```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt


(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

print(x_train.shape)
print(x_test.shape)
print(y_train)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
(50000, 32, 32, 3)
(10000, 32, 32, 3)
[[6]
 [9]
 [9]
 ...
 [9]
 [1]
 [1]]
```

```
y_train_new=np.squeeze(y_train,axis=1)
y_test_new=np.squeeze(y_test,axis=1)
print(y_train_new.shape)
print(y_train_new)
```

```
(50000,)
[6 9 9 ... 9 1 1]
```

```
class_labels=['airplanes', 'cars', 'birds', 'cats', 'deer', 'dogs', 'frogs', 'horses',
              'ships', 'trucks']
plt.figure(figsize=(15,15))
for i in range(100):
  plt.subplot(10,10,i+1)
  plt.imshow(x_train[i])
  plt.axis('off')
  plt.title(class_labels[y_train_new[i]])


# Preprocessing
x_train=x_train/255
x_test=x_test/255


model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu',
                                 input_shape=(32,32,3)))
model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding='valid', activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 conv2d_1 (Conv2D)           (None, 30, 30, 64)        18496

 max_pooling2d (MaxPooling2  (None, 15, 15, 64)        0
 D)

 dropout (Dropout)           (None, 15, 15, 64)        0

 batch_normalization (Batch  (None, 15, 15, 64)        256
 Normalization)

 flatten (Flatten)           (None, 14400)             0

 dense (Dense)               (None, 512)               7373312

 dense_1 (Dense)             (None, 10)                5130

=================================================================
```

```
Total params: 7398090 (28.22 MB)
Trainable params: 7397962 (28.22 MB)
Non-trainable params: 128 (512.00 Byte)
_________________________________________________________________
```

```
model.compile(optimizer="rmsprop",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
```

```
hist = model.fit(x_train,y_train_new,batch_size=32,epochs=5, verbose=1,validation_split=.3)
```

```
# Using the save() method, the model will be saved to the file system in the 'SavedModel' format.
model.save("model_saved")
```

```
reloaded_model = tf.keras.models.load_model('model_saved')
```

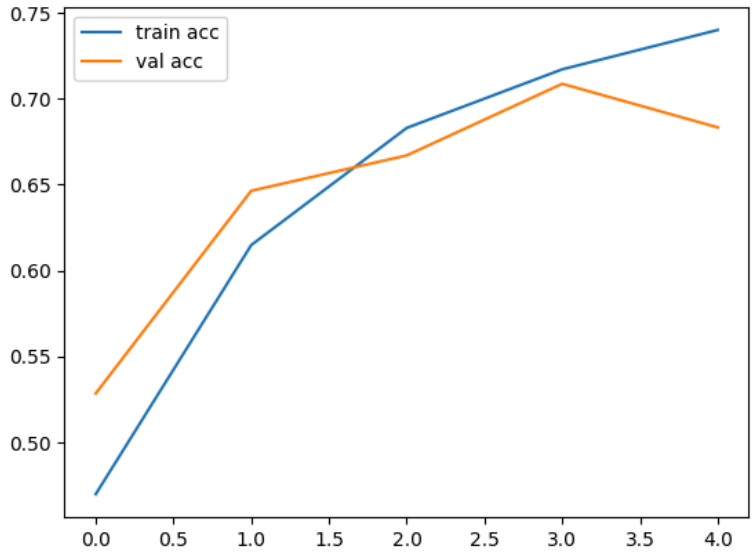
```
dict=hist.history
dict.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
plt.plot(dict['loss'],label='train loss')
plt.plot(dict['val_loss'],label='val loss')
plt.legend()
```

```
plt.plot(dict['accuracy'],label='train acc')
plt.plot(dict['val_accuracy'],label='val acc')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7eb393f88fd0>
```




```
y_pred=model.predict(x_test)
```

```
313/313 [==============================] - 13s 41ms/step
```

```
print(y_test_new[0])
print(y_pred[0])
print(np.argmax(y_pred[0]))
```

```
3
[4.5671612e-05 4.0409667e-03 1.6291294e-02 1.2875368e-01 5.5368023e-06
 2.5623497e-02 1.3495984e-03 2.0637200e-04 8.0470902e-01 1.8974297e-02]
8
```

```
y_label=np.argmax(y_pred,axis=1)
print(y_label)
```

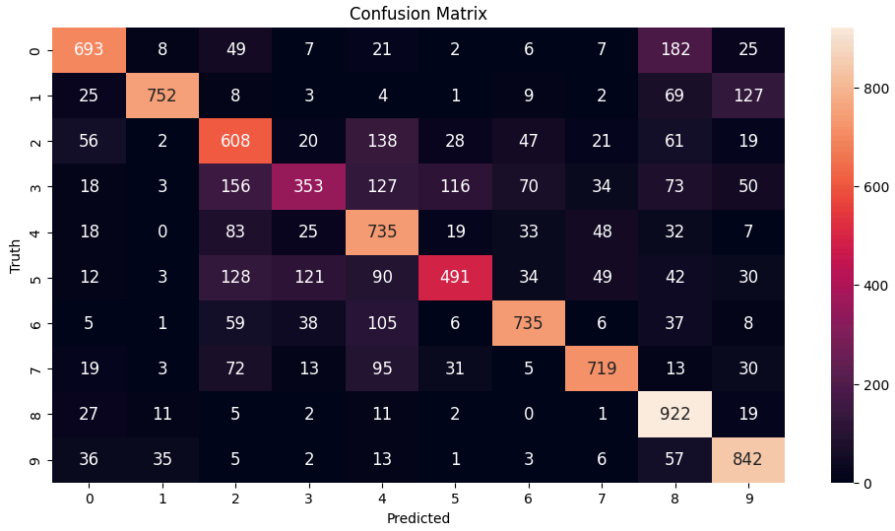
```
[8 8 8 ... 5 1 7]
```

```
plt.figure(figsize=(15,15))
for i in range(100):
  plt.subplot(10,10,i+1)
  plt.imshow(x_test[i])
  plt.axis('off')
```

```
test_loss,test_acc=model.evaluate(x_test,y_test_new)
print(test_acc)
```

```
313/313 [==============================] - 13s 42ms/step - loss: 1.0884 - accuracy: 0.6850
0.6850000023841858
```

```
# Generate a confusion matrix for the test dataset.
cm = tf.math.confusion_matrix(labels=y_test_new, predictions=y_label)
# Plot the confusion matrix as a heatmap.
plt.figure(figsize=[12, 6])
import seaborn as sn
sn.heatmap(cm, annot=True, fmt="d", annot_kws={"size": 12})
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```




```
n=np.sum(y_label==y_test_new)
n/x_test.shape[0]
```

```
0.685
```