Návod použití

Google Colab

Pokud se nacházíte v rozhraní Google Colab, složka "DMP_Neuronove_site", ve které se nachází tento notebook a program "custom_ann.py", by se měla na Vašem Google Disku nacházet ve výchozí složce "Můj Disk".

všechny buňky lze naráz spustit pomocí "Běh -> Spustit vše" v horní liště. Samostatné buňky lze spustit pomocí kliknutím na danou buňku a klávesami "Ctrl+Enter".

Chvilku možná bude trvat, než se notebook připojí k běhovému prostředí, které doporučuji nastavit na vzdálenou GPU pomocí "Běh -> Zvolit běhové prostředí -> GPU".

Program v určité části požádá o přístup k Vašemu Google Disku, aby mohl naimportovat neuronovou síť ze souboru "custom_cnn.py". Pokud povolíte, zbytek programu by měl proběhnout bez problému. Pokud se po udělení přístupu notebook zasekne, odpojte a smažte běh pomocí "Běh -> Odpojit a smazat běh" a spusťte manuálně buňku po buňce.

Jiné rozhraní

Pokud se nacházíte v jiném rozhraní, pak zakomentujte nebo smažte všechny řádky kódu okomentované "# Pro Colab" a ujistěte se, že všechny použité knihovny máte nainstalované na počítači nebo ve Vašem virtuálním prostředí.

Pro správnou funkci musíte být připojeni k internetu, protože datový soubor Fashion MNIST je nahráván z cloudové databáze knihovny Deeplake.

Importování knihoven

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

Importování mé vlastní neuronové sítě z google drive do google colab

```
from google.colab import drive # Pro Colab
drive.mount('/content/gdrive') # Pro Colab
```

```
Mounted at /content/gdrive
```

```
import sys # Pro Colab
sys.path.append('/content/gdrive/My Drive/DMP_Neuronove_site') # Pro Colab
```

```
from custom_ann import Artificial_Neural_Network
```

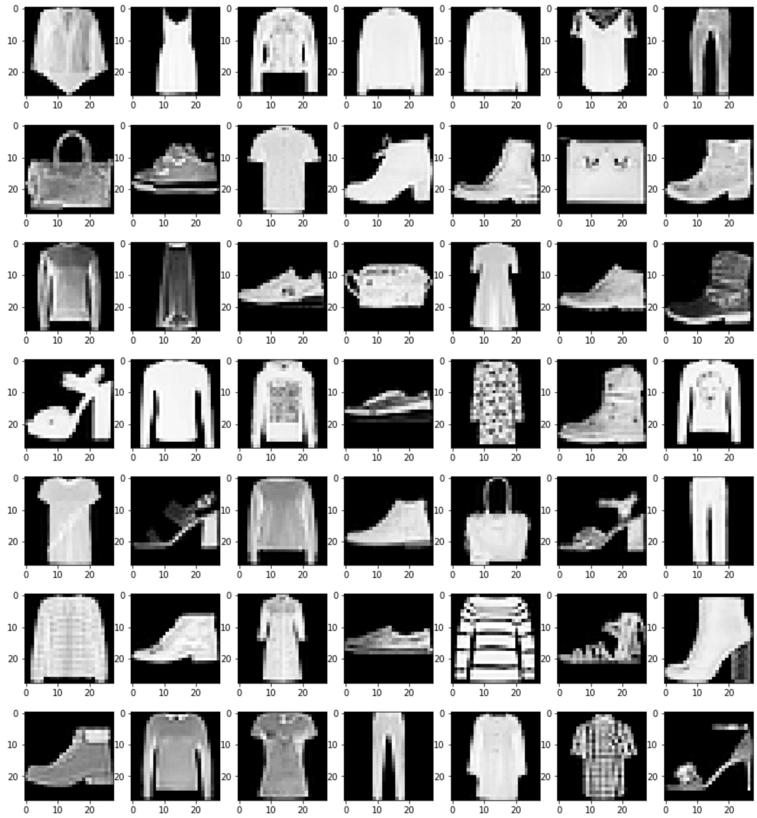
Nahrání a zpracování dat

```
# Nahrání a rozdělení dat
fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
X_valid, X_train = X_train_full[:5000], X_train_full[5000:]
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]

# Normalizace vstupních dat do hodnot v rozmezí 0-1
X_train_norm, X_test_norm, X_valid_norm = X_train / 255, X_test / 255, X_valid / 255

# Manuální zploštění snímků pro mou vlastní neuronovou síť
X_train_flattened = X_train_norm.reshape(X_train_norm.shape[0], 784, 1)
X_test_flattened = X_test_norm.reshape(X_test_norm.shape[0], 784, 1)
X_valid_flattened = X_valid_norm.reshape(X_valid_norm.shape[0], 784, 1)

# Úprava vstupních dat pro vstup neuronové sítě vytvořené pomocí TensorFlow Keras
X_train_norm = X_train_norm.reshape(X_train_norm.shape[0], 28, 28, 1)
X_test_norm = X_test_norm.reshape(X_test_norm.shape[0], 28, 28, 1)
X_valid_norm = X_valid_norm.reshape(X_valid_norm.shape[0], 28, 28, 1)

# K=ódování výstupních dat do cílových vektorů
y_train_encoded = tf.keras.utils.to_categorical(y_train)
y_test_encoded = tf.keras.utils.to_categorical(y_test)
```

```
y_test_encoded = tf.keras.utils.to_categorical(y_test)
y_valid_encoded = tf.keras.utils.to_categorical(y_valid)
```

Ilustrační snímky z datového souboru Fashion MNIST

```
class_names = ['T_shirt/top', 'Trousers', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

fig, axes = plt.subplots(7, 7, figsize = (15, 17))
for row in axes:
  for axe in row:
    i = np.random.randint(len(X_train))
    axe.imshow(X_train[i], cmap="Greys_r")
```



Učení mé vlastní neuronové sítě

```
custom_model = Artificial_Neural_Network((784, 32, 16, 10))
custom_model.stochastic_gradient_descent(X_train_flattened, y_train_encoded,
                                         epochs=30, mini_batch_size=10, learning_rate=0.1,
                                         validation_data=(X_valid_flattened, y_valid_encoded))
```

```
Epoch 1 -> Training loss: 0.45438462246334277, Training accuracy: 0.6829090909090909, Validation loss: 0.44330408046591413, Validat
Epoch 2 -> Training loss: 0.3601396237071496, Training accuracy: 0.748890909090909, Validation loss: 0.3513637453419897, Validation
Epoch 3 -> Training loss: 0.3206062284852381, Training accuracy: 0.7798181818181819, Validation loss: 0.31237417425083847, Validati
Epoch 4 -> Training loss: 0.2939074572556077, Training accuracy: 0.7988363636363637, Validation loss: 0.2858914558880485, Validatio
Epoch 5 -> Training loss: 0.2794990468681299, Training accuracy: 0.8078909090909091, Validation loss: 0.2728838300979265, Validatio
```

```
Epoch 6 -> Training loss: 0.2646166109492382, Training accuracy: 0.8184181818181818, Validation loss: 0.2599742125665879, Validatio
Epoch 7 -> Training loss: 0.25679309341869694, Training accuracy: 0.8248363636363636, Validation loss: 0.2527087257753105, Validati
Epoch 8 -> Training loss: 0.25014804626850384, Training accuracy: 0.8293090909090909, Validation loss: 0.248052158773793, Validatio
Epoch 9 -> Training loss: 0.24161108825971309, Training accuracy: 0.8361454545454545, Validation loss: 0.24094297218574875, Validat
Epoch 10 -> Training loss: 0.23716334497972485, Training accuracy: 0.8398909090909091, Validation loss: 0.23889155833525905, Valida
Epoch 11 -> Training loss: 0.23321028489107085, Training accuracy: 0.8413090909090909, Validation loss: 0.23626373366747716, Valida
Epoch 12 -> Training loss: 0.22690634195230427, Training accuracy: 0.8451636363636363, Validation loss: 0.2299715729415016, Validat
Epoch 13 -> Training loss: 0.22394141958107652, Training accuracy: 0.8492181818181819, Validation loss: 0.23047749170432652, Valida
Epoch 14 -> Training loss: 0.21942481369676536, Training accuracy: 0.8528363636363636, Validation loss: 0.22633732143218985, Valida
Epoch 15 -> Training loss: 0.21833152832378264, Training accuracy: 0.8522, Validation loss: 0.2248630903759242, Validation accuracy
Epoch 16 -> Training loss: 0.21548711678369586, Training accuracy: 0.8540545454545454, Validation loss: 0.22317459889528055, Valida
Epoch 17 -> Training loss: 0.21159615578040897, Training accuracy: 0.8570363636363636, Validation loss: 0.21999428609302749, Valida
Epoch 18 -> Training loss: 0.20813011094852468, Training accuracy: 0.8603454545454545, Validation loss: 0.21886483840002635, Valida
Epoch 19 -> Training loss: 0.20805597938672538, Training accuracy: 0.8601818181818182, Validation loss: 0.21757610902594912, Valida
Epoch 20 -> Training loss: 0.20476115262106856, Training accuracy: 0.8635636363636363, Validation loss: 0.21720965355638044, Valida
Epoch 21 -> Training loss: 0.20205175889658583, Training accuracy: 0.8643090909090909, Validation loss: 0.21304969480750824, Valida
Epoch 22 -> Training loss: 0.20170074279716757, Training accuracy: 0.8646181818181818, Validation loss: 0.2142686967727012, Validat
Epoch 23 -> Training loss: 0.20177244335666777, Training accuracy: 0.863909090909091, Validation loss: 0.21451401829592986, Validat
Epoch 24 -> Training loss: 0.19781680928776763, Training accuracy: 0.8679272727272728, Validation loss: 0.21178312635583857, Valida
Epoch 25 -> Training loss: 0.19418735171582144, Training accuracy: 0.8700363636363636, Validation loss: 0.20946216860864728, Valida
Epoch 26 -> Training loss: 0.1953100201300055, Training accuracy: 0.8694727272727273, Validation loss: 0.21366646812994755, Validat
Epoch 27 -> Training loss: 0.19214366580835585, Training accuracy: 0.8714363636363637, Validation loss: 0.20853602317188574, Valida
Epoch 28 -> Training loss: 0.19063069833277657, Training accuracy: 0.8730181818181818, Validation loss: 0.20794548789737705, Valida
Epoch 29 -> Training loss: 0.1886395249383492, Training accuracy: 0.8746727272727273, Validation loss: 0.20656865031982152, Validat
Epoch 30 -> Training loss: 0.1876396443307317, Training accuracy: 0.8750545454545454, Validation loss: 0.2067915994989725, Validati
```

Vytvoření a učení neuronové sítě pomocí TensorFlow Keras

```
keras_model = tf.keras.models.Sequential([
      tf.keras.layers.Conv2D(32, (3, 3), activation="relu",
                       input_shape=(28, 28, 1)),
      tf.keras.layers.Conv2D(32, (3, 3), activation="relu"),
      tf.keras.layers.MaxPooling2D((2, 2)),
      tf.keras.layers.Dropout(0.25),
      tf.keras.layers.Conv2D(64, (3, 3), activation="relu"),
      tf.keras.layers.MaxPooling2D((2, 2)),
      tf.keras.layers.Dropout(0.25),
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(128, activation="relu"),
      tf.keras.layers.Dropout(0.5),
      tf.keras.layers.Dense(64, activation="relu"),
      tf.keras.layers.Dropout(0.5),
      tf.keras.layers.Dense(10, activation="softmax")
])
```

```
keras_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 26, 26, 32)        320

 conv2d_7 (Conv2D)           (None, 24, 24, 32)        9248

 max_pooling2d_4 (MaxPooling  (None, 12, 12, 32)       0
 2D)

 dropout_8 (Dropout)         (None, 12, 12, 32)        0

 conv2d_8 (Conv2D)           (None, 10, 10, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 dropout_9 (Dropout)         (None, 5, 5, 64)          0

 flatten_2 (Flatten)         (None, 1600)              0

 dense_6 (Dense)             (None, 128)               204928

 dropout_10 (Dropout)        (None, 128)               0

 dense_7 (Dense)             (None, 64)                8256

 dropout_11 (Dropout)        (None, 64)                0

 dense_8 (Dense)             (None, 10)                650

=================================================================
Total params: 241,898
Trainable params: 241,898
```

```
Non-trainable params: 0
_________________________________________________________________
```

```
keras_model.compile(optimizer = "nadam",
                    loss = "categorical_crossentropy",
                    metrics=["accuracy"])
```

```
keras_model_history = keras_model.fit(X_train_norm, y_train_encoded, epochs=30, validation_data = (X_valid_norm, y_valid_encoded))
```

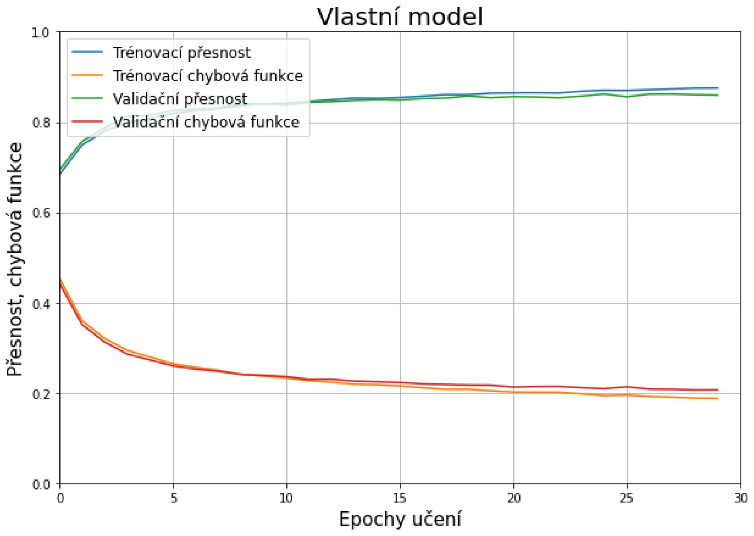
```
Epoch 1/30
1719/1719 [==============================] - 112s 64ms/step - loss: 0.7568 - accuracy: 0.7194 - val_loss: 0.3976 - val_accuracy:
Epoch 2/30
1719/1719 [==============================] - 104s 60ms/step - loss: 0.4808 - accuracy: 0.8293 - val_loss: 0.3114 - val_accuracy:
Epoch 3/30
1719/1719 [==============================] - 106s 62ms/step - loss: 0.4076 - accuracy: 0.8584 - val_loss: 0.2778 - val_accuracy:
Epoch 4/30
1719/1719 [==============================] - 104s 60ms/step - loss: 0.3725 - accuracy: 0.8727 - val_loss: 0.2681 - val_accuracy:
Epoch 5/30
1719/1719 [==============================] - 111s 64ms/step - loss: 0.3446 - accuracy: 0.8796 - val_loss: 0.2705 - val_accuracy:
Epoch 6/30
1719/1719 [==============================] - 103s 60ms/step - loss: 0.3293 - accuracy: 0.8853 - val_loss: 0.2515 - val_accuracy:
Epoch 7/30
1719/1719 [==============================] - 103s 60ms/step - loss: 0.3194 - accuracy: 0.8894 - val_loss: 0.2449 - val_accuracy:
Epoch 8/30
1719/1719 [==============================] - 102s 59ms/step - loss: 0.3071 - accuracy: 0.8931 - val_loss: 0.2374 - val_accuracy:
Epoch 9/30
1719/1719 [==============================] - 103s 60ms/step - loss: 0.3007 - accuracy: 0.8962 - val_loss: 0.2290 - val_accuracy:
Epoch 10/30
1719/1719 [==============================] - 103s 60ms/step - loss: 0.2924 - accuracy: 0.8994 - val_loss: 0.2261 - val_accuracy:
Epoch 11/30
1719/1719 [==============================] - 103s 60ms/step - loss: 0.2847 - accuracy: 0.9018 - val_loss: 0.2264 - val_accuracy:
Epoch 12/30
1719/1719 [==============================] - 100s 58ms/step - loss: 0.2807 - accuracy: 0.9026 - val_loss: 0.2438 - val_accuracy:
Epoch 13/30
1719/1719 [==============================] - 98s 57ms/step - loss: 0.2753 - accuracy: 0.9041 - val_loss: 0.2287 - val_accuracy: 0
Epoch 14/30
1719/1719 [==============================] - 99s 58ms/step - loss: 0.2721 - accuracy: 0.9049 - val_loss: 0.2166 - val_accuracy: 0
Epoch 15/30
1719/1719 [==============================] - 99s 58ms/step - loss: 0.2653 - accuracy: 0.9075 - val_loss: 0.2223 - val_accuracy: 0
Epoch 16/30
1719/1719 [==============================] - 97s 56ms/step - loss: 0.2611 - accuracy: 0.9089 - val_loss: 0.2214 - val_accuracy: 0
Epoch 17/30
1719/1719 [==============================] - 94s 55ms/step - loss: 0.2589 - accuracy: 0.9095 - val_loss: 0.2153 - val_accuracy: 0
Epoch 18/30
1719/1719 [==============================] - 95s 55ms/step - loss: 0.2591 - accuracy: 0.9082 - val_loss: 0.2227 - val_accuracy: 0
Epoch 19/30
1719/1719 [==============================] - 100s 58ms/step - loss: 0.2595 - accuracy: 0.9104 - val_loss: 0.2157 - val_accuracy:
Epoch 20/30
1719/1719 [==============================] - 98s 57ms/step - loss: 0.2545 - accuracy: 0.9118 - val_loss: 0.2296 - val_accuracy: 0
Epoch 21/30
1719/1719 [==============================] - 98s 57ms/step - loss: 0.2535 - accuracy: 0.9114 - val_loss: 0.2105 - val_accuracy: 0
Epoch 22/30
1719/1719 [==============================] - 98s 57ms/step - loss: 0.2505 - accuracy: 0.9125 - val_loss: 0.2125 - val_accuracy: 0
Epoch 23/30
1719/1719 [==============================] - 102s 59ms/step - loss: 0.2500 - accuracy: 0.9129 - val_loss: 0.2174 - val_accuracy:
Epoch 24/30
1719/1719 [==============================] - 98s 57ms/step - loss: 0.2432 - accuracy: 0.9147 - val_loss: 0.2046 - val_accuracy: 0
Epoch 25/30
1719/1719 [==============================] - 100s 58ms/step - loss: 0.2403 - accuracy: 0.9160 - val_loss: 0.2087 - val_accuracy:
Epoch 26/30
1719/1719 [==============================] - 99s 57ms/step - loss: 0.2434 - accuracy: 0.9173 - val_loss: 0.2147 - val_accuracy: 0
Epoch 27/30
1719/1719 [==============================] - 100s 58ms/step - loss: 0.2390 - accuracy: 0.9170 - val_loss: 0.2121 - val_accuracy:
Epoch 28/30
1719/1719 [==============================] - 99s 57ms/step - loss: 0.2381 - accuracy: 0.9181 - val_loss: 0.2175 - val_accuracy: 0
Epoch 29/30
```

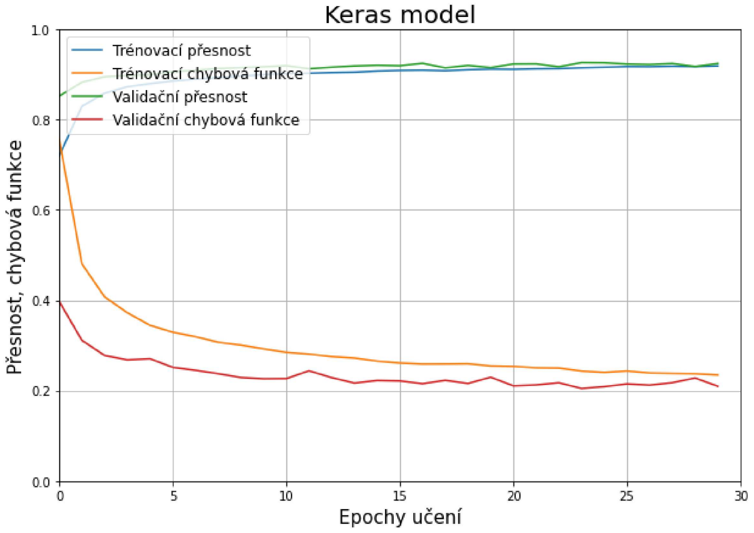
Testování a porovnání výsledných modelů

Porovnání průběhu učení neuronových sítí

```
histories = {"Vlastní model": custom_model.history, "Keras model": keras_model_history.history}
fig, axes = plt.subplots(len(histories), 1, figsize = (10, 15))
for index, (title, history) in enumerate(histories.items()):
  axes[index].set_title(title, fontsize=20)
  axes[index].set_xlabel('Epochy učení', fontsize=15)
  axes[index].set_ylabel('Přesnost, chybová funkce', fontsize=15)
  axes[index].plot(history["accuracy"])
  axes[index].plot(history["loss"])
  axes[index].plot(history["val_accuracy"])
```

```
    axes[index].plot(history["val_loss"])
    axes[index].grid(True)
    axes[index].set_xlim(0, len(history["accuracy"]))
    axes[index].set_ylim(0, 1)
    axes[index].legend(["Trénovací přesnost", "Trénovací chybová funkce", "Validační přesnost",  "Validační chybová funkce"], loc="upper le
```





Testování modelů

```
custom_model.evaluate(X_test_flattened, y_test_encoded)
```

```
Test loss: 0.22970746994031216, Test accuracy: 0.8438
```

```
keras_model.evaluate(X_test_norm, y_test_encoded)
```

```
313/313 [==============================] - 5s 17ms/step - loss: 0.2367 - accuracy: 0.9175
[0.2367277592420578, 0.9175000190734863]
```

Ilustrační predikce

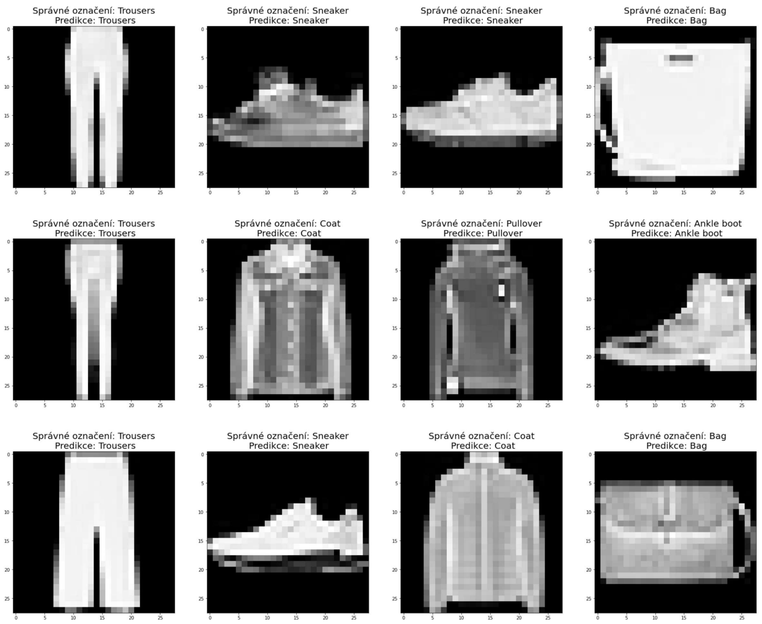
Ilustrační predikce mé vlastní neuronové sítě

```
predictions = custom_model.predict(X_test_flattened).argmax(axis=1)
predictions
```

```
array([9, 2, 1, ..., 8, 1, 5])
```

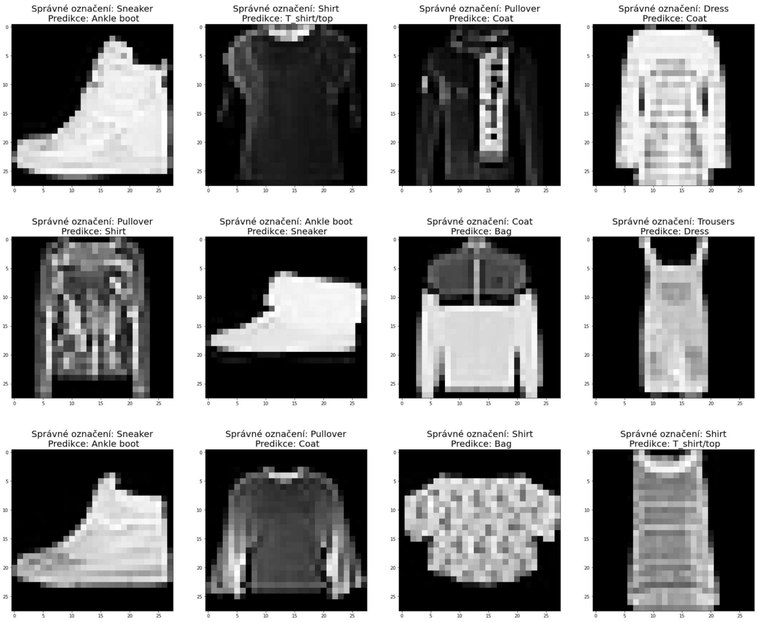
Správné predikce

```
fig, axes = plt.subplots(3, 4, figsize = (, 25))
for row in axes:
  for axe in row:
    i = np.random.randint(len(predictions))
    while y_test[i] != predictions[i]:
      i = np.random.randint(len(predictions))
    axe.imshow(X_test[i], cmap="Greys_r")
    axe.set_title(f"Správné označení: {class_names[y_test[i]]}\n Predikce: {class_names[predictions[i]]}"
```



Špatné predikce

```
fig, axes = plt.subplots(3, 4, figsize = (30, 25))
for row in axes:
  for axe in row:
    i = np.random.randint(len(predictions))
    while y_test[i] == predictions[i]:
      i = np.random.randint(len(predictions))
    axe.imshow(X_test[i], cmap="Greys_r")
    axe.set_title(f"Správné označení: {class_names[y_test[i]]}\n Predikce: {class_names[predictions[i]]}", fontsize=20)
```



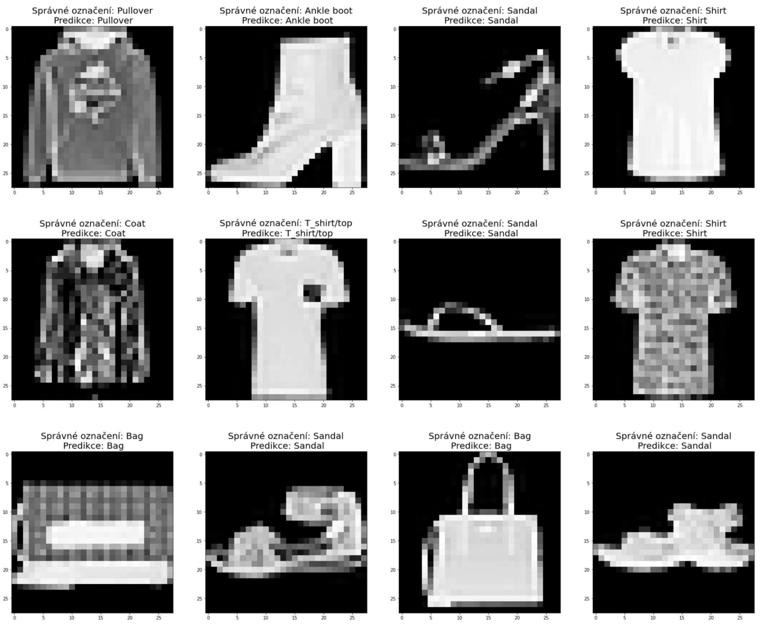
Ilustrační predikce neuronové sítě vytvořené pomocí TensorFlow Keras

```
predictions = keras_model.predict(X_test).argmax(axis=1)
predictions
```

```
313/313 [==============================] - 4s 13ms/step
array([5, 2, 1, ..., 6, 1, 5])
```

Správné predikce

```
fig, axes = plt.subplots(3, 4, figsize = (30, 25))
for row in axes:
  for axe in row:
    i = np.random.randint(len(predictions))
    while y_test[i] != predictions[i]:
      i = np.random.randint(len(predictions))
    axe.imshow(X_test[i], cmap="Greys_r")
    axe.set_title(f"Správné označení: {class_names[y_test[i]]}\n Predikce: {class_names[predictions[i]]}", fontsize=20)
```



Špatné predikce

```
fig, axes = plt.subplots(3, 4, figsize = (30, 25))
for row in axes:
  for axe in row:
    i = np.random.randint(len(predictions))
    while y_test[i] == predictions[i]:
      i = np.random.randint(len(predictions))
    axe.imshow(X_test[i], cmap="Greys_r")
    axe.set_title(f"Správné označení: {class_names[y_test[i]]}\n Predikce: {class_names[predictions[i]]}", fontsize=20)
```


Správné označení: T_shirt/top
Predikce: Coat
Správné označení: Pullover
Predikce: Coat
Správné označení: Ankle boot
Predikce: Sandal
Správné označení: T_shirt/top
Predikce: Shirt
Správné označení: Dress
Predikce: Coat
Správné označení: Pullover
Predikce: Shirt
Správné označení: Ankle boot
Predikce: Shirt
Správné označení: Sneaker
Predikce: Sandal
Správné označení: Coat
Predikce: Shirt
Správné označení: Sneaker
Predikce: Sandal
Správné označení: Dress
Predikce: Shirt
Správné označení: Sneaker
Predikce: T_shirt/top