**Original **

Importing IMDB Dataset

```
import keras
```

```
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

```
<string>:6: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:159: Vis
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:160: Vis
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

+ Code + Text

```
max([max(sequence) for sequence in train_data])
```

```
9999
```

Preparing Data

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    # Create an all-zero matrix of shape (len(sequences), dimension)
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.  # set specific indices of results[i] to 1s
    return results

# Our vectorized training data
x_train = vectorize_sequences(train_data)
# Our vectorized test data
x_test = vectorize_sequences(test_data)
```

```
x_train[0]
```

```
array([0., 1., 1., ..., 0., 0., 0.])
```

```
# Our vectorized labels
```

```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

Building the Network

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))


from keras import optimizers
from keras import losses
from keras import metrics

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses.binary_crossentropy,
              metrics=['accuracy'])
```

Validating the Approach

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]


history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 2s 42ms/step - loss: 0.6023 - accuracy: 0.6755
Epoch 2/20
30/30 [==============================] - 1s 32ms/step - loss: 0.3343 - accuracy: 0.8932
Epoch 3/20
30/30 [==============================] - 1s 32ms/step - loss: 0.2328 - accuracy: 0.9276
Epoch 4/20
30/30 [==============================] - 1s 31ms/step - loss: 0.1837 - accuracy: 0.9425
Epoch 5/20
30/30 [==============================] - 1s 32ms/step - loss: 0.1442 - accuracy: 0.9563
Epoch 6/20
```

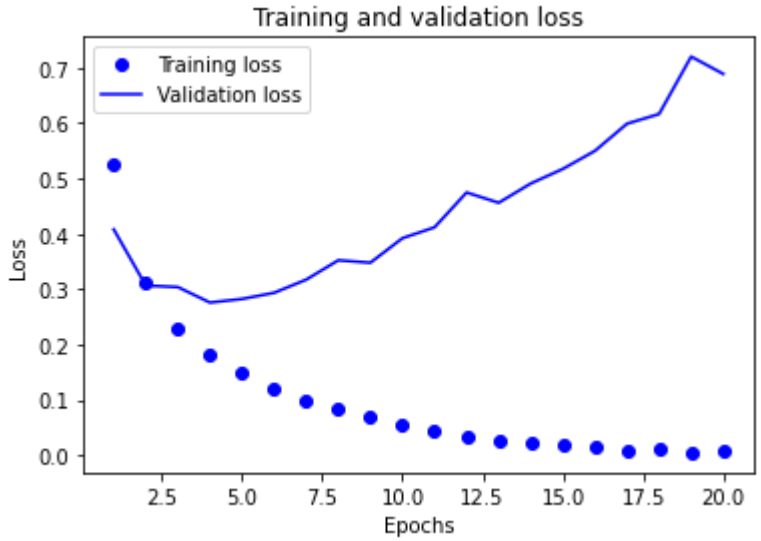
```
30/30 [==============================] - 1s 32ms/step - loss: 0.1194 - accuracy: 0.9661
Epoch 7/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0980 - accuracy: 0.9736
Epoch 8/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0804 - accuracy: 0.9773
Epoch 9/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0616 - accuracy: 0.9867
Epoch 10/20
30/30 [==============================] - 1s 31ms/step - loss: 0.0506 - accuracy: 0.9899
Epoch 11/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0397 - accuracy: 0.9933
Epoch 12/20
30/30 [==============================] - 1s 31ms/step - loss: 0.0342 - accuracy: 0.9932
Epoch 13/20
30/30 [==============================] - 1s 31ms/step - loss: 0.0259 - accuracy: 0.9965
Epoch 14/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0212 - accuracy: 0.9971
Epoch 15/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0163 - accuracy: 0.9985
Epoch 16/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0112 - accuracy: 0.9991
Epoch 17/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0075 - accuracy: 0.9997
Epoch 18/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0086 - accuracy: 0.9987
Epoch 19/20
30/30 [==============================] - 1s 31ms/step - loss: 0.0044 - accuracy: 0.9998
Epoch 20/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0059 - accuracy: 0.9989
```

```
history_dict = history.history
history_dict.keys()
```

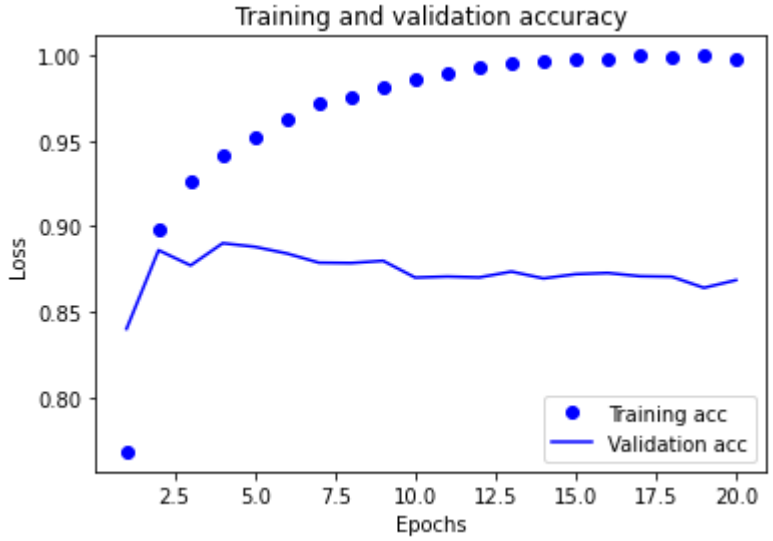
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
import matplotlib.pyplot as plt

accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(accuracy) + 1)

# "bo" is for "blue dot"
plt.plot(epochs, loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
plt.show()
```




```
plt.clf()   # clear figure
accuracy_values = history_dict['accuracy']
val_accuracy_values = history_dict['val_accuracy']

plt.plot(epochs, accuracy, 'bo', label='Training acc')
plt.plot(epochs, val_accuracy, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```




```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(x_val, y_val, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
20/20 [==============================] - 1s 24ms/step - loss: 0.6114 - accuracy: 0.6688
Epoch 2/4
20/20 [==============================] - 1s 25ms/step - loss: 0.3546 - accuracy: 0.9019
Epoch 3/4
20/20 [==============================] - 0s 24ms/step - loss: 0.2488 - accuracy: 0.9349
Epoch 4/4
20/20 [==============================] - 0s 23ms/step - loss: 0.1929 - accuracy: 0.9485
782/782 [==============================] - 1s 2ms/step - loss: 0.3029 - accuracy: 0.8793
```

```
results
```

```
[0.30288684368133545, 0.8793200254440308]
```

Tensorboard and Callbacks

Importing IMDB Dataset

```
%tensorflow_version 2.x
%load_ext tensorboard
import keras
import tensorflow as tf
import datetime
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

```
<string>:6: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:159: Vis
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:160: Vis
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

```
max([max(sequence) for sequence in train_data])
```

```
9999
```

Preparing the Data

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    # Create an all-zero matrix of shape (len(sequences), dimension)
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.  # set specific indices of results[i] to 1s
    return results

# Our vectorized training data
x_train = vectorize_sequences(train_data)
# Our vectorized test data
x_test = vectorize_sequences(test_data)
```

```
x_train[0]
```

```
array([0., 1., 1., ..., 0., 0., 0.])
```

```
# Our vectorized labels
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

Building the Network

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
from keras import optimizers
from keras import losses
from keras import metrics

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses.binary_crossentropy,
```

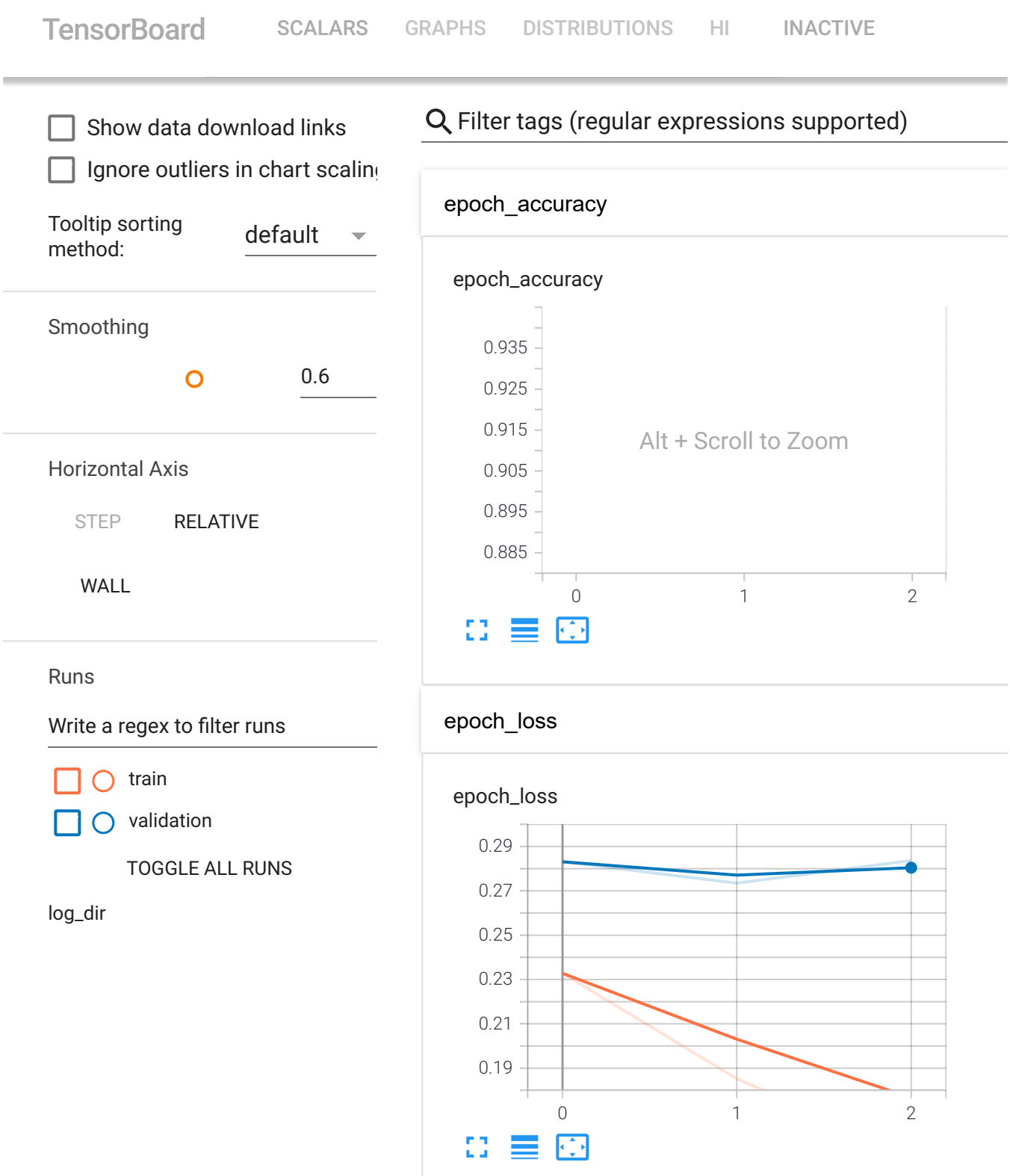
```
              metrics=['accuracy'])

log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

#mkdir my_log_dir
```

Validating the Apporach

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]


callbacks = [
             keras.callbacks.TensorBoard(
                 log_dir = 'log_dir',
                 histogram_freq=1,
                 embeddings_freq=1
             ),
             keras.callbacks.EarlyStopping(
                 monitor='val_accuracy',
                 patience=1
             ),
             keras.callbacks.ModelCheckpoint(
                 filepath='ML bpalazzo_5.ipynb',
                 monitor='val_accuracy',
                 save_best_only=True
             )
]


history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val),
                    callbacks=callbacks)
```

```
Epoch 1/20
30/30 [==============================] - 2s 52ms/step - loss: 0.2328 - accuracy: 0.9243
INFO:tensorflow:Assets written to: ML bpalazzo_5.ipynb/assets
Epoch 2/20
30/30 [==============================] - 1s 36ms/step - loss: 0.1852 - accuracy: 0.9414
INFO:tensorflow:Assets written to: ML bpalazzo_5.ipynb/assets
Epoch 3/20
30/30 [==============================] - 1s 36ms/step - loss: 0.1514 - accuracy: 0.9521
```

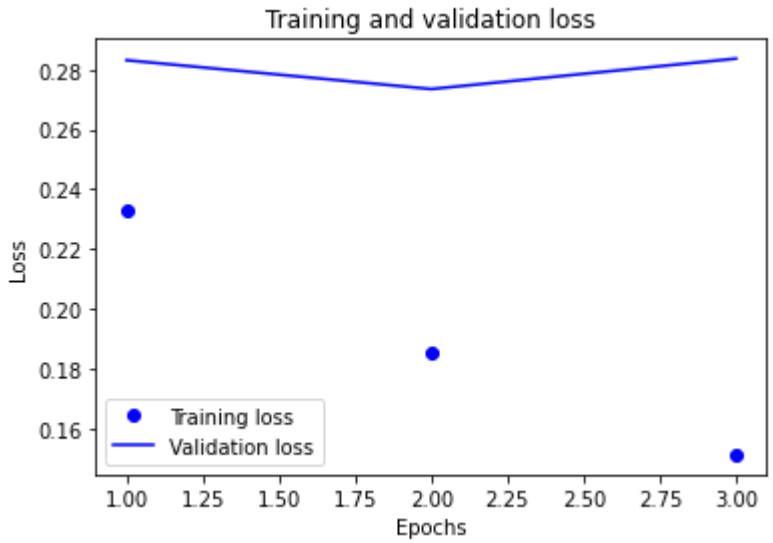
```
%tensorboard --logdir log_dir
```



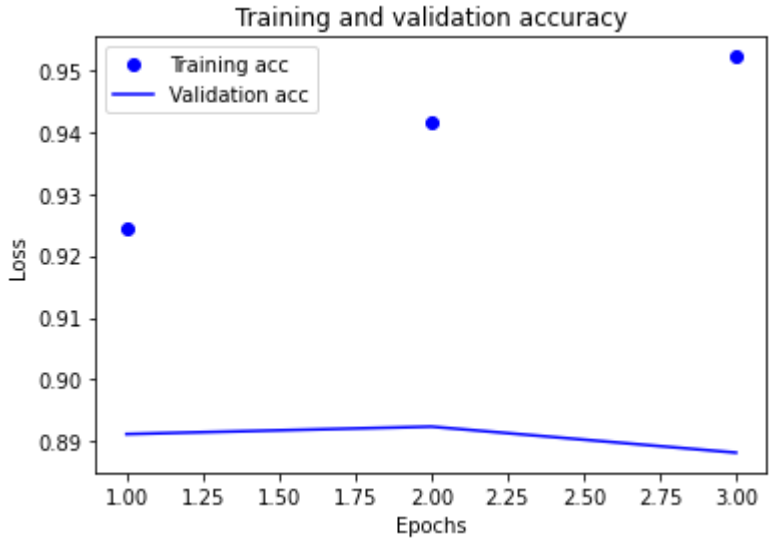

```
history_dict = history.history
history_dict.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
import matplotlib.pyplot as plt

accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(accuracy) + 1)

# "bo" is for "blue dot"
plt.plot(epochs, loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```




```
plt.clf()   # clear figure
accuracy_values = history_dict['accuracy']
val_accuracy_values = history_dict['val_accuracy']

plt.plot(epochs, accuracy, 'bo', label='Training acc')
plt.plot(epochs, val_accuracy, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```




```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(x_val, y_val, epochs=3, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/3
20/20 [==============================] - 1s 28ms/step - loss: 0.6479 - accuracy: 0.6688
Epoch 2/3
20/20 [==============================] - 1s 27ms/step - loss: 0.4408 - accuracy: 0.8879
Epoch 3/3
20/20 [==============================] - 1s 26ms/step - loss: 0.3143 - accuracy: 0.9175
782/782 [==============================] - 1s 2ms/step - loss: 0.3412 - accuracy: 0.8751
```

```
results
```

```
[0.34116330742836, 0.8750799894332886]
```

✓ 0s completed at 4:10 PM