### Hyperparameter tuning

To perform hyperparameter tuning, we need to define the search space, that is to say which hyperparameters need to be optimized and in what range. Here, for this relatively small model, there are already hyperparameters that can be tuned:

1. Dropout rate
2. Number of filters for the convolutional layers
3. Number of units for the dense layer
4. Corresponding activation function

**Install Keras Tuner**

pip install keras-tuner

**Import required Libraries**

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.datasets import fashion\_mnist

from tensorflow.keras.layers import Conv2D, Dense, Dropout, Flatten, MaxPooling2D

print(tf.\_\_version\_\_)

### Load Data

(train\_images, train\_labels),(test\_images, test\_labels)= fashion\_mnist.load\_data()

train\_images= train\_images/255.0

test\_images= test\_images/255.0

train\_images= train\_images.reshape(len(train\_images), 28, 28,1)

test\_images= test\_images.reshape(len(test\_images), 28, 28,1)

train\_images.shape

### Hyperparameter tuning

def build\_model(hp):

    model= keras.Sequential()

    model.add(

        Conv2D(

            filters=16,

            kernel\_size=3,

            activation="relu",

            input\_shape=(28,28,1),

        )

    )

    model.add(Conv2D(filters=16, activation="relu", kernel\_size=3))

    model.add(MaxPooling2D(pool\_size=2))

    model.add(

        Dropout(

            rate=hp.Float(

                "dropout\_1", min\_value=0.0, max\_value=0.5, default=0.25, step=0.05,

            )

        )

    )

    model.add(Conv2D(filters=32, kernel\_size=3, activation="relu"))

    model.add(

        Conv2D(

            filters=hp.Choice("num\_filters", values=[32, 64], default=64,),

            activation="relu",

            kernel\_size=3,

        )

    )

    model.add(MaxPooling2D(pool\_size=2))

    model.add(

        Dropout(

            rate=hp.Float(

                "dropout\_2", min\_value=0.0, max\_value=0.5, default=0.25, step=0.05,

            )

        )

    )

    model.add(Flatten())

    model.add(

        Dense(

            units=hp.Int(

                "units", min\_value=32, max\_value=512, step=32, default=128

            ),

            activation=hp.Choice(

                "dense\_activation",

                values=["relu", "tanh", "sigmoid"],

                default="relu",

            ),

        )

    )

    model.add(

        Dropout(

            rate=hp.Float(

                "dropout\_3", min\_value=0.0, max\_value=0.5, default=0.25, step=0.05

            )

        )

    )

    model.add(Dense(10, activation="softmax"))

    model.compile(optimizer=keras.optimizers.Adam(

        hp.Choice('learning\_rate', [1e-2, 1e-3, 1e-4])),

                  loss='sparse\_categorical\_crossentropy',

                  metrics=['accuracy'])

return model

### Random Search to find best hyperparameters

from kerastuner import RandomSearch

from kerastuner.engine.hyperparameters import HyperParameters

tuner\_search= RandomSearch(build\_model,

                           objective='val\_accuracy',

                           max\_trials=6,

                           directory='output',

                           project\_name='mnist\_fashion'

                          )

tuner\_search.search(train\_images, train\_labels, epochs=10, validation\_split=0.1)

### Best Hyperparameters Determined

tuner\_search.results\_summary()

model= tuner\_search.get\_best\_models(num\_models=1)[0]

model.summary()

history= model.fit(train\_images, train\_labels, epochs=10, validation\_split=0.1, initial\_epoch=3)

### Make Predictions

predictions= model.predict(test\_images)

### Visualize the prediction

import matplotlib.pyplot as plt

import numpy as np

plt.imshow(test\_images[0].reshape(28,28))

print('Actual\_label :', test\_labels[0])

print('Predicted\_label :', np.argmax(predictions[0]))

plt.imshow(test\_images[0].reshape(28,28))

print('Actual\_label :', test\_labels[10])

print('Predicted\_label :', np.argmax(predictions[10]))

**list all data in history**

print(history.history.keys())

**summarize history for accuracy**

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

**summarize history for loss**

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()