

Przetwarzanie obrazów: Obiekty geograficzne

Projekt II

in python with TensorFlow

Mateusz Sołoducha
grupa 3

Wstęp

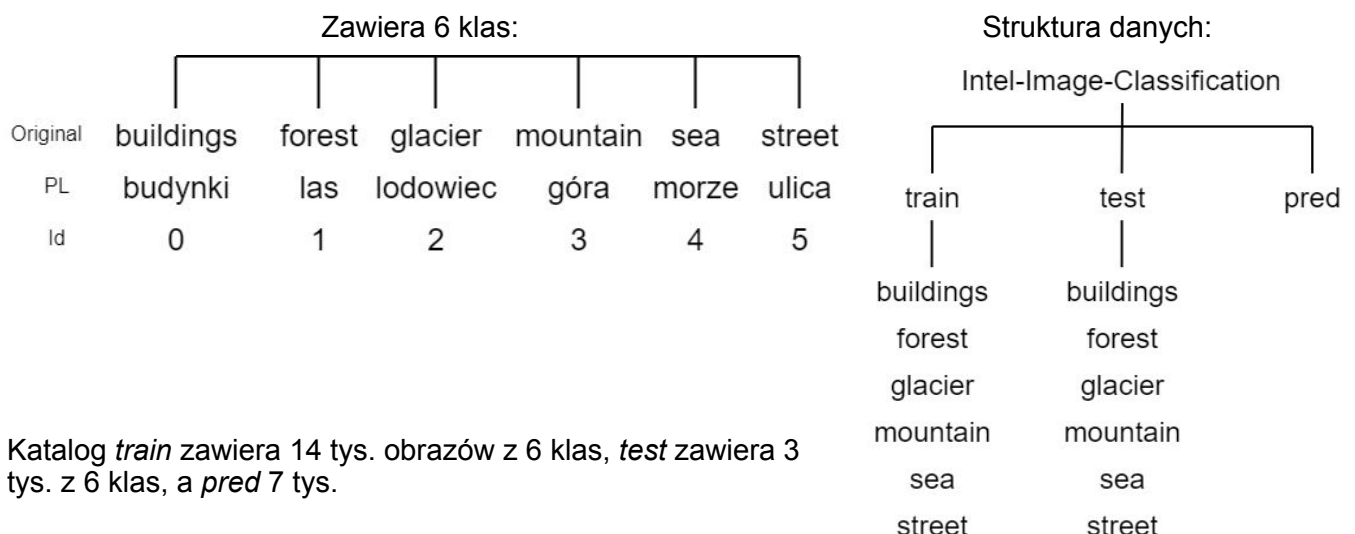
GitHub: <https://github.com/Maaateusz/classifiers---image-processing>

W projekcie korzystam z wielu paczek, najważniejsze z nich to: graphviz, tensorflow, keras, seaborn, sklearn, matplotlib, numpy.

W celu zwiększenia wydajności trenowania sieci neuronowych dodatkowo korzystam z mocy udostępnionej przez moją kartę graficzną Nvidia (jak używać GPU - <https://www.tensorflow.org/install/gpu>).

Baza Danych

Baza danych na której pracowałem nazywa się **Intel Image Classification**, pochodzi z <https://www.kaggle.com/puneet6060/intel-image-classification>. Jest to zbiór około 25 tys. obrazów reprezentujących kilka obiektów geograficznych.



Przykładowy obraz z każdej klasy



Żadne zdjęcia nie zawierają błędów, wszystkie są w rozdzielczości 150x150 pikseli z 3 warstwami koloru, dzięki czemu baza danych nie potrzebuje wstępnej obróbki i jest od razu gotowa do używania.

Klasyfikatory

Poniżej przedstawię wyniki klasyfikacji obrazów na drzewie decyzyjnym, naiwnym klasyfikatorze Bayesowskim i klasyfikatorze K - najbliższych sąsiadów. Ze względu na użytą metodę pobierania i przetwarzania danych, używam tylko zbioru treningowego i dziele go, na zbiór testowy wielkości 20% całego zbioru i zbiór treningowy 80% (użyta i dostosowana do moich potrzeb funkcja pochodzi z <https://medium.com/swlh/image-classification-with-k-nearest-neighbours-51b3a289280>). Dane treningowe to obrazy przekonwertowane na macierz i znormalizowane do wartości [0, 1], zamiast [0, 255].

Drzewo decyzyjne [Decision Tree]

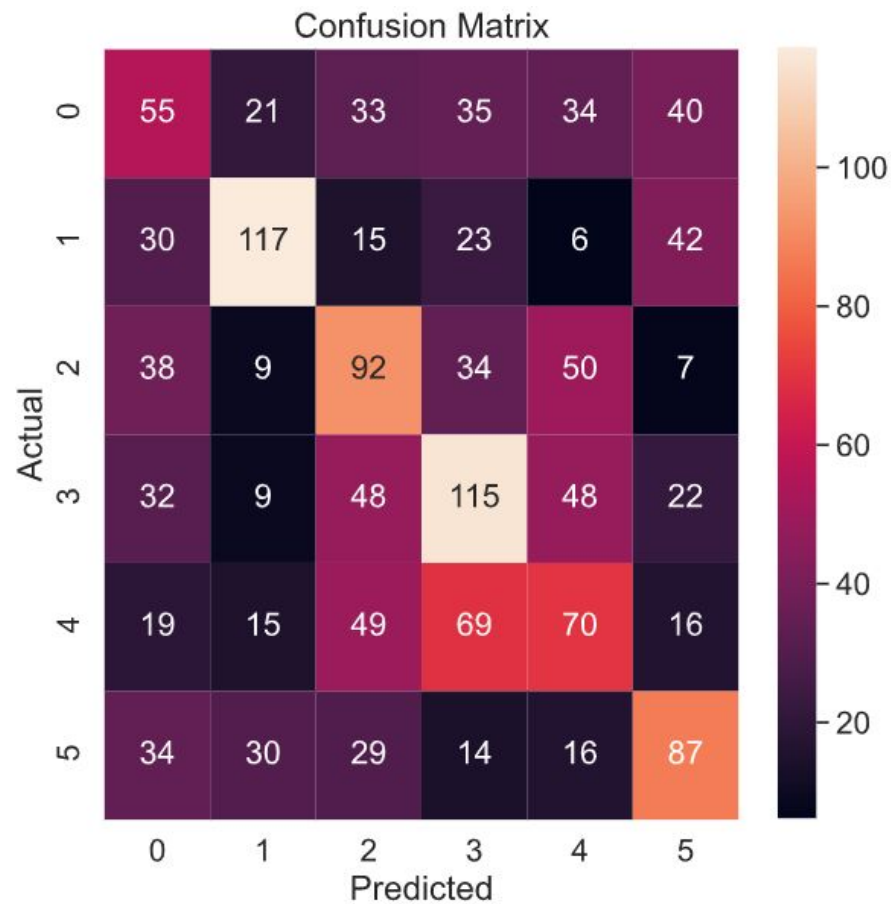
Dokładność: 38.203%

Czas wykonywania: 423.92s

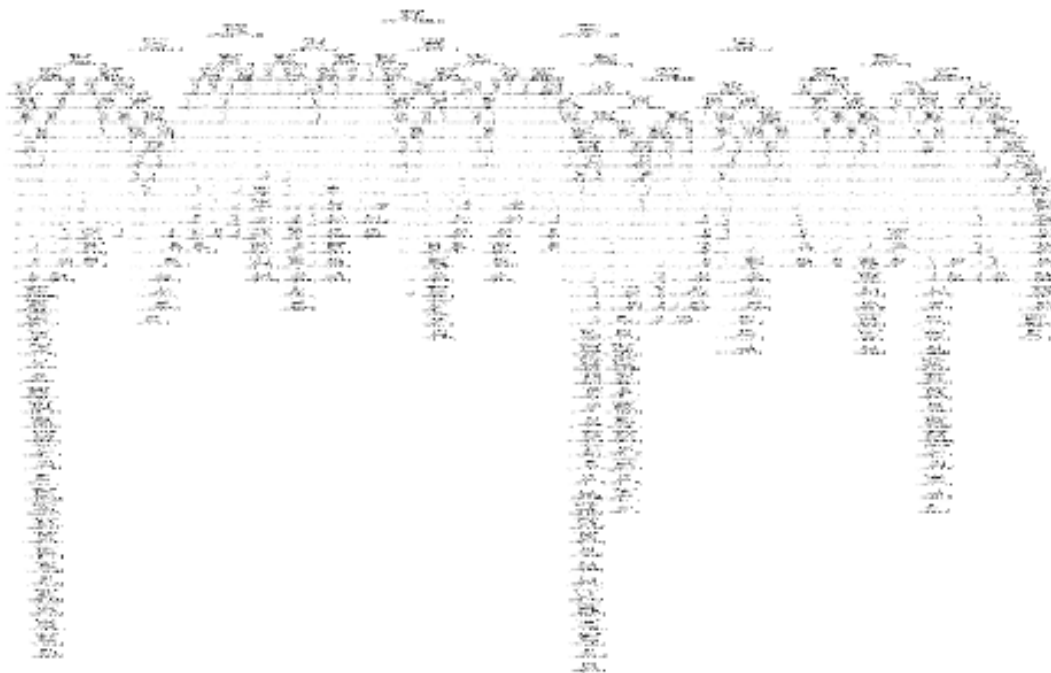
Pierwszy użyty przeze mnie klasyfikator to drzewo decyzyjne. Jak widać poniżej dokładność jest fatalna, klasyfikator w ogóle sobie radzi z poprawną klasyfikacją. Czas trenowania jest przeciętny, zajmuje około 7 minut. Udało mi się wygenerować graficzne przedstawienie tego drzewa decyzyjnego, lecz jest to tylko ciekawostka, ponieważ z uwagi na złożoność całego drzewa jest ono nieczytelne.

Kod:

```
dtc = DecisionTreeClassifier()
fit = dtc.fit(X_train, y_train)
score = dtc.score(X_val, y_val) * 100
```



Wygenerowane drzewo decyzyjne:



Jedna z gałęzi drzewa:

```

samples = 411
value = [0, 1, 2, 402, 6, 0]

  gini = 0.0
  samples = 1
  value = [0, 0, 0, 0, 1, 0]

    gini = 0.0
    samples = 1
    value = [0, 0, 0, 0, 1, 0]

      gini = 0.0
      samples = 1
      value = [0, 0, 0, 0, 1, 0]

        X[20019] <= 0.005
        gini = 0.024
        samples = 407
        value = [0, 1, 2, 402, 2, 0]
          X[6796] <= 0.175
          gini = 0.015
          samples = 398
          value = [0, 1, 2, 395, 0, 0]
            X[14377] <= 0.652
            gini = 0.01
            samples = 395
            value = [0, 1, 1, 393, 0, 0]
              X[19046] <= 0.187
              gini = 0.444
              samples = 3
              value = [0, 0, 1, 2, 0, 0]
                gini = 0.0
                samples = 1
                value = [0, 0, 1, 0, 0, 0]

              gini = 0.0
              samples = 13
              value = [0, 0, 0, 13, 0, 0]

```

Naiwny klasyfikator Bayesowski [Naive Bayes]

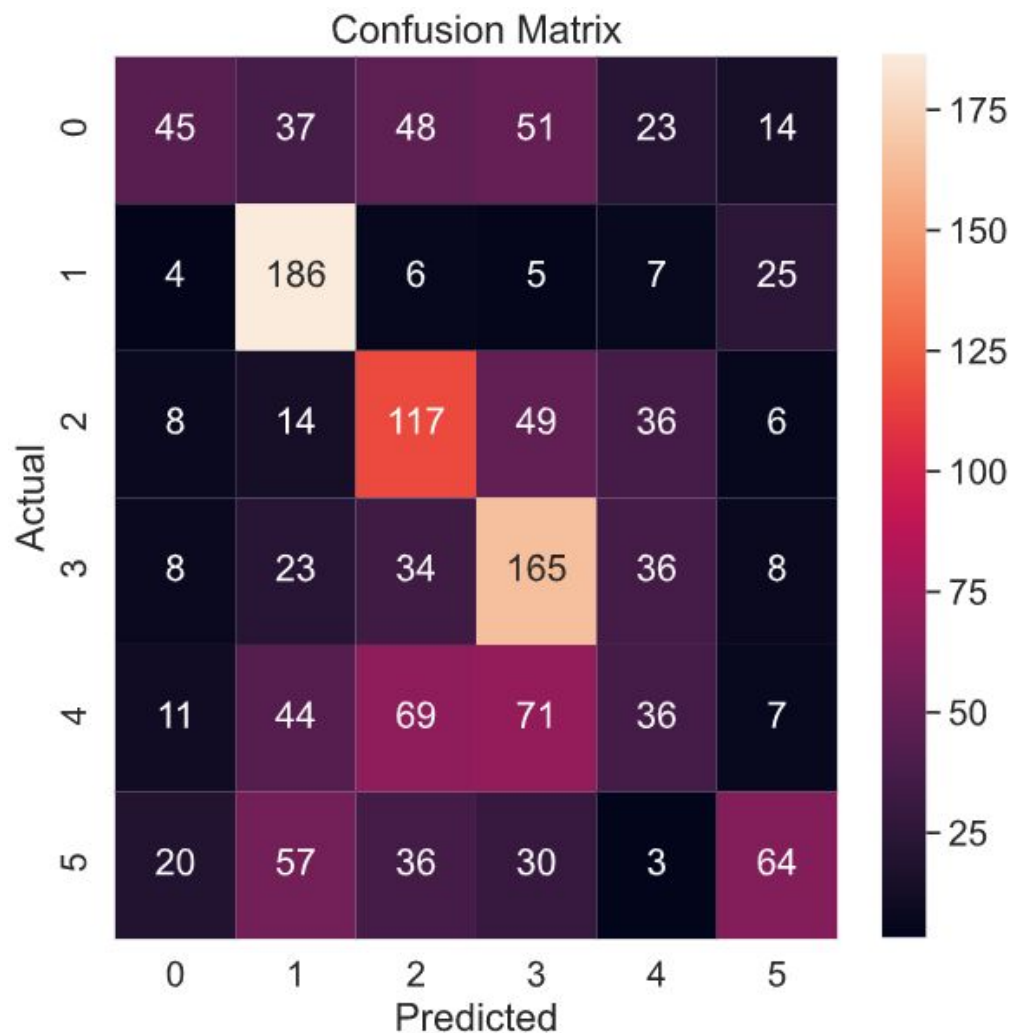
Dokładność: 43.692%

Czas wykonywania: 5.51s

Naiwny klasyfikator Bayesowski radzi sobie tylko trochę lepiej niż drzewa decyzyjne, ale dokładność wciąż jest niemal losowa. Chociaż, jak widać na macierzy błędów, klasyfikator ten trzy klasy rozpoznaje lepiej niż inne. Pod względem czasu trenowania jest to najszybszy klasyfikator.

Kod:

```
gnb = GaussianNB()
test_classes_predicted = gnb.fit(X_train, y_train).predict(X_val)
score = metrics.accuracy_score(y_val, test_classes_predicted) * 100
```



Klasyfikator K najbliższych sąsiadów [k-nearest neighbors, k-NN]

Klasyfikator został wykonany dla $k = [1, 3, 5, 10, 25, 47, 91]$.

Klasyfikator K najbliższych sąsiadów radzi sobie równie źle jak drzewa decyzyjne. Na macierzy błędów widać, że jednej klasy nie był w stanie rozpoznać. Czas trenowania jest relatywnie długi.

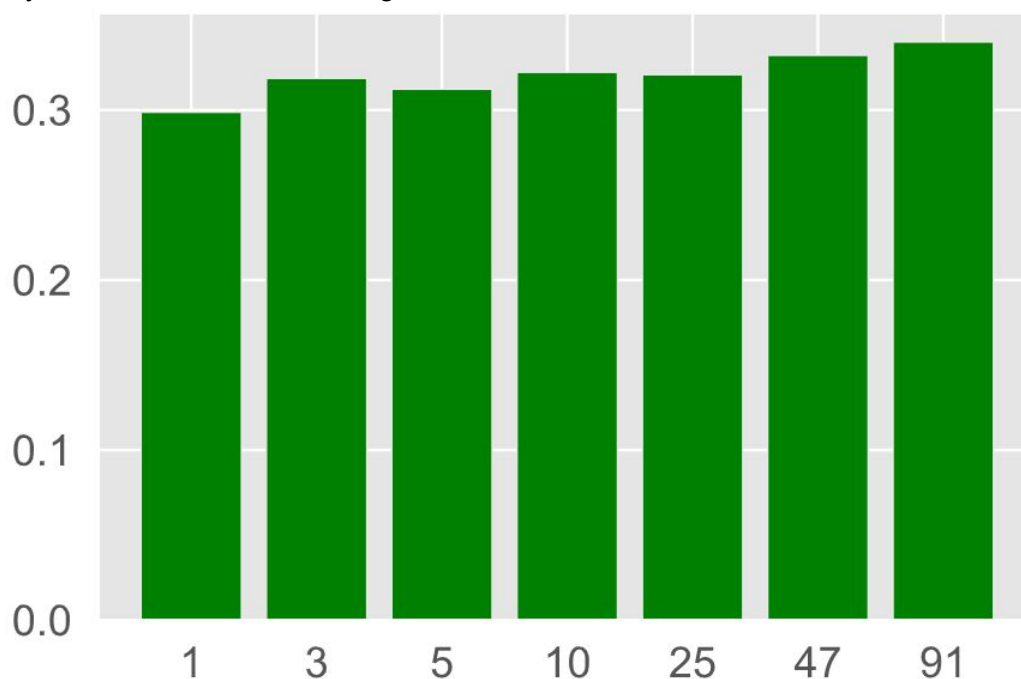
Kod:

```
knn_list = [1, 3, 5, 10, 25, 47, 91]
for k in range(len(knn_list)):
    neigh = KNeighborsClassifier(n_neighbors=knn_list[k])
    neigh.fit(X_train, y_train)
    score = neigh.score(X_val, y_val)
```

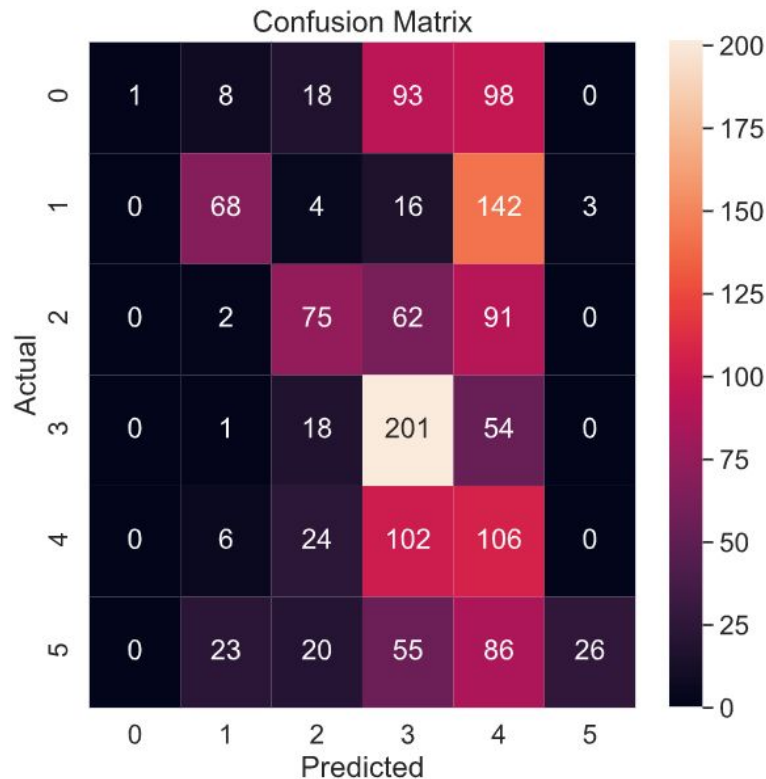
Tabela dokładności i czasu dla danego k:

k	1	3	5	10	25	47	91
Dokładność	29.86%	31.86%	31.21%	32.21%	32.07%	33.21%	33.99%
Czas	698.26s	673.62s	676.20s	675.48s	656.41s	654.12s	654.6 s

Wykres dokładności dla danego k:



Macierz błędów dla k = 91:



Sieci Neuronowe

W sieciach neuronowych do trenowania używam całego zbioru treningowego, do walidacji 20/30% zbioru testowego, a do testowania pozostałą część zbioru testowego. Dane wczytuje za pomocą funkcji z paczki `tf.keras.preprocessing.image_dataset_from_directory`.

Testowałem sieci z wieloma parametrami i ustawieniami. Najlepszą metodą optymalizacji kompilacji okazał się algorytm Adam. Używałem sztucznego generowania nowych danych (data augmentation), niestety przy tej bazie danych wyniki były tylko gorsze.

Kod wczytywania i podziału danych:

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    pathlib.Path.joinpath(data_dir, "train/"),
    image_size=(img_height, img_width),
    batch_size=batch_size)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    pathlib.Path.joinpath(data_dir, "test/"),
    validation_split=validation_split,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    pathlib.Path.joinpath(data_dir, "test/"),
```

```
validation_split=validation_split,
subset="training",
seed=123,
image_size=(img_height, img_width),
batch_size=batch_size)
```

Podział danych:

20%	30%
<u>__Training dataset__</u> Found 14034 files belonging to 6 classes.	<u>__Training dataset__</u> Found 14034 files belonging to 6 classes.
<u>__Validation dataset__</u> Found 3000 files belonging to 6 classes. Using 600 files for validation.	<u>__Validation dataset__</u> Found 3000 files belonging to 6 classes. Using 900 files for validation.
<u>__Test dataset__</u> Found 3000 files belonging to 6 classes. Using 2400 files for training.	<u>__Test dataset__</u> Found 3000 files belonging to 6 classes. Using 2100 files for training.

Kod kompilowania sieci:

```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'])
```

Kod trenowania sieci:

```
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs)
```

Kod testowania poprawności sieci:

```
test_scores = model.evaluate(test_ds)
```

Sieć Neuronowa [Neural Network, NN]

Podział testowy: 20%
Liczba epok: 30
Wielkość partii: 32
Dokładność: 58.375%
Czas wykonywania: 414.00s

Klasyczna sieć neuronowa korzystająca tylko z warstw gęstych. Dokładność w okolicach 60% to wciąż mało, ale sieć już rozróżnia poszczególne klasy.

Kod modelu:

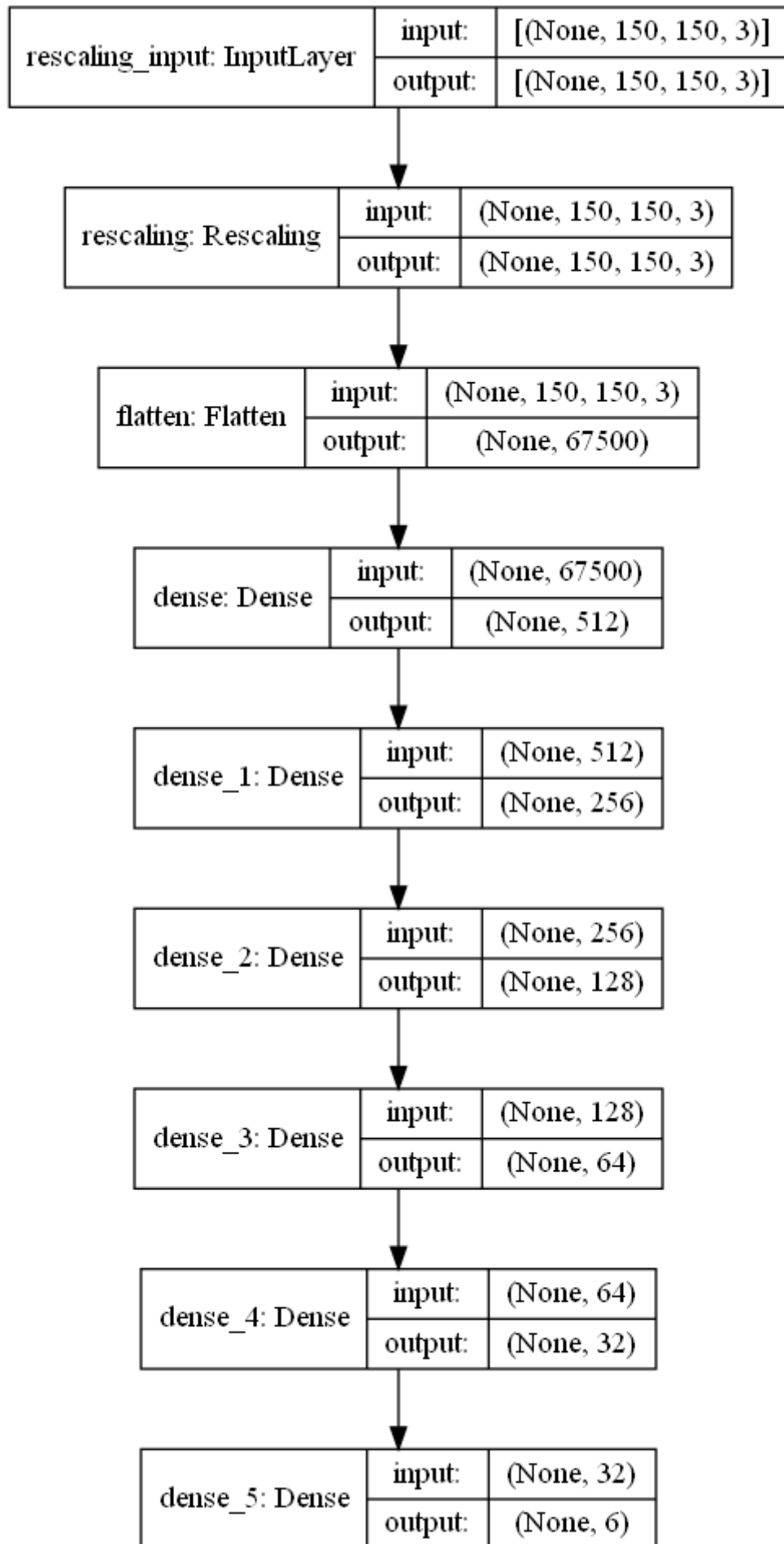
```
model = Sequential([
    layers.experimental.preprocessing.Rescaling(
        1./255,
        input_shape=(img_height, img_width, 3)
    ),
    keras.layers.Flatten(),
    keras.layers.Dense(512, activation='relu' ),
    keras.layers.Dense(256, activation='relu' ),
    keras.layers.Dense(128, activation='relu' ),
    keras.layers.Dense(64, activation='relu' ),
    keras.layers.Dense(32, activation='relu' ),
    keras.layers.Dense(len(class_names))
])
```

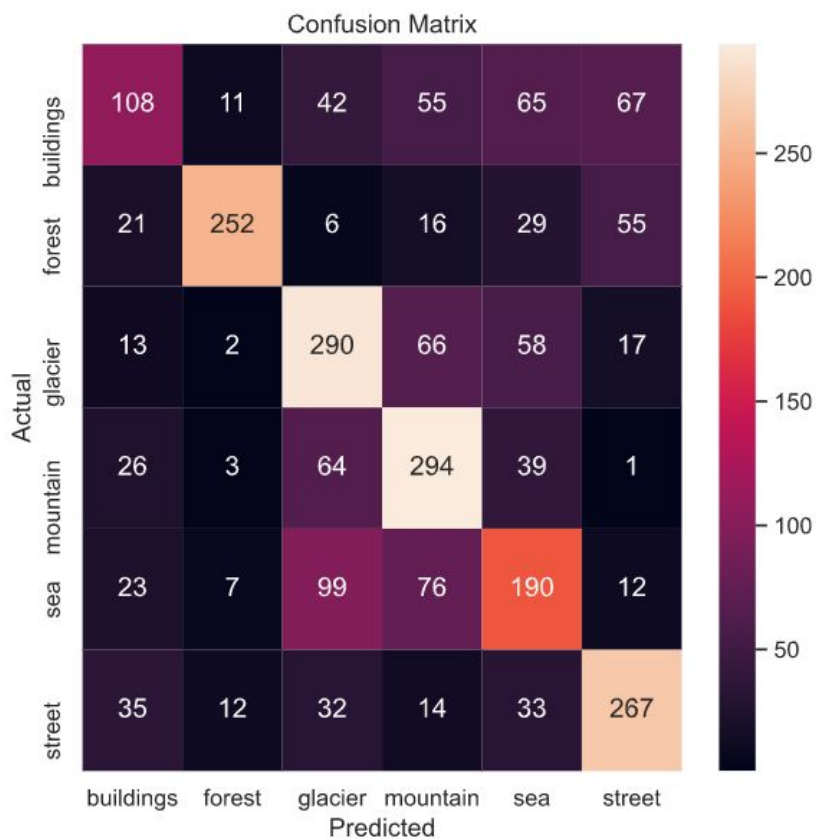
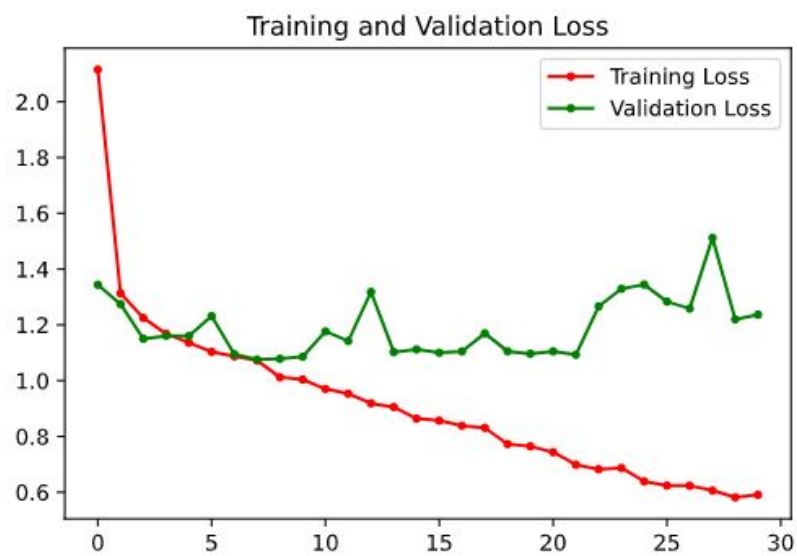
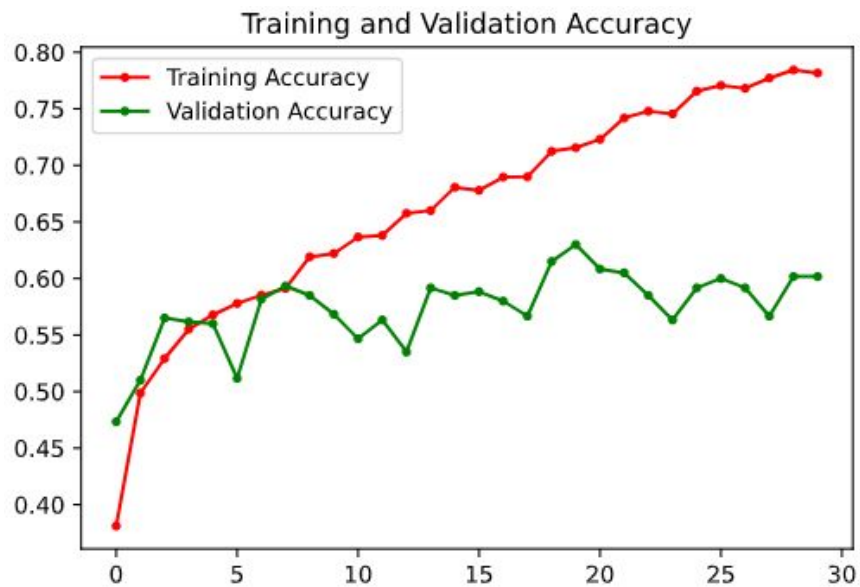
Podsumowanie modelu:

Model: "sequential_1"


Layer (type)	Output Shape	Param #
=====		
rescaling (Rescaling)	(None, 150, 150, 3)	0
flatten (Flatten)	(None, 67500)	0
dense (Dense)	(None, 512)	34560512
dense_1 (Dense)	(None, 256)	131328
dense_2 (Dense)	(None, 128)	32896
dense_3 (Dense)	(None, 64)	8256
dense_4 (Dense)	(None, 32)	2080
dense_5 (Dense)	(None, 6)	198
=====		
Total params: 34,735,270		
Trainable params: 34,735,270		
Non-trainable params: 0		

Podsumowanie modelu:






Przewidywanie klas:

Class: 'street', Acc: 99.80% 




Class: 'street', Acc: 100.00% 



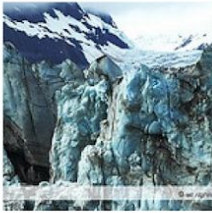
Class: 'street', Acc: 70.87%




Class: 'sea', Acc: 84.12% 



Class: 'glacier', Acc: 78.98%



Class: 'sea', Acc: 40.41% 



Class: 'forest', Acc: 99.14%



Class: 'mountain', Acc: 99.77%



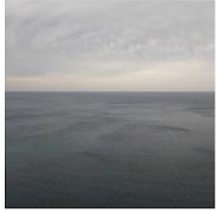
Class: 'glacier', Acc: 89.41%




Class: 'mountain', Acc: 90.43%



Class: 'sea', Acc: 40.05%




Class: 'street', Acc: 98.97% 




Class: 'mountain', Acc: 99.26%



Class: 'mountain', Acc: 39.58% 



Class: 'mountain', Acc: 82.60% 




Class: 'mountain', Acc: 87.17%



Class: 'sea', Acc: 79.85%



Class: 'sea', Acc: 76.06% 




Class: 'mountain', Acc: 44.25%



Class: 'mountain', Acc: 99.72%



Class: 'glacier', Acc: 39.45% 




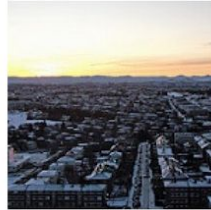
Class: 'street', Acc: 53.21%



Class: 'glacier', Acc: 64.88%



Class: 'mountain', Acc: 79.99% 



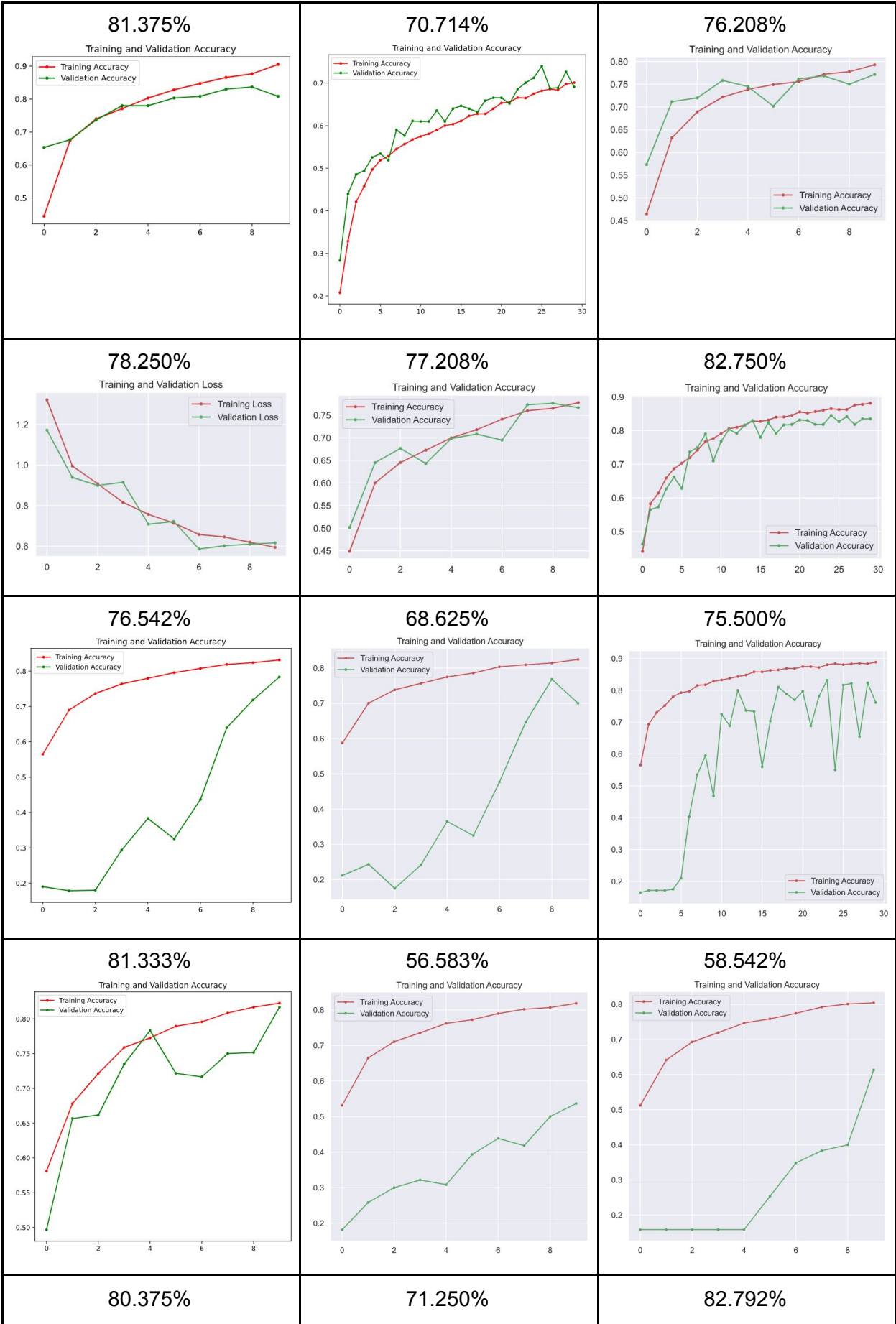
Class: 'sea', Acc: 44.42%

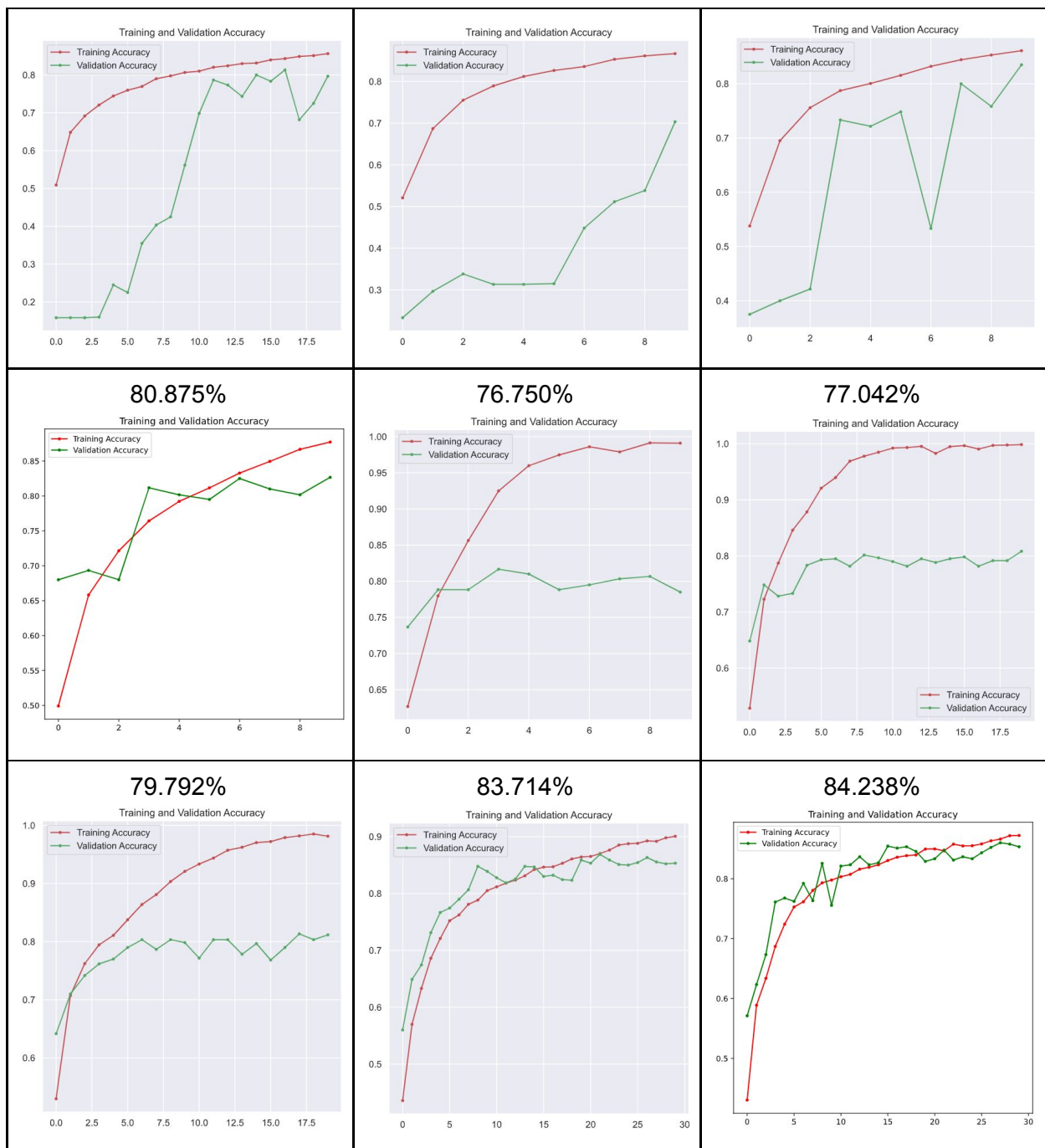


Konwolucyjna Sieć Neuronowa [Convolutional Neural Network, CNN]

Z wielu testów widocznych poniżej konwolucyjna sieć neuronowa uzyskiwała średnio 76% poprawności.

Najlepszy wynik jaki udało mi się osiągnąć to 85%, co można uznać za dobry wynik. Taka sieć prawidłowo klasyfikuje wszystkie klasy. Nadal występują pomyłki, jak można wywnioskować z macierzy błędów, **niepoprawne przewidywania występują przy budynkach i ulicach, oraz górach i lodowcach**, nie jest to zaskakujące, ponieważ tu mylą się także ludzie, ulice zazwyczaj występują między budynkami, a lodowce często są w górach.





Najlepszy model CNN:

Podział testowy: 30%

Liczba epok: 30

Wielkość partii: 128

Dokładność: 85.143%

Czas wykonywania: 460.89s

Kod modelu:

```
model = Sequential([
    layers.experimental.preprocessing.Rescaling(
        1./255,
        input_shape=(img_height, img_width,3)
    ),
    keras.layers.Conv2D(32, 3, activation='relu'),
```

```

keras.layers.MaxPool2D(),
layers.Dropout(0.1),

keras.layers.Conv2D(16, 6, activation='relu'),
keras.layers.MaxPool2D(),
layers.Dropout(0.3),

keras.layers.Conv2D(16, 10, activation='relu'),
keras.layers.MaxPool2D(),
layers.Dropout(0.3),

keras.layers.Flatten(),

keras.layers.Dense(128, activation='relu'),
layers.Dropout(0.5),

keras.layers.Dense(64, activation='relu'),
layers.Dense(len(class_names))
])

```

Podsumowanie modelu:

Model: "sequential_5"

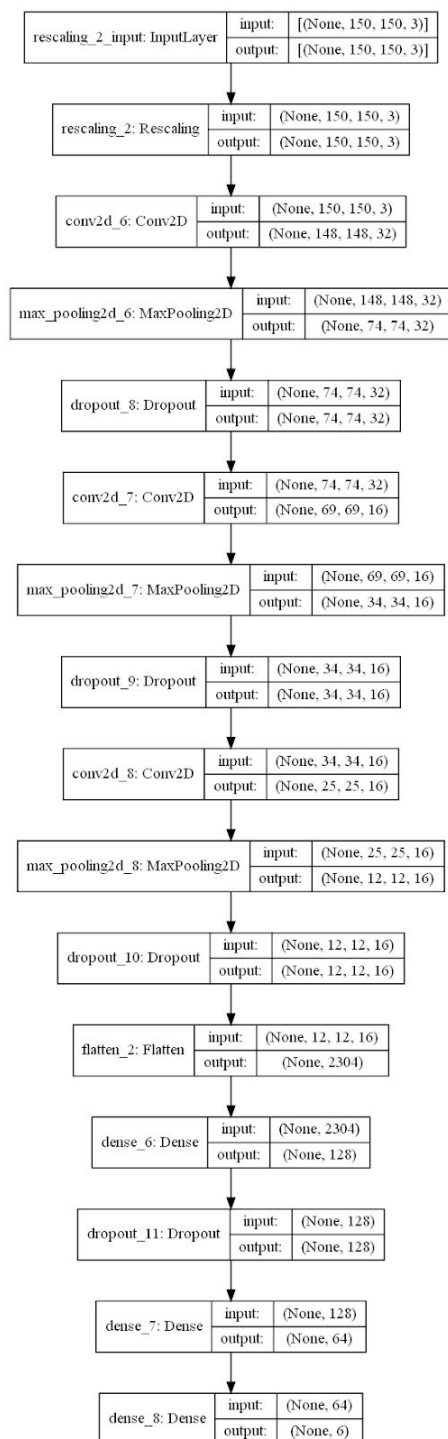
Layer (type)	Output Shape	Param #
=====		
rescaling_2 (Rescaling)	(None, 150, 150, 3)	0
conv2d_6 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_6 (MaxPooling2)	(None, 74, 74, 32)	0
dropout_8 (Dropout)	(None, 74, 74, 32)	0
conv2d_7 (Conv2D)	(None, 69, 69, 16)	18448
max_pooling2d_7 (MaxPooling2)	(None, 34, 34, 16)	0
dropout_9 (Dropout)	(None, 34, 34, 16)	0
conv2d_8 (Conv2D)	(None, 25, 25, 16)	25616
max_pooling2d_8 (MaxPooling2)	(None, 12, 12, 16)	0
dropout_10 (Dropout)	(None, 12, 12, 16)	0
flatten_2 (Flatten)	(None, 2304)	0
dense_6 (Dense)	(None, 128)	295040

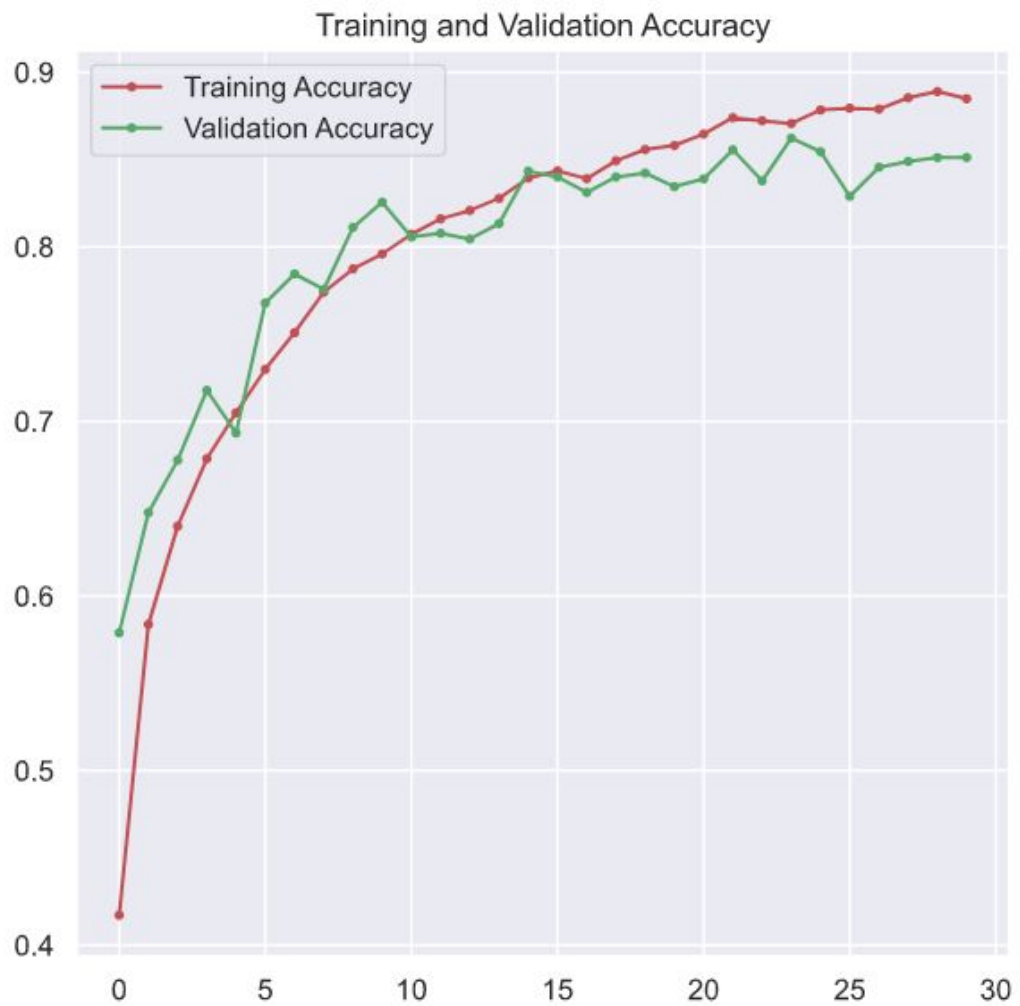
dropout_11 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 64)	8256
dense_8 (Dense)	(None, 6)	390

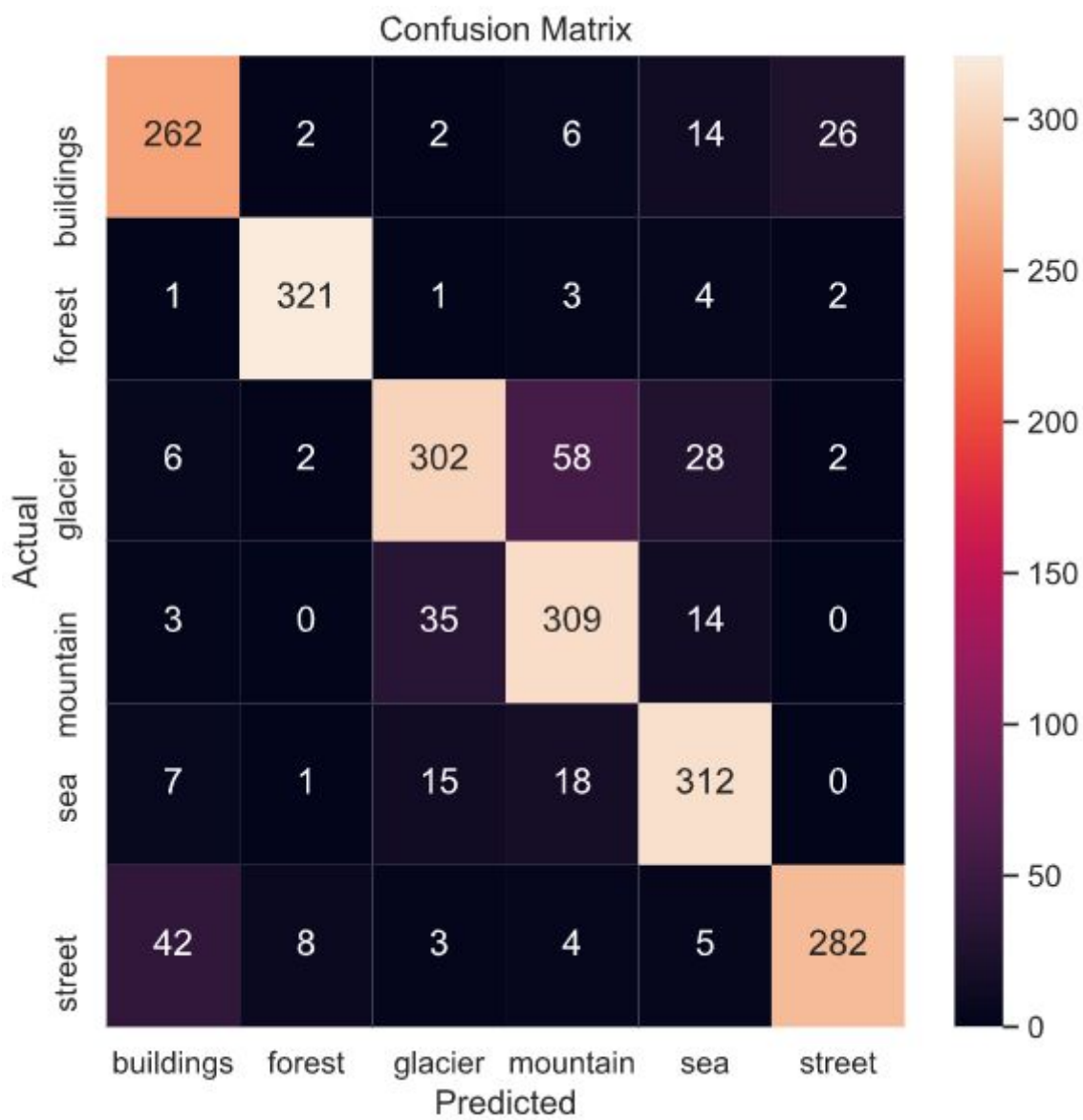
=====

Total params: 348,646
Trainable params: 348,646
Non-trainable params: 0

Podsumowanie modelu:







Przewidywanie klas:

Class: 'buildings', Acc: 58.99%



Class: 'street', Acc: 99.83%



Class: 'street', Acc: 96.03% ↑



Class: 'street', Acc: 80.80% ✓



Class: 'glacier', Acc: 82.03% ↑



Class: 'buildings', Acc: 99.83% ✓



Class: 'forest', Acc: 99.79%



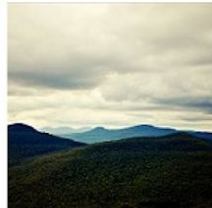
Class: 'mountain', Acc: 98.67%



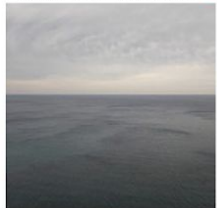
Class: 'glacier', Acc: 99.01% ↑



Class: 'mountain', Acc: 99.56% ↑



Class: 'sea', Acc: 62.62% ↑



Class: 'sea', Acc: 85.17%



Class: 'mountain', Acc: 95.50% ✓



Class: 'sea', Acc: 100.00% ✓



Class: 'sea', Acc: 88.25% ✓



Class: 'mountain', Acc: 98.11% ↑



Class: 'sea', Acc: 99.92% ↑



Class: 'glacier', Acc: 66.46% ✓



Class: 'glacier', Acc: 50.41% ✗



Class: 'mountain', Acc: 99.88%



Class: 'sea', Acc: 97.08% ✓



Class: 'street', Acc: 97.91% ↑



Class: 'glacier', Acc: 98.85% ↑



Class: 'sea', Acc: 98.53% ✗



Class: 'sea', Acc: 99.99% ↑



Model VGG 16

Podział testowy: 30%

Liczba epok: 15

Wielkość partii: 32

Dokładność: 92.476%

Czas wykonywania: 2452.92s

Do ostatniej sieci wykorzystałem wytrenowany model VGG 16. Do optymalizacji użyłem algorytm Nadam.

Kod modelu VGG-16 i ostatecznego oraz kompilacji:

```
model_VGG = VGG16(  
    weights='imagenet',  
    include_top = False,  
    input_shape=(img_height, img_width, 3)  
)  
  
model = Sequential([  
    model_VGG,  
  
    keras.layers.Flatten(),  
  
    keras.layers.Dense(128, activation='relu'),  
    layers.Dropout(0.4),  
  
    layers.Dense(len(class_names))  
)  
  
model.compile(  
    optimizer=optimizers.Nadam(lr=2e-5),  
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
    metrics=['accuracy'])
```

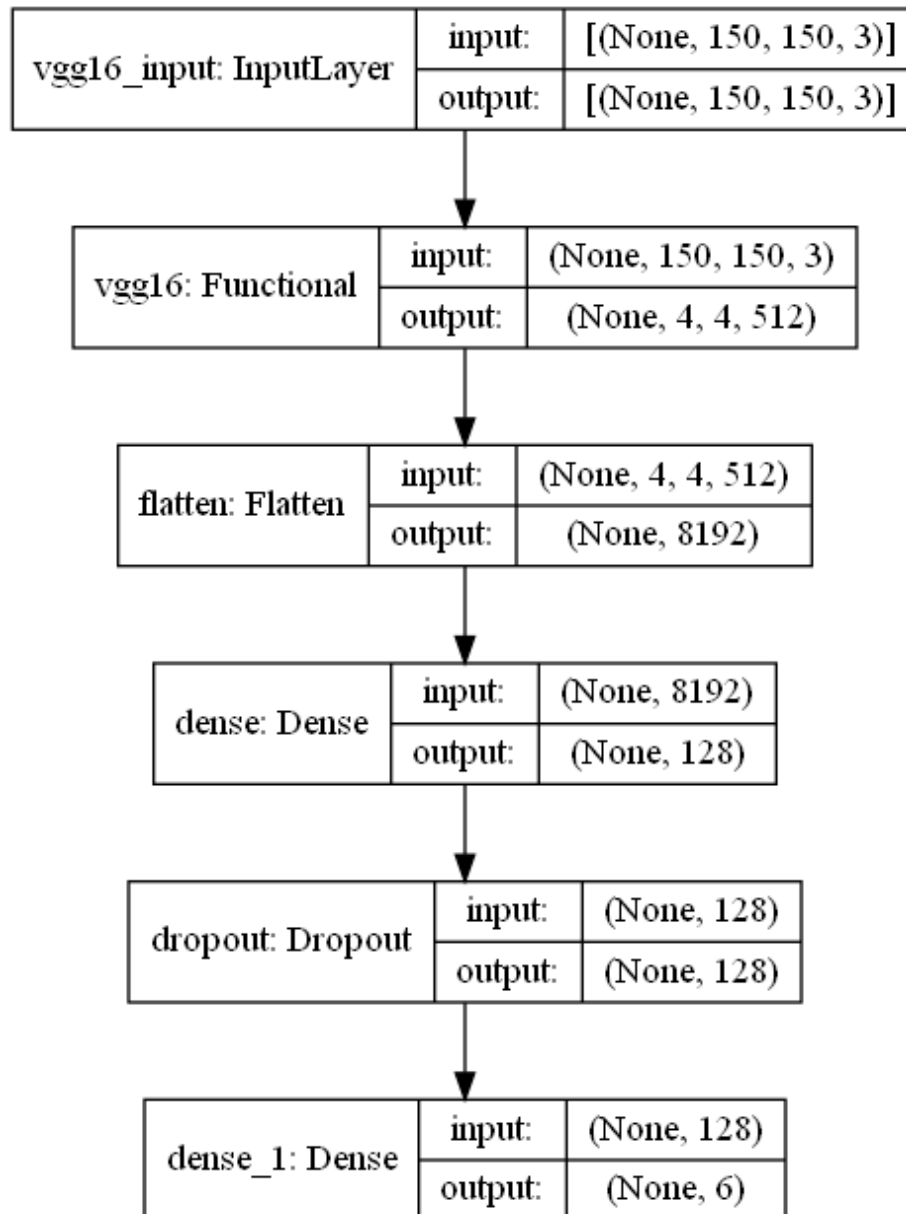
Podsumowanie modelu VGG-16 i ostatecznego:

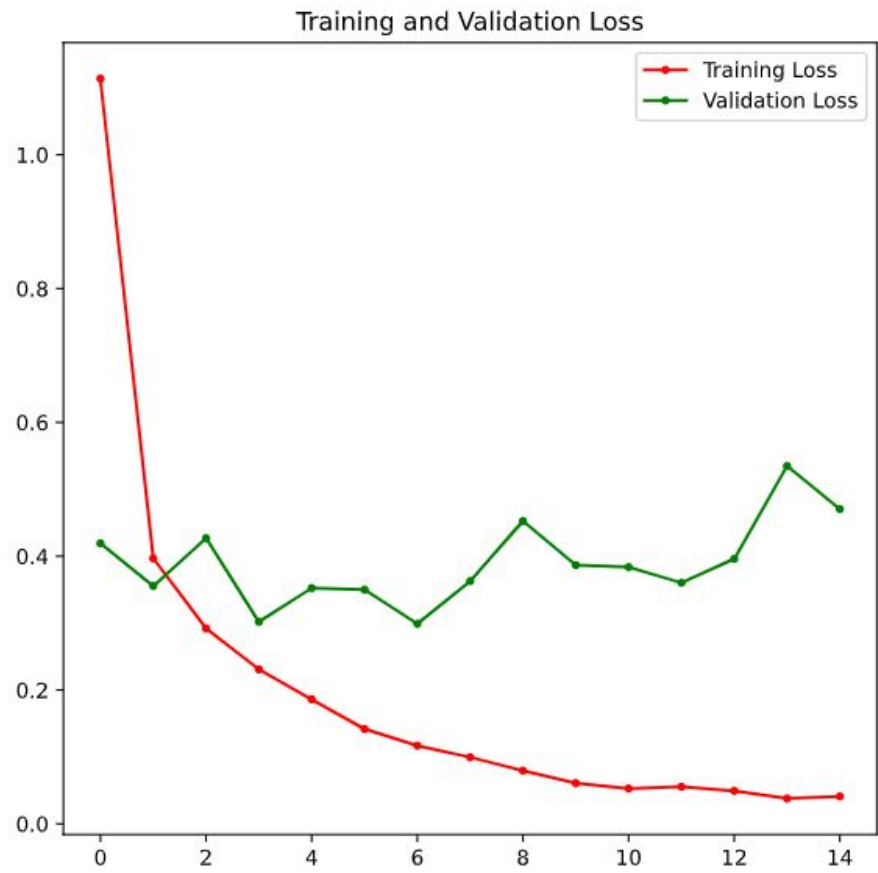
Model: "vgg16"

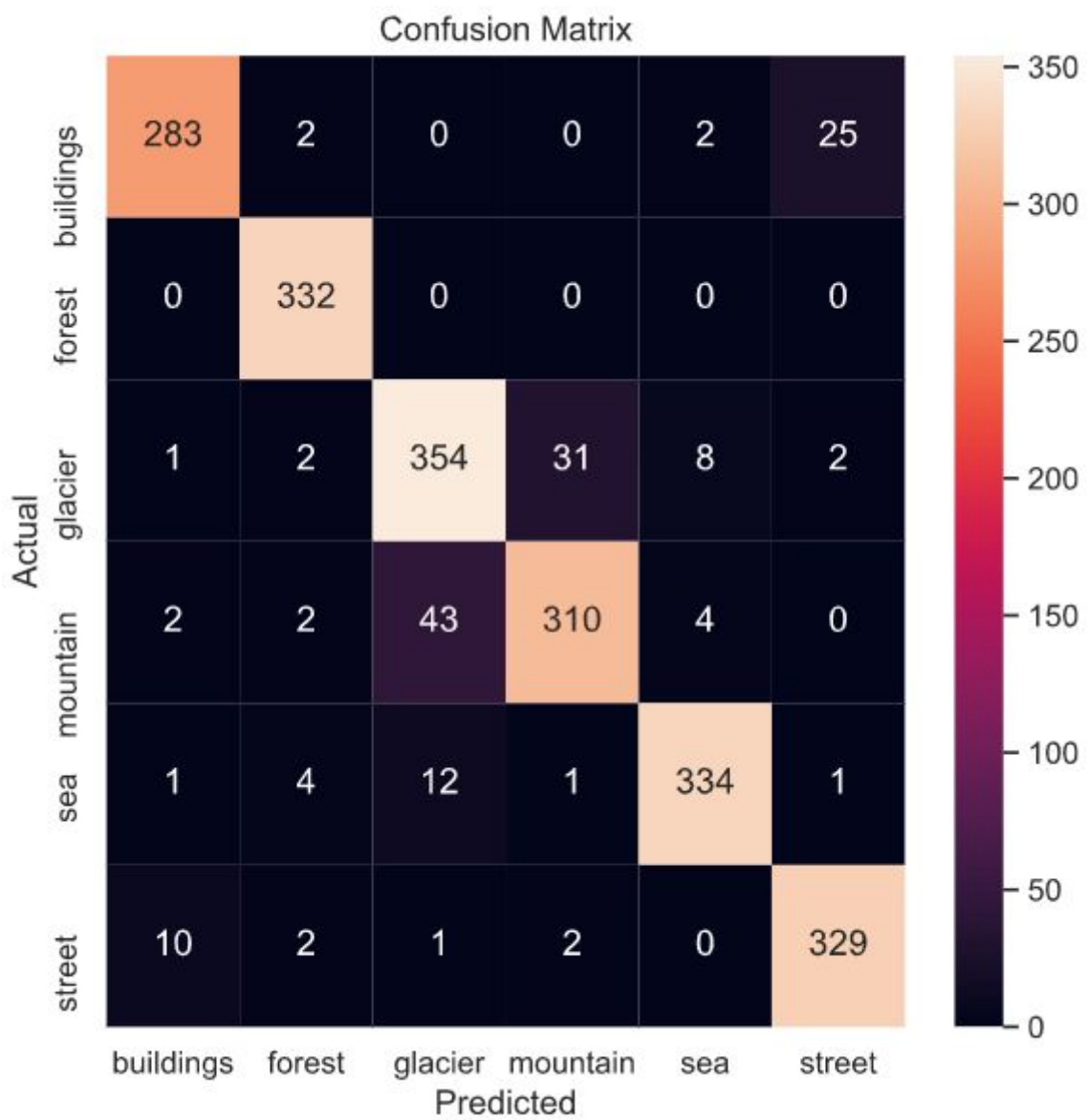
Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 150, 150, 3)]	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080

block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
=====		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		
Model: "sequential_1"		
Layer (type)	Output Shape	Param #
=====		
vgg16 (Functional)	(None, 4, 4, 512)	14714688
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 128)	1048704
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 6)	774
=====		
Total params: 15,764,166		
Trainable params: 15,764,166		
Non-trainable params: 0		

Podsumowanie modelu:







Przewidywanie klas:

Class: 'buildings', Acc: 98.80% ↑



Class: 'street', Acc: 100.00%



Class: 'street', Acc: 100.00%



Class: 'street', Acc: 99.97%



Class: 'glacier', Acc: 100.00% ↑



Class: 'buildings', Acc: 99.58%



Class: 'forest', Acc: 100.00%



Class: 'mountain', Acc: 95.05%



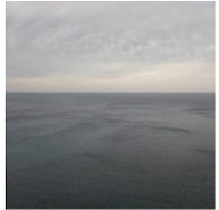
Class: 'glacier', Acc: 100.00%



Class: 'mountain', Acc: 99.90%



Class: 'sea', Acc: 100.00% ↑



Class: 'sea', Acc: 100.00% ↑



Class: 'mountain', Acc: 89.37% ↓



Class: 'sea', Acc: 100.00%



Class: 'sea', Acc: 100.00% ↑



Class: 'mountain', Acc: 99.61%



Class: 'sea', Acc: 100.00%



Class: 'glacier', Acc: 99.74% ↓



Class: 'mountain', Acc: 99.65% ✓



Class: 'mountain', Acc: 100.00%



Class: 'sea', Acc: 100.00%



Class: 'street', Acc: 100.00%



Class: 'glacier', Acc: 100.00%



Class: 'sea', Acc: 98.99% ✗



Class: 'sea', Acc: 100.00%






Podsumowanie

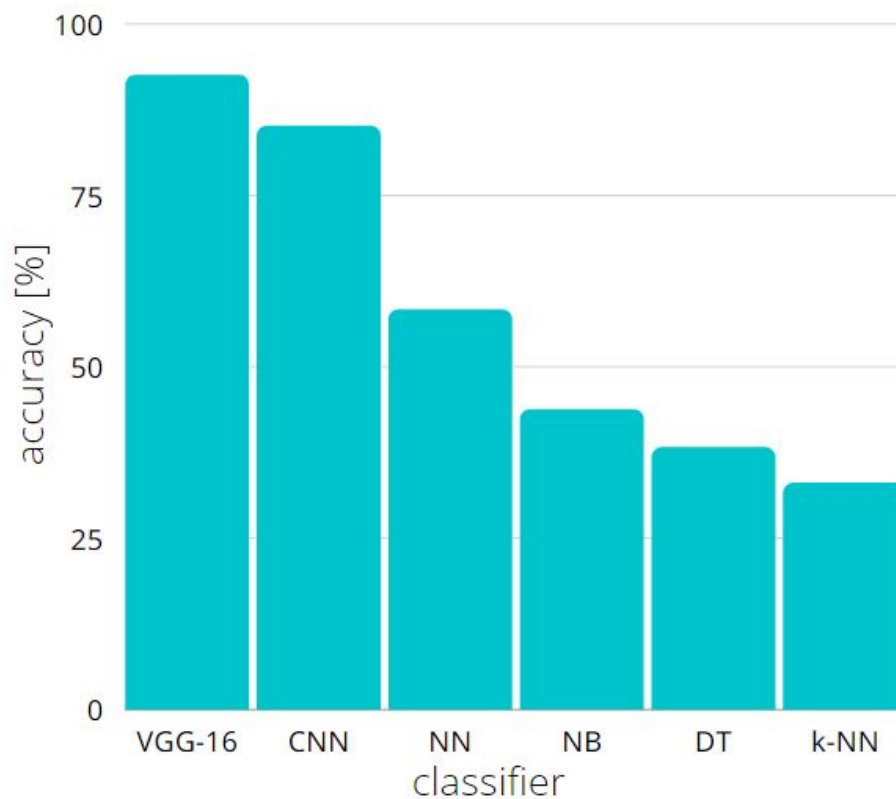
Klasyfikowanie całych scen, czyli obiektów geograficznych to dosyć trudny problem. Nie radzą sobie z nim klasyczne algorytmy. Najskuteczniejsze okazały się konwolucyjne sieci neuronowe.

Najtrudniejszymi obiektami do sklasyfikowania są góry i lodowce, nawet najlepszy model z 92% dokładności błędnie je klasyfikuje. Powodem tego jest duże podobieństwo w kształtach i dominujących kolorach między tymi dwoma klasami.

Rozmiary wytrenowanych modeli:

 projekt_2_classic_nn.h5	407 108 KB
 projekt_2_cnn_85pr.h5	4 151 KB
 projekt_2_cnn_VGG16_92pr.h5	184 861 KB

Celność klasyfikatorów:



Czas trenowania klasyfikatorów:

