Kayvan_Shah_Assignment5

April 3, 2023

1 Installing & Importing required Modules & Libraries

[2]:
```
!pip install ipython-autotime
%matplotlib inline
%load_ext autotime
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting ipython-autotime
  Downloading ipython_autotime-0.3.1-py2.py3-none-any.whl (6.8 kB)
Requirement already satisfied: ipython in /usr/local/lib/python3.9/dist-packages
(from ipython-autotime) (7.34.0)
Collecting jedi>=0.16
  Downloading jedi-0.18.2-py2.py3-none-any.whl (1.6 MB)
                           1.6/1.6 MB
21.6 MB/s eta 0:00:00
Requirement already satisfied: matplotlib-inline in
/usr/local/lib/python3.9/dist-packages (from ipython->ipython-autotime) (0.1.6)
Requirement already satisfied: pygments in /usr/local/lib/python3.9/dist-
packages (from ipython->ipython-autotime) (2.14.0)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in
/usr/local/lib/python3.9/dist-packages (from ipython->ipython-autotime) (3.0.38)
Requirement already satisfied: decorator in /usr/local/lib/python3.9/dist-
packages (from ipython->ipython-autotime) (4.4.2)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.9/dist-
packages (from ipython->ipython-autotime) (4.8.0)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.9/dist-
packages (from ipython->ipython-autotime) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.9/dist-
packages (from ipython->ipython-autotime) (5.7.1)
Requirement already satisfied: backcall in /usr/local/lib/python3.9/dist-
packages (from ipython->ipython-autotime) (0.2.0)
Requirement already satisfied: setuptools>=18.5 in
/usr/local/lib/python3.9/dist-packages (from ipython->ipython-autotime) (67.6.1)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in
/usr/local/lib/python3.9/dist-packages (from jedi>=0.16->ipython->ipython-
autotime) (0.8.3)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.9/dist-
```

```
packages (from pexpect>4.3->ipython->ipython-autotime) (0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.9/dist-packages
(from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython->ipython-autotime)
(0.2.6)
Installing collected packages: jedi, ipython-autotime
Successfully installed ipython-autotime-0.3.1 jedi-0.18.2
time: 336 µs (started: 2023-04-03 06:04:55 +00:00)
```

[3]:

```python
from pprint import pprint

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn.metrics import classification_report

import tensorflow as tf
```

```
time: 4.62 s (started: 2023-04-03 06:04:55 +00:00)
```

2 TPU Configuration

[4]:

```python
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Running on TPU ', tpu.master())
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.TPUStrategy(tpu)
else:
    strategy = tf.distribute.get_strategy()

print("REPLICAS: ", strategy.num_replicas_in_sync)
```

```
Running on TPU  grpc://10.85.248.106:8470
REPLICAS:  8
time: 12.9 s (started: 2023-04-03 06:04:59 +00:00)
```

3 Data Preparation

[5]:
```
# Load data
fashion_mnist = tf.keras.datasets.fashion_mnist.load_data()

# Train, Validation & Test Split
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist
X_train, y_train = X_train_full[:-5000], y_train_full[:-5000]
X_valid, y_valid = X_train_full[-5000:], y_train_full[-5000:]

# Scale the Dataset
X_train, X_valid, X_test = X_train / 255., X_valid / 255., X_test / 255.
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
time: 3.4 s (started: 2023-04-03 06:05:12 +00:00)
```

4 Exploratory Data Analysis

[6]:
```
print("Train size:", X_train.shape)
print("Validation size:", X_valid.shape)
print("Test size:", X_test.shape)
```

```
Train size: (55000, 28, 28)
Validation size: (5000, 28, 28)
Test size: (10000, 28, 28)
time: 4.02 ms (started: 2023-04-03 06:05:16 +00:00)
```

[7]:
```
class_names = [
    'T-shirt/top', 'Trouser', 'Pullover',
    'Dress', 'Coat','Sandal', 'Shirt',
    'Sneaker', 'Bag', 'Ankle boot'
]
```

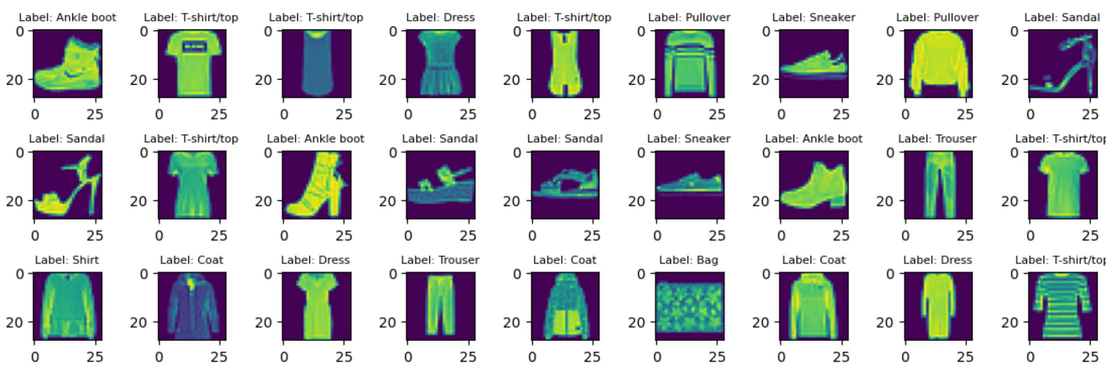
```
time: 620 µs (started: 2023-04-03 06:05:16 +00:00)
```

[8]:
```python
def plot_mnist_data(num_row = 3, num_col = 5):
    # get a segment of the dataset
    num = num_row*num_col
    images = X_train[:num]
    labels = y_train[:num]

    # plot images
    fig, axes = plt.subplots(num_row, num_col, figsize=(1.25*num_col,1.
↪25*num_row))
    for i in range(num_row*num_col):
        ax = axes[i//num_col, i%num_col]
        ax.imshow(images[i])
        ax.set_title('Label: {}'.format(class_names[labels[i]]), fontsize=8)
    plt.tight_layout()
    plt.show()
```

time: 5.4 ms (started: 2023-04-03 06:05:16 +00:00)

[9]:
```python
plot_mnist_data(num_row = 3, num_col = 9)
```



time: 11.3 s (started: 2023-04-03 06:05:16 +00:00)

5 Converting to Tensorflow dataset

[10]:
```python
AUTO = tf.data.experimental.AUTOTUNE
BATCH_SIZE = 16 * strategy.num_replicas_in_sync
```

time: 666 µs (started: 2023-04-03 06:05:27 +00:00)

[11]:
```python
train = (
    tf.data.Dataset
    .from_tensor_slices((X_train, y_train))
    .repeat()
```

```
    .shuffle(128)
    .batch(BATCH_SIZE)
    .prefetch(AUTO)
)

valid = (
    tf.data.Dataset
    .from_tensor_slices((X_valid, y_valid))
    .batch(BATCH_SIZE)
    .cache()
    # .repeat()
    .shuffle(64)
    .prefetch(AUTO)
)

test = (
    tf.data.Dataset
    .from_tensor_slices((X_test, y_test))
    .batch(BATCH_SIZE)
    .cache()
    .prefetch(AUTO)
)
```

```
time: 7.64 s (started: 2023-04-03 06:05:27 +00:00)
```

6 Train Models

6.1 Part One

6.1.1 Fashion MNIST Neural Network

Follow the instructions in Chapter 10 of Aurelien (Hands-on Machine Learning) to create a fourlayer neural network (1 Flatten Layer and 3 Dense Layers) and train it on the Fashion MNIST dataset.

What to turn in: - The CPU Times and Wall Times returned by fit() from the training process - Generate loss and accuracy versus epoch plots (see Figure 10-11) - The accuracy, precision and recall on the test test - The precision and recall values by class_id on the test test. There are 10 classes.

6.1.2 Build Model

[14]:
```
def build_basic_model():
    tf.random.set_seed(42)
    model = tf.keras.Sequential([
        tf.keras.layers.Flatten(input_shape=[28, 28]),
        tf.keras.layers.Dense(300, activation="relu"),
        tf.keras.layers.Dense(100, activation="relu"),
        tf.keras.layers.Dense(10, activation="softmax")
```

```
    ])

    model.compile(
        loss="sparse_categorical_crossentropy",
        optimizer="sgd",
        metrics=[tf.keras.metrics.sparse_categorical_accuracy]
    )
    return model
```

```
time: 932 µs (started: 2023-04-03 06:06:54 +00:00)
```

[15]:
```
# Clear backend sessions to reset name counters
tf.keras.backend.clear_session()

with strategy.scope():
    model = build_basic_model()

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 300)               235500

 dense_1 (Dense)             (None, 100)               30100

 dense_2 (Dense)             (None, 10)                1010

=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
time: 269 ms (started: 2023-04-03 06:06:56 +00:00)
```

6.1.3 Train Model

[16]:
```
EPOCHS = 50

n_steps = X_train.shape[0] // BATCH_SIZE
```

```
time: 640 µs (started: 2023-04-03 06:07:52 +00:00)
```

[17]:
```
%%time
train_history = model.fit(
    train,
    steps_per_epoch=n_steps,
    validation_data=valid,
    epochs=EPOCHS
)
```

```
Epoch 1/50
429/429 [==============================] - 11s 20ms/step - loss: 1.0817 -
sparse_categorical_accuracy: 0.6757 - val_loss: 0.7024 -
val_sparse_categorical_accuracy: 0.7718
Epoch 2/50
429/429 [==============================] - 8s 19ms/step - loss: 0.6432 -
sparse_categorical_accuracy: 0.7903 - val_loss: 0.5739 -
val_sparse_categorical_accuracy: 0.8100
Epoch 3/50
429/429 [==============================] - 8s 18ms/step - loss: 0.5583 -
sparse_categorical_accuracy: 0.8127 - val_loss: 0.5188 -
val_sparse_categorical_accuracy: 0.8224
Epoch 4/50
429/429 [==============================] - 8s 18ms/step - loss: 0.5159 -
sparse_categorical_accuracy: 0.8250 - val_loss: 0.4940 -
val_sparse_categorical_accuracy: 0.8298
Epoch 5/50
429/429 [==============================] - 7s 17ms/step - loss: 0.4902 -
sparse_categorical_accuracy: 0.8333 - val_loss: 0.4818 -
val_sparse_categorical_accuracy: 0.8314
Epoch 6/50
429/429 [==============================] - 8s 18ms/step - loss: 0.4723 -
sparse_categorical_accuracy: 0.8383 - val_loss: 0.4578 -
val_sparse_categorical_accuracy: 0.8414
Epoch 7/50
429/429 [==============================] - 8s 18ms/step - loss: 0.4591 -
sparse_categorical_accuracy: 0.8425 - val_loss: 0.4544 -
val_sparse_categorical_accuracy: 0.8376
Epoch 8/50
429/429 [==============================] - 8s 19ms/step - loss: 0.4478 -
sparse_categorical_accuracy: 0.8459 - val_loss: 0.4396 -
val_sparse_categorical_accuracy: 0.8428
Epoch 9/50
429/429 [==============================] - 8s 18ms/step - loss: 0.4386 -
sparse_categorical_accuracy: 0.8490 - val_loss: 0.4453 -
val_sparse_categorical_accuracy: 0.8364
Epoch 10/50
429/429 [==============================] - 7s 17ms/step - loss: 0.4297 -
sparse_categorical_accuracy: 0.8526 - val_loss: 0.4246 -
```

```
val_sparse_categorical_accuracy: 0.8494
Epoch 11/50
429/429 [==============================] - 8s 18ms/step - loss: 0.4224 -
sparse_categorical_accuracy: 0.8544 - val_loss: 0.4179 -
val_sparse_categorical_accuracy: 0.8496
Epoch 12/50
429/429 [==============================] - 7s 17ms/step - loss: 0.4155 -
sparse_categorical_accuracy: 0.8560 - val_loss: 0.4133 -
val_sparse_categorical_accuracy: 0.8524
Epoch 13/50
429/429 [==============================] - 8s 18ms/step - loss: 0.4095 -
sparse_categorical_accuracy: 0.8591 - val_loss: 0.4259 -
val_sparse_categorical_accuracy: 0.8508
Epoch 14/50
429/429 [==============================] - 7s 17ms/step - loss: 0.4039 -
sparse_categorical_accuracy: 0.8604 - val_loss: 0.4016 -
val_sparse_categorical_accuracy: 0.8596
Epoch 15/50
429/429 [==============================] - 7s 17ms/step - loss: 0.3989 -
sparse_categorical_accuracy: 0.8615 - val_loss: 0.3998 -
val_sparse_categorical_accuracy: 0.8588
Epoch 16/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3949 -
sparse_categorical_accuracy: 0.8629 - val_loss: 0.3947 -
val_sparse_categorical_accuracy: 0.8600
Epoch 17/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3886 -
sparse_categorical_accuracy: 0.8661 - val_loss: 0.3905 -
val_sparse_categorical_accuracy: 0.8582
Epoch 18/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3846 -
sparse_categorical_accuracy: 0.8669 - val_loss: 0.3864 -
val_sparse_categorical_accuracy: 0.8616
Epoch 19/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3802 -
sparse_categorical_accuracy: 0.8683 - val_loss: 0.3829 -
val_sparse_categorical_accuracy: 0.8628
Epoch 20/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3767 -
sparse_categorical_accuracy: 0.8694 - val_loss: 0.3850 -
val_sparse_categorical_accuracy: 0.8634
Epoch 21/50
429/429 [==============================] - 7s 17ms/step - loss: 0.3721 -
sparse_categorical_accuracy: 0.8712 - val_loss: 0.3814 -
val_sparse_categorical_accuracy: 0.8648
Epoch 22/50
429/429 [==============================] - 9s 20ms/step - loss: 0.3696 -
sparse_categorical_accuracy: 0.8715 - val_loss: 0.3784 -
```

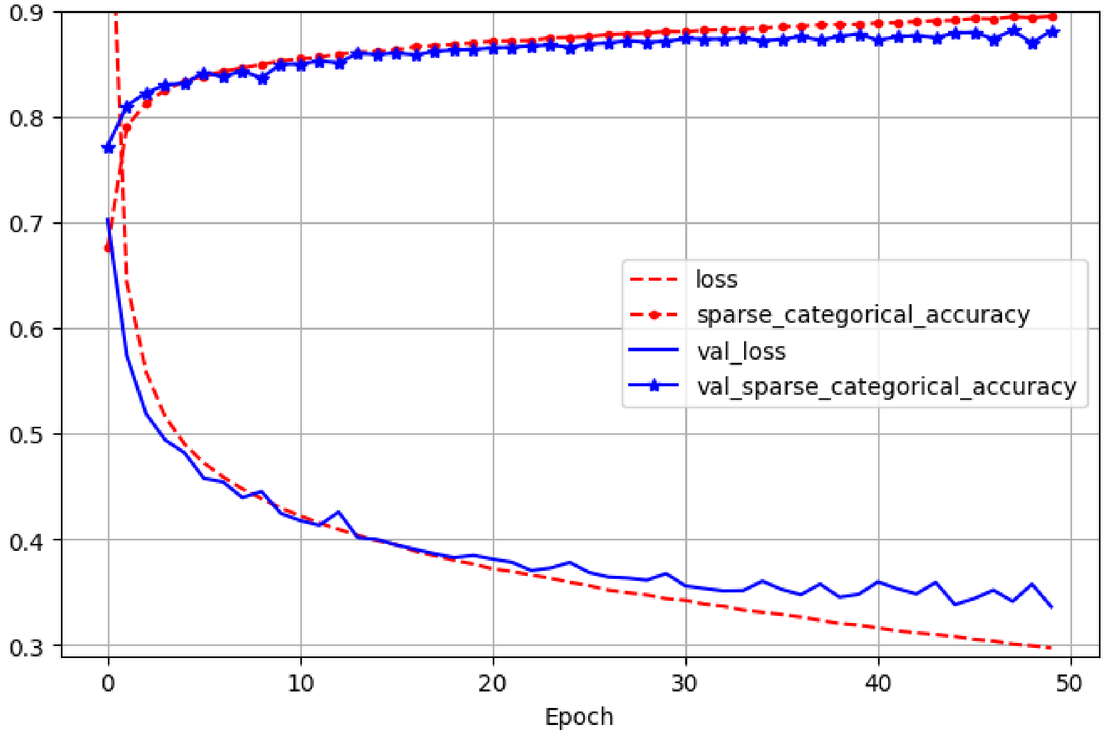
```
val_sparse_categorical_accuracy: 0.8654
Epoch 23/50
429/429 [==============================] - 7s 17ms/step - loss: 0.3663 -
sparse_categorical_accuracy: 0.8718 - val_loss: 0.3705 -
val_sparse_categorical_accuracy: 0.8668
Epoch 24/50
429/429 [==============================] - 8s 20ms/step - loss: 0.3633 -
sparse_categorical_accuracy: 0.8744 - val_loss: 0.3729 -
val_sparse_categorical_accuracy: 0.8674
Epoch 25/50
429/429 [==============================] - 7s 17ms/step - loss: 0.3595 -
sparse_categorical_accuracy: 0.8747 - val_loss: 0.3782 -
val_sparse_categorical_accuracy: 0.8654
Epoch 26/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3564 -
sparse_categorical_accuracy: 0.8755 - val_loss: 0.3687 -
val_sparse_categorical_accuracy: 0.8686
Epoch 27/50
429/429 [==============================] - 7s 17ms/step - loss: 0.3520 -
sparse_categorical_accuracy: 0.8775 - val_loss: 0.3644 -
val_sparse_categorical_accuracy: 0.8694
Epoch 28/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3497 -
sparse_categorical_accuracy: 0.8781 - val_loss: 0.3634 -
val_sparse_categorical_accuracy: 0.8716
Epoch 29/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3475 -
sparse_categorical_accuracy: 0.8787 - val_loss: 0.3615 -
val_sparse_categorical_accuracy: 0.8702
Epoch 30/50
429/429 [==============================] - 7s 17ms/step - loss: 0.3440 -
sparse_categorical_accuracy: 0.8803 - val_loss: 0.3676 -
val_sparse_categorical_accuracy: 0.8710
Epoch 31/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3424 -
sparse_categorical_accuracy: 0.8801 - val_loss: 0.3562 -
val_sparse_categorical_accuracy: 0.8736
Epoch 32/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3387 -
sparse_categorical_accuracy: 0.8816 - val_loss: 0.3536 -
val_sparse_categorical_accuracy: 0.8732
Epoch 33/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3368 -
sparse_categorical_accuracy: 0.8821 - val_loss: 0.3513 -
val_sparse_categorical_accuracy: 0.8734
Epoch 34/50
429/429 [==============================] - 7s 17ms/step - loss: 0.3331 -
sparse_categorical_accuracy: 0.8826 - val_loss: 0.3516 -
```

```
val_sparse_categorical_accuracy: 0.8740
Epoch 35/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3311 -
sparse_categorical_accuracy: 0.8838 - val_loss: 0.3606 -
val_sparse_categorical_accuracy: 0.8718
Epoch 36/50
429/429 [==============================] - 7s 17ms/step - loss: 0.3290 -
sparse_categorical_accuracy: 0.8846 - val_loss: 0.3527 -
val_sparse_categorical_accuracy: 0.8728
Epoch 37/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3266 -
sparse_categorical_accuracy: 0.8855 - val_loss: 0.3478 -
val_sparse_categorical_accuracy: 0.8756
Epoch 38/50
429/429 [==============================] - 7s 17ms/step - loss: 0.3235 -
sparse_categorical_accuracy: 0.8866 - val_loss: 0.3579 -
val_sparse_categorical_accuracy: 0.8720
Epoch 39/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3205 -
sparse_categorical_accuracy: 0.8871 - val_loss: 0.3454 -
val_sparse_categorical_accuracy: 0.8756
Epoch 40/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3191 -
sparse_categorical_accuracy: 0.8870 - val_loss: 0.3481 -
val_sparse_categorical_accuracy: 0.8780
Epoch 41/50
429/429 [==============================] - 7s 17ms/step - loss: 0.3164 -
sparse_categorical_accuracy: 0.8882 - val_loss: 0.3599 -
val_sparse_categorical_accuracy: 0.8720
Epoch 42/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3136 -
sparse_categorical_accuracy: 0.8885 - val_loss: 0.3534 -
val_sparse_categorical_accuracy: 0.8752
Epoch 43/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3116 -
sparse_categorical_accuracy: 0.8898 - val_loss: 0.3485 -
val_sparse_categorical_accuracy: 0.8764
Epoch 44/50
429/429 [==============================] - 8s 19ms/step - loss: 0.3100 -
sparse_categorical_accuracy: 0.8905 - val_loss: 0.3594 -
val_sparse_categorical_accuracy: 0.8746
Epoch 45/50
429/429 [==============================] - 7s 17ms/step - loss: 0.3081 -
sparse_categorical_accuracy: 0.8909 - val_loss: 0.3383 -
val_sparse_categorical_accuracy: 0.8788
Epoch 46/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3055 -
sparse_categorical_accuracy: 0.8925 - val_loss: 0.3441 -
```

```
val_sparse_categorical_accuracy: 0.8794
Epoch 47/50
429/429 [==============================] - 7s 17ms/step - loss: 0.3039 -
sparse_categorical_accuracy: 0.8922 - val_loss: 0.3520 -
val_sparse_categorical_accuracy: 0.8734
Epoch 48/50
429/429 [==============================] - 8s 18ms/step - loss: 0.3010 -
sparse_categorical_accuracy: 0.8942 - val_loss: 0.3413 -
val_sparse_categorical_accuracy: 0.8814
Epoch 49/50
429/429 [==============================] - 8s 19ms/step - loss: 0.2995 -
sparse_categorical_accuracy: 0.8934 - val_loss: 0.3577 -
val_sparse_categorical_accuracy: 0.8694
Epoch 50/50
429/429 [==============================] - 8s 18ms/step - loss: 0.2973 -
sparse_categorical_accuracy: 0.8947 - val_loss: 0.3363 -
val_sparse_categorical_accuracy: 0.8810
CPU times: user 3min 36s, sys: 31.4 s, total: 4min 7s
Wall time: 6min 30s
time: 6min 30s (started: 2023-04-03 06:07:54 +00:00)
```

6.1.4 Evaluate Model

[27]:
```
pd.DataFrame(train_history.history).plot(
    figsize=(8, 5), ylim=[0.29, 0.9],
    grid=True, xlabel="Epoch",
    style=["r--", "r--.", "b-", "b-*"],
)
plt.show()
```




```
time: 510 ms (started: 2023-04-03 06:16:21 +00:00)
```

[28]:
```
model.evaluate(test)
```

```
79/79 [==============================] - 3s 21ms/step - loss: 0.3582 -
sparse_categorical_accuracy: 0.8730
```

[28]:
```
[0.3581647276878357, 0.8730000257492065]
```

```
time: 3.54 s (started: 2023-04-03 06:16:31 +00:00)
```

[29]:
```
y_pred = model.predict(X_test)
y_pred = y_pred.argmax(axis=-1)

print(classification_report(y_test, y_pred, target_names=class_names))
```

```
313/313 [==============================] - 4s 9ms/step
              precision    recall  f1-score   support

 T-shirt/top       0.84      0.79      0.82      1000
     Trouser       0.98      0.97      0.98      1000
    Pullover       0.80      0.78      0.79      1000
       Dress       0.85      0.90      0.87      1000
        Coat       0.81      0.78      0.79      1000
```

```
      Sandal       0.96      0.95      0.96      1000
       Shirt       0.67      0.70      0.69      1000
     Sneaker       0.92      0.95      0.93      1000
         Bag       0.95      0.96      0.96      1000
  Ankle boot       0.96      0.95      0.95      1000

    accuracy                           0.87     10000
   macro avg       0.87      0.87      0.87     10000
weighted avg       0.87      0.87      0.87     10000

time: 4.59 s (started: 2023-04-03 06:16:41 +00:00)
```

6.2 Part 2

6.2.1 Fashion MNIST Convolutional Neural Network

Repeat part problem 2, but this time create a convolution neural network using the Fashion MNIST network in Chapter 14 of Aurelien.

6.2.2 Build Model

[30]:
```
from functools import partial

def build_cnn_model():
    DefaultConv2D = partial(
        tf.keras.layers.Conv2D,
        kernel_size=3,
        padding="same",
        activation="relu",
        kernel_initializer="he_normal"
    )

    model = tf.keras.Sequential([
        DefaultConv2D(filters=64, kernel_size=7, input_shape=[28, 28, 1]),
        tf.keras.layers.MaxPool2D(),
        DefaultConv2D(filters=128),
        DefaultConv2D(filters=128),
        tf.keras.layers.MaxPool2D(),
        DefaultConv2D(filters=256),
        DefaultConv2D(filters=256),
        tf.keras.layers.MaxPool2D(),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(
            units=128, activation="relu", kernel_initializer="he_normal"
        ),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(
            units=64, activation="relu", kernel_initializer="he_normal"
```

```
        ),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(units=10, activation="softmax")
    ])

    model.compile(
        loss="sparse_categorical_crossentropy",
        optimizer="adam",
        metrics=[tf.keras.metrics.sparse_categorical_accuracy]
    )
    return model
```

time: 1.33 ms (started: 2023-04-03 06:16:53 +00:00)

[36]:
```
# Clear backend sessions to reset name counters
tf.keras.backend.clear_session()

with strategy.scope():
    model_cnn = build_cnn_model()

model_cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 64)        3200

 max_pooling2d (MaxPooling2D  (None, 14, 14, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 128)       73856

 conv2d_2 (Conv2D)           (None, 14, 14, 128)       147584

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 128)        0
 2D)

 conv2d_3 (Conv2D)           (None, 7, 7, 256)         295168

 conv2d_4 (Conv2D)           (None, 7, 7, 256)         590080

 max_pooling2d_2 (MaxPooling  (None, 3, 3, 256)        0
 2D)

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 128)               295040
```

```
 dropout (Dropout)           (None, 128)              0

 dense_1 (Dense)             (None, 64)               8256

 dropout_1 (Dropout)         (None, 64)               0

 dense_2 (Dense)             (None, 10)               650

=================================================================
Total params: 1,413,834
Trainable params: 1,413,834
Non-trainable params: 0
_________________________________________________________________
time: 1.06 s (started: 2023-04-03 06:20:16 +00:00)
```

6.2.3 Train Model

[37]:
```
EPOCHS = 9

n_steps = X_train.shape[0] // BATCH_SIZE
```

```
time: 667 µs (started: 2023-04-03 06:20:23 +00:00)
```

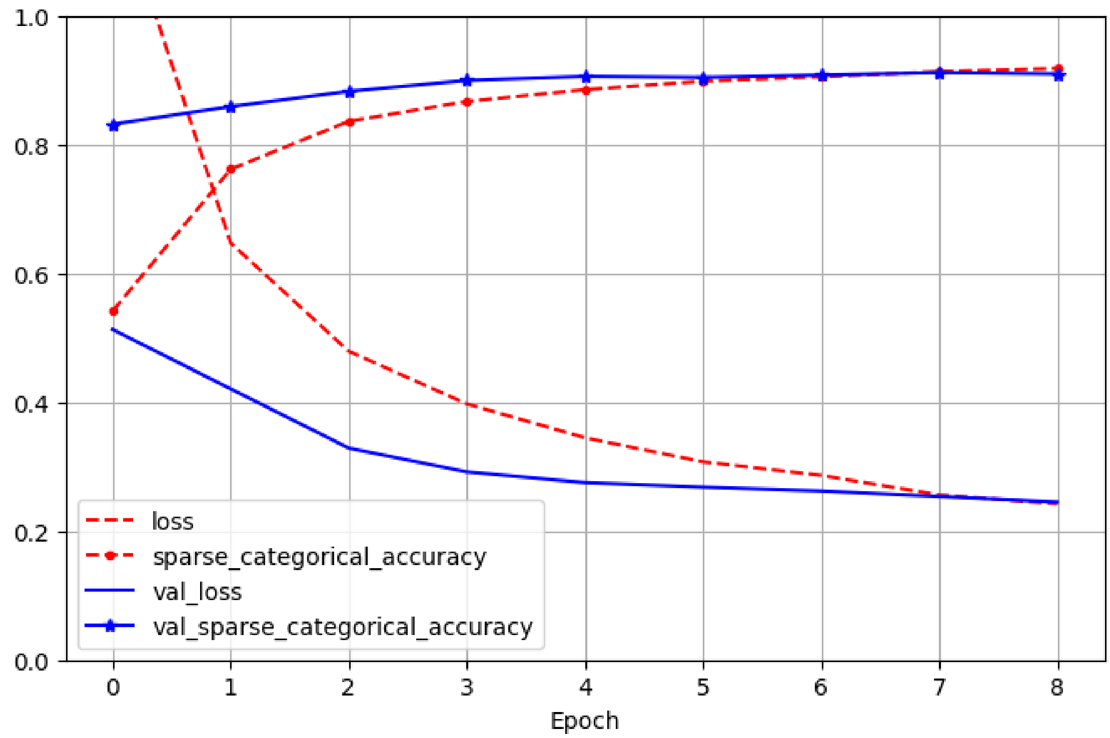
[38]:
```
%%time
train_history_cnn = model_cnn.fit(
    train,
    steps_per_epoch=n_steps,
    validation_data=valid,
    epochs=EPOCHS
)
```

```
Epoch 1/9
429/429 [==============================] - 16s 23ms/step - loss: 1.2032 -
sparse_categorical_accuracy: 0.5415 - val_loss: 0.5128 -
val_sparse_categorical_accuracy: 0.8316
Epoch 2/9
429/429 [==============================] - 8s 19ms/step - loss: 0.6471 -
sparse_categorical_accuracy: 0.7616 - val_loss: 0.4206 -
val_sparse_categorical_accuracy: 0.8588
Epoch 3/9
429/429 [==============================] - 9s 20ms/step - loss: 0.4790 -
sparse_categorical_accuracy: 0.8361 - val_loss: 0.3287 -
val_sparse_categorical_accuracy: 0.8826
Epoch 4/9
429/429 [==============================] - 9s 20ms/step - loss: 0.3973 -
sparse_categorical_accuracy: 0.8668 - val_loss: 0.2917 -
val_sparse_categorical_accuracy: 0.8994
```

```
Epoch 5/9
429/429 [==============================] - 8s 20ms/step - loss: 0.3448 -
sparse_categorical_accuracy: 0.8852 - val_loss: 0.2751 -
val_sparse_categorical_accuracy: 0.9056
Epoch 6/9
429/429 [==============================] - 9s 21ms/step - loss: 0.3075 -
sparse_categorical_accuracy: 0.8983 - val_loss: 0.2681 -
val_sparse_categorical_accuracy: 0.9040
Epoch 7/9
429/429 [==============================] - 9s 20ms/step - loss: 0.2863 -
sparse_categorical_accuracy: 0.9050 - val_loss: 0.2621 -
val_sparse_categorical_accuracy: 0.9078
Epoch 8/9
429/429 [==============================] - 9s 20ms/step - loss: 0.2561 -
sparse_categorical_accuracy: 0.9130 - val_loss: 0.2535 -
val_sparse_categorical_accuracy: 0.9116
Epoch 9/9
429/429 [==============================] - 9s 21ms/step - loss: 0.2422 -
sparse_categorical_accuracy: 0.9181 - val_loss: 0.2450 -
val_sparse_categorical_accuracy: 0.9096
CPU times: user 47.4 s, sys: 6.02 s, total: 53.4 s
Wall time: 1min 28s
time: 1min 28s (started: 2023-04-03 06:20:25 +00:00)
```

6.2.4 Evaluate Model

[39]:
```
pd.DataFrame(train_history_cnn.history).plot(
    figsize=(8, 5), ylim=[0, 1],
    grid=True, xlabel="Epoch",
    style=["r--", "r--.", "b-", "b-*"],
)
plt.show()
```




```
time: 282 ms (started: 2023-04-03 06:21:57 +00:00)
```

[40]:
```
model_cnn.evaluate(test)
```

```
79/79 [==============================] - 2s 16ms/step - loss: 0.2616 -
sparse_categorical_accuracy: 0.9157
```

[40]:
```
[0.26163095235824585, 0.9157000184059143]
```

```
time: 2.48 s (started: 2023-04-03 06:22:08 +00:00)
```

[41]:
```
y_pred = model_cnn.predict(X_test)
y_pred = y_pred.argmax(axis=-1)

print(classification_report(y_test, y_pred, target_names=class_names))
```

```
313/313 [==============================] - 6s 15ms/step
              precision    recall  f1-score   support

 T-shirt/top       0.89      0.83      0.86      1000
     Trouser       0.99      0.98      0.99      1000
    Pullover       0.84      0.89      0.86      1000
       Dress       0.94      0.90      0.92      1000
        Coat       0.84      0.89      0.86      1000
```

```
      Sandal       0.98      0.99      0.99      1000
       Shirt       0.75      0.75      0.75      1000
     Sneaker       0.97      0.97      0.97      1000
         Bag       0.99      0.99      0.99      1000
  Ankle boot       0.98      0.97      0.97      1000

    accuracy                           0.92     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.92      0.92      0.92     10000

time: 6.92 s (started: 2023-04-03 06:22:13 +00:00)
```

7 References

[1] Jigsaw Multilingual Toxic Comment Classification - Roberta-large-2