10 - Keras Functional

Ludwig Krippahl

Keras Functional

Summary

- Transfer learning
- Exercise with Keras Functional API
- First assignment

Transfer learning

Transfer learning

Two different tasks with shared relevant factors

- Shared lower level features:

- E.g. distinguish between cats and dogs, or between horses and donkeys
- The low level features are mostly the same, only the higher level classification layers need to change

- Shared higher level representations:

- E.g. speech recognition
- The high level generation of sentences is the same for different speakers
- However, the low level feature extraction may need to be tailored to each speaker

Domain adaptation

Same underlying function but different domains

- We want to model the same mapping from input to output, but are using different sets of examples
- E.g. sentiment analysis

- Model was trained on customer reviews for movies and songs
- Now we need to do the same for electronics
- There should be only one mapping from words to happy or unhappy, but we are training on different sets with different words
- This is one example where unsupervised training (DAE) can help

Concept drift

Similar to Transfer learning or domain adaptation

- But occurs when the change is gradual over time
- This can be because the actual mapping has changed
  - E.g. changes in the economy change the factors predicting credit risk or purchases
- Or because the data distribution is changing
  - E.g. as the brand becomes more popular, customer base changes from specialized to general

Transfer learning

Take advantage of previously trained models

- E.g. Image recognition networks available in Keras:

- https://keras.rstudio.com/articles/applications.html

Break down model and problems into simpler parts

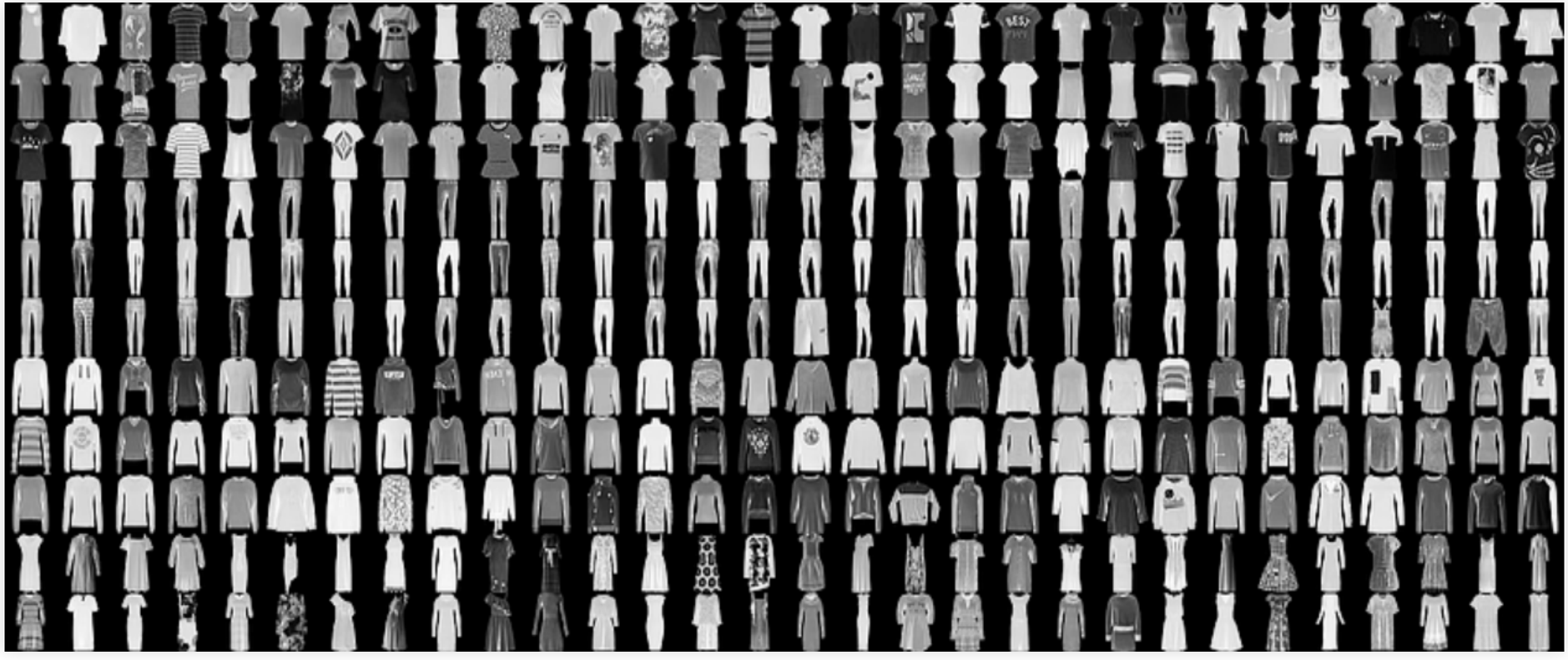
- Train one or a few layers at a time, using previously trained as inputs
- Train simpler model in part of the task or set and then add more complexity

Exercise: transfer learning Fashion MNIST to MNIST

Exercise

Transfer learning

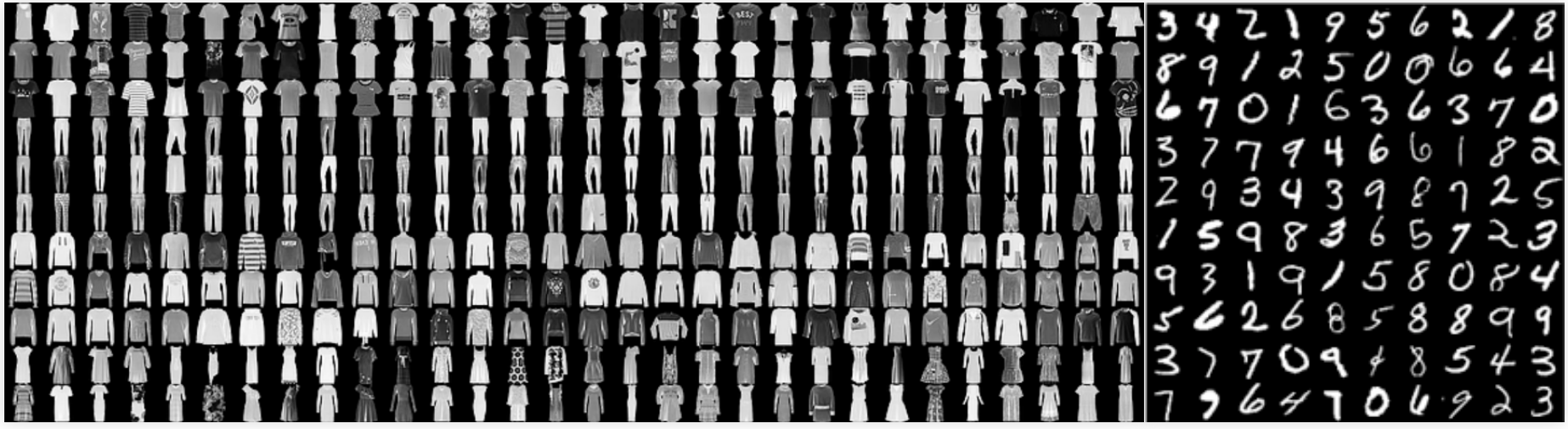
- Use network trained on Fashion MNIST



Exercise

Transfer learning

- Use network trained on Fashion MNIST



- Use Keras functional API and pre-trained model

Exercise

Transfer learning

- Using pre-trained model:

- Create same graph as model we trained last week
- Load weights from last week's trained model
- Fix weights of convolutional part (won't be retrained)
- Recreate the dense classifier and train it on MNIST

Exercise

Prepare dataset

- Same process as last week, but with MNIST instead of Fashion MNIST

```
from tensorflow import keras
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, BatchNormalization, Conv2D, Dense
from tensorflow.keras.layers import MaxPooling2D, Activation, Flatten, Dropout

((trainX, trainY), (testX, testY)) = keras.datasets.mnist.load_data()
trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
testX = testX.reshape((testX.shape[0], 28, 28, 1))
trainX = trainX.astype("float32") / 255.0
testX = testX.astype("float32") / 255.0
```

- Note: use normalization (convert to range [0,1]) if you used it last week. If not don't change inputs.
- Don't use standardization in this case

- All pixels in same range and scale but some are always 0, so variance is 0

Exercise

- Keras functional API

- Layer objects are callable (implement `__call__` method)
- Receive tensors as arguments, return tensors
- `Input` instantiates a Keras tensor (shaped like the data)
- Naming layers helps finding them again in the model

- We start by recreating the original model architecture

```
inputs = Input(shape=(28,28,1),name='inputs')
layer = Conv2D(32, (3, 3), padding="same", input_shape=(28,28,1))(inputs)
layer = Activation("relu")(layer)
layer = BatchNormalization(axis=-1)(layer)
layer = Conv2D(32, (3, 3), padding="same")(layer)
layer = Activation("relu")(layer)
layer = BatchNormalization(axis=-1)(layer)
layer = MaxPooling2D(pool_size=(2, 2))(layer)
...
```

Exercise

- Chain layers as in the original model
- Create the old model and compile it
- Load previous weights and disable training in the old model
- Note: `Flatten` layer named for use in new model

```
...
features = Flatten(name='features')(layer)
layer = Dense(512)(features)
layer = Activation("relu")(layer)
layer = BatchNormalization()(layer)
layer = Dropout(0.5)(layer)
layer = Dense(10)(layer)
layer = Activation("softmax")(layer)

old_model = Model(inputs = inputs, outputs = layer)
old_model.compile(optimizer=SGD(), loss='mse')
old_model.load_weights('fashion_model')
for layer in old_model.layers:
    layer.trainable = False
```

Exercise

- Create new dense layers for new model

- The input for this layer is the `"features"` layer in old model

- Then create the new model with:

- Old model `inputs` as input
- New softmax layer as output

- This chains old CNN to new dense layer

```
layer = Dense(512)(old_model.get_layer('features').output)
layer = Activation("relu")(layer)
layer = BatchNormalization()(layer)
layer = Dropout(0.5)(layer)
layer = Dense(10)(layer)
layer = Activation("softmax")(layer)
model = Model(inputs = old_model.get_layer('inputs').output, outputs = layer)
```

Exercise

- Now train

```
NUM_EPOCHS = 5
BS=512
opt = SGD(lr=1e-2, momentum=0.9)
model.compile(loss="categorical_crossentropy",
              optimizer=opt, metrics=["accuracy"])
model.summary()
H = model.fit(trainX, trainY,validation_data=(testX, testY),
             batch_size=BS, epochs=NUM_EPOCHS)
```

- Summary:

```
Total params: 1,679,082
Trainable params: 1,612,298
Non-trainable params: 66,784
```

- Compare with training all model from the start

Assignment 1

Gotta Catch 'Em All

- Pokemon sightings near FCT

Assignment 1

- Pokemon images, segmentation masks, main and secondary types
- Images are 64x64 pixels, 10 pokemon types (several per type)

Assignment 1

Three tasks:

- Multiclass classification
  - Predict the main type of a pokemon given the image
  - Each pokemon has exactly one main type
- Multilabel classification
  - Predict the type or types of a pokemon given the image
  - Each pokemon has one or two types
- Semantic segmentation
  - Identify the pixels corresponding to the pokemon

Assignment 1

Data provided

- Download the TP1.zip file

- dataset/
- Folder with images, csv with labels, list of pokemon types
- tp1_utils.py
- Functions for loading data and exporting results
- TP1.txt
- Questions to answer (your report)
- tp1.py
- Main file for your code. You can write other modules if you want, but this is the main one

Assignment 1

Loading data:

```
def load_data():
  """
  Returns dictionary with data set, in the following keys:
      'train_X':      training images, RGB, shape (4000,64,64,3)
      'test_X':       test images, RGB, shape (500,64,64,3)
      'train_masks':  training masks, shape (4000,64,64,1)
      'test_masks':   test masks, shape (500,64,64,1)
      'train_classes':classes for training, shape (4000,10),
                      one-hot encoded
      'train_labels': labels for training,  shape (4000,10)
                      with 1 on main and secondary types
      'test_classes': test classes, shape (500,10),
                      one-hot encoded
      'test_labels':  test labels,  shape (500,10)
                      with 1 on main and secondary types
  """
```

Assignment 1

Exporting images

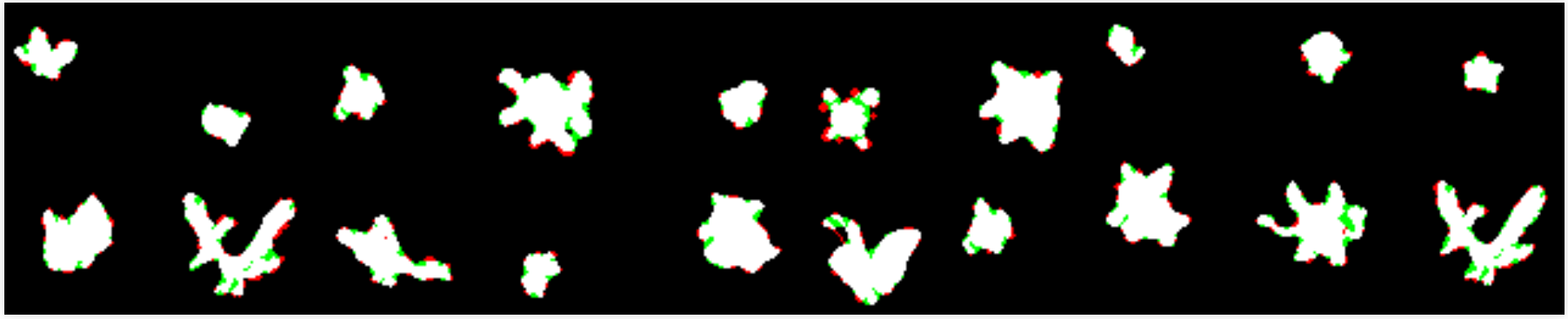
```
def images_to_pic(f_name,images,width = 20):
    """
    Saves a single image file with the images provided in the images array.
    Example:
        data = load_data()
        images_to_pic('test_set.png',data['test_X'])
    """
```

```
images_to_pic('testset.png',ds['test_X'][:20],width=10)
```

Assignment 1

```
def compare_masks(f_name,masks1,masks2,width = 20):
    """
    Creates a single image file comparing the two sets of masks
    Red pixels are pixels in mask1 but not in mask2
    Green pixels are in mask2 but not in mask2
    White pixels are in both and black in neither.
    """
```

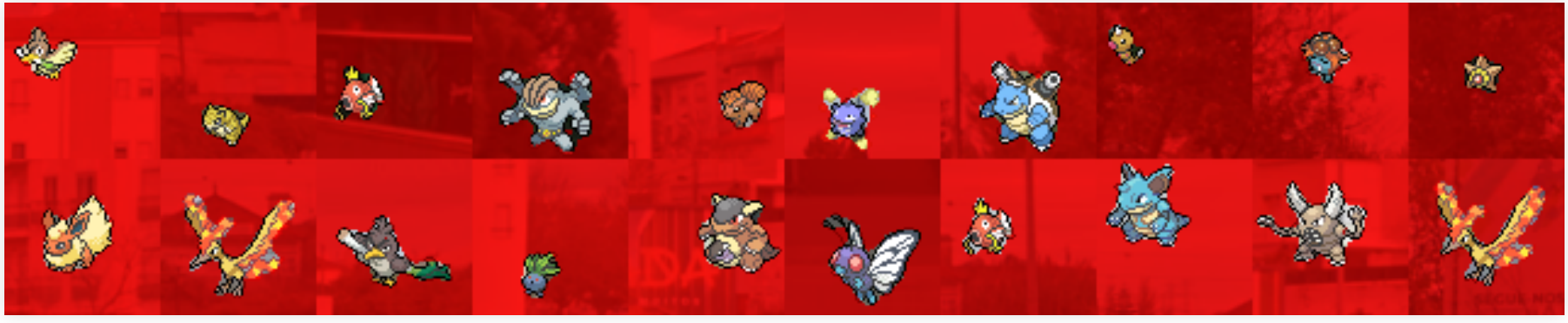
```
predicts = model.predict(ds['test_X'])
compare_masks('test_predicted.png',ds['test_masks'][:20],predicts[:20],width=10
```



Assignment 1

```
def overlay_masks(f_name,images,masks,width = 20):
    """
    Creates a single image file overlaying the masks and the images
    Region outside the masks are turned red and darker
    Example:
        predicts = model.predict(data['test_X'])
        overlay_masks('test_overlay.png',data['test_X'],predicts)
    """
```

```
predicts = model.predict(ds['test_X'])
overlay_masks('test_overlay.png',ds['test_X'][:20],predicts[:20],width=10)
```



Assignment 1

What you need to do:

- Build models, experiment, evaluate error (train and validation).
- Select best one for each task, evaluate on test set.
- Use appropriate activations and losses

- Multiclass: only one class for each example
- Multilabel: each example can have several labels
- Segmentation: pokemon or not, binary classification for each pixel

Assignment 1

Keras

- Simple models can be created with the sequential API
- But functional is more flexible and may be more practical
- Check documentation for layers:

```
BatchNormalization, Conv2D, MaxPooling2D,
Activation, Flatten, Dropout, Dense,
UpSampling2D, GlobalAveragePooling2D
```

- This is a very simple dataset to minimize hardware requirements
  - ~2.5GB RAM when training
- Few images (4000 for training + validation), likely to overfit with complex models
- Experiment

Assignment 1

Optional task (2 points out of 20)

- Use pretrained networks for feature extraction

```
core = tf.keras.applications.MobileNetV2(include_top=False,
                                         input_shape=(64,64,3))
```

- Input and output tensors in these nets:

- `core.input`, `core.output`

- Need to preprocess images (converts 0..255 into -1..1)

```
X = tf.keras.applications.mobilenet.preprocess_input(ds['train_X']*255)
```

- Check available networks:

- https://keras.io/api/applications/

- Suggestion: use the smallest (MobileNetV2, EfficientNetB0, ...)

Assignment 1

Optional task (2 points out of 20)

- Use pretrained networks for feature extraction
- Freeze the convolutional part, train the dense classifier for the two classification tasks.
- Discuss the results

- Trained faster?
- More overfitting? Less?
- Why?

Summary

Keras Functional

Summary

- Transfer learning
- Keras functional API
- Assignment 1

- Assignment submission:
- From April 18 to May 2 (+48h for fixing submission problems)
- I will extend deadline for group formation to April 17
- Check the instructions online
- (I can update them if anything needs clarification)

Exercises

- Today: transfer learning and anything left behind
- Next 2 weeks for the assignment (with few exercises)