DataLab Preparation (Week 4, DataLab II, Wednesday)

2. Convolutional Neural Networks

**2a Keras offers three different ways for building models: sequential model, functional API, and model subclassing. Explain these three strategies, provide examples, and indicate in which situations we should favor one option over the others.**

1. Sequential model

* This is the most approachable API, limited to simple stacks of layers.
* Its applicability is extremely limited: it can only express models with a single input and a single output, applying one layer after the other in a sequential fashion.
* It is best suitable for situations where we want to build simple models.

Example:

# The network architecture

from tensorflow import keras

from tensorflow.keras import layers

model = keras.Sequential([

layers.Dense(512, activation='relu'),

layers.Dense(10, activation='softmax')

])

# The compilation step

model.compile(optimizer='rmsprop',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Preparing the image data

train\_images = train\_images.reshape((60000, 28 \* 28))

train\_images = train\_images.astype('float32') / 255

test\_images = test\_images.reshape((10000, 28 \* 28))

test\_images = test\_images.astype('float32') / 255

# Fit the model to the training data

model.fit(train\_images, train\_labels, epochs=5, batch\_size=128)

1. The Functional API

* It represents a mid-point between usability and flexibility. It is the most commonly used model-building API.
* It facilitates model visualisation and feature extraction because a Functional model is an explicit graph structure. This is also the reason why you can reuse previous layer outputs.
* It is an explicit data structure, ehere you can view, inspect, and modify layers.
* It is best used for models with multiple inputs and/ or multiple outputs.

Example:

import numpy as np

import keras

from keras import layers

num\_samples = 1280

vocabulary\_size = 10000

num\_tags = 100

num\_departments = 4

title = keras.Input(shape=(vocabulary\_size,), name="title") # Variable-length sequence of ints

text\_body = keras.Input(shape=(vocabulary\_size,), name="text\_body") # Variable-length sequence of ints

tags = keras.Input(shape=(num\_tags,), name="tags") # Binary vectors of size `num\_tags`

features = layers.Concatenate()([title, text\_body, tags])

features = layers.Dense(64, activation="relu")(features)

priority = layers.Dense(1, activation="sigmoid", name="priority")(features)

department = layers.Dense(num\_departments, activation="softmax", name="department")(features)

model = keras.Model(inputs=[title, text\_body, tags], outputs=[priority, department])

title\_data = np.random.randint(0, 2, size = (num\_samples, vocabulary\_size))

text\_body\_data = np.random.randint(0, 2, size = (num\_samples, vocabulary\_size))

tags\_data = np.random.randint(0, 2, size = (num\_samples, num\_tags))

priority\_data = np.random.random(size=(num\_samples, 1))

department\_data = np.random.randint(0, 2, size = (num\_samples, num\_departments))

model.compile(optimizer="adam",

loss=["mean\_squared\_error", "categorical\_crossentropy"],

metrics=["mean\_squared\_error", "accuracy"])

model.fit([title\_data, text\_body\_data, tags\_data],

[priority\_data, department\_data],

epochs=1)

model.evaluate([title\_data, text\_body\_data, tags\_data],

[priority\_data, department\_data])

priority\_preds, department\_preds = model.predict([title\_data, text\_body\_data, tags\_data])

1. Subclassing the Model class

* It is a low-level option where you write everything from scratch, making it the most advanced model-building pattern.
* You won’t get access to Keras features.
* You are more at risk of making mistakes.
* It is a piece of bytecode.
* Once the model is instantiated, its forward pass becomes a complete black box.
* It is ideal if you want full control over every little detail.

Example:

class CustomerTicketModel(keras.Model):

def \_init\_ (self, num\_departments):

super().\_init\_()

self.concat\_layer = layers.Concatenate()

self.mixing\_layer = layers.Dense(64, activation="relu")

self.priority\_scorer = layers.Dense(1, activation="sigmoid")

self.department\_classifier = layers.Dense(num\_departments, activation="softmax")

def call(self, inputs):

title = inputs["title"]

text\_body = inputs["text\_body"]

tags = inputs["tags"]

features = self.concat\_layer([title, text\_body, tags])

features = self.mixing\_layer(features)

priority = self.priority\_scorer(features)

department = self.department\_classifier(features)

return priority, department

model = CustomerTicketModel(num\_departments=4)

priority, department = model({"title": title\_data, "text\_body": text\_body\_data, "tags": tags\_data})

model.compile(optimizer="adam",

loss=["mean\_squared\_error", "categorical\_crossentropy"],

metrics=["mean\_squared\_error", "accuracy"])

model.fit({"title": title\_data, "text\_body": text\_body\_data, "tags": tags\_data},

(priority\_data, department\_data),

epochs=1)

model.evaluate({"title": title\_data, "text\_body": text\_body\_data, "tags": tags\_data},

(priority\_data, department\_data))

priority\_preds, department\_preds = model.predict({"title": title\_data, "text\_body": text\_body\_data, "tags": tags\_data})

**2b What is model checkpoint in Keras? What does it allow you to do?**

Model Checkpoint is a callback method which lets you continually save the model at different points during training, and optionally save only the version of the model that achieved the best performance at the end of an epoch.

**2c Explain in your own words the process of creating customized training and evaluation loops in Keras.**

Creating a customized training loop in Keras involves building an artificial fit() function. Firstly, we compute the model’s output inside a gradient tape to obtain a loss value for the batch of data used at the point of running a specific epoch. Then we retrieve the gradients of the loss. Lastly, we update the model’s weights so as to lower the loss value of the current batch of data. The steps are repeated for the batches of data available.

Creating a customized evaluation loop in Keras on the other hand, involves building an artificial evaluate() function. It is comprised of a ‘for’ loop that calls a test\_step() function, whose purpose is to compute the model’s outputs to obtain a loss value for the batch of data.

**2d Explain why the patterns learned by CNNs are translation-invariant.**

The patters learned by CNNs are translation-invariant due to the propriety of their convolutional layers which learn local patters, unlike Dense layers, which learn global patterns in their input feature space. This makes CNNs data-efficient when processing images because the visual world is fundamentally translation-invariant.

**2e Explain how CNNs can learn spatial hierarchies of patterns.**

First convolutional layer learns small local patterns such as edges, lines, or simple textures; second convolutional layer will learn larger patterns made of the features of the first layers, and so on. The visual world is fundamentally spatially hierarchical, which allows CNNs to efficiently learn increasingly complex and abstract visual concepts.

3. Error Analysis

**3a Describe the steps of the error analysis process.**

Error analysis systematically identifies the most common errors made by the model.

After detecting that there is a certain type of error, we need to decide if it is worth fixing it. The way to do this is by collecting a random subset (approx 50-100 error samples) of the test set examples that were misclassified and then counting how many have that error. If the percentage is higher than accepted, it is important to fix them. If the model has an error of ε, then we want a testing size of approx. 100/ε. Additionally, we need to periodically update the test set to minimize overfitting because the errors that will be fixed are concentrated in the initial test set used to identify the error categories.

Afterwards, we manually examine the errors and try to identify the most common categories of errors. We then count how many errors belong to which category. The most prevalent error class is prioritized.

We then move onto fixing the error type by applying different measures, such as using mislabeled data as input and minimizing the error, or going back to the dataset and correcting the labels, etc. It all depends on the error at hand.

Fixing the errors on the training set is also important because that is what the model is fed. Therefore, performing error analysis on the training data should also be done.

**3b What is the primary purpose of conducting error analysis in machine learning?**

[] To increase the training speed of the model.

[x] To identify and prioritize the most common errors made by the model.

[] To enhance the graphical representation of data.

[] To reduce the number of features in the model.

**3c Describe how you would perform error analysis on the model of your creative brief** project. What kinds of errors would you look for in the images of your dataset?

I would first analyse the chosen metrics for evaluating my model. Then I would compare plot the number of incorrectly classified images to see in which class they are most prevalent. I would select a sample of correctly and incorrectly classified images to get an idea of the problem at hand. Lastly, depending on the detected problem, I would take action. The error that I believe would be found is images of non-alcoholic beverages being misclassified as alcoholic.

**3d Consider a model designed to classify outdoor images into three weather categories: sunny, cloudy, and rainy. Let's assume that after training and testing the model, you noticed that the following images were incorrectly classified by the model.**

**Based on the images and predicted/labeled classes, perform the following tasks:**

1. Review the Images: Carefully examine the provided collection of misclassified images. Pay attention to the variety in weather conditions and the nature of the errors.
2. Identify Error Types: Identify the types of errors present in these images.
3. Categorize Errors: Group the misclassified images based on the type of error.
4. Justify Categorization: Provide explanations for why each image falls into its respective category or categories.

* Most images were incorrectly labelled as ‘rainy’ (62.5% of test set)
* The most misclassified samples were images labelled ‘sunny’ (56.25% of train set). I hypothesise that this is happening because the data is not labelled properly. For example, images 4, 10, 11, 16 are improperly categorised as ‘sunny’ when they are not.
* The model appears to categorise images that contain umbrellas as ‘rainy’ even when that is not the case.
* The misclassified images as ‘rainy’ are: 1, 6, 7, 10-16. I would put images into the category False Positive (mislabelling as 'rainy').

**3e Based on the results of the categorization done for the previous exercise, calculate the percentage of errors in each category (quantitative analysis). Then, discuss which category has the highest error rate.**

Rainy: 62.5%

Cloudy: 25%

Sunny: 12.5%

I believe the root of the error in the category ‘rainy’ is due to multiple factors, such as ambiguity in rain features such as raindrops, dark clouds, wet surfaces, and umbrellas.

Rain-related elements like raindrops, dark clouds, and umbrellas represent challenges in classification. Inconsistent labelling criteria contribute to misclassification, which leads to the model being unable to generalise properly. Additionally, insufficient contextual information further complicates accurate classification.

**3f Based on the results of the quantitative analysis performed in the previous exercise, formulate hypotheses about why these errors occurred. Then, suggest and discuss potential improvements for the model and/or dataset.**

Data is not labelled properly => go back to the dataset and make sure the labels are in order

Not enough training data => expand the dataset to ensure a wide range of rainy weather conditions to improve model robustness

Dataset Imbalance => augment the dataset to balance the distribution of weather classes, preventing the model from being biased towards the majority class

Contextual Information => provide contextual information during model training, such as weather conditions at the time the image was captured