Alla svar är genererade från min RAG chatbot som använder litteraturen från skolan samt egna böcker och källor som jag tycker håller hög standard.

## 1. Hur är AI, Maskininlärning och Deep Learning relaterat?

AI, machine learning, and deep learning are related in a hierarchical manner. AI is the broadest field, encompassing systems that mimic human intelligence. Machine learning is a subset of AI, focusing on systems that learn from data without explicit programming. Deep learning is a specialized part of machine learning, utilizing neural networks with multiple layers to handle complex data patterns.

**Hierarchical Relationship:**

1. **Artificial Intelligence (AI)**: The broadest category, involving systems that perform tasks requiring human intelligence.
2. **Machine Learning (ML)**: A subset of AI, focusing on algorithms that learn from data to make decisions or predictions.
3. **Deep Learning (DL)**: A specialized subset of ML, using multi-layered neural networks to model complex data patterns.

This hierarchy reflects how each field builds upon the previous one, enabling increasingly sophisticated capabilities in data processing and pattern recognition.

## 2. Hur är Tensorflow och Keras relaterat?

TensorFlow and Keras are closely integrated. Keras is a high-level Deep Learning API that is part of TensorFlow. It was originally a standalone project but is now officially included in TensorFlow, serving as its preferred high-level API. Keras provides an easy-to-use interface for building models, leveraging TensorFlow's core functionalities for computations. This integration means that Keras cannot function independently and requires TensorFlow to operate, making Keras an essential and inseparable component of the TensorFlow ecosystem.

## 3. Vad är en parameter? Vad är en hyperparameter?

In machine learning, **parameters** and **hyperparameters** are distinct concepts that play crucial roles in model training and performance.

- **Parameters**: These are the internal variables of a machine learning model that are learned from the training data. They are adjusted during the training

process to minimize the loss function and enable the model to make accurate predictions. Examples include the weights and biases in a neural network or the coefficients in a linear regression model. Parameters are intrinsic to the model and are not set by the practitioner.

- **Hyperparameters**: These are external settings that are chosen by the data analyst before training the model. They control the learning process and can significantly affect the model's performance. Common examples of hyperparameters include the learning rate, batch size, number of epochs, and the choice of optimizer. The document emphasizes the importance of hyperparameters like the learning rate, which can be determined through methods such as starting with a low rate and increasing it to observe training dynamics. The choice of optimizer and batch size are also highlighted as critical hyperparameters, with batch size impacting both computational efficiency and training stability.

In summary, while parameters are learned from the data during training, hyperparameters are set by the practitioner to guide the learning process. The document provides insights into hyperparameters but does not explicitly discuss parameters.

## 4. När man skall göra modellval och modellutvärdering kan man använda tränings-, validerings- och testdataset. Förklara hur de olika delarna kan användas.

When performing model selection and evaluation, the datasets are used as follows:

1. **Training Dataset**:
    - **Purpose**: To train the model, allowing it to learn patterns and relationships from the data.
    - **Process**: The model is built using this dataset, and hyperparameters are adjusted based on training performance.
2. **Validation Dataset**:
    - **Purpose**: To evaluate the model after training to assess its generalization ability and prevent overfitting.
    - **Process**: The model is tested on this dataset to tune hyperparameters and select the best-performing model.
3. **Test Dataset**:

- **Purpose**: To provide a final, unbiased evaluation of the model's performance on unseen data.
- **Process**: The finalized model is tested on this dataset to estimate its real-world performance.

This structured approach ensures that the model is both accurately trained and effectively generalizes to new data.

## 5. Förklara vad nedanstående kod gör:

<think> Alright, let's break down the provided code step by step to understand what it does.

**1. Determining the Number of Columns:**

```
n_cols = x_train.shape[1]
```

This line extracts the number of columns (features) from the training dataset

`x_train`

. The shape attribute of a numpy array or pandas DataFrame returns a tuple representing its dimensions. Here,

`x_train.shape[1]`

gives the number of columns.

**2. Initializing the Neural Network Model:**

```
nn_model = Sequential()
```

This creates a sequential neural network model using Keras. Sequential models are built by adding layers one after another, which is straightforward for simple architectures.

**3. Adding the First Hidden Layer:**

```
nn_model.add(Dense(100, activation='relu', input_shape=(n_cols,)))
```

- **Dense Layer (100 units):** This is a fully connected neural network layer with 100 neurons.
- **ReLU Activation:** The Rectified Linear Unit (ReLU) function is used to introduce non-linearity into the model. It outputs

`max(0, input)`

, which helps the model learn more complex patterns in the data.

- **Input Shape:**

`input_shape=(n_cols,)`

specifies the shape of the input data. Here, it's set to match the number of columns in the training data.

**4. Adding Dropout for Regularization:**

```
nn_model.add(Dropout(rate=0.2))
```

- **Dropout:** This technique randomly sets 20% (rate=0.2) of the neurons during training to zero. It helps prevent overfitting by forcing the network to learn multiple representations of the data, making it more robust.

**5. Adding the Second Hidden Layer:**

```
nn_model.add(Dense(50, activation='relu'))
```

- **Dense Layer (50 units):** Another fully connected layer with 50 neurons.
    - **ReLU Activation:** Again, ReLU is used here for the same reasons as in the first hidden layer.

**6. Adding the Output Layer:**

```
nn_model.add(Dense(1, activation='sigmoid'))
```

- **Dense Layer (1 unit):** This is the output layer with a single neuron, suitable for binary classification tasks.
    - **Sigmoid Activation:** The sigmoid function outputs values between 0 and 1, which is ideal for binary classification problems where the target variable is a probability.

**7. Compiling the Model:**

```
nn_model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy'])
```

- **Optimizer:** The Adam optimizer is used to update the model's weights during training. It's a stochastic gradient descent optimizer that adapts the learning rate for each parameter.
- **Loss Function:**

`binary_crossentropy`

is the loss function used for binary classification. It measures the difference between the model's predicted probabilities and the actual labels.

- **Metrics:** The model tracks accuracy during training, which is the proportion of correctly classified examples.

**8. Implementing Early Stopping:**

```
early_stopping_monitor = EarlyStopping(patience=5)
```

- **Early Stopping:** This callback stops the training process if the model's performance on the validation set does not improve for 5 consecutive epochs (iterations over the entire training dataset). This prevents overfitting by halting training when the model stops generalizing well.

**9. Training the Model:**

```
nn_model.fit(
    x_train,
    y_train,
    validation_split=0.2,
    epochs=100,
    callbacks=[early_stopping_monitor])
```

- **Training Data:** The model is trained on

`x_train`

and

`y_train`

, which are the feature and target training datasets.

- **Validation Split:** 20% of the training data is used for validation to monitor the model's performance during training.
- **Epochs:** The model is trained for up to 100 epochs. However, with early stopping, training may stop earlier if the model's performance on the validation set plateaus.
- **Callbacks:** The early stopping monitor is included to potentially halt training early to prevent overfitting.

**Summary:** The code defines a neural network model for binary classification using the Keras Sequential API. It includes two hidden layers with ReLU activation and a Dropout layer for regularization. The model is compiled with the Adam optimizer and binary cross-entropy loss, tracking accuracy as a metric. Early stopping is implemented to prevent overfitting, and the model is trained on the provided data with a portion set aside for validation. This approach is designed to build a robust model that generalizes well to unseen data.

## 6. Vad är syftet med att regularisera en modell?

The purpose of regularizing a model is to reduce overfitting by constraining the model, making it less sensitive to small variations in the training data. Regularization achieves this by adding a penalty term to the cost function, which encourages smaller weights and reduces variance without significantly increasing bias.

## 7. ”Dropout” är en regulariseringsteknik, vad är det för något?

Dropout is a regularization technique employed in neural networks to mitigate overfitting by introducing stochastic noise during training. The process involves randomly deactivating a specified fraction of neurons, determined by a dropout rate p, where 0<p<1. This forces the network to learn robust features by preventing reliance on any single neuron, effectively training an ensemble of sub-networks. During inference, techniques like Monte Carlo Dropout (MC Dropout) can be applied to provide uncertainty estimates by sampling from these sub-networks. This approach enhances model generalization and offers a practical way to assess prediction confidence in real-world applications.

**Mathematical Notation:**

- Dropout rate: p
- Active neurons during training: $1-p$

**Code Example (Python):**

```
from tensorflow.keras.layers import Dropout

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense


# Example model with dropout

model = Sequential()

model.add(Dense(64, activation='relu', input_shape=(784,)))

model.add(Dropout(0.5))  # Dropout rate of 0.5

model.add(Dense(10, activation='softmax'))
```

This code snippet demonstrates the integration of a dropout layer with a rate of 0.5 in a neural network model, effectively deactivating half of the neurons during training to enhance regularization.

### 8. ”Early stopping” är en regulariseringsteknik, vad är det för något?

Early stopping is a regularization technique that involves monitoring the model's performance on a validation set during training. When the validation performance begins to degrade, indicated by an increase in the validation objective after a certain number of epochs, the training process is halted. This prevents the model from overfitting by stopping before it learns the noise in the training data. The document mentions using 20% of the training data as a validation set for this purpose, calculating around 375 minibatch updates per epoch. Thus, early stopping acts as a form of regularization by stopping training when validation performance starts to decrease.

### 9. Din kollega frågar dig vilken typ av neuralt nätverk som är populärt för bildanalys, vad svarar du?

Convolutional Neural Networks (CNNs), particularly the ResNet50 model pretrained on the ImageNet database, are popular for image recognition tasks. While Multilayer Perceptrons (MLPs) are also used for image classification, such as in Google Images, CNNs like ResNet50 are more widely recognized and utilized due to their architecture, which is specifically designed to handle spatial data efficiently.

### 10. Förklara översiktligt hur ett ”Convolutional Neural Network” fungerar.

A Convolutional Neural Network (CNN) operates by processing images through a series of convolution and pooling layers to extract hierarchical features. The network begins by applying multiple filters to the input image, generating feature maps that highlight specific patterns. These feature maps are then downsampled using pooling layers, reducing their spatial dimensions while retaining important information. This process is repeated, with increasing numbers of filters to capture more complex features. Once the feature maps are sufficiently small, they are flattened into a 1D array and fed into fully-connected layers to produce class probabilities. Regularization techniques like dropout and

weight decay are used to prevent overfitting, enhancing the network's generalization ability.

---

**Step-by-Step Explanation:**

1. **Convolutional Layers:**
    - **Filters:** The network applies multiple small filters to the input image. Each filter convolves with the image, scanning across its width and height.
    - **Feature Maps:** Each filter generates a feature map, which highlights specific features in the image. The number of feature maps is equal to the number of filters used.
    - **Channels:** The input image typically has three channels (RGB), while the feature maps from multiple filters combine to form a three-dimensional output.
2. **Activation Function:**
    - **ReLU (Rectified Linear Unit):** After the convolution operation, the ReLU activation function is applied to each feature map. This introduces non-linearity, allowing the network to learn more complex features beyond simple linear combinations.
3. **Pooling Layers:**
    - **Max Pooling:** This layer reduces the spatial dimensions of the feature maps by summarizing non-overlapping regions (e.g., 2x2 blocks) with the maximum value. This downsampling step adds translation invariance and reduces computational complexity.
4. **Architecture:**
    - **Stacking Layers:** A CNN typically consists of multiple convolutional and pooling layers stacked sequentially. Each convolutional layer increases the number of filters, allowing the network to detect more intricate features as it deepens.
    - **Hierarchical Learning:** The initial layers learn basic features (edges, lines), while deeper layers combine these to form more complex features (textures, objects), enabling the network to perform tasks like image classification effectively.

By systematically applying these operations, a CNN efficiently extracts and classifies features from images, leveraging both local and global information.

## 11. Vad gör nedanstående kod?

First, I'll start by examining the code itself. The first line,

```
model.save("model_file.keras")
```

, is used to save a Keras model to a file. The second line,

```
my_model = load_model("model_file.keras")
```

, loads the saved model back into the environment.

## 12. Deep Learning modeller kan ta lång tid att träna, då kan GPU via t.ex. Google Colab skynda på träningen avsevärt. Skriv mycket kortfattat vad CPU och GPU är.

**GPU (Graphics Processing Unit):** A specialized electronic circuit designed to quickly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device. GPUs are particularly adept at parallel processing, making them highly efficient for tasks like 3D rendering, scientific computations, and machine learning.

**CPU (Central Processing Unit):** The primary component of a computer that processes most of the data and performs most of the calculations and logical operations. CPUs are optimized for serial processing, executing instructions in a sequential manner, and are essential for running the operating system and applications.

In the context of performance, GPUs can offer significant speed improvements for parallel tasks, while CPUs remain crucial for handling sequential operations and general computing tasks.