
VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFOMATIKOS FAKULTETAS

DUOMENŲ MOKSLO BAKALAURAS

# Vaizdų klasifikavimas

Ataskaita

Atliko: Simona Gelžinytė 3 k. 2 gr.

Vadovas: dr. Viktor Medvedev

Vilnius, 2022

# Turinys

# ĮVADAS

## Tyrimo tikslas

Apmokyti konvoliucinį neuroninį tinklą vaizdams klasifikuoti.

## Tyrimo uždaviniai

- Paruošti duomenis
- Sukurti programą, kurioje įgyvendinti konvoliuciniai neuroniniai vaizzdams klasifikuoti.

## Duomenys

Šiame tyrime buvo naudoti duomenys iš https://www.cs.toronto.edu/~kriz/cifar.html. Duomenų rinkinį sudaro 60000 nuotraukų, kurių dydis yra 32 x 32 pikselių Kiekvienas paveiksliukas patenka į tik vieną iš 10 klasių

Klasės:

• Lėktuvas

• Automobilis • Paukštis

• Katė

• Elnias

• Šuo

• Varlė

• Arklys

• Laivas

• Sunkvežimis

# PAGRINDINĖ DALIS

## Duomenų paruošimas

```
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tqdm import tqdm
TEST_DATASET_SIZE = 10000
```

```
(train_images, train_labels), (test_images, test_labels) =
datasets.cifar10.load_data()
X = np.concatenate((train_images,test_images))
Y = np.concatenate((train_labels,test_labels))
train_images, test_images, train_labels, test_labels =
train_test_split(X, Y, test_size=TEST_DATASET_SIZE, random_state=4)
train_images, test_images = train_images / 255.0, test_images / 255.0
```

Kadangi duomenys yra RGB reikšmės, jos svyruoja nuo 0 iki 255, norint kad modelis geriau mokytųsi, reikia normalizuoti duomenis, tad padalinu reikšmes iš 255, kad gautųsi skalė 0 - 1

Tensorflow automatiškai parenka kad testavimo aibės dydis yra 10000, bet programoje galima parinkti bet kokį aibės dydį

Programa buvo leidžiama panaudojant tensorflow-metal kuris išnaudoja GPU.

Kompiuteris: M1 Macbook PRO
10 CPU branduoliai: 8 didelio efektyvumo ir 2 didelio pajėgumo
16 GPU branduolių
16 GB Atminties

https://www.tensorflow.org/tutorials/images/cnn

Pagal oficialius mokymus pasirinkau siūlomus neuroninio tinklo sluoksnius: Conv2D, MaxPooling2D, Flatten, Dense.

Aktyvacijos funkcija: Relu

Optimizacijos funkcija: Adam

Praradimo matavimo funkcija: Sparse Categorical Crossentropy

Modelio metrika: atspėjamų klasių procentas

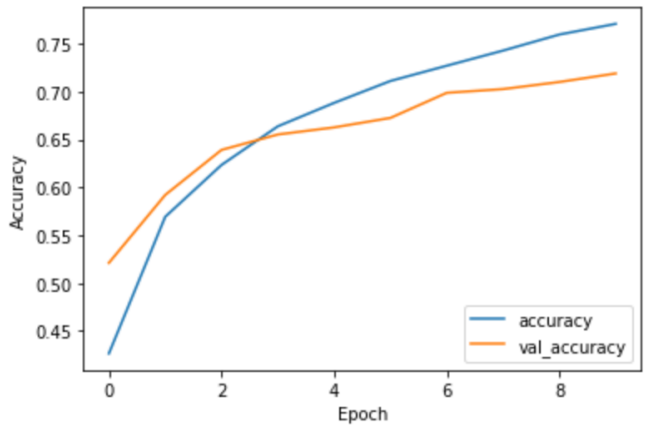
Epochos: 10

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu' ,
input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu' ))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu' ))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu' ))
model.add(layers.Dense(10))
model.compile(optimizer = 'adam' ,
                    loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                   metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                      validation_data=(test_images, test_labels), verbose
= 1, batch_size = 64)
```

```
Epoch 1/10
782/782 [==============================] - 75s 94ms/step - loss: 1.5680 - accuracy: 0.4278 - val_loss: 1.4137 - val_accuracy: 0.4924
Epoch 2/10
782/782 [==============================] - 72s 93ms/step - loss: 1.2136 - accuracy: 0.5676 - val_loss: 1.1530 - val_accuracy: 0.5945
Epoch 3/10
782/782 [==============================] - 71s 91ms/step - loss: 1.0495 - accuracy: 0.6298 - val_loss: 1.0380 - val_accuracy: 0.6261
Epoch 4/10
782/782 [==============================] - 71s 91ms/step - loss: 0.9432 - accuracy: 0.6701 - val_loss: 0.9380 - val_accuracy: 0.6721
Epoch 5/10
782/782 [==============================] - 74s 94ms/step - loss: 0.8695 - accuracy: 0.6969 - val_loss: 1.0135 - val_accuracy: 0.6610
Epoch 6/10
782/782 [==============================] - 72s 92ms/step - loss: 0.8096 - accuracy: 0.7183 - val_loss: 1.0757 - val_accuracy: 0.6451
Epoch 7/10
782/782 [==============================] - 72s 92ms/step - loss: 0.7638 - accuracy: 0.7341 - val_loss: 0.8293 - val_accuracy: 0.7132
Epoch 8/10
782/782 [==============================] - 74s 94ms/step - loss: 0.7174 - accuracy: 0.7521 - val_loss: 0.8571 - val_accuracy: 0.7053
Epoch 9/10
782/782 [==============================] - 73s 93ms/step - loss: 0.6763 - accuracy: 0.7639 - val_loss: 0.8378 - val_accuracy: 0.7146
Epoch 10/10
782/782 [==============================] - 73s 94ms/step - loss: 0.6335 - accuracy: 0.7784 - val_loss: 0.8214 - val_accuracy: 0.7263
```

Kaip matome nuo 5 ar 6 epochos modelio tikslumo augimas nelabai stipriai augo.

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```




```
test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2)
print("Model accuracy: ", test_acc)
```

```
313/313 - 4s - loss: 0.8199 - accuracy: 0.7188 - 4s/epoch - 14ms/step
Model accuracy:  0.7188000082969666
```

Žinant kad buvo naudojama tik 10 epochų, 71.8% tikslumo rezultatas yra neblogas kaip pirmam bandymui.

## Tyrimas

Pasižiūrėsiu modelio reakciją keičiant skirtingus hyper parametrus

RELU SGD 32

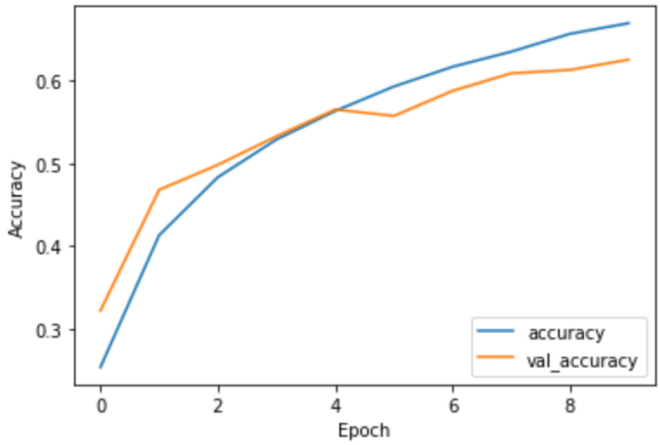
```
A = "relu"
O = tf.keras.optimizers.SGD()
B = 32

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation=A , input_shape=(32, 32,
3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation=A ))
model.add(layers.Dense(10))
model.compile(optimizer = O ,
                  loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
```

```
                    metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                      validation_data=(test_images, test_labels), verbose
= 2, batch_size = B)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2)
print("Model accuracy: ", test_acc)
```




```
313/313 - 4s - loss: 1.0804 - accuracy: 0.6254 - 4s/epoch - 13ms/step
Model accuracy:  0.6254000067710876
```

RELU SGD 64

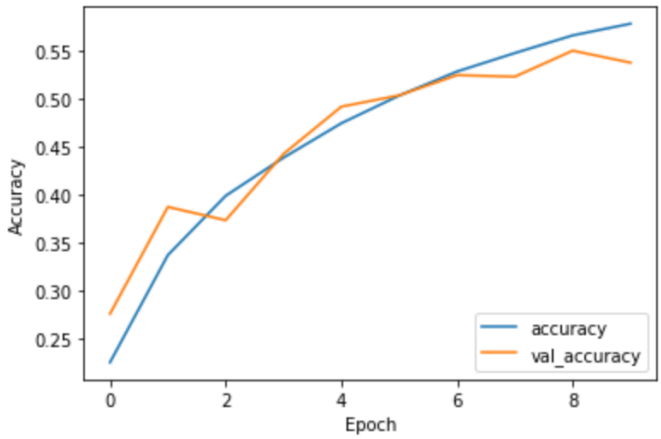
```
A = "relu"
O = tf.keras.optimizers.SGD()
B = 64

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation=A , input_shape=(32, 32,
3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation=A ))
model.add(layers.Dense(10))
model.compile(optimizer = O ,
```

```
                    loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                    metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                      validation_data=(test_images, test_labels), verbose
= 2, batch_size = B)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2)
print("Model accuracy: ", test_acc)
```




```
313/313 - 4s - loss: 1.2876 - accuracy: 0.5377 - 4s/epoch - 14ms/step
Model accuracy:  0.5376999974250793
```

RELU SGD 128

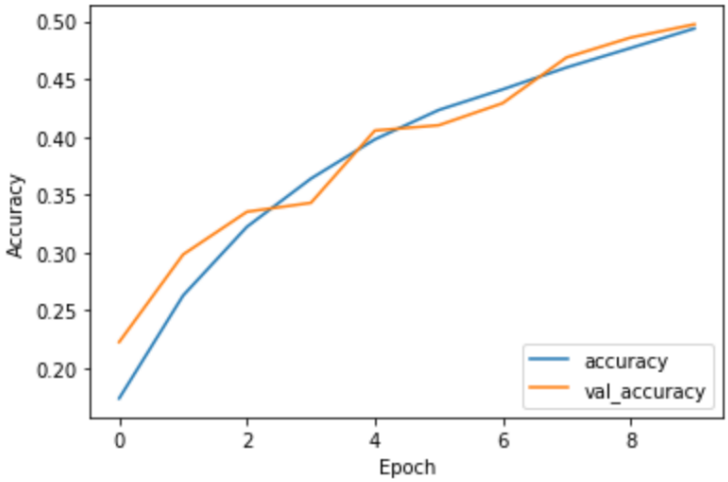
```
A = "relu"
O = tf.keras.optimizers.SGD()
B = 128

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation=A , input_shape=(32, 32,
3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation=A ))
```

```
model.add(layers.Dense(10))
model.compile(optimizer = O ,
                  loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels), verbose
= 2, batch_size = B)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2)
print("Model accuracy: ", test_acc)
```




```
313/313 - 4s - loss: 1.3892 - accuracy: 0.4974 - 4s/epoch - 14ms/step
Model accuracy:  0.4973999857902527
```

RELU ADAM 32

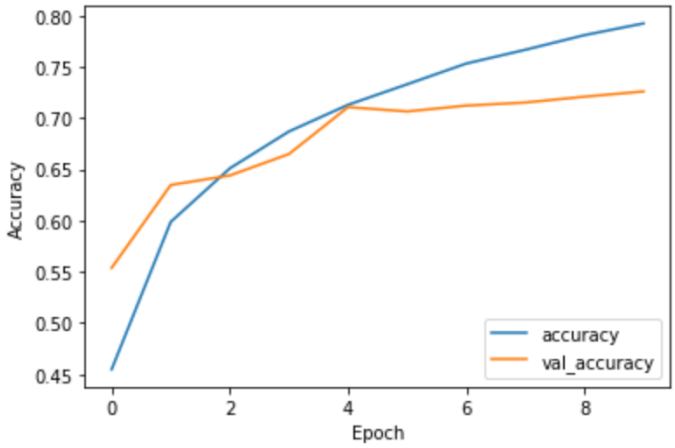
```
A = "relu"
O = tf.keras.optimizers.Adam()
B = 32

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation=A , input_shape=(32, 32,
3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
```

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation=A ))
model.add(layers.Dense(10))
model.compile(optimizer = O ,
                  loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels), verbose
= 2, batch_size = B)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2)
print("Model accuracy: ", test_acc)
```




```
313/313 - 4s - loss: 0.8408 - accuracy: 0.7260 - 4s/epoch - 13ms/step
Model accuracy:  0.7260000109672546
```

RELU ADAM 64

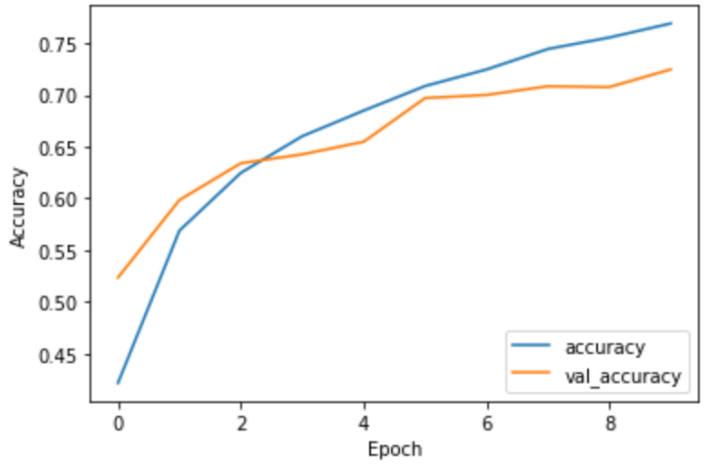
```
A = "relu"
O = tf.keras.optimizers.Adam()
B = 64

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation=A , input_shape=(32, 32,
3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.MaxPooling2D((2, 2)))
```

```
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation=A ))
model.add(layers.Dense(10))
model.compile(optimizer = O ,
                  loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels), verbose
= 2, batch_size = B)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2)
print("Model accuracy: ", test_acc)
```




```
313/313 - 4s - loss: 0.8144 - accuracy: 0.7248 - 4s/epoch - 12ms/step
Model accuracy:  0.7247999906539917
```

RELU ADAM 128

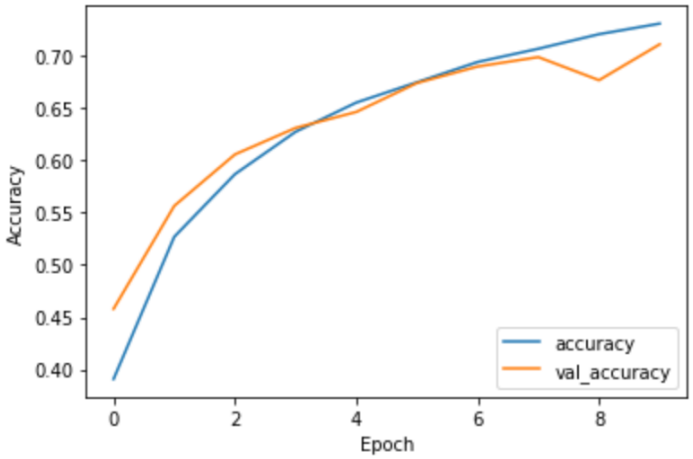
```
A = "relu"
O = tf.keras.optimizers.Adam()
B = 128

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation=A , input_shape=(32, 32,
3)))
model.add(layers.MaxPooling2D((2, 2)))
```

```
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation=A ))
model.add(layers.Dense(10))
model.compile(optimizer = O ,
                  loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels), verbose
= 2, batch_size = B)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2)
print("Model accuracy: ", test_acc)
```




```
313/313 - 4s - loss: 0.8353 - accuracy: 0.7114 - 4s/epoch - 14ms/step
Model accuracy:  0.7113999724388123
```

SOFTMAX SGD 32

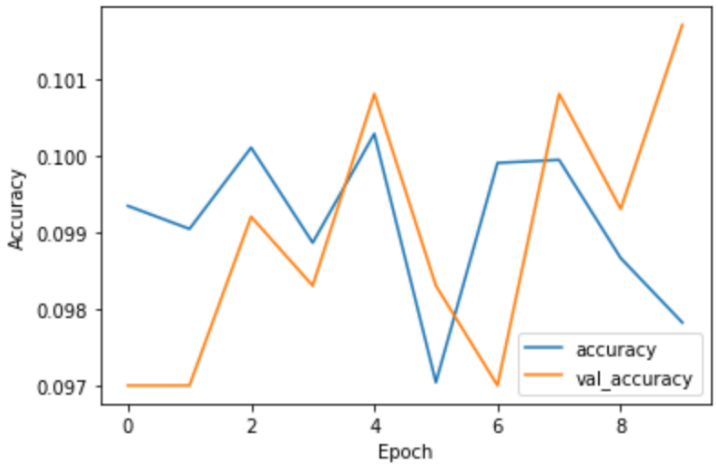
```
A = "softmax"
O = tf.keras.optimizers.SGD()
B = 32

model = models.Sequential()
```

```
model.add(layers.Conv2D(32, (3, 3), activation=A , input_shape=(32, 32,
3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation=A ))
model.add(layers.Dense(10))
model.compile(optimizer = O ,
                  loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels), verbose
= 2, batch_size = B)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2)
print("Model accuracy: ", test_acc)
```




```
313/313 - 6s - loss: 2.3026 - accuracy: 0.1017 - 6s/epoch - 19ms/step
Model accuracy:  0.10170000046491623
```

SOFTMAX SGD 64

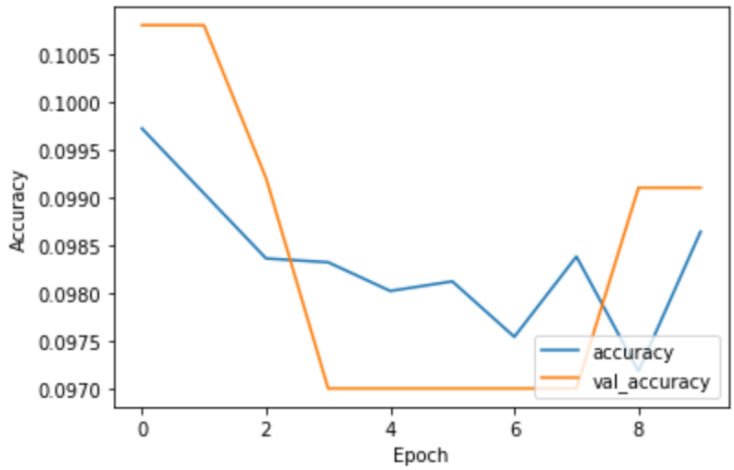
```
A = "softmax"
O = tf.keras.optimizers.SGD()
```

```
B = 64

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation=A , input_shape=(32, 32,
3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation=A ))
model.add(layers.Dense(10))
model.compile(optimizer = O ,
                  loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels), verbose
= 2, batch_size = B)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2)
print("Model accuracy: ", test_acc)
```




```
313/313 - 6s - loss: 2.3026 - accuracy: 0.0991 - 6s/epoch - 19ms/step
Model accuracy:  0.09910000115633011
```

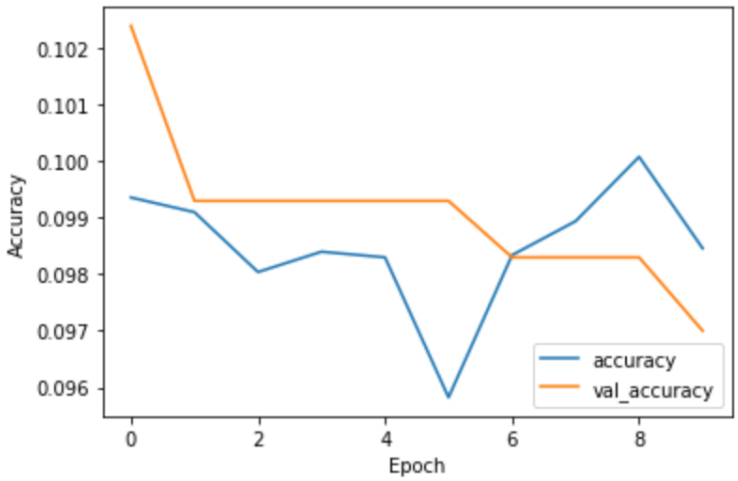
SOFTMAX SGD 128

```
A = "softmax"
O = tf.keras.optimizers.SGD()
B = 128

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation=A , input_shape=(32, 32,
3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation=A ))
model.add(layers.Dense(10))
model.compile(optimizer = O ,
                  loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels), verbose
= 2, batch_size = B)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2)
print("Model accuracy: ", test_acc)
```




```
313/313 - 6s - loss: 2.3027 - accuracy: 0.0970 - 6s/epoch - 19ms/step
Model accuracy:  0.09700000286102295
```

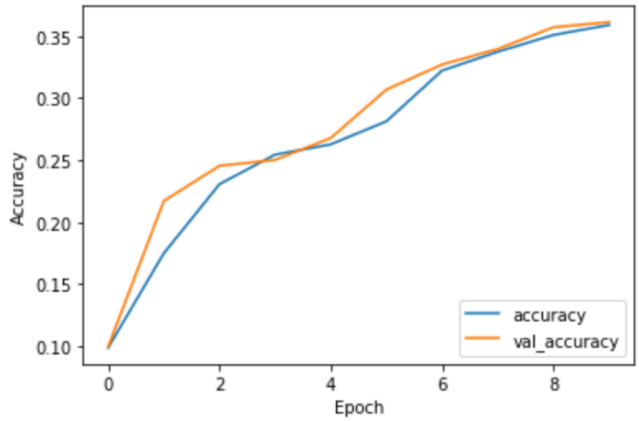
SOFTMAX ADAM 32

```
A = "softmax"
O = tf.keras.optimizers.Adam()
B = 32

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation=A , input_shape=(32, 32,
3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation=A ))
model.add(layers.Dense(10))
model.compile(optimizer = O ,
                  loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels), verbose
= 2, batch_size = B)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2)
print("Model accuracy: ", test_acc)
```




```
313/313 - 6s - loss: 1.6717 - accuracy: 0.3614 - 6s/epoch - 19ms/step
Model accuracy:  0.3614000082015991
```

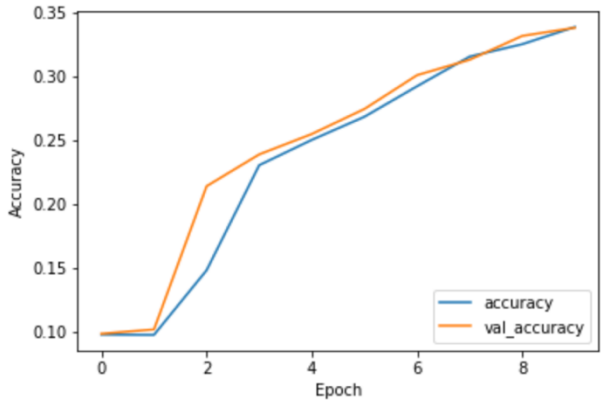
SOFTMAX ADAM 64

```
A = "softmax"
O = tf.keras.optimizers.Adam()
B = 64

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation=A , input_shape=(32, 32,
3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation=A ))
model.add(layers.Dense(10))
model.compile(optimizer = O ,
                  loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels), verbose
= 2, batch_size = B)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2)
print("Model accuracy: ", test_acc)
```




```
313/313 - 6s - loss: 1.7567 - accuracy: 0.3382 - 6s/epoch - 19ms/step
Model accuracy:  0.33820000290870667
```

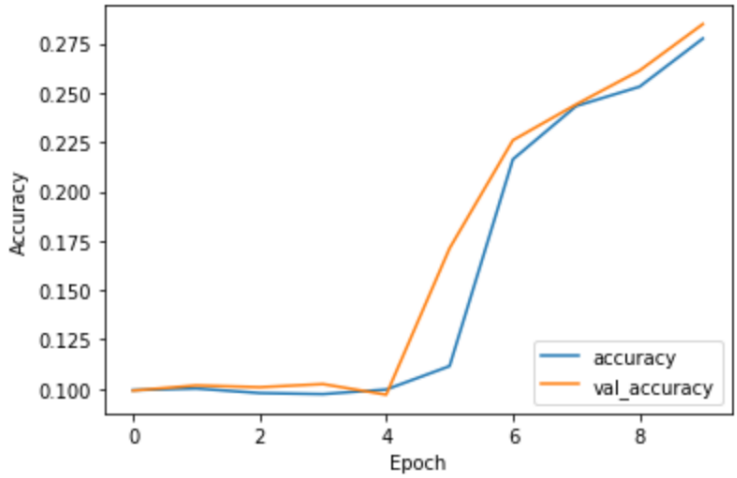
SOFTMAX ADAM 128

```
A = "softmax"
O = tf.keras.optimizers.Adam()
B = 128

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation=A , input_shape=(32, 32,
3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation=A ))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation=A ))
model.add(layers.Dense(10))
model.compile(optimizer = O ,
                  loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels), verbose
= 2, batch_size = B)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2)
print("Model accuracy: ", test_acc)
```




```
313/313 - 6s - loss: 1.9388 - accuracy: 0.2849 - 6s/epoch - 19ms/step
Model accuracy:  0.2849000096321106
```

## Rezultatai

| Optimizer | Activation | Batch_size | Accuracy |
|---|---|---|---|
| adam | relu | 32 | 0,726 |
| | | 64 | 0,724 |
| | | 128 | 0,711 |
| | softmax | 32 | 0,361 |
| | | 64 | 0,338 |
| | | 128 | 0,284 |
| sgd | relu | 32 | 0,625 |
| | | 64 | 0,538 |
| | | 128 | 0,497 |
| | softmax | 32 | 0,101 |
| | | 64 | 0,099 |
| | | 128 | 0,097 |

Geriausias rezultatas kai:
Optimizavimo algoritmas: Adam
Aktyvacijos funkcija: Relu
Paketo dydis: 32
Rezultatas: 0.726

```
def get_n_test_data(n):
    (train_images, train_labels), (test_images, test_labels) =
datasets.cifar10.load_data()
    X = np.concatenate((train_images,test_images))
    Y = np.concatenate((train_labels,test_labels))
```

```
    train_images, test_images, train_labels, test_labels =
train_test_split(X, Y, test_size=n, random_state=4)
    train_images, test_images = train_images / 255.0, test_images /
255.0
    return test_images, test_labels
```

```
x30, y30 = get_n_test_data(30)
```

Prognozuojant nuo 30 atsitiktinai pasirinktų duomenų:

```
test_loss, test_acc = model.evaluate(x30,  y30)
print("Model accuracy: ", test_acc)
```

```
1/1 [==============================] - 0s 33ms/step - loss: 0.6106 - accuracy: 0.8000
Model accuracy:  0.800000011920929
```

Confusion matrix:

```
y_pred = model.predict(test_images)
con_mat = tf.math.confusion_matrix(labels=test_labels,
predictions=[np.argmax(i) for i in y_pred]).numpy()
con_mat
```

```
313/313 [==============================] - 5s 16ms/step
array([[803,  16,  36,  15,  32,  14,   8,  11,  51,  38],
       [ 23, 830,   5,   6,   1,   6,  18,   2,  25,  77],
       [ 83,  11, 614,  33, 107,  54,  57,  15,  10,   7],
       [ 24,   9,  70, 480,  88, 183,  71,  19,  10,  16],
       [ 20,   3,  43,  32, 761,  42,  34,  37,   7,   4],
       [  6,   2,  48, 127,  68, 694,  29,  27,   4,  12],
       [  4,   6,  37,  37,  54,  35, 838,   5,   6,   4],
       [ 19,   7,  28,  45, 127,  78,  11, 655,   5,  17],
       [ 57,  25,  16,  12,  16,  12,  12,   1, 803,  42],
       [ 23,  84,   7,  12,   6,  10,   7,   7,  29, 823]], dtype=int32)
```

```
classes =
["airplane","automobile","bird","cat","deer","dog","frog","horse","ship
","truck"]
for i,j in zip(model.predict(x30, verbose=2), y30):
    pred = classes[np.argmax(i)]
    fact = classes[j[0]]
    print(pred == fact, ": Predicted:", pred ," True:", fact)
```

executed by Simona Gelzinyte
15:33 (0 minutes ago)
executed in 0.526 s

```
/epoch - 32ms/step
d: deer  True: deer
ed: deer  True: horse
True : Predicted: bird  True: bird
True : Predicted: deer  True: deer
True : Predicted: truck  True: truck
True : Predicted: deer  True: deer
True : Predicted: deer  True: deer
True : Predicted: deer  True: deer
True : Predicted: airplane  True: airplane
True : Predicted: dog  True: dog
True : Predicted: deer  True: deer
False : Predicted: cat  True: dog
True : Predicted: automobile  True: automobile
True : Predicted: ship  True: ship
False : Predicted: ship  True: airplane
True : Predicted: automobile  True: automobile
True : Predicted: frog  True: frog
True : Predicted: truck  True: truck
True : Predicted: dog  True: dog
True : Predicted: deer  True: deer
True : Predicted: frog  True: frog
True : Predicted: ship  True: ship
True : Predicted: truck  True: truck
False : Predicted: bird  True: cat
False : Predicted: airplane  True: horse
True : Predicted: airplane  True: airplane
True : Predicted: dog  True: dog
False : Predicted: dog  True: cat
True : Predicted: bird  True: bird
True : Predicted: automobile  True: automobile
```

# IŠVADOS

- Didžiausią itaką modelio efektytvumui dare aktyvacijos funkcija;
- Modeliai su mažesnėm batch_size reikšmėm pasirode geresni;
- Adam optimizavimo funkcija pasirode stipriai geresnė nei SGD;
- Imant epochų skaičių <10, modelio taiklumas stipriai mažėja
- Modelio spėjimas su 30 duomenų eilučių davė geresnius rezulatus nei buvo įvertintas modelis.
- Ten kur modelis suklydo, dažniausia klaida yra gyvūnų rūšių sumaišymas.
- Geriausiai atspėta klasė yra elnias.
- Blogiausiai atspėta klasė yra taip katė ir arklys.