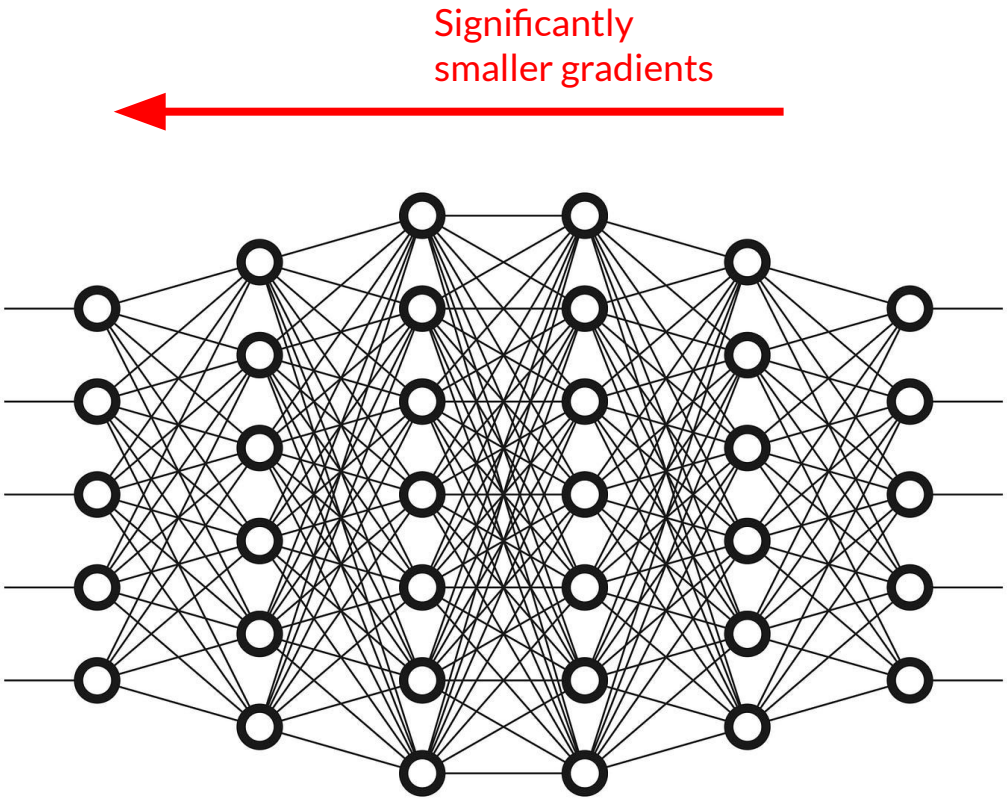
NNs More Concepts

Common gradient problems

- Vanishing gradients.
- Exploding gradients.

Vanishing gradients



Vanishing gradients

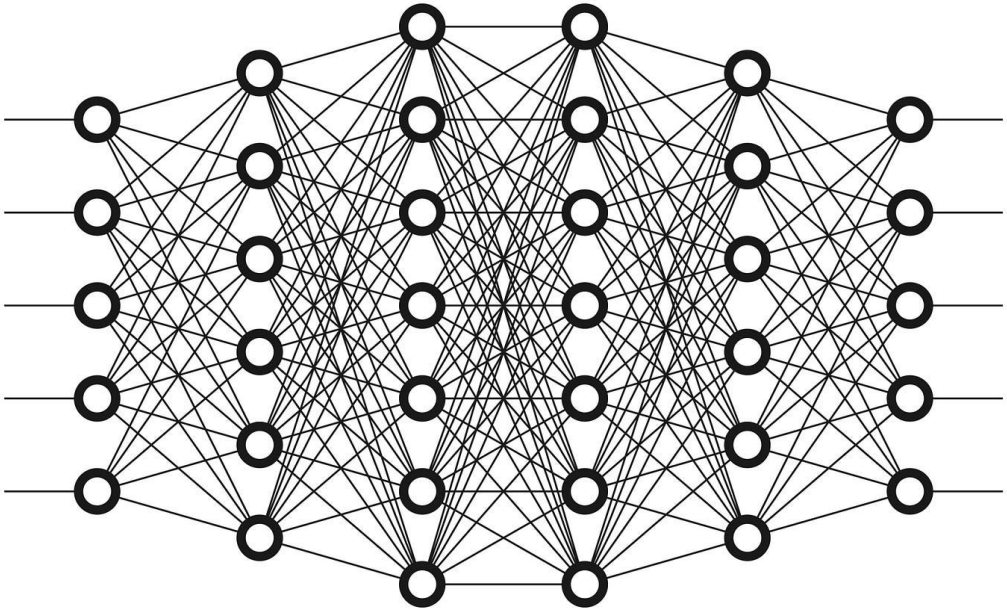
- Symptoms:
  - Learning takes very long or stops completely.
  - Weights closer to the inputs do not change much.
  - Weights shrink exponentially.

Vanishing gradients

- Symptoms:
  - Learning takes very long or stops completely.
  - Weights closer to the inputs do not change much.
  - Weights shrink exponentially.
- Solutions:
  - ReLU or similar.
  - Batch normalization.
  - Gradient clipping.
  - Reduce number of layers.

Exploding gradients

Significantly
larger gradients



Exploding gradients

- Symptoms:
  - Exponential weight growth.
  - Weights overflow (NaN).
  - Weights closer to the inputs are larger.

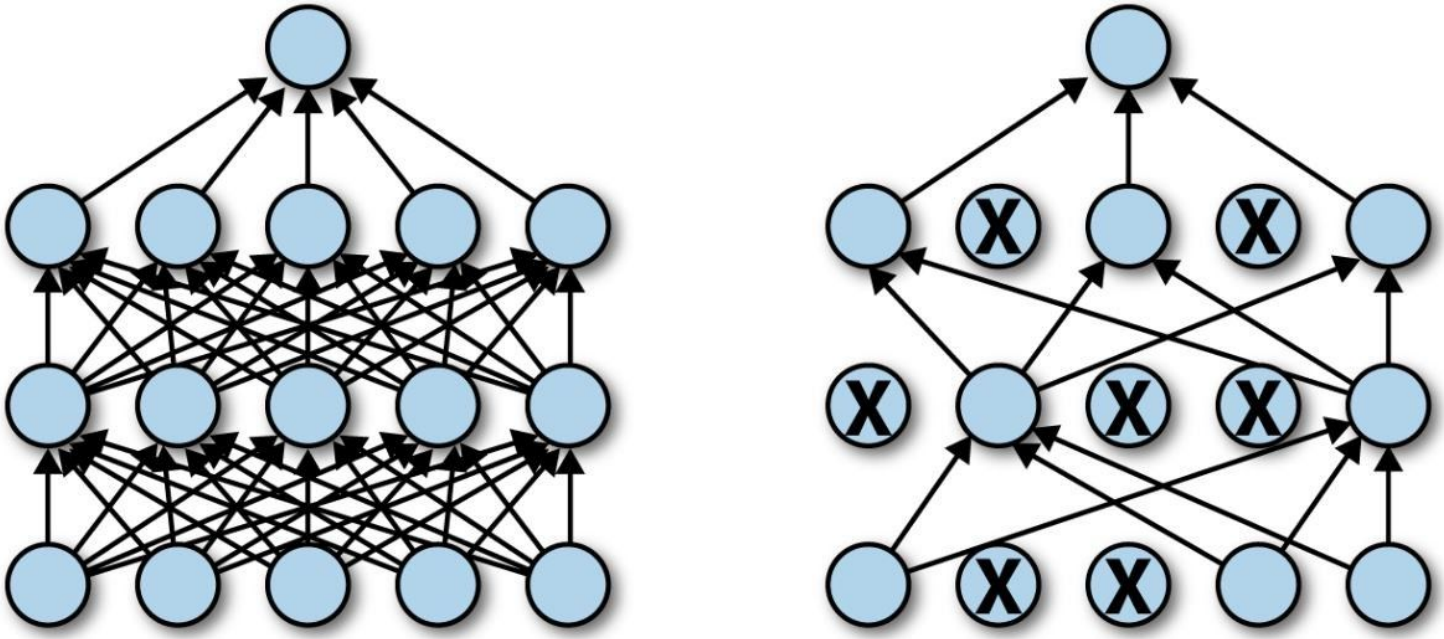
Exploding gradients

- Symptoms:
  - Exponential weight growth.
  - Weights overflow (NaN).
  - Weights closer to the inputs are larger.
- Solutions:
  - Proper weight initialization.
  - Gradient clipping.
  - Reduce the number of layers.

Regularization (for avoiding overfitting)

- Early stopping.
- Weight initialization.
- Dropout.
- Batch normalization.

Dropout



Dropout layer – Keras

```
nn_model = Sequential([
    Input(X_train.shape[1]),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(len(class_names), activation='softmax')
])
```

- Randomly set 20% of parameters to 0 at each mini-batch.
- Applied to the previous Dense layers.

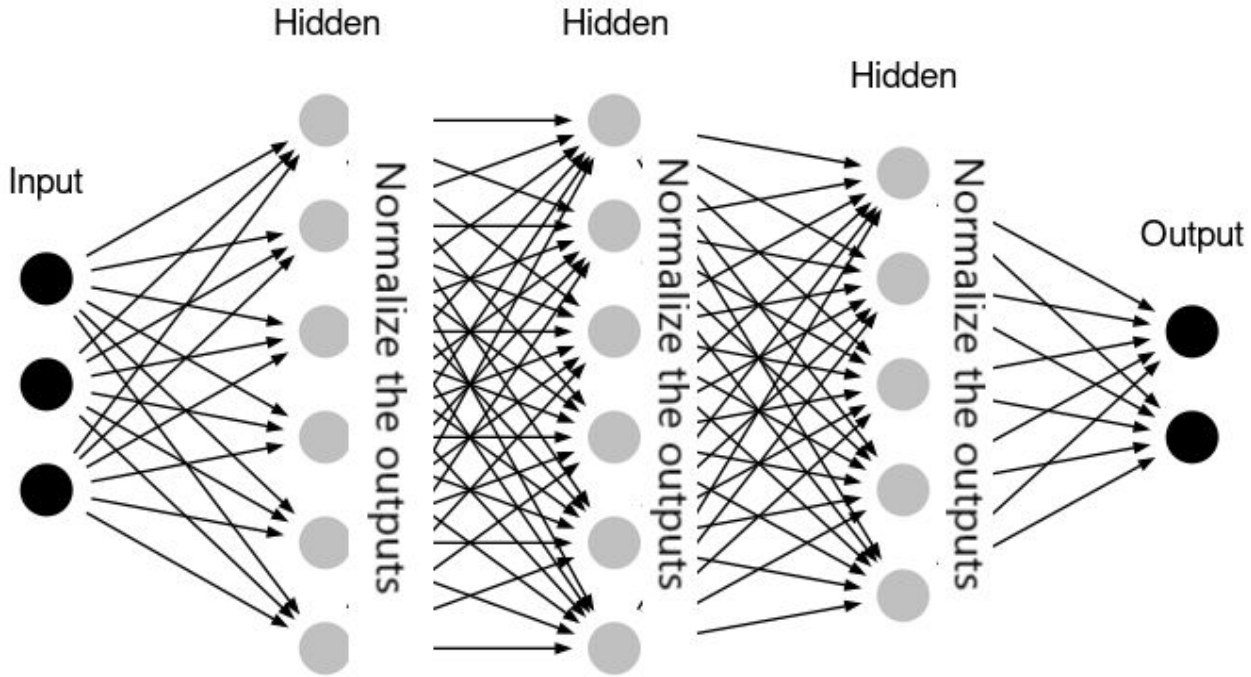
Dropout layer – Keras

```
cnn_model = Sequential([
    Input(shape=(256, 256, 1)),
    Rescaling(1./255),
    Conv2D(filters=16, kernel_size=2, strides=1, activation='relu'),
    Dropout(0.1),
    MaxPooling2D(pool_size=2, strides=None),
    Conv2D(filters=32, kernel_size=2, strides=1, activation='relu'),
    Dropout(0.1),
    MaxPooling2D(pool_size=2, strides=None),
    Flatten(),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(len(class_names), activation='softmax')
])
```

Dropout layer – Recommendations

- Droput rate:
  - Simple NNs: 20% or 30%
  - Complex NNs: 40% or 50%
  - CNNs: 10% or 20%
- Not necessarily used in each layer.

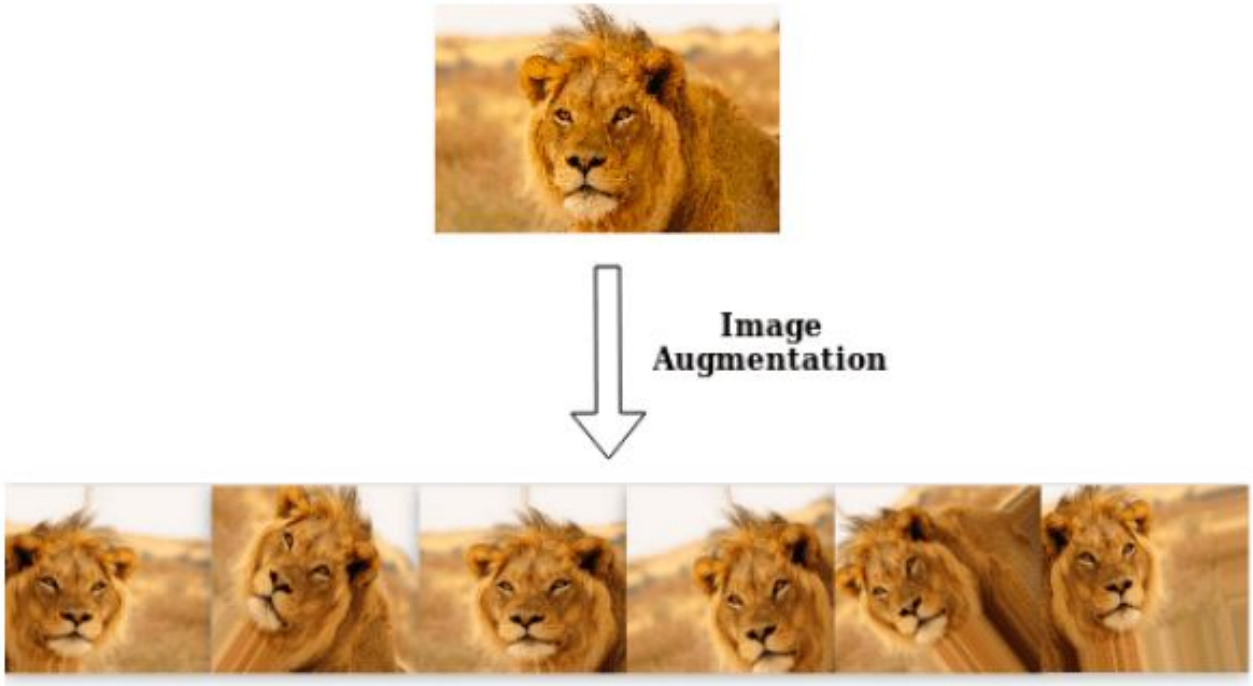
Batch normalization



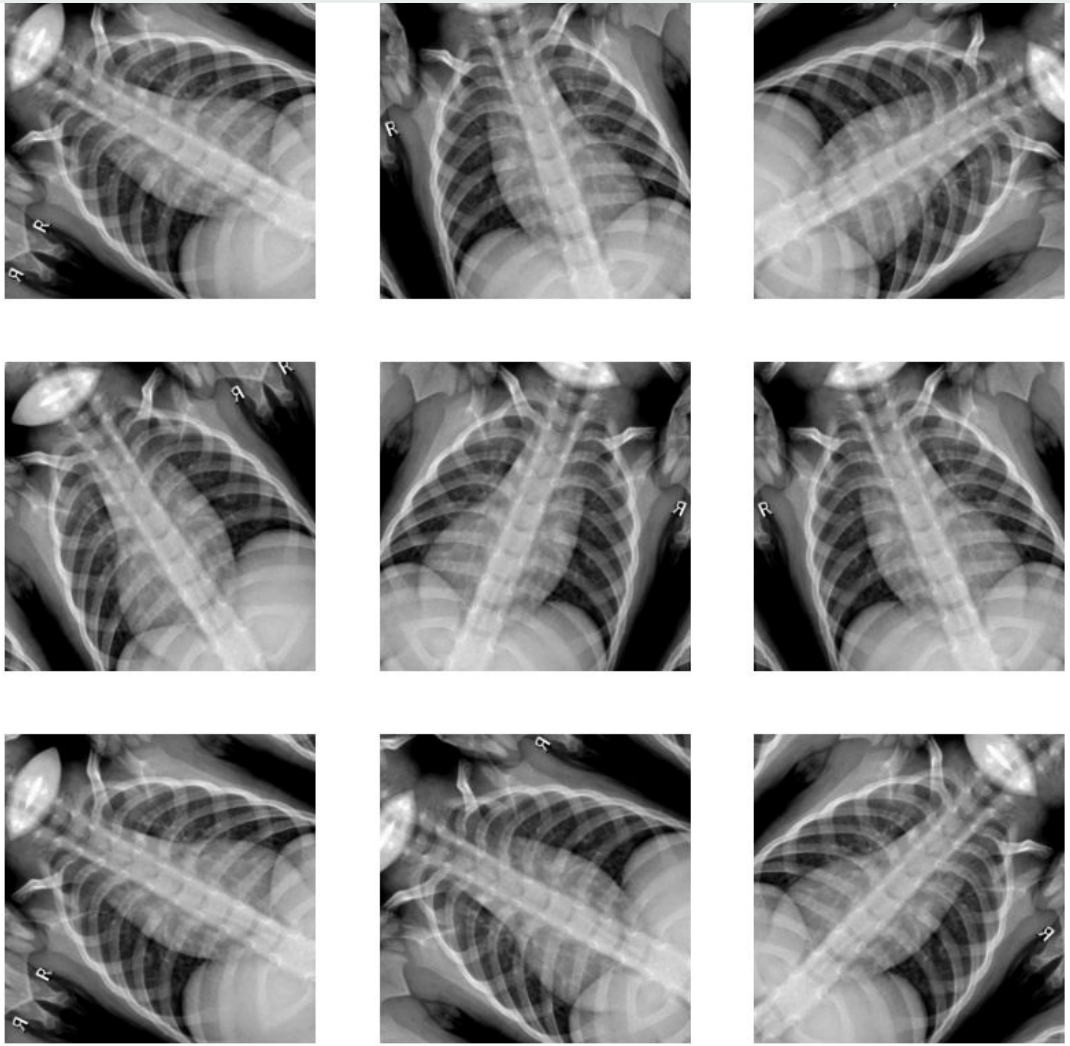
BatchNormalization layer – Recommendations

- Don’t use it when:
  - Small mini-batches.
  - RNNs.

Data augmentation

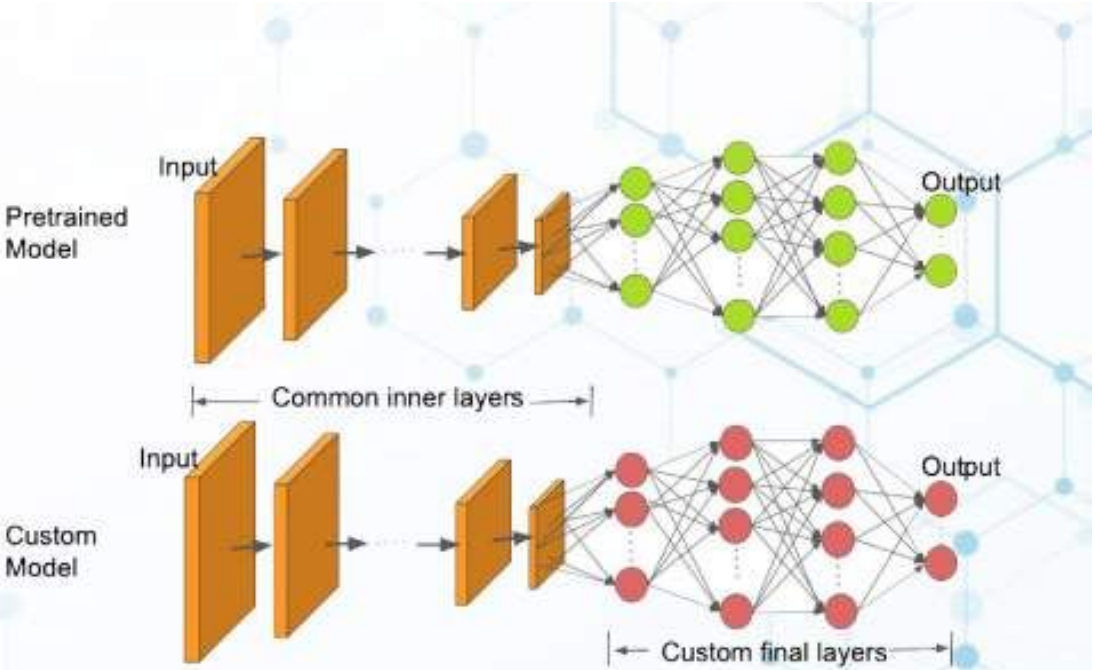


```
cnn_model = Sequential([
    Input(shape=(256, 256, 1)),
    Rescaling(1./255),
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
    Conv2D(filters=16, kernel_size=2, strides=1, activation='relu'),
    MaxPooling2D(pool_size=2, strides=None),
    Flatten(),
    Dense(32, activation='relu'),
    Dense(len(class_names), activation='softmax')
])
```



analytic Steps

Transfer Learning



```
from tensorflow.keras.applications import MobileNetV2
```

```
base_model = MobileNetV2(
    weights='imagenet',
    include_top=False,
    # Not the default size, but we can use it anyway
    input_shape=(256, 256, 3))

for layer in base_model.layers:
    layer.trainable = False
```

```
# Create a new sequential model
cnn_model = Sequential()

# # Resize the images to 224x224
# cnn_model.add(Resizing(224, 224))

cnn_model.add(Input(shape=(256, 256, 1)))

# Rescale the grayscale images
cnn_model.add(Rescaling(1./255))

# Expand the 1 channel input to 3 channels
# to match MobileNetV2 input requirements
cnn_model.add(Conv2D(3, (3, 3)))

# Add all the MobileNetV2 layers
cnn_model.add(base_model)

# Add your custom layers
cnn_model.add(Flatten())
cnn_model.add(Dense(32, activation='relu'))
cnn_model.add(Dense(len(class_names), activation='softmax'))

# Compile the model
cnn_model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```