Assignment 06

Name: Kesav Adithya Venkidusamy
Course: DSC650 - Big Data
Instructor: Amirfarrokh Iranitalab

Assignment 6.1

Using section 5.1 in Deep Learning with Python as a guide (listing 5.3 in particular), create a ConvNet model that classifies images in the MNIST digit dataset. Save the model, predictions, metrics, and validation plots in the dsc650/assignments/assignment06/results directory. If you are using JupyterHub, you can include those plots in your Jupyter notebook.

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 [15]:

```
# 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 [16]:
```
## Instantiating 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 [17]:
```
## Adding classifer on top of convnet
model.add(layers.Flatten())
model.add(layers.Dense(10, activation='softmax'))
```

In [18]:
```
## Showing model summary
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 3, 3, 128)         73856
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                11530
=================================================================
Total params: 104,202
Trainable params: 104,202
Non-trainable params: 0
_________________________________________________________________
```

In [19]:
```
# Compile the model
model.compile(optimizer="rmsprop",
    loss="categorical_crossentropy",
    metrics=["accuracy"])
```

Model Validation

In [20]:
```
## 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

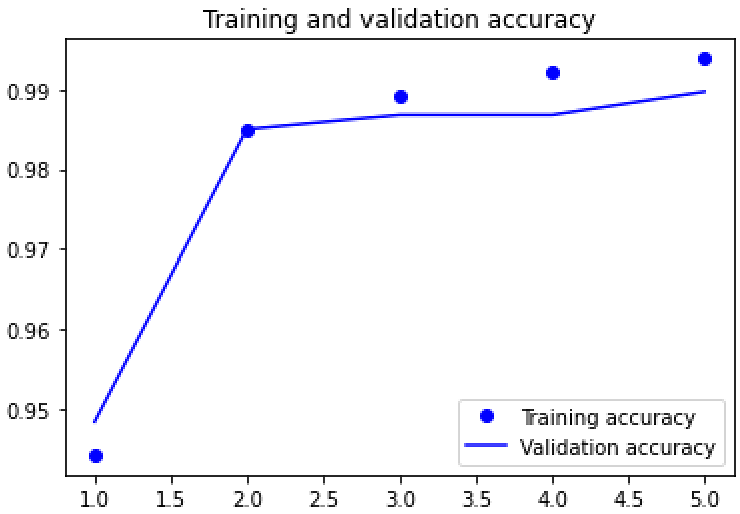
In [21]:
```
# Train the model
history = model.fit(partial_train_images,
                    partial_train_labels,
                    epochs=5,
                    batch_size=64,
                    validation_data=(validation_images, validation_labels))
```

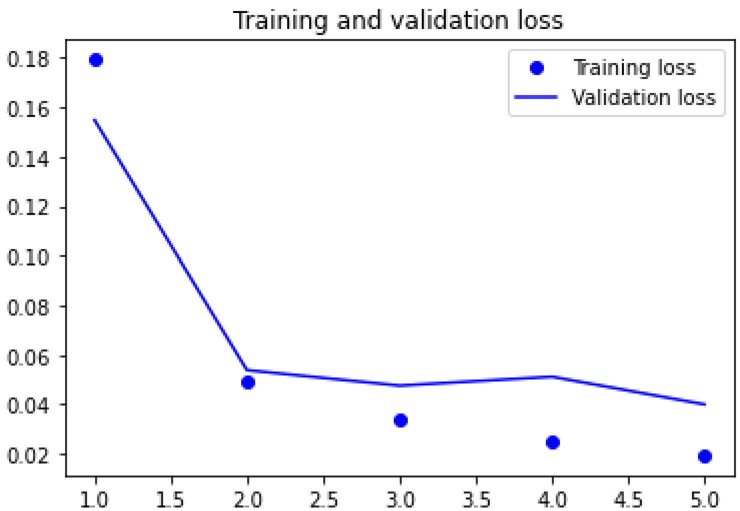
```
Epoch 1/5
782/782 [==============================] - 11s 15ms/step - loss: 0.1793 - accuracy: 0.9442 - val_loss: 0.1548 - val_accur
acy: 0.9484
Epoch 2/5
782/782 [==============================] - 11s 14ms/step - loss: 0.0491 - accuracy: 0.9849 - val_loss: 0.0538 - val_accur
acy: 0.9850
Epoch 3/5
782/782 [==============================] - 11s 14ms/step - loss: 0.0339 - accuracy: 0.9893 - val_loss: 0.0475 - val_accur
acy: 0.9868
Epoch 4/5
782/782 [==============================] - 11s 14ms/step - loss: 0.0252 - accuracy: 0.9921 - val_loss: 0.0511 - val_accur
acy: 0.9868
Epoch 5/5
782/782 [==============================] - 11s 14ms/step - loss: 0.0193 - accuracy: 0.9939 - val_loss: 0.0400 - val_accur
acy: 0.9897
```

Plotting Model Output and Loss

In [22]:
```
# 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 [23]:
```
# 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 [==============================] - 1s 4ms/step - loss: 0.0289 - accuracy: 0.9911
Test accuracy: 99.1%
Test loss: 0.029
```

The model accuracy is 99.1% and loss is only 0.029; The accuracy is increased significantly and loss is reduced a lot by adding Conv2D and MaxPooling2D layers

Save Model

In [24]:
```
model.save('results/mnist')
```

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

Assignment 6.2

Assignment 6.2.a

Using section 5.2 in Deep Learning with Python as a guide, create a ConvNet model that classifies images CIFAR10 small images classification dataset. Do not use dropout or data-augmentation in this part. Save the model, predictions, metrics, and validation plots in the dsc650/assignments/assignment06/results directory. If you are using JupyterHub, you can include those plots in your Jupyter notebook.

Load the data & Data preparation

In [25]:
```
# 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)
```

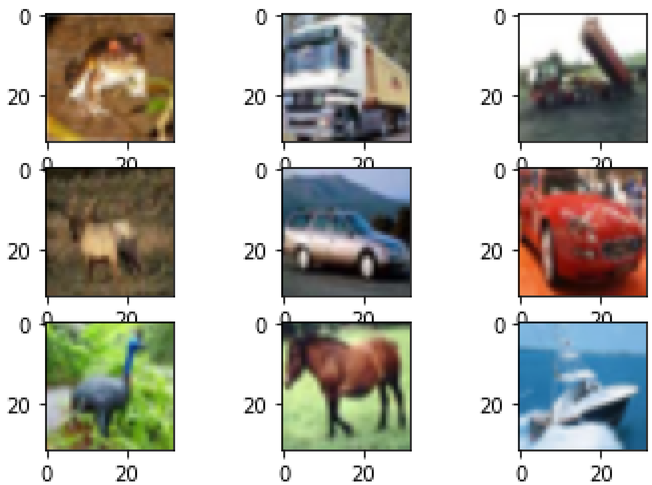
In [26]:
```
# summarize loaded dataset
print('Train: X=%s, y=%s' % (x_train.shape, y_train.shape))
```

```
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 [27]:

```
# 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 [28]:

```
## 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 [29]:

```
## Instantiating 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 [30]:

```
## Adding classifer on top of convnet
model.add(layers.Flatten())
model.add(layers.Dense(10, activation='softmax'))
```

In [31]:

```
## Showing model summary
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
rescaling (Rescaling)        (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 4, 4, 128)         73856
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 98,378
Trainable params: 98,378
Non-trainable params: 0
_________________________________________________________________
```

In [32]:

```
# Compile model
model.compile(optimizer="rmsprop",
```

```
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"])
```

Model Training

In [33]:

```
# 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 [==============================] - 13s 20ms/step - loss: 1.6908 - accuracy: 0.3901 - val_loss: 1.4259 - val_accur
acy: 0.4851
Epoch 2/30
625/625 [==============================] - 12s 20ms/step - loss: 1.3411 - accuracy: 0.5253 - val_loss: 1.1883 - val_accur
acy: 0.5744
Epoch 3/30
625/625 [==============================] - 12s 20ms/step - loss: 1.1757 - accuracy: 0.5865 - val_loss: 1.0877 - val_accur
acy: 0.6131
Epoch 4/30
625/625 [==============================] - 12s 20ms/step - loss: 1.0576 - accuracy: 0.6312 - val_loss: 1.0978 - val_accur
acy: 0.6277
Epoch 5/30
625/625 [==============================] - 13s 20ms/step - loss: 0.9626 - accuracy: 0.6679 - val_loss: 1.0099 - val_accur
acy: 0.6548
Epoch 6/30
625/625 [==============================] - 12s 20ms/step - loss: 0.8841 - accuracy: 0.6920 - val_loss: 0.9303 - val_accur
acy: 0.6802
Epoch 7/30
625/625 [==============================] - 12s 20ms/step - loss: 0.8182 - accuracy: 0.7183 - val_loss: 0.9346 - val_accur
acy: 0.6833
Epoch 8/30
625/625 [==============================] - 12s 20ms/step - loss: 0.7581 - accuracy: 0.7387 - val_loss: 0.9623 - val_accur
acy: 0.6784
Epoch 9/30
625/625 [==============================] - 12s 20ms/step - loss: 0.7045 - accuracy: 0.7583 - val_loss: 0.9184 - val_accur
acy: 0.6901
Epoch 10/30
625/625 [==============================] - 12s 20ms/step - loss: 0.6561 - accuracy: 0.7736 - val_loss: 0.8863 - val_accur
acy: 0.7017
Epoch 11/30
625/625 [==============================] - 12s 20ms/step - loss: 0.6121 - accuracy: 0.7880 - val_loss: 0.8783 - val_accur
acy: 0.7119
Epoch 12/30
```

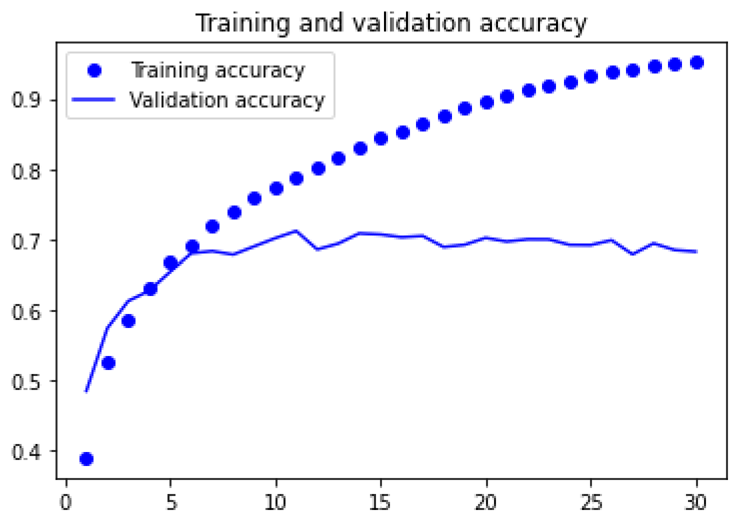
```
625/625 [==============================] - 12s 20ms/step - loss: 0.5683 - accuracy: 0.8031 - val_loss: 1.0025 - val_accur
acy: 0.6857
Epoch 13/30
625/625 [==============================] - 12s 20ms/step - loss: 0.5273 - accuracy: 0.8168 - val_loss: 0.9672 - val_accur
acy: 0.6940
Epoch 14/30
625/625 [==============================] - 12s 20ms/step - loss: 0.4920 - accuracy: 0.8298 - val_loss: 0.9277 - val_accur
acy: 0.7085
Epoch 15/30
625/625 [==============================] - 12s 20ms/step - loss: 0.4547 - accuracy: 0.8434 - val_loss: 0.9808 - val_accur
acy: 0.7070
Epoch 16/30
625/625 [==============================] - 12s 20ms/step - loss: 0.4191 - accuracy: 0.8527 - val_loss: 1.0156 - val_accur
acy: 0.7030
Epoch 17/30
625/625 [==============================] - 12s 20ms/step - loss: 0.3856 - accuracy: 0.8652 - val_loss: 1.0040 - val_accur
acy: 0.7049
Epoch 18/30
625/625 [==============================] - 12s 20ms/step - loss: 0.3560 - accuracy: 0.8767 - val_loss: 1.1285 - val_accur
acy: 0.6891
Epoch 19/30
625/625 [==============================] - 12s 20ms/step - loss: 0.3272 - accuracy: 0.8863 - val_loss: 1.1621 - val_accur
acy: 0.6923
Epoch 20/30
625/625 [==============================] - 12s 20ms/step - loss: 0.3010 - accuracy: 0.8951 - val_loss: 1.1480 - val_accur
acy: 0.7021
Epoch 21/30
625/625 [==============================] - 12s 20ms/step - loss: 0.2758 - accuracy: 0.9035 - val_loss: 1.2192 - val_accur
acy: 0.6969
Epoch 22/30
625/625 [==============================] - 12s 20ms/step - loss: 0.2535 - accuracy: 0.9124 - val_loss: 1.2460 - val_accur
acy: 0.7001
Epoch 23/30
625/625 [==============================] - 12s 20ms/step - loss: 0.2339 - accuracy: 0.9182 - val_loss: 1.3402 - val_accur
acy: 0.7000
Epoch 24/30
625/625 [==============================] - 12s 20ms/step - loss: 0.2127 - accuracy: 0.9255 - val_loss: 1.3950 - val_accur
acy: 0.6922
Epoch 25/30
625/625 [==============================] - 12s 20ms/step - loss: 0.1943 - accuracy: 0.9317 - val_loss: 1.4656 - val_accur
acy: 0.6919
Epoch 26/30
625/625 [==============================] - 12s 20ms/step - loss: 0.1786 - accuracy: 0.9378 - val_loss: 1.4800 - val_accur
acy: 0.6988
Epoch 27/30
625/625 [==============================] - 12s 20ms/step - loss: 0.1652 - accuracy: 0.9421 - val_loss: 1.6897 - val_accur
acy: 0.6787
Epoch 28/30
625/625 [==============================] - 12s 20ms/step - loss: 0.1538 - accuracy: 0.9460 - val_loss: 1.6737 - val_accur
```

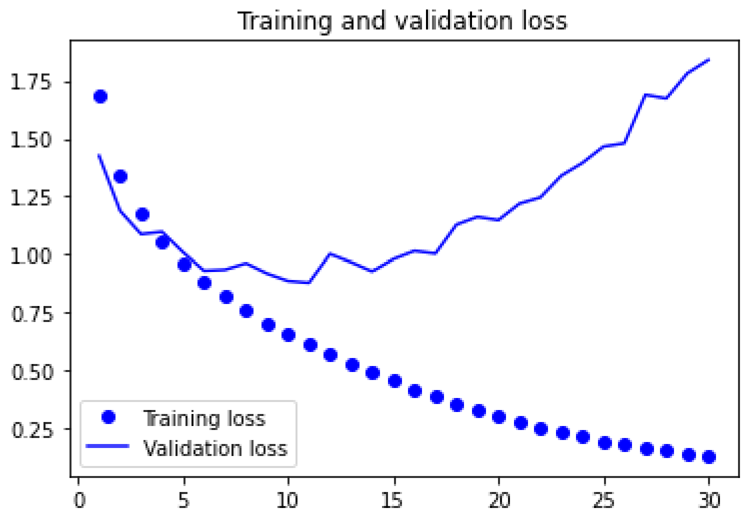
```
acy: 0.6943
Epoch 29/30
625/625 [==============================] - 12s 20ms/step - loss: 0.1429 - accuracy: 0.9496 - val_loss: 1.7823 - val_accur
acy: 0.6849
Epoch 30/30
625/625 [==============================] - 12s 20ms/step - loss: 0.1312 - accuracy: 0.9521 - val_loss: 1.8399 - val_accur
acy: 0.6826
```

Plotting the result

In [34]:

```
# 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 [35]:
```
# 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 [==============================] - 2s 5ms/step - loss: 1.8304 - accuracy: 0.6820
Test accuracy: 68.2%
Test loss: 1.830
```

The accuracy score without dropout and data augmentation turned out as 68.2% and loss is 1.83

Save Model

In [36]:
```
model.save('results/without_dropout_augmentation')
```

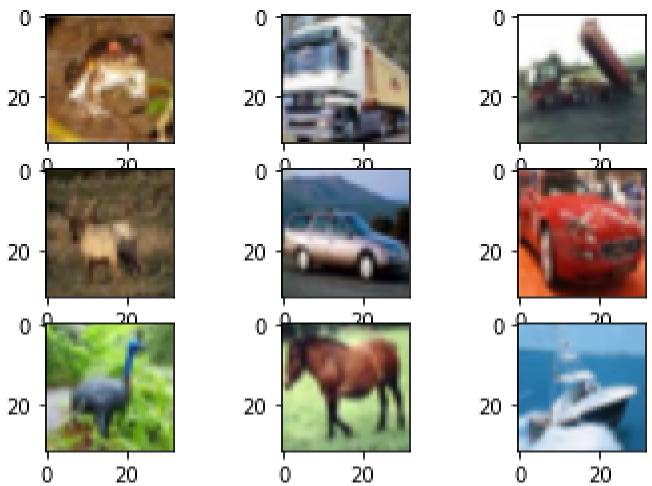
```
INFO:tensorflow:Assets written to: results/without_dropout_augmentation/assets
```

Assignment 6.2.b

Using section 5.2 in Deep Learning with Python as a guide, create a ConvNet model that classifies images CIFAR10 small images classification dataset. This time includes dropout and data-augmentation. Save the model, predictions, metrics, and validation plots in the dsc650/assignments/assignment06/results directory. If you are using JupyterHub, you can include those plots in your Jupyter notebook.

Model Building with dropout or data-augmentation

In [2]:
```
#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 [3]:
```
# 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 [4]:
```
# 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 [5]:
```
#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 [6]:
```
#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 [7]:
```
# 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 [9]:

```
# Building 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, input_shape=(img_rows, img_cols, channels
model.add(layers.BatchNormalization(axis=-1))
model.add(layers.Conv2D(32, (3, 3), activation='relu',kernel_regularizer=None,padding='same'))
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,padding='same'))
model.add(layers.BatchNormalization(axis=-1))
model.add(layers.Conv2D(64, (3, 3), activation='relu',kernel_regularizer=None,padding='same'))
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,padding='same'))
model.add(layers.BatchNormalization(axis=-1))
model.add(layers.Conv2D(128, (3, 3), activation='relu',kernel_regularizer=None,padding='same'))
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=opt2)
```

In [10]:

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

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_3 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_5 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_6 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
batch_normalization_7 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1049088
_________________________________________________________________
batch_normalization_8 (Batch (None, 512)               2048
```

```
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 1,345,066
Trainable params: 1,343,146
Non-trainable params: 1,920
_________________________________________________________________
```

Train the model

In [11]:

```
# 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, validation_data=(x_test, y_test))
```

```
Epoch 1/30
391/390 [==============================] - 64s 165ms/step - loss: 1.9764 - accuracy: 0.3486 - val_loss: 2.3218 - val_accu
racy: 0.2912
Epoch 2/30
391/390 [==============================] - 62s 160ms/step - loss: 1.4459 - accuracy: 0.4767 - val_loss: 1.4339 - val_accu
racy: 0.4932
Epoch 3/30
391/390 [==============================] - 62s 159ms/step - loss: 1.2586 - accuracy: 0.5454 - val_loss: 1.5103 - val_accu
racy: 0.5330
Epoch 4/30
391/390 [==============================] - 61s 157ms/step - loss: 1.1504 - accuracy: 0.5864 - val_loss: 1.2886 - val_accu
racy: 0.5760
Epoch 5/30
391/390 [==============================] - 62s 158ms/step - loss: 1.0675 - accuracy: 0.6179 - val_loss: 1.2597 - val_accu
racy: 0.5807
Epoch 6/30
391/390 [==============================] - 62s 158ms/step - loss: 1.0110 - accuracy: 0.6384 - val_loss: 1.1089 - val_accu
racy: 0.6319
Epoch 7/30
391/390 [==============================] - 62s 157ms/step - loss: 0.9549 - accuracy: 0.6612 - val_loss: 0.9950 - val_accu
racy: 0.6728
Epoch 8/30
391/390 [==============================] - 61s 156ms/step - loss: 0.9154 - accuracy: 0.6763 - val_loss: 0.9506 - val_accu
racy: 0.6776
Epoch 9/30
391/390 [==============================] - 61s 156ms/step - loss: 0.8755 - accuracy: 0.6913 - val_loss: 0.9720 - val_accu
racy: 0.6753
Epoch 10/30
391/390 [==============================] - 62s 157ms/step - loss: 0.8435 - accuracy: 0.7037 - val_loss: 0.8679 - val_accu
racy: 0.7084
Epoch 11/30
```

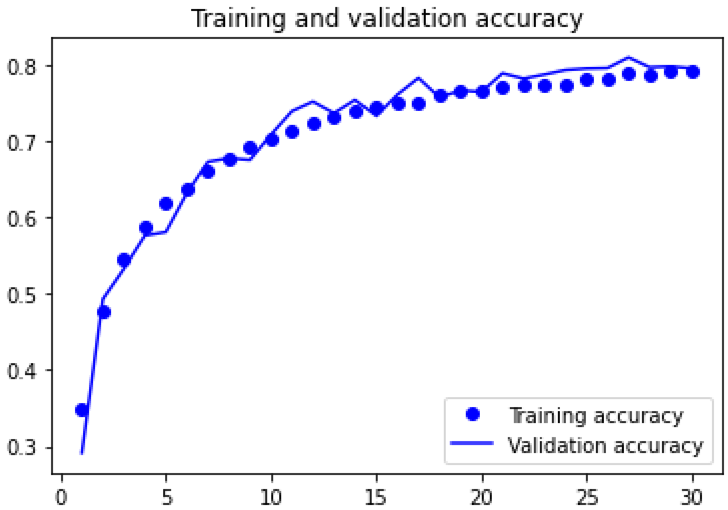
```
391/390 [==============================] - 62s 158ms/step - loss: 0.8139 - accuracy: 0.7146 - val_loss: 0.7724 - val_accu
racy: 0.7394
Epoch 12/30
391/390 [==============================] - 62s 158ms/step - loss: 0.7911 - accuracy: 0.7250 - val_loss: 0.7326 - val_accu
racy: 0.7519
Epoch 13/30
391/390 [==============================] - 62s 158ms/step - loss: 0.7665 - accuracy: 0.7310 - val_loss: 0.8034 - val_accu
racy: 0.7368
Epoch 14/30
391/390 [==============================] - 62s 159ms/step - loss: 0.7459 - accuracy: 0.7398 - val_loss: 0.7303 - val_accu
racy: 0.7541
Epoch 15/30
391/390 [==============================] - 62s 158ms/step - loss: 0.7355 - accuracy: 0.7446 - val_loss: 0.8183 - val_accu
racy: 0.7331
Epoch 16/30
391/390 [==============================] - 61s 157ms/step - loss: 0.7223 - accuracy: 0.7500 - val_loss: 0.7204 - val_accu
racy: 0.7611
Epoch 17/30
391/390 [==============================] - 62s 159ms/step - loss: 0.7099 - accuracy: 0.7514 - val_loss: 0.6449 - val_accu
racy: 0.7830
Epoch 18/30
391/390 [==============================] - 62s 158ms/step - loss: 0.6906 - accuracy: 0.7607 - val_loss: 0.7473 - val_accu
racy: 0.7574
Epoch 19/30
391/390 [==============================] - 61s 156ms/step - loss: 0.6763 - accuracy: 0.7658 - val_loss: 0.7326 - val_accu
racy: 0.7666
Epoch 20/30
391/390 [==============================] - 61s 157ms/step - loss: 0.6743 - accuracy: 0.7653 - val_loss: 0.7045 - val_accu
racy: 0.7646
Epoch 21/30
391/390 [==============================] - 62s 158ms/step - loss: 0.6594 - accuracy: 0.7718 - val_loss: 0.6221 - val_accu
racy: 0.7891
Epoch 22/30
391/390 [==============================] - 62s 158ms/step - loss: 0.6507 - accuracy: 0.7728 - val_loss: 0.6501 - val_accu
racy: 0.7822
Epoch 23/30
391/390 [==============================] - 62s 158ms/step - loss: 0.6462 - accuracy: 0.7744 - val_loss: 0.6350 - val_accu
racy: 0.7875
Epoch 24/30
391/390 [==============================] - 62s 158ms/step - loss: 0.6419 - accuracy: 0.7748 - val_loss: 0.6372 - val_accu
racy: 0.7934
Epoch 25/30
391/390 [==============================] - 61s 156ms/step - loss: 0.6317 - accuracy: 0.7816 - val_loss: 0.6073 - val_accu
racy: 0.7954
Epoch 26/30
391/390 [==============================] - 61s 156ms/step - loss: 0.6246 - accuracy: 0.7816 - val_loss: 0.6120 - val_accu
racy: 0.7960
Epoch 27/30
391/390 [==============================] - 61s 157ms/step - loss: 0.6081 - accuracy: 0.7886 - val_loss: 0.5805 - val_accu
```

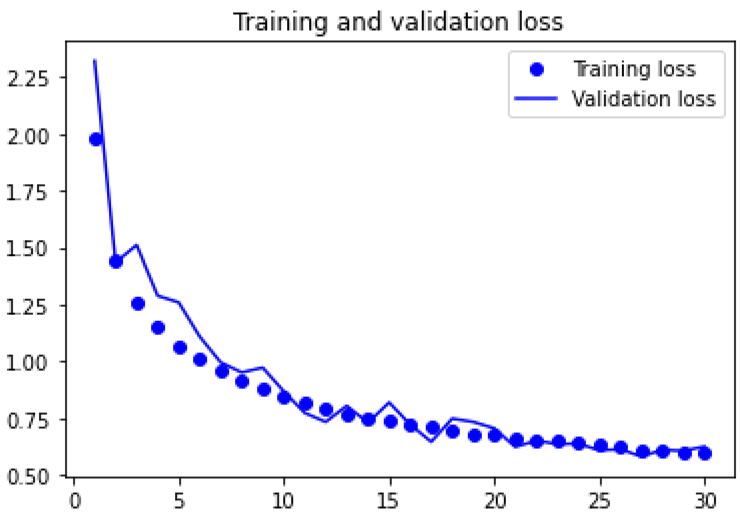
```
racy: 0.8099
Epoch 28/30
391/390 [==============================] - 61s 156ms/step - loss: 0.6075 - accuracy: 0.7881 - val_loss: 0.6088 - val_accu
racy: 0.7971
Epoch 29/30
391/390 [==============================] - 61s 156ms/step - loss: 0.5978 - accuracy: 0.7928 - val_loss: 0.6052 - val_accu
racy: 0.7981
Epoch 30/30
391/390 [==============================] - 62s 158ms/step - loss: 0.5985 - accuracy: 0.7910 - val_loss: 0.6238 - val_accu
racy: 0.7957
```

Plot the model

In [12]:

```
# 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 [13]:

```python
# 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 13ms/step - loss: 0.6238 - accuracy: 0.7957
Test accuracy: 79.6%
Test loss: 0.624
```

The accuracy scre with dropout and data augmentation has been increased to 79.6%

Save the model

In [14]:
```
model.save('results/with_dropout_augmentation')
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.8/site-packages/tensorflow/python/ops/resource_variable_ops.py:1813: calli
ng BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and wil
l be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: results/with_dropout_augmentation/assets
```

Assignment 6.3

Load the ResNet50 model. Perform image classification on five to ten images of your choice. They can be personal images or publically available images. Include the images in dsc650/assignments/assignment06/images/. Save the predictions dsc650/assignments/assignment06/results/predictions/resnet50 directory. If you are using JupyterHub, you can include those plots in your Jupyter notebook.

Loading libraries

In [7]:
```
# Load libraries
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np
from IPython.display import Image, display
import os
```

Define Model

In [2]:
```
# Load model
model = ResNet50(weights='imagenet')
```

Image Classification

In [13]:
```
## 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 [16]:

```
## Reading the image present in images directory
## calling 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 deer.jpg image
```

```
Displaying the prediction result for the image: <module 'tensorflow.keras.preprocessing.image' from '/opt/conda/lib/pytho
n3.8/site-packages/tensorflow/keras/preprocessing/image/__init__.py'>
Predicted: [('n12998815', 'agaric', 0.09936827), ('n02423022', 'gazelle', 0.08818725), ('n02115913', 'dhole', 0.07938722
5)]

Displaying polar.jpg image
```

```
Displaying the prediction result for the image: <module 'tensorflow.keras.preprocessing.image' from '/opt/conda/lib/pytho
n3.8/site-packages/tensorflow/keras/preprocessing/image/__init__.py'>
Predicted: [('n02510455', 'giant_panda', 0.99944896), ('n02447366', 'badger', 0.00021097742), ('n02134084', 'ice_bear',
0.0001567416)]

Displaying hipo.jpg image
```

```
Displaying the prediction result for the image: <module 'tensorflow.keras.preprocessing.image' from '/opt/conda/lib/pytho
n3.8/site-packages/tensorflow/keras/preprocessing/image/__init__.py'>
Predicted: [('n02422106', 'hartebeest', 0.23387302), ('n02410509', 'bison', 0.15939221), ('n02132136', 'brown_bear', 0.06
565593)]

Displaying dolphin.jpg image
```

```
Displaying the prediction result for the image: <module 'tensorflow.keras.preprocessing.image' from '/opt/conda/lib/pytho
n3.8/site-packages/tensorflow/keras/preprocessing/image/__init__.py'>
```

```
Predicted: [('n02071294', 'killer_whale', 0.878125), ('n01484850', 'great_white_shark', 0.07984153), ('n01491361', 'tiger
_shark', 0.013457938)]

Displaying dog.jpg image
```

```
Displaying the prediction result for the image: <module 'tensorflow.keras.preprocessing.image' from '/opt/conda/lib/pytho
n3.8/site-packages/tensorflow/keras/preprocessing/image/__init__.py'>
Predicted: [('n02091635', 'otterhound', 0.464588), ('n02099601', 'golden_retriever', 0.24075353), ('n02113799', 'standard
_poodle', 0.09137372)]

Displaying zebra.jpg image
```

```
Displaying the prediction result for the image: <module 'tensorflow.keras.preprocessing.image' from '/opt/conda/lib/pytho
n3.8/site-packages/tensorflow/keras/preprocessing/image/__init__.py'>
Predicted: [('n02391049', 'zebra', 0.9975666), ('n02422106', 'hartebeest', 0.001230741), ('n02422699', 'impala', 0.000562
362)]

Displaying tiger.jpg image
```

```
Displaying the prediction result for the image: <module 'tensorflow.keras.preprocessing.image' from '/opt/conda/lib/pytho
n3.8/site-packages/tensorflow/keras/preprocessing/image/__init__.py'>
Predicted: [('n02129604', 'tiger', 0.87079287), ('n02123159', 'tiger_cat', 0.110966384), ('n02391049', 'zebra', 0.0066124
92)]
```

In [ ]:

```

```