



BST 261: Data Science II

Lecture 12

**Text Generation,
Deep Dream,
Neural Style Transfer**

**Heather Mattie
Harvard T.H. Chan School of Public Health
Spring 2 2021**

Recipe of the Day!

Watermelon Salad with Feta and Cucumber





Neural Style Transfer

Neural Style Transfer

- ◎ Another major development in deep-learning-driven image modification
- ◎ Introduced by [Leon Gatys et al.](#) in 2015
- ◎ Variations have been introduced, some even as smartphone apps
- ◎ Neural style transfer consists of **applying the style** of a reference image to a target image **while conserving the content** of the target image

The authors have created a [website](#) where you can upload your own photo and choose a style!

1 Upload photo

The first picture defines the scene you would like to have painted.



2 Choose style

Choose among predefined styles or upload your own style image.



3 Submit

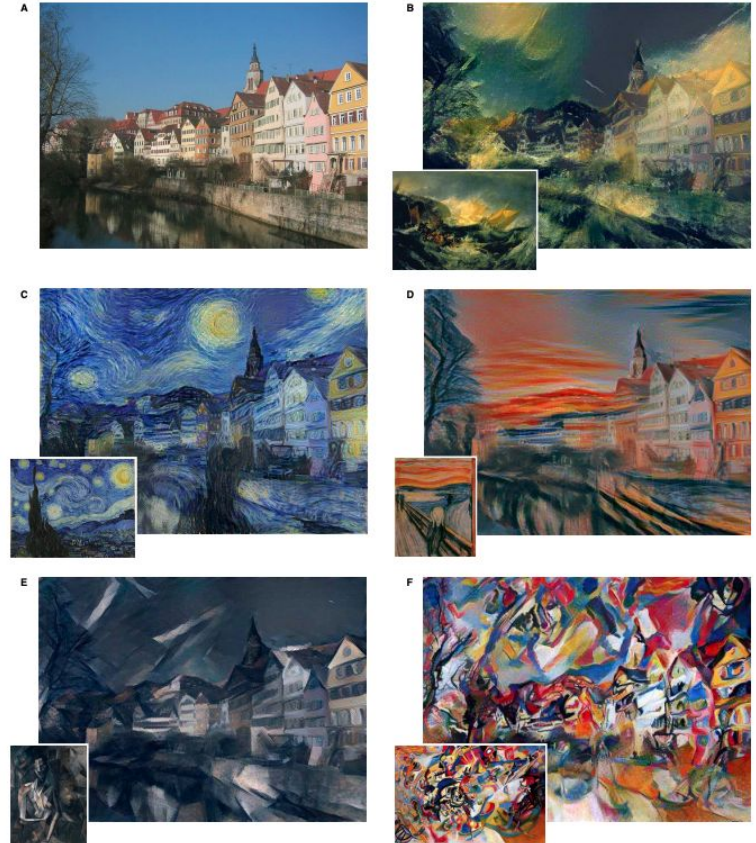
Our servers paint the image for you. You get an email when it's done.



Neural Style Transfer

Image Style Transfer Using CNNs

“ A Neural Algorithm of Artistic Style that can separate and recombine the image content and style of natural images. The algorithm allows us to produce new images of high perceptual quality that combine the content of an arbitrary photograph with the appearance of numerous well-known artworks”



Neural Style Transfer

- ◎ Style
 - Textures, colors, and visual patterns in the image, at various spatial scales
- ◎ Content
 - Higher-level macrostructure of the image

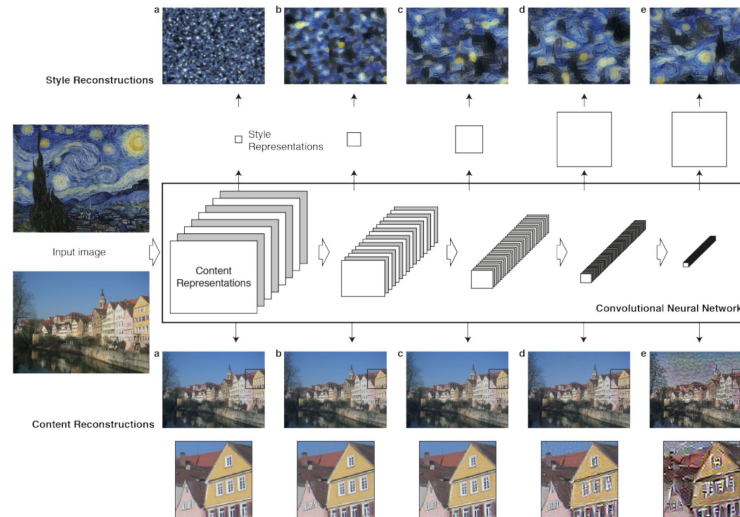
Neural Style Transfer

- ◎ Like other neural nets, we need to define and minimize a loss function
 - We want to conserve the content of the original image, while adopting the style of the reference image
 - If we can mathematically define content and style, our loss function would be:

$$\text{Loss} = \text{distance}(\text{style}(\text{ref_image}) - \text{style}(\text{generated_image})) + \text{distance}(\text{content}(\text{original_image}) - \text{content}(\text{generated_image}))$$

Neural Style Transfer

- Deep CNNs offer a way to define this loss function mathematically
- Recall:
 - Activations from earlier layers in a network contain local information about the image
 - Activations from higher layers contain increasingly global, abstract information



Neural Style Transfer



Neural Style Transfer

◎ Content loss

- The content of an image is more global and abstract and should be captured by the representations of later layers
- Loss is the L2 norm between the activations of an upper layer in a pretrained convnet, computed over the target image, and the activations of the same layer computed over the generated image
- Ensures the generated image will look similar to the original target image

Neural Style Transfer

◎ Style loss

- Uses multiple layers of the CNN
- Try to capture the appearance of the style reference image at all spatial scales extracted by the convnet, not just a single scale
- Use the Gram matrix of a layer's activations: the inner product of the feature maps of a given layer
- This inner product can be understood as representing a map of the correlations between the layer's features

Neural Style Transfer

◎ Summary

- Preserve content by maintaining similar high-level layer activations between the target content image and the generated image. The convnet should “see” both the target image and the generated image as containing the same things
- Preserve style by maintaining similar correlations within activations for both low-level layers and high-level layers. Feature correlations capture textures: the generated image and the style-reference image should share the same textures at different spatial scales

Neural Style Transfer

[Awesome blog post](#)

Content C



+

Style S



=

Generated Image G



Picasso Dancer

Neural Style Transfer



Input image



Reference style image



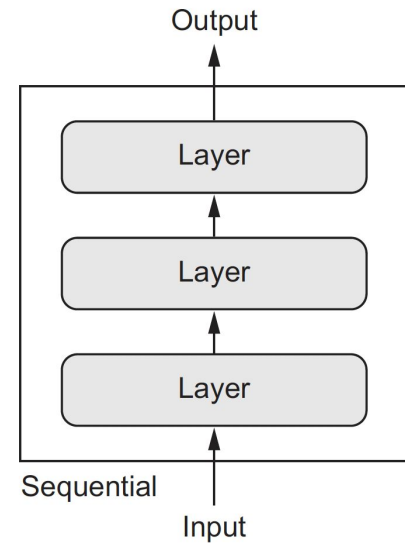
Result

The background of the slide is a light gray network pattern. It consists of numerous small circles, some of which are solid gray and others are hollow with a gray outline. These circles are interconnected by a web of thin, light gray lines, creating a complex, organic structure that resembles a neural network or a molecular structure.

Advanced Architectures

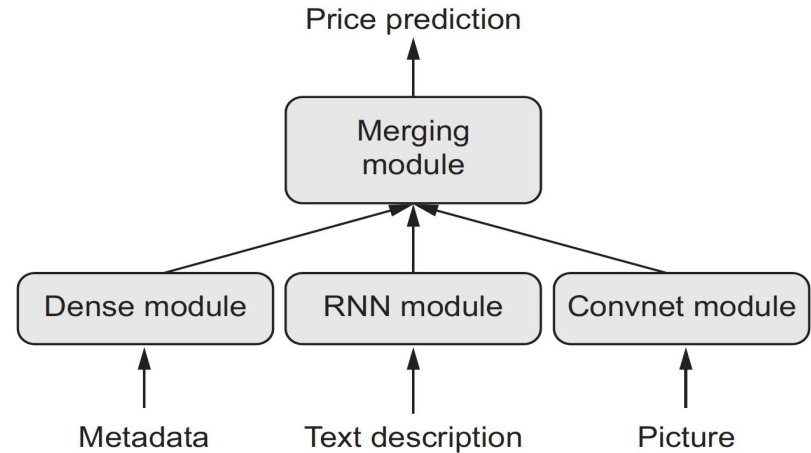
Beyond the Sequential Model

- Throughout the course we have assumed each network has exactly one input and exactly one output, and that it consists of a linear stack of layers
- But what if we have **multiple types** of inputs? Or multiple types of outputs?
- We can change the network structure - Keras makes this easy to do



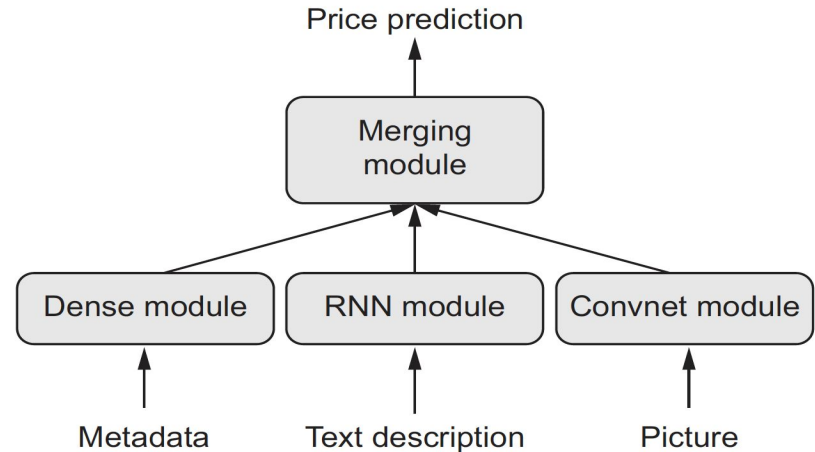
Multimodal (Multi-inputs) Model

- ⦿ Multimodal inputs merge data coming from different input sources, processing each type of data using different kinds of neural layers
- ⦿ Example: predict the most likely market price of a second-hand piece of clothing, using the following inputs:
 - User-provided **metadata** (brand, age, etc.)
 - User-provided **text** description
 - **Picture** of the item



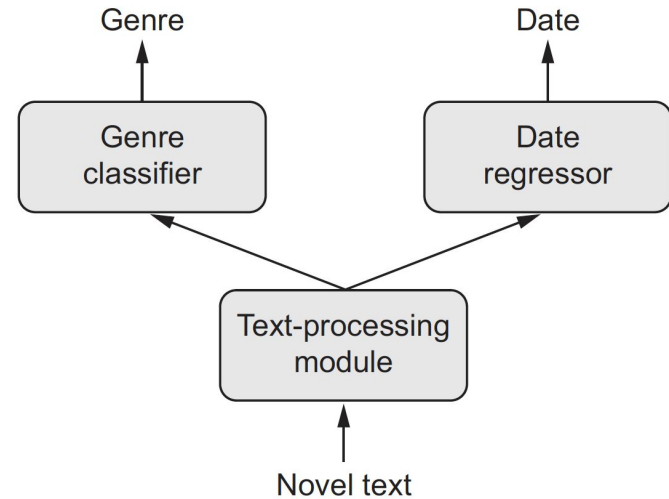
Multimodal (Multi-inputs) Model

- ◎ Suboptimal approach: **train three separate models** and then do a weighted average of their predictions
 - Information may be redundant
- ◎ Better approach: **jointly learn** a more accurate model of the data by using a model that can see all available input modalities simultaneously: a model with three input branches



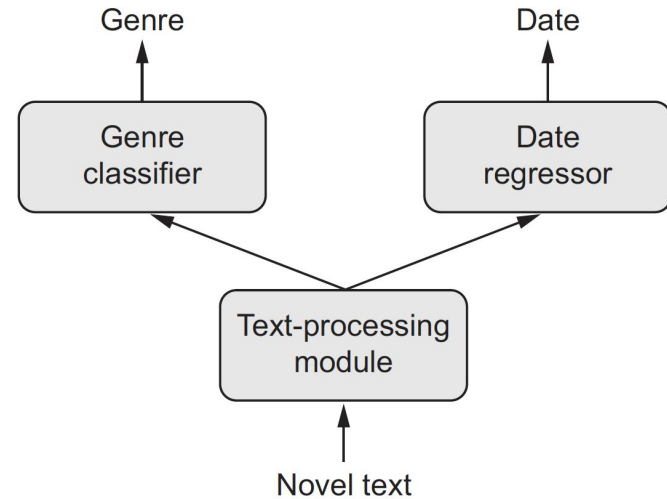
Multi-output (multihead) Model

- ◎ Predict multiple target attributes (outputs) of input data
- ◎ Example: predict the genre and date of a novel using the novel's text



Multi-output (multihead) Model

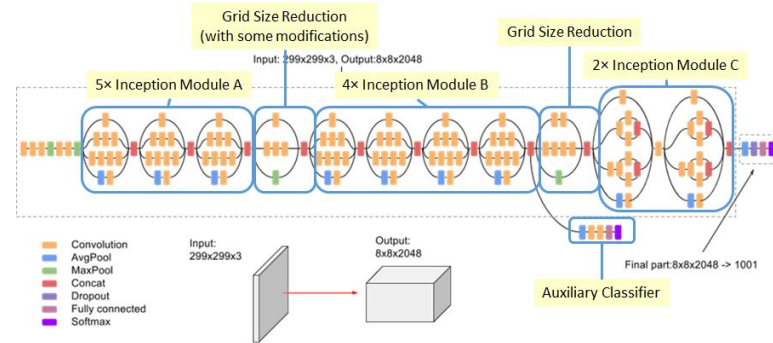
- ◎ Suboptimal approach: **train two separate models**: one for the genre and one for the date
 - But these attributes aren't statistically independent
- ◎ Better approach: **jointly predict** both genre and date at the same time
 - Correlations between date and genre of the novel helps training
- ◎ [Thesis work](#) on predicting the genre of novels using machine learning and deep learning



Directed Acyclic Graphs (DAGs)...

... not THOSE DAGs.

- ⊙ Nonlinear network architectures
- ⊙ Examples
 - Inception models
 - ResNet
 - Etc.

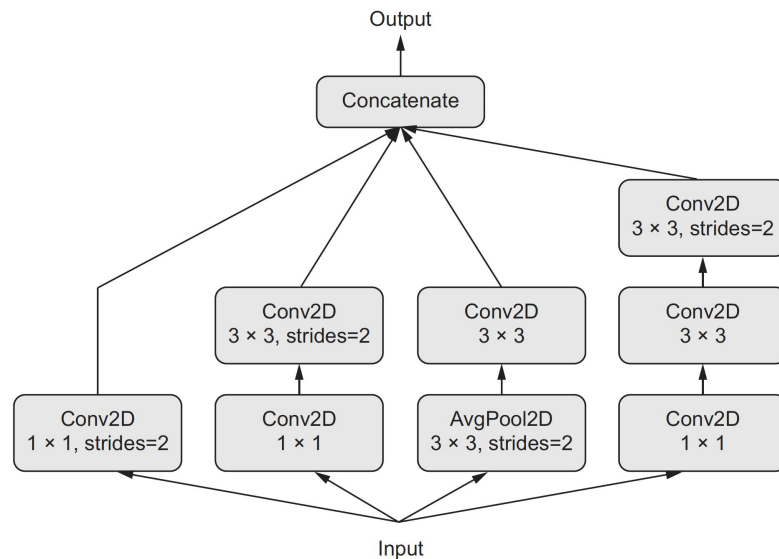


Directed Acyclic Graphs (DAGs)...

... not THOSE DAGs.

- ⊙ Nonlinear network architectures
- ⊙ Examples
 - Inception models
 - ResNet
 - Etc.

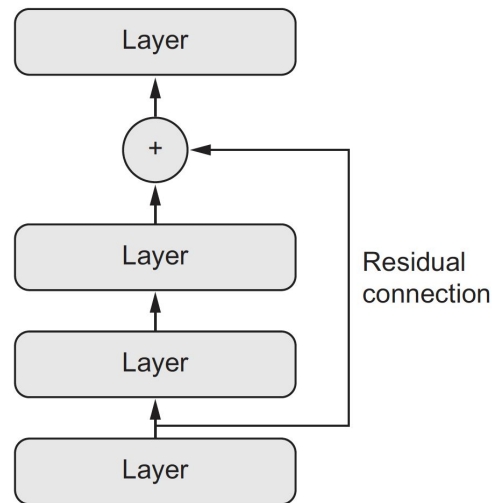
The input is processed by several convolutional branches whose outputs are merged back into a single tensor



Directed Acyclic Graphs (DAGs)...

... not THOSE DAGs.

- ⊙ Nonlinear network architectures
- ⊙ Examples
 - Inception models
 - ResNet
 - Etc.
- ⊙ A residual connection consists of reinjecting previous representations into the downstream flow of data by adding a past output tensor to a later output tensor.
- ⊙ This helps prevent information loss along the data-processing flow.



The Functional API in Keras

- ◎ Directly manipulate tensors
- ◎ Use layers as **functions** that take tensors and return tensors

```
1 model = keras.Sequential([
2     layers.Dense(32, activation = 'relu', input_shape = (64,)),
3     layers.Dense(32, activation = 'relu'),
4     layers.Dense(10, activation = 'softmax')
5 ])
6
7 input_tensor = Input(shape = (64,))
8 x = layers.Dense(32, activation = 'relu')(input_tensor)
9 x = layers.Dense(32, activation = 'relu')(x)
10 output_tensor = layers.Dense(10, activation = 'softmax')(x)
11 model = Model(input_tensor, output_tensor)
```

Sequential model
(what we've seen
before)

Functional equivalent
to the above model

The Model class turns an input tensor
and output tensor into a model

The Functional API in Keras

Keras retrieves every layer involved in going from the input tensor to the output tensor and brings them together into a graph-like structure (a Model)

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 64)	0
dense_4 (Dense)	(None, 32)	2080
dense_5 (Dense)	(None, 32)	1056
dense_6 (Dense)	(None, 10)	330
Total params: 3,466		
Trainable params: 3,466		
Non-trainable params: 0		

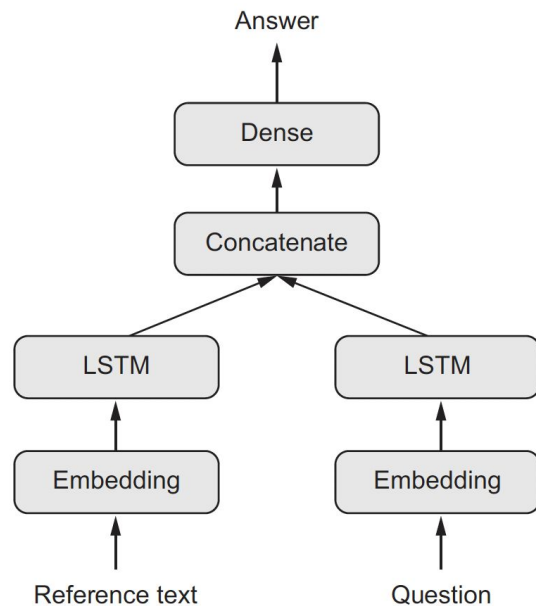
The Functional API in Keras

Everything is the same when you compile, train and evaluate an instance of Model:

```
1 model.compile(loss = 'categorical_crossentropy',  
2               optimizer = 'rmsprop',  
3               metrics = ['accuracy'])  
4  
5  
6 history = model.fit(x_train, y_train,  
7                    epochs = 10,  
8                    batch_size = 128)
```

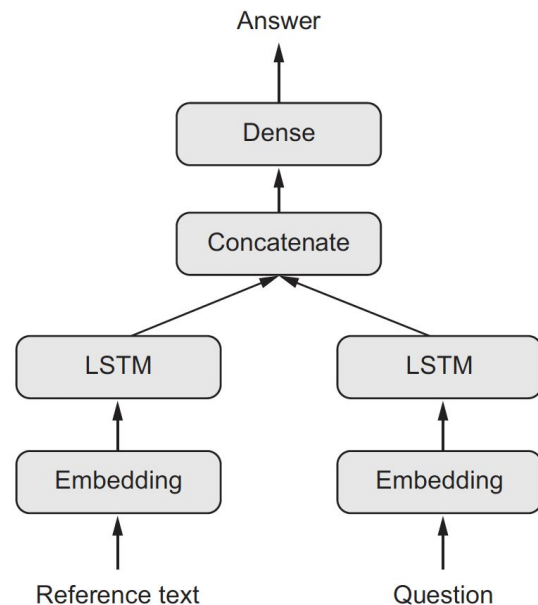

Multi-input models

- ◎ The functional API can be used to build models that have multiple inputs.
- ◎ Typically, such models at some point merge their different input branches using a layer that can combine several tensors: by adding them, concatenating them, and so on.
- ◎ This is usually done via a Keras merge operation such as `keras.layers.add`, `keras.layers.concatenate`, etc.



Multi-input models

- Example: **question-answering model**
- A typical question-answering model has two inputs: a natural-language question and a text snippet (such as a news article) providing information to be used for answering the question.
- The model must then produce an answer: in the simplest possible setup, this is a one-word answer obtained via a softmax over some predefined vocabulary



Multi-input models

```
from keras.models import Model
from keras import layers
from keras import Input
text_vocabulary_size = 10000
question_vocabulary_size = 10000
answer_vocabulary_size = 500

text_input = Input(shape=(None,), dtype = 'int32', name = 'text')
embedded_text = layers.Embedding(64, text_vocabulary_size)(text_input)
encoded_text = layers.LSTM(32)(embedded_text)

question_input = Input(shape=(None,), dtype = 'int32', name = 'question')
embedded_question = layers.Embedding(32, question_vocabulary_size)(question_input)
encoded_question = layers.LSTM(16)(embedded_question)

concatenated = layers.concatenate([encoded_text, encoded_question], axis = -1)
answer = layers.Dense(answer_vocabulary_size, activation = 'softmax')(concatenated)

model = Model([text_input, question_input], answer)
model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['acc'])
```

} Branch for encoding
the text input

} Branch for encoding
the question

Multi-input models

How do you train this two-input model?

There are two possible APIs:

1. You can feed the model a list of Numpy arrays as inputs, or
2. You can feed it a dictionary that maps input names to Numpy arrays.

Fitting using a
list of inputs



```
import numpy as np
num_samples = 1000
max_length = 100
text = np.random.randint(1, text_vocabulary_size, size = (num_samples, max_length))

question = np.random.randint(1, question_vocabulary_size,
                             size = (num_samples, max_length))
answers = np.random.randint(0, 1, size = (num_samples, answer_vocabulary_size))

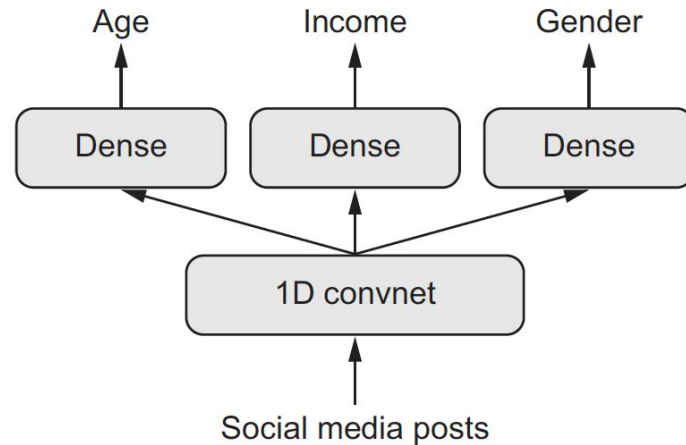
model.fit([text, question], answers, epochs = 10, batch_size = 128)
model.fit({'text': text, 'question': question}, answers, epochs=10, batch_size=128)
```

Fitting using
a dictionary
of inputs



Multi-output models

- Example: a network that attempts to simultaneously predict different properties of the data, such as a network that takes as input a series of social media posts from a single anonymous person and tries to predict attributes of that person, such as age, gender, and income level



Multi-output Models

Can specify different functions
for different outcomes

```
from keras import layers
from keras import Input
from keras.models import Model
vocabulary_size = 50000
num_income_groups = 10
posts_input = Input(shape=(None,), dtype = 'int32', name = 'posts')
embedded_posts = layers.Embedding(256, vocabulary_size)(posts_input)

x = layers.Conv1D(128, 5, activation='relu')(embedded_posts)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation='relu')(x)

age_prediction = layers.Dense(1, name = 'age')(x)
income_prediction = layers.Dense(num_income_groups, activation='softmax', name='income')(x)
gender_prediction = layers.Dense(1, activation='sigmoid', name='gender')(x)
model = Model(posts_input, [age_prediction, income_prediction, gender_prediction])
```


Multi-output Models

- ◎ This model requires the ability to specify different loss functions for different heads of the network:
 - Age prediction is a scalar regression task
 - Gender prediction is a binary classification task
 - Income prediction in this example is a multiclass classification task
 - ◎ Income categories
- ◎ But because gradient descent requires you to minimize a scalar, you must combine these losses into a single value in order to train the model.
- ◎ The simplest way to combine different losses is to sum them all.
 - In Keras, you can use either a list or a dictionary of losses in **compile** to specify different objects for different outputs; the resulting loss values are summed into a global loss, which is minimized during training.

```
model.compile(optimizer='rmsprop', loss = ['mse', 'categorical_crossentropy', 'binary_crossentropy'])
```

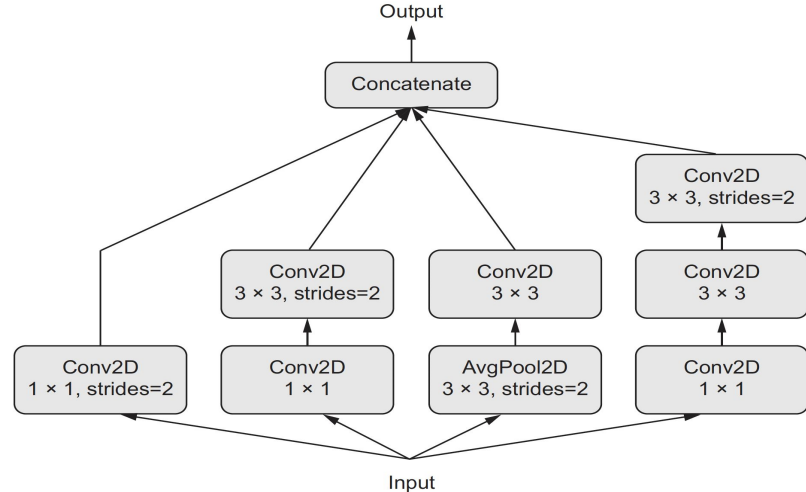
Multi-output Models

- ◎ A note on imbalanced loss contributions
 - Imbalances will cause the model representations to be optimized preferentially for the task with the largest individual loss, at the expense of the other tasks
 - To remedy this, you can assign different levels of importance to the loss values in their contribution to the final loss
 - This is particularly useful if the losses' values use different scales
 - Example:
 - ◎ MSE takes values around 3-5
 - ◎ Cross-entropy loss can be as low as 0.1
 - ◎ Here we could assign a weight of 10 for the cross-entropy loss and a weight of 0.25 to the MSE loss

```
model.compile(optimizer = 'rmsprop',  
loss=['mse', 'categorical_crossentropy', 'binary_crossentropy'], loss_weights = [0.25, 1., 10.]
```

DAGs of Layers

- ⦿ You can also code complex network architectures in Keras
- ⦿ Can specify independent branches
- ⦿ Need to make sure the output of each branch is the same size so you can concatenate them at the end



```
from keras import layers

branch_a = layers.Conv2D(128, 1, activation='relu', strides=2)(x)

branch_b = layers.Conv2D(128, 1, activation='relu')(x)
branch_b = layers.Conv2D(128, 3, activation='relu', strides=2)(branch_b)

branch_c = layers.AveragePooling2D(3, strides=2)(x)
branch_c = layers.Conv2D(128, 3, activation='relu')(branch_c)

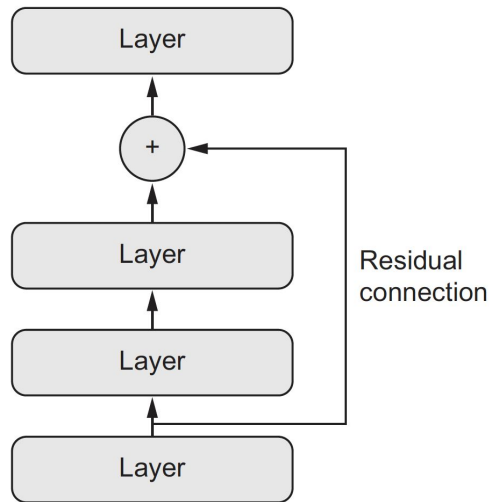
branch_d = layers.Conv2D(128, 1, activation='relu')(x)
branch_d = layers.Conv2D(128, 3, activation='relu')(branch_d)
branch_d = layers.Conv2D(128, 3, activation='relu', strides=2)(branch_d)

output = layers.concatenate([branch_a, branch_b, branch_c, branch_d], axis = -1)
```

Residual Connections

- Residual connections are a common graph-like network component found in many post-2015 network architectures, including Xception
- In general, adding residual connections to any model that has more than 10 layers is likely to be beneficial
- A residual connection consists of making the output of an earlier layer available as input to a later layer, effectively creating a shortcut in a sequential network.
 - Rather than being concatenated to the later activation, the earlier output is summed with the later activation, which assumes that both activations are the same size

Reintroduces x



```
from keras import layers
x = ...
y = layers.Conv2D(128, 3, activation='relu', padding='same')(x)
y = layers.Conv2D(128, 3, activation='relu', padding='same')(y)
y = layers.Conv2D(128, 3, activation='relu', padding='same')(y)
y = layers.add([y, x])
```

The background of the slide is a light gray network pattern. It consists of numerous small circles, some of which are solid blue and others are hollow white with blue outlines. These circles are interconnected by a web of thin, light gray lines, creating a complex, organic structure that resembles a neural network or a molecular structure.

Advanced Architecture Patterns

Batch Normalization

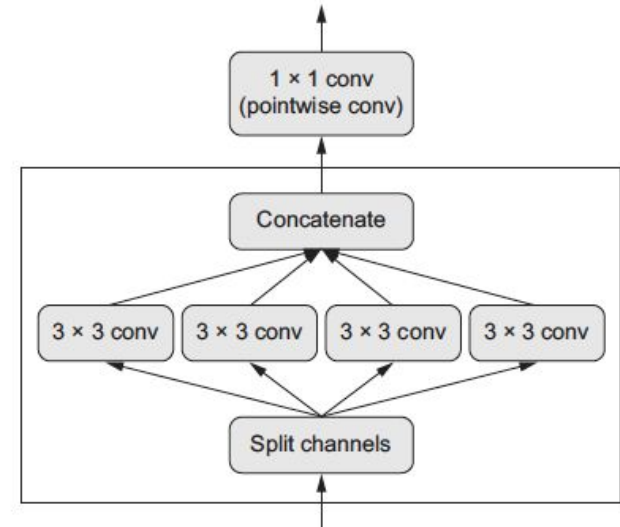
- ◎ Normalization is a broad category of methods that seek to make different samples seen by a machine learning model more similar to each other, which helps the model learn and generalize well to new data
- ◎ We have already done normalization by transforming data to have mean 0 and standard deviation equal to 1

Batch Normalization

- ◎ Batch normalization is a type of layer (**BatchNormalization** in Keras)
 - It can adaptively normalize data even as the mean and variance change over time during training.
 - It works by internally maintaining an exponential moving average of the batch-wise mean and variance of the data seen during training.
- ◎ The main effect of batch normalization is that it helps with gradient propagation—much like residual connections—and thus allows for deeper networks

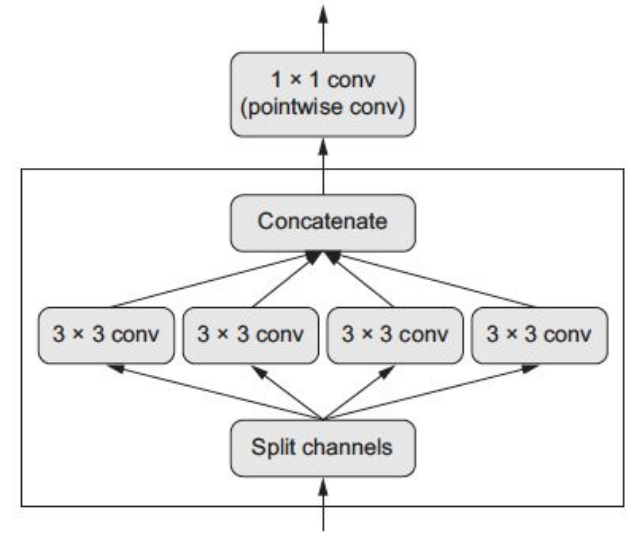
Depthwise Separable Convolution

- ◎ A depthwise separable convolution layer performs a spatial convolution on each channel of its input, independently, before mixing output channels via a pointwise convolution
 - Separates the learning of spatial features and the learning of channel-wise features
 - Makes sense if you assume that spatial locations in the input are highly correlated but different channels are fairly independent



Depthwise Separable Convolution

- It requires significantly fewer parameters and involves fewer computations, making it much faster
- Tends to learn better representations using less data, resulting in better-performing models
- Are the basis for the Xception architecture



Depthwise Separable Convolution

Example of an image classification task on a small data set

```
from keras.models import Sequential, Model
from keras import layers

height = 64
width = 64
channels = 3
num_classes = 10

model = Sequential()
model.add(layers.SeparableConv2D(32, 3, activation='relu', input_shape=(height, width, channels)))
model.add(layers.SeparableConv2D(64, 3, activation='relu'))
model.add(layers.MaxPooling2D(2))

model.add(layers.SeparableConv2D(64, 3, activation='relu'))
model.add(layers.SeparableConv2D(128, 3, activation='relu'))
model.add(layers.MaxPooling2D(2))

model.add(layers.SeparableConv2D(64, 3, activation='relu'))
model.add(layers.SeparableConv2D(128, 3, activation='relu'))
model.add(layers.GlobalAveragePooling2D())

model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
```

Hyperparameter Optimization

- ◎ There is no way of knowing which values are the optimal ones before building and training your model
- ◎ Even with experience and intuition, your first pass at the values will be suboptimal
- ◎ There are no formal rules to tell you which values are the best ones for your task
- ◎ You can (and we have in this course) tweak the value of each hyperparameter by hand
 - But this is inefficient
- ◎ It's better to let the machine do this, and there is a whole field of research around this
 - Bayesian optimization
 - Genetic algorithms
 - Simple random search
 - Grid search
 - Etc.

The decision between manual and automated comes down to a balance between understanding your model and computational cost

Hyperparameter Optimization

◎ The process:

1. Choose a set of hyperparameters (automatically)
2. Build the corresponding model
3. Fit it to your training data and measure the final performance on validation data
4. Choose the next set of hyperparameters to try (automatically)
5. Repeat
6. Eventually, measure performance on your test data

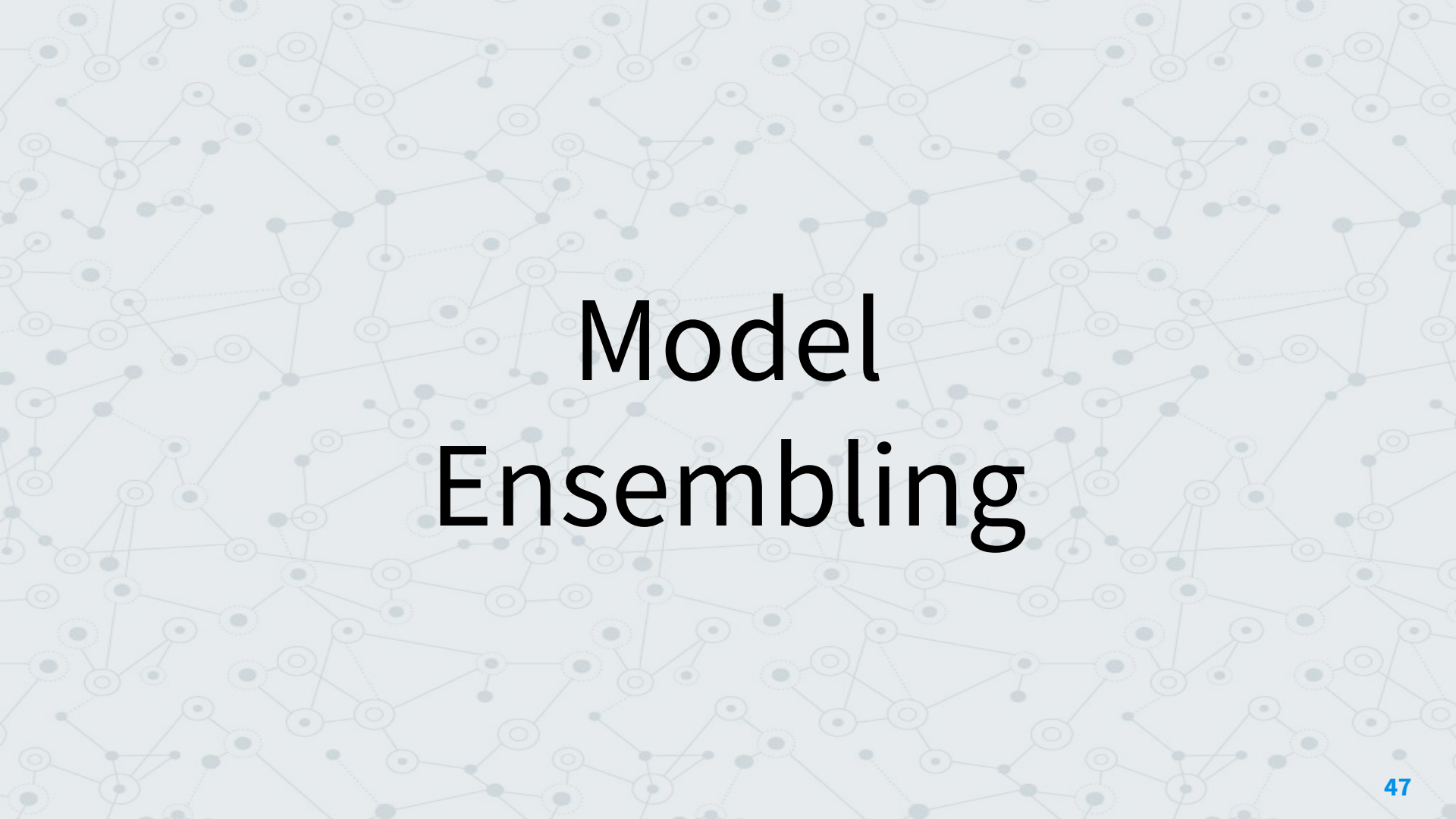
Hyperparameter Optimization

- ◎ More tools available each year
 - One option is [Hyperopt](#)
 - ◎ A Python library for hyperparameter optimization that internally uses trees of Parzen estimators to predict sets of hyperparameters that are likely to work well
 - Another option is [Hperas](#)
 - ◎ Another library that integrates Hyperopt for use with Keras
 - [Weights and Biases](#)
 - [SageMaker](#)
 - [Comet.ml](#)
 - [This great post](#)

Hyperparameter Optimization

⦿ CAUTION!!

- Can easily overfit to the validation data
- Can be very computationally expensive

The background of the slide is a light gray network pattern. It consists of numerous small circles, some of which are solid gray and others are hollow with a gray outline. These circles are interconnected by a web of thin, light gray lines, creating a complex, organic structure that resembles a neural network or a data graph.

Model Ensembling

Model Ensembling

- ◎ Ensembling consists of pooling together the predictions of a set of different models to produce better predictions
- ◎ Ensemble models are **as good or better** than one model alone
- ◎ Assumes that different good models trained independently are **likely to be good for different reasons** - each model looks at slightly different aspects of the data to make its predictions, getting part of the “truth”, but not all of it
- ◎ In Keras can combine predictions in different ways (mean, weighted average, etc.)
- ◎ SuperLearner

[In R](#)

[In Python](#)

SuperLearner

