Computer Vision with Deep Learning

Regularization

Fatma Güney

COMP411/511 - Fall 2024

PLAN

Parameter Penalties

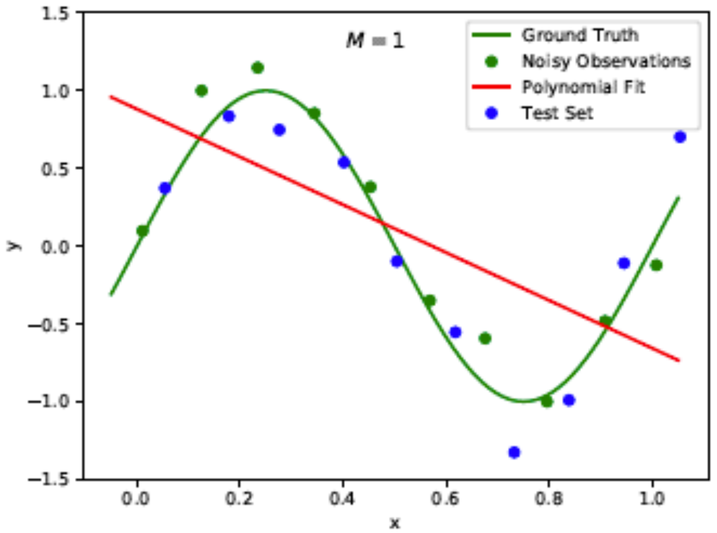
Early Stopping

Ensembe Methods

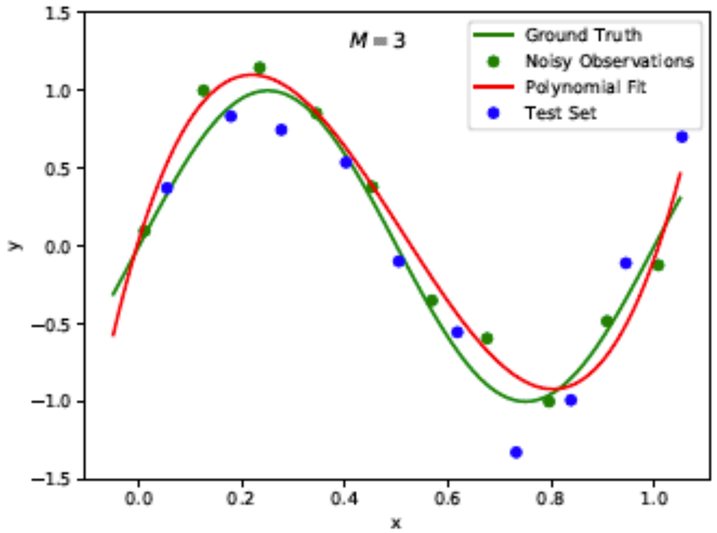
Dropout

Data Augmentation

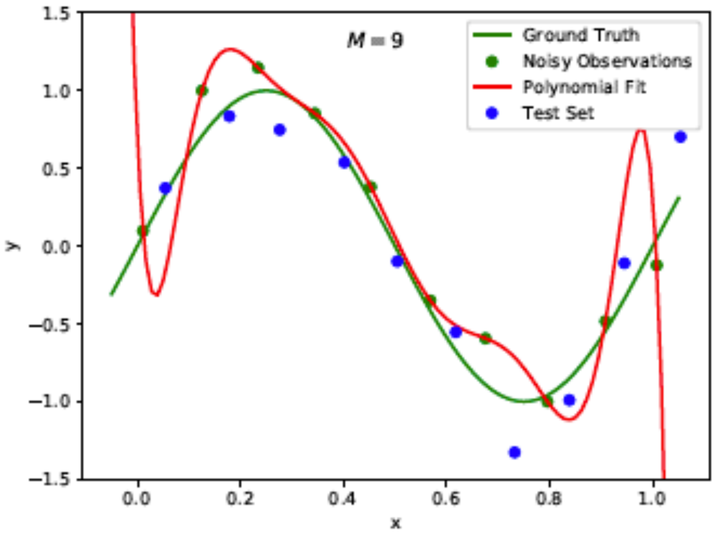
Capacity, Overfitting, and Underfitting



Capacity too low



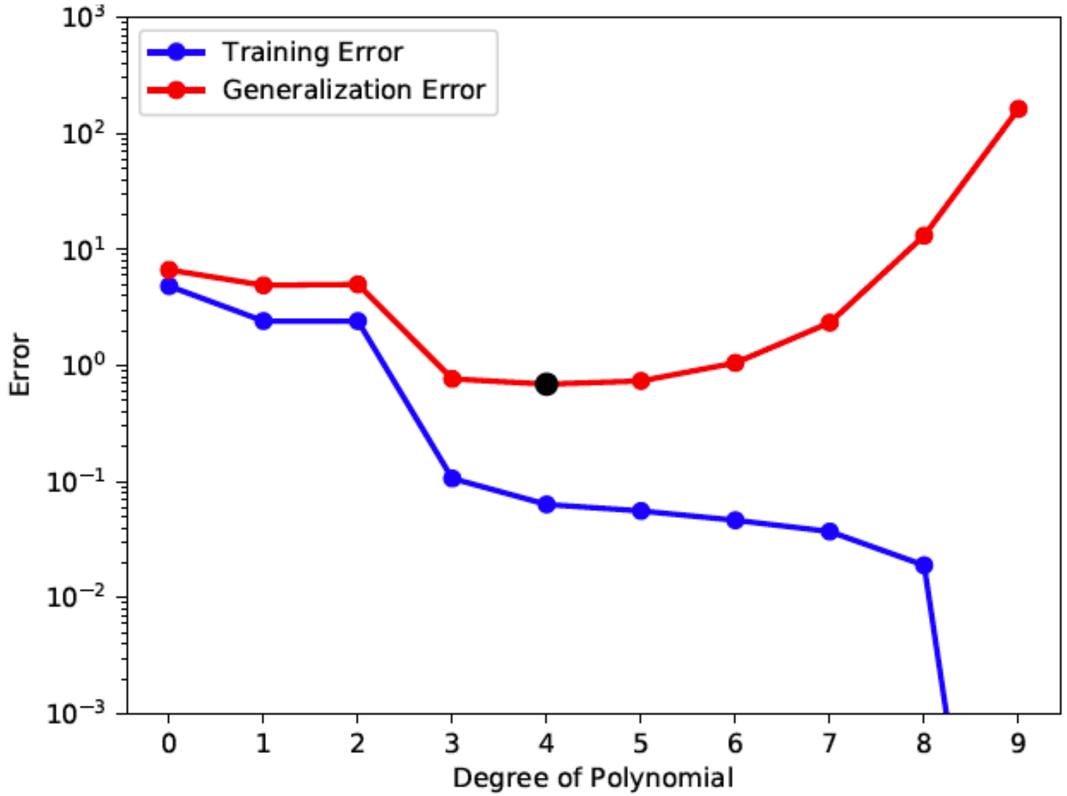
Capacity about right

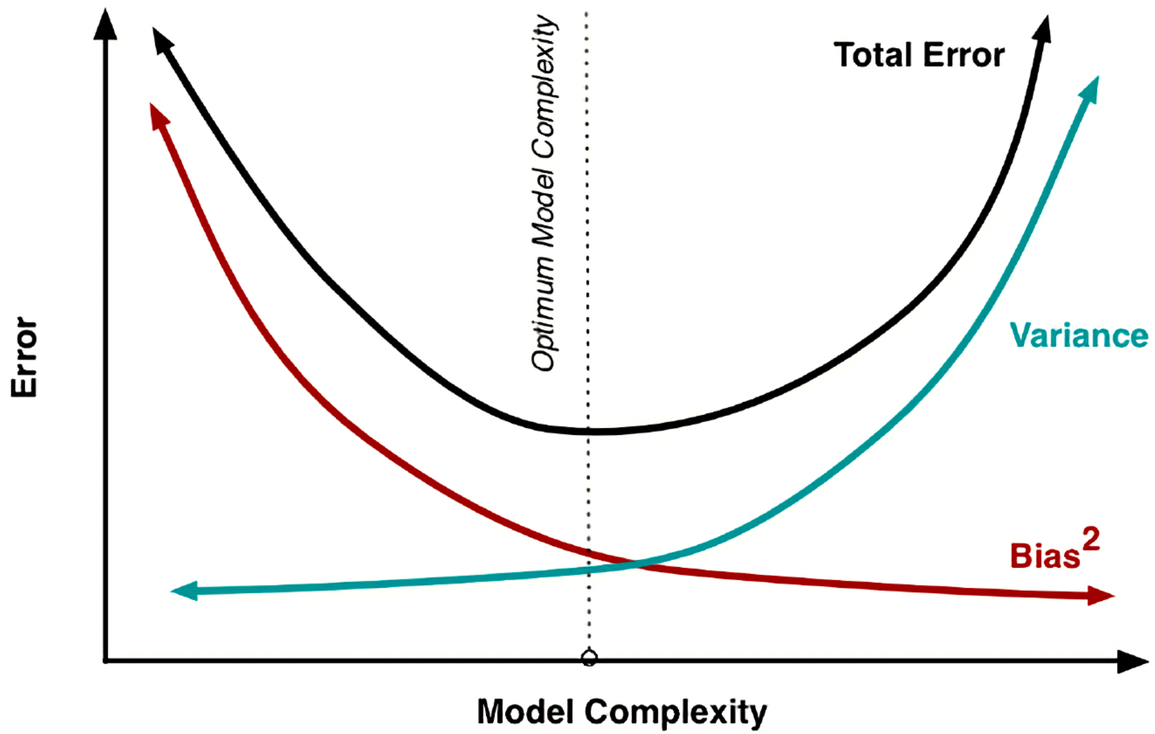


Capacity too large

- **Underfitting**: Model too simple, does not achieve low error on training set
- **Overfitting**: Training error small, but test error (= generalization error) large
- **Regularization**: Take model from third regime (right) to second regime (middle)

Capacity, Overfitting, and Underfitting

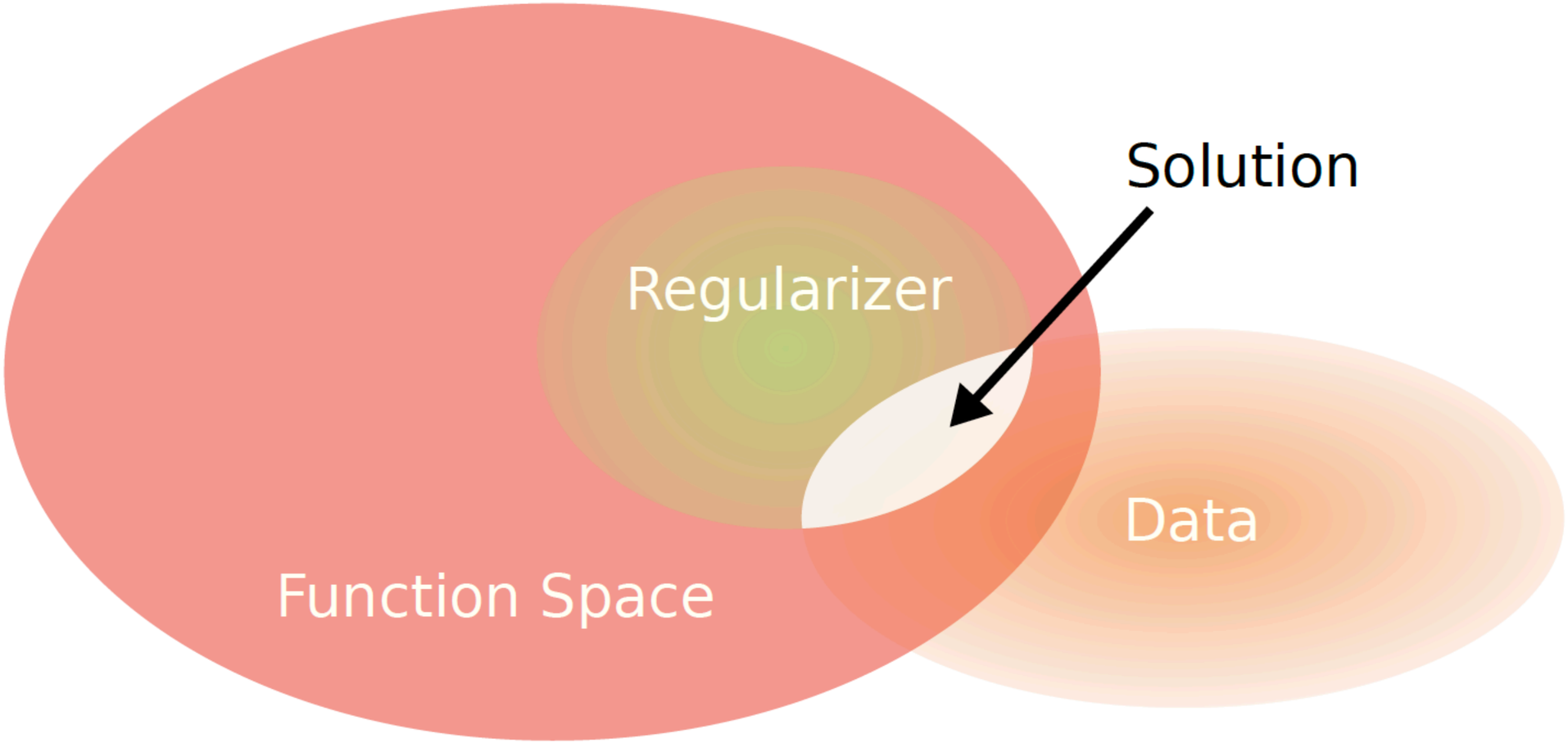




Regularization:

- ✦ Trades **increased bias** for **reduced variance**
- ✦ Goal is to **minimize generalization error** despite using **large model family**

Function Space View



Parameter Penalties

Parameter Penalties

Let $\mathscr{X} = (\mathbf{X}, \mathbf{y})$ denote the dataset and $\mathbf{w}$ the model parameters. We can limit the model capacity by adding a parameter norm penalty $\mathscr{R}$ to the loss $\mathscr{L}$:

$$\underbrace{\tilde{\mathscr{L}}(\mathscr{X}, \mathbf{w})}_{\text{Total Loss}} = \underbrace{\mathscr{L}(\mathscr{X}, \mathbf{w})}_{\text{Original Loss}} + \underbrace{\alpha \mathscr{R}(\mathbf{w})}_{\text{Regularizer}}$$

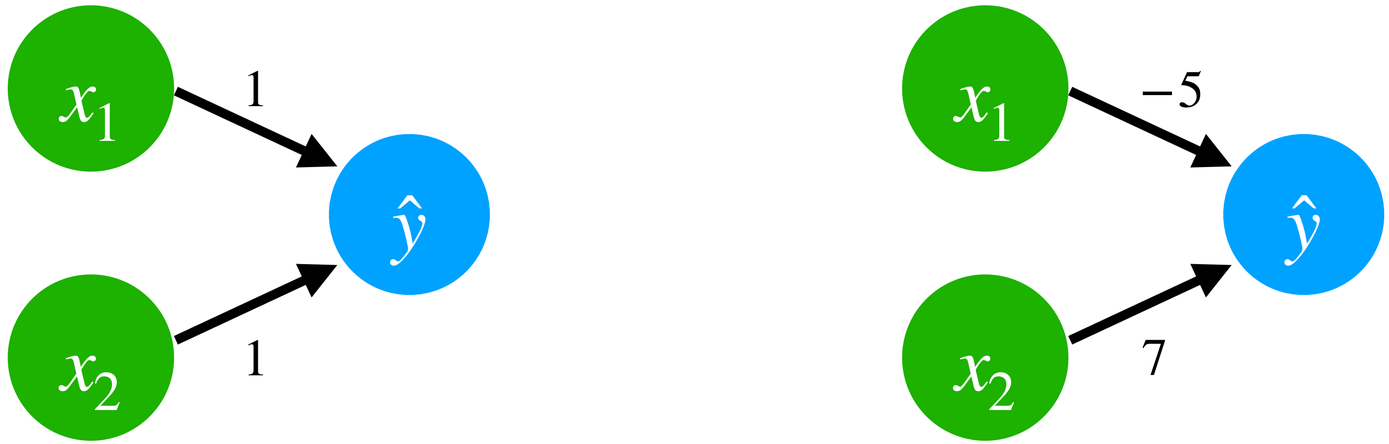
where $\alpha \in [0, \text{inf}]$ controls the strength of the regularizer.

- $\mathscr{R}$ quantifies the size of the parameters / model capacity
- Minimizing $\tilde{\mathscr{L}}$ will decrease both $\mathscr{L}$ and $\mathscr{R}$
- Typically, $\mathscr{R}$ is applied only to the weights (not the bias) of the affine layers
- Often, $\mathscr{R}$ drives weights closer to the origin (in absence of prior knowledge)

Parameter Penalties

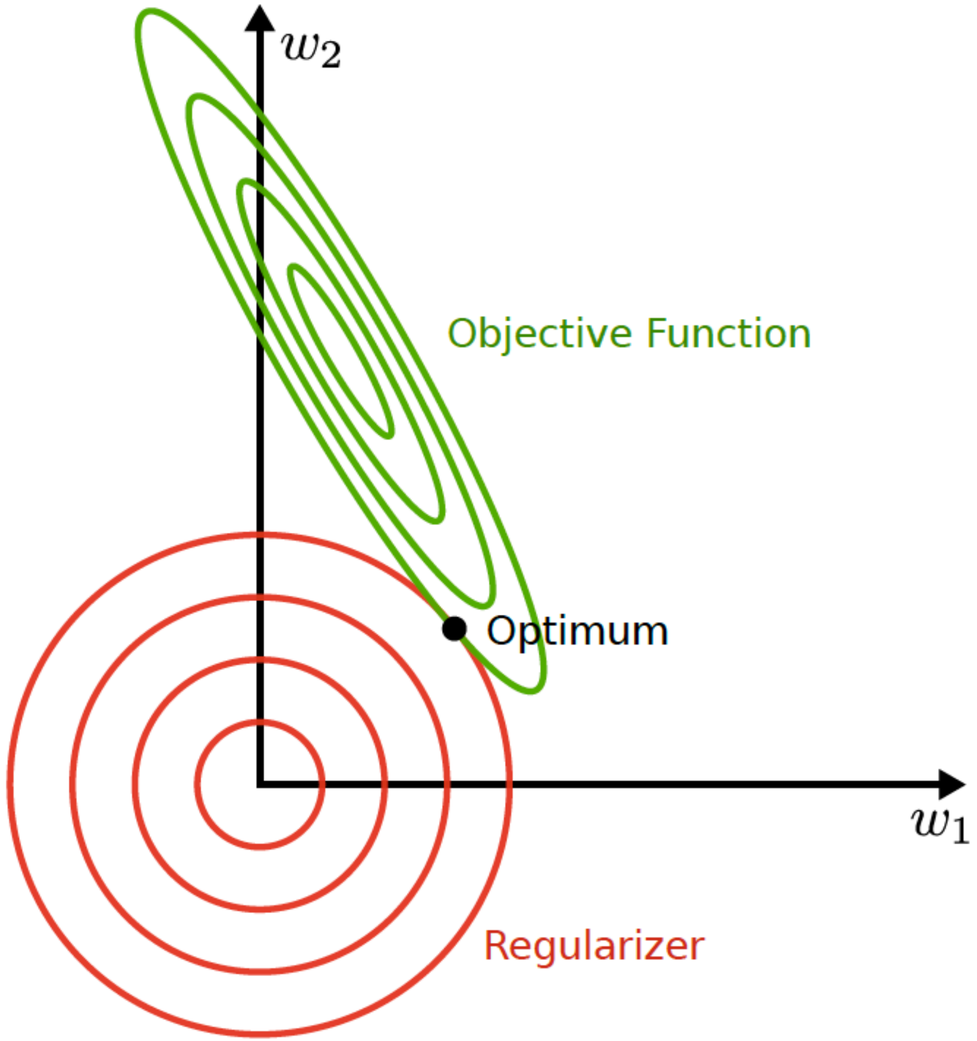
Why do we want the weights/inputs to be small?

- Suppose x_1 and x_2 are nearly identical.
  The following two networks make nearly the **same predictions**:



- But the second network might predict wrongly if the test distribution is slightly different (x_1 and x_2 match less closely) $\Rightarrow$ **Worse generalization**

Parameter Penalties



L_2 Regularization

Weight decay (=ridge regression) uses a (squared) L_2 penalty $\mathscr{R}(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2$:

$$\begin{aligned}\tilde{\mathscr{L}}(\mathscr{X}, \mathbf{w}) &= \mathscr{L}(\mathscr{X}, \mathbf{w}) + \alpha\mathscr{R}(\mathbf{w}) \\ &= \mathscr{L}(\mathscr{X}, \mathbf{w}) + \frac{\alpha}{2}\mathbf{w}^T\mathbf{w}\end{aligned}$$

The **parameter updates** during gradient descent are given by:

$$\begin{aligned}\mathbf{w}^{t+1} &= \mathbf{w}^t - \eta\nabla_{\mathbf{w}}\tilde{\mathscr{L}}(\mathscr{X}, \mathbf{w}^t) \\ &= \mathbf{w}^t - \eta\Big(\nabla_{\mathbf{w}}\mathscr{L}(\mathscr{X}, \mathbf{w}^t) + \alpha\mathbf{w}^t\Big) \\ &= (1 - \eta\alpha)\mathbf{w}^t - \eta\nabla_{\mathbf{w}}\mathscr{L}(\mathscr{X}, \mathbf{w}^t)\end{aligned}$$

Thus, we **decay the weights** at each training iteration before the gradient update.

L_2 Regularization

What happens over the the **entire course** of training? Let $\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathscr{L}(\mathscr{X}, \mathbf{w})$ denote the solution to the unregularized objective and consider a **quadratic approximation** $\hat{\mathscr{L}}$ of the unregularized loss $\mathscr{L}$ around $\mathbf{w}^*$

$$\hat{\mathscr{L}}(\mathscr{X}, \mathbf{w}) = \mathscr{L}(\mathscr{X}, \mathbf{w}^*) + \mathbf{g}^T(\mathbf{w} - \mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T\mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$

$$= \mathscr{L}(\mathscr{X}, \mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T\mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$

with gradient vector $\mathbf{g} = \mathbf{0}$ a semi-positive Hessian matrix $\mathbf{H}$.

When including the **regularization term**, this approximation becomes

$$\hat{\mathscr{L}}(\mathscr{X}, \mathbf{w}) = \mathscr{L}(\mathscr{X}, \mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T\mathbf{H}(\mathbf{w} - \mathbf{w}^*) + \frac{\alpha}{2}\mathbf{w}^T\mathbf{w}$$

L_2 Regularization

When including the **regularization term**, this approximation becomes

$$\hat{\mathscr{L}}(\mathscr{X}, \mathbf{w}) = \mathscr{L}(\mathscr{X}, \mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w} - \mathbf{w}^*) + \frac{\alpha}{2}\mathbf{w}^T\mathbf{w}$$

The minimum $\tilde{\mathbf{w}}$ of the **regularized objective** $\hat{\mathscr{L}}(\mathscr{X}, \mathbf{w})$ is attained at $\nabla_{\mathbf{w}} \hat{\mathscr{L}}(\mathscr{X}, \mathbf{w}) = \mathbf{0}$:

$$\mathbf{H}(\tilde{\mathbf{w}} - \mathbf{w}^*) + \alpha\tilde{\mathbf{w}} = \mathbf{0}$$

$$(\mathbf{H} + \alpha\mathbf{I})\tilde{\mathbf{w}} = \mathbf{H}\mathbf{w}^*$$

$$\tilde{\mathbf{w}} = (\mathbf{H} + \alpha\mathbf{I})^{-1}\mathbf{H}\mathbf{w}^*$$

Thus, as α approaches 0, the regularized solution $\tilde{\mathbf{w}}$ approaches $\mathbf{w}^*$.

L_2 Regularization

What happens if α grows?

Consider the **decomposition** $\mathbf{H} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$ of the **symmetric Hessian matrix** into a diagonal matrix of eigenvalues $\mathbf{\Lambda}$ and an orthonormal basis of eigenvectors $\mathbf{Q}$:

L_2 Regularization

What happens if α grows?

Consider the **decomposition** $\mathbf{H} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$ of the **symmetric Hessian matrix** into a diagonal matrix of eigenvalues $\mathbf{\Lambda}$ and an orthonormal basis of eigenvectors $\mathbf{Q}$:

$$
\begin{aligned}
\tilde{\mathbf{w}} &= (\mathbf{H} + \alpha\mathbf{I})^{-1}\mathbf{H}\mathbf{w}* \\
&= (\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T + \alpha\mathbf{I})^{-1}\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T\mathbf{w}* \\
&= \Big(\mathbf{Q}\big(\mathbf{\Lambda} + \alpha\mathbf{I}\big)\mathbf{Q}^T\Big)^{-1}\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T\mathbf{w}* \\
&= \mathbf{Q}\big(\mathbf{\Lambda} + \alpha\mathbf{I}\big)^{-1}\Lambda\mathbf{Q}^T\mathbf{w}*
\end{aligned}
$$

L_2 Regularization

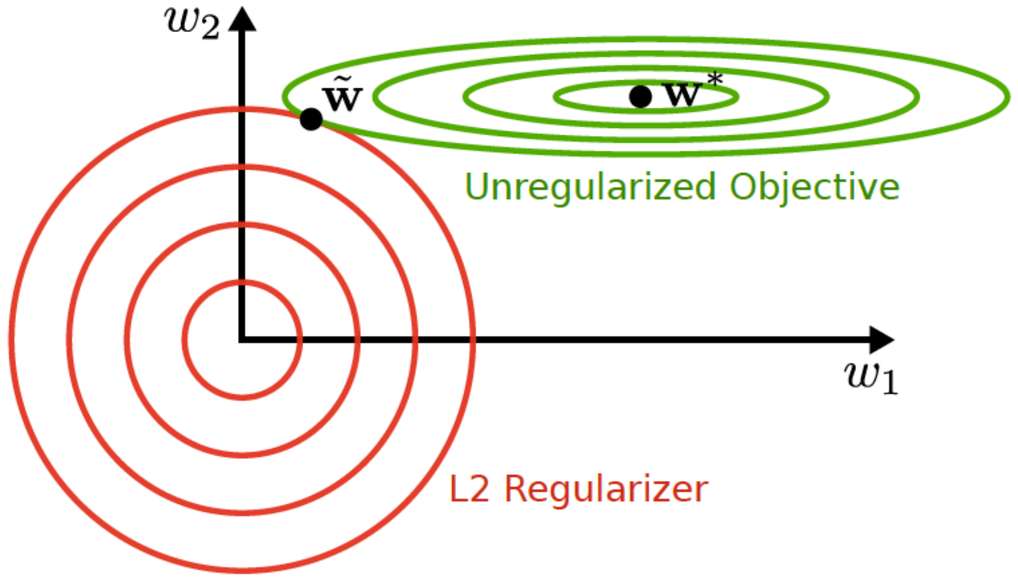
What happens if α grows?

Consider the **decomposition** $\mathbf{H} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$ of the **symmetric Hessian matrix** into a diagonal matrix of eigenvalues $\mathbf{\Lambda}$ and an orthonormal basis of eigenvectors $\mathbf{Q}$:

$$
\begin{aligned}
\tilde{\mathbf{w}} &= (\mathbf{H} + \alpha\mathbf{I})^{-1}\mathbf{H}\mathbf{w}^* \\
&= (\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T + \alpha\mathbf{I})^{-1}\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T\mathbf{w}^* \\
&= \left(\mathbf{Q}\big(\mathbf{\Lambda} + \alpha\mathbf{I}\big)\mathbf{Q}^T\right)^{-1}\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T\mathbf{w}^* \\
&= \mathbf{Q}\big(\mathbf{\Lambda} + \alpha\mathbf{I}\big)^{-1}\Lambda\mathbf{Q}^T\mathbf{w}^*
\end{aligned}
$$

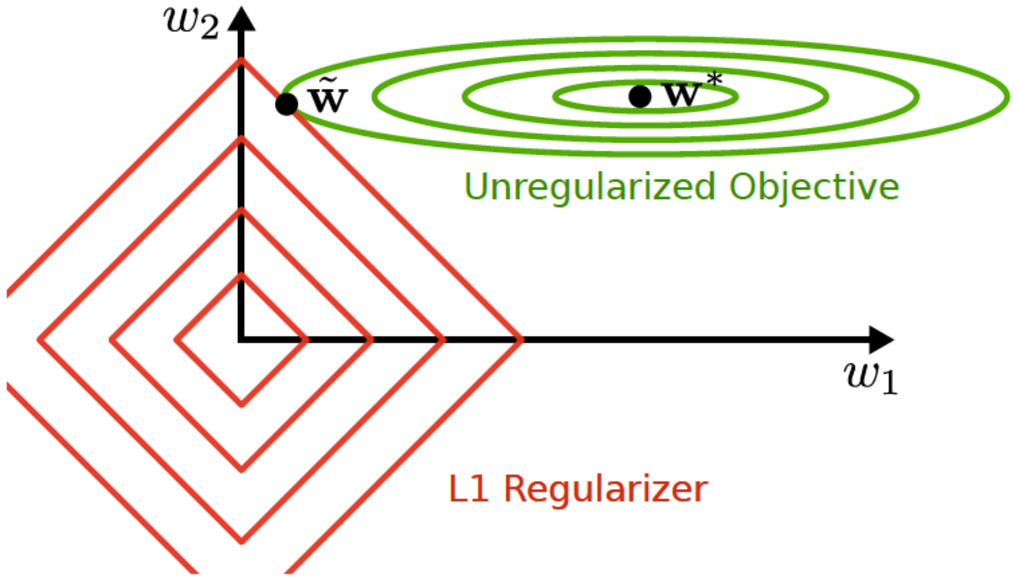
Thus, the component of $\mathbf{w}^*$ that is aligned with the i-th eigenvector of $\mathbf{H}$ is **rescaled** by a factor of $\dfrac{\lambda_i}{\lambda_i + \alpha}$. Regularization affects directions with small eigenvalues $\lambda_i \ll \alpha$.

L_2 Regularization



- Contours of unregularized objective $\mathscr{L}(\mathscr{X}, \mathbf{w})$ and L_2 regularizer $\mathscr{R}(\mathbf{w})$
- At $\tilde{\mathbf{w}}$, the competing objectives **reach an equilibrium** (solution to regularized loss)
- Along w_1, eigenvalue of $\mathbf{H}$ is small (low curvature) ⇒ **strong effect** of regularizer
- Along w_2, eigenvalue of $\mathbf{H}$ is large (high curvature) ⇒ **small effect** of regularizer

L_1 Regularization



- Contours of unregularized objective $\mathscr{L}(\mathscr{X}, \mathbf{w})$ and L_1 regularizer $\mathscr{R}(\mathbf{w})$
- At $\tilde{\mathbf{w}}$, the competing objectives **reach an equilibrium** (solution to regularized loss)
- L_1 regularized loss function: $\tilde{\mathscr{L}}(\mathscr{X}, \mathbf{w}) = \mathscr{L}(\mathscr{X}, \mathbf{w}) + \alpha \|\mathbf{w}\|_1$
- L_1 regularization results in a solution which is more sparse (compared to L_2)

L_2 vs. L_1 Regularization

Example: Assume 3 input features: $\mathbf{x} = (1,2,1)^T$

The following two **linear classifiers** $f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x})$ yield the **same result/loss**:

- $\mathbf{w}_1 = (0,0.75,0)^T \Rightarrow$ ignores 2 features
- $\mathbf{w}_2 = (0.25,0.5,0.25)^T \Rightarrow$ takes all features into account

But the L_1 and L_2 regularizer **prefer different solutions**!

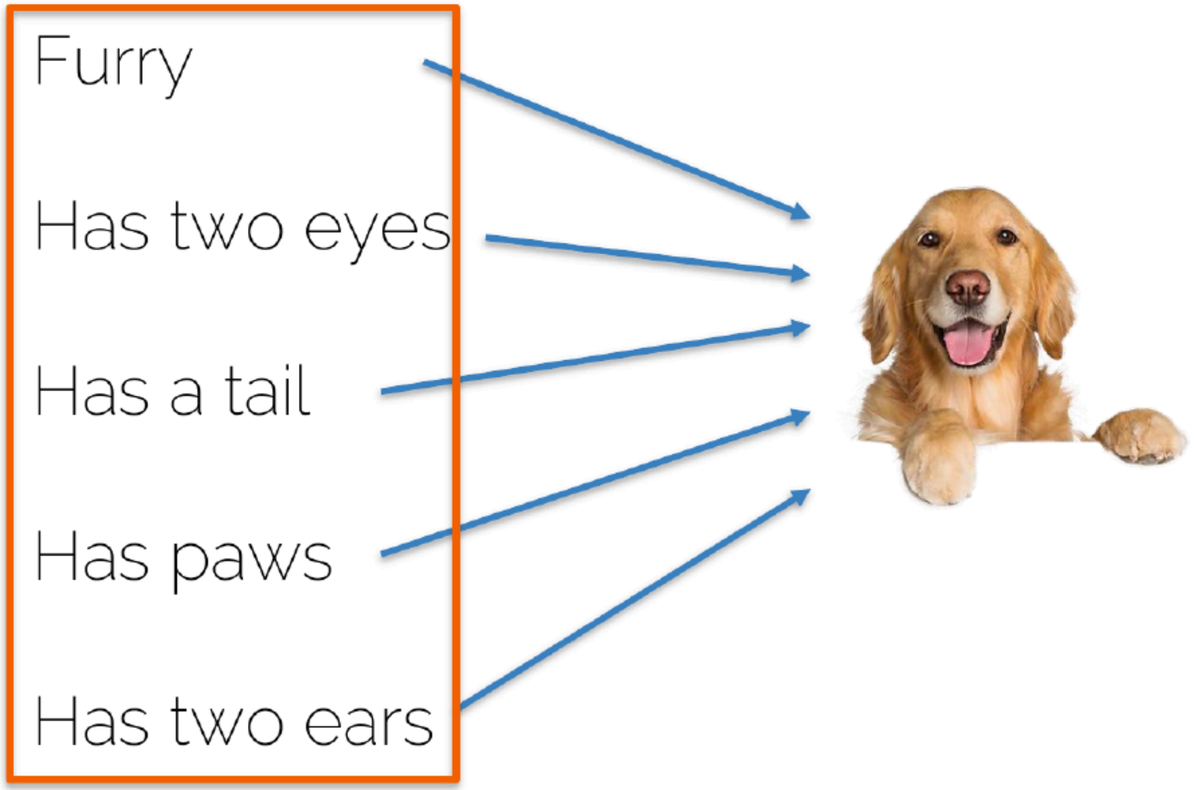
L_2 **Regularization**:

- $\|\mathbf{w}_1\|_2^2 = 0 + 0.75^2 + 0 = 0.5625$
- $\|\mathbf{w}_2\|_2^2 = 0.25^2 + 0.5^2 + 0.25^2 = \mathbf{0.375}$

L_1 **Regularization**:

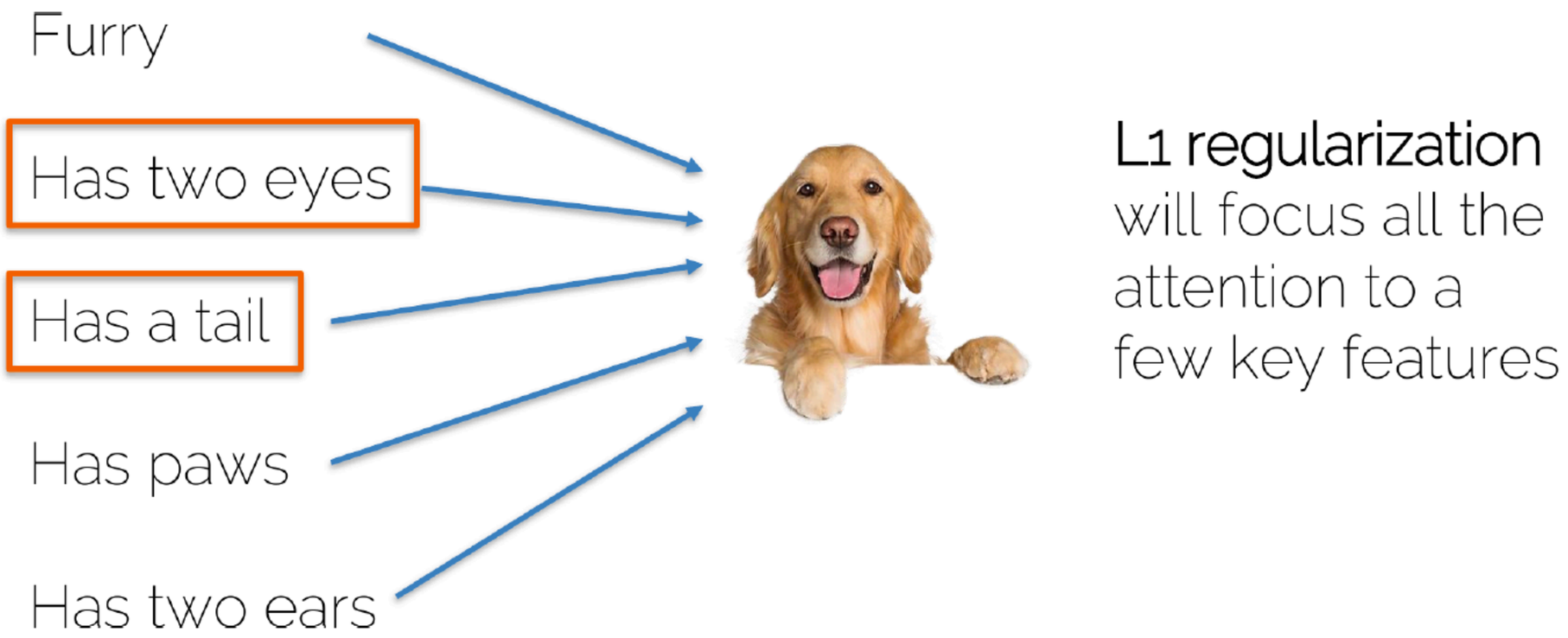
- $\|\mathbf{w}_1\|_1 = 0 + 0.75 + 0 = \mathbf{0.75}$
- $\|\mathbf{w}_2\|_1 = 0.25 + 0.5 + 0.25 = 1$

L_2 vs. L_1 Regularization



L2 regularization will take all information into account to make decisions

L_2 vs. L_1 Regularization

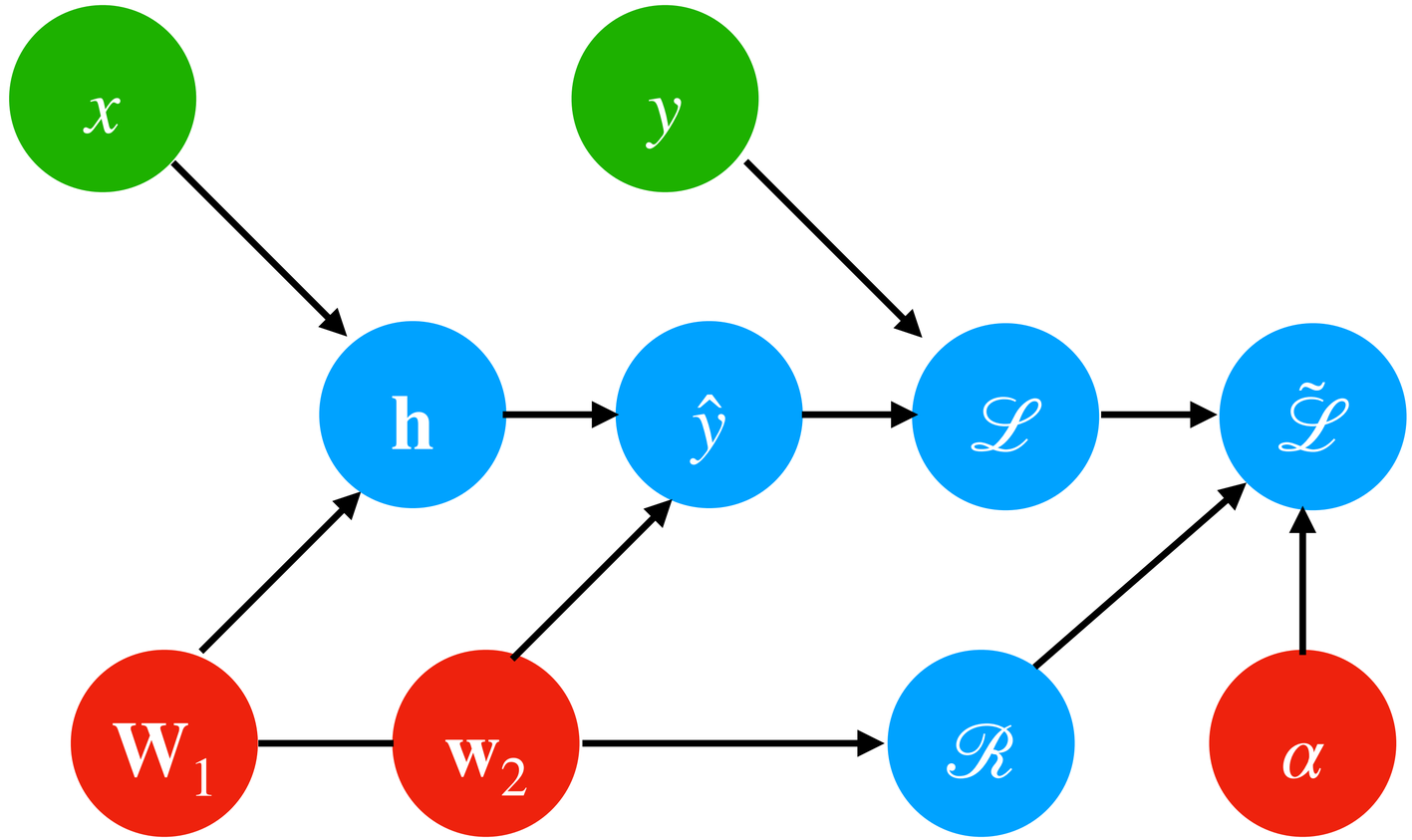


Interpretation as MAP Inference

L_2 regularization can be interpreted as **Bayesian maximum-a-posteriori (MAP) estimation** of the network parameters $\mathbf{w}$ with a Gaussian prior applied to $\mathbf{w}$:

$$
\begin{aligned}
\tilde{\mathbf{w}} &= \arg\max_{\mathbf{w}} p(\mathbf{w} \mid \mathbf{y}, \mathbf{X}) \\
&= \arg\max_{\mathbf{w}} p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) p(\mathbf{w}) \\
&= \arg\max_{\mathbf{w}} \log p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) + \log p(\mathbf{w}) \\
&= \arg\max_{\mathbf{w}} \log p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) + \log \mathcal{N}(\mathbf{w} \mid \alpha^{-1}\mathbf{I}) \\
&= \arg\min_{\mathbf{w}} -\log p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) + \frac{\alpha}{2}\mathbf{w}^T\mathbf{w}
\end{aligned}
$$

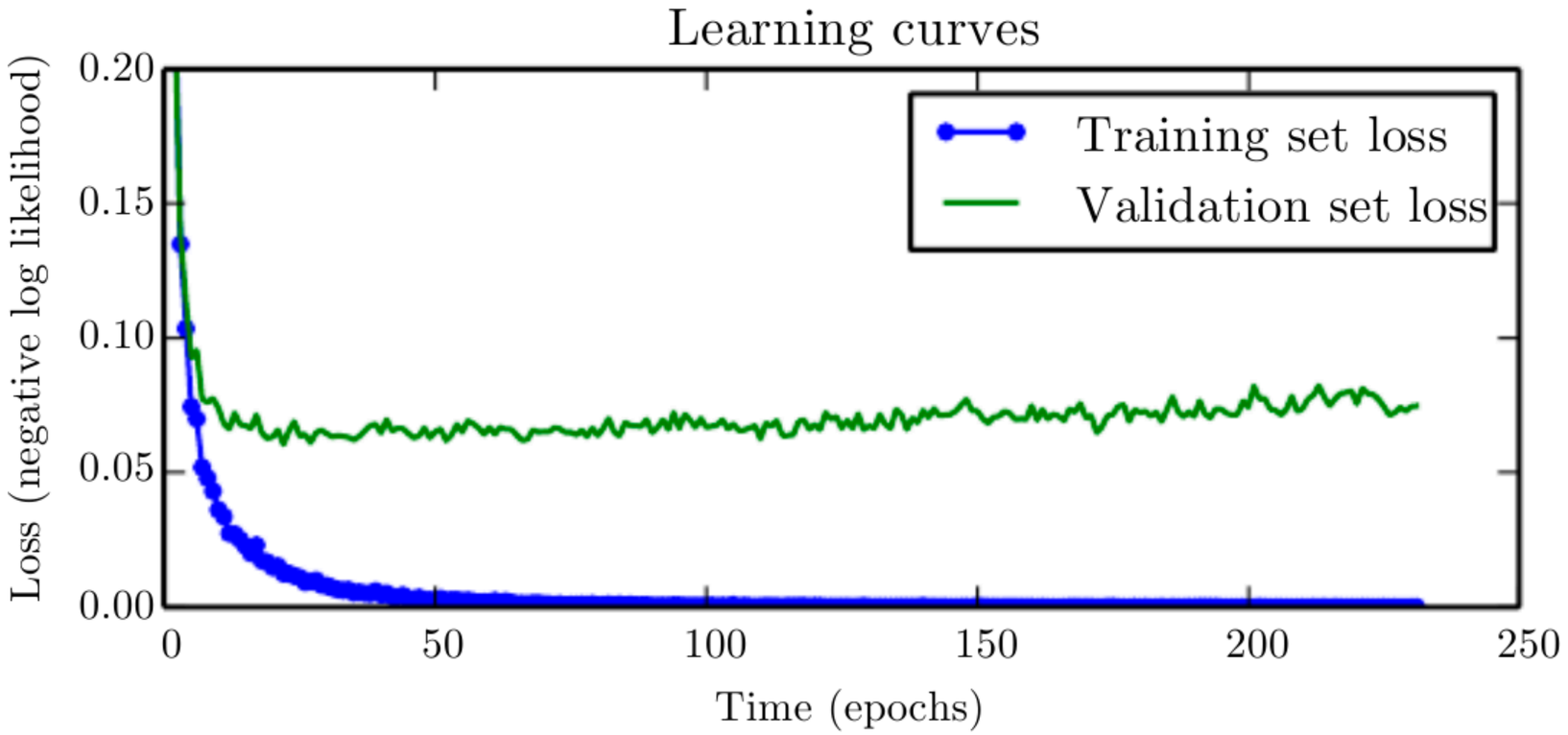
Computation Graph



✦ The combination of loss functions is straight forward (compute nodes)

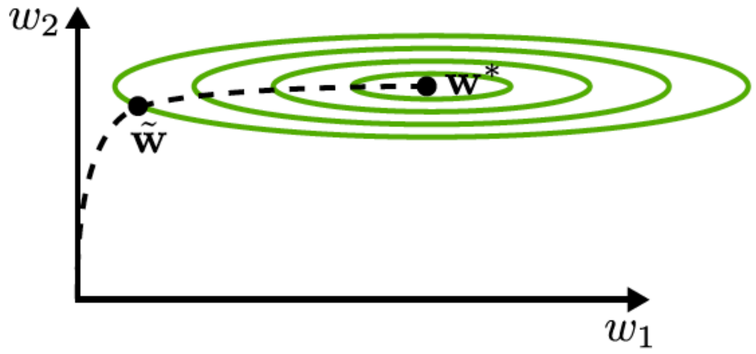
Early Stopping

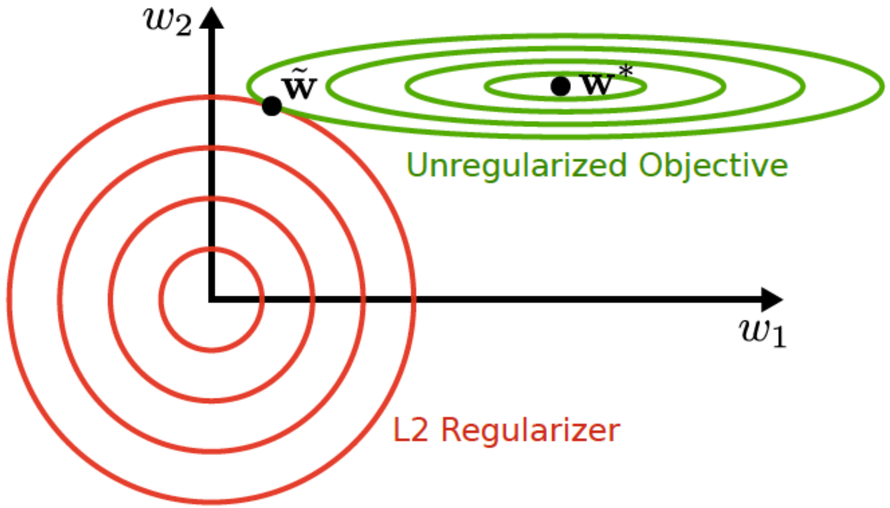
Early Stopping



- While training error decreases over time, validation error starts increasing again
- Thus: train for some time and **return parameters with lowest validation error**

Early Stopping vs. Parameter Penalties





Early Stopping:

- Dashed: Trajectory taken by SGD
- Trajectory stops at $\tilde{\mathbf{w}}$ before reaching the minimum $\mathbf{w}^*$

L_2 Regularization:

- Regularize objective with L_2 penalty
- Penalty forces minimum of regularized loss $\tilde{\mathbf{w}}$ closer to origin

✦ Under some assumptions, both are equivalent (see Chapter 7.8 of text book)

Early Stopping

Early Stopping:

- ✦ Most commonly used form of regularization in deep learning
- ✦ Effective, simple, and computationally efficient form of regularization
- ✦ Training time can be viewed as hyperparameter ⇒ model selection problem
- ✦ Efficient as a single training run tests all hyperparameters (unlike weight decay)
- ✦ Only cost: periodically evaluate validation error on validation set
- ✦ Validation set can be small, and evaluation less frequently

Remark: If little training data is available, one can perform a second training phase where the model is retrained from scratch on all training data using the same number of training iterations determined by the early stopping procedure

Ensemble Methods

Ensemble Methods

Idea:

- ✦ Train several models separately for the same task
- ✦ At inference time: average results
- ✦ Thus, often also called “model averaging”

Intuition:

- ✦ Different models make different errors on the test set
- ✦ By averaging, we obtain a more robust estimate without a better model!
- ✦ Works best if models are maximally uncorrelated
- ✦ Winning entries of challenges are often ensembles
- ✦ Drawback: requires evaluation of multiple models at inference time

Ensemble Methods

Consider K regression models, each of which has an error of ϵ_k with variances $\mathscr{E}[\epsilon_k^2] = v$ and covariances $\mathscr{E}[\epsilon_k \epsilon_l] = c$. The **expected square error of the ensemble predictor** (with each model having the same weight) is given as:

$$
\begin{aligned}
\mathscr{E}\left[\left(\frac{1}{K}\sum_k \epsilon_k\right)^2\right] &= \frac{1}{K^2}\mathscr{E}\left[\sum_k \left(\epsilon_k^2 + \sum_{l\neq k} \epsilon_k \epsilon_l\right)\right] \\
&= \frac{1}{K^2}\left(\sum_k \mathscr{E}\left[\epsilon_k^2\right] + \sum_k \sum_{l\neq k} \mathscr{E}\left[\epsilon_k \epsilon_l\right]\right) \\
&= \frac{1}{K^2}\left(Kv + K(K-1)c\right) \\
&= \frac{1}{K}v + \frac{K-1}{K}c
\end{aligned}
$$

Ensemble Methods

Consider K regression models, each of which has an error of $\epsilon_k \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$ with variances $\mathscr{E}[\epsilon_k^2] = v$ and covariances $\mathscr{E}[\epsilon_k \epsilon_l] = c$. The **ensemble error** is given by:

$$\mathscr{E}\left[\left(\frac{1}{K}\sum_k \epsilon_k\right)^2\right] = \frac{1}{K}v + \frac{K-1}{K}c$$

- If errors are correlated ($c = v$), the ensemble error becomes $v \Rightarrow$ no gain
- If errors are uncorrelated ($c = 0$), the ensemble error reduces to $\frac{1}{K}v$

Thus:

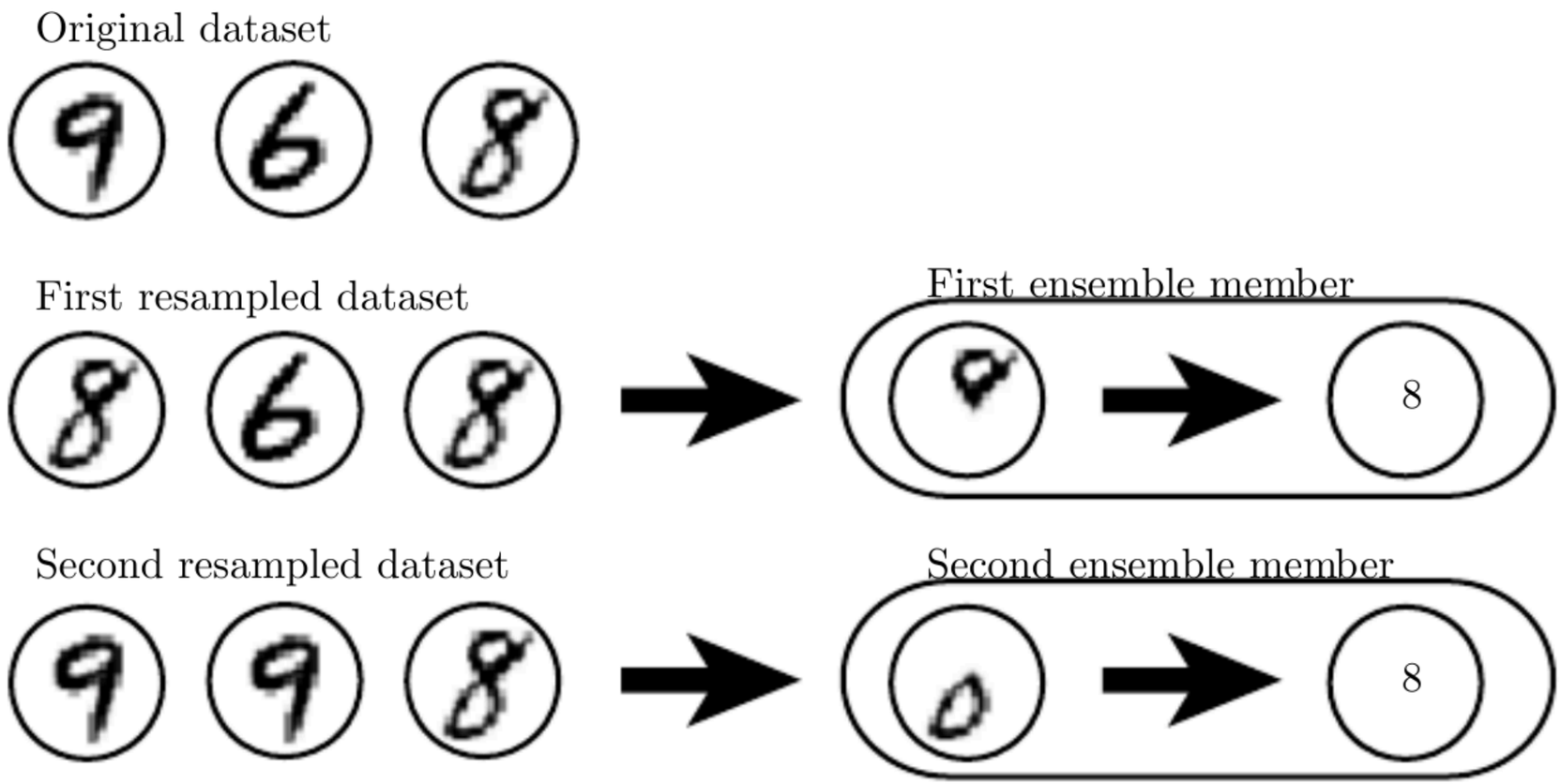
- **Ensemble maximally effective if errors maximally uncorrelated**

Ensemble Methods

Different Types of Ensemble Methods:

- **Initialization**: Train networks starting from different random initialization on same dataset or using different minibatches (via stochastic gradient descent). This often already introduces some independence.
- **Model**: Use different models, architectures, losses or hyperparameters
- **Bagging**: Train networks on different random draws (with replacement) from the original dataset. Thus, each dataset likely misses some of the examples from the original dataset and contains some duplicates.

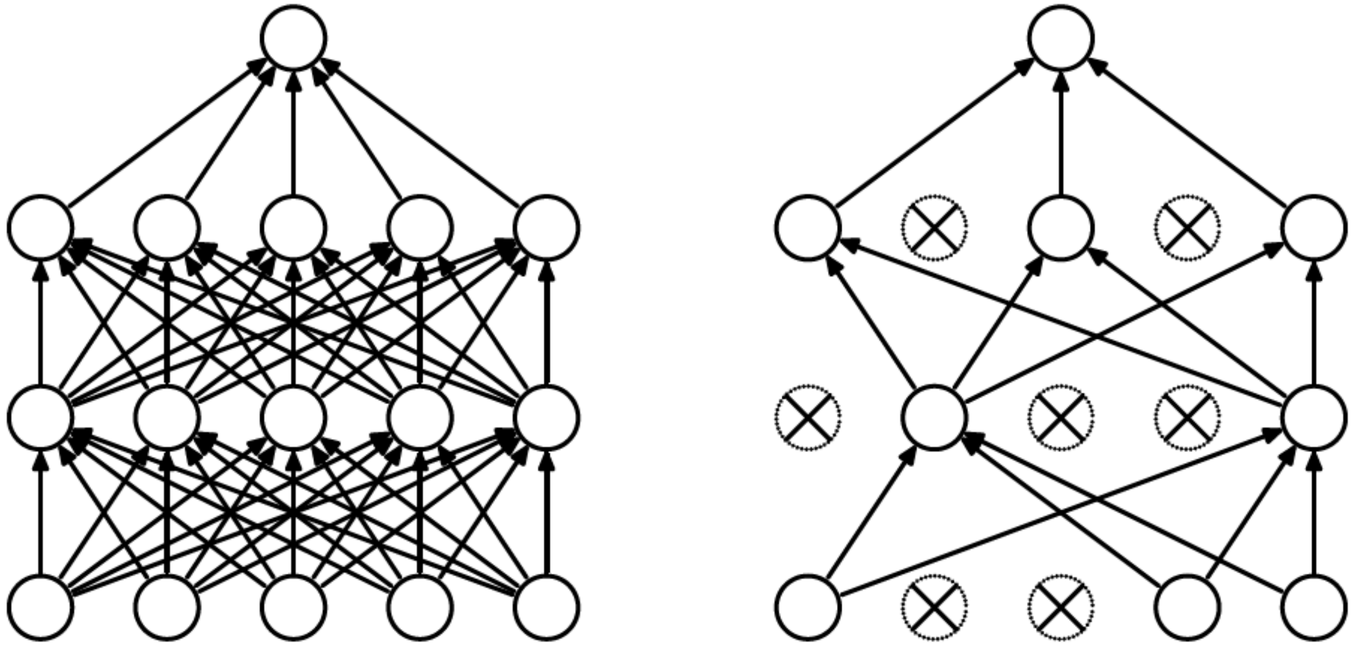
Bagging Example



- First model learns to detect top “loop”, second model detects bottom “loop”

Dropout

Dropout

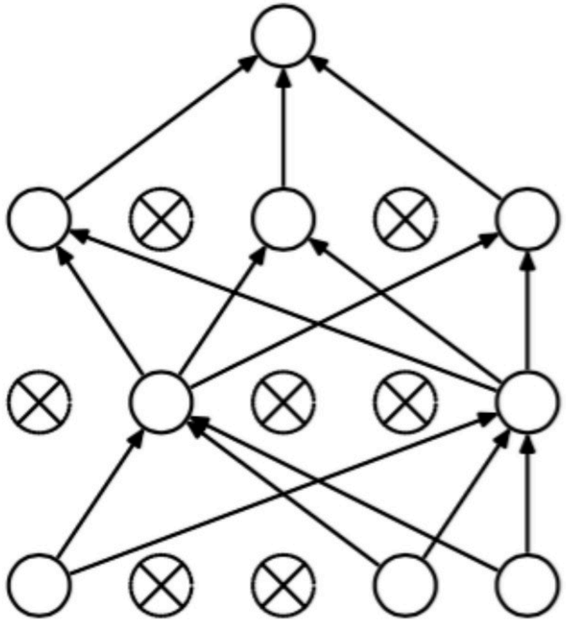


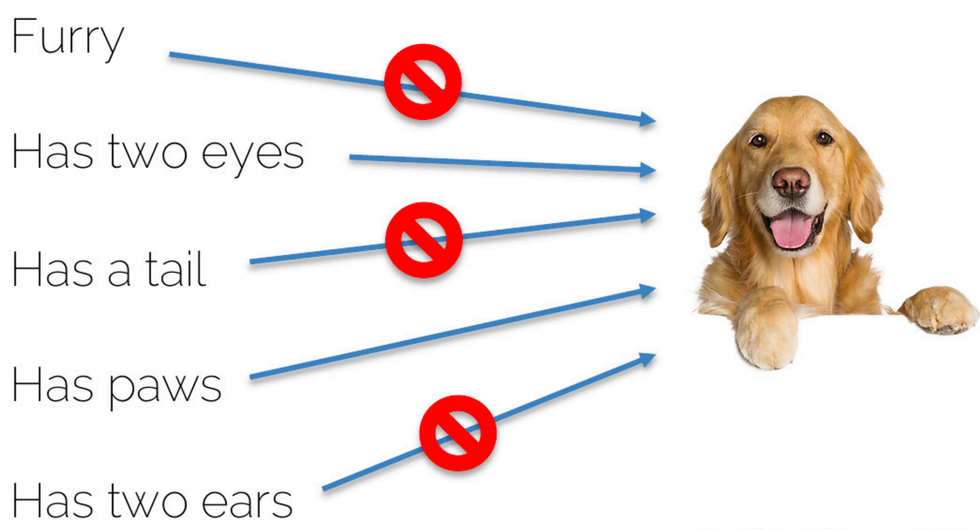
Idea:

- During training, **set neurons to zero** with probability μ (typically $\mu = 0.5$)
- Each binary mask is one model, changes randomly with every training iteration
- Creates **ensemble "on the fly"** from a single network with shared parameters

Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov: Dropout: a simple way to prevent neural networks from overfitting. JMLR, 2014.

Dropout





[Leal-Taixe & Niessner, I2DL]

Why is this a good idea?

- Forces the network to learn a **redundant representation** ⇒ regularization
- Reduces effective **model capacity** ⇒ larger models, longer training
- Prevents **co-adaptation** of features (units can't learn to undo output of others)
- Requires only **one forward pass at inference time** ⇒ Why?

Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov: Dropout: a simple way to prevent neural networks from overfitting. JMLR, 2014.

Dropout

Inference:

- Dropout makes the output random. Formally, we have:

$$\hat{y}_{\mathbf{z}} = f_{\mathbf{w}}(\mathbf{x}, \mathbf{z})$$

Here, $\mathbf{z}$ is a binary mask with one element per unit drawn i.i.d. from a Bernoulli $p(z_i) = \mu^{1-z_i}(1-\mu)^{z_i}$ where $z_i = 0$ if neuron i is removed from the network

- At inference time, we want to calculate the **ensemble prediction**:

$$\hat{y}_{\mathbf{z}} = f_{\mathbf{w}}(\mathbf{x}) = \mathcal{E}_{\mathbf{z}}\left[f_{\mathbf{w}}(\mathbf{x}, \mathbf{z})\right] = \sum_{\mathbf{z}} p(\mathbf{z}) f_{\mathbf{w}}(\mathbf{x}, \mathbf{z})$$

- For M neurons, we have 2^M terms $\Rightarrow$ this is not tractable

Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov: Dropout: a simple way to prevent neural networks from overfitting. JMLR, 2014.

Dropout

Let us consider a simple **linear model:**

$$f_{\mathbf{w}}(\mathbf{x}) = w_1 x_1 + w_2 x_2$$

$$f_{\mathbf{w}}(\mathbf{x}, \mathbf{z}) = z_1 w_1 x_1 + z_2 w_2 x_2$$

Assuming $\mu = 0.5$, during training we optimize the **expectation over the ensemble**:

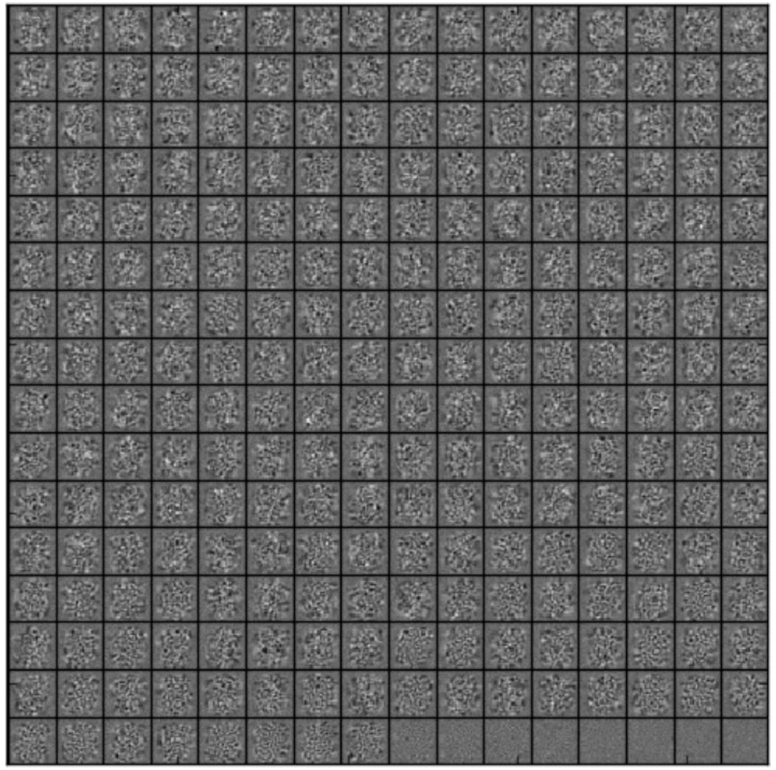
$$\begin{aligned}\mathcal{E}_{\mathbf{z}}\left[f_{\mathbf{w}}(\mathbf{x}, \mathbf{z})\right] &= \frac{1}{4}(0 + 0) + \frac{1}{4}(w_1 x_1 + 0) + \frac{1}{4}(0 + w_2 x_2) + \frac{1}{4}(w_1 x_1 + w_2 x_2) \\ &= \frac{1}{2}(w_1 x_1 + w_2 x_2) = \frac{1}{2} f_{\mathbf{w}}(\mathbf{x})\end{aligned}$$

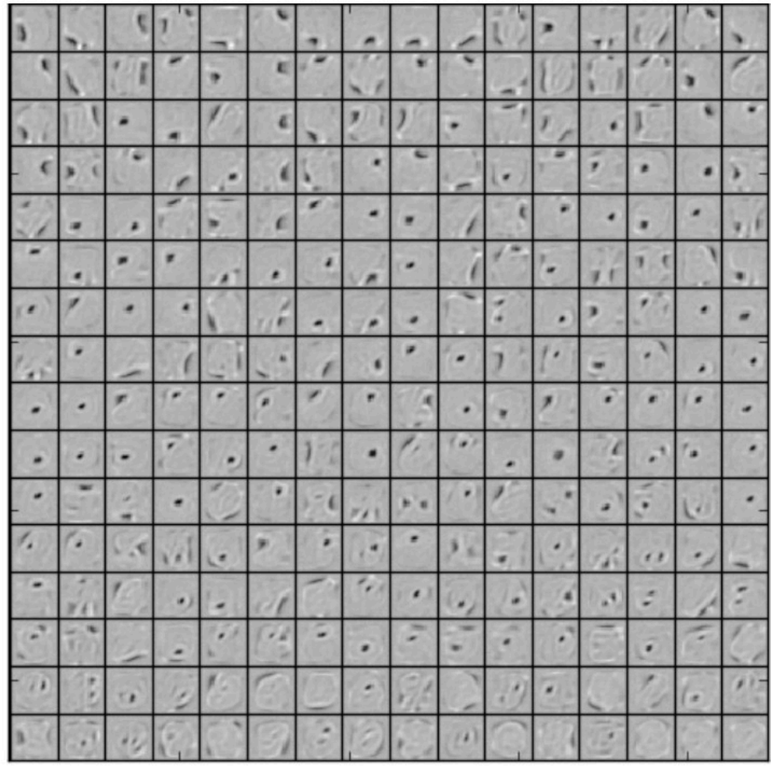
Thus, at test time, we must **multiply the trained weights** by $1 - \mu$.

Remark: This weight scaling inference is only an approximation for non-linear models.

Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov: Dropout: a simple way to prevent neural networks from overfitting. JMLR, 2014.

Dropout





- Features of an Autoencoder on MNIST with a single hidden layer of 256 ReLUs
- Right: With dropout ⇒ less co-adaptation and thus better generalization

Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov: Dropout: a simple way to prevent neural networks from overfitting. JMLR, 2014.

Dropout

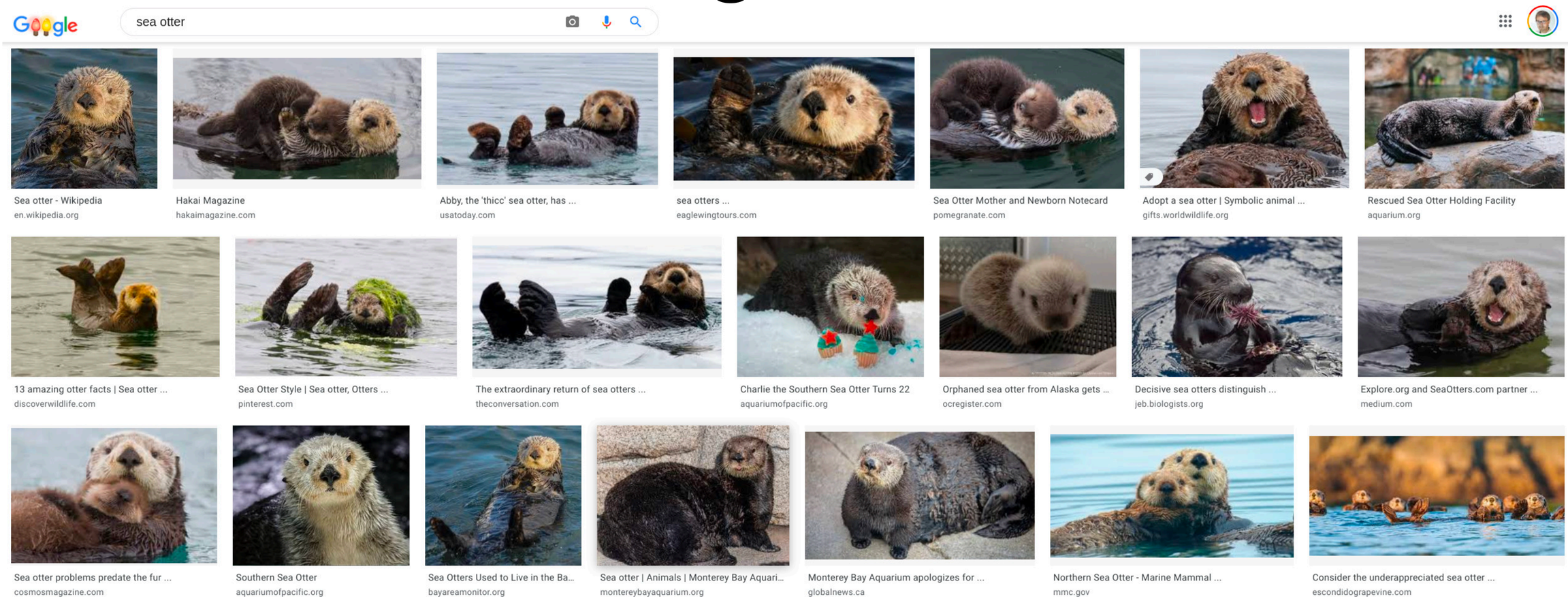
Method	Test Classification error %
L2	1.62
L2 + L1 applied towards the end of training	1.60
L2 + KL-sparsity	1.55
Max-norm	1.35
Dropout + L2	1.25
Dropout + Max-norm	**1.05**

Table 9: Comparison of different regularization methods on MNIST.

Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov: Dropout: a simple way to prevent neural networks from overfitting. JMLR, 2014.

Data Augmentation

Data Augmentation

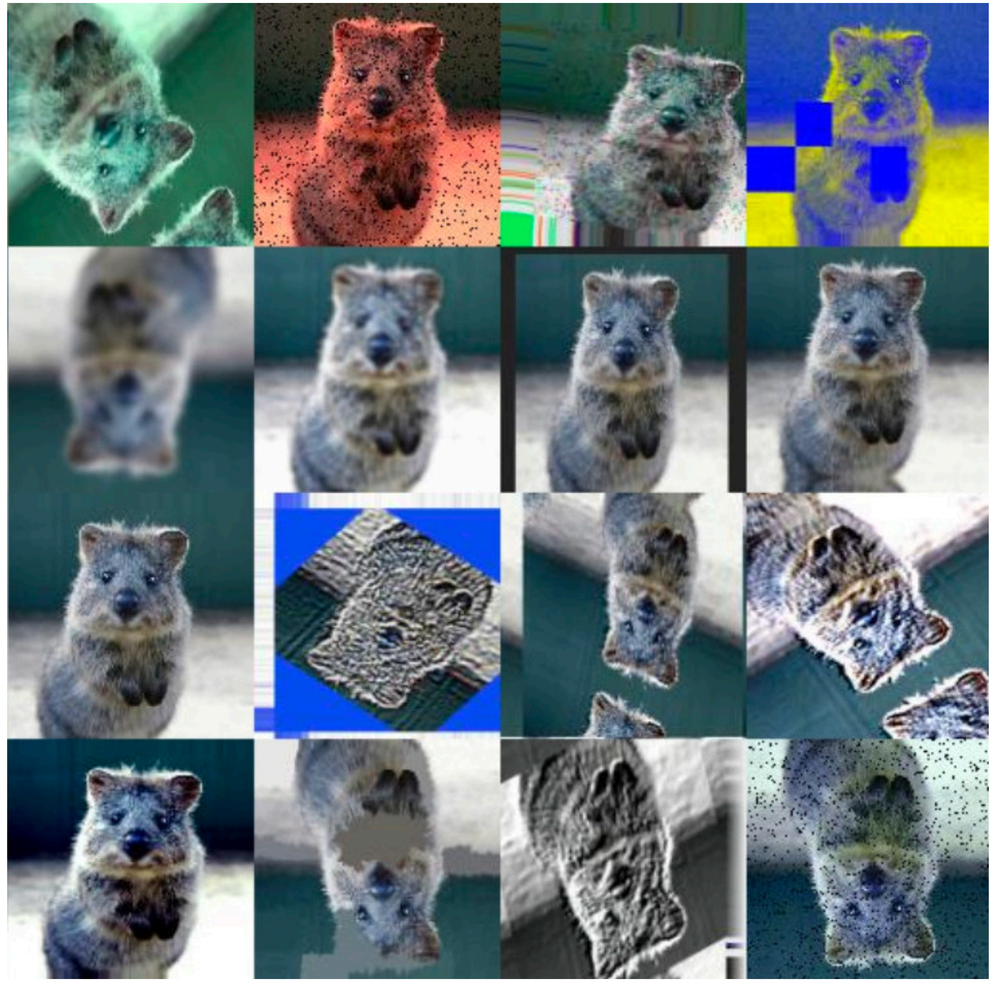


Motivation:

- ✦ Deep neural networks must be invariant to a wide variety of input variations
- ✦ Often large intra-class variation in terms of pose, appearance, lighting, etc.

Data Augmentation

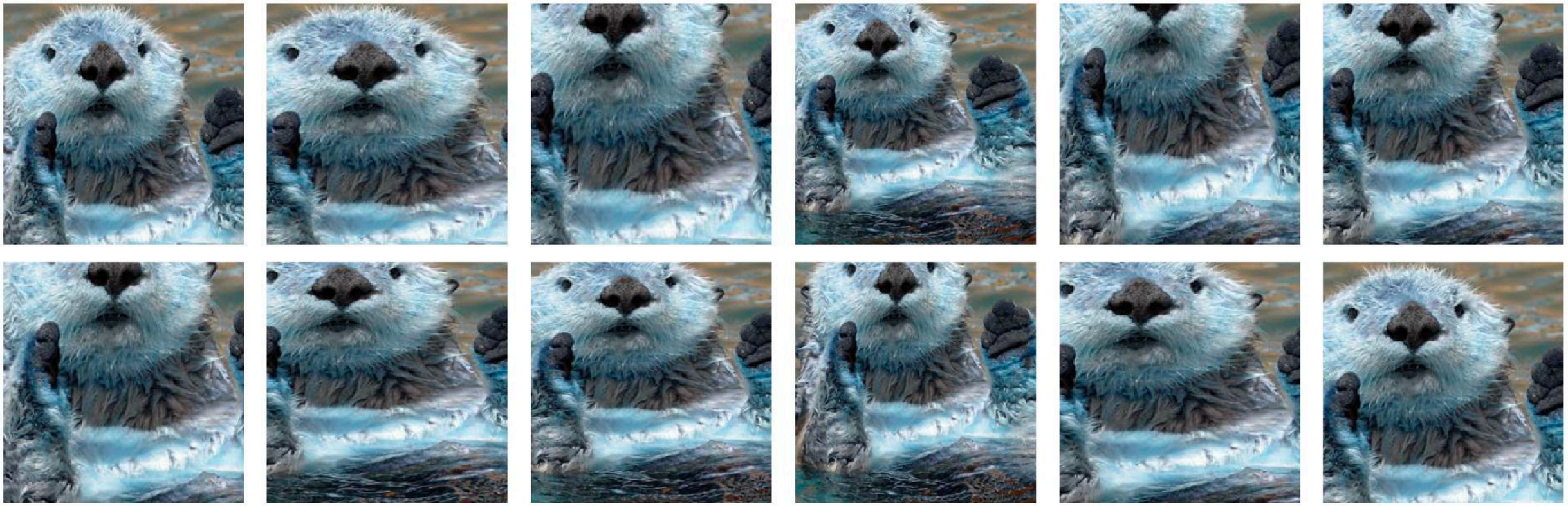
- Best way towards better generalization is to **train on more data**
- However, data in practice often limited
- Goal of data augmentation: create **“fake” data** from the existing data (on the fly) and add it to the training set
- New data must **preserve semantics**
- Even **simple operations** like translation or adding per-pixel noise often already greatly improve generalization
- https://github.com/aleju/imgaug



Geometric Transformations

Data Augmentation: Geometry

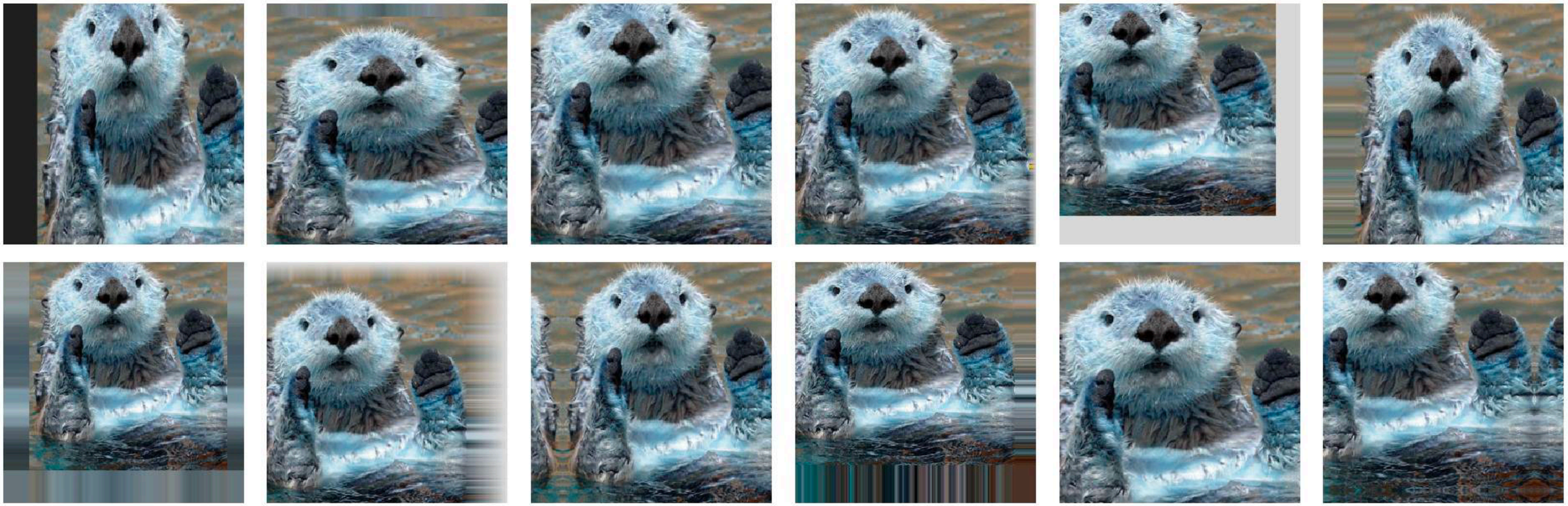
Image Cropping



```
iaa.Crop(px=(1,64))
```

Data Augmentation: Geometry

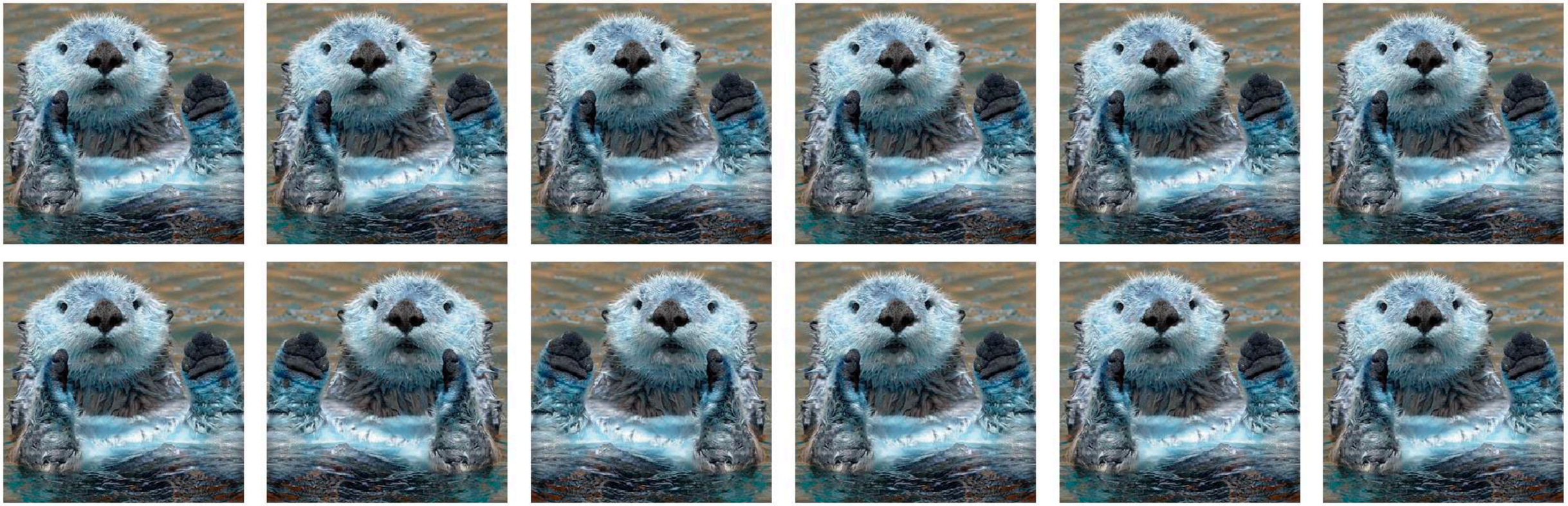
Image Cropping and Padding



```
iaa.CropAndPad(percent=(-0.2, 0.2),pad mode=iaa.ia.ALL,pad cval=(0, 255))
```

Data Augmentation: Geometry

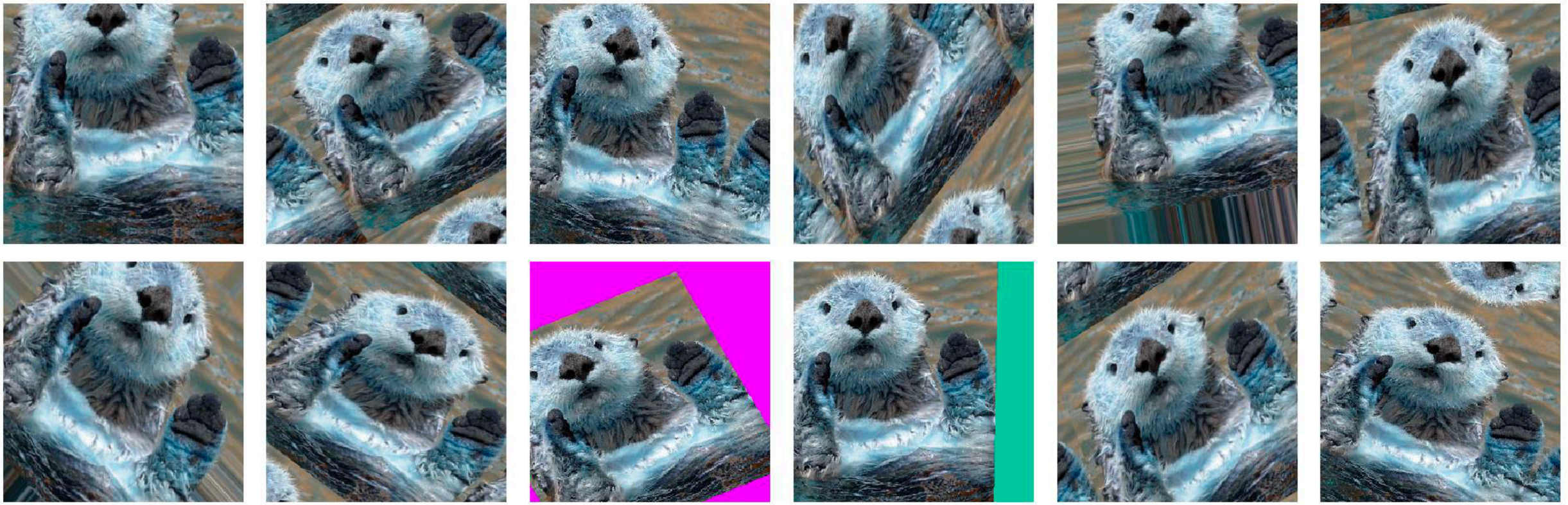
Horizontal Image Flipping



```
iaa.Fliplr(0.5)
```

Data Augmentation: Geometry

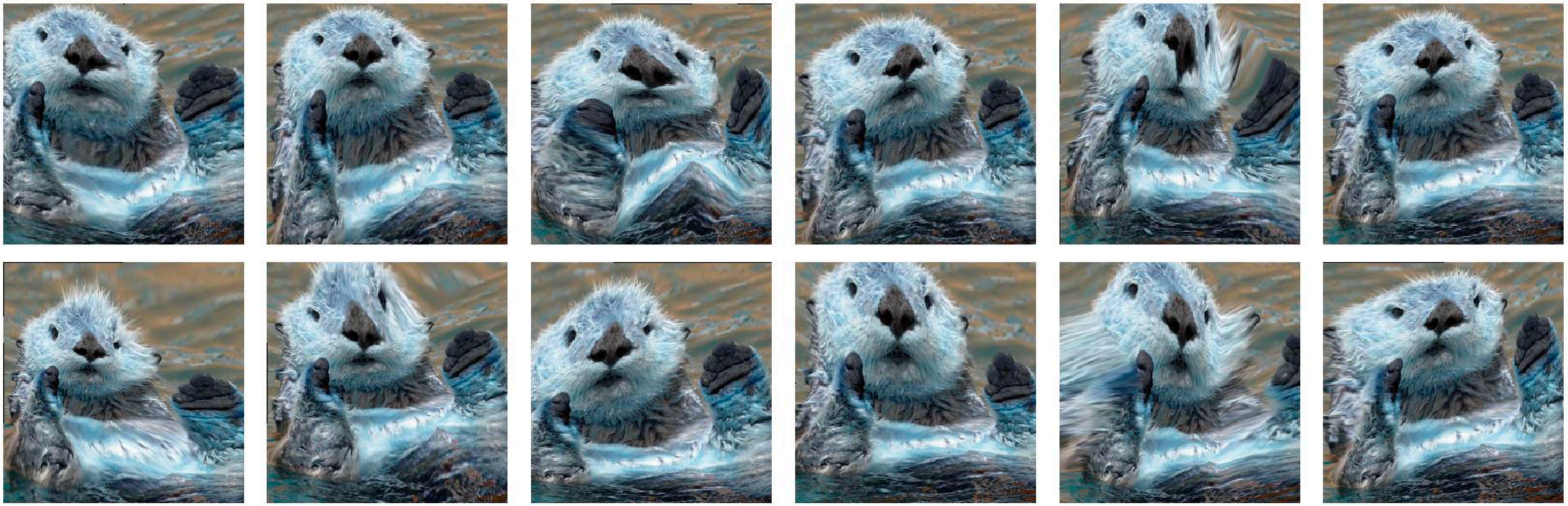
Affine Transformation



`iaa.Affine(0)`

Data Augmentation: Geometry

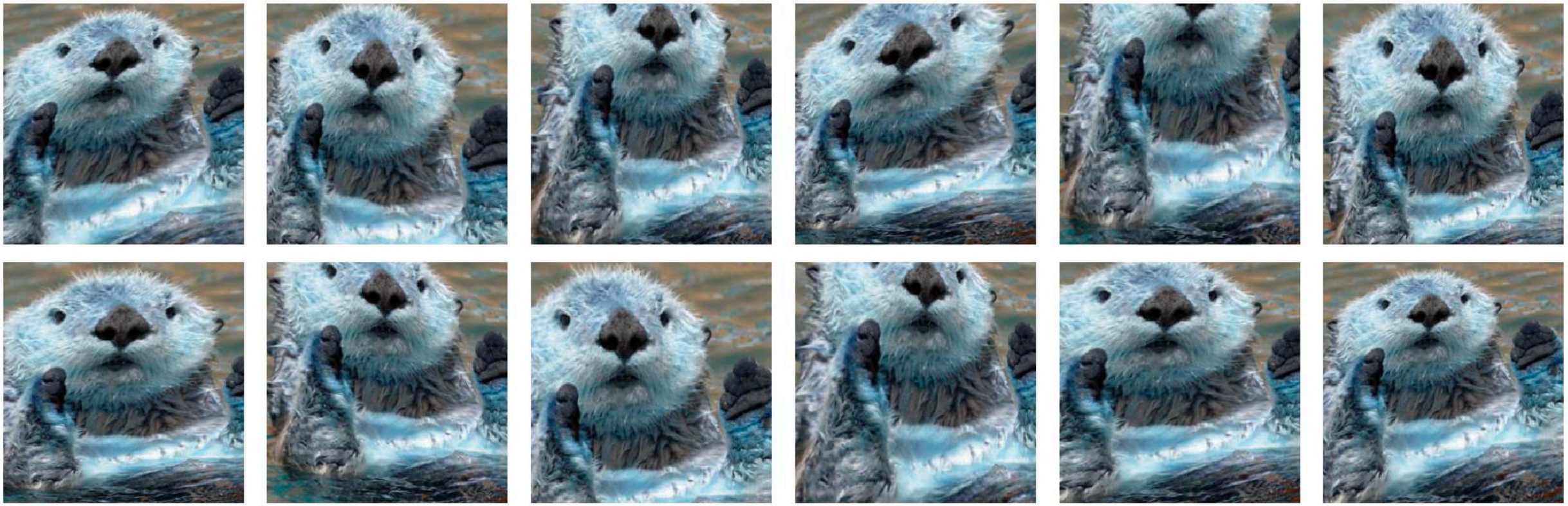
Piecewise Affine Transformation



```
iaa.PiecewiseAffine(scale=(0.01, 0.1))
```

Data Augmentation: Geometry

Persepctive Transformation

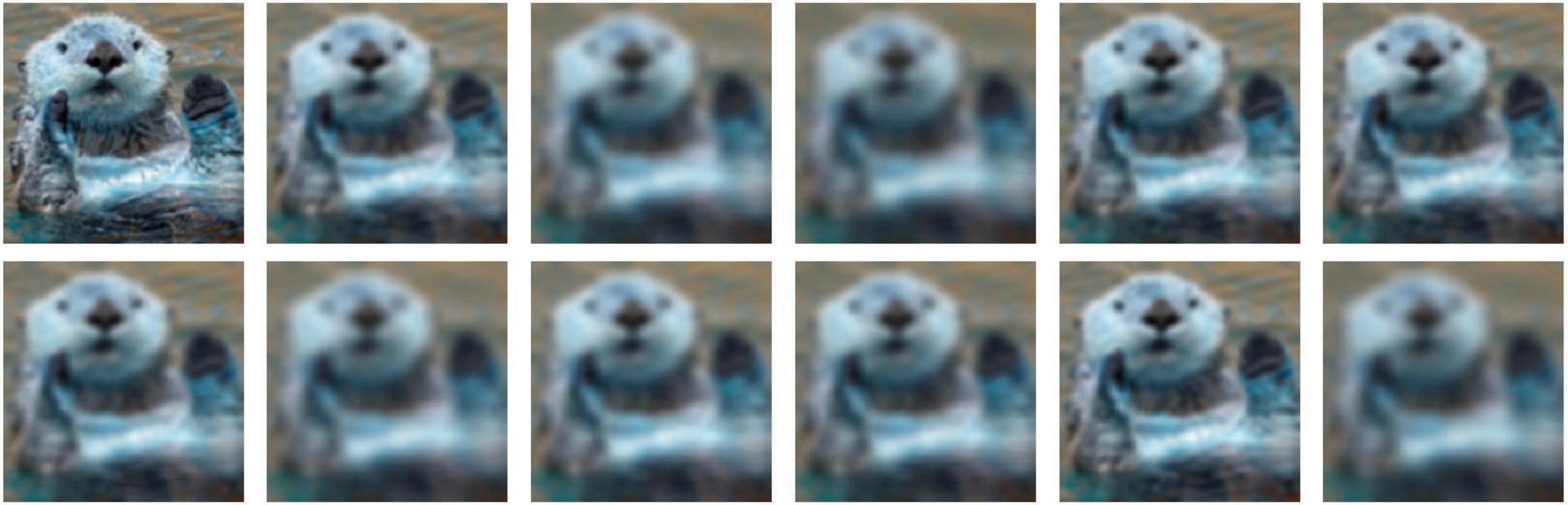


```
iaa.PerspectiveTransform(scale=(0, 0.4))
```

Local Filters

Data Augmentation: Local Filters

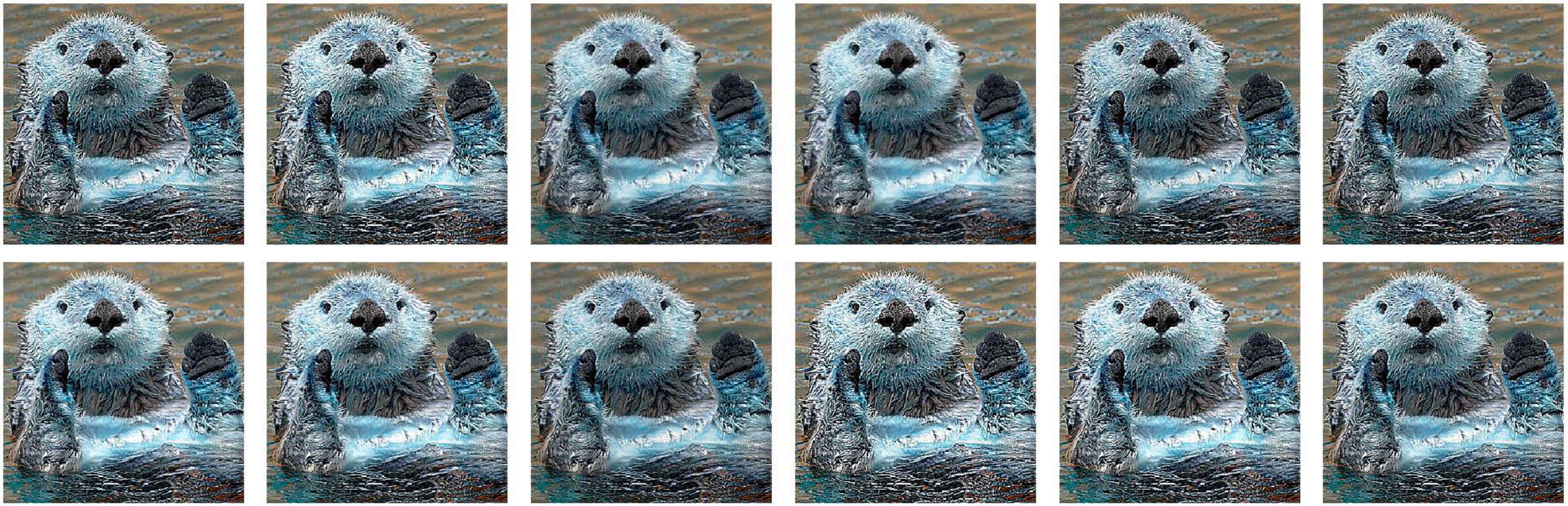
Gaussian Blur



```
iaa.GaussianBlur(sigma=(0.0, 10.0))
```

Data Augmentation: Local Filters

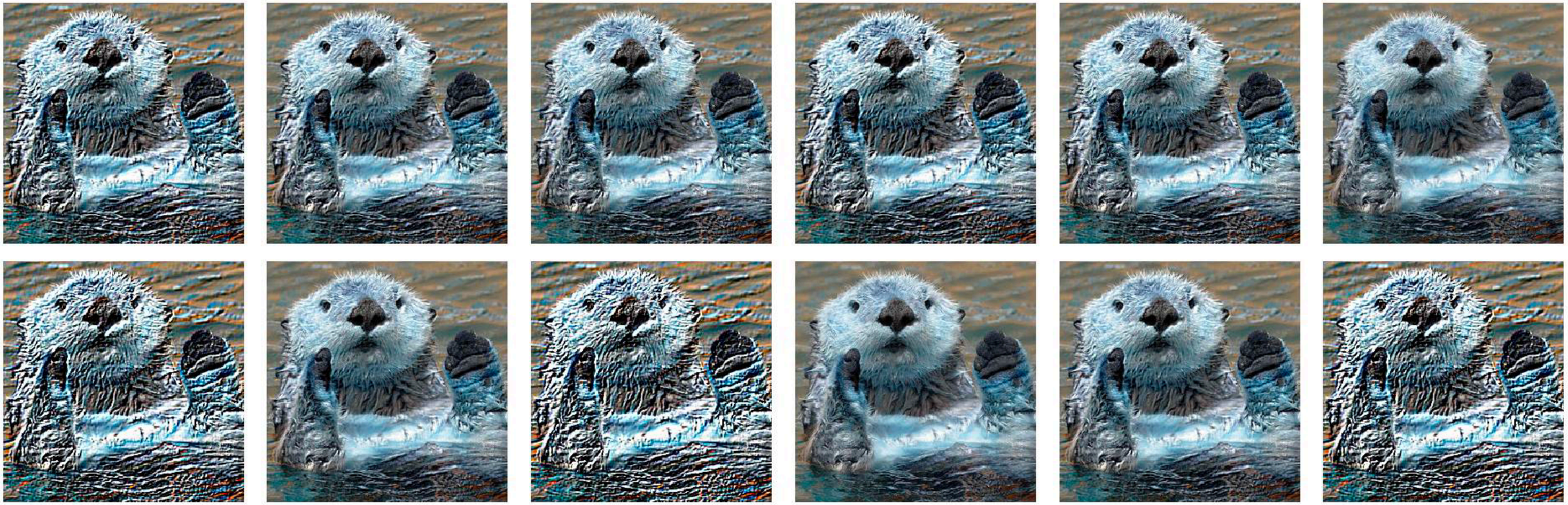
Image Sharpening



```
iaa.Sharpen(alpha=(0, 0.5), lightness=(0.75, 1.25))
```

Data Augmentation: Local Filters

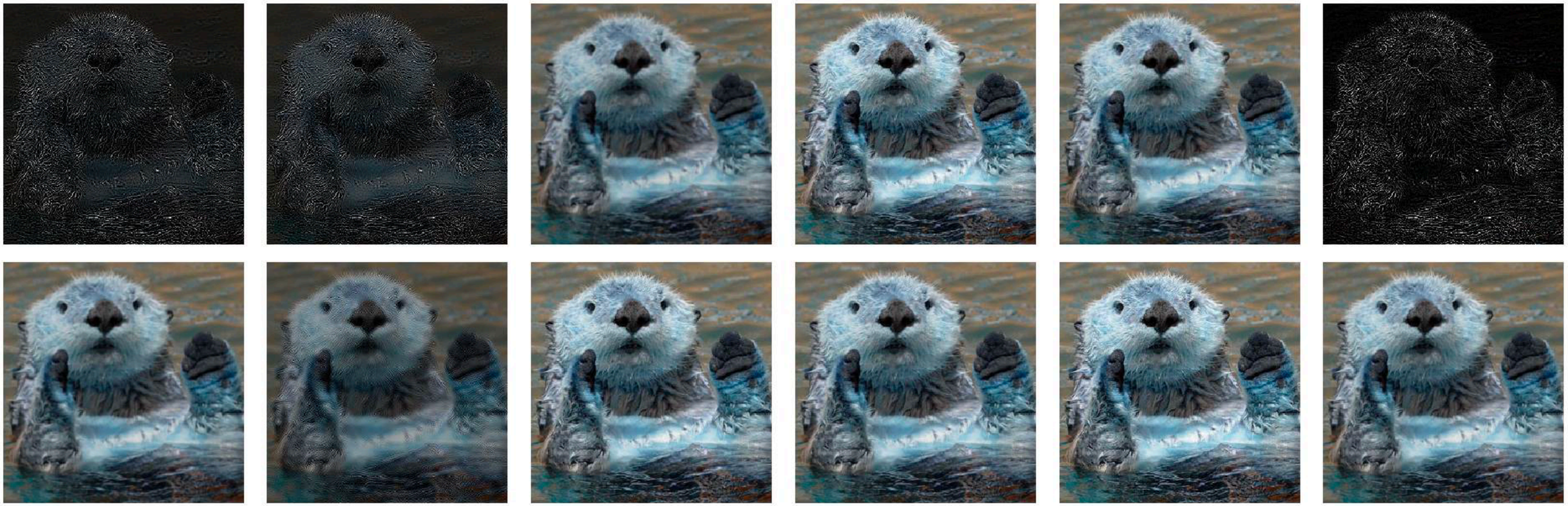
Emboss Effect



```
iaa.Emboss(alpha=(0, 1.0), strength=(0, 2.0))
```

Data Augmentation: Local Filters

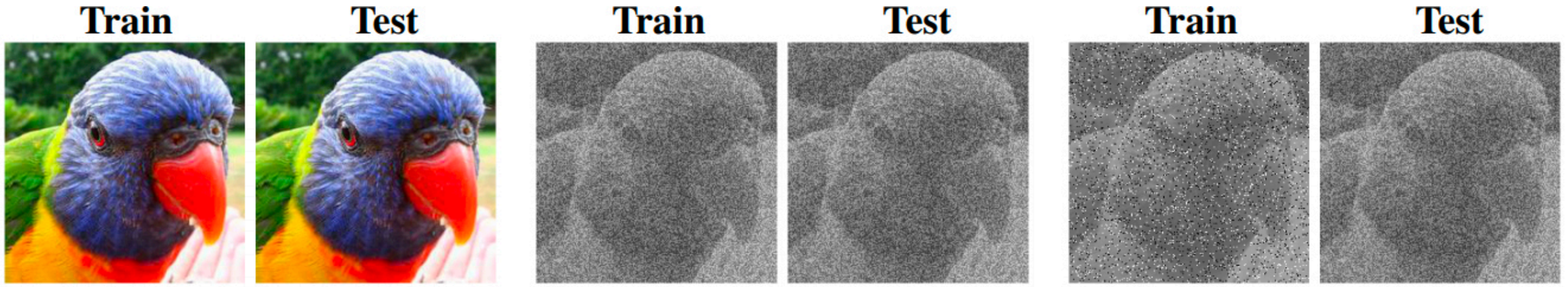
Edge Detection



```
iaa.EdgeDetect(alpha=(0, 1.0))
```

Adding Noise

Data Augmentation: Noise



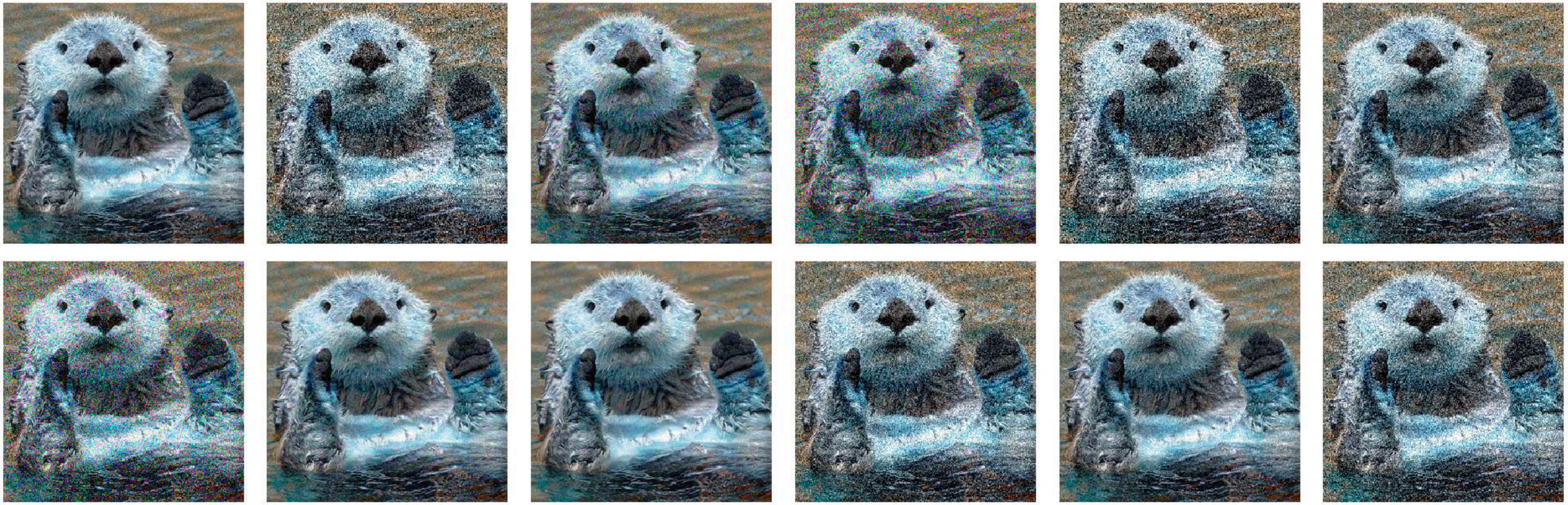
(a) Super-human performance (b) Super-human performance (c) Chance level performance

Figure 1: Classification performance of ResNet-50 trained from scratch on (potentially distorted) ImageNet images. **(a)** Classification performance when trained on standard colour images and tested on colour images is close to perfect (better than human observers). **(b)** Likewise, when trained and tested on images with additive uniform noise, performance is super-human. **(c)** Striking generalisation failure: When trained on images with salt-and-pepper noise and tested on images with uniform noise, performance is at chance level—even though both noise types do not seem much different to human observers.

Geirhos, Temme, Rauber, Schutt, Bethge and Wichmann: Generalisation in humans and deep neural networks. NeurIPS, 2018.

Data Augmentation: Noise

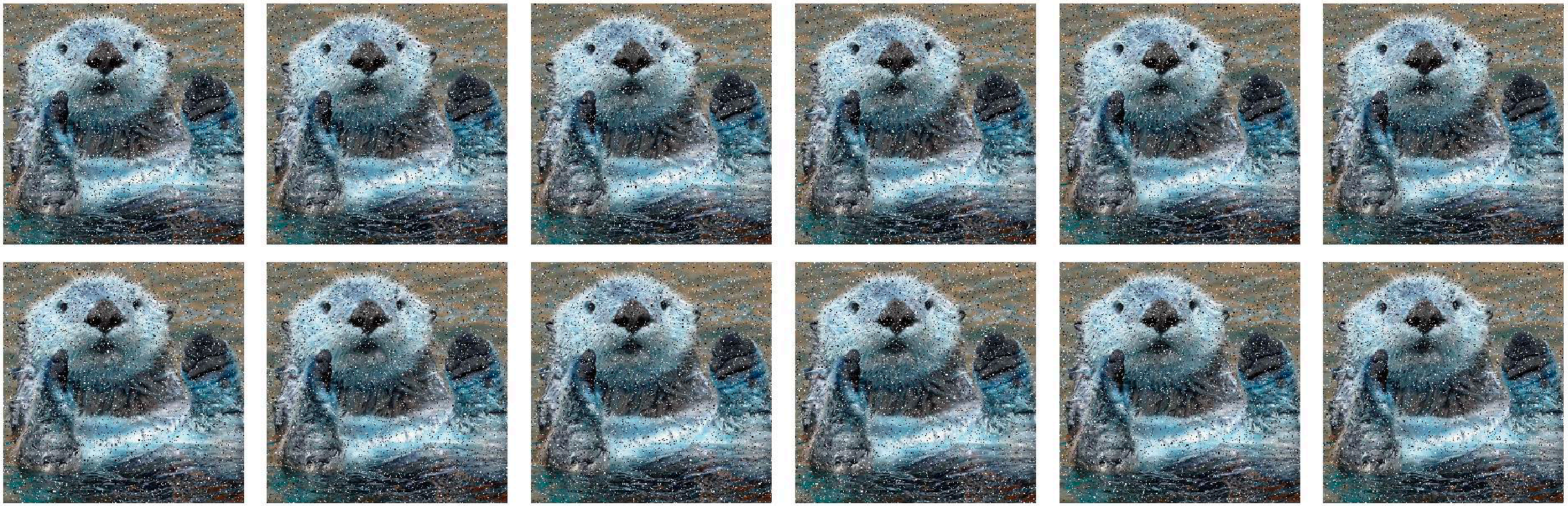
Gaussian Noise



```
iaa.AdditiveGaussianNoise(loc=0, scale=(0.0, 0.2*255)
```

Data Augmentation: Noise

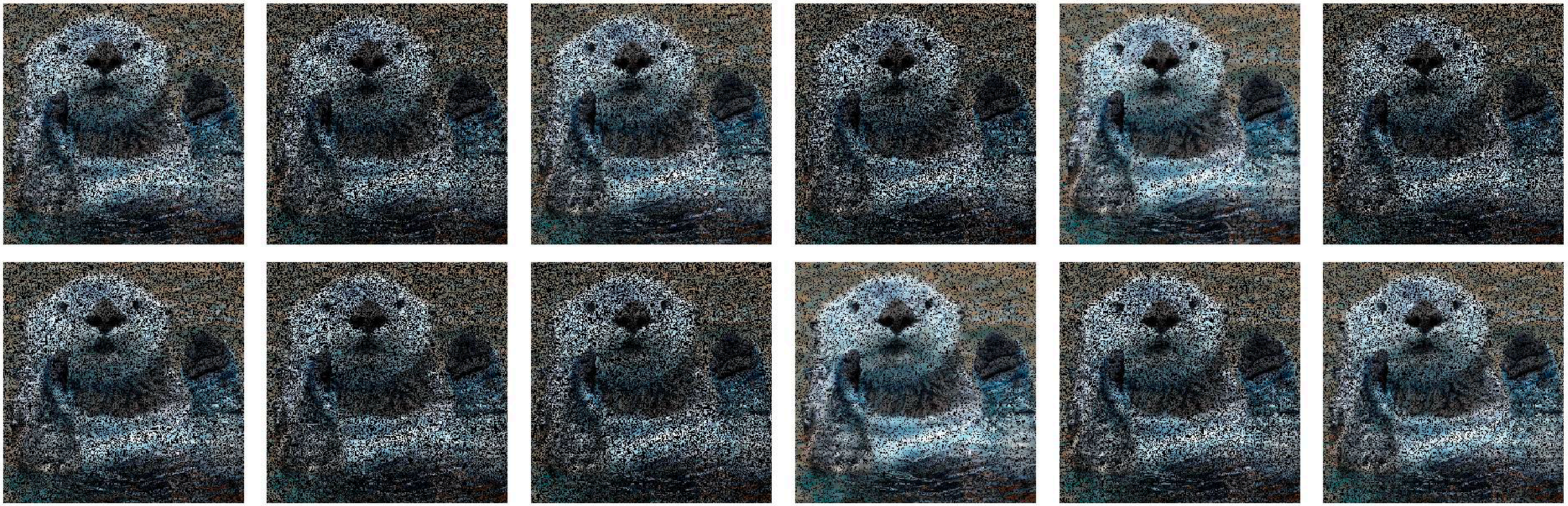
Salt and Pepper Noise



```
iaa.SaltAndPepper(0.1)
```

Data Augmentation: Noise

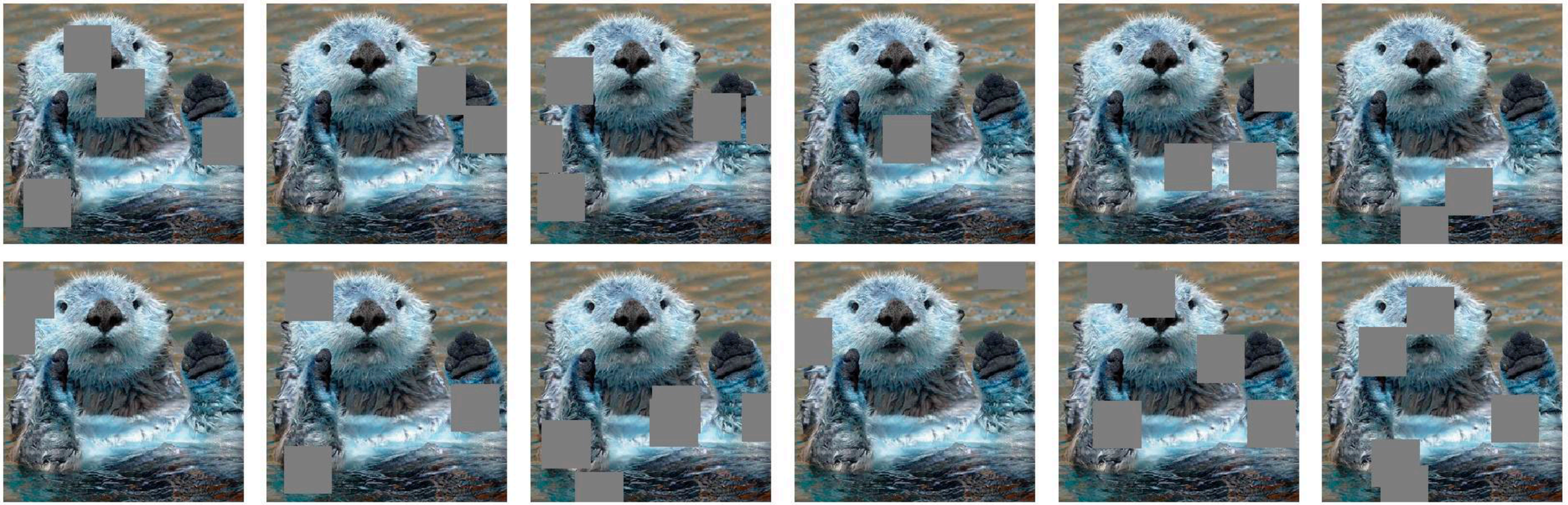
Dropout Noise



```
iaa.Dropout((0.01, 0.5))
```

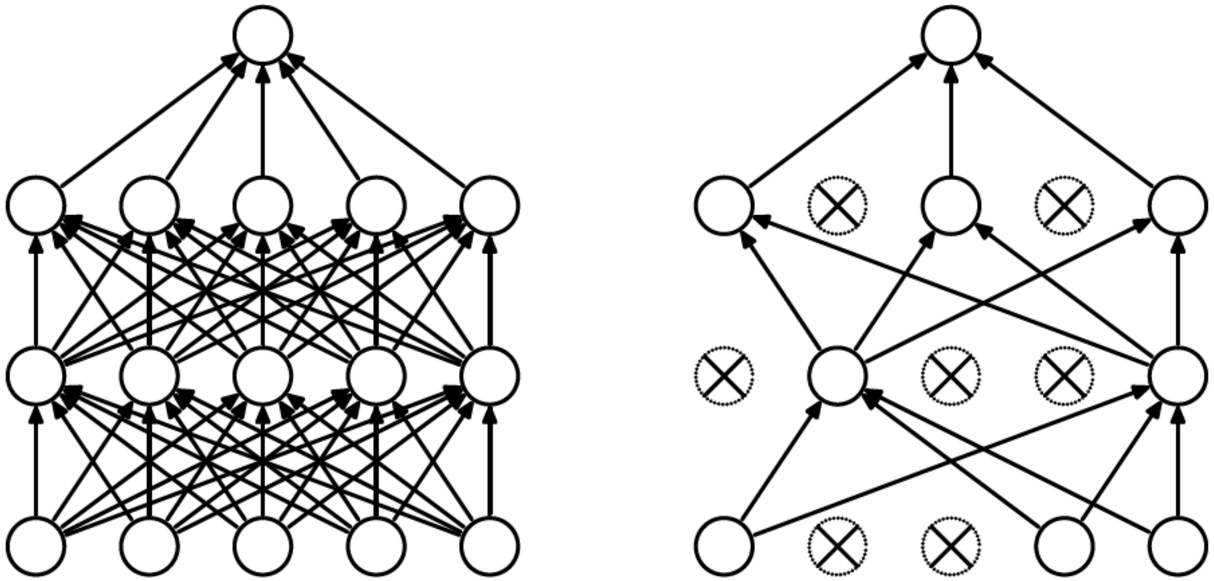
Data Augmentation: Noise

Cutout



```
iaa.Cutout(nb iterations=(1, 5), size=0.2, squared=False)
```

Data Augmentation: Noise



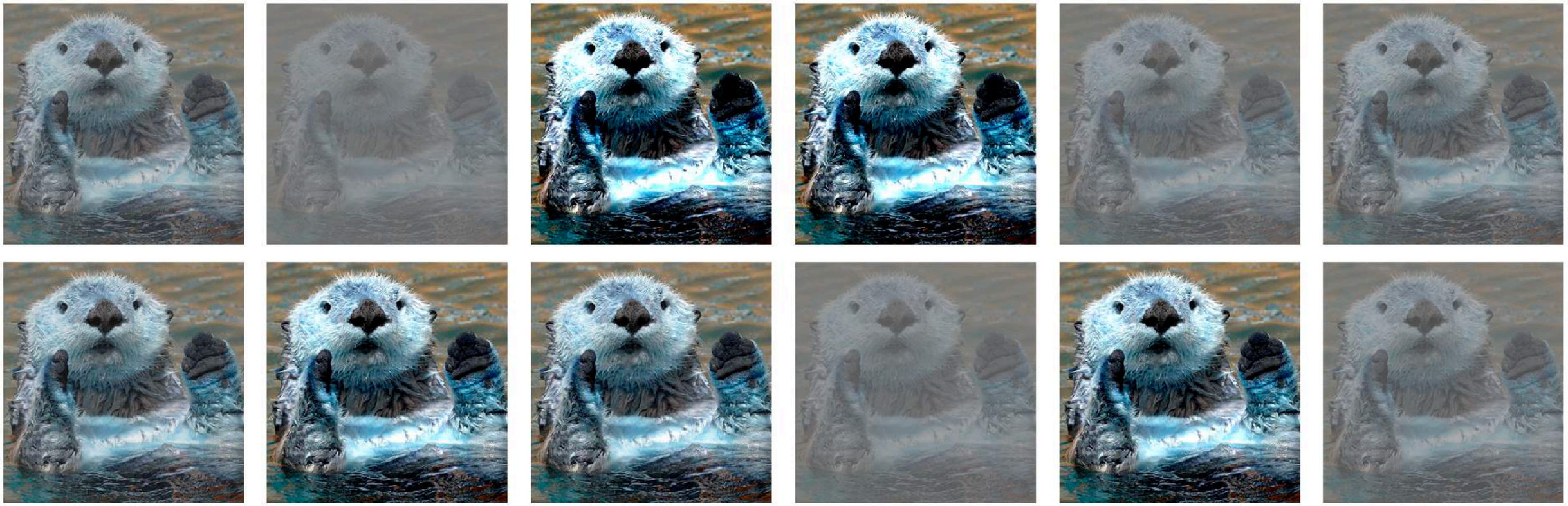
Noise Augmentation:

- ✦ Noise can also be applied to the **hidden units**, not only to the input
- ✦ Prominent example: **Dropout**

Color Transformations

Data Augmentation: Color

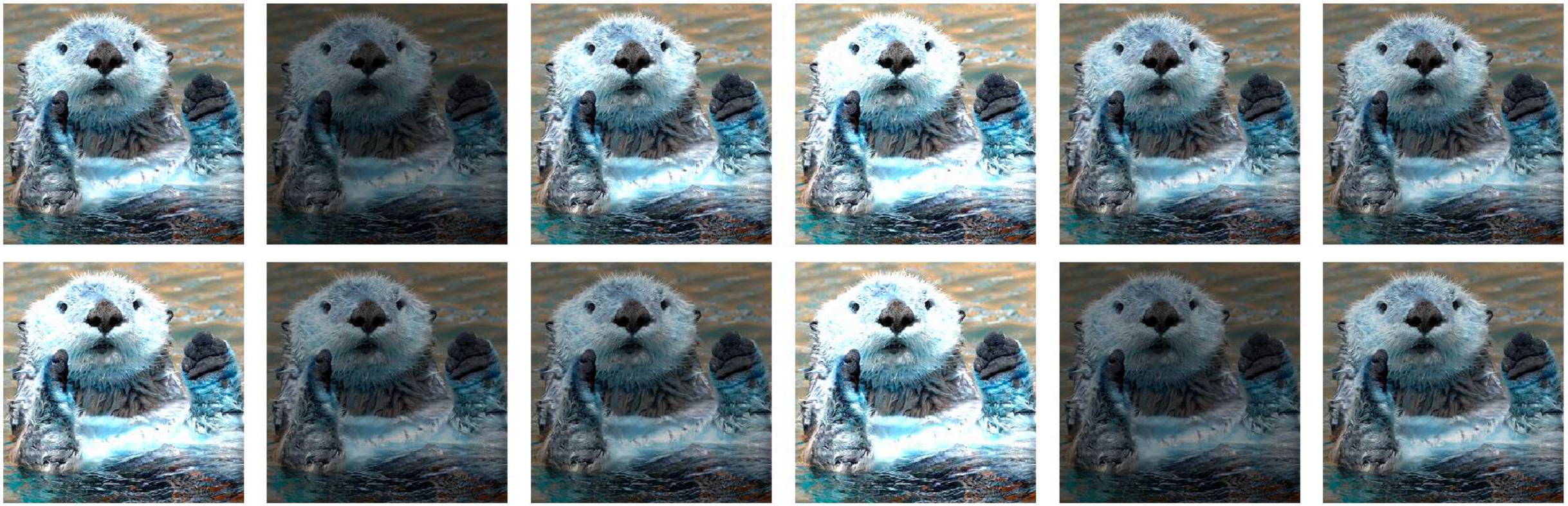
Constrast



```
iaa.LinearContrast((0.1, 2.0), per_channel=0)
```

Data Augmentation: Color

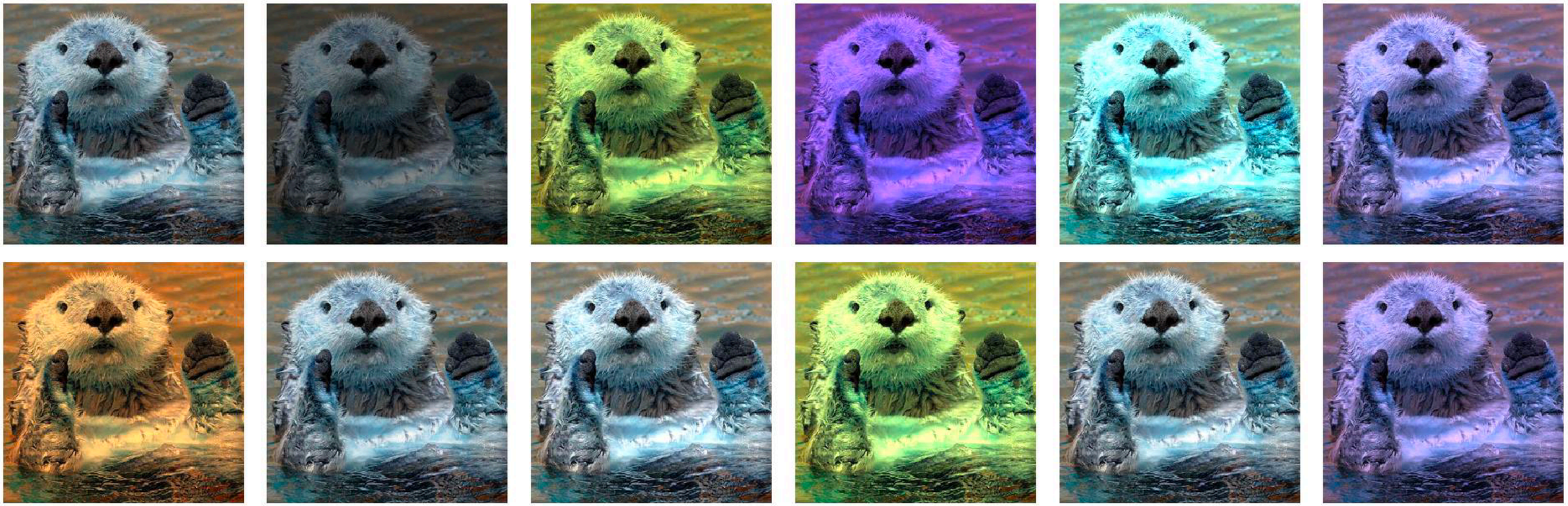
Brightness



```
iaa.Multiply((0.5, 1.5), per_channel=0)
```

Data Augmentation: Color

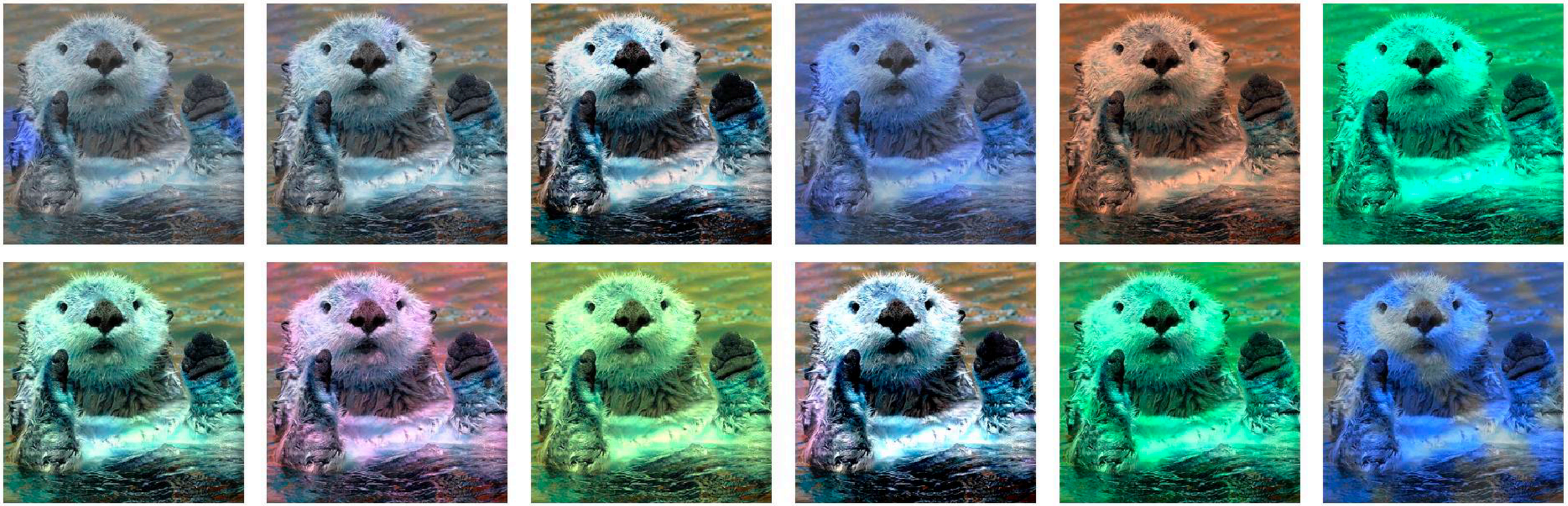
Brightness per Channel



```
iaa.Multiply((0.5, 1.5), per_channel=0.5)
```

Data Augmentation: Color

Local Brigthness



```
iaa.FrequencyNoiseAlpha(exponent=(-4, 0), first=iaa.Multiply((0.5, 1.5), per channel=True))
```

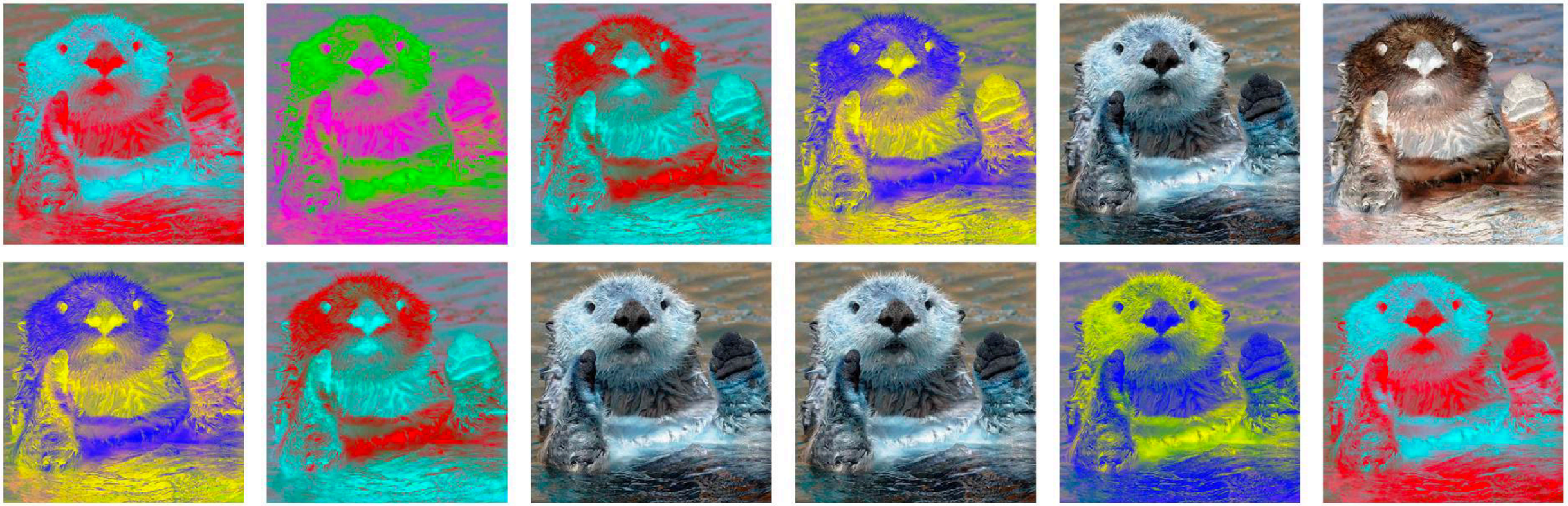
Data Augmentation: Color

Hue and Saturation

```
iaa.AddToHueAndSaturation((-50, 50))
```

Data Augmentation: Color

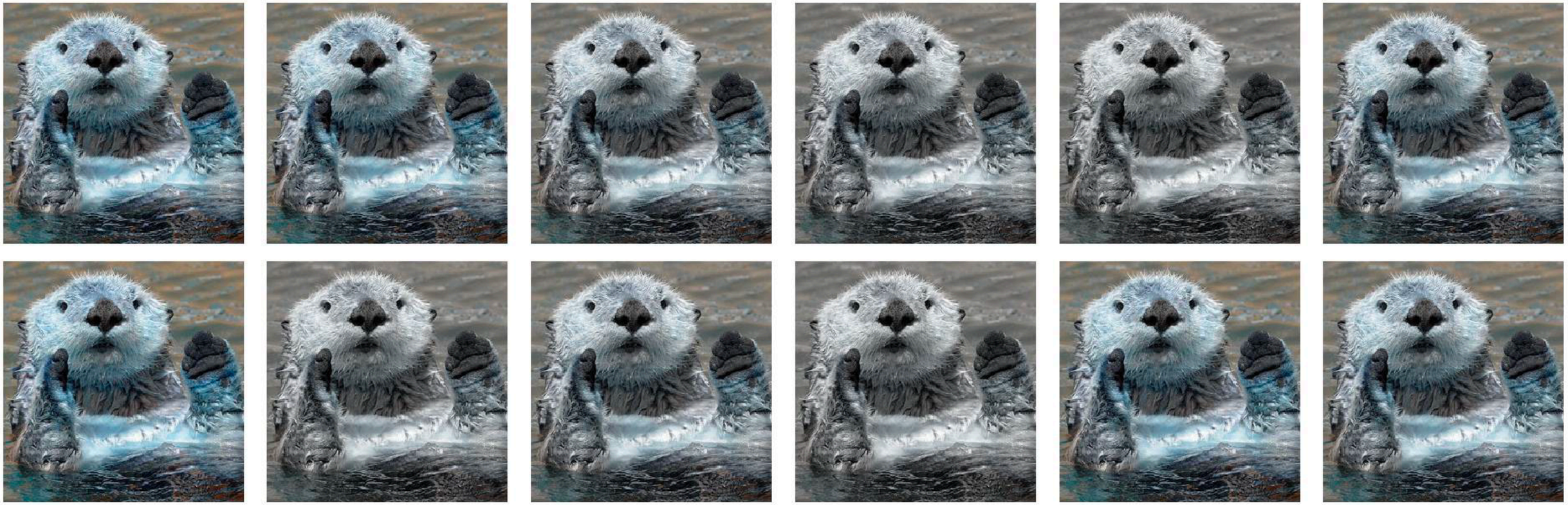
Color Inversion



```
iaa.Invert(0.5, per_channel=0.75)((-50, 50))
```

Data Augmentation: Color

Grayscale

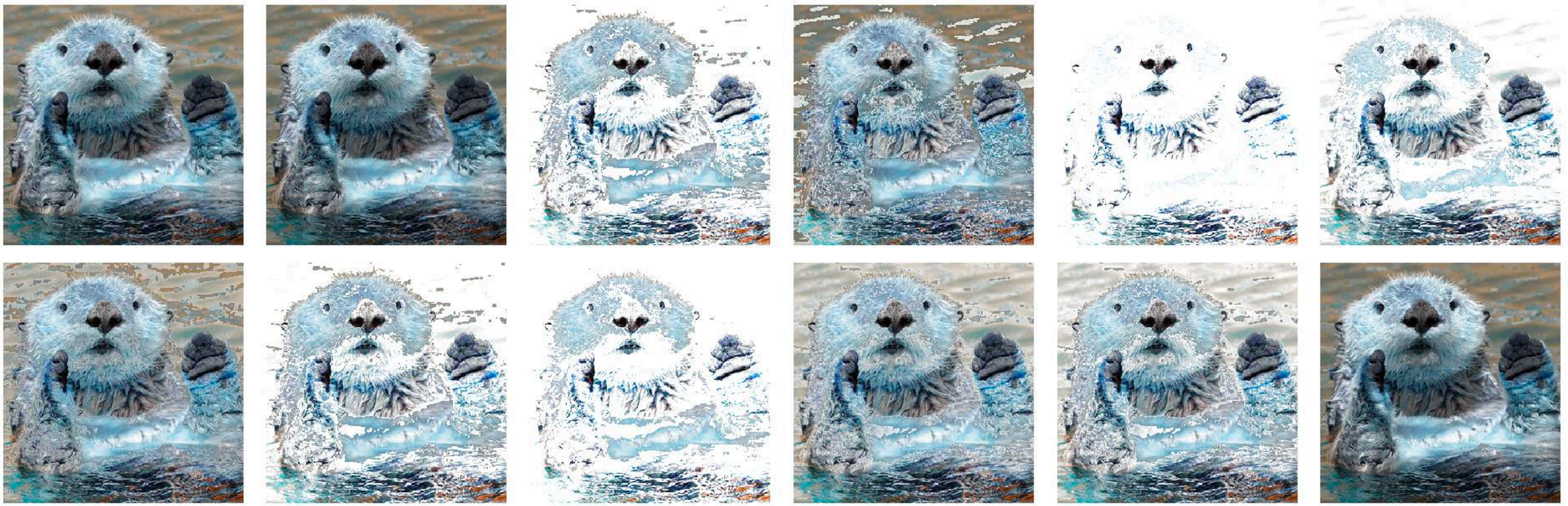


```
iaa.Grayscale(alpha=(0.0, 1.0))
```

Weathers

Data Augmentation: Weather

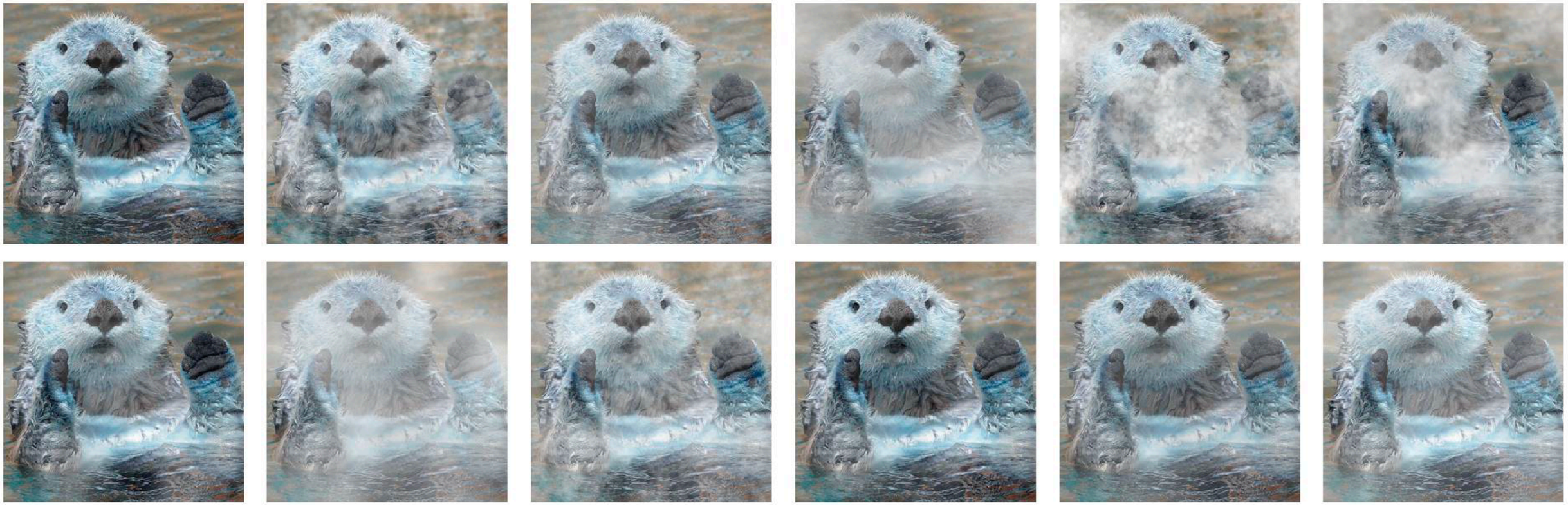
Snow



```
iaa.FastSnowyLandscape(lightness threshold=(100, 255),lightness multiplier=(1.0, 4.0))
```

Data Augmentation: Weather

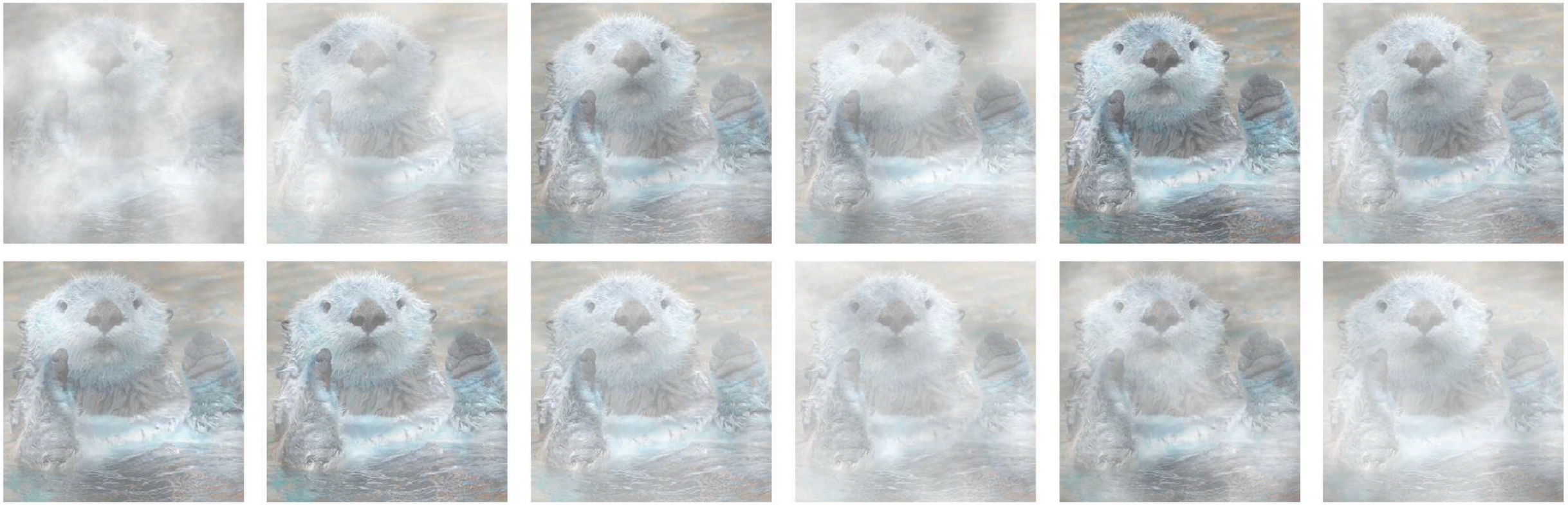
Clouds



`iaa.Clouds()`

Data Augmentation: Weather

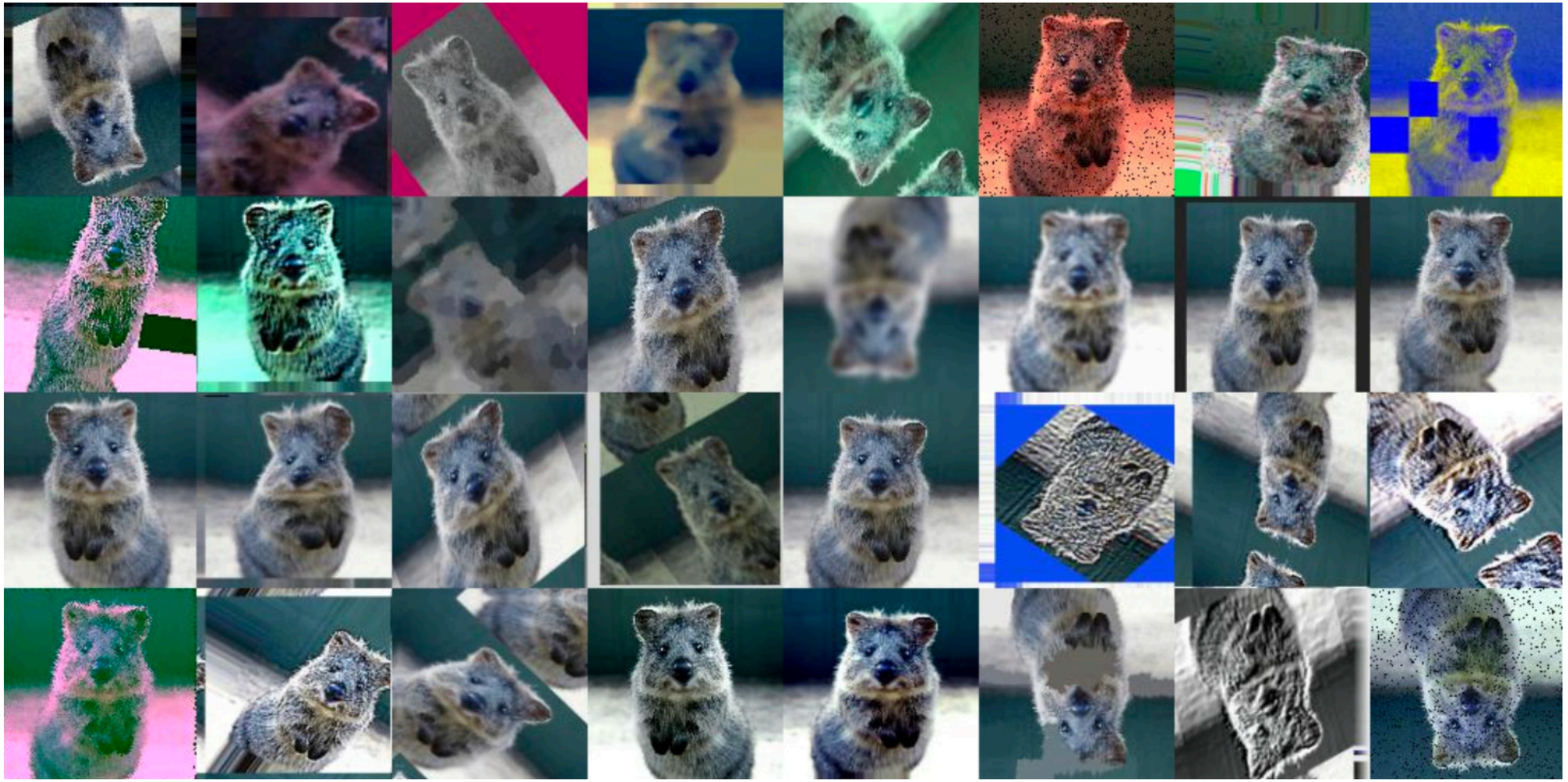
Fog



`iaa.Fog()`

Random Combinations

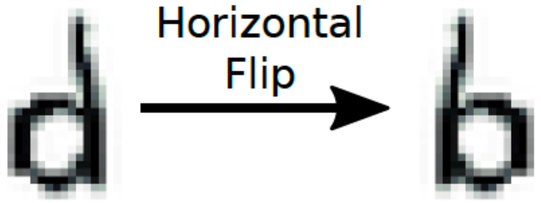
Data Augmentation: Random Combination



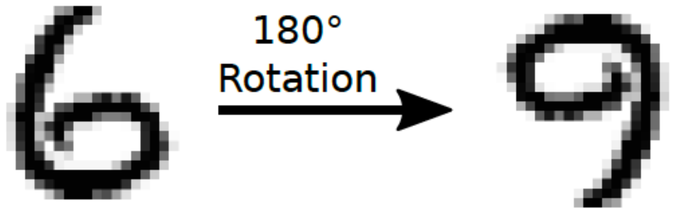
Output Transformations

Data Augmentation: Output Transformations

- For some classification tasks, e.g., **handwritten letter recognition**, be careful to not apply transformations that would change the output class
- **Example 1**: Horizontal flips changes the interpretation of the letter 'd':

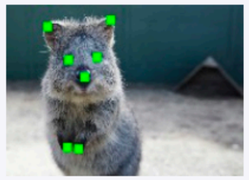
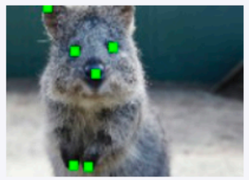
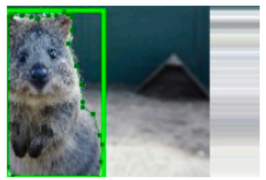
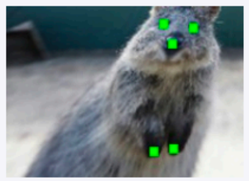
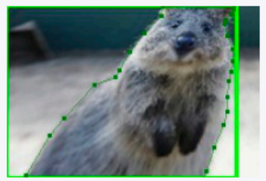


- **Example 2**: 180 rotations changes the interpretation of the number '6':



- **Remark**: For **general object recognition**, flips and rotations can often be useful!

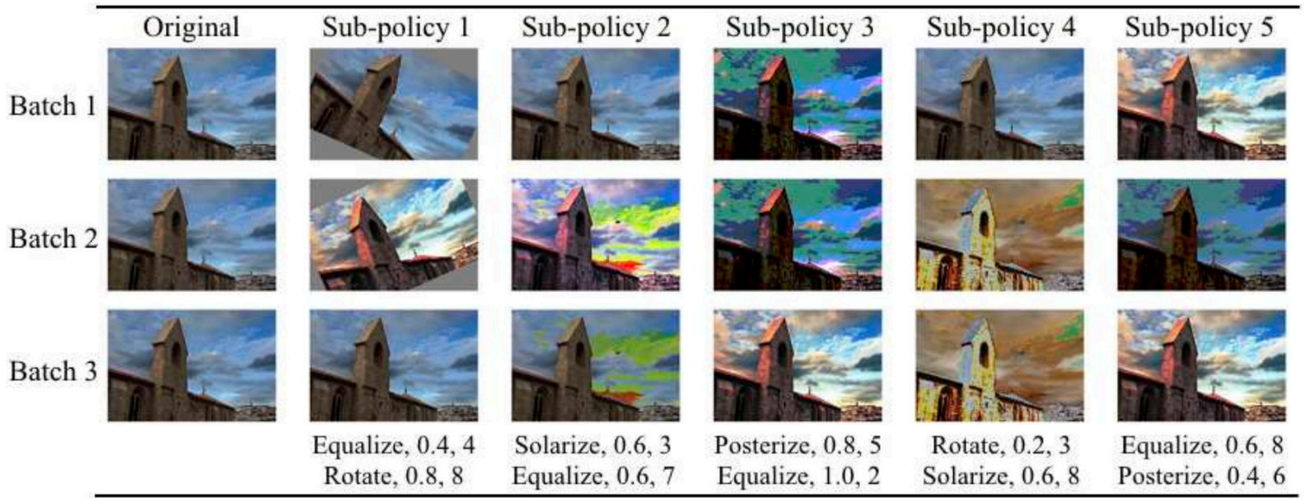
Data Augmentation: Output Transformations

Original Input					
Gauss. Noise + Contrast + Sharpen					
Affine					
Crop + Pad					
Fliplr + Perspective					

✦ For dense prediction tasks (depth/instance/keypoints), also **transform targets**

Data Augmentation

- ✦ When comparing two networks, make sure you **use the same augmentation**
- ✦ Consider data augmentation as a **part of your network design**
- ✦ It is important to specify the right distributions (often done empirically)
- ✦ Can also be **combined with ensemble idea**:
  - At training time, sample random crops/scales and train one model
  - At inference time, average predictions for a fixed set of crops of the test image
- ✦ AutoAugment uses reinforcement learning to **find strategies automatically**:



Cubuk, Zoph, Mane, Vasudevan, Le: AutoAugment: Learning Augmentation Strategies From Data. CVPR, 2019.