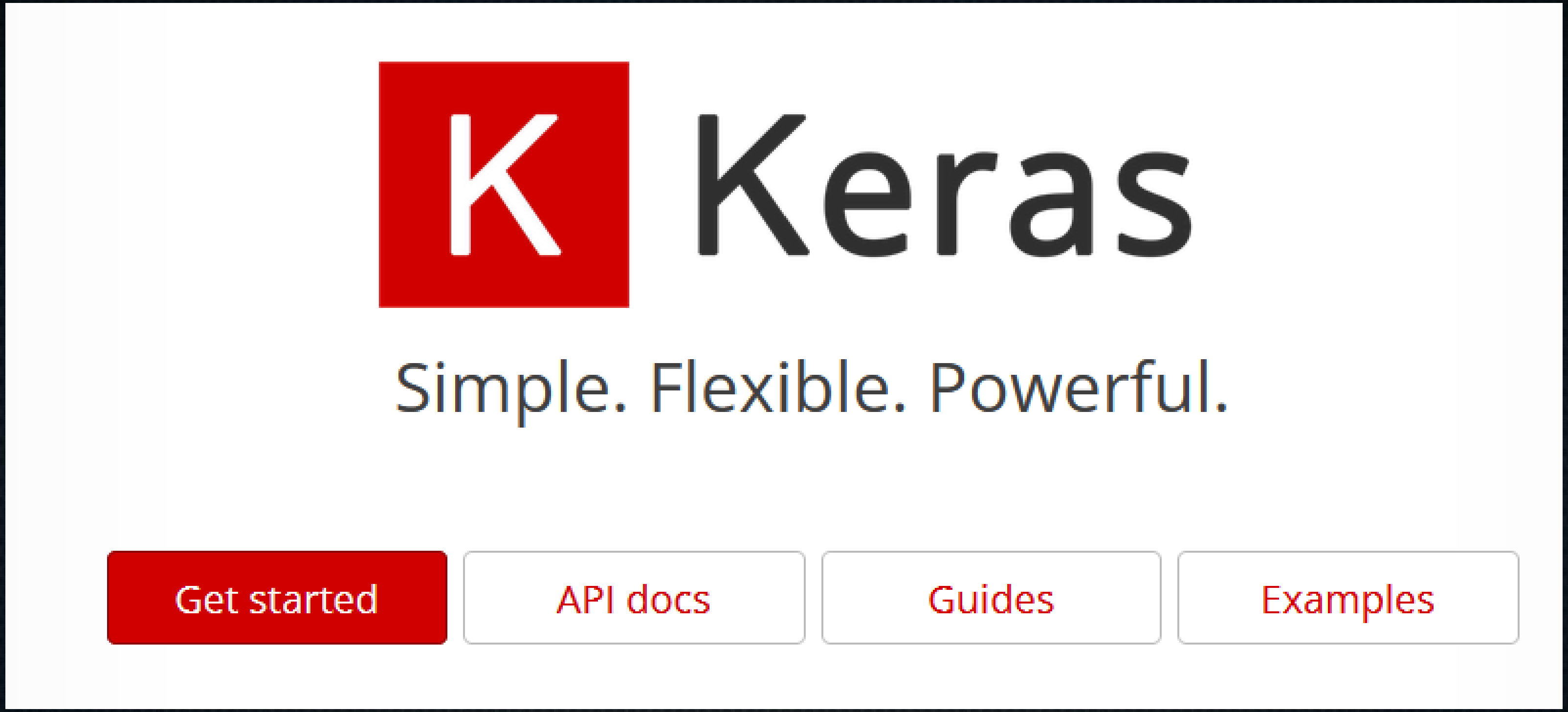
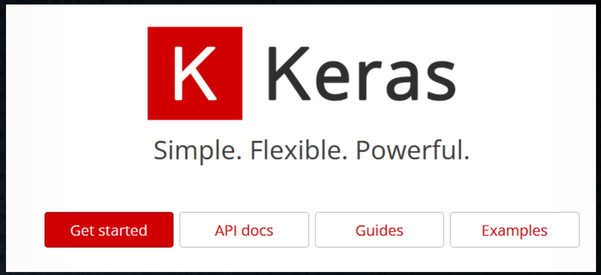
# Keras


Keras
Simple. Flexible. Powerful.
Get started
API docs
Guides
Examples

# Keras

- Similar to Scikitlearn, but for neural networks:
  - Built-in ways to preprocess and load data
  - Many layers available (dense, convolutions, MaxPool, etc.)
  - Cross-validation, dropout, hyperparameter optimalisation all supported.



# Keras example: making a model

```python
# Let's say we expect our inputs to be RGB images of arbitrary size
inputs = keras.Input(shape=(None, None, 3))
```

```python
from tensorflow.keras import layers

# Center-crop images to 150x150
x = CenterCrop(height=150, width=150)(inputs)
# Rescale images to [0, 1]
x = Rescaling(scale=1.0 / 255)(x)

# Apply some convolution and pooling layers
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(3, 3))(x)
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(3, 3))(x)
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)

# Apply global average pooling to get flat feature vectors
x = layers.GlobalAveragePooling2D()(x)

# Add a dense classifier on top
num_classes = 10
outputs = layers.Dense(num_classes, activation="softmax")(x)
```

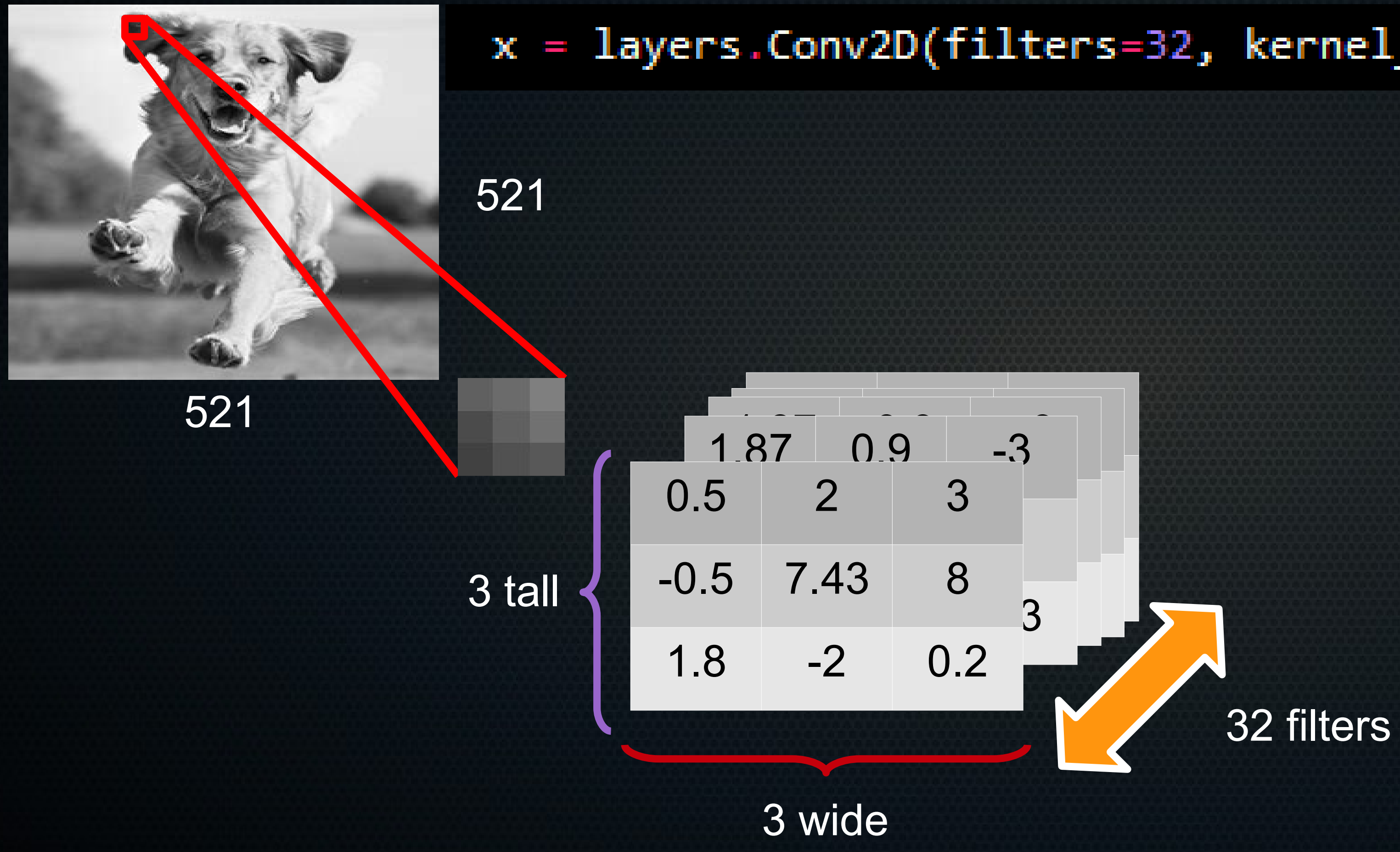
```python
model = keras.Model(inputs=inputs, outputs=outputs)
```

```python
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
```

```python
model.fit(numpy_array_of_samples, numpy_array_of_labels,
          batch_size=32, epochs=10)
```

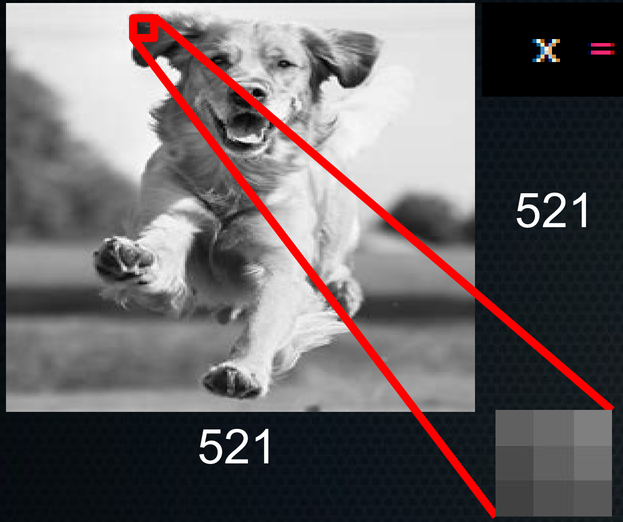
# Keras example: making a model

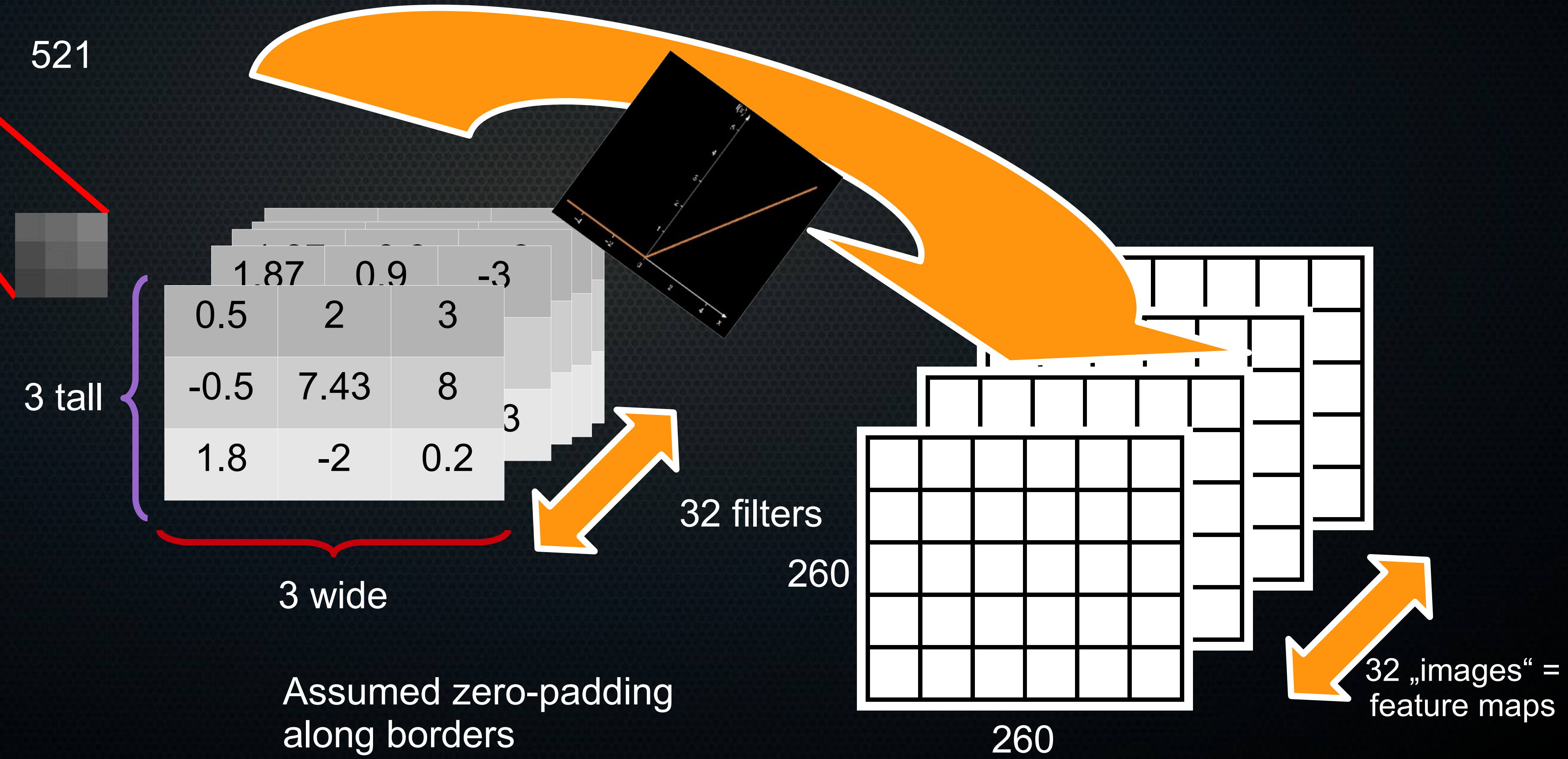
```
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)
```



# Keras example: making a model

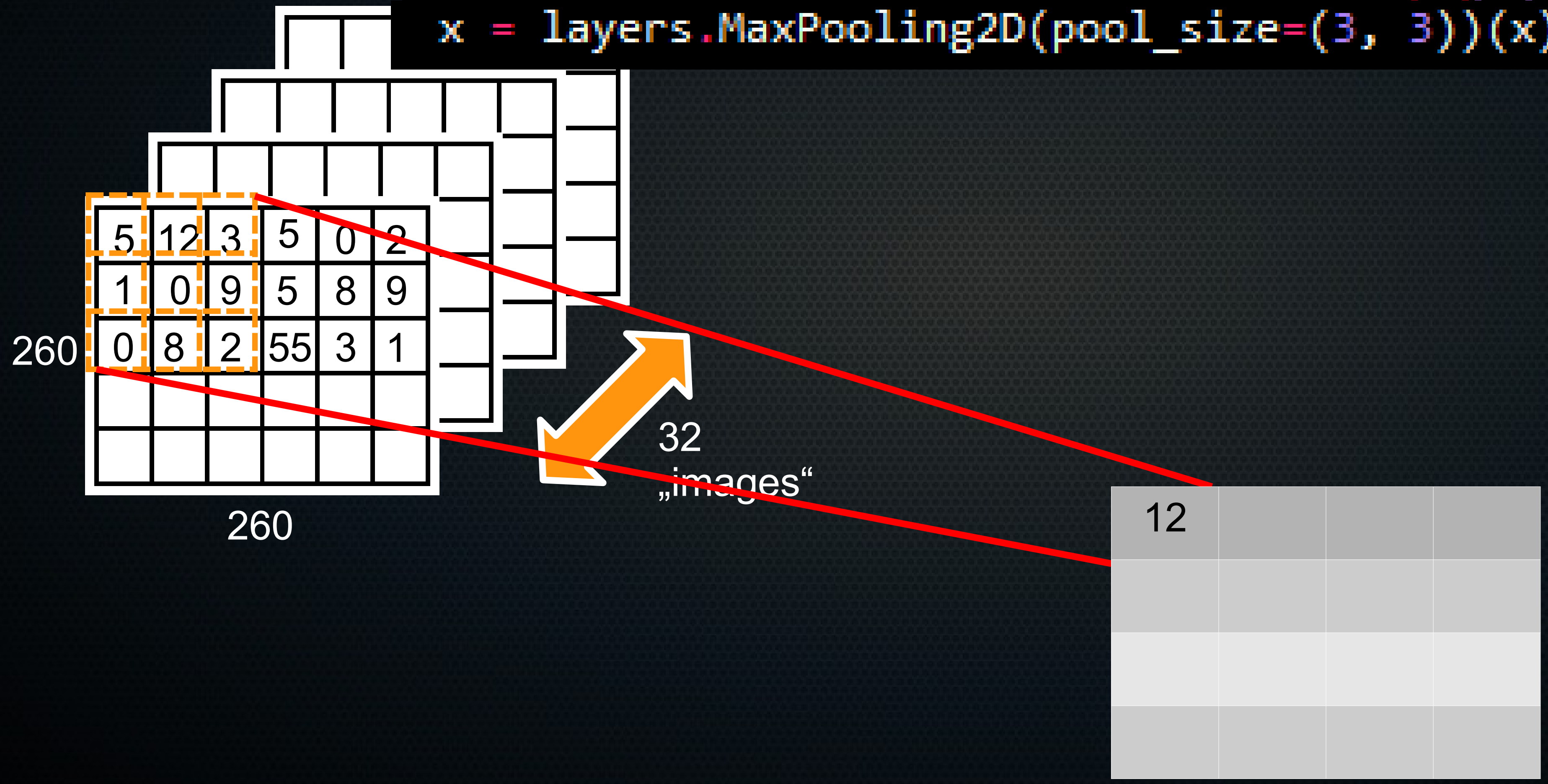
```python
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)
```



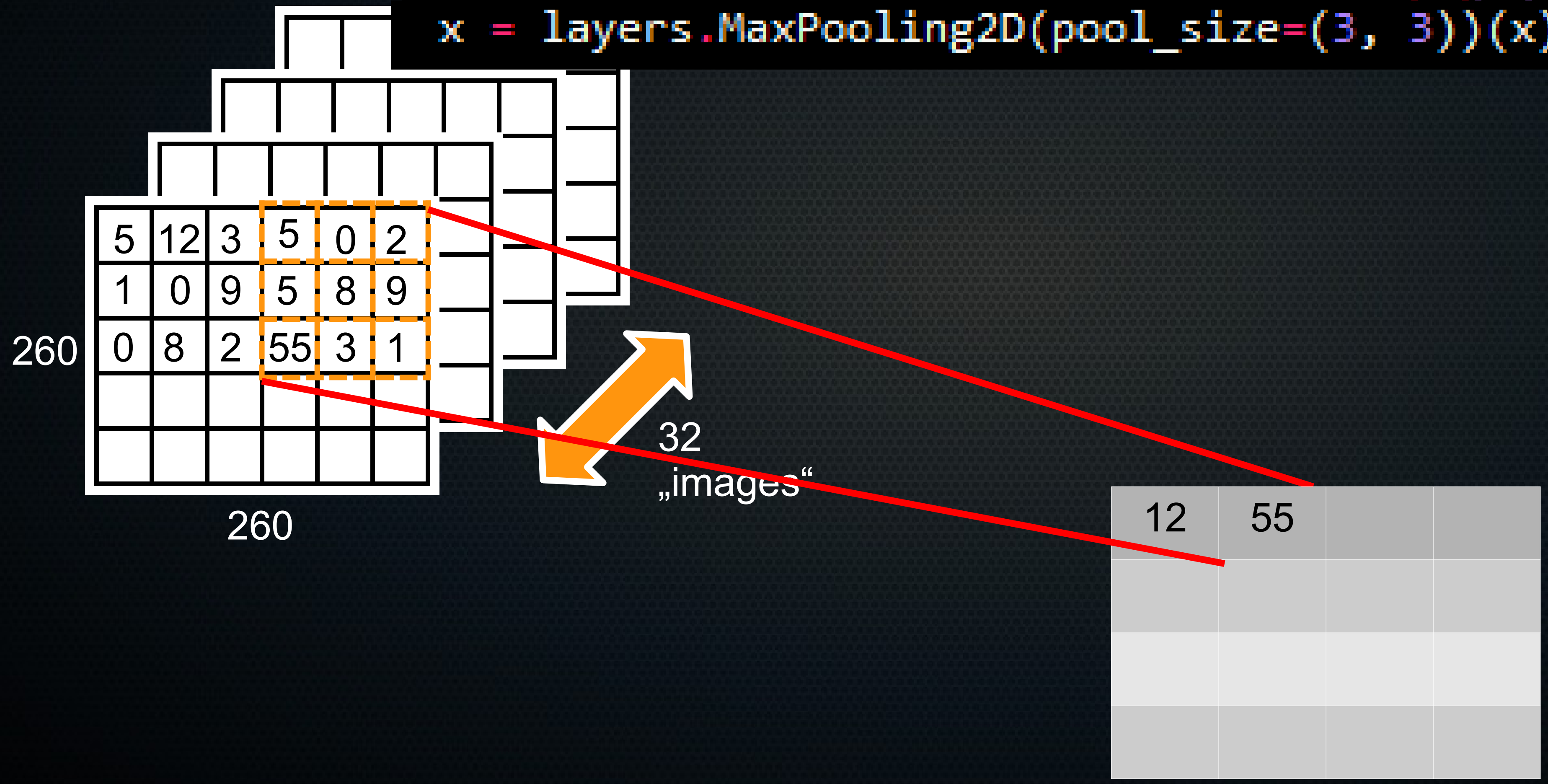
# Keras example: making a model

```
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(3, 3))(x)
```



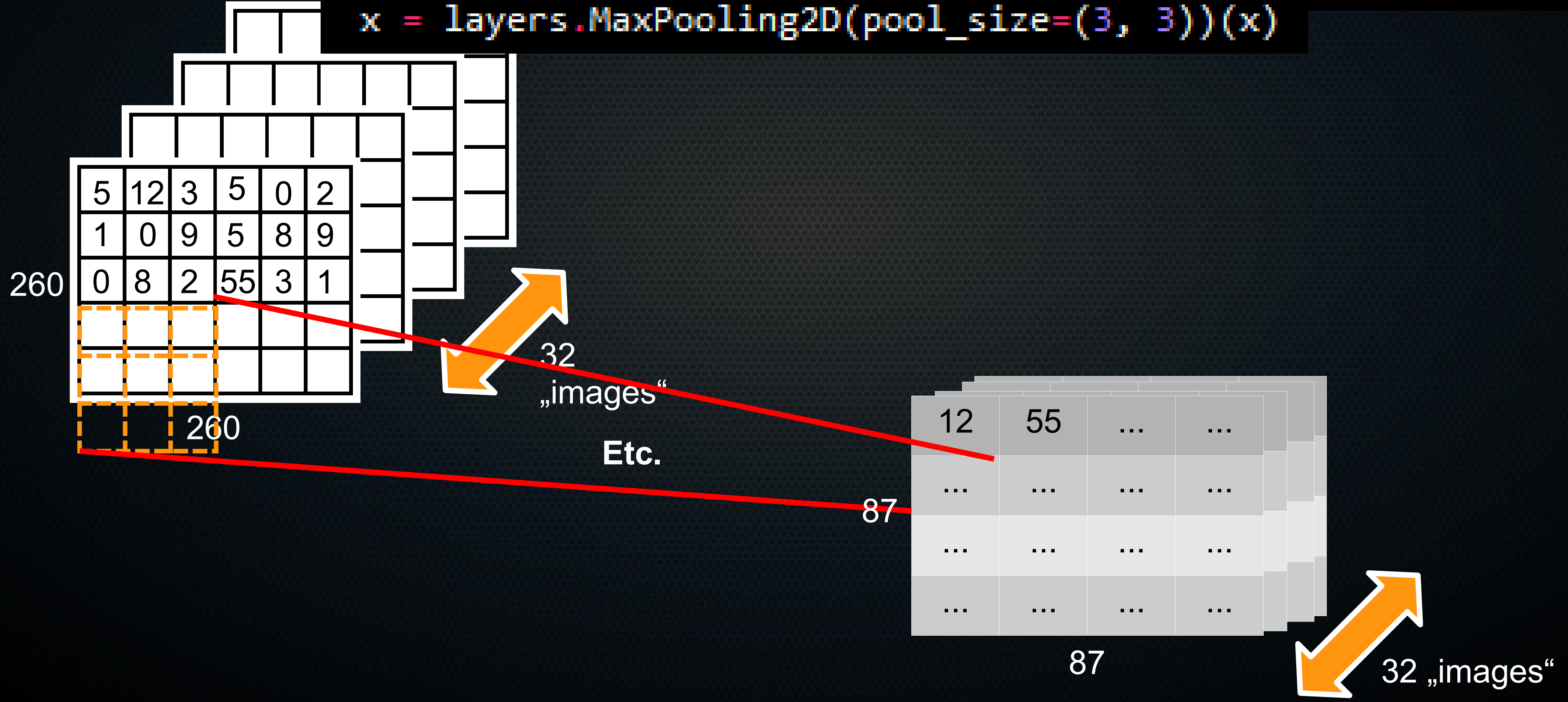
# Keras example: making a model

```
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(3, 3))(x)
```



# Keras example: making a model

```
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(3, 3))(x)
```



# Keras example: making a model

```
x = layers.GlobalAveragePooling2D()(x)
```

| | | | |
|---|---|---|---|
| 0 | 33 | 42 | 237 |
| 58 | 12 | 8 | 231 |
| 200 | 12 | 4 | 37 |
| 45 | 145 | 90 | 89 |

4

4

32 „images“

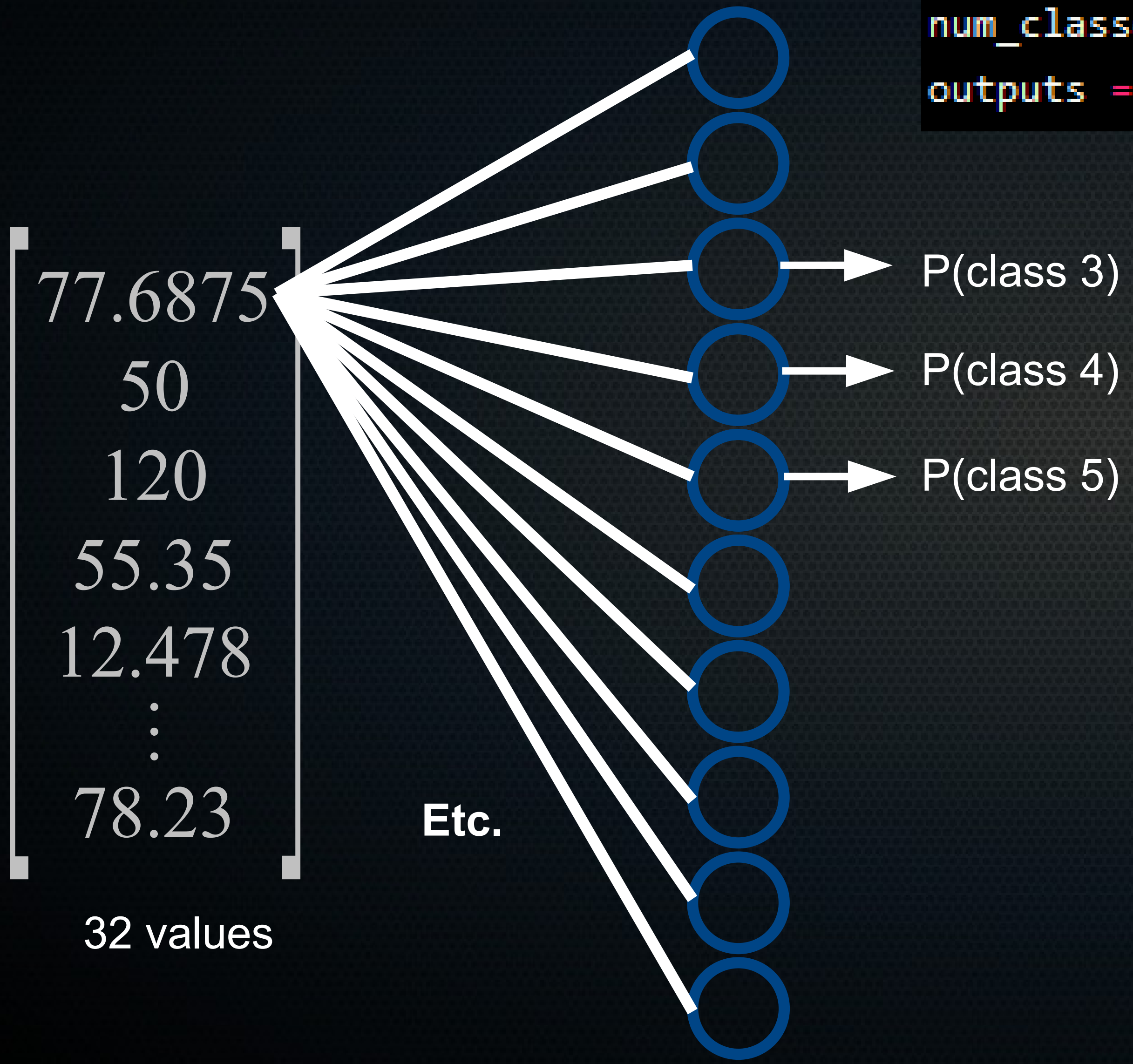
Average per feature map

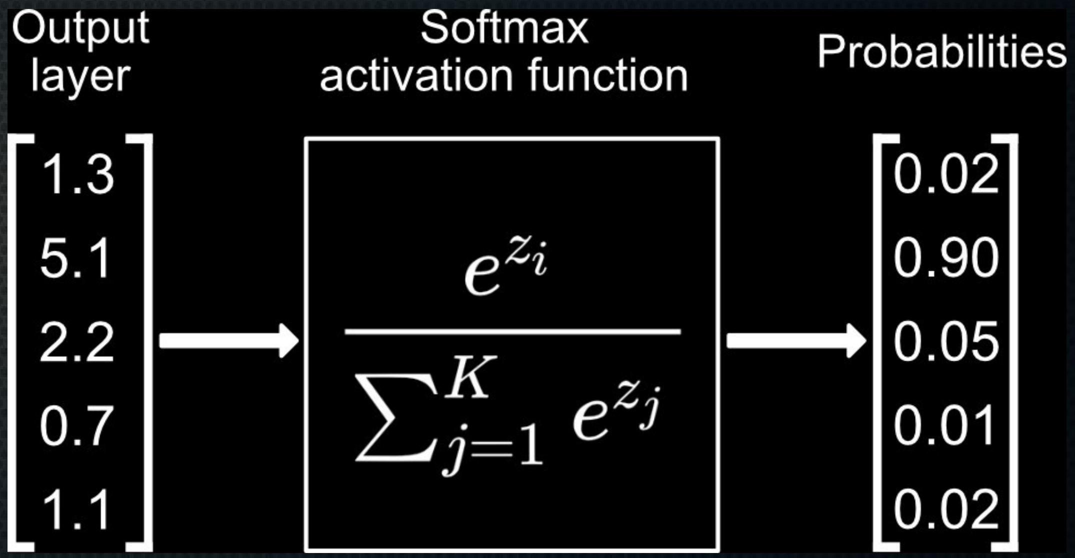
$$\begin{bmatrix} 77.6875 \\ 50 \\ 120 \\ 55.35 \\ 12.478 \\ \vdots \\ 78.23 \end{bmatrix}$$

32 values

# Keras example: making a model

```
num_classes = 10
outputs = layers.Dense(num_classes, activation="softmax")(x)
```



# Keras example: making a model

```
# Let's say we expect our inputs to be RGB images of arbitrary size
inputs = keras.Input(shape=(None, None, 3))
```

```
from tensorflow.keras import layers

# Center-crop images to 150x150
x = CenterCrop(height=150, width=150)(inputs)
# Rescale images to [0, 1]
x = Rescaling(scale=1.0 / 255)(x)

# Apply some convolution and pooling layers
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(3, 3))(x)
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(3, 3))(x)
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)

# Apply global average pooling to get flat feature vectors
x = layers.GlobalAveragePooling2D()(x)

# Add a dense classifier on top
num_classes = 10
outputs = layers.Dense(num_classes, activation="softmax")(x)
```

```
model = keras.Model(inputs=inputs, outputs=outputs)
```

```
data = np.random.randint(0, 256, size=(64, 200, 200, 3)).astype("float32")
processed_data = model(data)
print(processed_data.shape)
```

```
(64, 10)
```

# Keras example: making a model

```python
# Let's say we expect our inputs to be RGB images of arbitrary size
inputs = keras.Input(shape=(None, None, 3))
```

```python
from tensorflow.keras import layers

# Center-crop images to 150x150
x = CenterCrop(height=150, width=150)(inputs)
# Rescale images to [0, 1]
x = Rescaling(scale=1.0 / 255)(x)

# Apply some convolution and pooling layers
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(3, 3))(x)
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(3, 3))(x)
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)

# Apply global average pooling to get flat feature vectors
x = layers.GlobalAveragePooling2D()(x)

# Add a dense classifier on top
num_classes = 10
outputs = layers.Dense(num_classes, activation="softmax")(x)
```

```python
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None, None, 3)]   0
_________________________________________________________________
center_crop_1 (CenterCrop)   (None, 150, 150, 3)       0
_________________________________________________________________
rescaling_1 (Rescaling)      (None, 150, 150, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 47, 47, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 32)        9248
_________________________________________________________________
global_average_pooling2d (Gl (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 10)                330
=================================================================
Total params: 19,722
Trainable params: 19,722
Non-trainable params: 0
_________________________________________________________________
```

# Keras example: training a model

```
model.compile(optimizer=keras.optimizers.RMSprop(learning_rate=1e-3),
              loss=keras.losses.CategoricalCrossentropy())
model.fit(numpy_array_of_samples, numpy_array_of_labels,
          batch_size=32, epochs=10)
```

# Keras example: training a model

```
model.compile(optimizer=keras.optimizers.RMSprop(learning_rate=1e-3),
              loss=keras.losses.CategoricalCrossentropy())
model.fit(numpy_array_of_samples, numpy_array_of_labels,
          batch_size=32, epochs=10)
```

Step size adapative to prevent updates from exploding for large gradients or slowing to a crawl for small gradients

*For each Parameter* $w^j$

$(j \text{ subscript dropped for clarity})$

$$\nu_t = \rho \nu_{t-1} + (1 - \rho) * g_t^2$$

$$\Delta \omega_t = -\frac{\eta}{\sqrt{\nu_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta \omega_t$$

$\eta$ : *Initial Learning rate*
$\nu_t$ : *Exponential Average of squares of gradients*
$g_t$ : *Gradient at time t along* $\omega^j$

# Keras example: training a model

```
model.compile(optimizer=keras.optimizers.RMSprop(learning_rate=1e-3),
              loss=keras.losses.CategoricalCrossentropy())
model.fit(numpy_array_of_samples, numpy_array_of_labels,
          batch_size=32, epochs=10)
```

Use mini-batch gradient updating: calculate gradient not using average error on *all* training examples, but average error on *32* training examples.

# Keras example: training a model

```
model.compile(optimizer=keras.optimizers.RMSprop(learning_rate=1e-3),
              loss=keras.losses.CategoricalCrossentropy())
model.fit(numpy_array_of_samples, numpy_array_of_labels,
          batch_size=32, epochs=10)
```

Run through all the training data 10 times

(normally you would stop training when you see that performance on training set goes up but on validation set goes down → overfitting)

# Keras example: calculating more metrics during training

```
model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=[keras.metrics.SparseCategoricalAccuracy(name="acc")],
)
history = model.fit(dataset, epochs=1)
```

# Keras example: evaluate on test set after training + predicting on unlabeled data

```
loss, acc = model.evaluate(val_dataset)  # returns loss and metrics
print("loss: %.2f" % loss)
print("acc: %.2f" % acc)
```

```
157/157 [==============================] - 0s 688us/step - loss: 0.1041 - acc: 0.9692
loss: 0.10
acc: 0.97
```

```
predictions = model.predict(val_dataset)
```

# Cross-validation Keras: combine with scikit-learn!

```
from sklearn.model_selection import RepeatedKFold, cross_val_score
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor

def buildmodel():
    model= Sequential([
        Dense(10, activation="relu"),
        Dense(5, activation="relu"),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse', metrics=['mse'])
    return(model)

estimator= KerasRegressor(build_fn=buildmodel, epochs=100, batch_size=10, verbose=0)
kfold= RepeatedKFold(n_splits=5, n_repeats=100)
results= cross_val_score(estimator, x, y, cv=kfold, n_jobs=2)  # 2 cpus
results.mean()  # Mean MSE
```

# Cross-validation Keras: combine with scikit-learn!

```
from sklearn.model_selection import RepeatedKFold, cross_val_score
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor

def buildmodel():
    model= Sequential([
        Dense(10, activation="relu"),
        Dense(5, activation="relu"),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse', metrics=['mse'])
    return(model)

estimator= KerasRegressor(build_fn=buildmodel, epochs=100, batch_size=10, verbose=0)
kfold= RepeatedKFold(n_splits=5, n_repeats=100)
results= cross_val_score(estimator, x, y, cv=kfold, n_jobs=2)  # 2 cpus
results.mean()  # Mean MSE
```

Function that returns a valid Keras model

# Cross-validation Keras: combine with scikit-learn!

```
from sklearn.model_selection import RepeatedKFold, cross_val_score
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor

def buildmodel():
    model= Sequential([
        Dense(10, activation="relu"),
        Dense(5, activation="relu"),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse', metrics=['mse'])
    return(model)

estimator= KerasRegressor(build_fn=buildmodel, epochs=100, batch_size=10, verbose=0)
kfold= RepeatedKFold(n_splits=5, n_repeats=100)
results= cross_val_score(estimator, x, y, cv=kfold, n_jobs=2)  # 2 cpus
results.mean()  # Mean MSE
```

Wrapper that instantiates a Keras model as a scikit-learn regressor object (with a .fit() and .predict() method, etc.)

# Cross-validation Keras: combine with scikit-learn!

```
from sklearn.model_selection import RepeatedKFold, cross_val_score
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor

def buildmodel():
    model= Sequential([
        Dense(10, activation="relu"),
        Dense(5, activation="relu"),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse', metrics=['mse'])
    return(model)

estimator= KerasRegressor(build_fn=buildmodel, epochs=100, batch_size=10, verbose=0)
kfold= RepeatedKFold(n_splits=5, n_repeats=100)
results= cross_val_score(estimator, x, y, cv=kfold, n_jobs=2)  # 2 cpus
results.mean()  # Mean MSE
```

Split data in 5 folds 100 times, for each train the model on 4 folds and validate on 1.

Show average mean-squared error over these 100 5-fold cross-validations.

# Hyperparameter tuning Keras

```
def build_model(hp):
    inputs = keras.Input(shape=(784,))
    x = layers.Dense(
        units=hp.Int('units', min_value=32, max_value=512, step=32),
        activation='relu'))(inputs)
    outputs = layers.Dense(10, activation='softmax')(x)
    model = keras.Model(inputs, outputs)
    model.compile(
        optimizer=keras.optimizers.Adam(
            hp.Choice('learning_rate',
                      values=[1e-2, 1e-3, 1e-4])),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])
    return model
```

Function that builds your model.
Take argument hyperparameter (hp).

# Hyperparameter tuning Keras

```
def build_model(hp):
    inputs = keras.Input(shape=(784,))
    x = layers.Dense(
        units=hp.Int('units', min_value=32, max_value=512, step=32),
        activation='relu'))(inputs)
    outputs = layers.Dense(10, activation='softmax')(x)
    model = keras.Model(inputs, outputs)
    model.compile(
        optimizer=keras.optimizers.Adam(
            hp.Choice('learning_rate',
                      values=[1e-2, 1e-3, 1e-4])),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])
    return model
```

For number of neurons, search from 32 to 512 neurons in steps of 32.

Range of numbers = hp.Int()

# Hyperparameter tuning Keras

```
def build_model(hp):
    inputs = keras.Input(shape=(784,))
    x = layers.Dense(
        units=hp.Int('units', min_value=32, max_value=512, step=32),
        activation='relu'))(inputs)
    outputs = layers.Dense(10, activation='softmax')(x)
    model = keras.Model(inputs, outputs)
    model.compile(
        optimizer=keras.optimizers.Adam(
            hp.Choice('learning_rate',
                      values=[1e-2, 1e-3, 1e-4])),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])
    return model
```

For learning rate in the Adam optimizer, pick from 3 specific options.

Specific options = hp.Choice()

# Hyperparameter tuning Keras

```
import keras_tuner

tuner = keras_tuner.tuners.Hyperband(
  build_model,
  objective='val_loss',
  max_epochs=100,
  max_trials=200,
  executions_per_trial=2,
  directory='my_dir')
```

max_trials = how many different models you try maximally.

executions_per_trial: how many times you train a model with the same parameters
→ since neural net weights and biases are randomly instantiated, training the same model twice might not get the exact same outcome
→ reduce variance in model performance measure.

# Hyperparameter tuning Keras

```
import keras_tuner

tuner = keras_tuner.tuners.Hyperband(
  build_model,
  objective='val_loss',
  max_epochs=100,
  max_trials=200,
  executions_per_trial=2,
  directory='my_dir')
```

Get the best fit

```
tuner.search(x_train, y_train, epochs=2, validation_data=(x_val, y_val))
```

```
models = tuner.get_best_models(num_models=2)
```

# Hyperparameter tuning Keras

```
tuner.results_summary()
```

```
Results summary
Results in my_dir/helloworld
Showing 10 best trials
Objective(name='val_accuracy', direction='max')
Trial summary
Hyperparameters:
units: 480
learning_rate: 0.001
Score: 0.9730499982833862
Trial summary
Hyperparameters:
units: 160
learning_rate: 0.001
Score: 0.9692499935626984
Trial summary
Hyperparameters:
units: 320
learning_rate: 0.0001
Score: 0.9421000182628632
```

See a summary of the (top N) hyperparameter choices

(here for a tuner with max_trials = 3, so just 3)

# Break for practical 2

- Guided exercises where they:
  - Reimplement the simple neural network they made on Wednesday the previous week (hopefully) in Keras, train it and predict on some new data
  - Perform hyperparameter optimisation (i.e. full training pipeline with bells and whistles) for MNIST dataset classification (starting from some okay convolutional architecture).