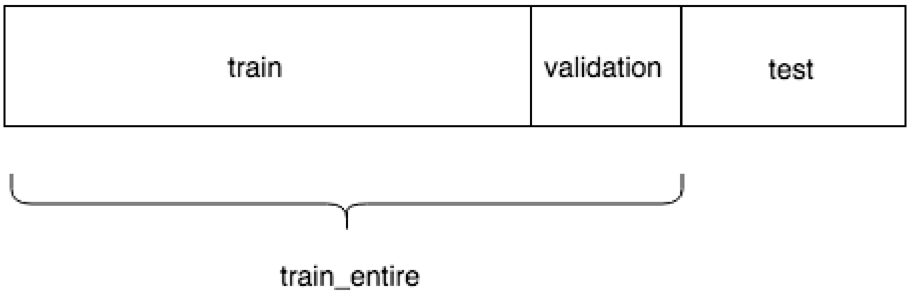
Exercise P2 - TDT 4265 Computer Vision and Deep Learning

Run the cell below to load the MNIST data set, which you will use in **Task 1 and Task 2**.

As you recall from the class, we generally split the training set into train + vaildation as shows in the following image (This will be done at the end of this section).



```
#@title Default title text
print('Ensure that we have Keras installed')
!pip install -q keras
from keras.datasets import mnist

# The data, split between train and test sets
# x_train is a list of training images, y_train is a list og training lables
# x_test is a list of test images, y_test is a list of test lables
(x_train, y_train), (x_test, y_test) = mnist.load_data()

print('x_train size: {}, y_train size: {}\n'.format(len(x_train), len(y_train)))

# Show the format of one randomly chosen image and label (123)
image = x_train[123]
label = y_train[123]

print('Image shape: {}'.format(image.shape))
print('Label: {}'.format(label))

print('x_train size: {}, y_train size: {}\n'.format(len(x_train), len(y_train)))
```

```
Ensure that we have Keras installed
x_train size: 60000, y_train size: 60000

Image shape: (28, 28)
Label: 7
x_train size: 60000, y_train size: 60000
```

Task 0: Create a helper function to plot and visulize the training

(might be smart to solve after Task 1)

Input: history (keras history from the training)

GOAL:

1. Create a graph that plots **Loss vs epochs** and one that plots **Accuracy vs epochs** of the **training and validation set**

```
%matplotlib inline
from matplotlib import pyplot as plt

def plot_training_score(history):
  print('Availible variables to plot: {}'.format(history.history.keys()))
  plt.plot(history.history['loss'])
  plt.plot(history.history['val_loss'])
  plt.title('Loss')
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['train', 'validate'], loc = 'upper left')
  plt.show()

  plt.plot(history.history['acc'])
  plt.plot(history.history['val_acc'])
  plt.title('Accuracy')
  plt.ylabel('Accuracy')
  plt.xlabel('Epoch')
  plt.legend(['train', 'validate'], loc = 'upper left')
  plt.show()

  # TODO: Visulize the plot, to be applied after traing is complete
```

Task 1 (20%): Classification using fully-connected neural networks

Dataset: MNIST

Subtask 1.1

1. Print the model structure
2. Visualize the training as it progresses (Task 0) - static plot when training is complete is OK
3. Output the final score of the test set, **should be above 90%**

```
from keras.models import Sequential
from keras.utils import to_categorical
from keras.layers import Dense  # Import the nessesarly layers
# Remember, you can normalize the data

NTRAIN = 60000
NTEST = 10000

input_train = x_train.astype('float32')/255
input_test = x_test.astype('float32')/255
labels_train = y_train
labels_test = y_test


# TODO: Flatten all images (Both training and testing images)
input_train_flat = input_train.reshape((NTRAIN, image.shape[0] * image.shape[1]))
input_test_flat = input_test.reshape((NTEST, image.shape[0] * image.shape[1]))

# TODO: Convert class vectors to binary class matrices (one-hot encoding)
labels_train_one_hot = to_categorical(labels_train)
labels_test_one_hot = to_categorical(labels_test)


# Returns a compiled model
def fully_connected_model(img_width, img_height):
  model = Sequential()  # Initalize a new model
  model.add(Dense(img_width*img_height, activation = 'relu', input_shape = (image.shape[0] * i
  #model.add(Dense(img_width*img_height, activation = 'relu'))
  model.add(Dense(10, activation = 'softmax'))


  model.compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy', metrics = ['accuracy
  return model

model = fully_connected_model(len(input_train[0]),len(input_train[0]))
model.summary()


# TODO: Train your model (training returns history)
history = model.fit(input_train_flat, labels_train_one_hot, validation_split = 0.2, epochs = 1


# Plot the training using helper function created in task 0
plot_training_score(history)

# TODO: Evaluate your model and print the score of the test set
score = model.evaluate(input_test_flat, labels_test_one_hot)
print("Accuracy:", score[1])
```

```
Non-trainable params: 0
_________________________________________________________________
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [==============================] - 6s 115us/step - loss: 0.2314 - acc:
Epoch 2/10
20864/48000 [===========>..................] - ETA: 2s - loss: 0.1001 - acc: 0.9690
Epoch 3/10
48000/48000 [==============================] - 5s 102us/step - loss: 0.0620 - acc:
Epoch 4/10
48000/48000 [==============================] - 5s 102us/step - loss: 0.0451 - acc:
Epoch 5/10
48000/48000 [==============================] - 5s 102us/step - loss: 0.0329 - acc:
Epoch 6/10
 1216/48000 [..............................] - ETA: 4s - loss: 0.0206 - acc: 0.9934
Epoch 7/10
```

Subtask 1.2

1. **Briefly** explain the plot from subtask 1.1
2. **Briefly** justify your model structure from subtask 1.1

1 As the number of epoches increases the accuracy of the model on the validation set increases, while the loss stays fairly stable. After epoch 2 you can see the accuracy of the training set surpass the accuracy of the validation set, which might indicate overfitting. Still, it is not a big problem, and the accuracy of the of the validation set increases through almost all the epochs which indicates a good model as the end result.

2 The structure was simple enough to get a good accuracy with a short training time. It was thought that a larger model, while not providing better performance, would give rise to a possibility of overfitting.

The output layer has 10 neurons, with a softmax as this is a classifier, corresponding to the ten different symbols in the set.

Task 2 (20%): Classification using convolutional layers + fully connected layers (i.e. CNN)

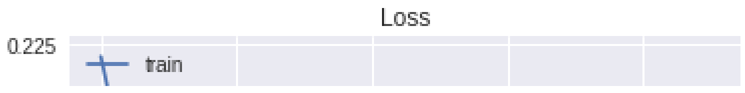
Dataset: MNIST

Subtask 2.1

1. Create a model using convolutional layers
2. Print the model structure
3. Visualize the training as it progresses (Task 0) - static plot when training is complete is OK
4. Output the final score of the test set, **should be above 98%**

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten

labels_train = y_train
labels_test = y_test
labels_train_one_hot = to_categorical(labels_train)
labels_test_one_hot = to_categorical(labels_test)

# TODO: Reshape all images (Both training and testing images) to the following
# format: (total_images, height, width, channel), eg. (4000, 28, 28, 1)
input_train_conv = x_train.reshape((60000, 28, 28, 1))
input_test_conv = x_test.reshape((10000, 28, 28, 1))

# Returns a compiled model
def conv_model(img_width, img_height):
  model = Sequential()  # Initalize a new model
  model.add(Conv2D(32,(3,3) , activation = 'sigmoid', input_shape=(img_width,img_height,1)))
  model.add(MaxPooling2D((2,2)))
  model.add(Conv2D(64,(3,3), activation = 'relu'))
  model.add(MaxPooling2D((2,2)))
  model.add(Conv2D(64,(3,3), activation = 'relu'))
  model.add(Flatten())
  model.add(Dense(64,activation = 'relu'))
  model.add(Dense(10, activation = 'softmax'))

  model.compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy', metrics = ['accuracy
  return model

model = conv_model(28,28)
model.summary()

# TODO: Train your model
history = model.fit(input_train_conv, labels_train_one_hot, validation_split = 0.2, epochs = 1



# TODO: Plot the training using the helper function created in task 0
plot_training_score(history)

# TODO: Evaluate your model and print the score of the test set
score = model.evaluate(input_test_conv, labels_test_one_hot)
print("Accuracy:", score[1])
```

```
Non-trainable params: 0
_________________________________________________________________
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
16896/48000 [=========>...................] - ETA: 7s - loss: 0.4404 - acc: 0.8559
Epoch 2/10
48000/48000 [==============================] - 11s 225us/step - loss: 0.0580 - acc:
Epoch 3/10
34368/48000 [====================>.........] - ETA: 2s - loss: 0.0405 - acc: 0.9874
Epoch 4/10
48000/48000 [==============================] - 11s 225us/step - loss: 0.0336 - acc:
Epoch 5/10
38080/48000 [======================>.......] - ETA: 2s - loss: 0.0282 - acc: 0.9917
Epoch 6/10
48000/48000 [==============================] - 11s 225us/step - loss: 0.0234 - acc:
Epoch 7/10
38976/48000 [=======================>......] - ETA: 1s - loss: 0.0214 - acc: 0.9929
Epoch 8/10
48000/48000 [==============================] - 11s 225us/step - loss: 0.0192 - acc:
Epoch 9/10
39552/48000 [=======================>......] - ETA: 1s - loss: 0.0200 - acc: 0.9943
Epoch 10/10
48000/48000 [==============================] - 11s 226us/step - loss: 0.0159 - acc:
Availible variables to plot: dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```



Subtask 2.2

1. **Briefly** discuss the following concepts of your network:
    - Filter size
    - Number of filters of each layer
    - Dropout layer (if applicable)
    - Max pooling (if applicable)
    - Regularization (if applicable)
2. **Briefly** justify your model structure from subtask 2.1

1

- **Filter Size:** Our convolutional layers have a 3x3 size filter. This means that there is a filter of size 3x3 which is convolved over each location and gives a feature map as output. The max-pooling layers has a 2x2 filter, which means that for each 2x2 of the feature maps the biggest value is chosen. This reduces the spatial size which aids in faster computation.
- **Number of filters:** Our first convolutional layer has 32 filter, while the second has 64, which is typical for a CNN to be able to extract more specific features. Choosing the amount of filters is a balancing act between accuracy and overfitting.
- **Max Pooling**: As already said we are using two 2x2 max-pooling layers to bring down the spatial complexity and the computational expense.

2

We are using two convolutional layers to extract features from the images and two max pooling layers to reduce the compexity of the feature maps. The input images are small, so that should be sufficient. This is shown in the accuracy in the end test. To be able to classify the images into one of the ten categories the feature maps need to be flattened and passed trought a couple of dense layers, where the last layer has a softmax activation function.

Task 3 (20%): Cfar10 image contest

Dataset: CIFAR-10

Get as high accuracy as you can on the CIFAR-10 image dataset.

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. Your task is to do this classification using CNN with as high accuracy as you can without using transfer learning / pretrained networks.

More information on this dataset: https://www.cs.toronto.edu/~kriz/cifar.html

Suggested alterations:

- Number of hidden layers/nodes
- Number of kernels
- Number of layers/nodes in the fully connected head

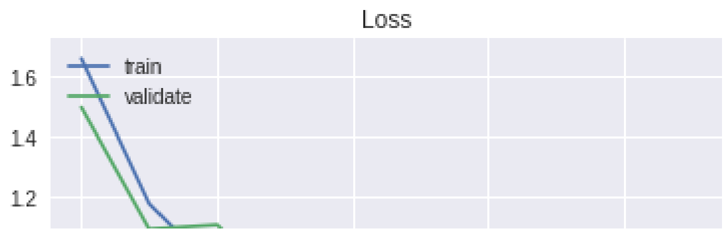
NB: Transfer learning is not allowed in this task

Double-click (or enter) to edit

```
from keras.models import Sequential
from keras.datasets import cifar10
from keras.utils import to_categorical
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout


%matplotlib inline
from matplotlib import pyplot as plt

def plot_training_score(history):
  print('Availible variables to plot: {}'.format(history.history.keys()))
  plt.plot(history.history['loss'])
  plt.plot(history.history['val_loss'])
  plt.title('Loss')
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['train', 'validate'], loc = 'upper left')
  plt.show()

  plt.plot(history.history['acc'])
  plt.plot(history.history['val_acc'])
  plt.title('Accuracy')
  plt.ylabel('Accuracy')
  plt.xlabel('Epoch')
  plt.legend(['train', 'validate'], loc = 'upper left')
  plt.show()

NTRAIN = 50000
NTEST = 10000
img_height = 32
img_width = 32


(x_train_cfar, y_train_cfar), (x_test_cfar, y_test_cfar) = cifar10.load_data()

x_train_cfar = x_train_cfar.astype('float32')/255
x_test_cfar = x_test_cfar.astype('float32')/255
```

```
input_train_conv = x_train_cfar.reshape((NTRAIN, img_height, img_width, 3))
input_test_conv = x_test_cfar.reshape((NTEST, img_height, img_width, 3))
labels_train_one_hot = to_categorical(y_train_cfar,10)
labels_test_one_hot = to_categorical(y_test_cfar,10)

def conv_model(imgw,imgh):
  model = Sequential()
  model.add(Conv2D(32,(3,3) , activation = 'relu', input_shape=(imgw,imgh,3)))
  #model.add(MaxPooling2D((2,2)))
  model.add(Dropout(0.25))
  model.add(Conv2D(64,(3,3), activation = 'relu'))
  model.add(Dropout(0.3))
  model.add(MaxPooling2D((2,2)))
  model.add(Conv2D(64,(3,3), activation = 'relu'))
  model.add(Dropout(0.23))
  model.add(MaxPooling2D((2,2)))
  model.add(Conv2D(128,(3,3), padding = 'same', activation = 'relu'))


  model.add(Flatten())
  model.add(Dense(128, activation = 'relu'))
  model.add(Dense(10, activation = 'softmax'))

  model.compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy', metrics = ['accuracy
  return model

model = conv_model(img_height,img_width)
model.summary()

# TODO: Train your model
history = model.fit(input_train_conv, labels_train_one_hot, validation_split = 0.2, epochs = 1

# TODO: Plot the training using the helper function created in task 0
plot_training_score(history)

# TODO: Evaluate your model and print the score of the test set
score = model.evaluate(input_test_conv, labels_test_one_hot)
print("Accuracy:", score[1])
```

```
Non-trainable params: 0
_________________________________________________________________
Train on 40000 samples, validate on 10000 samples
Epoch 1/10
 1600/40000 [>.............................] - ETA: 26s - loss: 2.4273 - acc: 0.127
Epoch 2/10
40000/40000 [==============================] - 18s 451us/step - loss: 1.1771 - acc:
Epoch 3/10
 1728/40000 [>.............................] - ETA: 16s - loss: 0.9725 - acc: 0.637
Epoch 4/10
40000/40000 [==============================] - 18s 450us/step - loss: 0.7922 - acc:
Epoch 5/10
 1728/40000 [>.............................] - ETA: 16s - loss: 0.6565 - acc: 0.770
Epoch 6/10
40000/40000 [==============================] - 18s 449us/step - loss: 0.5855 - acc:
Epoch 7/10
 1728/40000 [>.............................] - ETA: 16s - loss: 0.4887 - acc: 0.834
Epoch 8/10
40000/40000 [==============================] - 18s 447us/step - loss: 0.4346 - acc:
Epoch 9/10
 1728/40000 [>.............................] - ETA: 16s - loss: 0.3347 - acc: 0.888
Epoch 10/10
40000/40000 [==============================] - 18s 448us/step - loss: 0.3385 - acc:
Availible variables to plot: dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```



Task 4 (40%) - Classification using transfer learning

Dataset:

We have prepared a cat-dog dataset which includes:

- A train set of 1000 cats + 1000 dogs.
- A validation set of 400 cats + 400 dogs.
- A test set of 600 cats + 600 dogs.

The dataset is stored with the following format:

```
├── dataset
    ├── test600
    │   ├── cat
    │   └── dog
    ├── train1000
    │   ├── cat
    │   └── dog
    └── validation400
        ├── cat
        └── dog
```

```
# Run this code to download the dataset
%%capture
!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wg
!rm -r dataset; unzip temp.zip -d dataset; rm temp.zip;
```

Subtask 4.1

- Download a pretrained convolutionary neural network (e.g. VGG16) (and problem specific weights).
- Freeze a subset of upper layers weights.
- Train on the cat-dog dataset with a fresh fully connected head.
- If needed use data augmentation and any other technique to increase your accuracy.

NB: The images are not of equal size

```
%matplotlib inline
from matplotlib import pyplot as plt
from keras.applications.vgg16 import VGG16
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.utils import to_categorical
from keras.preprocessing import image
import numpy as np
import os

NTEST = 600
NTRAIN = 1001
NVAL = 400
IMG_HEIGHT = 100
IMG_WIDTH = 100
CHANNELS = 3

def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        img = image.load_img(os.path.join(folder,filename), target_size = (IMG_HEIGHT, IMG_WID
        if img is not None:
            img = image.img_to_array(img)
            img = np.asarray(img)
            images.append(img)
    images = np.asarray(images)
    return images

# Load all data
all_test_cat_images = load_images_from_folder('./dataset/test600/cat')
all_test_dog_images = load_images_from_folder('./dataset/test600/dog')
test_img = np.append(all_test_cat_images, all_test_dog_images, axis = 0)

all_val_cat_images = load_images_from_folder('./dataset/validation400/cat')
all_val_dog_images = load_images_from_folder('./dataset/validation400/dog')
val_img = np.append(all_val_cat_images,all_val_dog_images,axis = 0)

all_train_cat_images  = load_images_from_folder('./dataset/train1000/cat')
all_train_dog_images  = load_images_from_folder('./dataset/train1000/dog')
train_img = np.append(all_train_cat_images,all_train_dog_images,axis = 0)

# Flatten images

test_img = test_img.reshape((NTEST*2,IMG_HEIGHT,IMG_WIDTH,CHANNELS))
val_img = val_img.reshape((NVAL*2,IMG_HEIGHT,IMG_WIDTH,CHANNELS))
train_img = train_img.reshape((NTRAIN*2,IMG_HEIGHT,IMG_WIDTH,CHANNELS))

# Normalize data
test_img = test_img.astype('float32')/255
val_img = val_img.astype('float32')/255
train_img = train_img.astype('float32')/255

# Make labels
test_labels = np.append([1]*NTEST, [0]*NTEST)
val_labels = np.append([1]*NVAL, [0]*NVAL)
train_labels = np.append([1]*NTRAIN, [0]*NTRAIN)
```

```
#One (super)hot encoding
test_labels = to_categorical(test_labels)
val_labels = to_categorical(val_labels)
train_labels = to_categorical(train_labels)

base_model = VGG16(include_top = False, weights='imagenet', input_shape = (IMG_HEIGHT, IMG_WID

top_model = Sequential()
for layer in base_model.layers:
  layer.trainable = False
  top_model.add(layer)

top_model.add(Flatten())
top_model.add(Dense(64, activation='relu'))
top_model.add(Dropout(0.5))
top_model.add(Dense(2, activation='softmax'))

top_model.summary()
top_model.compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy', metrics = ['accura

history = top_model.fit(train_img, train_labels, epochs = 100, batch_size = 64, validation_dat
plot_training_score(history)

score = top_model.evaluate(test_img, test_labels)
print(score[1])
```

```
2002/2002 [==============================] - 7s 4ms/step - loss: 0.0194 - acc: 0.99
Epoch 90/100
2002/2002 [==============================] - 7s 4ms/step - loss: 0.0166 - acc: 0.99
Epoch 91/100
1472/2002 [=====================>........] - ETA: 1s - loss: 0.0134 - acc: 0.994620
Epoch 92/100
2002/2002 [==============================] - 7s 4ms/step - loss: 0.0187 - acc: 0.99
Epoch 93/100
2002/2002 [==============================] - 7s 4ms/step - loss: 0.0204 - acc: 0.99
Epoch 94/100
2002/2002 [==============================] - 7s 4ms/step - loss: 0.0177 - acc: 0.99
Epoch 95/100
2002/2002 [==============================] - 7s 4ms/step - loss: 0.0166 - acc: 0.99
Epoch 96/100
2002/2002 [==============================] - 7s 4ms/step - loss: 0.0174 - acc: 0.99
Epoch 97/100
1472/2002 [=====================>........] - ETA: 1s - loss: 0.0320 - acc: 0.989120
Epoch 98/100
2002/2002 [==============================] - 7s 4ms/step - loss: 0.0314 - acc: 0.98
Epoch 99/100
```

Subtask 4.2

1. Output the resulting accuracy and **briefly** explain what you have done.
2. Perform (between 2 to 5) different experiments of different hyper parameters, different model designs, different activation functions, etc. Then report the following:

   2.1. **Briefly** explain your models and make a comparison table of the accuracies you got.

   2.2. Explain why and which experiments performed better/worse than the others.

1

Read the data, flattened it and fixed the size of each image to 100x100 in 3 channels.

Created a network based on VGG16 with the pretrained weights from training with the imagenet dataset. We made our own fully connected head by making a fully connected layer with 256 filters, a dropout layer to reduce overfitting and a fully connected layer with two filters to classify the images into the two classes. The layers in VGG16 was frozen.

2

Firsty we tried to experiment with different dropput-rates in the last layer, one run with 0.5, 0.3 and 0.1. There was a small difference in the resulting score between all the three dropout-rates, but a clear difference in the loss in the validate-set. A lower dropout-rate gave more stable but a higher loss, while a higher rate gave a much more varied but lower loss. A dropout-rate of 0.5 gave the best score and the lowest loss so we contiued with that. A lower dropout-rate probably gives a lower score because the network ends up slightly overfitted.

Dropout-rate	Test score
0.1	0.840
0.3	0.845
0.5	0.848

We also tried different numbers of filters in the fully connected head. We tried 256, 128 and 64. Here The optimal number of filters turned out to be 128. This might be because 64 is too few to classify all features, while 256 becomes too many and causes some overfitting.

# of filters	Test score
64	0.842
128	0.858
256	0.848