Vilniaus Universitetas

Matematikos ir Informatikos Fakultetas
Duomenų mokslas III kursas   2 grupė

Matas Gaulia

4 užduotis

2021

Duomenys: 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

Prepare data

```
In [30]: (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 mokytusi, reikia normalizuoti duomenis, tad padalinu reikšmes iš 255, kad gautusi 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 AIR
8 CPU branduoliai: 4 didelio efektyvumo ir 4 didelio pajėgumo
7 GPU branduoliai
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

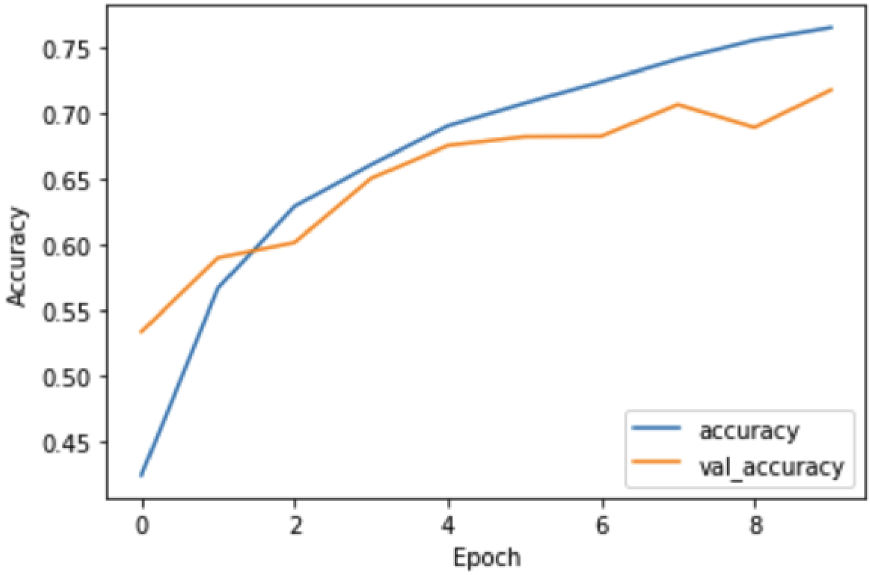
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu' , input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu' ))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu' ))
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
  5/782 [..............................] - ETA: 10s - loss: 2.3064 - accuracy: 0.0813
2021-12-14 05:23:35.293377: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:112] Plugin opti
mizer for device_type GPU is enabled.
782/782 [==============================] - ETA: 0s - loss: 1.5833 - accuracy: 0.4237
2021-12-14 05:23:44.523361: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:112] Plugin opti
mizer for device_type GPU is enabled.
782/782 [==============================] - 10s 13ms/step - loss: 1.5833 - accuracy: 0.4237 - val_loss: 1.2872 - val_a
ccuracy: 0.5334
Epoch 2/10
782/782 [==============================] - 10s 13ms/step - loss: 1.2183 - accuracy: 0.5667 - val_loss: 1.1486 - val_a
ccuracy: 0.5899
Epoch 3/10
782/782 [==============================] - 10s 12ms/step - loss: 1.0560 - accuracy: 0.6292 - val_loss: 1.1253 - val_a
ccuracy: 0.6016
Epoch 4/10
782/782 [==============================] - 10s 13ms/step - loss: 0.9715 - accuracy: 0.6612 - val_loss: 0.9900 - val_a
ccuracy: 0.6507
Epoch 5/10
782/782 [==============================] - 10s 13ms/step - loss: 0.8874 - accuracy: 0.6907 - val_loss: 0.9321 - val_a
ccuracy: 0.6756
Epoch 6/10
782/782 [==============================] - 10s 13ms/step - loss: 0.8387 - accuracy: 0.7080 - val_loss: 0.9211 - val_a
ccuracy: 0.6823
Epoch 7/10
782/782 [==============================] - 10s 13ms/step - loss: 0.7906 - accuracy: 0.7245 - val_loss: 0.9123 - val_a
ccuracy: 0.6826
Epoch 8/10
782/782 [==============================] - 10s 13ms/step - loss: 0.7436 - accuracy: 0.7415 - val_loss: 0.8371 - val_a
ccuracy: 0.7070
Epoch 9/10
782/782 [==============================] - 10s 13ms/step - loss: 0.7010 - accuracy: 0.7562 - val_loss: 0.9174 - val_a
ccuracy: 0.6894
Epoch 10/10
782/782 [==============================] - 11s 13ms/step - loss: 0.6727 - accuracy: 0.7659 - val_loss: 0.8377 - val_a
ccuracy: 0.7183
```

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

Taip pat matome kad nėra per didelio mokinimo (over-fitting) problemos, nes modelio metrika tolygiai 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)
```

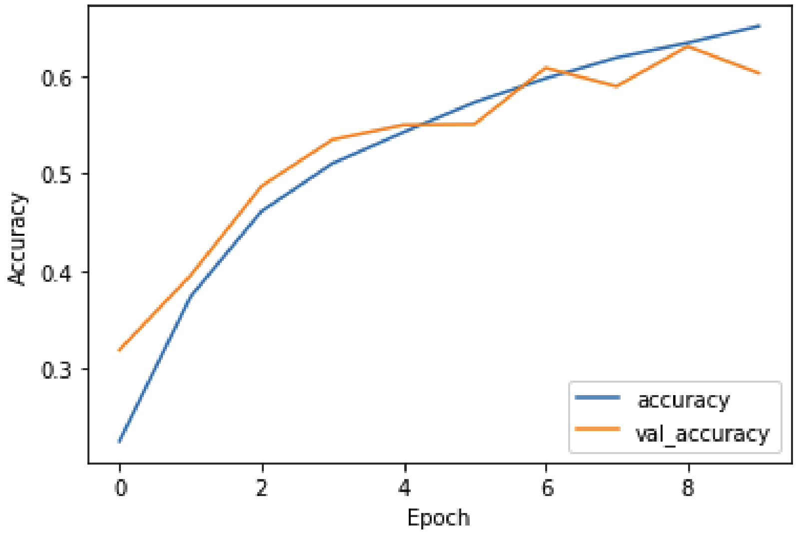
```
313/313 - 1s - loss: 0.8377 - accuracy: 0.7183 - 1s/epoch - 5ms/step
Model accuracy:  0.7183000445365906
```

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

Tyrimas:

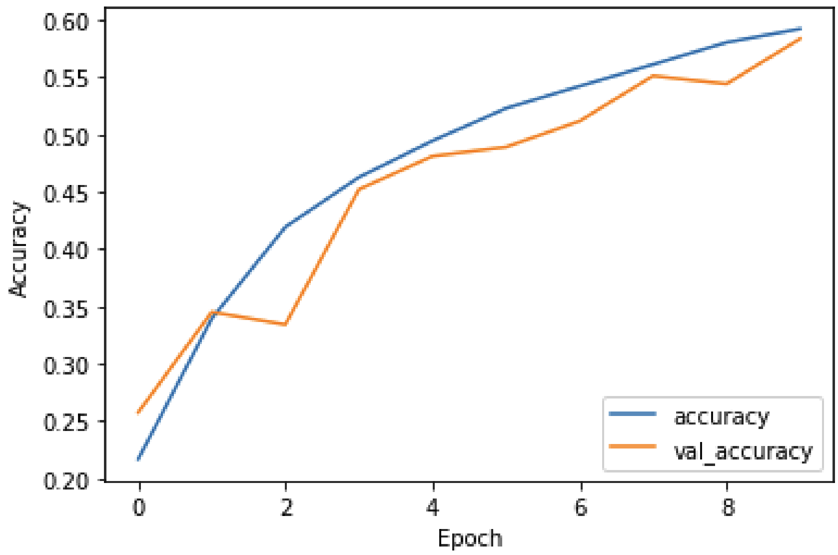
Pasižiūrėsiu modelio reakciją keičiant skirtingus hyper parametrus

RELU SGD 32



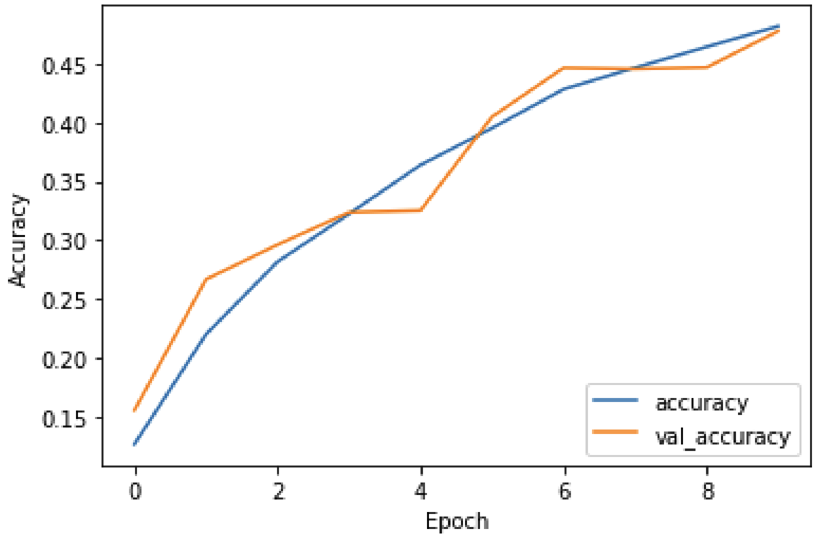

```
313/313 - 1s - loss: 1.1242 - accuracy: 0.6028 - 1s/epoch - 5ms/step
Model accuracy:  0.6028000116348267
```

RELU SGD 64



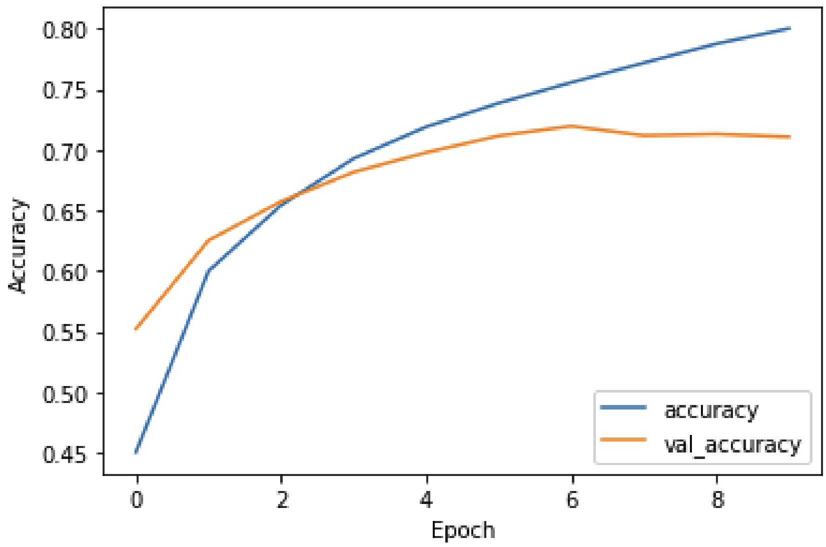

```
313/313 - 1s - loss: 1.1810 - accuracy: 0.5836 - 1s/epoch - 4ms/step
Model accuracy:  0.5836000442504883
```

RELU SGD 128



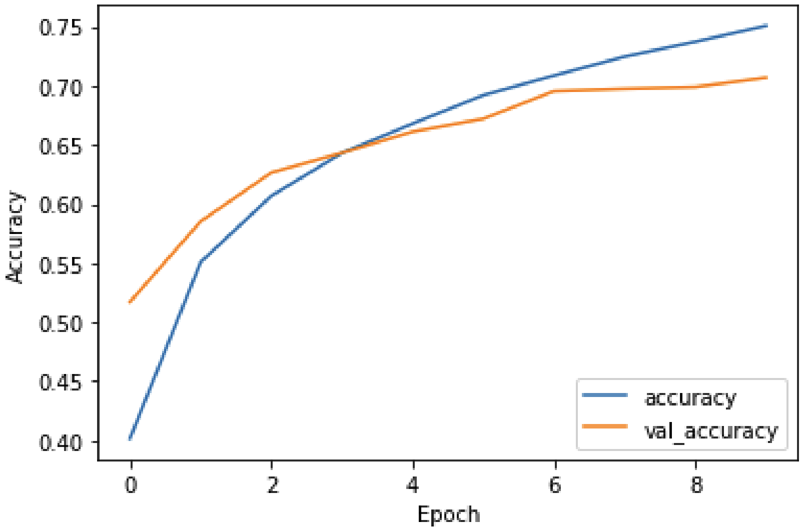

```
313/313 - 1s - loss: 1.4451 - accuracy: 0.4774 - 1s/epoch - 5ms/step
Model accuracy:  0.47740003466606140
```

RELU ADAM 32



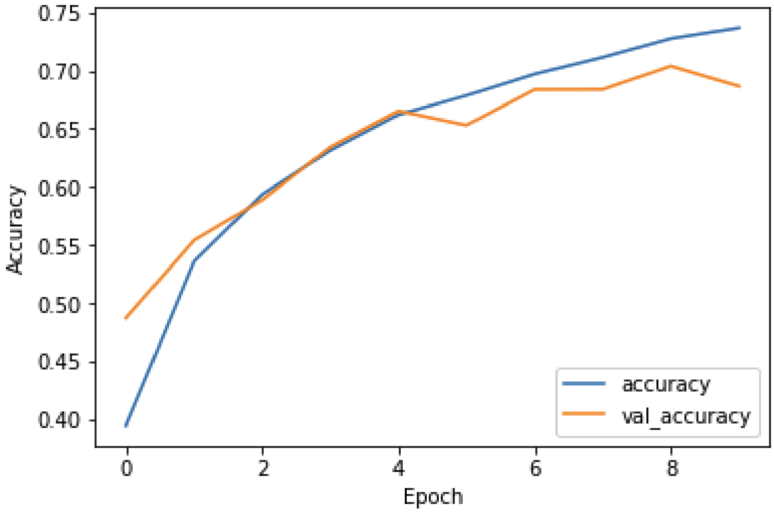

```
313/313 - 1s - loss: 0.8680 - accuracy: 0.7108 - 1s/epoch - 4ms/step
Model accuracy:  0.710800051689148
```

RELU ADAM 64



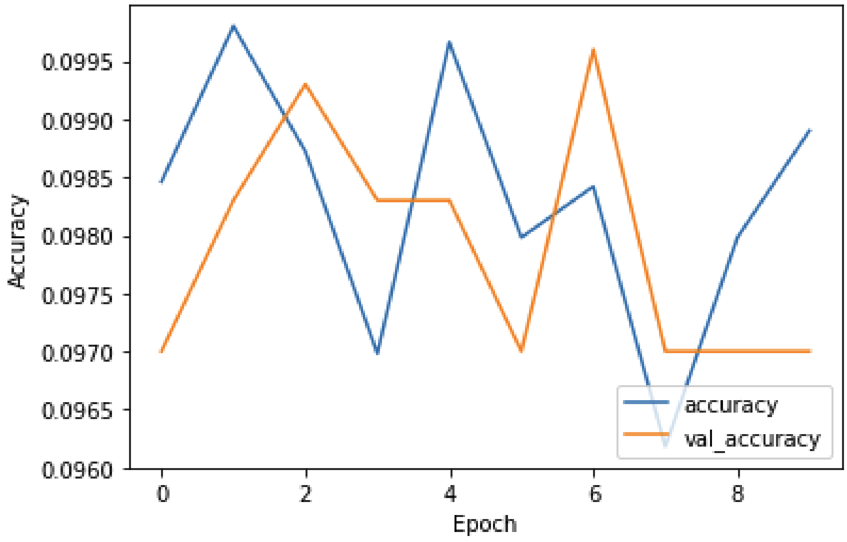

```
313/313 - 1s - loss: 0.8464 - accuracy: 0.7068 - 1s/epoch - 5ms/step
Model accuracy:  0.7068000435829163
```

RELU ADAM 128



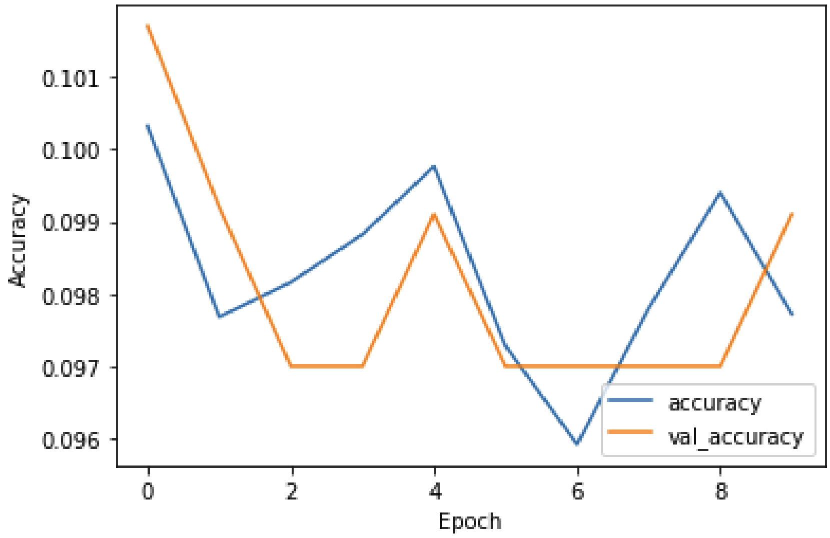

```
313/313 - 1s - loss: 0.9046 - accuracy: 0.6866 - 1s/epoch - 5ms/step
Model accuracy:  0.6866000294685364
```

SOFTMAX SGD 32



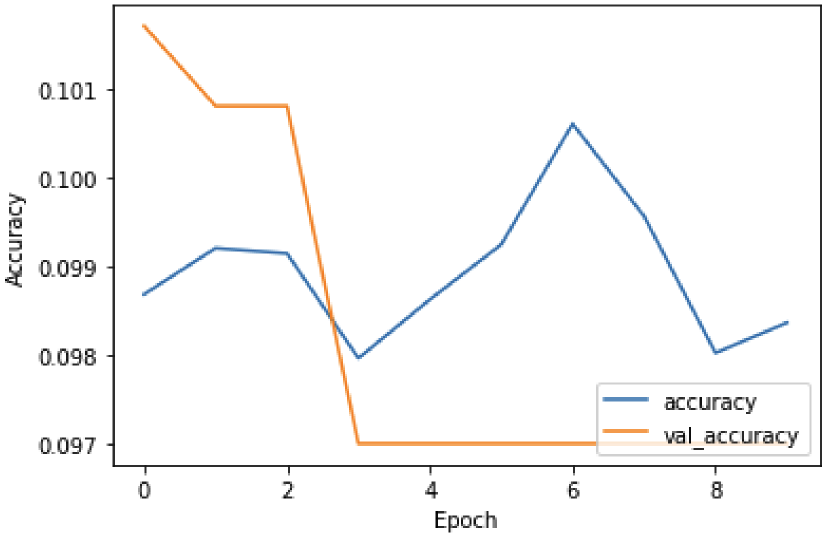

```
313/313 - 2s - loss: 2.3027 - accuracy: 0.0970 - 2s/epoch - 8ms/step
Model accuracy:  0.09700000286102295
```

SOFTMAX SGD 64



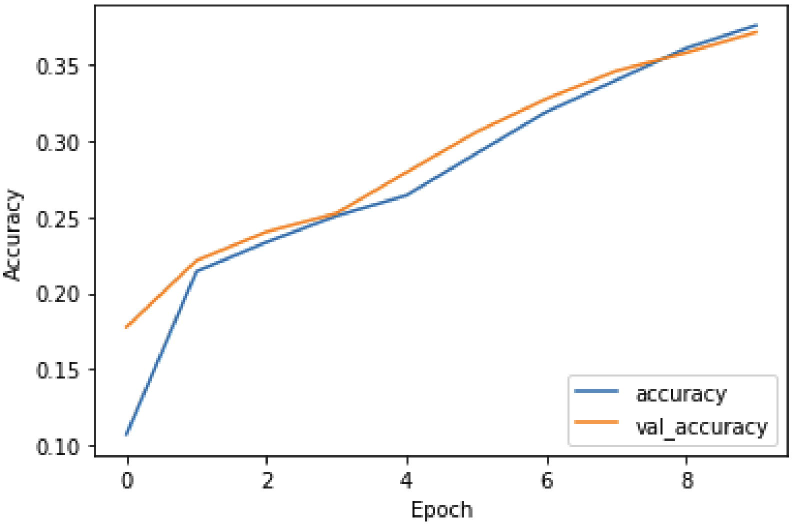

```
313/313 - 2s - loss: 2.3027 - accuracy: 0.0991 - 2s/epoch - 7ms/step
Model accuracy:  0.09910000115633011
```

SOFTMAX SGD 128



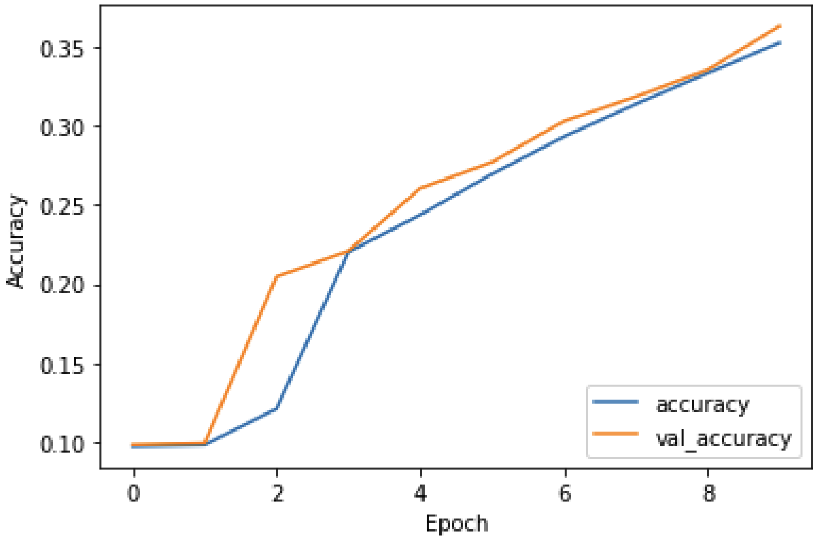

```
313/313 - 2s - loss: 2.3026 - accuracy: 0.0970 - 2s/epoch - 8ms/step
Model accuracy:  0.09700000286102295
```

SOFTMAX ADAM 32



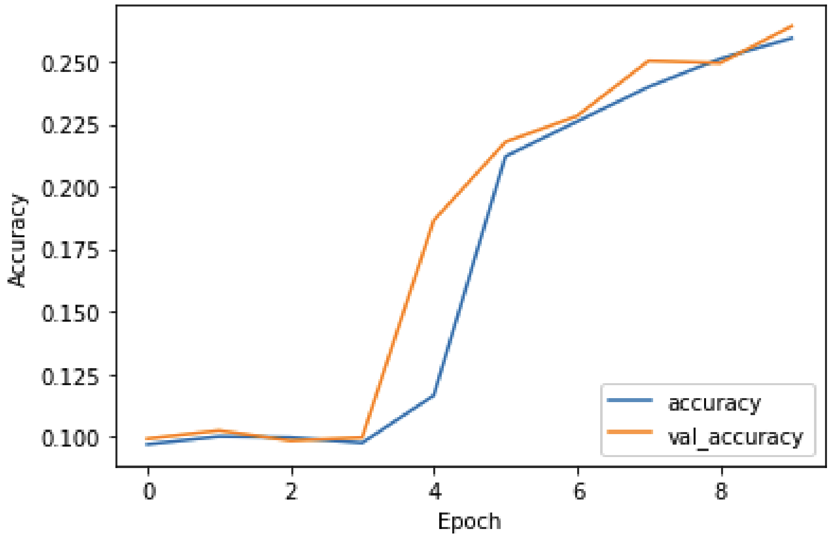

```
313/313 - 2s - loss: 1.6502 - accuracy: 0.3709 - 2s/epoch - 7ms/step
Model accuracy:  0.3709000051021576
```

SOFTMAX ADAM 64




```
313/313 - 2s - loss: 1.7451 - accuracy: 0.3628 - 2s/epoch - 7ms/step
Model accuracy:  0.3628000319004059
```

SOFTMAX ADAM 128




```
313/313 - 2s - loss: 1.9402 - accuracy: 0.2643 - 2s/epoch - 7ms/step
Model accuracy:  0.26430001854896545
```

Rezultatai

Optimizer	Activation	Batch size	
adam	relu	32	0.7100
		64	0.7070
		128	0.6870
	softmax	32	0.3710
		64	0.3630
		128	0.2640
sgd	relu	32	0.6030
		64	0.5840
		128	0.4770
	softmax	32	0.0970
		64	0.0990
		128	0.0970

Geriausias rezultatas kai:
Optimizavimo algoritmas: Adam
Aktyvacijos funkcija: Relu
Paketo dydis: 32
Rezultatas: 0.71

```
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 195ms/step - loss: 0.6619 - accuracy: 0.8667
Model accuracy:  0.8666667342185974
```

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
```

```
array([[767,  25,  47,  13,  15,   7,   7,  22,  99,  22],
       [ 21, 787,   5,  13,   3,   3,  21,   5,  46,  89],
       [ 69,   7, 645,  36,  44,  62,  68,  38,  17,   5],
       [ 24,   7,  59, 489,  26, 202,  81,  36,  36,  10],
       [ 39,   4,  97,  66, 567,  53,  55,  82,  12,   8],
       [ 12,   2,  53, 185,  22, 653,  30,  47,   9,   4],
       [  7,   9,  47,  62,  20,  19, 843,   9,   7,   3],
       [ 15,   3,  33,  42,  51,  66,  11, 738,  10,  23],
       [ 57,  34,  11,  16,   5,   4,   5,   3, 838,  23],
       [ 24,  74,   6,  20,   1,   9,   5,  10,  38, 821]], 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)
```

```
1/1 - 0s - 14ms/epoch - 14ms/step
True : Predicted: deer  True: deer
True : Predicted: horse  True: horse
True : Predicted: bird  True: bird
False : Predicted: horse  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
True : Predicted: dog  True: dog
True : Predicted: automobile  True: automobile
False : Predicted: airplane  True: ship
True : Predicted: airplane  True: airplane
True : Predicted: automobile  True: automobile
True : Predicted: frog  True: frog
True : Predicted: truck  True: truck
True : Predicted: dog  True: dog
False : Predicted: cat  True: deer
True : Predicted: frog  True: frog
True : Predicted: ship  True: ship
True : Predicted: truck  True: truck
False : Predicted: airplane  True: cat
True : Predicted: horse  True: horse
True : Predicted: airplane  True: airplane
True : Predicted: dog  True: dog
True : Predicted: cat  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 ant 30 duomenų eilučių davė geresnius rezulatus nei buvo įvertintas modelis
- Ten kur modelis suklydo, dažniausia klaida yra gyvūnų rūšių sumaišymas, tai gali būti dėl to kad paveiksliukų raiška labai stipriai sumažinta
- Geriausiai atspėta klasė yra varlės
- Blogiausiai atspėta klasė yra katės