Deep Learning (CS 470, CS 570)

Module 4, Lecture 4: Image Augmentation and Dropout Regularization

Deep Learning Objective and Challenges

One of the major objectives of machine learning as well as deep learning techniques is generalization i.e. the learning such aspects or features of the data that would be applicable even outside the training data set. While generalized learning can be a big challenge for the deep learning algorithms, there are certain techniques that facilitate this. We will learn two such techniques such as **data augmentation** and **dropout regularization**.

Image Augmentation

Within class variation: variation of data among many samples of a same class.

Example:




Class Truck. Variation of shapes, viewing angles, scaling etc.

Image Augmentation

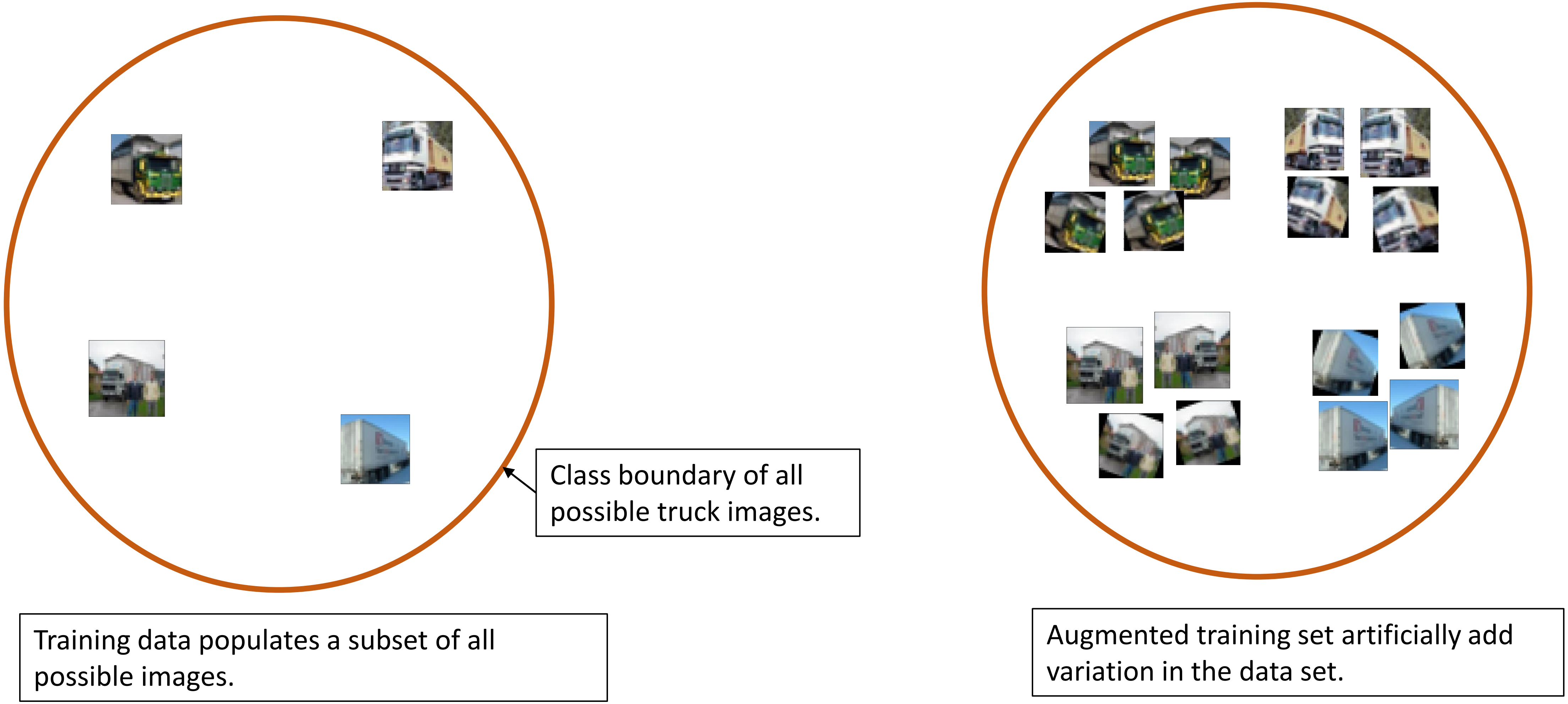


Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

Image Augmentation

Why image augmentation!

- Machine learning model needs to learn how to ignore within-class-variations
- Image augmentation artificially generates samples with within-class-variations
- This helps the model to generalize and improve performance.

How to verify your augmentation is good!

Ask the question whether the augmented images can be part of the real world (test) data.

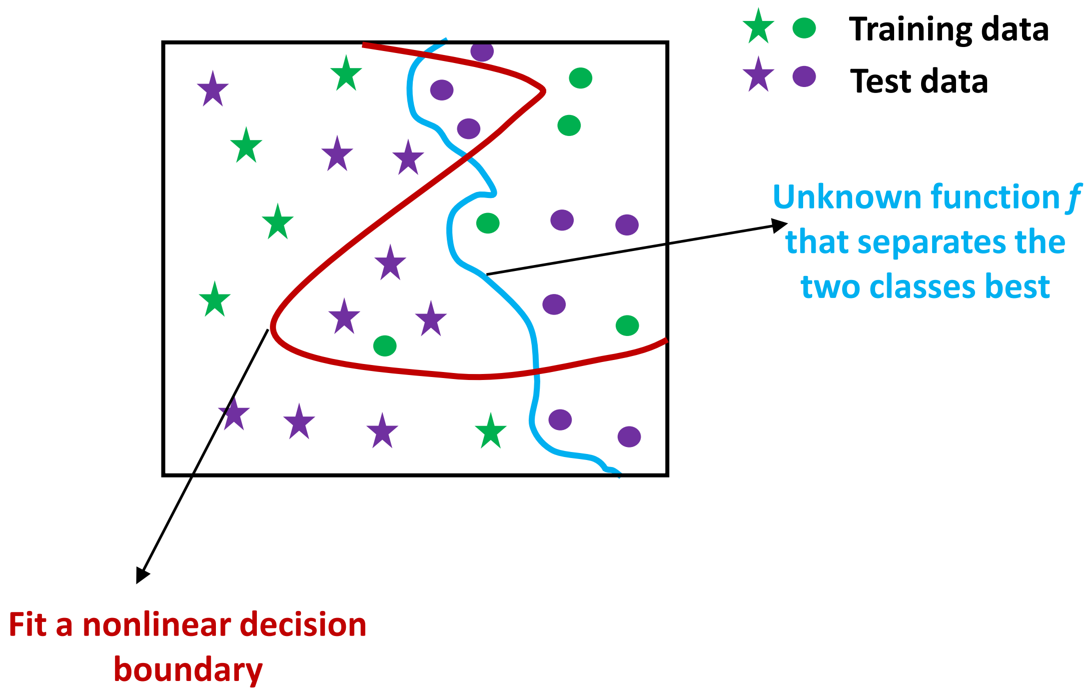
Quiz: which augmented images are good and why?

Original image

Augmented images

Dropout Regularization

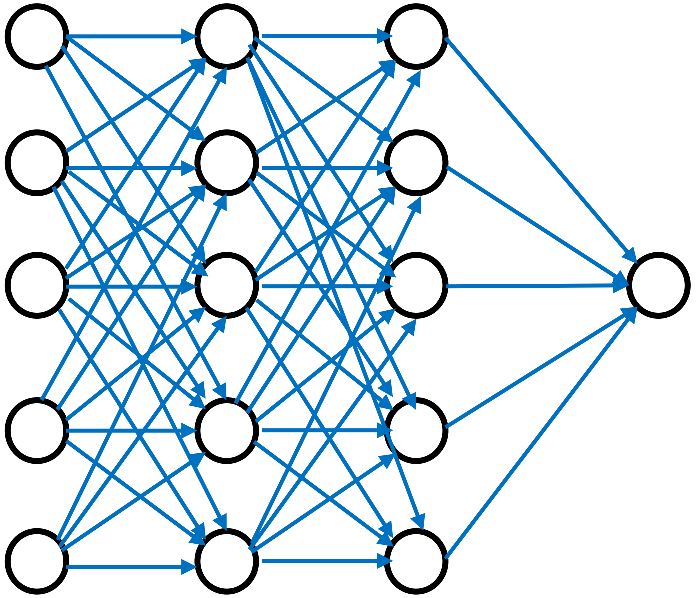
Problem of overfitting during training:



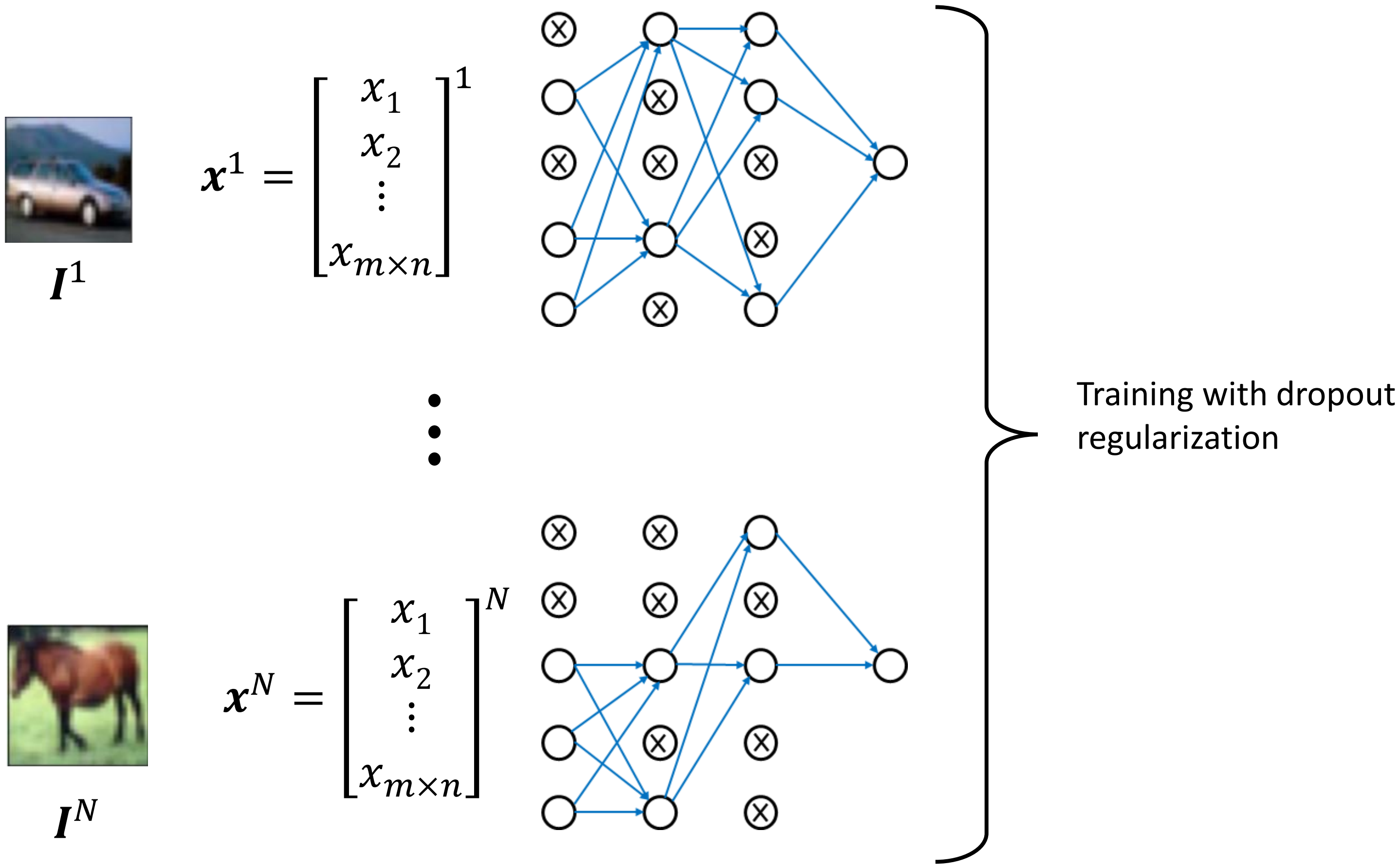
Overfitting:

- Decision boundary is fit based on the training data to get best training accuracy.
- In the process decision boundary might fit the noise (bad data points) in the data
- Result:
  - High training accuracy
  - Low test accuracy
- Solution: dropout regularization

Dropout Regularization



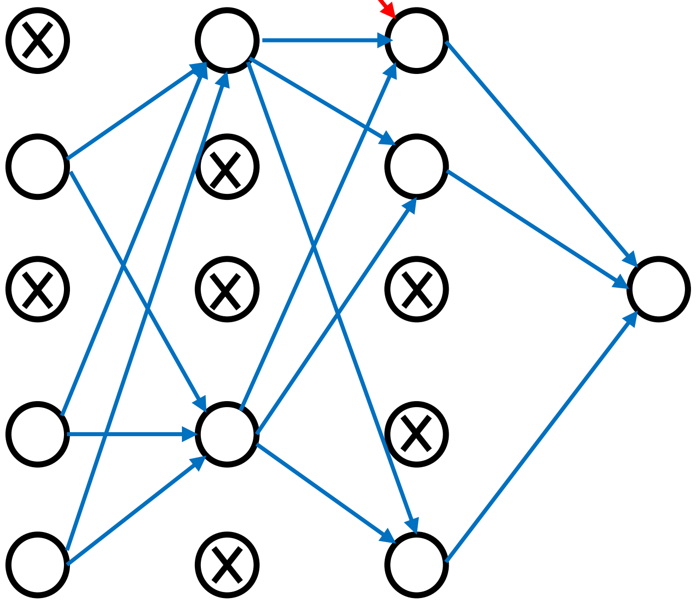
MLP architecture



Dropout regularization: each iteration of training, drop a node and its connections with a probability $\boldsymbol{p}$

Dropout Regularization

$$\boxed{z^{3}{}_{1} = \frac{1}{p}\sum_{i\in[1,4]} w_i x_i}$$



Training time:

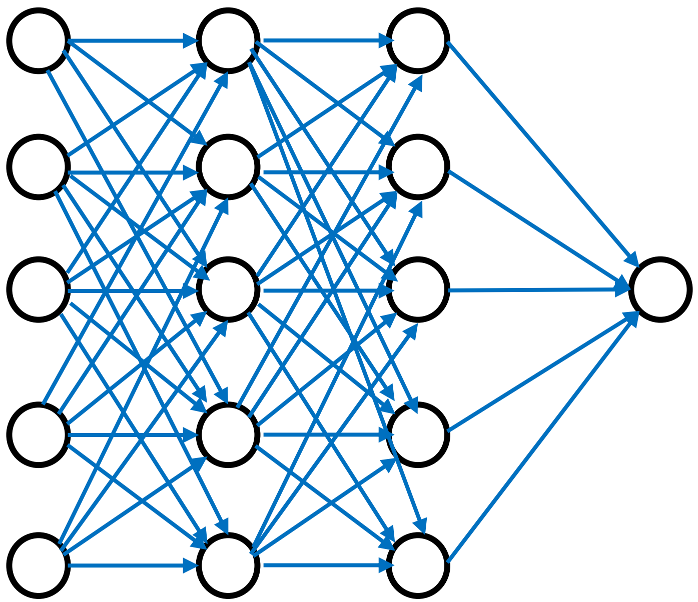
- For each training iteration of a batch or stochastic gradient descent
  - For each node:
    drop it and all its connections
    with probability p
  - During the forward pass normalize the input of a node so that it matches the expected input at the testing time.
  - During the backward pass, update the connections weights that are active

In each iteration of the training, we are generating a different network. All the networks generated during the training are overlapping, i.e. they share the common connection weights.

Dropout Regularization

Testing time:

- We use an average network formed by all the training networks.
- Like ensemble methods, the output of the network acts as the average output of all the training networks.



In each iteration of the training, we are generating a different network. All the networks generated during the training are overlapping, i.e. they share the common connection weights.

Dropout Regularization

Importing packages and loading dataset:

```
[3] # TensorFlow and tf.keras
    import tensorflow as tf
    from tensorflow import keras

    # Helper libraries
    import numpy as np
    import matplotlib.pyplot as plt

    print(tf.__version__)
```

```
2.3.0
```

```
[25] # load digit dataset
     (train_images, train_labels), (test_images, test_labels) = keras.datasets.mnist.load_data()


     print("Shape of the training dataset, number of images and resolution:", train_images.shape)
     print("All distinct training labels:", np.unique(train_labels))
```

```
Shape of the training dataset, number of images and resolution: (60000, 28, 28)
All distinct training labels: [0 1 2 3 4 5 6 7 8 9]
```

Dropout Regularization

MLP model without dropout:

```
#Model old
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(10)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10, validation_split=0.1)
```

Dropout Regularization

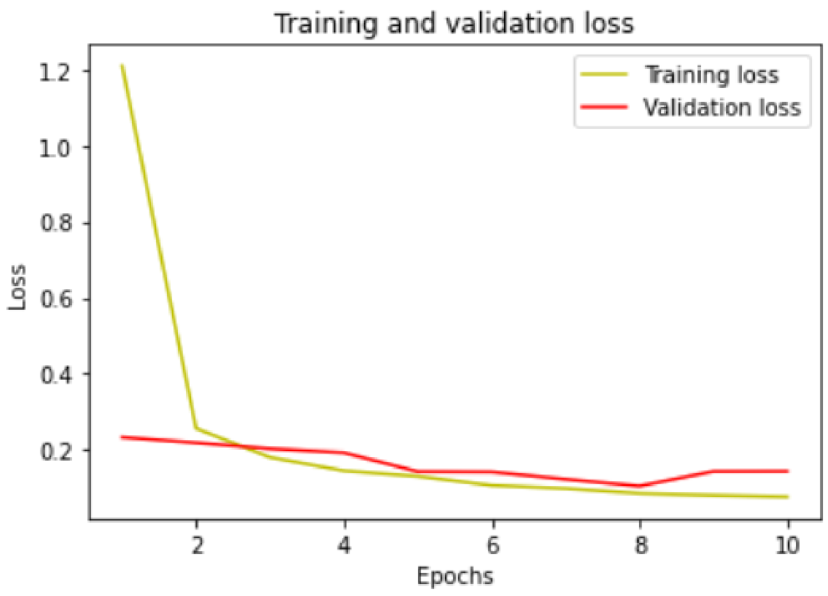
MLP model with dropout:

```
#Model dropout
from tensorflow.keras import datasets, layers, models

model_dropout = models.Sequential()
model_dropout.add(layers.Flatten(input_shape=(28, 28)))      #input_shape=(28, 28) for digit
model_dropout.add(layers.Dense(128))
model_dropout.add(layers.Dropout(0.3))
model_dropout.add(layers.Activation('relu'))
model_dropout.add(layers.Dense(128))
model_dropout.add(layers.Dropout(0.2))
model_dropout.add(layers.Activation('relu'))
model_dropout.add(layers.Dense(10))



model_dropout.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history_dropout = model_dropout.fit(train_images, train_labels, epochs=10, validation_split=0.1)
```

Dropout Regularization

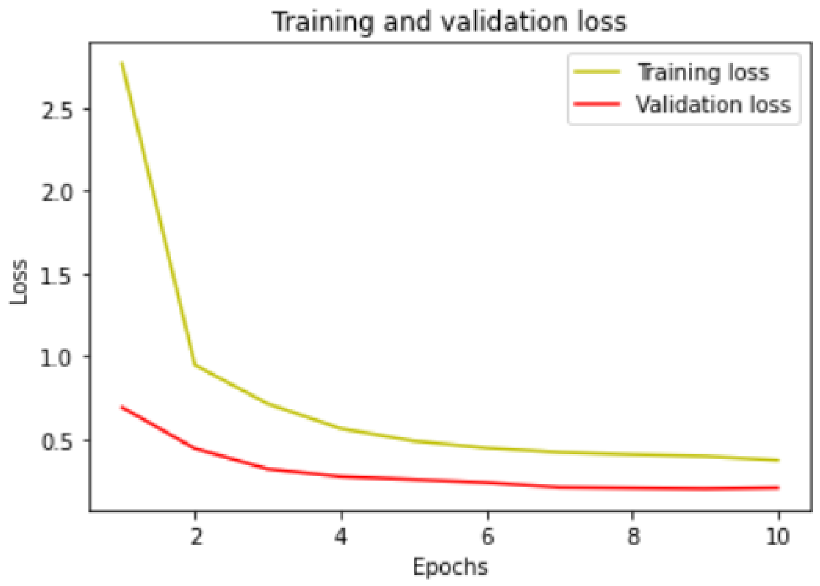
Training and validation loss without dropout:

```
[28] loss = history.history['loss']
     val_loss = history.history['val_loss']
     epochs = range(1, len(loss) + 1)
     plt.plot(epochs, loss, 'y', label='Training loss')
     plt.plot(epochs, val_loss, 'r', label='Validation loss')
     plt.title('Training and validation loss')
     plt.xlabel('Epochs')
     plt.ylabel('Loss')
     plt.legend()
     plt.show()
```



Training and validation loss with dropout:

```
loss = history_dropout.history['loss']
val_loss = history_dropout.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Additional Readings

[Data Augmentation]

[Dropout regularization]

[Overfitting in Deep Learning]