Revathi Atchi

700742168

NEURAL NETWORK AND DEEP LEARNING ASSIGNMENT-8

GITHUB LINK: https://github.com/revathiatchi/NeuralAssignment8.git

RECORDINGLINK:
https://github.com/revathiatchi/NeuralAssignment8/assets/156601745/e6dec86e-53bb-4c33-8688-a2e75d788e68

In class programming:

1. Add one more hidden layer to autoencoder

2. Do the prediction on the test data and then visualize one of the reconstructed version of that test data.

Also, visualize the same test data before reconstruction using Matplotlib

3. Repeat the question 2 on the denoising autoencoder

4. plot loss and accuracy using the history object

```
import keras
from keras.models import Sequential
from keras.preprocessing import image
from keras.layers import Activation,Dense,Dropout,Conv2D,Flatten,MaxPooling2D,BatchNormalization
from keras.datasets import cifar10
from keras import optimizers
from matplotlib import pyplot as plt
```

```
#generate cifar10 data
(x_train,y_train),(x_test,y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 3s 0us/step
```

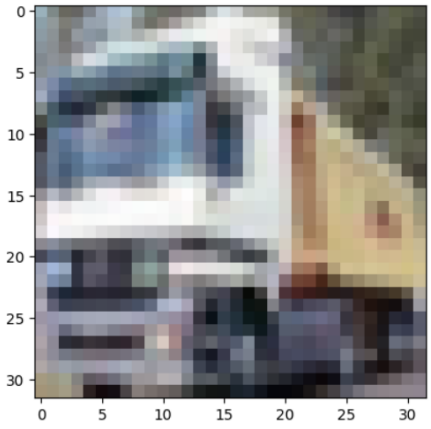
```
#config parameters
num_classes = 10
input_shape = x_train.shape[1:4]
optimizer = optimizers.Adam(lr=0.001)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

```
#convert label to one-hot
one_hot_y_train = keras.utils.to_categorical(y_train,num_classes=num_classes)
one_hot_y_test = keras.utils.to_categorical(y_test,num_classes=num_classes)
```

```
# check data
plt.imshow(x_train[1])
print(x_train[1].shape)
```

```
(32, 32, 3)
```




```
# build model(similar to VGG16, only change the input and output shape)
model = Sequential()
model.add(Conv2D(64,(3,3),activation='relu',input_shape=input_shape,padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))
```

```
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(4096,activation='relu'))
model.add(Dense(2048, activation='relu'))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

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

```
model.summary()
```

```
batch_normalization_7 (Bat  (None, 4, 4, 512)         2048
chNormalization)

conv2d_8 (Conv2D)           (None, 4, 4, 512)         2359808

batch_normalization_8 (Bat  (None, 4, 4, 512)         2048
chNormalization)

conv2d_9 (Conv2D)           (None, 4, 4, 512)         2359808

batch_normalization_9 (Bat  (None, 4, 4, 512)         2048
chNormalization)

max_pooling2d_3 (MaxPoolin  (None, 2, 2, 512)         0
g2D)

dropout_3 (Dropout)         (None, 2, 2, 512)         0

conv2d_10 (Conv2D)          (None, 2, 2, 512)         2359808

batch_normalization_10 (Ba  (None, 2, 2, 512)         2048
tchNormalization)

conv2d_11 (Conv2D)          (None, 2, 2, 512)         2359808

batch_normalization_11 (Ba  (None, 2, 2, 512)         2048
tchNormalization)

conv2d_12 (Conv2D)          (None, 2, 2, 512)         2359808

batch_normalization_12 (Ba  (None, 2, 2, 512)         2048
tchNormalization)

max_pooling2d_4 (MaxPoolin  (None, 1, 1, 512)         0
g2D)

dropout_4 (Dropout)         (None, 1, 1, 512)         0

flatten (Flatten)           (None, 512)               0

dense (Dense)               (None, 4096)              2101248

dense_1 (Dense)             (None, 2048)              8390656

dense_2 (Dense)             (None, 1024)              2098176

dropout_5 (Dropout)         (None, 1024)              0

dense_3 (Dense)             (None, 10)                10250

activation (Activation)     (None, 10)                0

=================================================================
Total params: 27331914 (104.26 MB)
```

```
history = model.fit(x=x_train, y=one_hot_y_train, batch_size=128, epochs=30, validation_split=0.1)
```

```
Epoch 2/30
352/352 [==============================] - 27s 76ms/step - loss: 1.3414 - accuracy: 0.5145 - val_loss: 1.4610 - val_accuracy: 0.5336
Epoch 3/30
352/352 [==============================] - 26s 75ms/step - loss: 1.0524 - accuracy: 0.6334 - val_loss: 1.1668 - val_accuracy: 0.6460
Epoch 4/30
352/352 [==============================] - 27s 76ms/step - loss: 0.8878 - accuracy: 0.7010 - val_loss: 0.8616 - val_accuracy: 0.7158
Epoch 5/30
352/352 [==============================] - 27s 78ms/step - loss: 0.7490 - accuracy: 0.7521 - val_loss: 0.7373 - val_accuracy: 0.7614
Epoch 6/30
352/352 [==============================] - 27s 77ms/step - loss: 0.6575 - accuracy: 0.7859 - val_loss: 0.6574 - val_accuracy: 0.7858
Epoch 7/30
352/352 [==============================] - 27s 78ms/step - loss: 0.5850 - accuracy: 0.8106 - val_loss: 0.7119 - val_accuracy: 0.7792
Epoch 8/30
352/352 [==============================] - 27s 78ms/step - loss: 0.5202 - accuracy: 0.8329 - val_loss: 0.7351 - val_accuracy: 0.7812
Epoch 9/30
352/352 [==============================] - 27s 76ms/step - loss: 0.4735 - accuracy: 0.8485 - val_loss: 0.5936 - val_accuracy: 0.8160
Epoch 10/30
352/352 [==============================] - 28s 78ms/step - loss: 0.4209 - accuracy: 0.8649 - val_loss: 0.7012 - val_accuracy: 0.7878
Epoch 11/30
352/352 [==============================] - 27s 76ms/step - loss: 0.3825 - accuracy: 0.8796 - val_loss: 0.5823 - val_accuracy: 0.8282
Epoch 12/30
352/352 [==============================] - 27s 78ms/step - loss: 0.3312 - accuracy: 0.8956 - val_loss: 0.5935 - val_accuracy: 0.8372
```

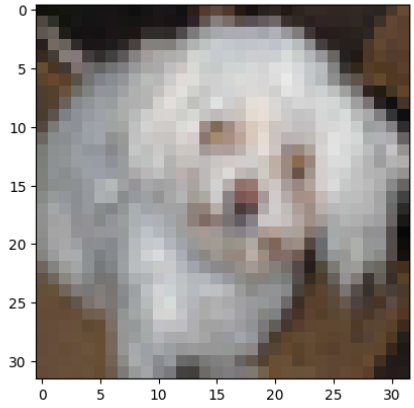
```
Epoch 13/30
352/352 [==============================] - 27s 76ms/step - loss: 0.2984 - accuracy: 0.9043 - val_loss: 0.5957 - val_accuracy: 0.8354
Epoch 14/30
352/352 [==============================] - 27s 76ms/step - loss: 0.2744 - accuracy: 0.9130 - val_loss: 0.6478 - val_accuracy: 0.8296
Epoch 15/30
352/352 [==============================] - 27s 76ms/step - loss: 0.2645 - accuracy: 0.9168 - val_loss: 0.5952 - val_accuracy: 0.8354
Epoch 16/30
352/352 [==============================] - 28s 78ms/step - loss: 0.2290 - accuracy: 0.9286 - val_loss: 0.5574 - val_accuracy: 0.8468
Epoch 17/30
352/352 [==============================] - 27s 78ms/step - loss: 0.1991 - accuracy: 0.9374 - val_loss: 0.6206 - val_accuracy: 0.8480
Epoch 18/30
352/352 [==============================] - 27s 76ms/step - loss: 0.1837 - accuracy: 0.9432 - val_loss: 0.7272 - val_accuracy: 0.8214
Epoch 19/30
352/352 [==============================] - 27s 78ms/step - loss: 0.1732 - accuracy: 0.9458 - val_loss: 0.7186 - val_accuracy: 0.8332
Epoch 20/30
352/352 [==============================] - 27s 78ms/step - loss: 0.1652 - accuracy: 0.9492 - val_loss: 0.8003 - val_accuracy: 0.8114
Epoch 21/30
352/352 [==============================] - 27s 78ms/step - loss: 0.2157 - accuracy: 0.9349 - val_loss: 0.5938 - val_accuracy: 0.8514
Epoch 22/30
352/352 [==============================] - 27s 76ms/step - loss: 0.2040 - accuracy: 0.9387 - val_loss: 0.6173 - val_accuracy: 0.8342
Epoch 23/30
352/352 [==============================] - 27s 78ms/step - loss: 0.1447 - accuracy: 0.9558 - val_loss: 0.6253 - val_accuracy: 0.8552
Epoch 24/30
352/352 [==============================] - 27s 76ms/step - loss: 0.1185 - accuracy: 0.9638 - val_loss: 0.6210 - val_accuracy: 0.8526
Epoch 25/30
352/352 [==============================] - 27s 76ms/step - loss: 0.1094 - accuracy: 0.9677 - val_loss: 0.7091 - val_accuracy: 0.8418
Epoch 26/30
352/352 [==============================] - 27s 77ms/step - loss: 0.0985 - accuracy: 0.9698 - val_loss: 0.7450 - val_accuracy: 0.8446
Epoch 27/30
352/352 [==============================] - 27s 78ms/step - loss: 0.0897 - accuracy: 0.9716 - val_loss: 0.6288 - val_accuracy: 0.8560
Epoch 28/30
352/352 [==============================] - 27s 78ms/step - loss: 0.0896 - accuracy: 0.9727 - val_loss: 0.7102 - val_accuracy: 0.8572
Epoch 29/30
352/352 [==============================] - 27s 78ms/step - loss: 0.0917 - accuracy: 0.9723 - val_loss: 0.7767 - val_accuracy: 0.8402
Epoch 30/30
352/352 [==============================] - 28s 78ms/step - loss: 0.0896 - accuracy: 0.9734 - val_loss: 0.7157 - val_accuracy: 0.8558
```

```
[ ]  # evaluate
     print(model.metrics_names)
     model.evaluate(x=x_test,y=one_hot_y_test,batch_size=512)
```

```
['loss', 'accuracy']
20/20 [==============================] - 5s 123ms/step - loss: 0.7212 - accuracy: 0.8502
[0.7211834192276001, 0.8501999974250793]
```

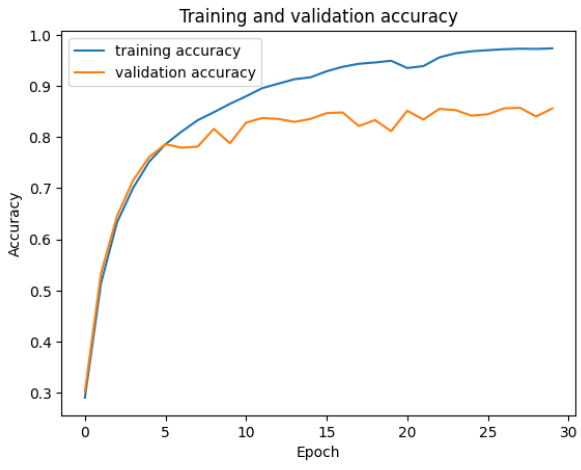
```
[ ]  model.save("keras-VGG16-cifar10.h5")
     plt.imshow(x_test[1000])

     result = model.predict(x_test[1000:1001]).tolist()
     predict = 0
     expect = y_test[1000][0]
     for i,_ in enumerate(result[0]):
        if result[0][i] > result[0][predict]:
         predict = i
     print("predict class:",predict)
     print("expected class:",expect)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
  saving_api.save_model(
1/1 [==============================] - 1s 784ms/step
predict class: 5
expected class: 5
```

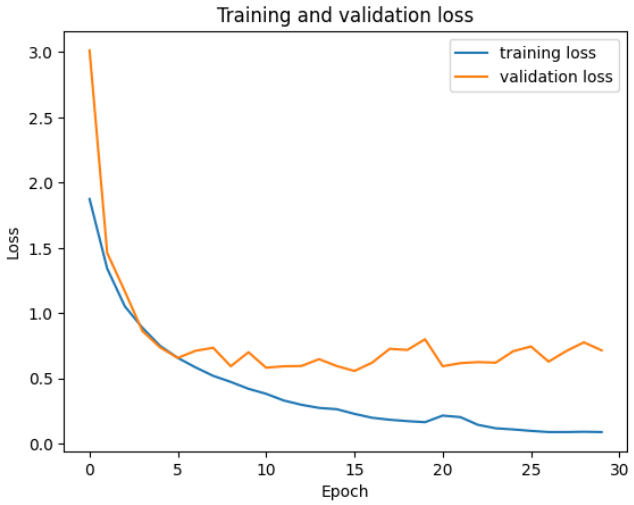


```
[ ] # save model
    model.save("keras-VGG16-cifar10.h5")
```

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

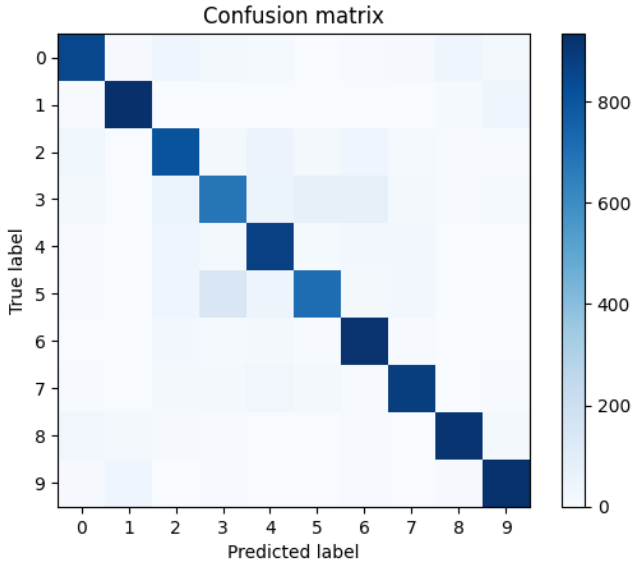


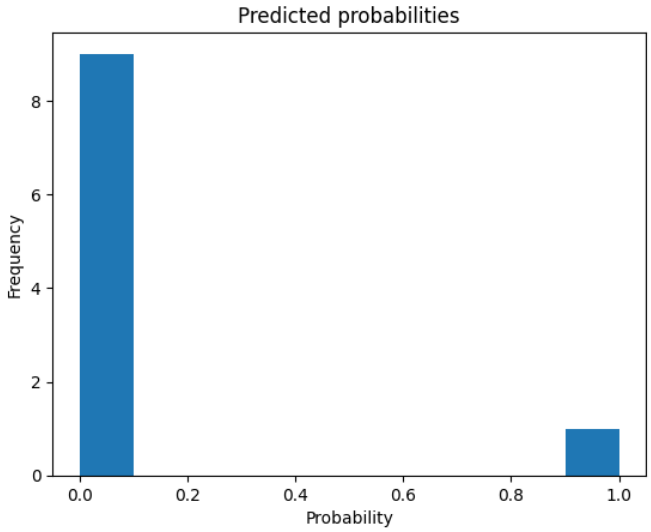
```
#plot the training and validation loss
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
import numpy as np
from sklearn.metrics import confusion_matrix

# calculate the confusion matrix
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = y_test.ravel()
cm = confusion_matrix(y_true, y_pred_classes)

# plot the confusion matrix
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion matrix')
plt.colorbar()
tick_marks = np.arange(num_classes)
plt.xticks(tick_marks, range(num_classes))
plt.yticks(tick_marks, range(num_classes))
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()

# plot a histogram of the predicted probabilities for a sample image
plt.hist(y_pred[1000])
plt.title('Predicted probabilities')
plt.xlabel('Probability')
plt.ylabel('Frequency')
plt.show()
```

```
313/313 [==============================] - 4s 11ms/step
```





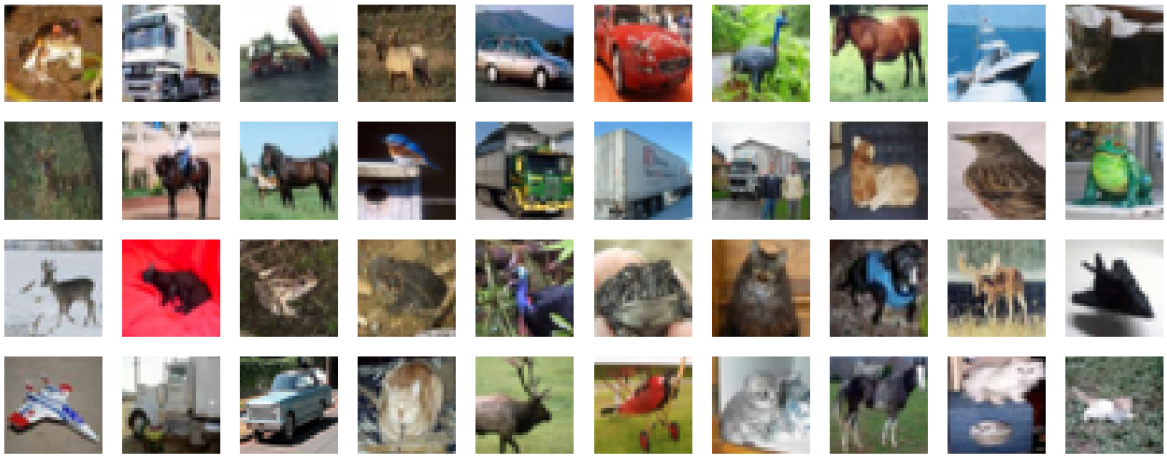
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.datasets import mnist

from tensorflow.keras.optimizers import RMSprop
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout, BatchNormalization
from keras.datasets import cifar10

%matplotlib inline
```

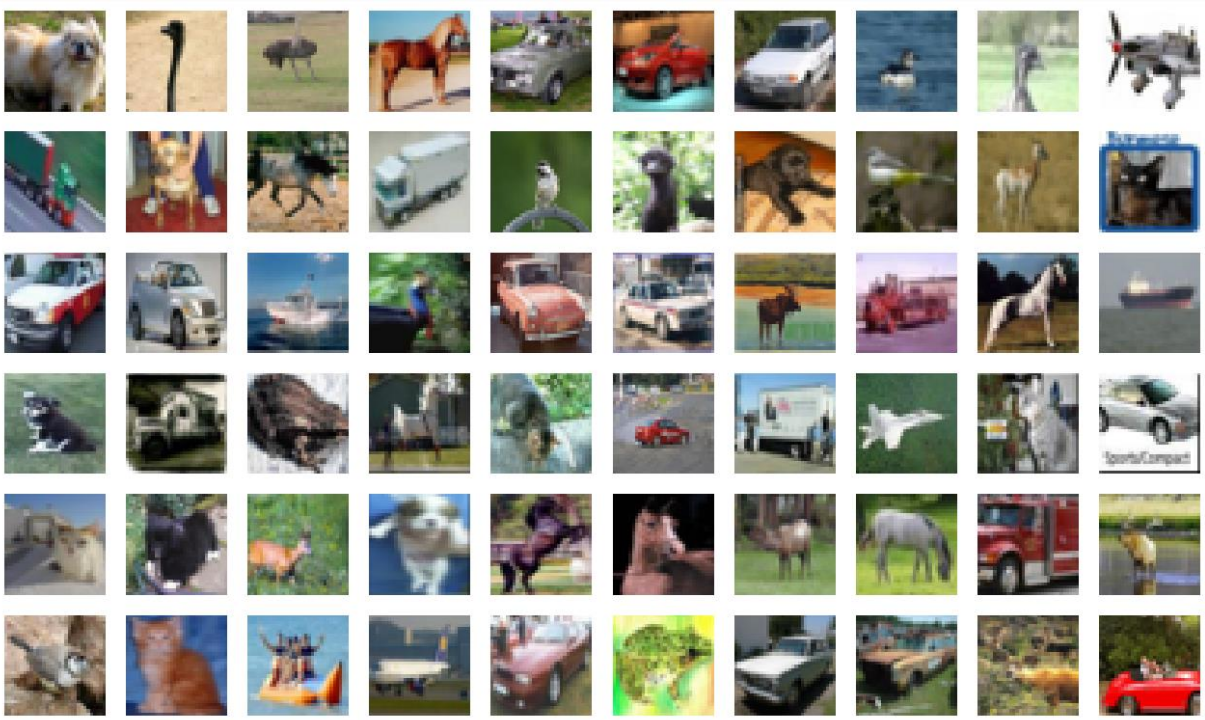
```
#cifar100 = tf.keras.datasets.cifar100
(X_train,Y_train) , (X_test,Y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
```

```
classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

```
plt.figure(figsize = (16,16))
for i in range(100):
  plt.subplot(10,10,1+i)
  plt.axis('off')
  plt.imshow(X_train[i], cmap = 'gray')
```





```
[7]  from sklearn.model_selection import train_test_split
     x_train, x_val, y_train, y_val = train_test_split(X_train,Y_train,test_size=0.2)
```

```
[8]  from tensorflow.keras.utils import to_categorical
     y_train = to_categorical(y_train, num_classes = 10)
     y_val = to_categorical(y_val, num_classes = 10)
```

```
[9]  print(x_train.shape)
     print(y_train.shape)
     print(x_val.shape)
     print(y_val.shape)
     print(X_test.shape)
     print(Y_test.shape)
```

```
     (40000, 32, 32, 3)
     (40000, 10)
     (10000, 32, 32, 3)
     (10000, 10)
     (10000, 32, 32, 3)
     (10000, 1)
```

```
[10] train_datagen = ImageDataGenerator(
         preprocessing_function = tf.keras.applications.vgg19.preprocess_input,
         rotation_range=10,
         zoom_range = 0.1,
         width_shift_range = 0.1,
         height_shift_range = 0.1,
         shear_range = 0.1,
         horizontal_flip = True
     )
     train_datagen.fit(x_train)

     val_datagen = ImageDataGenerator(preprocessing_function = tf.keras.applications.vgg19.preprocess_input)
     val_datagen.fit(x_val)
```

```
[11] from keras.callbacks import ReduceLROnPlateau
     learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                                 patience=3,
                                                 verbose=1,
                                                 factor=0.5,
                                                 min_lr=0.00001)
```

```
[12] vgg_model = tf.keras.applications.VGG19(
         include_top=False,
         weights=None,
         input_shape=(32,32,3),
     )

     vgg_model.summary()
```

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 block1_conv1 (Conv2D)       (None, 32, 32, 64)        1792

 block1_conv2 (Conv2D)       (None, 32, 32, 64)        36928

 block1_pool (MaxPooling2D)  (None, 16, 16, 64)        0

 block2_conv1 (Conv2D)       (None, 16, 16, 128)       73856

 block2_conv2 (Conv2D)       (None, 16, 16, 128)       147584

 block2_pool (MaxPooling2D)  (None, 8, 8, 128)         0

 block3_conv1 (Conv2D)       (None, 8, 8, 256)         295168

 block3_conv2 (Conv2D)       (None, 8, 8, 256)         590080

 block3_conv3 (Conv2D)       (None, 8, 8, 256)         590080

 block3_conv4 (Conv2D)       (None, 8, 8, 256)         590080

 block3_pool (MaxPooling2D)  (None, 4, 4, 256)         0

 block4_conv1 (Conv2D)       (None, 4, 4, 512)         1180160

 block4_conv2 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_conv3 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_conv4 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 2, 2, 512)         0

 block5_conv1 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv2 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv3 (Conv2D)       (None, 2, 2, 512)         2359808
```

```
 block5_conv4 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

=================================================================
Total params: 20024384 (76.39 MB)
Trainable params: 20024384 (76.39 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model = tf.keras.Sequential()
model.add(vgg_model)
model.add(Flatten())
model.add(Dense(1024, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dense(1024, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dense(256, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(10, activation = 'softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 1, 1, 512)         20024384

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 1024)              525312

 batch_normalization (Batch  (None, 1024)              4096
 Normalization)

 dense_1 (Dense)             (None, 1024)              1049600

 batch_normalization_1 (Bat  (None, 1024)              4096
 chNormalization)

 dense_2 (Dense)             (None, 256)               262400

 batch_normalization_2 (Bat  (None, 256)               1024
 chNormalization)

 dropout (Dropout)           (None, 256)               0

 dense_3 (Dense)             (None, 10)                2570

=================================================================
Total params: 21873482 (83.44 MB)
```

```
=================================================================
Total params: 21873482 (83.44 MB)
Trainable params: 21868874 (83.42 MB)
Non-trainable params: 4608 (18.00 KB)
_________________________________________________________________
```

```
optimizer = tf.keras.optimizers.SGD(learning_rate = 0.001, momentum = 0.9)
model.compile(optimizer= optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```