Kauno technologijos universitetas

Informatikos fakultetas

Gilusis mokymas. Laboratorinis darbas nr. 2

Laboratorinio darbo ataskaita

Greta Intaitė

Studentas / Studentė

Kaunas, 2024

Turinys

Įvadas

Modulio „Gilusis mokymasis“, kurio kodas yra KODAS antrojo laboratorinio darbo metu analizuojamos skirtingos neuroninių tinklų architektūros, jų sudarymas ir tinkamumo įvertinimas klasifikavimo uždaviniams. Tuo pat metu mokomasi naudotis bibliotekos „Tensorflow“ „Keras“ modeliais, duomenų apdorojimo ir paruošimo metodais. Šis tikslas pasiekiamas atliekant 2 penkių balų vertės užduotis:

1. Pagal priskirtą duomenų rinkinį paruošus duomenų aibes skirtingiems mokymo ir panaudojimo etapams, pagal duotas modelio architektūrų schemas sudaromi ir apmokomi modeliai, pateikiami jų apmokymo rezultatai, jie ištestuojami ir įvertinamas modelių tinkamumas pasirinkto uždavinio sprendimui. To paties uždavinio sprendimui sudaroma nauja architektūra, įvertinamas optimalus duomenų kiekis. Rezultatai išanalizuojami.

2. Surinkus savo duomenų rinkinį pasirinktos problemos sprendimui apmokomas pasirinktos architektūros modelis. Jo rezultatai palyginami, kai naudojami skirtingi apmokymo metodai ir duomenų papildymas.

Atlikus šias užduotis gebama paaiškinti, kaip veikia ir yra paruošiami neuroninių tinklų modeliai naudojant „Tensorflow“ ir „Keras“ karkasus, kaip juos pritaikyti sprendžiant realias problemas.

1. Pirma užduotis

Pirmos užduoties metu paruošiami duomenys apibendrinant „Fashion MNIST“ rinkinio duomenų klases pagal dėstytojo pateiktą užduoties variantą. Pagal individualų variantą parenkamos architektūros, kurios realizuojamos naudojant „Tensorflow“ ir „Keras“ karkasus. Modelių su skirtingomis architektūromis veikimas išanalizuojamas ir palyginamas tarpusavyje.

Variantai priskiriami pagal studento kodą. Šiame laboratoriniame darbe nagrinėjamas 6 numeris.

Eksperimentų reprodukcijai naudojama *set_seed*() funkcija su argumento reikšme 1001.

1.1. Duomenų paruošimas

Šeštam numeriui priskiriamas 1 duomenų rinkinio variantas. Originaliame duomenų rinkinyje yra 10 klasių, į kurias skirstomos drabužių tipų nuotraukos. Pagal 1 duomenų variantą šios klasės sujungiamos į 3 pagal žemiau pateiktą lentelę:

1 lentelė. Naudojamos klasės. Originalių klasių apjungimas

Klasės nr.	**Aprašai**	**Originalūs klasės nr.**	**Originalios klasės**
0	Upper clothes (liet. viršutinės kūno dalies drabužiai)	0, 2, 3, 4, 6	T-shirt/top, pullover, dress, coat, shirt
1	Lower clothes (liet. apatinės kūno dalies drabužiai)	1	Trouser
2	Shoes (liet. avalynė)	5, 7, 9	Sandal, sneaker, ankle boot

Galima pastebėti, kad pateiktoje lentelėje nėra numeriu 8 pažymėtos klasės, pavadinimu „Bag“. Transformuojant originalų duomenų rinkinį šį klasė turės būti pašalinta.

Šioms transformacijoms naudojamos funkcijos:

```
def replace_classes(arr, values, num_new):
  '''
  arr - original array of class names/numbers
  values - 2d array with collections of classes from original dataset grouped
          by their new values
  num_new - int value that matches the number of new classes
  -----
  Returns:
  arr - original array modified so that all classes are replaced by the number
        of their group
  not_in_legit_vals_indexes - indexes of rows that do not belong to any of the
                              new groups
  '''
  legit_vals = []
  for i in range(num_new):
    arr = np.where(np.isin(arr, values[i]), i, arr)
    legit_vals.append(i)
```

```
  print(legit_vals)
  not_in_legit_vals_indexes = np.where(~np.isin(arr, legit_vals))[0]
  return arr, not_in_legit_vals_indexes

def remove_classes(xs, ys, indexes):
  '''
  xs - numpy arrays of images
  ys - array of classes
  indexes - array of indexes of rows that should be deleted from xs and ys
  ---
  Returns:
  xs, ys - modified image and label arrays
  '''
  xs = np.delete(xs, indexes, axis=0)
  ys = np.delete(ys, indexes, axis=0)
  return xs, ys
```

Pašalinus 8 klasę iš 70000 lieka 63000 eilučių, kurias galima panaudoti apmokymui.

Likusios eilutės padalinamos į apmokymo, testavimo ir validavimo duomenų rinkinius taip, kad 70% eilučių atitenka apmokymo imčiai, 15% testavimo ir 15% validavimo. Neatsižvelgus į klasių disbalansą gaunamas modelis, kuris bet kokiu atveju renkasi klasę, kuriai priklausančių duomenų buvo daugiausia apmokymo imtyje. Todėl nepaisant gero tikslumo apmokymo ir testavimo imtims, pabandžius pritaikyti tokį modelį, pastebima, kad klasifikacijos rezultatai yra klaidingi. Į klasių pasiskirstymo disbalansą duomenų rinkinyje atsižvelgiama klases, kurioms priklauso mažiau duomenų, papildant jau egzistuojančių duomenų modifikuotomis versijomis.

1.2. Modelių architektūrų realizavimas ir apmokymas

Paruošus apmokymo, testavimo ir validavimo duomenų rinkinius, pereinama prie architektūrų realizavimo naudojant „Keras“ karkasą.

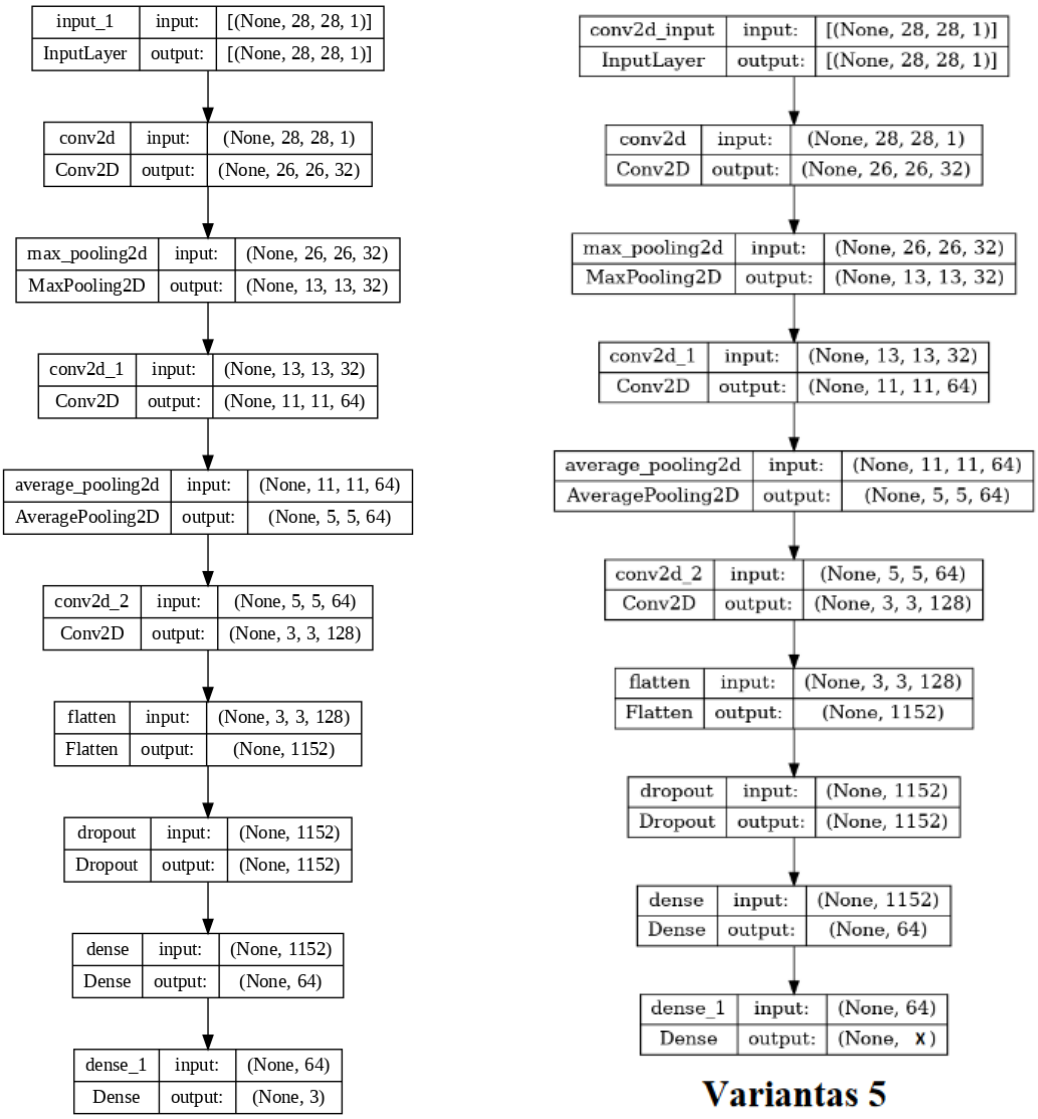
Šioje ataskaitoje aprašomas užduoties variantas realizuoja architektūras, kurių numeriai užduoties lape yra 5, 4, 10. Toliau bus pateikiama kiekvienos architektūros realizacija, schema, apmokymo ir testavimo procesai ir gauti rezultatai.

1.2.1. Penktas architektūros variantas

Penktas architektūros variantas sudarytas iš 10 sluoksnių. Panaudojant žemiau pateiktą kodą gaunama NR paveiksliuke pateikta modelio architektūros schema, ji palyginama su užduotyje pateikta schema.

```
model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPool2D(strides=(2,2)),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.AveragePooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(128, 3, activation='relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.2),
```

```
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(3, activation='relu')
])
```



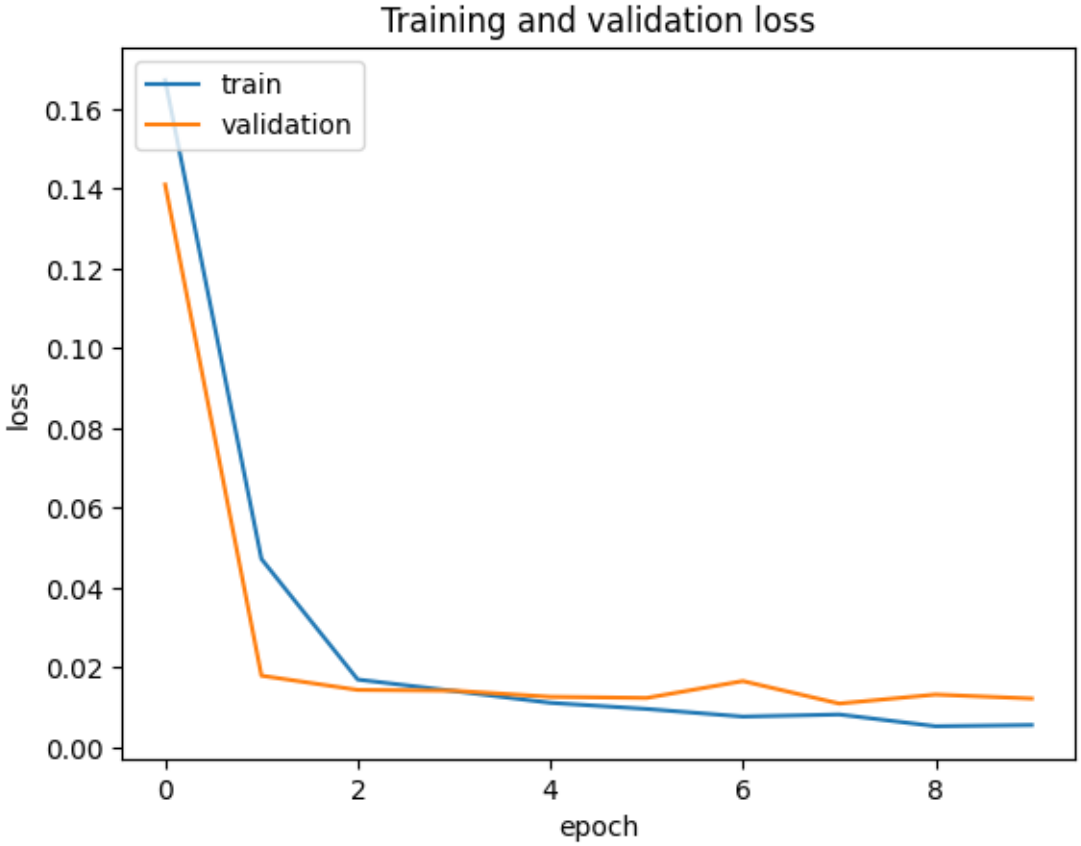
1 pav. kairėje – sudaryto modelio architektūros schema, dešinėje – užduotyje pateikta schema.

Įvertinus abi schemas, galima daryti išvadą, kad modelis sudarytas teisingai, kadangi atitinka sluoksnių tipai ir neuronų skaičiai. Paskutiniame sluoksnyje neuronų skaičius X pakeičiamas išeities klasių skaičiumi, šiuo atveju – 3.

Pradžioje, apmokant kiekvieną modelį naudojama 10 epochų. Stebint skirtumus tarpo validavimo ir apmokymo imties tikslumo ir paklaidos rezultatų, parenkama optimaliausia epocha ir sustabdomas modelio treniravimas. Tai reikalinga siekiant sutaupyti skaičiavimams reikalingų resursų ir išvengti persimokymo situacijos (angl. overfitting), kai modelis per daug prisitaiko prie apmokymo duomenų ir pateikia prastus rezultatus su nematytais duomenimis.

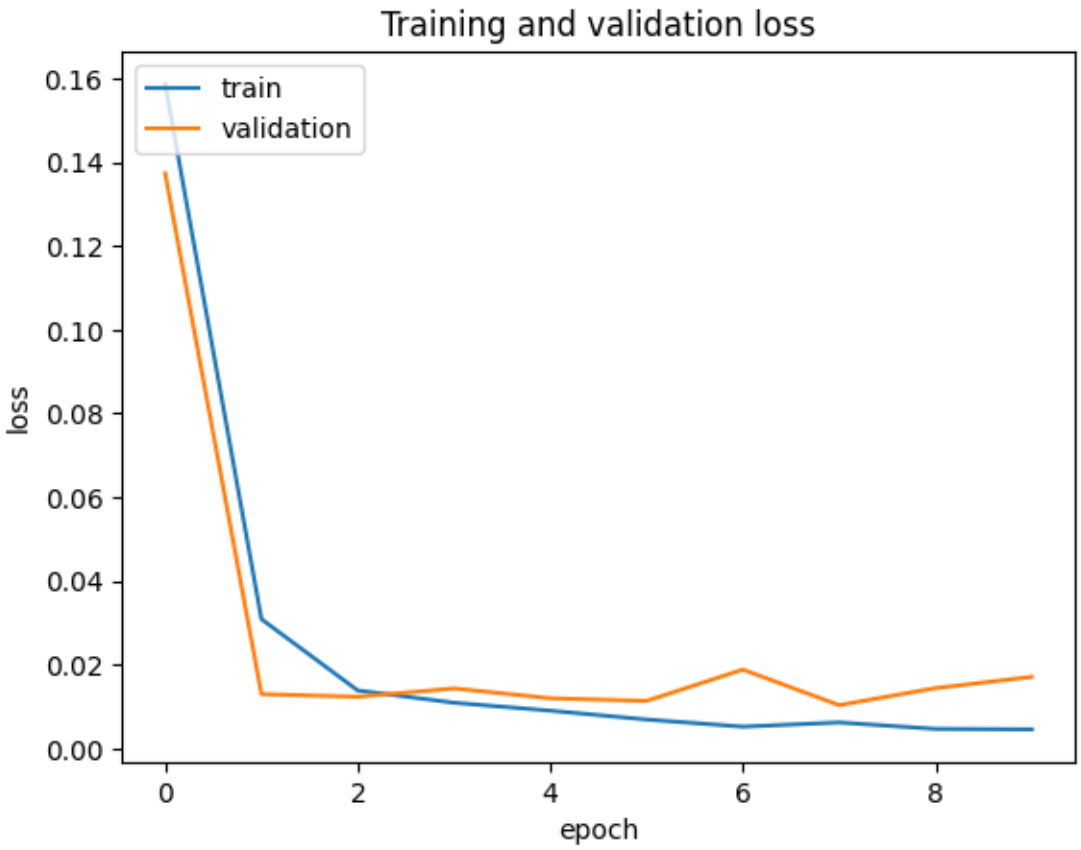
Apmokymo metu pateikiamas grafikas padeda įvertinti optimaliausią epochų skaičių ir stebėti, kai keičiasi modelio klasifikacijos tikslumas laiko, epochų atžvilgiu (žr. 2 pav.). Įvertinus šį grafiką, pastebima, kad aukštas tikslumas pasiekiamas jau po pirmų 2 epochų. Validavimo ir apmokymo imčių praradimo funkcijos reikšmės sąlyginai nedidelės ir sutampa ties 3 epocha, o nuo ketvirtos

validavimo imties funkcijos reikšmė ima didėti, kol apmokymo imties funkcijos rezultatas ir toliau pamažu mažėja. Atsižvelgiant į šiuos pastebėjimus nuspręsta apmokymą vykdyti 3 epochas.



2 pav. penktos architektūros modelio apmokymo procesas

3 pav. Teisingų ir klaidingų priskyrimų pasiskirstymas pagal klases



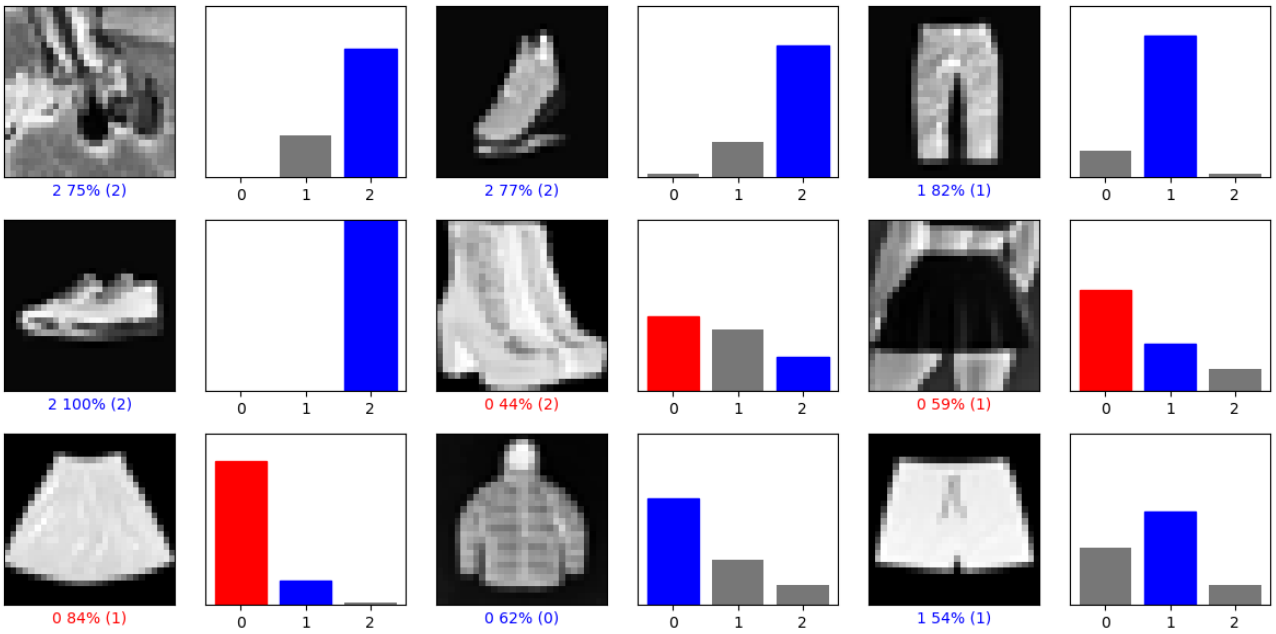
4 pav. ketvirtos architektūros modelio apmokymo procesas

5 pav. Teisingų ir klaidingų priskyrimų pasiskirstymas pagal klases

6 pav. dešimtos architektūros modelio apmokymo procesas

7 pav. Teisingų ir klaidingų priskyrimų pasiskirstymas pagal klases

Gauti modeliai išsaugoti „[numeris].keras“ formatu ir naujame faile išbandyti su drabužių nuotraukomis paimtomis iš internetinių parduotuvių ir reklamų. Tiksliausi rezultatai gauti su dešimtos architektūros modeliu:



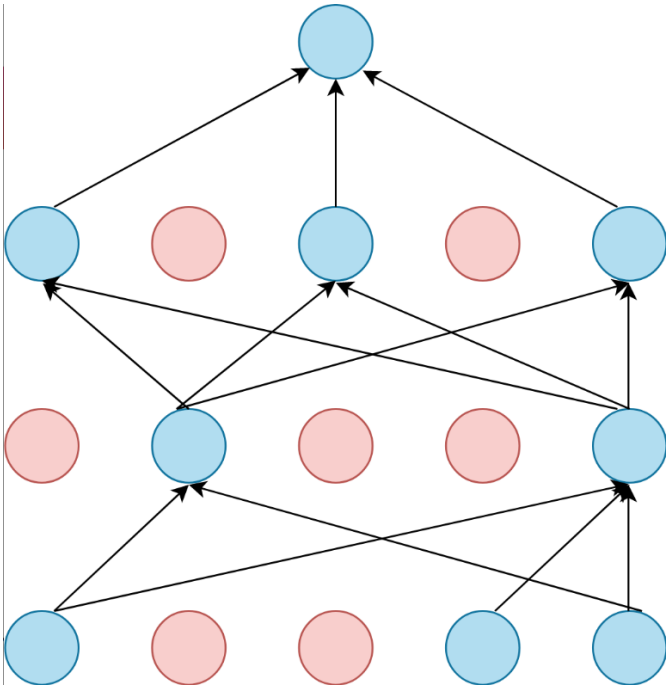
8 pav. Testavimo rezultatai naudojant architektūrą nr. 10

Problemų šiam modeliui kėlė auliniai batai, užimantys didesnę nuotraukos dalį lyginant su kitomis batų nuotraukomis, ir sijonai, kuriuos modelis priskyrė viršutiniams drabužiams. Pateikus modeliui modifikuotą aulinių batų nuotrauką – jos veidrodinį atspindį, modelis priskyrė šią nuotrauką klasei „Shoes“ (2 numeris) 100%.

Tad palyginus penktą, ketvirtą ir dešimtą architektūras, geriausi rezultatai išgauti naudojant dešimtą architektūrą. Atliekant sekančią laboratorinio darbo dalį – siūlant savo architektūrą šiam uždaviniui reikėtų atsižvelgti, kaip būtų galima išspręsti problemas su sijonų priskyrimu pirmai klasei – apatinės dalies drabužiams.

1.3. Kryžminis validavimas ir modelio architektūros kūrimas

Trečias pirmos užduoties punktas reikalauja pasiūlyti savo architektūrą vaizdų klasifikavimo užduoties sprendimui. Nuspręsta pasiremti trečia architektūra, kadangi ankstesnėse dalyse jos rezultatai buvo geriausi. Dešimto varianto architektūra modifikuojama pridedant kelis Dropout sluoksnius, kurie skirti persimokymo prevencijai. Dropout yra skirtas reguliarizacijai ir atsitiktinių įvesties vienetų reikšmes nustato kaip 0. Tokiu būdu sumažėja neuronų tarpusavio priklausomybė ir sumažėja persimokymo priklausomybė.



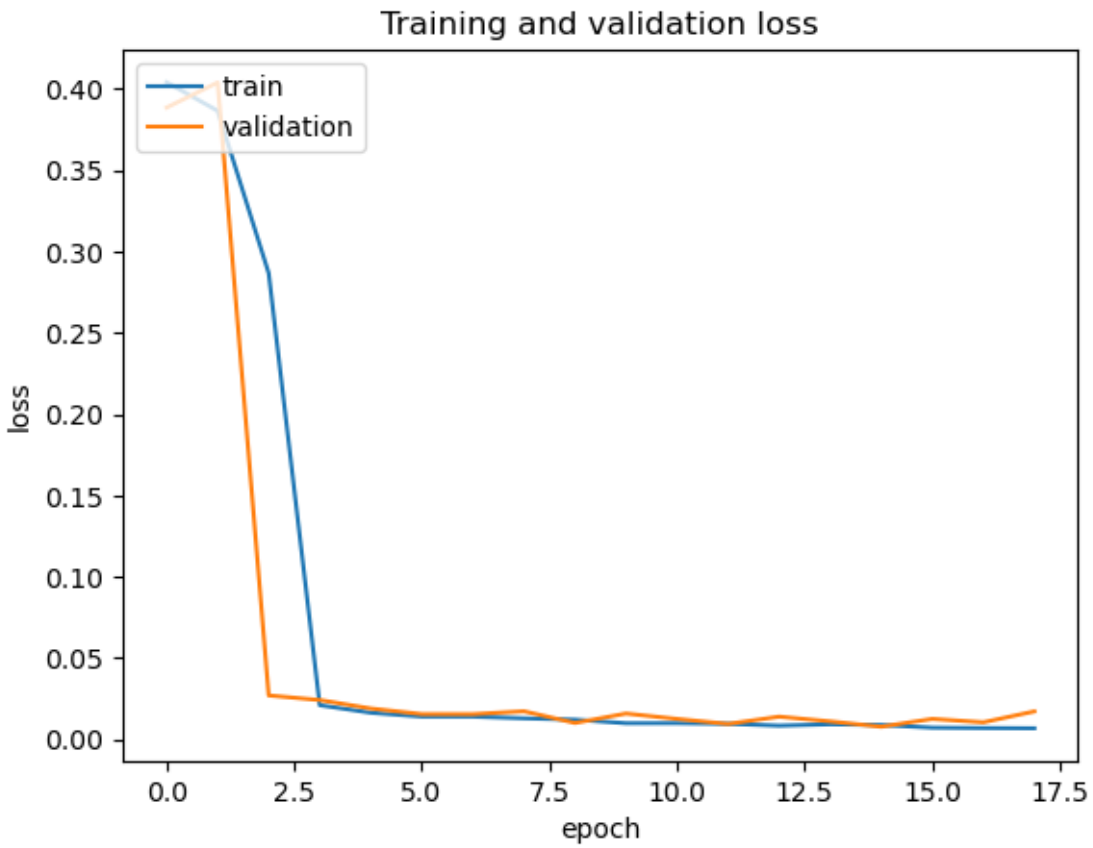
11 pav. ryšiai tarp neuronų sluoksniuose panaudojus Dropout. (www.educative.io)

```
model3 = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(strides=(2,2)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.AveragePooling2D(strides=(2,2)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(128, 3, activation='relu'),
    tf.keras.layers.AveragePooling2D(strides=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(3, activation='relu')
])
```

Naudojant kryžminį validavimą su tuo pačiu papildytu duomenų rinkiniu, jis sumažintas nuo 105 tūkst. įrašų iki 87 tūkst.

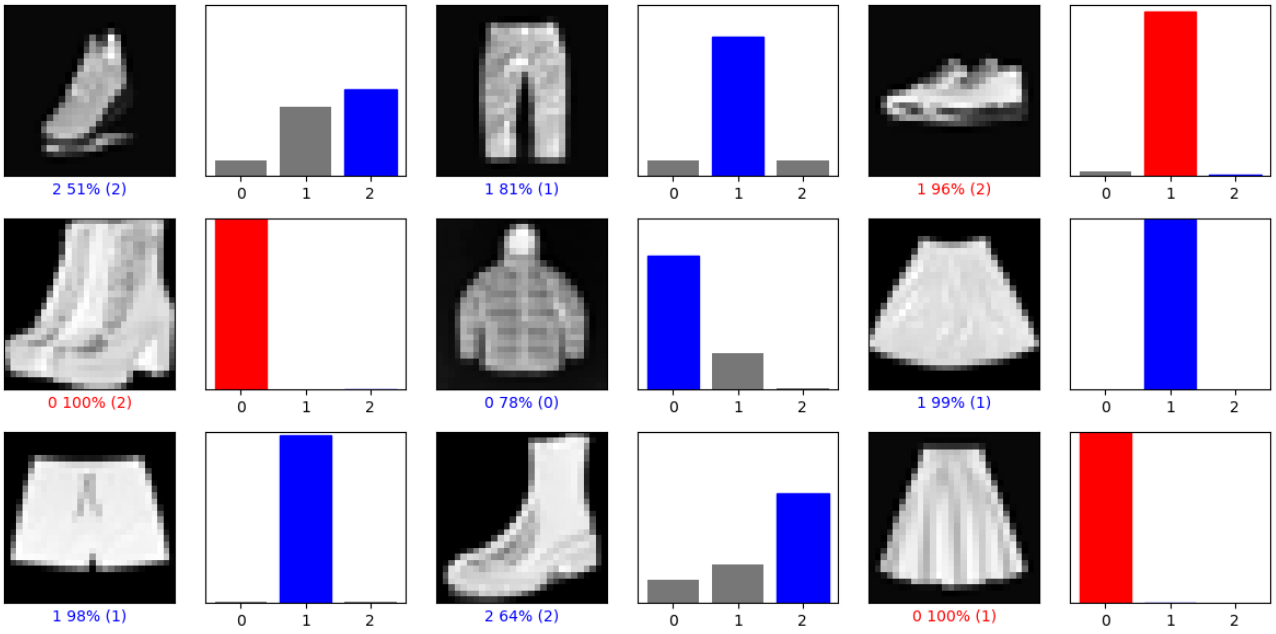
Pasiūlyta architektūra su sumažintu duomenų rinkiniu tenkinantį apmokymo lygį pasiekia ties 3-4 epocha. Po to išlieka pastovūs su nedideliais svyravimais.

Panaudojus atskirą testavimo rinkinį, kurį sudarė 9 nuotraukos, teisingai priskirtos 6 klasės. Sijonas atpažintas kaip apatinės kūno dalies drabužis, jei jis nebuvo midi ilgio ir ilgesnis. Problemų kilo su batų poromis. Pavieniai batai buvo kategorizuojami teisingai, tačiau batų poros priskiriamos klaidingoms klasėms priklausomai nuo užimamos erdvės nuotraukoje.



12 pav. Pasiūlytos architektūros apmokymo procesas

13 pav. Teisingų ir klaidingų priskyrimų pasiskirstymas pagal klases



14 pav. Teisingų ir klaidingų priskyrimų pasiskirstymas pagal klases

2. Antra dalis

Antroje laboratorinio darbo dalyje sprendžiamą vaizdų klasifikavimo problema: skirtingų rašymo priemonių atpažinimas. Duomenų rinkinyje, kuris sudarytas iš vaizdų paimtų iš internetinių parduotuvių ir nufotografuotų mobiliuoju telefonu, yra 3 skirtingos klasės: pieštukai, rašikliai ir flomasteriai. Pradiniame rinkinyje yra 100 nuotraukų.

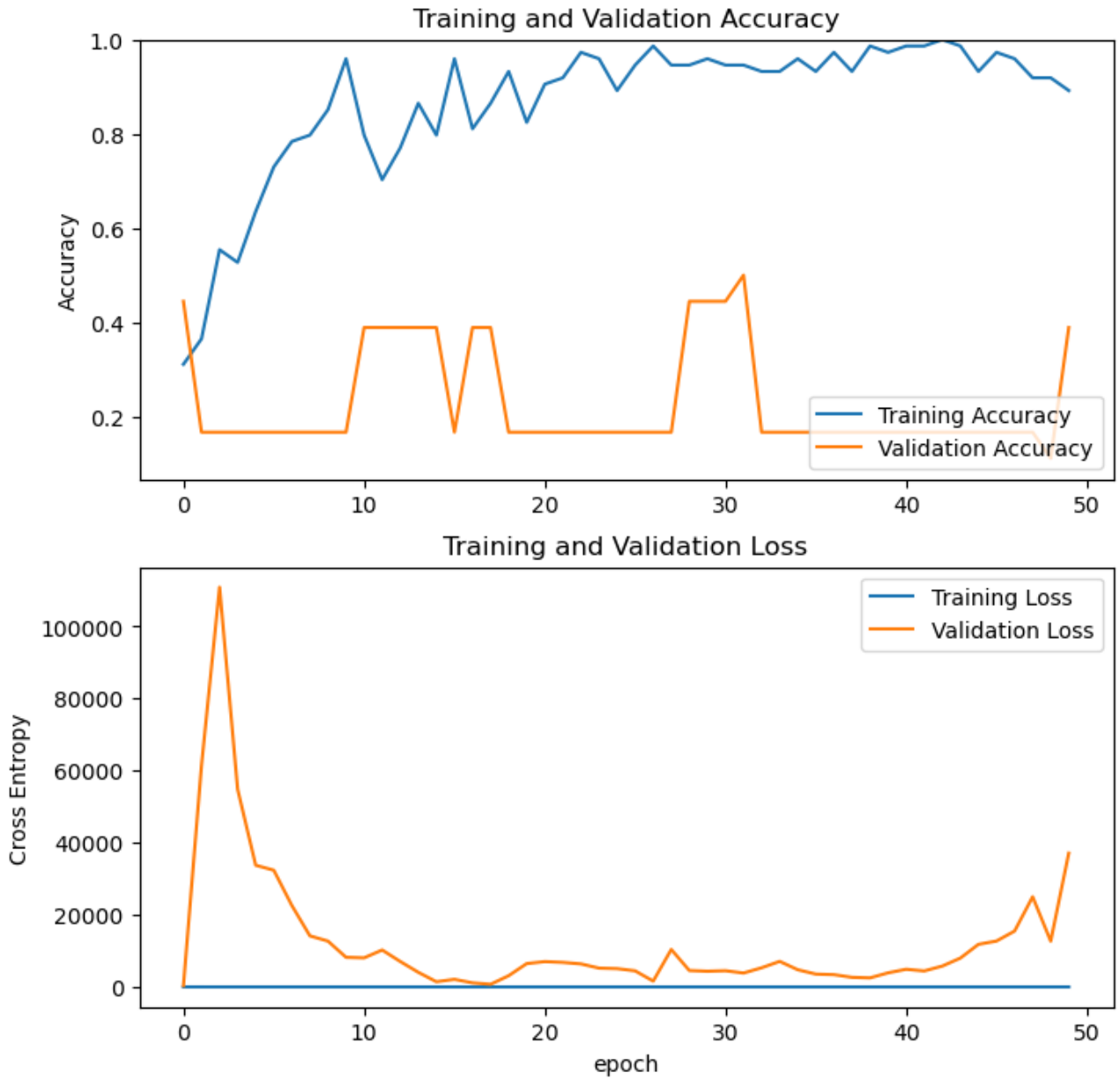
15 pav. Pavyzdžiai iš duomenų rinkinio

Užduoties sprendimui pasirinkta naudoti ResNet50 modelį – jo architektūrą ir svorius. Ši architektūra pasirinkta palyginus kelių skirtingų architektūrų informaciją, pateikta Keras dokumentacijoje. Plačiau pasidomėjus, sužinota, kad ResNet50 sėkmingai taikomas vaizdų atpažinimo užduotyse, dėl pasiekiamo tikslumo, kuris yra didesnis nei kitų mažesnių ar ne tokių gilių architektūrų [4].

2.1. Be svorių ir be augmentacijos

Naudojant tik ResNet50 architektūrą su 100 nuotraukų apmokymo procesas nesėkmingas. Anksti pastebima, kad apmokymo ir validavimo imties tikslumas ir loss funkcijos išsiskiria. Maksimalus pasiektas validacijos imties tikslumas 39%

```
base_model = ResNet50(weights=None, include_top=False, input_shape=(img_height,
img_width, 3))
model = Sequential([data_augmentation])
model.add(base_model)
model.add(GlobalAveragePooling2D())
model.add(Dense(num_classes, activation='softmax'))
model.compile(optimizer='adam',                 loss='categorical_crossentropy',
metrics=['accuracy'])
history = model.fit(train_ds, epochs=epochs, validation_data=val_ds,
callbacks=[early_stopping_callback])
```



16 pav. Apmokymo procesas, kai nenaudojami svoriai iš jau apmokyto modelio ir nenaudojama duomenų augmentacija

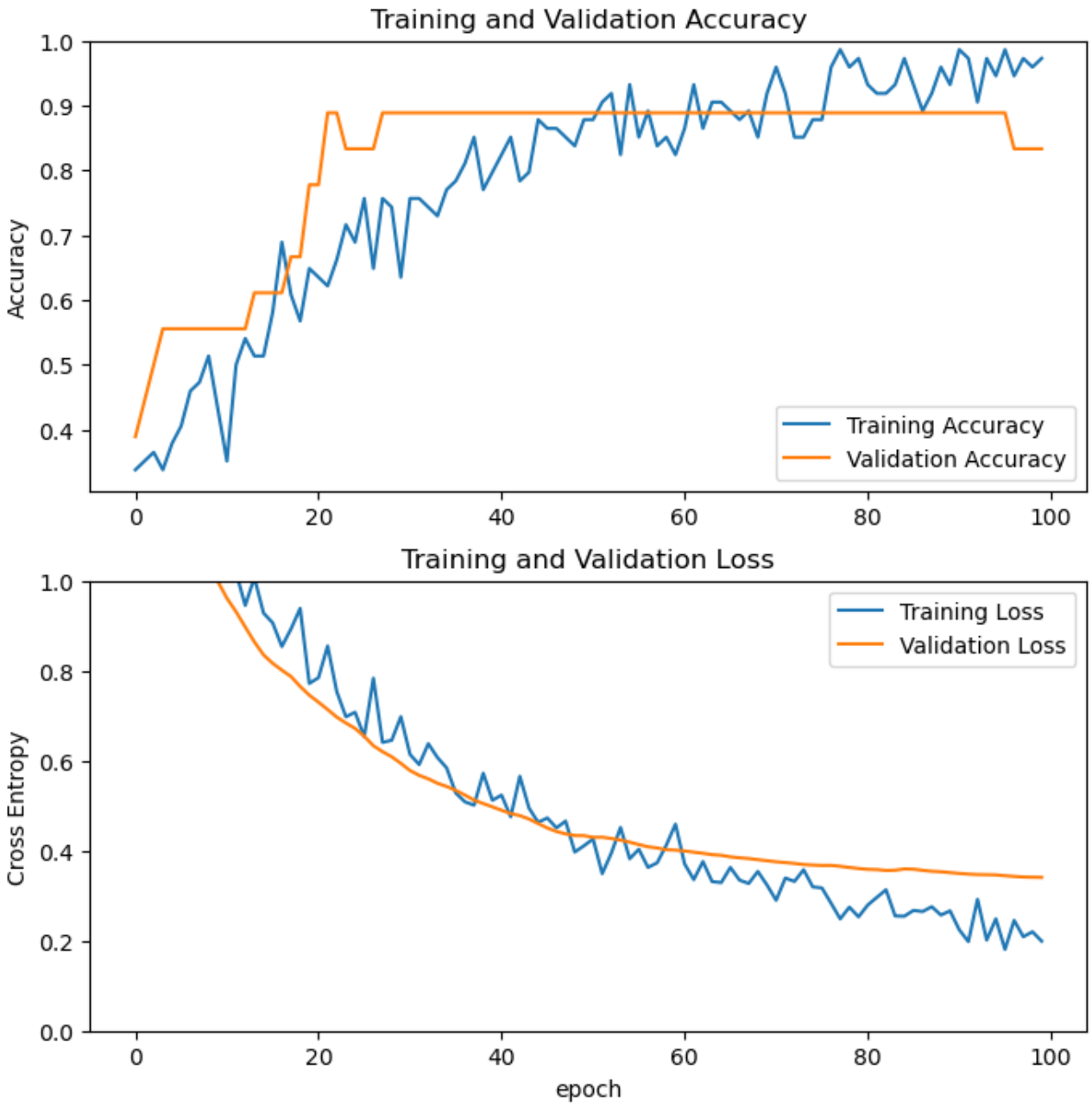
2.2. Su svoriais be augmentacijos

Apmokyto modelio svorių panaudojimas turėtų palengvinti apmokymo procesą ir padėti išgauti didesnį tikslumą. Naudojant Tensorflow ir Keras tai galima padaryti taip

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Flatten,
AveragePooling2D
base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=(img_height, img_width, 3))
base_model.trainable = False
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
```

```
image_batch, label_batch = next(iter(train_ds))
feature_batch = base_model(image_batch)
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
prediction_layer = tf.keras.layers.Dense(num_classes, activation='softmax')
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
preprocess_input = tf.keras.applications.resnet.preprocess_input
inputs = tf.keras.Input(shape=(img_height, img_width, 3))
#x = data_augmentation(inputs) atkomentuojama, kai taikoma duomenu
augmentacija
x = preprocess_input(inputs)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning
_rate), loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_ds, epochs=initial_epochs, validation_data=val_ds,
callbacks=[early_stopping_callback])
```

Pateiktame kode aprašomas ResNet50 modelio pritaikymas konkrečios užduoties sprendimui naudojant pasirinktą rinkinį, kai nekeičiant pradinio modelio svorių pridedami papildomi sluoksniai apdorojantys įvestis ir rezultatus, tarpinius paskaičiavimus.



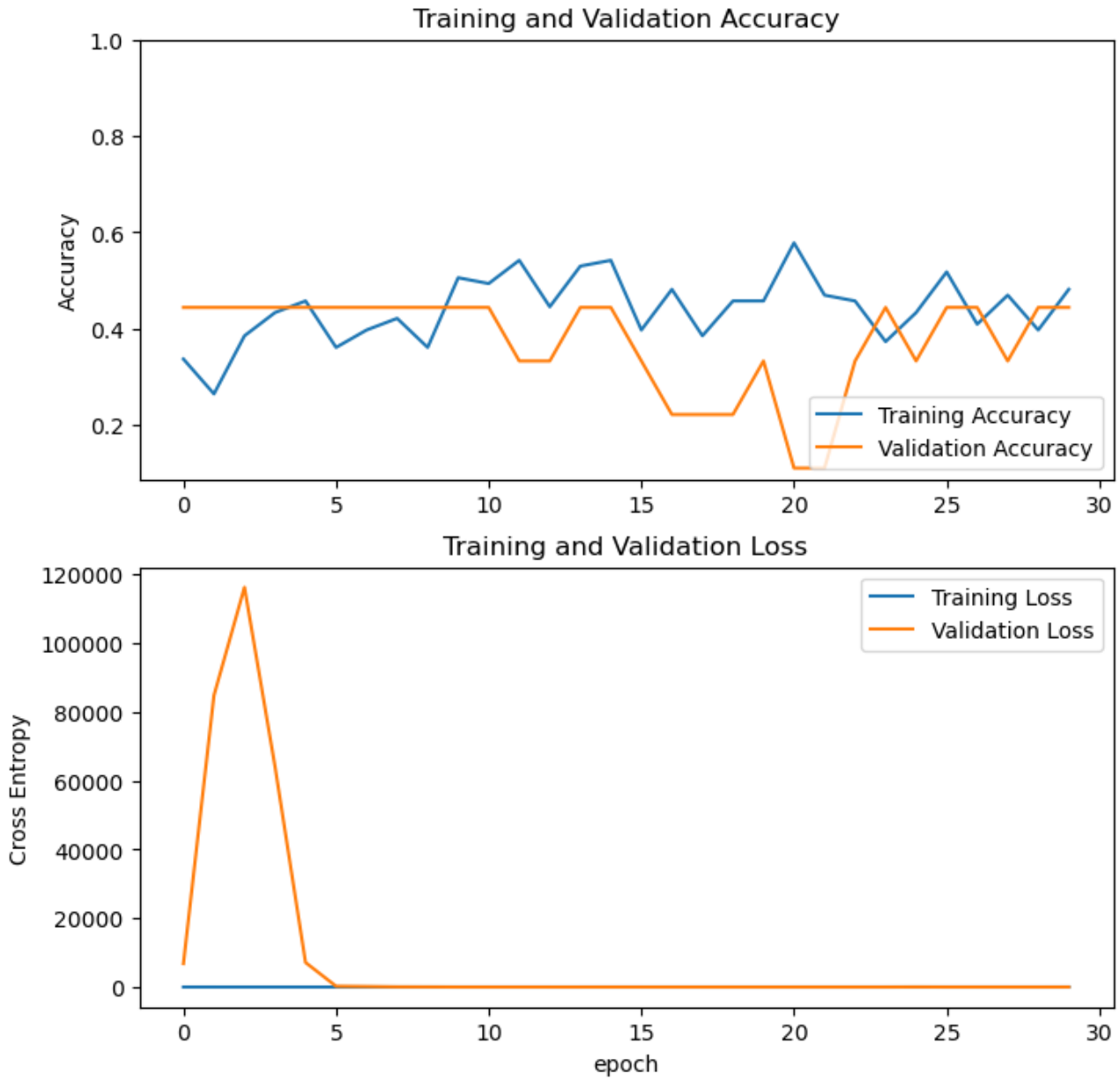
17 pav. Apmokymo procesas naudojant jau apmokyto modelio svorius ir nenaudojant duomenų augmentacijos

Naudojant apmokyto ResNet50 svorius, pakeičiant paskutinius sluoksnius, išgaunamas net 90% validavimo imties tikslumas. Tai daug daugiau nei gauta nenaudojant apmokyto modelio svorių. Tai pat pastebimas loss funkcijos mažėjimas, nėra didelės persimokymo rizikos.

2.3. Be svorių naudojant augmentaciją

Augementacija atliekama naudojant Keros karkaso siūlomus sluoksnius leidžiančius atlikti nuotraukų atspindėjimą, pasukimą.

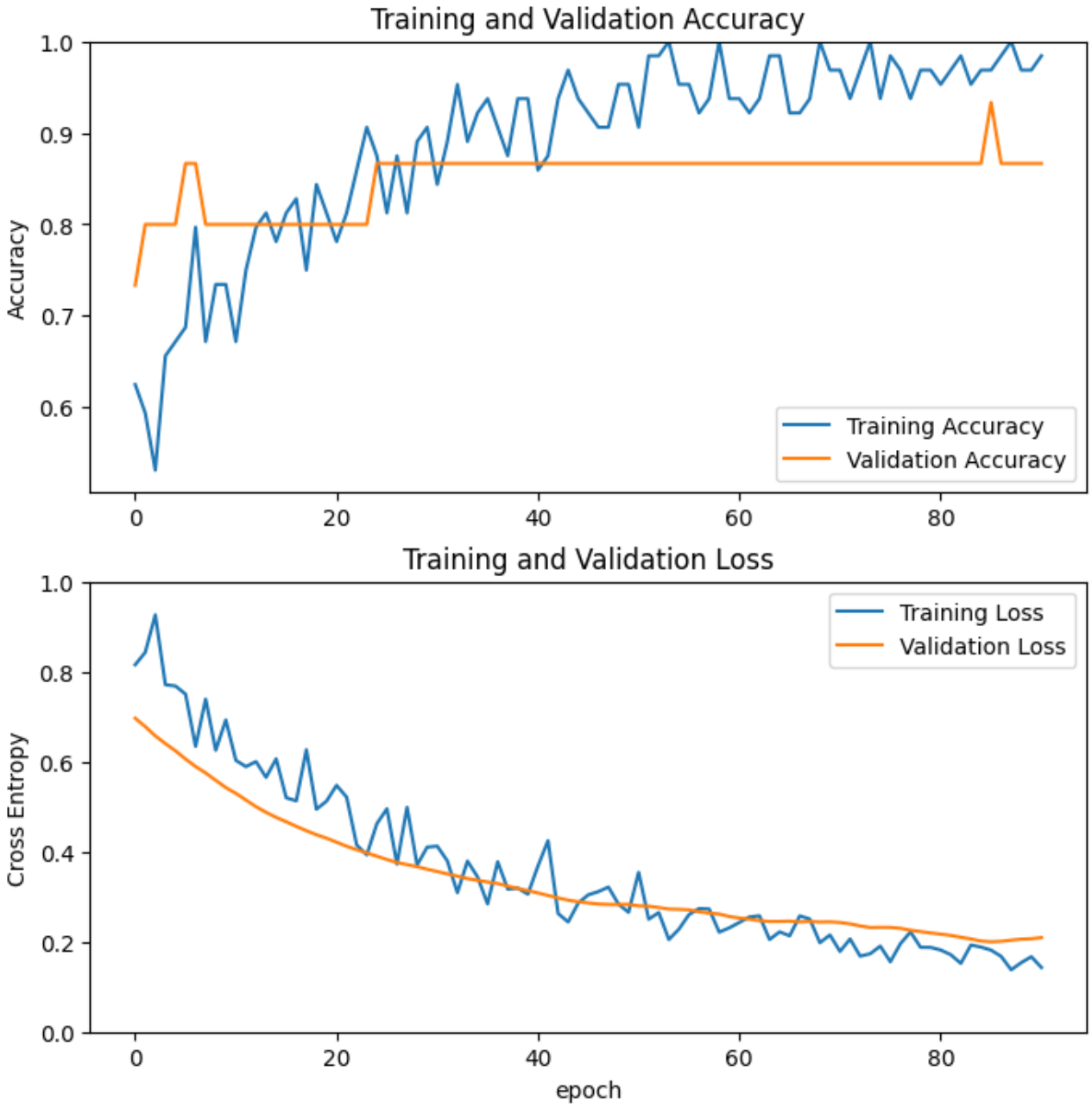
```
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.2),
])
```



18 pav. Apmokymo procesas naudojant jau apmokyto modelio svorius ir nenaudojant duomenų augmentacijos

Nenaudojant svorius, bet pritaikant duomenų augmentaciją skirtumas tarp apmokymo ir validavimo imčių tikslumo apmokymo metu yra mažesnis. Skirtumas tarp loss funkcijų išlieka didelis, tačiau greičiau ima mažėti ir pastebimai nebekyla jau po 5 epochos. Pasiekiamas 45% tikslumas.

2.4. Su svoriais naudojant augmentaciją



19 pav. Apmokymo procesas naudojant jau apmokyto modelio svorius ir duomenų augmentaciją

Naudojant apmokyto modelio svorius ir papildant duomenų rinkinį modifikuotomis nuotraukomis išgaunamas didesnis tikslumas nei nenaudojant svorių ar duomenų augmentacijos. Pastebima, kad pradinės apmokymo ir validavimo imčių loss funkcijos reikšmės yra mažesnės nei tuo atveju, kai duomenų augmentacija nenaudojama. Funkcijos reikšmės taip pat tolygiai mažėja ir išlieka panašios, o 16 pav. Pavaizduotame grafike matomas nedidelis išsiskyrimas tarp apmokymo ir validavimo imčių.

Pasiekiamas 95% tikslumas.

2 lentelė. Tikslumų palyginimas naudojant skirtingus metodus

	Be augmentacijos	**Su augmentacija**
Be svorių	39%	45%
Su svoriais	90%	95%

Palyginus 2 lentelėje pateiktus skaičius matoma, kad tikslumas buvo apie 5% didesnis naudojant duomenų augmentaciją nei jos nenaudojant ir daugiau nei 2 kartus didesnis naudojant jau apmokyto tinklo svorius.

Išvados

1. Laboratorinio darbo metu sėkmingai realizuotos skirtingos neuroninių tinklų architektūros, apmokymo naudojant jau anksčiau apmokyto modelio svorius, duomenų augmentacijos ir kryžminės patikros metodai.
2. Palyginus skirtingus modelius ir jų apmokymo procesus antroje dalyje, nustatyta, kad geriausi rezultatai pasiekiami naudojant svorių perkėlimą ir duomenų augmentaciją.
3. Didesnį tikslumą galima pasiekti naudojant didesnį duomenų rinkinį, atliekant išsamesnį parametrų derinimą.

Literatūros sąrašas

1. Tensorflow mokomoji medžiaga: https://www.tensorflow.org/tutorials/keras/classification
2. Keras dokumentacija: https://keras.io/api/
3. Apie reguliarizaciją: https://www.educative.io/answers/keras-dropout-layer-implement-regularization
4. K., Balavani. (2023). An Optimized Plant Disease Classification System Based on Resnet-50 Architecture and Transfer Learning. 1-5. doi: 10.1109/INCET57972.2023.10170368