Assignment 06

Author: Anjani Bonda

Date: 4/22/2023

Assignment 6.1

In [1]:
```
# Load all the required libraries
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from keras import models
from matplotlib import pyplot
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
from keras.optimizers import SGD, Adam
```

Load the MNIST dataset

In [2]:
```
# Load the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Model Building

In [4]:
```
## Instantiate a convnet
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
```

In [5]:
```
## Add classifer on top of convnet
model.add(layers.Flatten())
model.add(layers.Dense(10, activation='softmax'))
```

In [6]:
```
## Show model summary
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_3 (Conv2D)           (None, 26, 26, 32)        320       
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 13, 13, 32)       0         
 2D)                                                             
                                                                 
 conv2d_4 (Conv2D)           (None, 11, 11, 64)        18496     
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 5, 5, 64)         0         
 2D)                                                             
                                                                 
 conv2d_5 (Conv2D)           (None, 3, 3, 128)         73856     
                                                                 
 flatten (Flatten)           (None, 1152)              0         
                                                                 
 dense (Dense)               (None, 10)                11530     
                                                                 
=================================================================
Total params: 104,202
Trainable params: 104,202
Non-trainable params: 0
_________________________________________________________________
```

In [7]:
```
# Compile the model
model.compile(optimizer="rmsprop",
    loss="categorical_crossentropy",
    metrics=["accuracy"])
```

Model Validation

In [8]:
```
## Set aside a validation set (10000 samples)
# Data
validation_images = train_images[:10000]
partial_train_images = train_images[10000:]
# Labels
validation_labels = train_labels[:10000]
partial_train_labels = train_labels[10000:]
```

Model Training

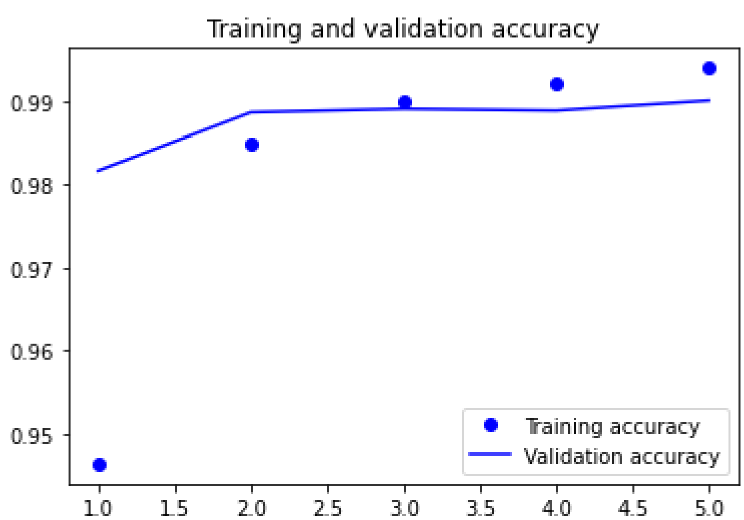
In [9]:
```
# Train the model
history = model.fit(partial_train_images,
                   partial_train_labels,
                   epochs=5,
                   batch_size=64,
                   validation_data=(validation_images, validation_labels))
```

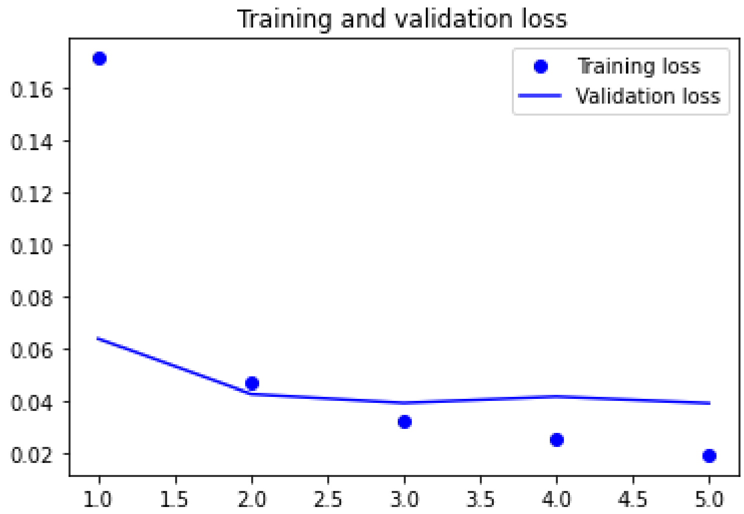
```
Epoch 1/5
782/782 [==============================] - 48s 61ms/step - loss: 0.1716 - accu
racy: 0.9464 - val_loss: 0.0637 - val_accuracy: 0.9817
Epoch 2/5
782/782 [==============================] - 45s 58ms/step - loss: 0.0471 - accu
racy: 0.9850 - val_loss: 0.0426 - val_accuracy: 0.9887
Epoch 3/5
782/782 [==============================] - 46s 59ms/step - loss: 0.0325 - accu
racy: 0.9901 - val_loss: 0.0392 - val_accuracy: 0.9891
Epoch 4/5
782/782 [==============================] - 47s 60ms/step - loss: 0.0251 - accu
racy: 0.9921 - val_loss: 0.0416 - val_accuracy: 0.9889
Epoch 5/5
782/782 [==============================] - 50s 64ms/step - loss: 0.0192 - accu
racy: 0.9940 - val_loss: 0.0391 - val_accuracy: 0.9901
```

Plotting Model Output and Loss

In [10]:

```
# Plot the training and validation accuracy and loss
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```





Evaluate the Model

In [11]:
```
# Evaluate the convnet
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc*100:.1f}%')
print(f'Test loss: {test_loss:.3f}')
```

```
313/313 [==============================] - 3s 9ms/step - loss: 0.0254 - accura
cy: 0.9920
Test accuracy: 99.2%
Test loss: 0.025
```

The model accuracy is 99% and loss is only 0.025; The accuracy is increased significantly and loss is reduced a lot by adding Conv2D and MaxPooling2D layers

Save Model

In [25]:
```
model.save('results/mnist')
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _j
it_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
3 of 3). These functions will not be directly callable after loading.
INFO:tensorflow:Assets written to: results/mnist/assets
INFO:tensorflow:Assets written to: results/mnist/assets
```

Assignment 6.2

Assignment 6.2.a

Load the data & Data preparation

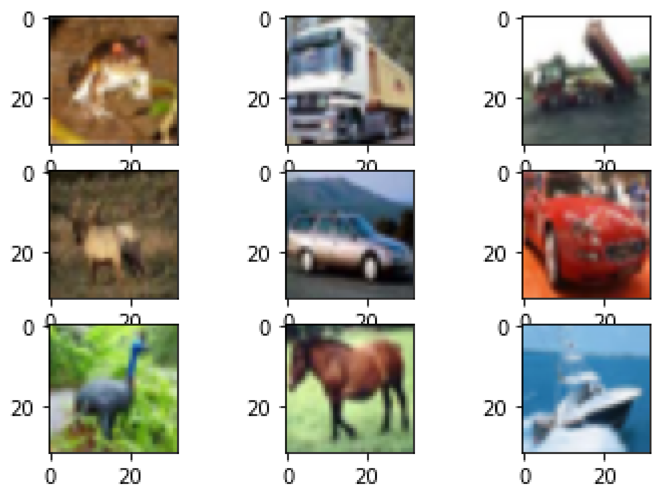
In [14]:
```
# Load the CIFAR10 data set
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
assert x_train.shape == (50000, 32, 32, 3)
assert x_test.shape == (10000, 32, 32, 3)
assert y_train.shape == (50000, 1)
assert y_test.shape == (10000, 1)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 7s 0us/step
```

In [15]:
```
# summarize loaded dataset
print('Train: X=%s, y=%s' % (x_train.shape, y_train.shape))
print('Test: X=%s, y=%s' % (x_test.shape, y_test.shape))
```

```
Train: X=(50000, 32, 32, 3), y=(50000, 1)
Test: X=(10000, 32, 32, 3), y=(10000, 1)
```

In [16]:
```
# plot first few images
for i in range(9):
    # define subplot
    pyplot.subplot(330 + 1 + i)
    # plot raw pixel data
    pyplot.imshow(x_train[i])
# show the figure
pyplot.show()
```



In [17]:
```
## Set aside a validation set (10,000 samples)
# Data
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
# Labels
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

Model Building without dropout or data-augmentation

In [18]:
```
## Instantiate a convnet
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.experimental.preprocessing.Rescaling(1./255))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
```

In [19]:
```
## Add classifer on top of convnet
model.add(layers.Flatten())
model.add(layers.Dense(10, activation='softmax'))
```

In [20]:
```
## Show model summary
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_6 (Conv2D)           (None, 30, 30, 32)        896       
                                                                 
 rescaling (Rescaling)       (None, 30, 30, 32)        0         
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 15, 15, 32)       0         
 2D)                                                             
                                                                 
 conv2d_7 (Conv2D)           (None, 13, 13, 64)        18496     
                                                                 
 max_pooling2d_5 (MaxPooling  (None, 6, 6, 64)         0         
 2D)                                                             
                                                                 
 conv2d_8 (Conv2D)           (None, 4, 4, 128)         73856     
                                                                 
 max_pooling2d_6 (MaxPooling  (None, 2, 2, 128)        0         
 2D)                                                             
                                                                 
 flatten_1 (Flatten)         (None, 512)               0         
                                                                 
 dense_1 (Dense)             (None, 10)                5130      
                                                                 
=================================================================
Total params: 98,378
Trainable params: 98,378
Non-trainable params: 0
_________________________________________________________________
```

In [21]:
```
# Compile model
model.compile(optimizer="rmsprop",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"])
```

Model Training

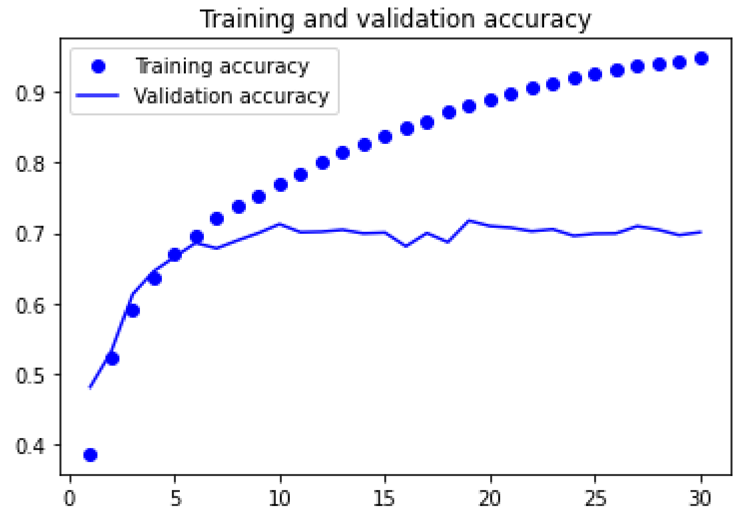
In [22]:
```
# Train model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=30,
                    batch_size=64,
                    validation_data=(x_val, y_val))
```

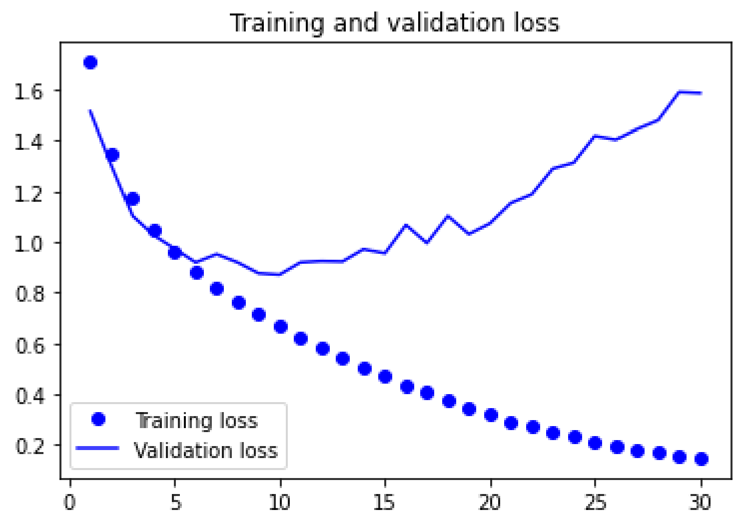
```
Epoch 1/30
625/625 [==============================] - 52s 82ms/step - loss: 1.7090 - accu
racy: 0.3868 - val_loss: 1.5139 - val_accuracy: 0.4812
Epoch 2/30
625/625 [==============================] - 51s 82ms/step - loss: 1.3484 - accu
racy: 0.5237 - val_loss: 1.3027 - val_accuracy: 0.5313
Epoch 3/30
625/625 [==============================] - 51s 82ms/step - loss: 1.1714 - accu
racy: 0.5907 - val_loss: 1.1026 - val_accuracy: 0.6129
Epoch 4/30
625/625 [==============================] - 51s 82ms/step - loss: 1.0516 - accu
racy: 0.6366 - val_loss: 1.0262 - val_accuracy: 0.6454
Epoch 5/30
625/625 [==============================] - 52s 83ms/step - loss: 0.9587 - accu
racy: 0.6717 - val_loss: 0.9731 - val_accuracy: 0.6658
Epoch 6/30
625/625 [==============================] - 53s 84ms/step - loss: 0.8819 - accu
racy: 0.6965 - val_loss: 0.9189 - val_accuracy: 0.6858
Epoch 7/30
625/625 [==============================] - 55s 88ms/step - loss: 0.8207 - accu
racy: 0.7209 - val_loss: 0.9514 - val_accuracy: 0.6780
Epoch 8/30
625/625 [==============================] - 55s 88ms/step - loss: 0.7647 - accu
racy: 0.7393 - val_loss: 0.9190 - val_accuracy: 0.6893
Epoch 9/30
625/625 [==============================] - 55s 88ms/step - loss: 0.7125 - accu
racy: 0.7530 - val_loss: 0.8759 - val_accuracy: 0.6998
Epoch 10/30
625/625 [==============================] - 55s 87ms/step - loss: 0.6661 - accu
racy: 0.7708 - val_loss: 0.8704 - val_accuracy: 0.7124
Epoch 11/30
625/625 [==============================] - 56s 90ms/step - loss: 0.6207 - accu
racy: 0.7850 - val_loss: 0.9192 - val_accuracy: 0.7009
Epoch 12/30
625/625 [==============================] - 51s 82ms/step - loss: 0.5799 - accu
racy: 0.8008 - val_loss: 0.9242 - val_accuracy: 0.7016
Epoch 13/30
625/625 [==============================] - 53s 85ms/step - loss: 0.5394 - accu
racy: 0.8141 - val_loss: 0.9224 - val_accuracy: 0.7044
Epoch 14/30
625/625 [==============================] - 51s 81ms/step - loss: 0.5042 - accu
racy: 0.8253 - val_loss: 0.9711 - val_accuracy: 0.6992
Epoch 15/30
625/625 [==============================] - 52s 83ms/step - loss: 0.4723 - accu
racy: 0.8376 - val_loss: 0.9550 - val_accuracy: 0.7004
Epoch 16/30
625/625 [==============================] - 51s 82ms/step - loss: 0.4360 - accu
racy: 0.8493 - val_loss: 1.0670 - val_accuracy: 0.6809
Epoch 17/30
625/625 [==============================] - 52s 83ms/step - loss: 0.4057 - accu
racy: 0.8583 - val_loss: 0.9952 - val_accuracy: 0.6997
Epoch 18/30
625/625 [==============================] - 52s 83ms/step - loss: 0.3746 - accu
racy: 0.8714 - val_loss: 1.1025 - val_accuracy: 0.6868
Epoch 19/30
625/625 [==============================] - 53s 84ms/step - loss: 0.3487 - accu
racy: 0.8791 - val_loss: 1.0307 - val_accuracy: 0.7174
Epoch 20/30
625/625 [==============================] - 54s 86ms/step - loss: 0.3209 - accu
racy: 0.8897 - val_loss: 1.0726 - val_accuracy: 0.7097
```

```
Epoch 21/30
625/625 [==============================] - 55s 88ms/step - loss: 0.2938 - accu
racy: 0.8977 - val_loss: 1.1536 - val_accuracy: 0.7073
Epoch 22/30
625/625 [==============================] - 54s 86ms/step - loss: 0.2740 - accu
racy: 0.9054 - val_loss: 1.1880 - val_accuracy: 0.7023
Epoch 23/30
625/625 [==============================] - 51s 81ms/step - loss: 0.2504 - accu
racy: 0.9128 - val_loss: 1.2892 - val_accuracy: 0.7051
Epoch 24/30
625/625 [==============================] - 51s 81ms/step - loss: 0.2343 - accu
racy: 0.9191 - val_loss: 1.3127 - val_accuracy: 0.6958
Epoch 25/30
625/625 [==============================] - 51s 82ms/step - loss: 0.2134 - accu
racy: 0.9265 - val_loss: 1.4179 - val_accuracy: 0.6988
Epoch 26/30
625/625 [==============================] - 52s 84ms/step - loss: 0.1982 - accu
racy: 0.9306 - val_loss: 1.4034 - val_accuracy: 0.6992
Epoch 27/30
625/625 [==============================] - 53s 84ms/step - loss: 0.1810 - accu
racy: 0.9371 - val_loss: 1.4447 - val_accuracy: 0.7095
Epoch 28/30
625/625 [==============================] - 52s 83ms/step - loss: 0.1696 - accu
racy: 0.9402 - val_loss: 1.4791 - val_accuracy: 0.7045
Epoch 29/30
625/625 [==============================] - 54s 86ms/step - loss: 0.1591 - accu
racy: 0.9434 - val_loss: 1.5899 - val_accuracy: 0.6968
Epoch 30/30
625/625 [==============================] - 55s 88ms/step - loss: 0.1486 - accu
racy: 0.9477 - val_loss: 1.5861 - val_accuracy: 0.7009
```

Plot the result

In [23]:
```
# Plot the training and validation accuracy and loss
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```





Evaluate the Model

In [24]:
```
# Evaluate the convnet
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc*100:.1f}%')
print(f'Test loss: {test_loss:.3f}')
```

```
313/313 [==============================] - 4s 11ms/step - loss: 1.6850 - accur
acy: 0.6929
Test accuracy: 69.3%
Test loss: 1.685
```

The accuracy score without dropout and data augmentation turned out as 69% and loss is 1.68

Save Model

In [26]:
```
model.save('results/without_dropout_augmentation')
```

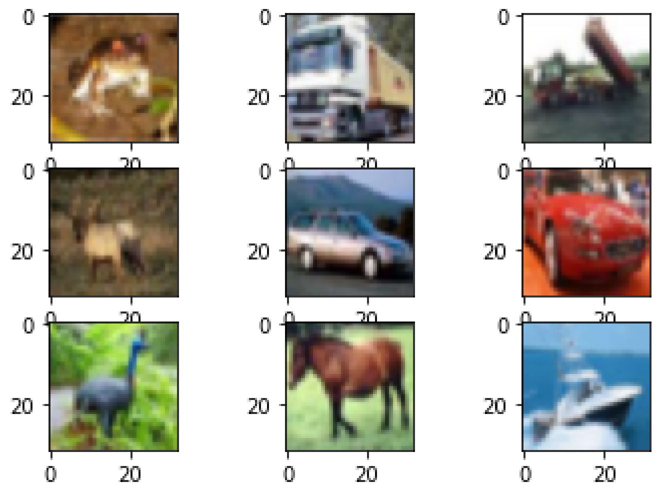
```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _j
it_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
3 of 3). These functions will not be directly callable after loading.
INFO:tensorflow:Assets written to: results/without_dropout_augmentation/assets
```

```
INFO:tensorflow:Assets written to: results/without_dropout_augmentation/assets
```

Assignment 6.2.b

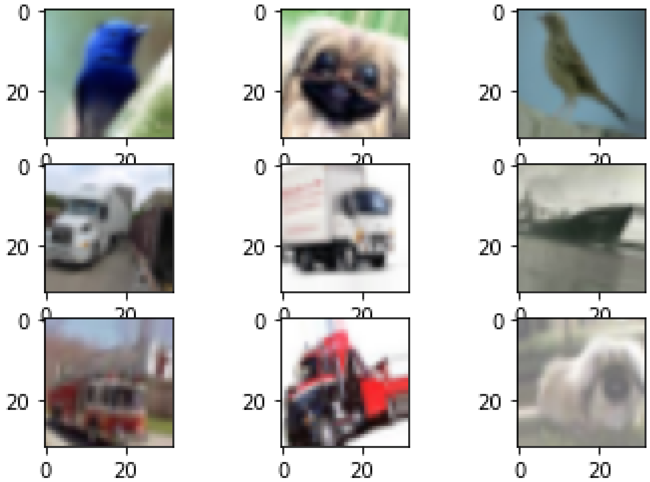
Model Building with dropout or data-augmentation

In [27]:
```
#load data
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
img_rows, img_cols , channels= 32,32,3
for i in range(0,9):
    plt.subplot(330 + 1 + i)
    plt.imshow(x_train[i])
plt.show()
```



In [28]:
```
# set up image augmentation
datagen = ImageDataGenerator(
    rotation_range=15,
    horizontal_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1
    #zoom_range=0.3
    )
datagen.fit(x_train)
```

In [29]:
```
# see example augmentation images
for X_batch, y_batch in datagen.flow(x_train, y_train, batch_size=9):
    for i in range(0, 9):
        plt.subplot(330 + 1 + i)
        plt.imshow(X_batch[i].astype(np.uint8))
    plt.show()
    break
```



In [30]:
```
#reshape into images
x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, channels)
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, channels)
input_shape = (img_rows, img_cols, 1)
print('x_train shape:', x_train.shape)
print(x_train.shape, 'train samples')
print(x_test.shape, 'test samples')
print(y_train.shape, 'target train samples')
print(y_test.shape, 'target test samples')
```

```
x_train shape: (50000, 32, 32, 3)
(50000, 32, 32, 3) train samples
(10000, 32, 32, 3) test samples
(50000, 1) target train samples
(10000, 1) target test samples
```

In [31]:
```
#convert integers to float; normalise and center the mean
x_train=x_train.astype("float32")
x_test=x_test.astype("float32")
mean=np.mean(x_train)
std=np.std(x_train)
x_test=(x_test-mean)/std
x_train=(x_train-mean)/std
```

In [32]:
```
# labels
num_classes=10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Model Building

In [33]:
```
# Build model with droupout

adm2=Adam(lr=0.001,decay=0, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
opt2=adm2

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_regularizer=None,
model.add(layers.BatchNormalization(axis=-1))
model.add(layers.Conv2D(32, (3, 3), activation='relu',kernel_regularizer=None,p
model.add(layers.BatchNormalization(axis=-1))
```

```
model.add(layers.MaxPooling2D(pool_size=(2, 2)))   # reduces to 16x16x3x32
model.add(layers.Dropout(0.5))

model.add(layers.Conv2D(64, (3, 3), activation='relu',kernel_regularizer=None,p
model.add(layers.BatchNormalization(axis=-1))
model.add(layers.Conv2D(64, (3, 3), activation='relu',kernel_regularizer=None,p
model.add(layers.BatchNormalization(axis=-1))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))   # reduces to 8x8x3x(2*32)
model.add(layers.Dropout(0.5))

model.add(layers.Conv2D(128, (3, 3), activation='relu',kernel_regularizer=None,
model.add(layers.BatchNormalization(axis=-1))
model.add(layers.Conv2D(128, (3, 3), activation='relu',kernel_regularizer=None,
model.add(layers.BatchNormalization(axis=-1))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))   # reduces to 4x4x3x(4*32)
model.add(layers.Dropout(0.5))

model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu',kernel_regularizer=None))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10, activation='softmax'))

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

```
/Users/anjanibonda/opt/anaconda3/lib/python3.9/site-packages/keras/optimizers/
optimizer_v2/adam.py:110: UserWarning: The `lr` argument is deprecated, use `l
earning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

In [34]:

```
## print the model summary
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_9 (Conv2D)           (None, 32, 32, 32)        896       
                                                                 
 batch_normalization (BatchN  (None, 32, 32, 32)       128       
 ormalization)                                                   
                                                                 
 conv2d_10 (Conv2D)          (None, 32, 32, 32)        9248      
                                                                 
 batch_normalization_1 (Batc  (None, 32, 32, 32)       128       
 hNormalization)                                                 
                                                                 
 max_pooling2d_7 (MaxPooling  (None, 16, 16, 32)       0         
 2D)                                                             
                                                                 
 dropout (Dropout)           (None, 16, 16, 32)        0         
                                                                 
 conv2d_11 (Conv2D)          (None, 16, 16, 64)        18496     
                                                                 
 batch_normalization_2 (Batc  (None, 16, 16, 64)       256       
 hNormalization)                                                 
                                                                 
 conv2d_12 (Conv2D)          (None, 16, 16, 64)        36928     
                                                                 
 batch_normalization_3 (Batc  (None, 16, 16, 64)       256       
 hNormalization)                                                 
                                                                 
 max_pooling2d_8 (MaxPooling  (None, 8, 8, 64)         0         
 2D)                                                             
                                                                 
 dropout_1 (Dropout)         (None, 8, 8, 64)          0         
                                                                 
 conv2d_13 (Conv2D)          (None, 8, 8, 128)         73856     
                                                                 
 batch_normalization_4 (Batc  (None, 8, 8, 128)        512       
 hNormalization)                                                 
                                                                 
 conv2d_14 (Conv2D)          (None, 8, 8, 128)         147584    
                                                                 
 batch_normalization_5 (Batc  (None, 8, 8, 128)        512       
 hNormalization)                                                 
                                                                 
 max_pooling2d_9 (MaxPooling  (None, 4, 4, 128)        0         
 2D)                                                             
                                                                 
 dropout_2 (Dropout)         (None, 4, 4, 128)         0         
                                                                 
 flatten_2 (Flatten)         (None, 2048)              0         
                                                                 
 dense_2 (Dense)             (None, 512)               1049088   
                                                                 
 batch_normalization_6 (Batc  (None, 512)              2048      
 hNormalization)                                                 
                                                                 
 dropout_3 (Dropout)         (None, 512)               0         
                                                                 
 dense_3 (Dense)             (None, 10)                5130      
```

```
=================================================================
Total params: 1,345,066
Trainable params: 1,343,146
Non-trainable params: 1,920
_________________________________________________________________
```

Train the model

In [35]:
```
# train with image augmentation
history=model.fit(datagen.flow(x_train, y_train, batch_size=128),
                    steps_per_epoch = len(x_train) / 128, epochs=30, validatior
```

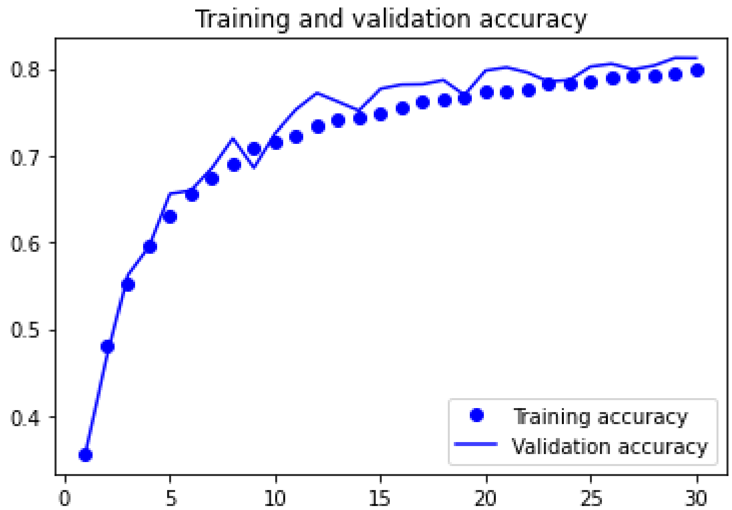
```
Epoch 1/30
390/390 [==============================] - 339s 864ms/step - loss: 1.9591 - ac
curacy: 0.3563 - val_loss: 1.9191 - val_accuracy: 0.3570
Epoch 2/30
390/390 [==============================] - 341s 871ms/step - loss: 1.4479 - ac
curacy: 0.4809 - val_loss: 1.6224 - val_accuracy: 0.4659
Epoch 3/30
390/390 [==============================] - 349s 894ms/step - loss: 1.2463 - ac
curacy: 0.5523 - val_loss: 1.2755 - val_accuracy: 0.5622
Epoch 4/30
390/390 [==============================] - 355s 909ms/step - loss: 1.1281 - ac
curacy: 0.5959 - val_loss: 1.2408 - val_accuracy: 0.5947
Epoch 5/30
390/390 [==============================] - 348s 891ms/step - loss: 1.0364 - ac
curacy: 0.6317 - val_loss: 1.0174 - val_accuracy: 0.6560
Epoch 6/30
390/390 [==============================] - 353s 904ms/step - loss: 0.9697 - ac
curacy: 0.6553 - val_loss: 1.0196 - val_accuracy: 0.6598
Epoch 7/30
390/390 [==============================] - 315s 807ms/step - loss: 0.9251 - ac
curacy: 0.6739 - val_loss: 0.9220 - val_accuracy: 0.6856
Epoch 8/30
390/390 [==============================] - 333s 852ms/step - loss: 0.8750 - ac
curacy: 0.6903 - val_loss: 0.8351 - val_accuracy: 0.7199
Epoch 9/30
390/390 [==============================] - 310s 794ms/step - loss: 0.8365 - ac
curacy: 0.7080 - val_loss: 0.9565 - val_accuracy: 0.6859
Epoch 10/30
390/390 [==============================] - 309s 792ms/step - loss: 0.8099 - ac
curacy: 0.7165 - val_loss: 0.8182 - val_accuracy: 0.7253
Epoch 11/30
390/390 [==============================] - 308s 788ms/step - loss: 0.7912 - ac
curacy: 0.7238 - val_loss: 0.7294 - val_accuracy: 0.7527
Epoch 12/30
390/390 [==============================] - 310s 794ms/step - loss: 0.7621 - ac
curacy: 0.7336 - val_loss: 0.6782 - val_accuracy: 0.7717
Epoch 13/30
390/390 [==============================] - 310s 793ms/step - loss: 0.7451 - ac
curacy: 0.7404 - val_loss: 0.7076 - val_accuracy: 0.7617
Epoch 14/30
390/390 [==============================] - 310s 794ms/step - loss: 0.7361 - ac
curacy: 0.7445 - val_loss: 0.7472 - val_accuracy: 0.7515
Epoch 15/30
390/390 [==============================] - 311s 796ms/step - loss: 0.7196 - ac
curacy: 0.7489 - val_loss: 0.6832 - val_accuracy: 0.7766
Epoch 16/30
390/390 [==============================] - 311s 797ms/step - loss: 0.6978 - ac
curacy: 0.7561 - val_loss: 0.6456 - val_accuracy: 0.7814
Epoch 17/30
390/390 [==============================] - 309s 791ms/step - loss: 0.6883 - ac
curacy: 0.7620 - val_loss: 0.6508 - val_accuracy: 0.7818
Epoch 18/30
390/390 [==============================] - 310s 794ms/step - loss: 0.6702 - ac
curacy: 0.7645 - val_loss: 0.6286 - val_accuracy: 0.7866
Epoch 19/30
390/390 [==============================] - 316s 810ms/step - loss: 0.6704 - ac
curacy: 0.7670 - val_loss: 0.6918 - val_accuracy: 0.7694
Epoch 20/30
390/390 [==============================] - 309s 792ms/step - loss: 0.6557 - ac
curacy: 0.7729 - val_loss: 0.5915 - val_accuracy: 0.7976
```

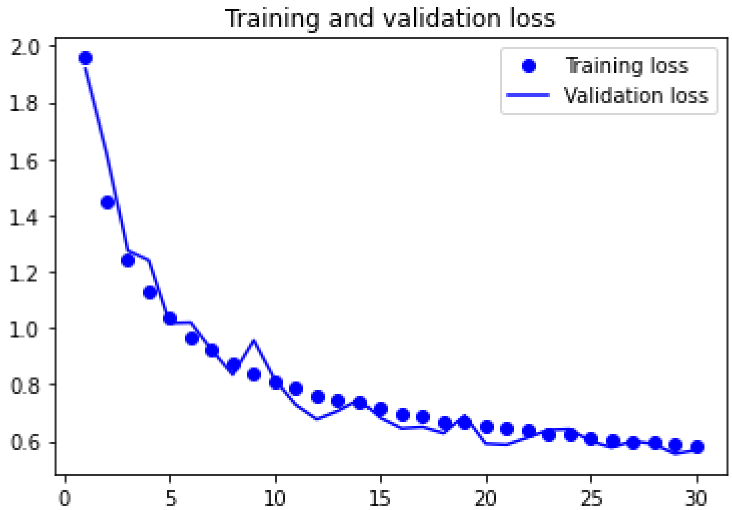
```
Epoch 21/30
390/390 [==============================] - 310s 795ms/step - loss: 0.6487 - ac
curacy: 0.7742 - val_loss: 0.5878 - val_accuracy: 0.8013
Epoch 22/30
390/390 [==============================] - 310s 793ms/step - loss: 0.6420 - ac
curacy: 0.7762 - val_loss: 0.6137 - val_accuracy: 0.7954
Epoch 23/30
390/390 [==============================] - 310s 793ms/step - loss: 0.6290 - ac
curacy: 0.7817 - val_loss: 0.6402 - val_accuracy: 0.7849
Epoch 24/30
390/390 [==============================] - 309s 791ms/step - loss: 0.6229 - ac
curacy: 0.7826 - val_loss: 0.6433 - val_accuracy: 0.7867
Epoch 25/30
390/390 [==============================] - 309s 792ms/step - loss: 0.6118 - ac
curacy: 0.7856 - val_loss: 0.6005 - val_accuracy: 0.8022
Epoch 26/30
390/390 [==============================] - 311s 796ms/step - loss: 0.6043 - ac
curacy: 0.7886 - val_loss: 0.5780 - val_accuracy: 0.8054
Epoch 27/30
390/390 [==============================] - 313s 801ms/step - loss: 0.5975 - ac
curacy: 0.7930 - val_loss: 0.6012 - val_accuracy: 0.7989
Epoch 28/30
390/390 [==============================] - 311s 795ms/step - loss: 0.5940 - ac
curacy: 0.7930 - val_loss: 0.5899 - val_accuracy: 0.8034
Epoch 29/30
390/390 [==============================] - 312s 797ms/step - loss: 0.5889 - ac
curacy: 0.7948 - val_loss: 0.5555 - val_accuracy: 0.8122
Epoch 30/30
390/390 [==============================] - 312s 800ms/step - loss: 0.5843 - ac
curacy: 0.7978 - val_loss: 0.5691 - val_accuracy: 0.8120
```

Plot the model

In [36]:

```
# Plot the training and validation accuracy and loss
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```





Model Evaluation

In [37]:
```
# Evaluate the convnet
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc*100:.1f}%')
print(f'Test loss: {test_loss:.3f}')
```

```
313/313 [==============================] - 18s 56ms/step - loss: 0.5691 - accu
racy: 0.8120
Test accuracy: 81.2%
Test loss: 0.569
```

The accuracy scre with dropout and data augmentation has been increased to 81%

Save the model

In [38]:
```
model.save('results/with_dropout_augmentation')
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _j
it_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convol
ution_op, _jit_compiled_convolution_op while saving (showing 5 of 6). These fu
nctions will not be directly callable after loading.
INFO:tensorflow:Assets written to: results/with_dropout_augmentation/assets
INFO:tensorflow:Assets written to: results/with_dropout_augmentation/assets
```

Assignment 6.3

Load libraries

In [49]:
```
# Load libraries
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_pre
import numpy as np
from IPython.display import Image, display
import os
```

Define Model

In [50]:
```
# Load model
model = ResNet50(weights='imagenet')
```

Image Classification

In [51]:
```
## Custom function to predict the input image using resnet50
def image_prediction(img_input):
    ## model prediction and printing result
    img_path = img_input
    img = image.load_img(img_path, target_size=(224, 224))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)

    preds = model.predict(x)

    # decode the results into a list of tuples (class, description, probability
    print("Displaying the prediction result for the image: {}".format(image))
    print('Predicted:', decode_predictions(preds, top=3)[0])
```

In [52]:
```
## Read the image present in images directory
## call image_prediction function
for img in os.listdir('images'):
    input_img = "images/"+img
    print('\nDisplaying {} image'.format(img))
    dis = Image(filename=input_img)
    display(dis)
    image_prediction(input_img)
```

```
Displaying dog.jpg image
```

```
1/1 [==============================] - 2s 2s/step
Displaying the prediction result for the image: <module 'keras.api._v2.keras.p
reprocessing.image' from '/Users/anjanibonda/opt/anaconda3/lib/python3.9/site-
packages/keras/api/_v2/keras/preprocessing/image/__init__.py'>
Downloading data from https://storage.googleapis.com/download.tensorflow.org/d
ata/imagenet_class_index.json
35363/35363 [==============================] - 0s 1us/step
Predicted: [('n02091635', 'otterhound', 0.46458784), ('n02099601', 'golden_ret
riever', 0.24075346), ('n02113799', 'standard_poodle', 0.0913736)]

Displaying hipo.jpg image
```

```
1/1 [==============================] - 0s 198ms/step
Displaying the prediction result for the image: <module 'keras.api._v2.keras.p
reprocessing.image' from '/Users/anjanibonda/opt/anaconda3/lib/python3.9/site-
packages/keras/api/_v2/keras/preprocessing/image/__init__.py'>
Predicted: [('n02422106', 'hartebeest', 0.23387279), ('n02410509', 'bison', 0.
15939173), ('n02132136', 'brown_bear', 0.0656558)]

Displaying deer.jpg image
```

```
1/1 [==============================] - 0s 194ms/step
Displaying the prediction result for the image: <module 'keras.api._v2.keras.p
reprocessing.image' from '/Users/anjanibonda/opt/anaconda3/lib/python3.9/site-
packages/keras/api/_v2/keras/preprocessing/image/__init__.py'>
Predicted: [('n12998815', 'agaric', 0.09936819), ('n02423022', 'gazelle', 0.08
8187516), ('n02115913', 'dhole', 0.07938728)]

Displaying dolphin.jpg image
```

```
1/1 [==============================] - 0s 165ms/step
Displaying the prediction result for the image: <module 'keras.api._v2.keras.p
reprocessing.image' from '/Users/anjanibonda/opt/anaconda3/lib/python3.9/site-
packages/keras/api/_v2/keras/preprocessing/image/__init__.py'>
Predicted: [('n02071294', 'killer_whale', 0.878125), ('n01484850', 'great_whit
e_shark', 0.07984153), ('n01491361', 'tiger_shark', 0.0134579)]

Displaying zebra.jpg image
```

```
1/1 [==============================] - 0s 168ms/step
Displaying the prediction result for the image: <module 'keras.api._v2.keras.p
reprocessing.image' from '/Users/anjanibonda/opt/anaconda3/lib/python3.9/site-
packages/keras/api/_v2/keras/preprocessing/image/__init__.py'>
Predicted: [('n02391049', 'zebra', 0.99756634), ('n02422106', 'hartebeest', 0.
0012307396), ('n02422699', 'impala', 0.0005623602)]

Displaying tiger.jpg image
```

```
1/1 [==============================] - 0s 201ms/step
Displaying the prediction result for the image: <module 'keras.api._v2.keras.p
reprocessing.image' from '/Users/anjanibonda/opt/anaconda3/lib/python3.9/site-
packages/keras/api/_v2/keras/preprocessing/image/__init__.py'>
Predicted: [('n02129604', 'tiger', 0.8707927), ('n02123159', 'tiger_cat', 0.11
096647), ('n02391049', 'zebra', 0.006612491)]

Displaying polar.jpg image
```

```
1/1 [==============================] - 0s 178ms/step
Displaying the prediction result for the image: <module 'keras.api._v2.keras.p
reprocessing.image' from '/Users/anjanibonda/opt/anaconda3/lib/python3.9/site-
packages/keras/api/_v2/keras/preprocessing/image/__init__.py'>
Predicted: [('n02510455', 'giant_panda', 0.9994486), ('n02447366', 'badger',
0.00021097696), ('n02134084', 'ice_bear', 0.00015674095)]
```

In [ ]: