# Layers (Many per Model):

## *Select 1D/2D/3D for the entire model (based on input), not per layer*

## Convolutional Layers

#### Tooltips (For all Conv)

This layer creates a convolution kernel that is convolved

(actually cross-correlated) with the layer input to produce a tensor of

outputs. If `use\_bias` is True (and a `bias\_initializer` is provided),

a bias vector is created and added to the outputs. Finally, if

`activation` is not `None`, it is applied to the outputs as well.

### 1d Convolutional (Conv) Layer

#### Parameters specified by UI

**inputs:** Tensor input.

**filters:** Integer, the dimensionality of the output space (i.e. the number

of filters in the convolution).

**kernel\_size:** An integer or tuple/list of a single integer, specifying the

length of the 1D convolution window.

**strides:** An integer or tuple/list of a single integer,

specifying the stride length of the convolution.

Specifying any stride value != 1 is incompatible with specifying

any `dilation\_rate` value != 1.

**padding:** One of `"valid"` or `"same"` (case-insensitive).

**activation:** Activation function. Set it to None to maintain a

linear activation. ***(See Activation Below)***

**use\_bias:** Boolean, whether the layer uses a bias.

**name:** A string, the name of the layer.

**reuse:** Boolean, whether to reuse the weights of a previous layer

by the same name.

### 2d Convolutional (Conv) Layer

#### Parameters Specified by UI

**filters:** Integer, the dimensionality of the output space (i.e. the number

of filters in the convolution).

**kernel\_size:** An integer or tuple/list of 2 integers, specifying the

width and height of the 2D convolution window.

Can be a single integer to specify the same value for

all spatial dimensions.

**strides:** An integer or tuple/list of 2 integers,

specifying the strides of the convolution along the height and width.

Can be a single integer to specify the same value for

all spatial dimensions.

Specifying any stride value != 1 is incompatible with specifying

any `dilation\_rate` value != 1.

**padding:** One of `"valid"` or `"same"` (case-insensitive).

**activation:** Activation function. Set it to None to maintain a

linear activation. ***(See Activation Below)***

**use\_bias:** Boolean, whether the layer uses a bias.

**name:** A string, the name of the layer. (String Field)

**reuse:** Boolean, whether to reuse the weights of a previous layer

by the same name.

### 3d Convolutional (Conv) Layer

#### Parameters specified by UI

**filters:** Integer, the dimensionality of the output space (i.e. the number

of filters in the convolution).

**kernel\_size:** An integer or tuple/list of 3 integers, specifying the

depth, height and width of the 3D convolution window.

Can be a single integer to specify the same value for

all spatial dimensions.

**strides**: An integer or tuple/list of 3 integers,

specifying the strides of the convolution along the depth,

height and width.

Can be a single integer to specify the same value for

all spatial dimensions.

Specifying any stride value != 1 is incompatible with specifying

any `dilation\_rate` value != 1.

**padding:** One of `"valid"` or `"same"` (case-insensitive).

**activation:** Activation function. Set it to None to maintain a

linear activation. ***(See Activation Below)***

**use\_bias:** Boolean, whether the layer uses a bias.

**name:** A string, the name of the layer.

**reuse:** Boolean, whether to reuse the weights of a previous layer

by the same name.

## Pooling Layers

### 1d Max Pooling (MaxPool) Layer

#### Parameters specified by UI

**pool\_size:** An integer or tuple/list of a single integer,

representing the size of the pooling window.

**strides:** An integer or tuple/list of a single integer, specifying the

strides of the pooling operation.

**padding:** A string. The padding method, either 'valid' or 'same'.

Case-insensitive.

**name:** A string, the name of the layer.

### 2d Max Pooling (MaxPool) Layer

#### Parameters specified by UI

**pool\_size:** An integer or tuple/list of 2 integers: (pool\_height, pool\_width)

specifying the size of the pooling window.

Can be a single integer to specify the same value for

all spatial dimensions.

**strides:** An integer or tuple/list of 2 integers,

specifying the strides of the pooling operation.

Can be a single integer to specify the same value for

all spatial dimensions.

**padding:** A string. The padding method, either 'valid' or 'same'.

Case-insensitive.

**name:** A string, the name of the layer.

### 3d Max Pooling (MaxPool) Layer

#### Parameters specified by UI

**pool\_size:** An integer or tuple/list of 3 integers:

(pool\_depth, pool\_height, pool\_width)

specifying the size of the pooling window.

Can be a single integer to specify the same value for

all spatial dimensions.

**strides:** An integer or tuple/list of 3 integers,

specifying the strides of the pooling operation.

Can be a single integer to specify the same value for

all spatial dimensions.

**padding:** A string. The padding method, either 'valid' or 'same'.

Case-insensitive.

**name:** A string, the name of the layer.

## Other Layers

### Fully Connected (Dense) Layer

#### Parameters specified by UI

**units:** Integer or Long, dimensionality of the output space.

**activation:** Activation function (callable). Set it to None to maintain a

linear activation. ***(See Activation Below)***

**use\_bias:** Boolean, whether the layer uses a bias.

**name:** String, the name of the layer.

**reuse:** Boolean, whether to reuse the weights of a previous layer

by the same name.

#### Tooltip

Functional interface for the densely-connected layer.

This layer implements the operation:

`outputs = activation(inputs.kernel + bias)`

Where `activation` is the activation function passed as the `activation`

argument (if not `None`), `kernel` is a weights matrix created by the layer,

and `bias` is a bias vector created by the layer

(only if `use\_bias` is `True`).

### Dropout (Drop) Layer

#### Parameters specified by UI

**rate:** The dropout rate, between 0 and 1. E.g. "rate=0.1" would drop out

10% of input units.

**training:** Either a Python boolean, or a TensorFlow boolean scalar tensor

(e.g. a placeholder). Whether to return the output in training mode

(apply dropout) or in inference mode (return the input untouched).

**name:** The name of the layer (string).

#### Tooltip

Dropout consists in randomly setting a fraction `rate` of input units to 0

at each update during training time, which helps prevent overfitting.

The units that are kept are scaled by `1 / (1 - rate)`, so that their

sum is unchanged at training time and inference time.

### Batch Normalization (BatchNorm) Layer

#### Parameters specified by UI

**offset (beta):** Shifts the normalized tensor by some offset tensor (idk how we should do this, maybe not).

**scale (gamma):** Scales the normalized tensor by some real value (any number).

**epsilon:** A small value for avoiding divide by zero errors (real number from (0,1) default to 10e-7).

**name:** The name of the layer (string).

#### Tooltip

As described in <http://arxiv.org/abs/1502.03167.>

Normalizes a tensor by `mean` and `variance`, and applies (optionally) a

`scale` to it, as well as an `offset` :

## Activation Functions

**Choices**: ReLU, ReLU6, CReLU, ExpLU, SoftPlus, SoftSign, Sigmoid, Tanh,

# Other Params:

## Loss Function (1 Per Model)

Choose 1 of:

### Log Loss

### Hinge Loss

### Mean Squared Error

### Absolute Difference

### Cosine Difference

### Mean Pairwise Squared Error

### Sigmoid Cross Entropy

### Softmax Cross Entropy

### Sparse Softmax Cross Entropy

## Optimizers (1 Per Model)

### Gradient Descent

#### Parameters specified by UI

**learning\_rate:** A Tensor or a floating point value. The learning

rate to use.

**use\_locking:** If True use locks for update operations.

**name:** Optional name prefix for the operations created when applying

gradients. Defaults to "GradientDescent".

#### Tooltip:

Optimizer that implements the gradient descent algorithm.

### Adadelta Optimizer

#### Parameters specified by UI

**learning\_rate:** A `Tensor` or a floating point value. The learning rate.

**rho:** A `Tensor` or a floating point value. The decay rate.

**epsilon:** A `Tensor` or a floating point value. A constant epsilon used

to better conditioning the grad update.

**use\_locking:** If `True` use locks for update operations.

**name:** Optional name prefix for the operations created when applying

gradients. Defaults to "Adadelta".

#### Tooltip:

Optimizer that implements the Adadelta algorithm.

See [M. D. Zeiler](http://arxiv.org/abs/1212.5701)

([pdf](<http://arxiv.org/pdf/1212.5701v1.pdf))>

### Adagrad Optimizer

#### Parameters specified by UI

**learning\_rate:** A `Tensor` or a floating point value. The learning rate.

**initial\_accumulator\_value:** A floating point value.

Starting value for the accumulators, must be positive.

**use\_locking**: If `True` use locks for update operations.

**name:** Optional name prefix for the operations created when applying

gradients. Defaults to "Adagrad".

#### Tooltip:

Optimizer that implements the Adagrad algorithm.

See this [paper](http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf)

or this

[intro](http://cs.stanford.edu/~ppasupat/a9online/uploads/proximal\_notes.pdf).

### Adagrad Dual Averaging Optimizer (AdagradDA)

#### Parameters specified by UI

**learning\_rate:** A `Tensor` or a floating point value. The learning rate.

**initial\_gradient\_squared\_accumulator\_value:** A floating point value.

Starting value for the accumulators, must be positive.

**l1\_regularization\_strength:** A float value, must be greater than or

equal to zero.

**l2\_regularization\_strength:** A float value, must be greater than or

equal to zero.

**use\_locking:** If `True` use locks for update operations.

**name:** Optional name prefix for the operations created when applying

gradients. Defaults to "AdagradDA".

#### Tooltip:

Adagrad Dual Averaging algorithm for sparse linear models.

See this [paper](http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf).

This optimizer takes care of regularization of unseen features in a mini batch by updating them when they are seen with a closed form update rule that is equivalent to having updated them on every mini-batch.

AdagradDA is typically used when there is a need for large sparsity in the

trained model. This optimizer only guarantees sparsity for linear models. Be

careful when using AdagradDA for deep networks as it will require careful

initialization of the gradient accumulators for it to train.

### Adam Optimizer

#### Parameters specified by UI

**learning\_rate:** A Tensor or a floating point value. The learning rate.

**beta1:** A float value or a constant float tensor.

The exponential decay rate for the 1st moment estimates.

**beta2:** A float value or a constant float tensor.

The exponential decay rate for the 2nd moment estimates.

**epsilon:** A small constant for numerical stability.

**use\_locking:** If True use locks for update operations.

**name:** Optional name for the operations created when applying gradients.

Defaults to "Adam".

#### Tooltip:

Optimizer that implements the Adam algorithm.

See [Kingma et. al., 2014](http://arxiv.org/abs/1412.6980)

([pdf](http://arxiv.org/pdf/1412.6980.pdf)).

The default value of 1e-8 for epsilon might not be a good default in

general. For example, when training an Inception network on ImageNet a

current good choice is 1.0 or 0.1.

### Momentum Optimizer

#### Parameters specified by UI

**learning\_rate:** A `Tensor` or a floating point value. The learning rate.

momentum: A `Tensor` or a floating point value. The momentum.

**use\_locking:** If `True` use locks for update operations.

name: Optional name prefix for the operations created when applying

gradients. Defaults to "Momentum".

**use\_nesterov:** If `True` use Nesterov Momentum.

#### Tooltip:

Optimizer that implements the Momentum algorithm.

Computes (if `use\_nesterov = False`):

accumulation = momentum \* accumulation + gradient

variable -= learning\_rate \* accumulation

Note that in the dense version of this algorithm, `accumulation` is updated

and applied regardless of a gradient's value, whereas the sparse version (when

the gradient is an `IndexedSlices`, typically because of `tf.gather` or an

embedding) only updates variable slices and corresponding `accumulation` terms

when that part of the variable was used in the forward pass.