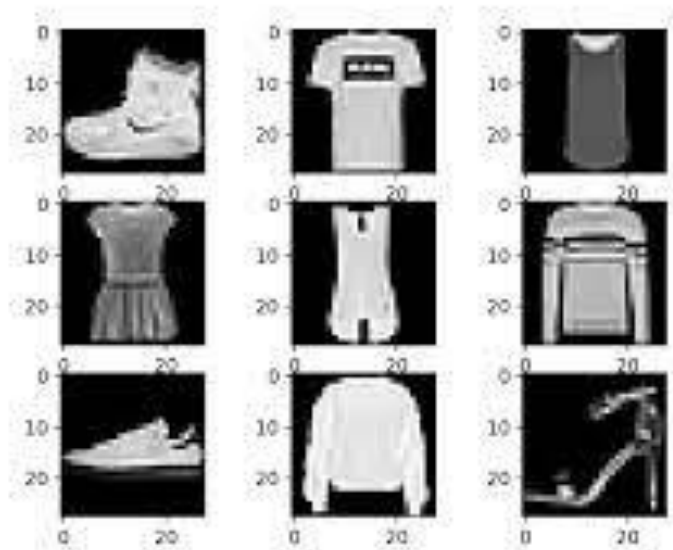


PREDICTING IMAGES OF CLOTHES USING CNN

WITH TENSORFLOW
DR. ALVIN ANG



CONTENTS

Step 1: Load the Data	3
A. Import All Libraries	4
B. Import the Data + Train Test Split	5
C. Creating Class Names	9
D. Reshape the Data, or rather, add in additional Parameter → Channels (RGB)	9
E. Data Normalization	9
F. Split the Data into Train / Validation / Test Datasets	10
G. Presetting the Ransom Seeds	11
II. Step 2: Build the Model Architecture	12
A. Summary of the Model	13
B. Visualize the Model	14
III. Step 3: Compile the Model	15
IV. Step 4: Train the Model	16
V. Step 5: Save the Model	17
VI. Step 6: Load the Model	18
VII. Step 7: Running a Prediction	19
A. Importing Image Reshaping Libraries	19
B. Importing and Previewing the T Shirt Image	20
C. Re-Coloring / Re-Shaping / Re-Sizing	21
VIII. Step 8: Visualize the Loss or Error	22
IX. Step 9: Visualize the Accuracy	24
X. Step 10: Evaluate the Model	25
About Dr. Alvin Ang	26

STEP 1: LOAD THE DATA

https://www.alvinang.sg/s/Predicting_Images_of_Clothes_using_CNN_by_Dr_Alvin_Ang.ipynb

<https://keras.io/datasets/#fashion-mnist-database-of-fashion-articles>

Step 1: Load the Data

Link <https://keras.io/datasets/#fashion-mnist-database-of-fashion-articles>

Dataset of 60,000 28x28 grayscale images of 10 fashion categories, along with a test set of 10,000 images. The class labels are:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

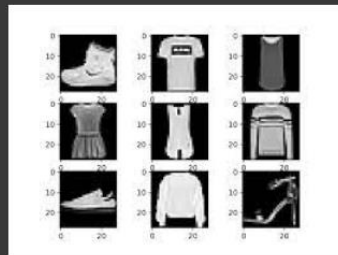
Usage:

```
from keras.datasets import fashion_mnist
```

```
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

Returns: 2 tuples:

1. x_train, x_test: uint8 array of grayscale image data with shape (num_samples, 28, 28).
2. y_train, y_test: uint8 array of labels (integers in range 0-9) with shape (num_samples,).



A. IMPORT ALL LIBRARIES

▼ 1a) Import All Libraries

```
✓  
0s [1] import numpy as np  
import pandas as pd  
%matplotlib inline  
import matplotlib as mpl  
import matplotlib.pyplot as plt
```

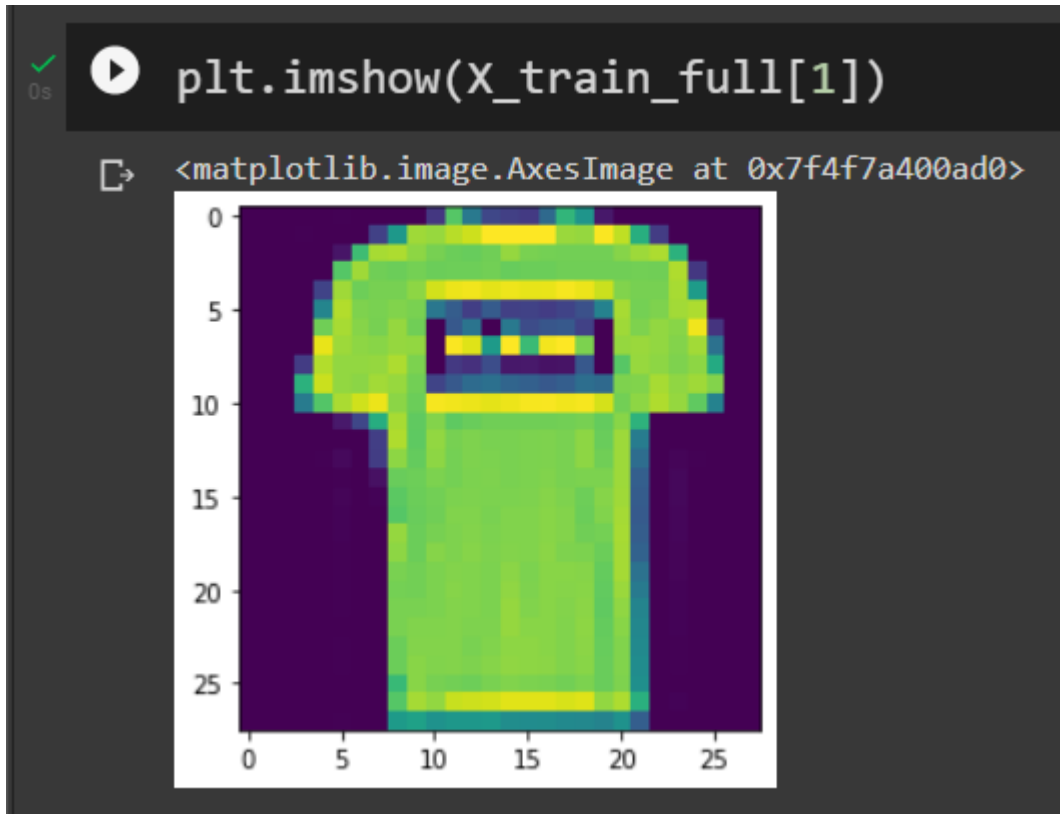
```
✓  
5s [2] import tensorflow as tf  
from tensorflow import keras
```

B. IMPORT THE DATA + TRAIN TEST SPLIT

1b) Import the Data + Train Test Split

```
▶ fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()

↳ Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [=====] - 0s 0us/step
40960/29515 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [=====] - 0s 0us/step
26435584/26421880 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
16384/5148 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [=====] - 0s 0us/step
4431872/4422102 [=====] - 0s 0us/step
```



```
0s ✓ X_train_full[1]
[ 0, 0],
[ 0, 0, 0, 164, 235, 214, 211, 220, 216, 201, 52, 71, 89,
 94, 83, 78, 70, 76, 92, 87, 206, 207, 222, 213, 219, 208,
 0, 0],
[ 0, 0, 0, 106, 187, 223, 237, 248, 211, 198, 252, 250, 248,
 245, 248, 252, 253, 250, 252, 239, 201, 212, 225, 215, 193, 113,
 0, 0],
[ 0, 0, 0, 0, 0, 17, 54, 159, 222, 193, 208, 192, 197,
 200, 200, 200, 200, 201, 203, 195, 210, 165, 0, 0, 0, 0,
 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 47, 225, 192, 214, 203, 206,
 204, 204, 205, 206, 204, 212, 197, 218, 107, 0, 0, 0, 0,
 0, 0],
[ 0, 0, 0, 0, 1, 6, 0, 46, 212, 195, 212, 202, 206,
 205, 204, 205, 206, 204, 212, 200, 218, 91, 0, 3, 1, 0,
 0, 0],
[ 0, 0, 0, 0, 0, 1, 0, 11, 197, 199, 205, 202, 205,
 206, 204, 205, 207, 204, 205, 205, 218, 77, 0, 5, 0, 0,
 0, 0],
[ 0, 0, 0, 0, 0, 3, 0, 2, 191, 198, 201, 205, 206,
 205, 205, 206, 209, 206, 199, 209, 219, 74, 0, 5, 0, 0,
 0, 0],
[ 0, 0, 0, 0, 0, 2, 0, 0, 188, 197, 200, 207, 207,
 204, 207, 207, 210, 208, 198, 207, 221, 72, 0, 4, 0, 0,
 0, 0],
[ 0, 0, 0, 0, 0, 2, 0, 0, 215, 198, 203, 206, 208,
 205, 207, 207, 210, 208, 200, 202, 222, 75, 0, 4, 0, 0,
 0, 0]
```

```
0s ✓ [9] X_train_full[1].shape

#we see that the handwritten digit 0
#in x_train[1] is represented by 28 rows x 28 columns of data
#each represents 0 (black) to 255 (white)

(28, 28)
```

```
▶ y_train_full[1]
```

```
#| Label | Description |  
#--- | --- |  
#| 0 | T-shirt/top  
#| 1 | Trouser  
#| 2 | Pullover  
#| 3 | Dress  
#| 4 | Coat  
#| 5 | Sandal  
#| 6 | Shirt  
#| 7 | Sneaker  
#| 8 | Bag  
#| 9 | Ankle boot
```

```
#we see that the prediction is 0 which is a T-shirt...
```

```
↳ 0
```

```
✓ [12] display(y_train_full)  
0s
```

```
array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

```
✓ [14] print(len(np.unique(y_train_full)))  
0s
```

```
#to see how many classes there are... 10 classes
```

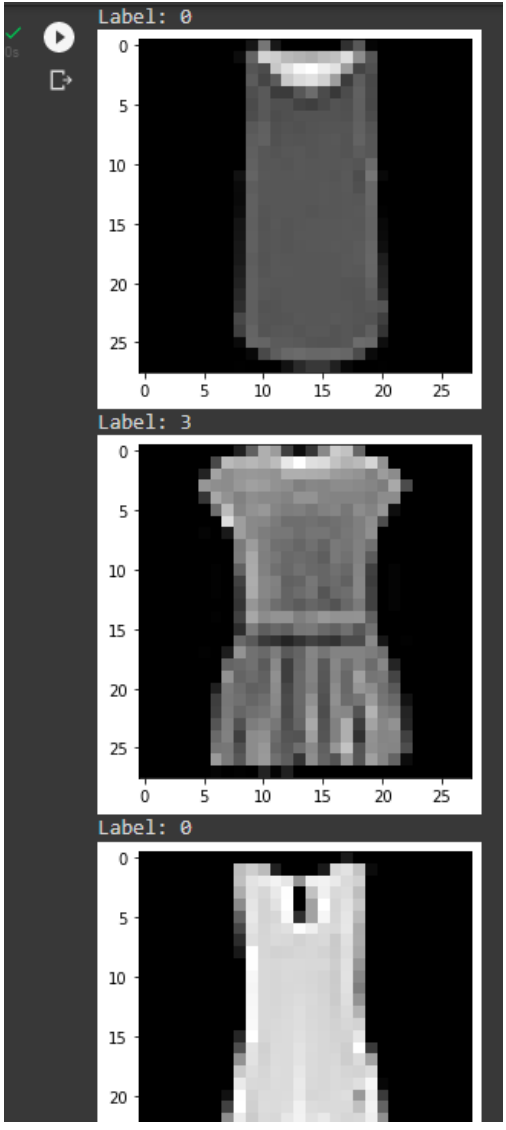
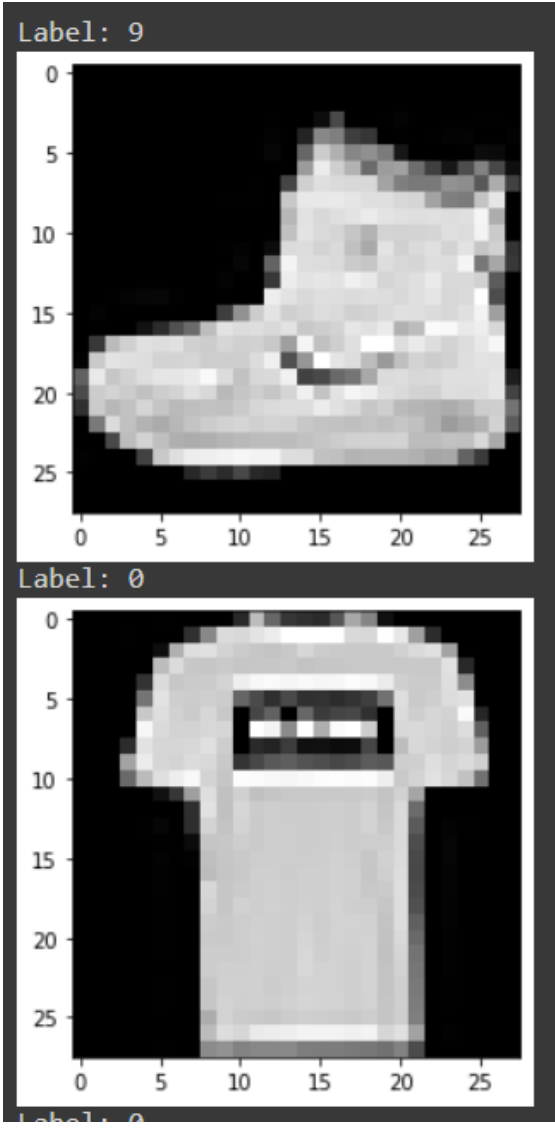
```
10
```

```
✓ ▶ for i in range(5):  
0s
```

```
print(f'Label: {y_train_full[i]}')
```

```
plt.imshow(X_train_full[i], cmap='gray')
```

```
plt.show()
```



C. CREATING CLASS NAMES

1c) Creating Class Names

```
[19] class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",  
                  "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]
```

D. RESHAPE THE DATA, OR RATHER, ADD IN ADDITIONAL PARAMETER → CHANNELS (RGB)

1d) Reshape the Data, or rather, add in additional Parameter → Channels (RGB)

```
[20] X_train_full = X_train_full.reshape((60000, 28, 28, 1))  
      X_test = X_test.reshape((10000, 28, 28, 1))  
  
      #this means that we convert the x_train to include another parameter called Channel
```

E. DATA NORMALIZATION

1e) Data normalization

We normalize the data dimensions so that they are of approximately the same scale.

```
[21] X_train_n = X_train_full / 255.  
      X_test_n = X_test / 255.  
  
      #pictures are from grayscale 0 to 255, so we divide by 255 to  
      #normalize from 0 to 1 probability  
  
      #0 is pure black  
      #255 (or 1) is pure white
```

F. SPLIT THE DATA INTO TRAIN / VALIDATION / TEST DATASETS

1f) Split the data into train/validation/test datasets

In the earlier step of importing the data, we had 60,000 datasets for training and 10,000 test datasets. Now we further split the training data into train/validation. Here is how each type of dataset is used in deep learning:

- **Training data** – used for training the model (x55,000 rows)
- **Validation data** – used for tuning the hyperparameters and evaluate the models (x5000 rows)
- **Test data** – used to test the model after the model has gone through initial vetting by the validation set. (x10,000 rows)

```
[22] X_valid, X_train = X_train_n[:5000], X_train_n[5000:]  
     y_valid, y_train = y_train_full[:5000], y_train_full[5000:]  
     X_test = X_test_n
```

```
[30] X_valid.shape  
     #5,000 rows of 28 x 28 grayscale images (1 channel)  
     (5000, 28, 28, 1)
```

```
[31] X_train.shape  
     #55,000 rows of 28 x 28 grayscale images (1 channel)  
     (55000, 28, 28, 1)
```

```
▶ y_valid.shape  
   #5,000 rows of labelled data corresponding to the images  
   (5000,)
```

```
[28] y_train.shape  
     #55,000 rows of labelled data corresponding to the images  
     (55000,)
```

```
[29] X_test.shape  
     #10,000 rows of 28 x 28 grayscale images (1 channel)  
     (10000, 28, 28, 1)
```

G. PRESETTING THE RANSOM SEEDS

▾ 1g) Presetting the Random Seeds (not entirely needed...)

```
✓ [33] np.random.seed(42)
0s    tf.random.set_seed(42)

#this step may be skipped because its not entirely important...
```

▼ Step 2: Build the Model Architecture

```
✓ 0s ▶ model = keras.models.Sequential()


#The CNN Layers...
model.add(keras.layers.Conv2D(filters = 32,
                              kernel_size = (3, 3),
                              strides=1,
                              padding='valid',
                              activation='relu',
                              input_shape=(28, 28, 1)))


#Pooling...
model.add(keras.layers.MaxPooling2D((2, 2)))

#The ANN Layers....
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

A. SUMMARY OF THE MODEL

2a) Summary of the Model

0s  `model.summary()`

 Model: "sequential_1"

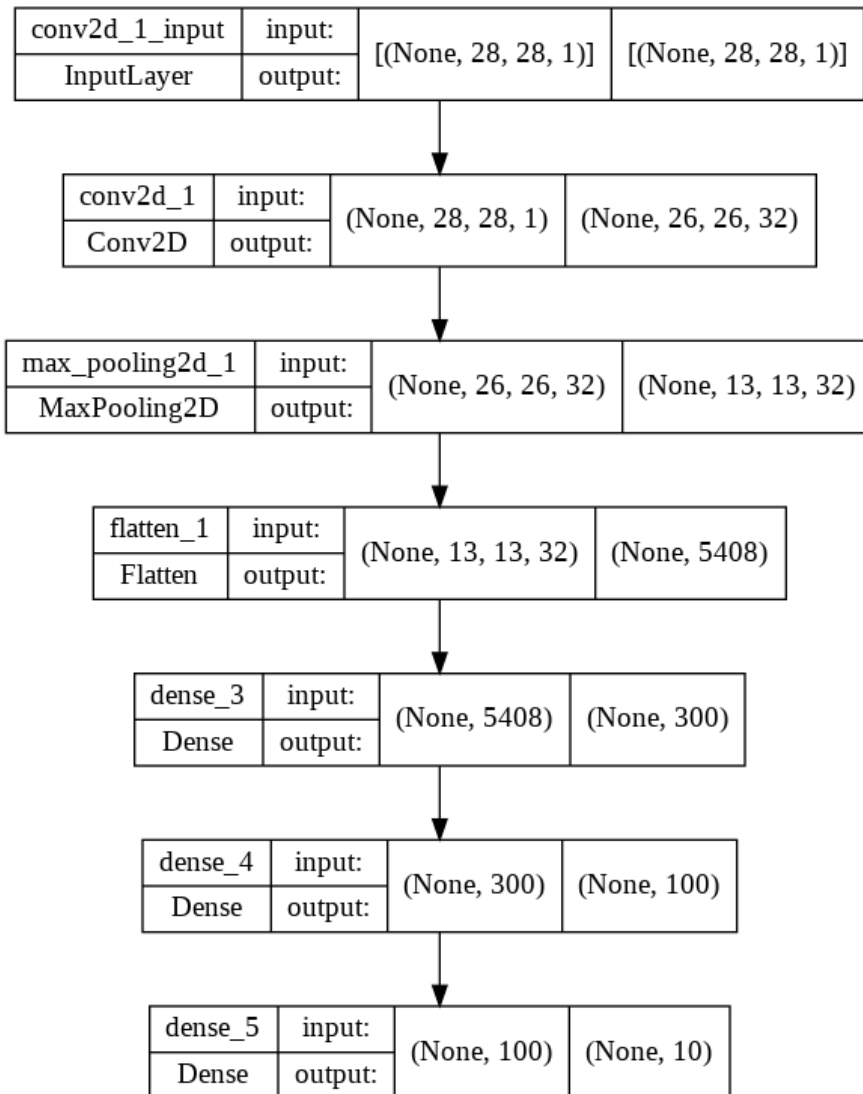
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_1 (MaxPooling 2D)	(None, 13, 13, 32)	0
flatten_1 (Flatten)	(None, 5408)	0
dense_3 (Dense)	(None, 300)	1622700
dense_4 (Dense)	(None, 100)	30100
dense_5 (Dense)	(None, 10)	1010

=====
Total params: 1,654,130
Trainable params: 1,654,130
Non-trainable params: 0
=====

B. VISUALIZE THE MODEL

2b) Visualize the Model

```
import pydot  
keras.utils.plot_model(model, 'model.png', show_shapes=True)
```



▼ Step 3: Compile the Model

```
✓ [38] model.compile(loss="sparse_categorical_crossentropy",  
0s optimizer="sgd",  
metrics=["accuracy"])  
  
#sparse_categorical_crossentropy is slower to train  
#but easier to code  
#try converting to categorical_crossentropy for faster processing  
  
#sparse_categorical_crossentropy dun need "One-Hot Encoding" for  
#both inputs and outputs  
  
#SGD is for cross entropy --> Classification  
#RMSprop is for Regression
```

IV. STEP 4: TRAIN THE MODEL

Step 4: Train the Model

```
[39] model_history = model.fit(  
    X_train, y_train,  
    epochs=30,  
    batch_size= 64,  
    validation_data=(X_valid, y_valid))
```

```
Epoch 1/30  
860/860 [=====] - 38s 44ms/step - loss: 0.8468 - accuracy: 0.7083 - val_loss: 0.8137 - val_accuracy: 0.7014  
Epoch 2/30  
860/860 [=====] - 38s 45ms/step - loss: 0.5360 - accuracy: 0.8066 - val_loss: 0.4983 - val_accuracy: 0.8106  
Epoch 3/30  
860/860 [=====] - 39s 45ms/step - loss: 0.4787 - accuracy: 0.8271 - val_loss: 0.9614 - val_accuracy: 0.6952  
Epoch 4/30  
860/860 [=====] - 39s 45ms/step - loss: 0.4423 - accuracy: 0.8420 - val_loss: 0.4144 - val_accuracy: 0.8568  
Epoch 5/30  
860/860 [=====] - 39s 45ms/step - loss: 0.4185 - accuracy: 0.8491 - val_loss: 0.4300 - val_accuracy: 0.8472  
Epoch 6/30  
860/860 [=====] - 39s 45ms/step - loss: 0.3953 - accuracy: 0.8577 - val_loss: 0.3779 - val_accuracy: 0.8696  
Epoch 7/30  
860/860 [=====] - 40s 46ms/step - loss: 0.3781 - accuracy: 0.8654 - val_loss: 0.4079 - val_accuracy: 0.8528  
Epoch 8/30  
860/860 [=====] - 39s 45ms/step - loss: 0.3633 - accuracy: 0.8696 - val_loss: 0.5108 - val_accuracy: 0.8154  
Epoch 9/30  
860/860 [=====] - 39s 45ms/step - loss: 0.3494 - accuracy: 0.8734 - val_loss: 0.3578 - val_accuracy: 0.8714  
Epoch 10/30  
860/860 [=====] - 39s 45ms/step - loss: 0.3374 - accuracy: 0.8781 - val_loss: 0.3892 - val_accuracy: 0.8550  
Epoch 11/30  
860/860 [=====] - 39s 45ms/step - loss: 0.3246 - accuracy: 0.8809 - val_loss: 0.3444 - val_accuracy: 0.8760  
Epoch 12/30
```


▼ Step 5: Save the Model



```
model.save('classification.h5')
```

```
#h5 means HDF5 format
```

```
#it saves the model configuration, weights
```

```
#bias and all hyperparameters
```

VI. STEP 6: LOAD THE MODEL

<https://www.alvinang.sg/s/classification.h5>

▾ Step 6: Load the Model

<https://www.alvinang.sg/s/classification.h5>

```
[ ] # model = keras.models.load_model('classification.h5')  
  
#though not needed here, we will load this model for future use  
#so that we don't have to run the 30 epochs as it takes very long
```

VII. STEP 7: RUNNING A PREDICTION

▼ Step 7: Running a Prediction



A. IMPORTING IMAGE RESHAPING LIBRARIES

▼ 7a) Importing Image Reshaping Libraries

```
✓ [41] import numpy as np  
0s import cv2  
from skimage import img_as_ubyte  
from skimage.color import rgb2gray  
from keras.models import load_model
```

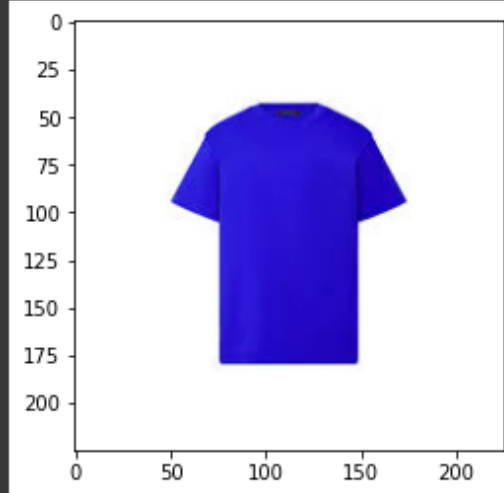
B. IMPORTING AND PREVIEWING THE T SHIRT IMAGE

7b) Importing and Previewing the T Shirt Image

```
✓ [42] shirt = cv2.imread('/content/t-shirt.jpg')
```

```
✓ plt.imshow(shirt)
```

```
↳ <matplotlib.image.AxesImage at 0x7f4f79e39750>
```



C. RE-COLORING / RE-SHAPING / RE-SIZING

7c) Re-Coloring / Re-Shaping / Re-Sizing

```
[44] shirt_gray = rgb2gray(shirt)
     shirt_gray_u8 = img_as_ubyte(shirt_gray)
```

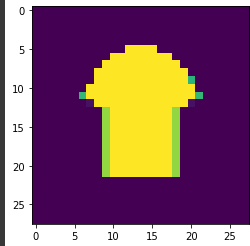
```
[45] [thresh, im_binary] = cv2.threshold(shirt_gray_u8, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
```

```
[46] img_resized = cv2.resize(im_binary,(28,28))
```

```
[47] im_gray_invert = 255 - img_resized
```

```
[48] plt.imshow(im_gray_invert)
```

<matplotlib.image.AxesImage at 0x7f4f79c37bd0>



```
[49] im_final = im_gray_invert.reshape(1,28,28,1)
```

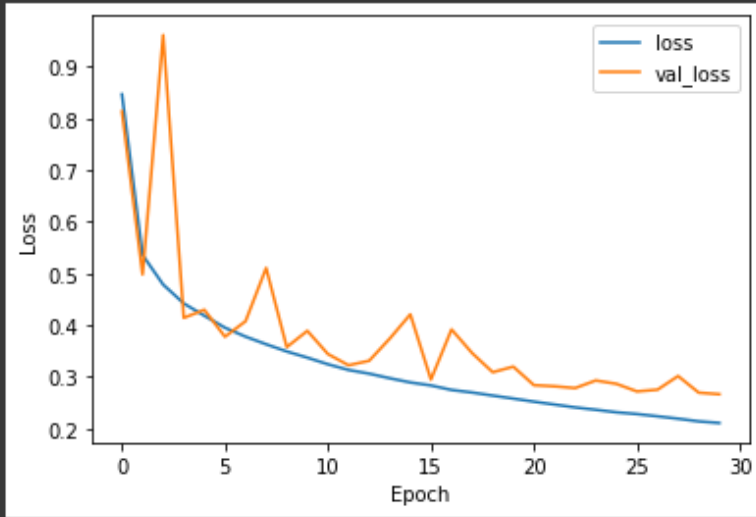
▼ Step 8: Visualize the Loss or Error

```
✓ [54] loss = model_history.history['loss']  
    val_loss = model_history.history['val_loss']  
    acc = model_history.history['accuracy']  
    val_acc = model_history.history['val_accuracy']  
    epoch = range(len(loss))
```



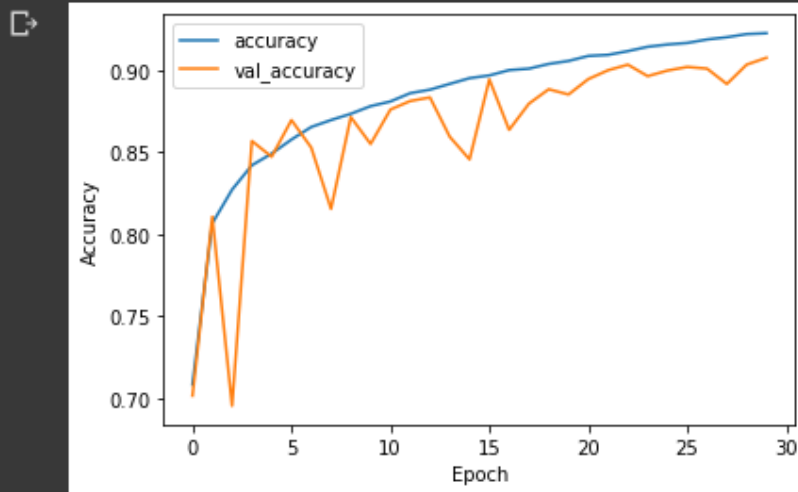
```
import matplotlib.pyplot as plt

plt.plot(epoch, loss, label = "loss")
plt.plot(epoch, val_loss, label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



▼ Step 9: Visualize the Accuracy

```
plt.plot(epoch, acc, label = 'accuracy')  
plt.plot(epoch, val_acc, label = 'val_accuracy')  
plt.xlabel("Epoch")  
plt.ylabel('Accuracy')  
plt.legend()  
plt.show()
```



X. STEP 10: EVALUATE THE MODEL

Step 10: Evaluate the Model

```
✓ [57] ev = model.evaluate(X_test_n, y_test)
```

```
3s 313/313 [=====] - 3s 11ms/step - loss: 0.2737 - accuracy: 0.9014
```

```
[58] ev
```

```
#Accuracy achieved of 90%
```

```
[0.2736997604370117, 0.9014000296592712]
```

ABOUT DR. ALVIN ANG



Dr. Alvin Ang earned his Ph.D., Masters and Bachelor degrees from NTU, Singapore. He is a scientist, entrepreneur, as well as a personal/business advisor. More about him at www.AlvinAng.sg.