Network Analysis

Group Assignment 1

Object Images

@ChenYu Wang - c.wang4@students.uu.nl, 6774326

@Lena Walcher - l.t.walcher@students.uu.nl, 2818833

@Ioannis Konstantakopoulos - i.konstantakopoulos@students.uu.nl, 0960047

@Hans Franke - h.a.franke@students.uu.nl, 6987680

In [9]:
```
import tensorflow as tf
import keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator #preprocessing images
```

In [10]:
```
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 15746176172422564165
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 6959755424
locality {
  bus_id: 1
  links {
  }
}
incarnation: 17890906695776707346
physical_device_desc: "device: 0, name: GeForce RTX 2070 with Max-Q Design, pci bus id: 0000:01:00.0, compute capabil
ity: 7.5"
]
```

Identifying objects from images with a high variation in object position, size and viewing angle is a particularly difficult problem, and a major application of artificial deep learning networks. Here we will build a network to do this.

This network has more layers and is more computationally-intensive than previous exercises, even using the low-resolution images we will use here. The CIFAR-10 dataset contains colour images of objects, each 32x32x3 pixels (for the three colour channels). These have 10 categories (or classes) of object (airplane, automobile, bird, cat, deer, dog, frog, horse, ship & truck) with 5,000 images in each, making a total of 50,000 images in the training set (x_train, y_train), randomly ordered with numerical labels for each (1=airplane, 2=automobile etc.). The test set (x_test, y_test) contains 10,000 images ordered by their label.

In [16]:
```
#Download the dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
```

In [8]:
```
#Save to disk (avoid the need of download one more time.. ~23 min :( ) Uncoment to save again, in case of step above u
sed!
#np.save('x_train', x_train)
#np.save('y_train', y_train)
#np.save('x_test', x_test)
#np.save('y_test', y_test)
```

In [11]:
```
#load Files
x_train = np.load('x_train.npy')
x_test = np.load('x_test.npy')
y_train = np.load('y_train.npy')
y_test = np.load('y_test.npy')
```

In [12]:
```
#scale the data
x_train = x_train / 255
x_test = x_test / 255
```

In [13]:
```
#Categorical Variables
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

Question 16:

Before fitting the model, show your teacher the code you used to define the model described here. (Question 16, 6 points)

In [14]:
```
#Defining model with Dropout Layers
model = keras.Sequential()
#Conv 1
model.add(keras.layers.Conv2D(filters=32, kernel_size=(3, 3),
                              activation="relu", input_shape=(32, 32, 3), padding='same'))
#Conv2
model.add(keras.layers.Conv2D(filters=32, kernel_size=(3, 3),
                              activation="relu"))
#maxPool
model.add(keras.layers.MaxPool2D(pool_size=(2, 2)))
model.add(keras.layers.Dropout(rate=0.25))
#----------------------#
#ADD NEW LAYERS
#Conv 3
model.add(keras.layers.Conv2D(filters=32, kernel_size=(3, 3),
                              activation="relu", padding='same'))
#Conv 4
model.add(keras.layers.Conv2D(filters=32, kernel_size=(3, 3),
                              activation="relu"))
#maxPool 2
model.add(keras.layers.MaxPool2D(pool_size=(2, 2)))
model.add(keras.layers.Dropout(rate=0.25))

#----------------------#
#Flattening
model.add(keras.layers.Flatten())

#fullDense
model.add(keras.layers.Dense(512, activation="relu"))
model.add(keras.layers.Dropout(rate=0.5))

#outputlayer
model.add(keras.layers.Dense(10, activation="softmax"))

#Compile the Model:
model.compile(loss='categorical_crossentropy',
       optimizer=keras.optimizers.RMSprop(lr=0.0001, decay=1e-6), metrics='accuracy')

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 15, 15, 32)        0
_________________________________________________________________
```

```
conv2d_2 (Conv2D)            (None, 15, 15, 32)        9248
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 32)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 6, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 512)               590336
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 624,106
Trainable params: 624,106
Non-trainable params: 0
_________________________________________________________________
```

Parameters (example to first layer)

Kernel (3x3 = 9)

Filters (32)

Bias (32 because if 1 for each filter)

9 * 32 + 32 = 896

In [15]:
```
# Fit the model
history = model.fit(x_train, y_train, batch_size=32,
                    epochs=20, verbose=1, validation_data=(x_test, y_test), shuffle=True)
```

```
Epoch 1/20
1563/1563 [==============================] - 25s 12ms/step - loss: 2.0678 - accuracy: 0.2270 - val_loss: 1.7991 - val
_accuracy: 0.3608
Epoch 2/20
1563/1563 [==============================] - 19s 12ms/step - loss: 1.6487 - accuracy: 0.4007 - val_loss: 1.4588 - val
_accuracy: 0.4749
Epoch 3/20
1563/1563 [==============================] - 97s 62ms/step - loss: 1.4894 - accuracy: 0.4581 - val_loss: 1.4300 - val
_accuracy: 0.4896
Epoch 4/20
1563/1563 [==============================] - 19s 12ms/step - loss: 1.3840 - accuracy: 0.5015 - val_loss: 1.2977 - val
_accuracy: 0.5402
Epoch 5/20
1563/1563 [==============================] - 21s 13ms/step - loss: 1.2961 - accuracy: 0.5330 - val_loss: 1.1801 - val
_accuracy: 0.5792
Epoch 6/20
1563/1563 [==============================] - 21s 13ms/step - loss: 1.2321 - accuracy: 0.5636 - val_loss: 1.1328 - val
_accuracy: 0.5967
Epoch 7/20
1563/1563 [==============================] - 21s 14ms/step - loss: 1.1909 - accuracy: 0.5766 - val_loss: 1.1142 - val
_accuracy: 0.6081
Epoch 8/20
1563/1563 [==============================] - 24s 15ms/step - loss: 1.1518 - accuracy: 0.5927 - val_loss: 1.0470 - val
_accuracy: 0.6312
Epoch 9/20
1563/1563 [==============================] - 23s 15ms/step - loss: 1.1102 - accuracy: 0.6072 - val_loss: 1.0138 - val
_accuracy: 0.6429
Epoch 10/20
1563/1563 [==============================] - 24s 15ms/step - loss: 1.0724 - accuracy: 0.6211 - val_loss: 1.0133 - val
_accuracy: 0.6420
Epoch 11/20
1563/1563 [==============================] - 24s 15ms/step - loss: 1.0390 - accuracy: 0.6329 - val_loss: 0.9770 - val
_accuracy: 0.6561
Epoch 12/20
1563/1563 [==============================] - 25s 16ms/step - loss: 1.0257 - accuracy: 0.6353 - val_loss: 0.9679 - val
_accuracy: 0.6577
Epoch 13/20
1563/1563 [==============================] - 25s 16ms/step - loss: 0.9990 - accuracy: 0.6455 - val_loss: 0.9312 - val
_accuracy: 0.6739
Epoch 14/20
1563/1563 [==============================] - 26s 17ms/step - loss: 0.9714 - accuracy: 0.6564 - val_loss: 0.9080 - val
_accuracy: 0.6823
Epoch 15/20
1563/1563 [==============================] - 28s 18ms/step - loss: 0.9587 - accuracy: 0.6641 - val_loss: 0.9077 - val
_accuracy: 0.6815
Epoch 16/20
1563/1563 [==============================] - 30s 19ms/step - loss: 0.9469 - accuracy: 0.6646 - val_loss: 0.8803 - val
_accuracy: 0.6920
Epoch 17/20
1563/1563 [==============================] - 29s 19ms/step - loss: 0.9336 - accuracy: 0.6702 - val_loss: 0.8753 - val
_accuracy: 0.6965
Epoch 18/20
1563/1563 [==============================] - 29s 19ms/step - loss: 0.9114 - accuracy: 0.6821 - val_loss: 0.8656 - val
_accuracy: 0.7007
Epoch 19/20
1563/1563 [==============================] - 32s 21ms/step - loss: 0.8991 - accuracy: 0.6865 - val_loss: 0.8522 - val
_accuracy: 0.7031
Epoch 20/20
1563/1563 [==============================] - 33s 21ms/step - loss: 0.8905 - accuracy: 0.6842 - val_loss: 0.8441 - val
_accuracy: 0.7048
```

In [14]:
```
#Save model to disk
model.save('model_objects')
```

```
INFO:tensorflow:Assets written to: model_objects\assets
```

In [20]:
```
#load model = > uncoment to run! :)
#model = keras.models.load_model('model_objects')
#model
```

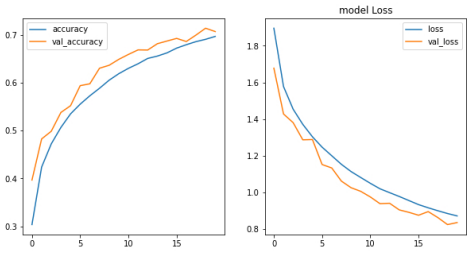
Out[20]:
```
<tensorflow.python.keras.engine.sequential.Sequential at 0x2562d2fdeb0>
```

Question 17:

Plot the training history and show it to your teacher (Question 17, 2 points)

In [15]:
```
#Accuracy
fig, [ax,ax1] = plt.subplots(figsize=(10,5),nrows=1,ncols=2)
plt.title('model accuracy')
ax.plot(history.history['accuracy'], label = 'accuracy')
ax.plot(history.history['val_accuracy'], label = 'val_accuracy')
ax.legend()
plt.title('model Loss')
ax1.plot(history.history['loss'], label = 'loss')
ax1.plot(history.history['val_loss'], label = 'val_loss')
ax1.legend()
```

Out[15]:
```
<matplotlib.legend.Legend at 0x209e7923070>
```



Question 18:

Discuss with your group, then describe to your teacher, how the training history differs from the convolutional model for digit recognition and why. (Question 18, 5 points)

The model is more complex, with 2 layers of convolutions (4 if we consider 2 conv x 2 sub conv). It by itself turn the model slower than before. Contribution to this we have now 3D array of image, so basicaly we come from 784 to 3072 (32x32x3) input size, it turns the model slower.

The model is a lot more complex with added layers and dropouts, with 2 layers of convolutions (4 if we consider 2 conv x 2 sub conv). It by itself turn the model slower than before. Contribution to this we have now 3D array of image, so basicaly we come from 784 to 3072 (32x32x3) input size, it turns the model slower. Epochs before was 6, now we have 20 epochs.

As one can see in the plot above, the accuracy of the model doesn't jump as high after the first epoch but continues to improve due to the smaller learning rate. The test accuracy is higher than the training accuracy but both are still rather low.

The pictures used in the model are natural images which makes them a lot more complex with more dimension within the picture. Classifying natural images makes the model more complex whereas in the convolution model we only used pictures of numbers that were less difficult to classify.

The accuracy reduces a lot (~70% from 98%), the complexity of images explain this behaviour.

Question 19:

Discuss with your group, then describe to your teacher, how the time taken for each training epoch differs from the convolutional model for digit recognition. Give several factors that may contribute to this difference (Question 19, 4 points)

It took a lot longer for each epoch to run as the model is much more complex and includes more layers. This might be due to:

- number of layers in the model (from 1 to 2)
  - Why? with 2 layers the model have more weights and neurons so more process timing
- optimizer
- input size (from 784,1) to (32,32,3)
  - why? images in 3D has more dimensions, so the size of filters for definition turns the model more complex and required more computations (even it is arrays..)
- batch size (from 128 to 32)
- epochs (total time differs not time by epoch)

Model_digit_recognition: *Adadelta*: Instead of accumulating all previously squared gradients, Adadelta limits the window of accumulated past gradients to some fixed size w. In this exponentially moving average is used rather than the sum of all the gradients.

Model_object_recognition: *RMSprop*: The centered version additionally maintains a moving average of the gradients, and uses that average to estimate the variance.

In [ ]: