten())
model2.add(Dense(128, activation='relu'))
model2.add(Dense(10, activation='softmax'))


model2.compile(loss=keras.losses.categorical_crossentropy,
               optimizer='adam',
               metrics=['accuracy'])
```

In [ ]:

```
model2.summary()
```

In [ ]:

```
plot_model(model2, to_file='model2_plot.png', show_shapes=True, show_layer_names=True)
```

In [ ]:

```
train_model2 = model2.fit(X_train, y_train,
                  batch_size=100,
                  epochs=60,
                  verbose=1,
                  validation_data=(X_val, y_val))
```

In [ ]:

```
plt.plot(train_model2.history['accuracy'])
plt.plot(train_model2.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

In [ ]:

```
plt.plot(train_model2.history['loss'])
plt.plot(train_model2.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

In [ ]:

```

```