Iceberg Challenge

SI

Grzegorz Podwika

03.06.2021

Streszczenie

Projekt ma na celu stworzenie klasyfikatora realizującego przewidywanie, czy dane zdjęcia satelitarne przedstawiają statek lub górę lodową. Program komputerowy na podstawie wcześniej dostarczonych danych, czyli zdjęć satelitarnych wraz z ich klasyfikacją dokonaną przez ekspertów i ich wiedzy geograficznej, trenuje klasyfikator który w kolejnym kroku zostaje poddany testom. Moim zadaniem jest analiza wyników programu i wybranie oraz opracowanie dającego najlepsze wyniki modelu klasyfikującego.

Spis treści

Rozdział 1

Wprowadzenie

Dryfujące góry lodowe od zawsze stanowiły zagrożenia dla nawigacji oraz różnych aktywności w miejscach takich jak morze wchodniego wybrzeża Kanady. Obecnie wiele instytucji oraz przeróżnych firm używa rozpoznania lotniczego, jak również wsparcia lądowego, aby monitorować czynniki środowiskowe i szacować ryzyko natrafienia na górę lodową. Jednakże, okazuje się że w pewnych obszarach z szczególnie surową pogodą te metody nie są wystarczające i w takich przypadkach z pomocą przychodzą satelity. Statoil jako międzynarodowa firma paliwowa blisko współpracuje z takimi podmiotami jak C-CORE, które używają danych z satelit od ponad 30 lat i mają zbudowane systemy nadzoru oparte na wizji komputerowej. Nie dziwi więc zainteresowanie Statoil'a w uzyskiwaniu nowych perspektyw, jak użyć uczenia maszynowego w celu jeszcze dokładniejszego wykrywania zagrożeń powodowanych przez góry lodowe.

1.1 Opis problemu

Podstawowym problemem jest stworzenie klasyfikatora, który przy klasyfikowaniu danych, czyli w tym przypadku czy dane zdjęcie przedstawia górę lodową czy statek, będzie popełniał możliwie najmniejszą ilość błędów przy predykcji następnych danych. Pod uwagę należy wziąć odpowiedni dobór danych na których nasz klasyfikator będzie trenowany, ponieważ na ich podstawie będzie podejmować decyzję. Trudnością w tym przypadku są także dwa zdjęcia satelitarne band_1, band_2 oraz kąt padania wiązki, ponieważ trzeba zdecydować, którego używać, a którego nie, a być może obu. Tworząc każdy model należy zadbać o to, by nie był niedouczony, ale także aby nie był przeuczony, ponieważ w takim przypadku model za bardzo podąża za

błędami/outlinerami niż za ogólnym trendem, którego chcemy uchwycić.

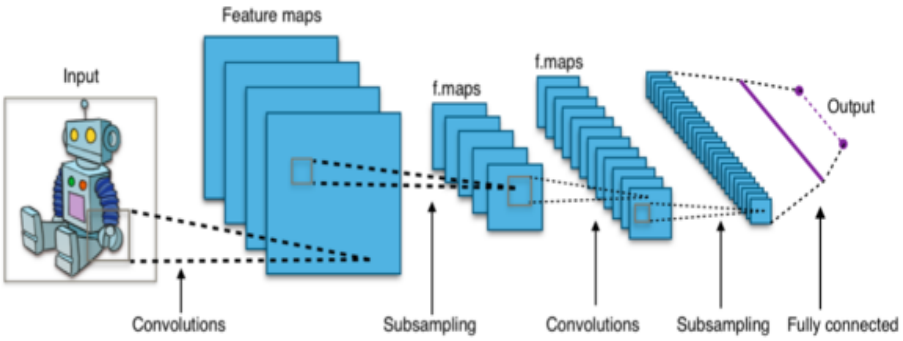
Rozdział 2

Realizacja

2.1 Wprowadzenie teoretyczne

Odmianą sieci neuronowej używaną do rozwiązania mojego problemu jest konwolucyjna/splotowa sieć neuronowa (Convolutional Neural Network).

Typowa architektura sieci splotowej:



Zaczynając od początku, możemy zdefiniować konwolucję jako operację matematyczną, która powoduje nałożenie filtra na wejściowy sygnał. W uczeniu maszynowym wykorzystujemy takie filtry do tworzenia sieci splotowych, które mają filtrować jakieś dane, najczęściej obrazy i je klasyfikować. Stosując konwolucję uzyskujemy takie zyski jak:

- po przetworzeniu obrazu filtrem pewne cechy obrazy zostają wypuklone, co ułatwia ich rozpoznawanie (feature extraction)
- z reguły nakładamy wiele filtrów naraz i każdy z nich może uwypuklać inne cechy. Np. przy rozpoznawaniu twarzy

jeden uwypukli oczy, drugi uszy, linię włosów lub ich brak

- w wyniku konwolucji uniezależniamy się od położenia obiektu na obrazie
- zmniejszamy szum na analizowanych obrazkach skupiając uwagę sieci na kluczowych cechach
- z reguły jednym z etapów w konwolucyjnej sieci neuronowej jest warstwa wykonująca operację tzw. poolingu, czyli pewnego rodzaju połączenia wartości kilku sąsiadujących pikseli w jeden. Zmniejsza to istotnie moc obliczeniową niezbędną do nauczenia sieci, nie tracąc jednocześnie ważnych informacji

W ogólności sieć splotowa składa się z warstwy wejściowej, wyjściowej jak i wielu warstw ukrytych pomiędzy nimi. Warstwy ukryte to najczęściej warstwy konwolucyjne/splotowe, warstwy aktywacji RELU, warstwy poolingowe(łączące), w pełni połączone lub wartswy normalizujące. Warstwy splotowe są podstawą sieci konwolucyjnych CNN, ponieważ zawierają wyuczone filtry (kernele), które wyodrębniają cechy odróżniające od siebie różne obrazy. Wartości w filtrach są dobierane i optymalizowane podczas trenowania sieci. Wagi dobierane są do filtra, który jest następnie przesuwany po całym zdjęciu. Tworząc warstę konwulucyjną musimy zdefiniować kilka hiperparametrów jak: rozmiar jądra (kernel size), wypełnienie (padding), kroki (strides), czy funkcję aktywacji.

2.2 Opis danych wejściowych

Zbiór danych na których działać będzie klasyfikator składa się z 1064 rekordów, z których każdy określa górę lodową lub statek. W każdej krotce znajduje się 4 cechy: id, band_1, band_2 oraz inc_angle.

- id - unikatowy identyfikator zdjęcia
- band_1, band_2 - spłaszczone dane zdjęcia. Każde pasmo ma 75x75 wartości w liście, więc spłaszczona lista ma 5625 elementów. Każda wartość ma fizyczne znaczenie - jest to liczba zmiennoprzecinkowa float z jednostką dB. Band 1 i Band 2 to sygnały charakteryzujące rozpraszanie wsteczne radaru wytwarzane przez różne polaryzacje przy określonym kącie padania. Polaryzacje odpowiadają HH (nadawaj/odbieraj poziomo) i HV (nadawaj poziomo i odbieraj pionowo).
- inc_angle - kąt padania sygnału, gdy robiono zdjęcia.

Na ich podstawie wyznaczana jest etykieta is_iceberg, która określa czy zdjęcia przedstawiają górę lodową - fałsz ("0"), prawda ("1").

Szybkie spojrzenie na dane:

```
# Closer look at data
train.head()
```

	id	band_1	band_2	inc_angle	is_iceberg
0	dfd5f913	[-27.878360999999998, -27.15416, -28.668615, -...	[-27.154118, -29.537888, -31.0306, -32.190483,...	43.9239	0
1	e25388fd	[-12.242375, -14.920304999999999, -14.920363, ...	[-31.506321, -27.984554, -26.645678, -23.76760...	38.1562	0
2	58b2aaa0	[-24.603676, -24.603714, -24.871029, -23.15277...	[-24.870956, -24.092632, -20.653963, -19.41104...	45.2859	1
3	4cfc3a18	[-22.454607, -23.082819, -23.998013, -23.99805...	[-27.889421, -27.519794, -27.165262, -29.10350...	43.8306	0
4	271f93f4	[-26.006956, -23.164886, -23.164886, -26.89116...	[-27.206915, -30.259186, -30.259186, -23.16495...	35.6256	0

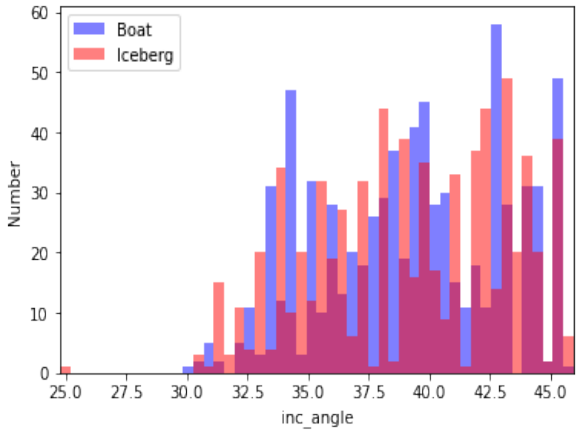
Id ma postać unikatowego Stringa, band_1, band_2 oraz inc_angle są wartościami zmiennoprzecinkowymi, is_iceberg jest wartością Boolean. Dane jak widać nie są znormalizowane.

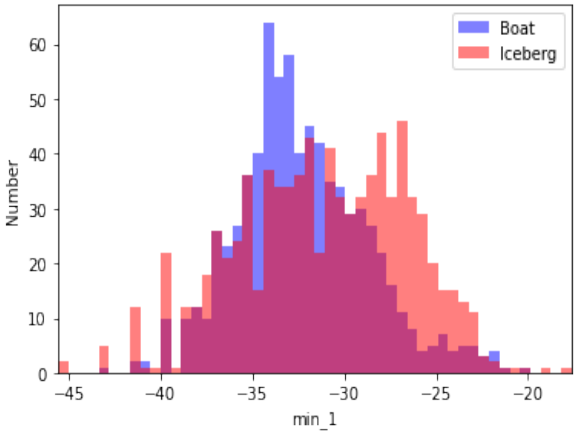
Spojrzenie na dane wraz z statystykami:

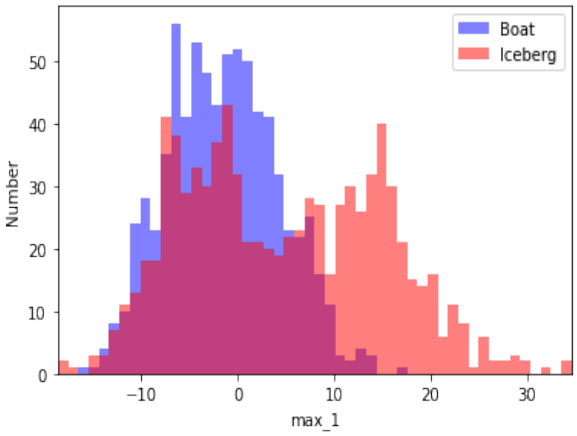
	id	band_1	band_2	inc_angle	is_iceberg	max_1	min_1	med_1	std_1	mean_1	max_2	min_2	m
0	dfd5f913	[-27.878360999999998, -27.15416, -28.668615, -...	[-27.154118, -29.537888, -31.0306, -32.190483,...	43.9239	0	-0.213149	-38.211376	-27.879921	2.764537	-27.911043	-11.252153	-41.135918	-30.00
1	e25388fd	[-12.242375, -14.920304999999999, -14.920363, ...	[-31.506321, -27.984554, -26.645678, -23.76760...	38.1562	0	12.570409	-23.125309	-13.654199	3.142532	-13.566554	0.044052	-34.765831	-25.48
2	58b2aaa0	[-24.603676, -24.603714, -24.871029, -23.15277...	[-24.870956, -24.092632, -20.653963, -19.41104...	45.2859	1	-9.918477	-33.391197	-22.935019	2.223905	-23.053698	-15.605879	-34.148819	-24.60
3	4cfc3a18	[-22.454607, -23.082819, -23.998013, -23.99805...	[-27.889421, -27.519794, -27.165262, -29.10350...	43.8306	0	4.795627	-32.204136	-23.303238	2.566233	-23.210771	-5.554516	-39.564053	-29.54
4	271f93f4	[-26.006956, -23.164886, -23.164886, -26.89116...	[-27.206915, -30.259186, -30.259186, -23.16495...	35.6256	0	-6.956036	-35.010487	-25.206615	2.305288	-25.280029	-9.434006	-40.276115	-25.46

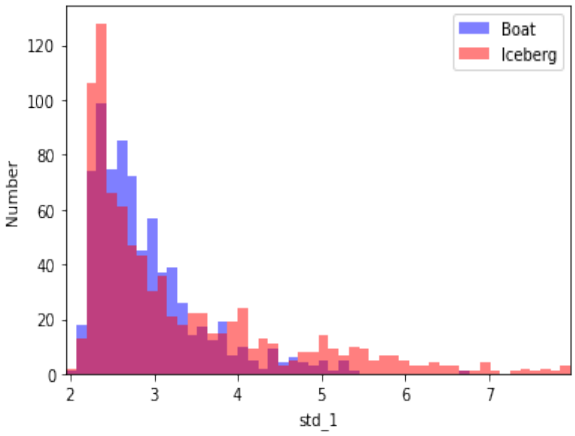
Dla każdego obiektu, dane statystyczne wahają się znacznie, wizualnie lepsze porównanie jest umieszczone poniżej.

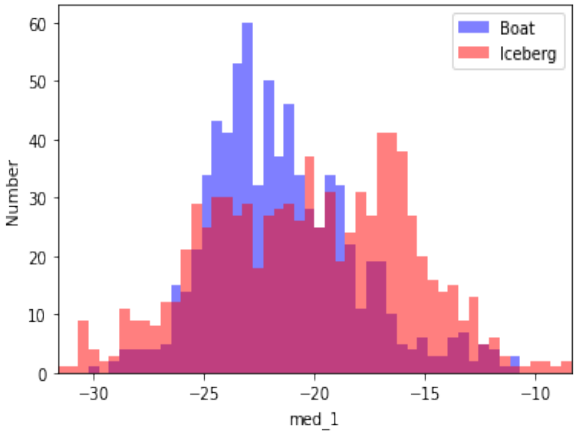
Rysowanie danych statystycznych:

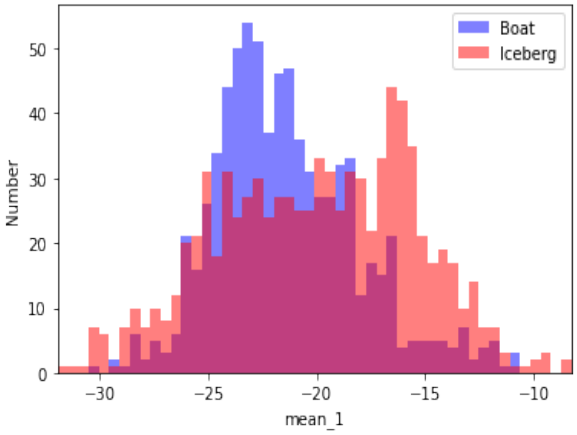


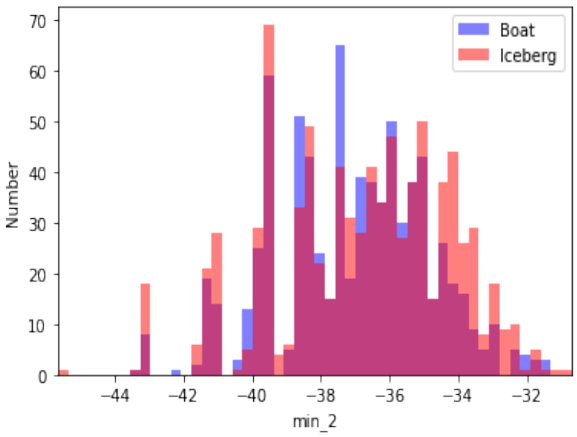


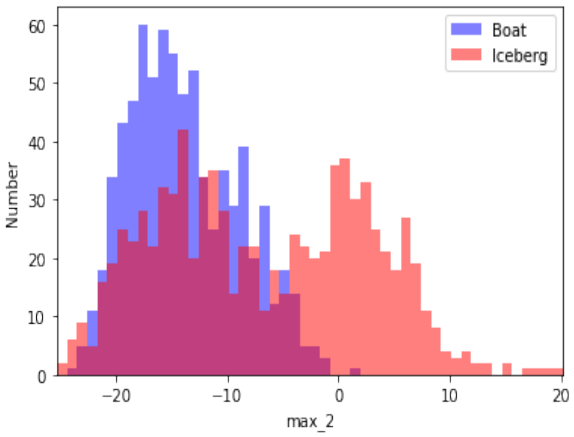

Boat
Iceberg
Number
max_1

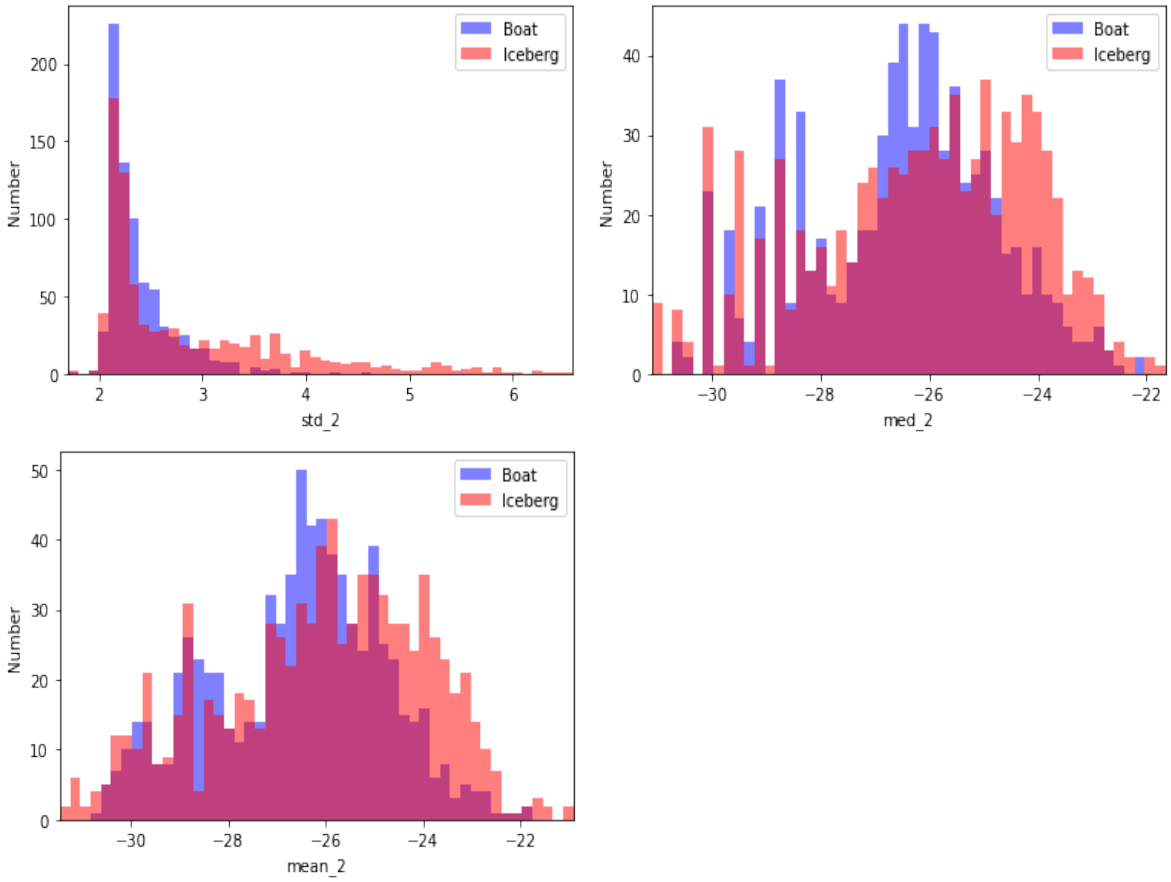

Boat
Iceberg
Number
std_1


Boat
Iceberg
Number
med_1


Boat
Iceberg
Number
mean_1

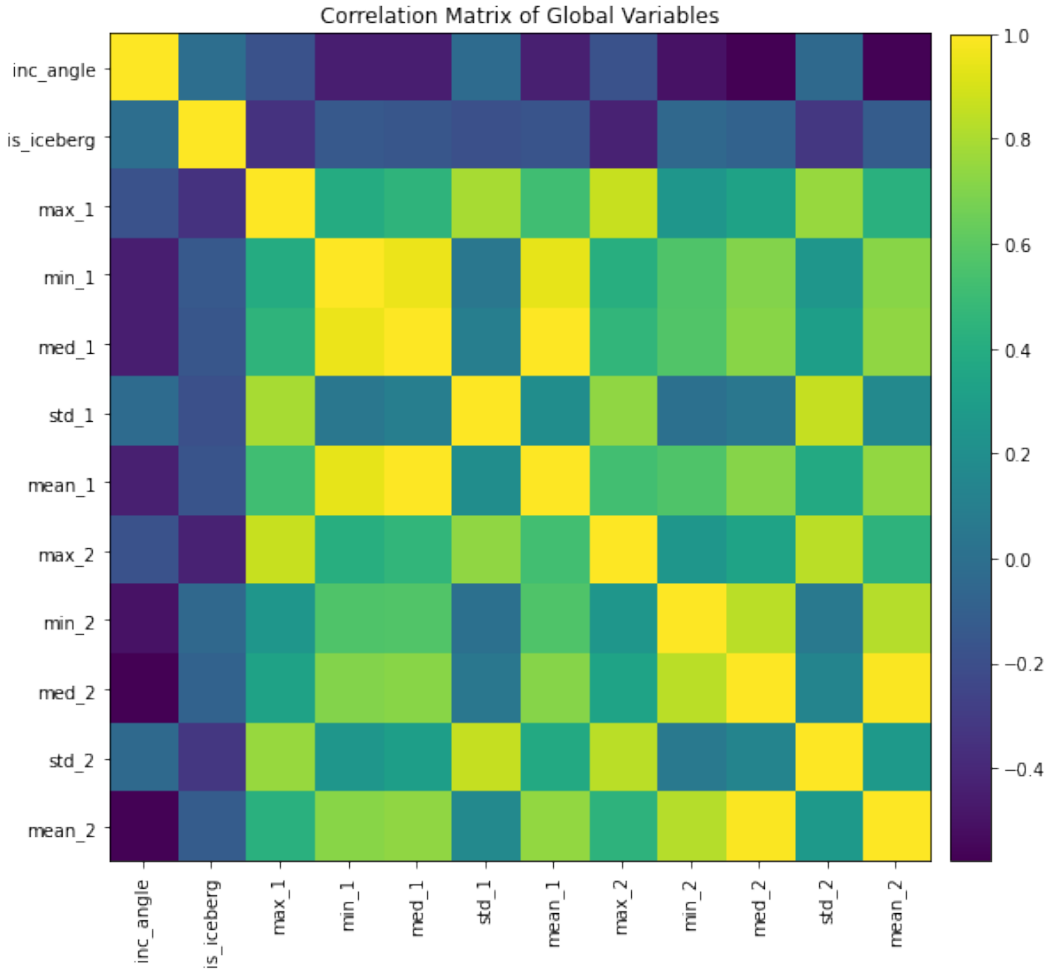

Boat
Iceberg
Number
min_2


Boat
Iceberg
Number
max_2



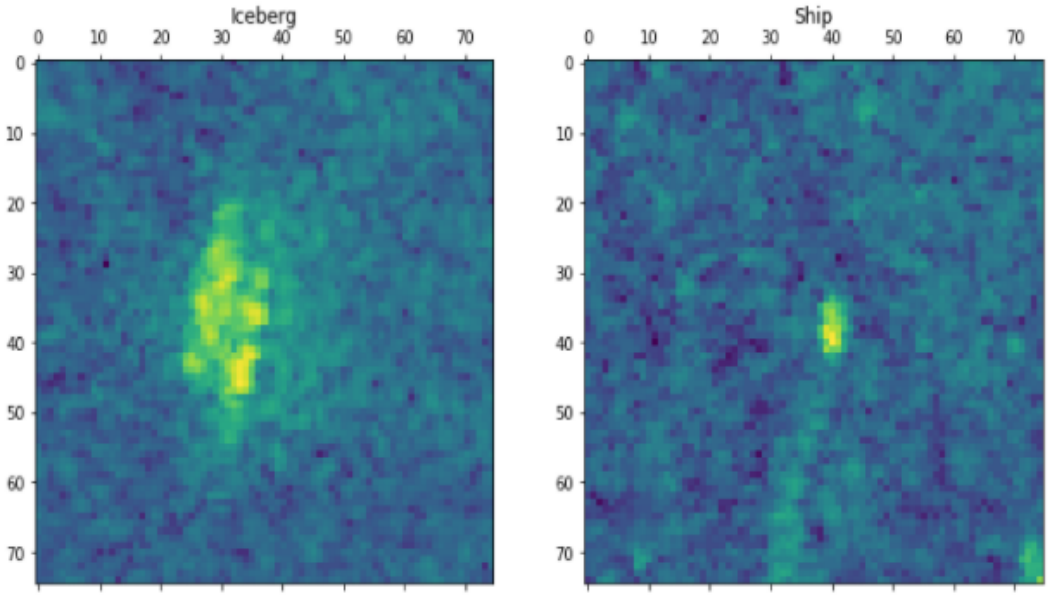
Dla pasma 1 (band_1) widzimy pewne znaczące różnice. Środkowy przedział (okolice 50%) ma mniej więcej taką samą wielkość dla obu etykiet, ale minimum, maksimum, odchylenie standardowe, mediana i średnia wykazują zauważalne różnice w pewnym zakresie wartości. Najwyraźniej te zmienne wydają się mieć pewną wrażliwość na to, co próbujemy zmierzyć. Moglibyśmy się tego spodziewać, gdyby na przykład góry lodowe były znacznie większe od statków, a zatem zajmowały więcej pikseli. Dla pasma 2 (band_2) otrzymałem podobne wyniki.

Wzajemna korelacja cech:

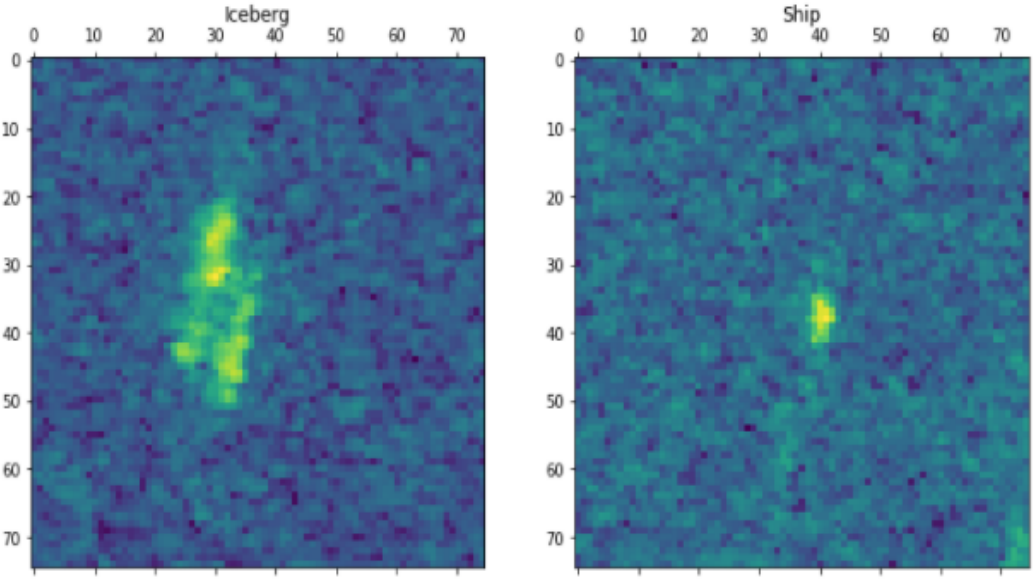


Widzimy dużą korelację pomiędzy niektórymi zmiennymi. W szczególności, mediana, min i wartość średnia są ze sobą wysoce skorelowane. Min i max są także skorelowane dla band_1, tak jak min i mediana dla obu pasm. Można zauważyć także pewne powiązania między pasmami. Na koniec widzimy antykorelację (-0.5) między wartością średnią, jak i medianą, a wartością inc_angle dla pasma band_2.

Spojrzenie 2D na górę lodową id12 i statek id14 (band_1):



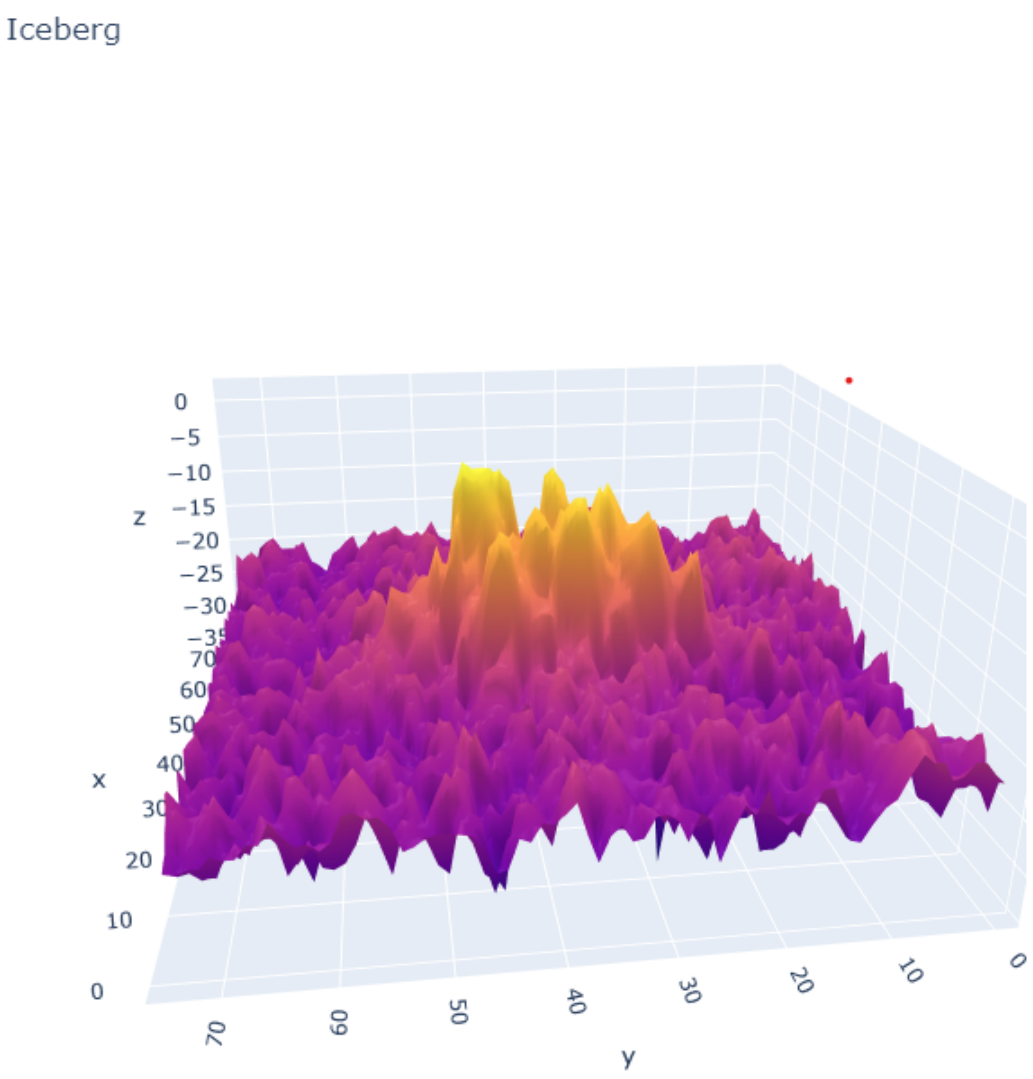
Spojrzenie 2D na górę lodową id12 i statek id14 (band_2):



Obrazy z poszczególnych pasm nie wiele się różnią. Widzimy tak-

że, że zdjęcia jako takie nie są za wysokiej jakości (75x75 pikseli). Widzimy także na tym przykładzie, że góra lodowa zajmuje (powinna zajmować) więcej pikseli, gdzie statek zajmuje ich znacznie mniej. Kształt góry lodowej, czy statku jest znacznie jaśniejszy na zdjęciu od otaczającej ten obiekt wody.

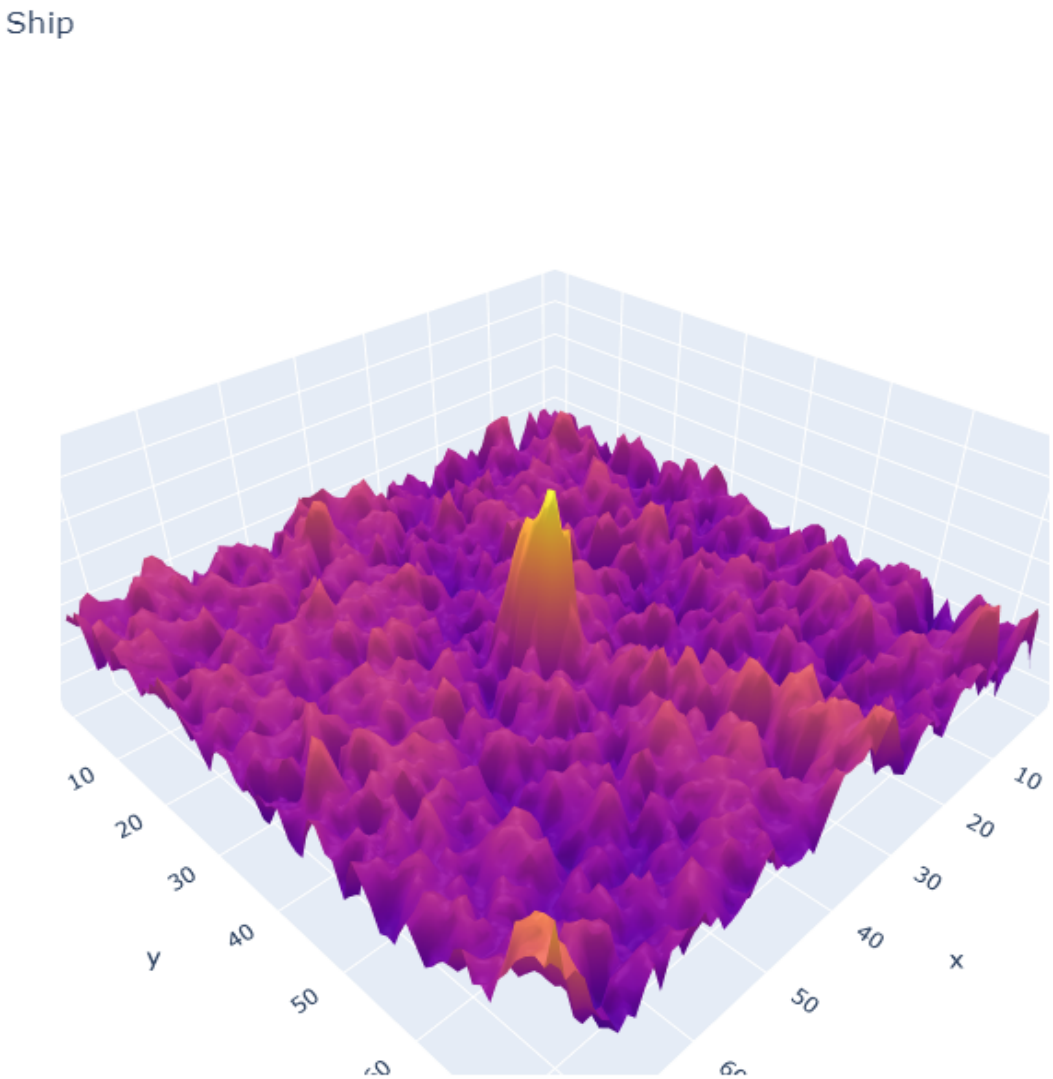
Spojrzenie 3D na górę lodową id12 (band_1):



Zdjęcie przestawia ciekawy rzut 3D na górę lodową. Nie jest to rzeczywiste zdjęcie, lecz tylko dane z radaru, dlatego też

kształt będzie miał różne wierzchołki i zniekształcenia.

Spojrzenie 3D na statek id14 (band_1):



Zdjęcie przestawia ciekawy rzut 3D na statek. W tym przedstawieniu statek wygląda jak pociągły punkt.

2.3 Przekształcenia w zbiorze danych

Syngały pochodzą od satelity Sentinal, która ma tylko H-transmitter (poziomy), dlatego wyjściowe sygnały mają postać HH i HV. Dlatego też wyodrębniłem te dwa pasma oraz wziąłem ich średnią jako trzecie pasmo, aby stworzyć 3 kanałowy odpowiednik RGB.

```
#Create 3 bands having HH, HV and avg of both
X_band_1=np.array([np.array(band).astype(np.float32).reshape(75, 75) for band in train["band_1"]])
X_band_2=np.array([np.array(band).astype(np.float32).reshape(75, 75) for band in train["band_2"]])
X_train = np.concatenate([X_band_1[:, :, :, np.newaxis],
                          X_band_2[:, :, :, np.newaxis],
                          ((X_band_1+X_band_2)/2)[:, :, :, np.newaxis]], axis=-1)

# Feature scaling doesn't make better results
# X_train /= 255
```

2.4 Normalizacja i standaryzacja danych

Moje testy wykazały, że skalowanie danych nie przynosi żadnych pozytywnych rezultatów, więc zaniechałem tej operacji.

2.5 Podział danych

Dane podzieliłem zgodnie z proporcjami 75% na dane treningowe oraz 25% na dane testujące.

```
target_train=train['is_iceberg']
X_train_cv, X_valid, y_train_cv, y_valid =
train_test_split(X_train, target_train, random_state=1, train_size=0.75)
```

2.6 Ustawienie stałych parametrów

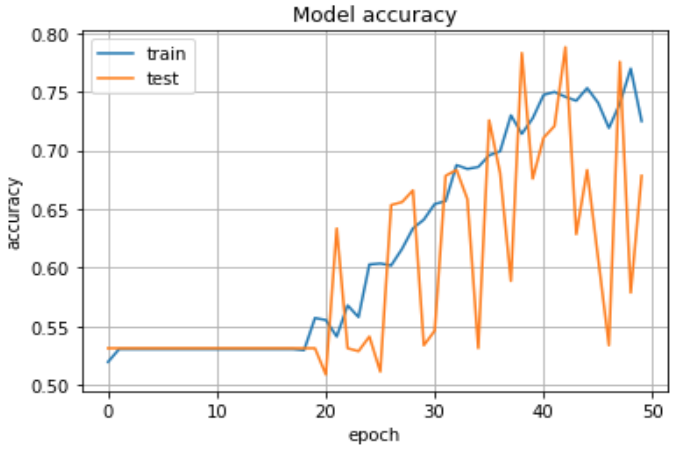
```
batch_size = 24
epochs = 50

# initiate optimizers
opt_sgd = keras.optimizers.SGD()
opt_adam = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
```

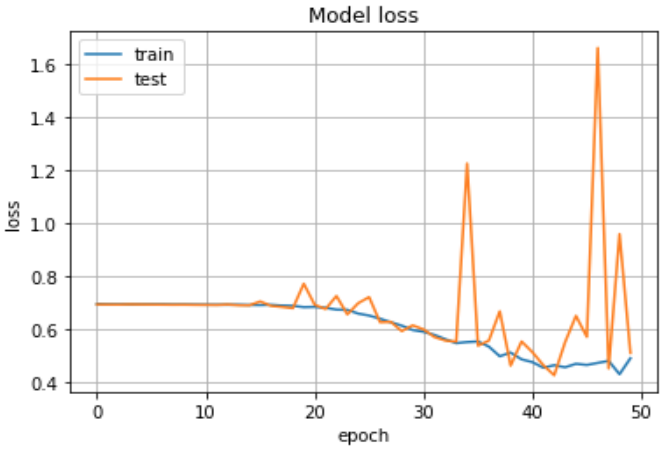
2.7 Trenowanie i ewaluacja modeli

Poniżej ukazane są wyniki dla modeli sieci splotowych uzyskane eksperymentalnie. Jakość modelu jest przedstawiona za pomocą 4 wskaźników takich jak: loss, accuracy, val_loss i val_accuracy. Te wskaźniki pokazują odpowiednio błąd i dokładność dla danych treningowych i testujących dla danej epoki.

Model 1 z optymalizatorem SGD:

```
Epoch 50/50
51/51 [==============================] - 5s 90ms/step - loss: 0.4882 - accurac
y: 0.7249 - val_loss: 0.5082 - val_accuracy: 0.6783
```

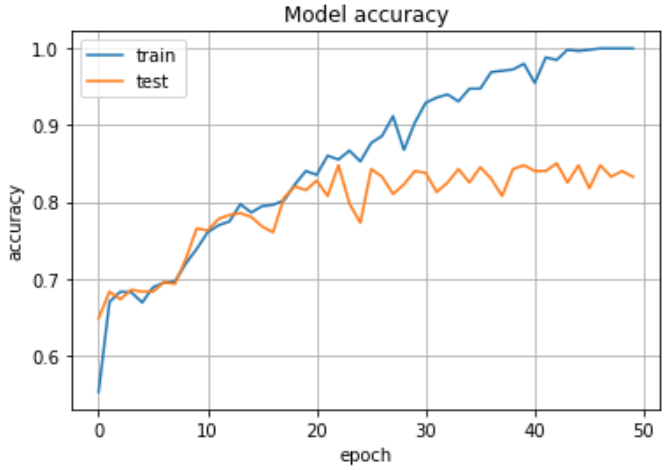


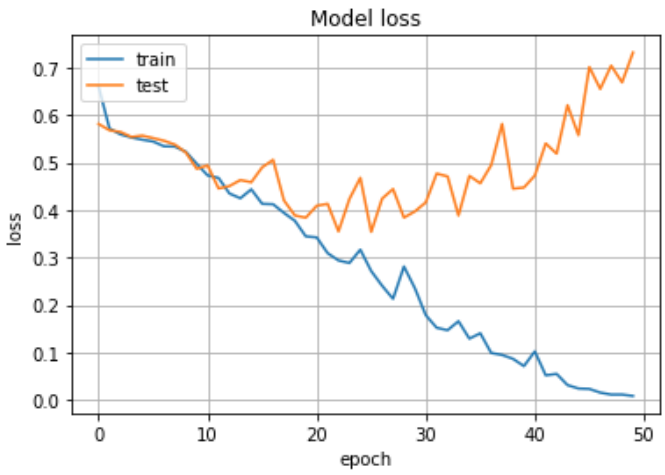


Podstawowy model uzyskał dokładność rzędu 72.5%, natomiast patrząc na wahania wskazań val_loss i val_accuracy postanowiłem zmienić optymalizator.

Model 1 z optymalizatorem Adam:

```
Epoch 50/50
51/51 [==============================] - 4s 86ms/step - loss: 0.0083 - accurac
y: 1.0000 - val_loss: 0.7323 - val_accuracy: 0.8329
```

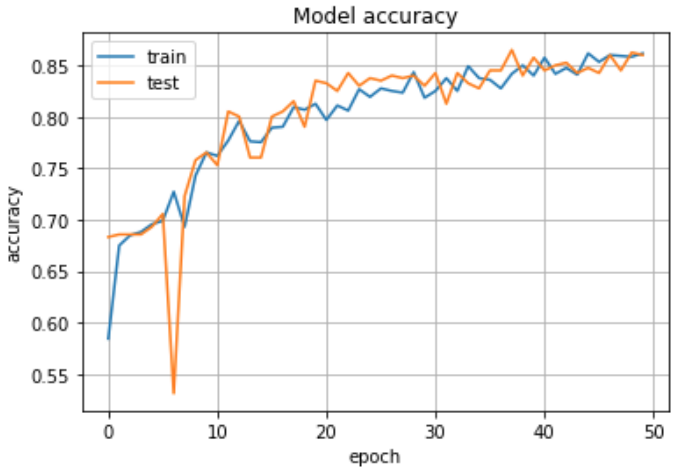


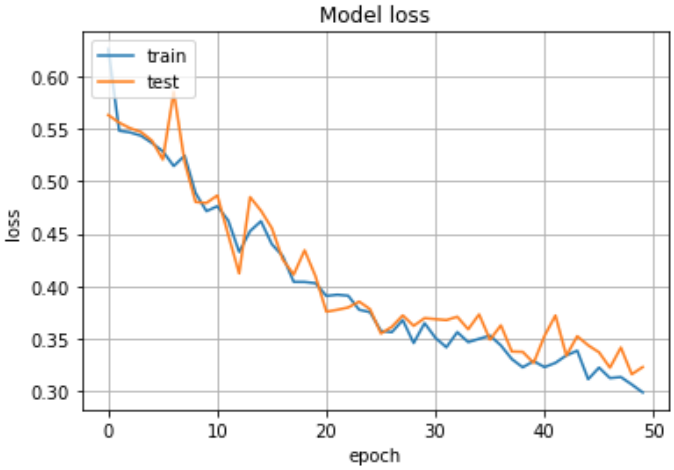


Podstawowy model uzyskał perfekcyjną dokładność rzędu 100%, natomiast patrząc wskazania val_acc, które spadało od 20 epoki oraz val_loss, które stale wzrastało od 20 epoki, można wysnuć wniosek, że ten model jest przeuczony i nie będzie się nadawał do późniejszego stosowania.

Model 2 z optymalizatorem Adam (najlepszy):

```
Epoch 50/50
51/51 [==============================] - 4s 82ms/step - loss: 0.2989 - accurac
y: 0.8620 - val_loss: 0.3230 - val_accuracy: 0.8603
```

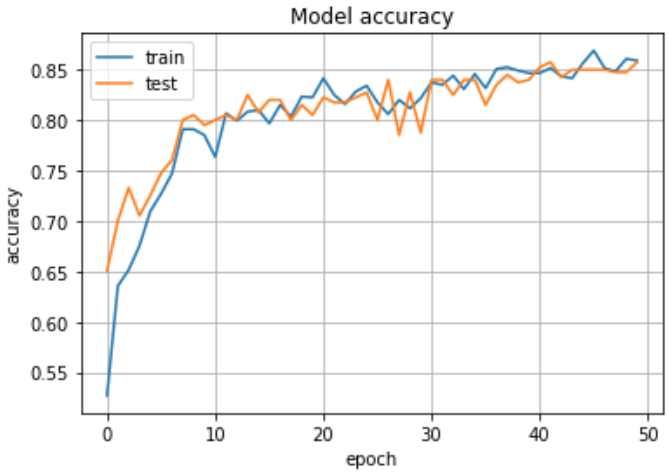


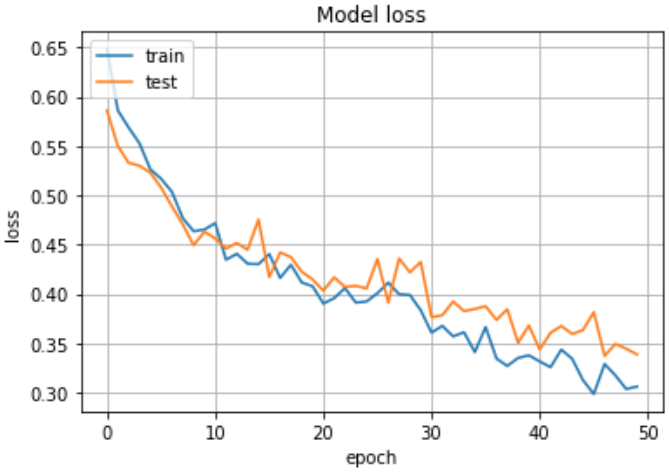


Drugi model z dodanymi Dropoutami nie przetrenowuje się, co widać od razu po stabilnym wzroście dokładności i spadku wartości błędu. Ten model uzyskuje dokładność rzędu 86.2% co jest najlepszym wynikiem jaki uzyskałem. Widać także, że jest tutaj miejsce na ewentualne dotrenowanie modelui przez X epok.

Model 2 z optymalizatorem Adam i większym startowym kernelem (7,7):

```
Epoch 50/50
51/51 [==============================] - 5s 106ms/step - loss: 0.3060 - accurac
y: 0.8595 - val_loss: 0.3385 - val_accuracy: 0.8579
```

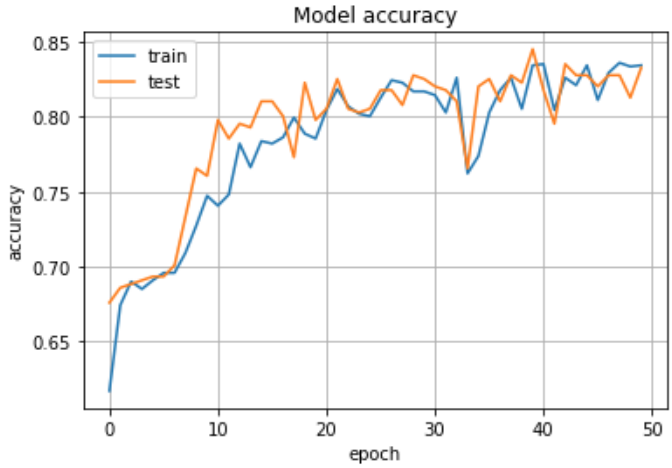


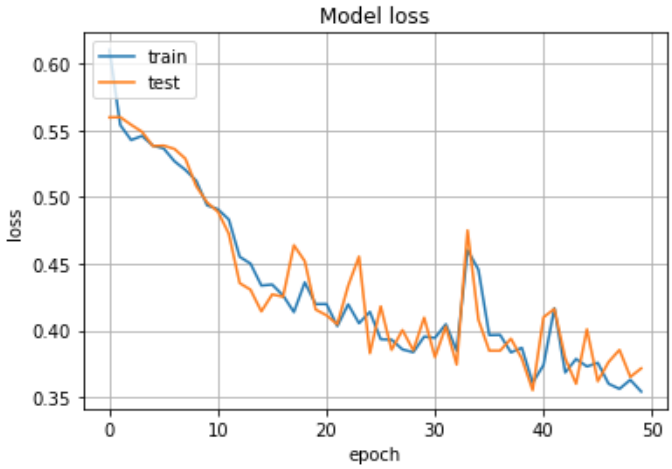


Model 2 z większym startowym kernelem (7, 7) uzyskał bardzo zbliżone wyniki do poprzedniego, widać że w tym przypadku nic taka optymalizacja nie daje.

Model 2 z optymalizatorem Adam i zeskalowanymi danymi:

```
Epoch 50/50
51/51 [==============================] - 5s 93ms/step - loss: 0.3543 - accurac
y: 0.8346 - val_loss: 0.3716 - val_accuracy: 0.8329
```

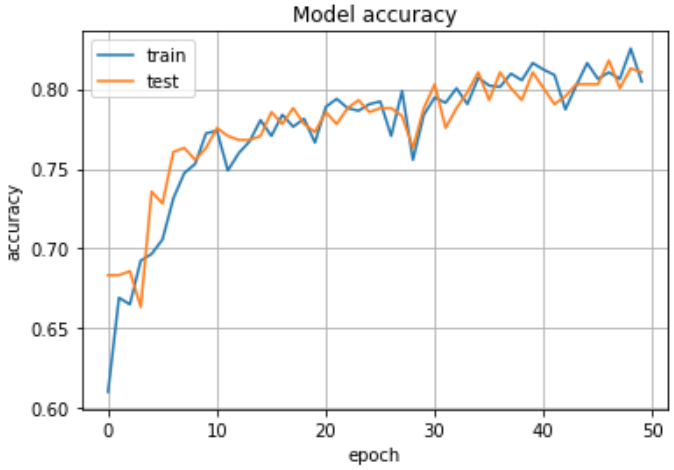


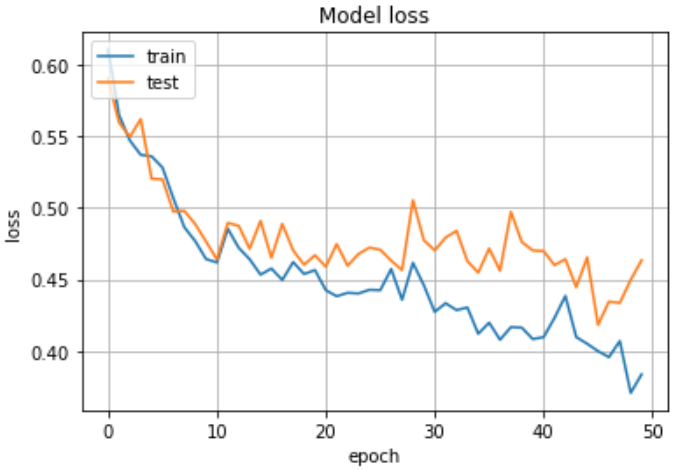


Model 2 działający na zeskalowanych danymi przyniósł pogorszenie jakości modelu, więc stwierdziłem aby nie skalować danych globalnie, dla wszystkich modeli.

Model 3 z optymalizatorem Adam (większa ilość warstw gęstych):

```
Epoch 50/50
51/51 [==============================] - 5s 95ms/step - loss: 0.3837 - accurac
y: 0.8047 - val_loss: 0.4636 - val_accuracy: 0.8105
```





W modelu 3 spróbowałem dodać więcej warstw gęstych przed przed funkcją klasyfikującą sigmoidalną, ale nie przyniosło to pozytywnych rezultatów, model osiągnął dokładność rzędu 80%.

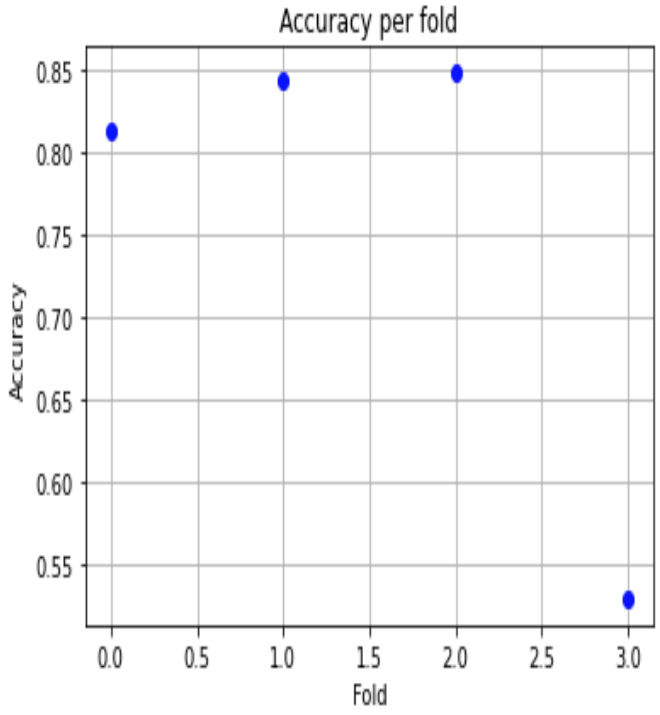
2.8 Badania symulacyjne

Metodą użytą do testowania najlepszego modelu (2) jest walidacja krzyżowa, a konkretnie StratifiedKFold. Jest to wariant zwykłego KFold'a, który zwraca warstwowo uporządkowane foldy. Foldy są tworzone poprzez zachowanie procentu próbek dla każdej z klas. Dokładniej mówiąc w każdym zbiorze trenującym znajdzie się 75% gór lodowcowych i statków, a w zbiorze trenującym proporcjonalnie 25% gór lodowcowych i statków. Przeprowadzone badanie ukazało, że pierwotnie podzielony random_statem zbiór na dane treningowe i testujące spisuje się bardzo dobrze i nie trzeba koniecznie rozdysponowywać danych proporcjonalnie.

Dokładność dla poszczególnego folda:

```
print(accuracy_per_fold)
```

```
[0.8129675984382629, 0.8428927659988403, 0.8478803038597107, 0.528678297996521]
```



Rozdział 3

Podsumowanie

Wytrenowany model sieci splotowej radzi sobie dobrze z klasyfikowaniem gór lodowcowych na morzu. Skalowanie zbioru nie poprawia jakości modeli. Wstępna analiza danych wykazała korelację pomiędzy niektórymi wartościami statystycznymi. Narysowanie przykładów góry lodowcowej i statku w przestrzenii dwuwymiarowej ukazało ciężkie zadanie klasyfikacji dla człowieka, gdzie właśnie sieci głębokie jak sieci splotowe mogą się wykazać. Rzut w 3D ukazał bardziej jakościowe spojrzenie na zdjęcia z perspektywy człowieka. Eksperymenty z modelami sieci splotowych ukazały, że najlepszym optymalizatorem jest Adam, że nie należy przesadzać z ilością warstw gęstych, że Dropouty są bardzo przydatne, aby ograniczyć przetrenowywanie się modelu, a także że warstwy MaxPoolingu także podnoszą jakość modelu. Krosswalidacja pokazała, że pierwotnie podzielone dane na zbiory treningowe i testujące są dobrze dobrane i można poprawnie trenować na nich modele. W celu zapewnienia lepszej dokładności prawdopodobnie należałoby skorzystać z dużo bardziej zaawansowanych sieci głębokich, których czas trenowania jest diametralnie dłuższy.

Dodatek A

Kod programu

```
### IMPORTS

import numpy as np
import pandas as pd
import keras
import seaborn as sns
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from sklearn.model_selection import train_test_split
from os.path import join as opj
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import StratifiedKFold

import plotly.offline as py
import plotly.graph_objs as go
import pylab
plt.rcParams['figure.figsize'] = 10, 10
%matplotlib inline

### Load data

train = pd.read_json("train.json")
test = pd.read_json("test.json")

### Exploratory data analysis

# Closer look at data
train.head()
```

```
# Calculate means, medians, mins and maxes for bands and take a look on it
data_train_visualisation = train
data_train_visualisation['inc_angle'] = pd.to_numeric(data_train_visualisation['inc_

data_train_visualisation['max_1'] = [np.max(np.array(x)) for x in train['band_1'] ]
data_train_visualisation['min_1'] = [np.min(np.array(x)) for x in train['band_1'] ]
data_train_visualisation['med_1'] = [np.median(np.array(x)) for x in train['band_1']
data_train_visualisation['std_1'] = [np.std(np.array(x)) for x in train['band_1'] ]
data_train_visualisation['mean_1'] = [np.mean(np.array(x)) for x in train['band_1']

data_train_visualisation['max_2'] = [np.max(np.array(x)) for x in train['band_2'] ]
data_train_visualisation['min_2'] = [np.min(np.array(x)) for x in train['band_2'] ]
data_train_visualisation['med_2'] = [np.median(np.array(x)) for x in train['band_2']
data_train_visualisation['std_2'] = [np.std(np.array(x)) for x in train['band_2'] ]
data_train_visualisation['mean_2'] = [np.mean(np.array(x)) for x in train['band_2']

data_train_visualisation.head()

### Plotting the Statistics

def plot_var(name,nbins=50):
    minval = data_train_visualisation[name].min()
    maxval = data_train_visualisation[name].max()
    plt.hist(data_train_visualisation.loc[data_train_visualisation.is_iceberg==1,nam
             bins=nbins,color='b',alpha=0.5,label='Boat')
    plt.hist(data_train_visualisation.loc[data_train_visualisation.is_iceberg==0,nam
             bins=nbins,color='r',alpha=0.5,label='Iceberg')
    plt.legend()
    plt.xlim([minval,maxval])
    plt.xlabel(name)
    plt.ylabel('Number')
    plt.show()

for col in ['inc_angle','min_1','max_1','std_1','med_1','mean_1','min_2','max_2','st
    plot_var(col)

### Plotting correlations between features

train_stats = data_train_visualisation.drop(['id','band_1','band_2'],axis=1)

corr = train_stats.corr()
fig = plt.figure(1, figsize=(10,10))
plt.imshow(corr,cmap='viridis')
```

```
labels = np.arange(len(train_stats.columns))
plt.xticks(labels,train_stats.columns,rotation=90)
plt.yticks(labels,train_stats.columns)
plt.title('Correlation Matrix of Global Variables')
cbar = plt.colorbar(shrink=0.85,pad=0.02)
plt.show()

### Generate the training data

#Create 3 bands having HH, HV and avg of both
X_band_1=np.array([np.array(band).astype(np.float32).reshape(75, 75) for band in tra
X_band_2=np.array([np.array(band).astype(np.float32).reshape(75, 75) for band in tra
X_train = np.concatenate([X_band_1[:, :, :, np.newaxis],
                          X_band_2[:, :, :, np.newaxis],
                          ((X_band_1+X_band_2)/2)[:, :, :, np.newaxis]], axis=-1)

# Feature scaling doesn't make better results
# X_train /= 255

### Plotting 3D iceberg and ship example

py.init_notebook_mode(connected=True)
def plotmy3d(c, name):

    data = [
        go.Surface(
            z=c
        )
    ]
    layout = go.Layout(
        title=name,
        autosize=False,
        width=700,
        height=700,
        margin=dict(
            l=65,
            r=50,
            b=65,
            t=90
        )
    )
    fig = go.Figure(data=data, layout=layout)
    py.iplot(fig)
plotmy3d(X_band_1[12,:,:], 'Iceberg')
```

```
plotmy3d(X_band_1[14,:,:], 'Ship')

### Plotting 2D iceberg and ship example using band1 and band2 feature

print('Is iceberg id12 = ', train['is_iceberg'][12])
print('Is iceberg id14 = ', train['is_iceberg'][14])

fig, (ax1, ax2) = plt.subplots(1,2, figsize = (12, 6))
ax1.matshow(X_band_1[12,:,:])
ax1.set_title('Iceberg')
ax2.matshow(X_band_1[14,:,:])
ax2.set_title('Ship')


fig, (ax1, ax2) = plt.subplots(1,2, figsize = (12, 6))
ax1.matshow(X_band_2[12,:,:])
ax1.set_title('Iceberg')
ax2.matshow(X_band_2[14,:,:])
ax2.set_title('Ship')

### Splitting data

target_train=train['is_iceberg']
X_train_cv, X_valid, y_train_cv, y_valid = train_test_split(X_train, target_train, r

### Setting constant parameters

batch_size = 24
epochs = 50

# initiate optimizers
opt_sgd = keras.optimizers.SGD()
opt_adam = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08,

### Training model

# First version of model with SGD optimizer

convNN = Sequential()
convNN.add(Conv2D(64, kernel_size=(3, 3),  \
                  input_shape=(75, 75, 3), \
                  activation='relu'))
```

```
convNN.add(Conv2D(64, kernel_size=(3, 3), strides = (2, 2), activation='relu'))
convNN.add(MaxPooling2D(pool_size=(2, 2)))

convNN.add(Flatten())

convNN.add(Dense(30))
convNN.add(Activation('relu'))

convNN.add(Dense(1))
convNN.add(Activation('sigmoid'))

convNN.summary()

convNN.compile(loss='binary_crossentropy',
                  optimizer=opt_sgd,
                  metrics=['accuracy'])

history = convNN.fit(X_train_cv, y_train_cv, batch_size=batch_size, epochs=epochs,
                          validation_data=(X_valid, y_valid), shuffle=True,
                          verbose=1)

# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.grid()
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.grid()
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# First version of model with Adam optimizer

convNN = Sequential()
```

```
# CNN 1
convNN.add(Conv2D(64, kernel_size=(3, 3),  \
                  input_shape=(75, 75, 3), \
                  activation='relu'))
# CNN 2
convNN.add(Conv2D(64, kernel_size=(3, 3), strides = (2, 2), activation='relu'))
convNN.add(MaxPooling2D(pool_size=(2, 2)))

convNN.add(Flatten())

convNN.add(Dense(30))
convNN.add(Activation('relu'))

convNN.add(Dense(1))
convNN.add(Activation('sigmoid'))

convNN.summary()

convNN.compile(loss='binary_crossentropy',
                  optimizer=opt_adam,
                  metrics=['accuracy'])

history = convNN.fit(X_train_cv, y_train_cv, batch_size=batch_size, epochs=epochs,
                         validation_data=(X_valid, y_valid), shuffle=True,
                         verbose=1)

# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.grid()
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.grid()
plt.xlabel('epoch')
```

```
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# Second version of model with Adam optimizer

convNN = Sequential()

# CNN 1
convNN.add(Conv2D(64, kernel_size=(3, 3),  \
                  input_shape=(75, 75, 3), \
                  activation='relu'))
convNN.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
convNN.add(Dropout(0.2))

# CNN 2
convNN.add(Conv2D(64, kernel_size=(3, 3), strides = (2, 2), activation='relu'))
convNN.add(MaxPooling2D(pool_size=(2, 2)))
convNN.add(Dropout(0.2))

convNN.add(Flatten())

convNN.add(Dense(30))
convNN.add(Activation('relu'))
convNN.add(Dropout(0.2))

convNN.add(Dense(1))
convNN.add(Activation('sigmoid'))

convNN.summary()

convNN.compile(loss='binary_crossentropy',
                  optimizer=opt_adam,
                  metrics=['accuracy'])

history = convNN.fit(X_train_cv, y_train_cv, batch_size=batch_size, epochs=epochs,
                         validation_data=(X_valid, y_valid), shuffle=True,
                         verbose=1)

# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
```

```
plt.grid()
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.grid()
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()


# Second version of model with Adam optimizer with (7,7) kernel_size

convNN = Sequential()

# CNN 1
convNN.add(Conv2D(64, kernel_size=(7, 7),  \
                  input_shape=(75, 75, 3), \
                  activation='relu'))
convNN.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
convNN.add(Dropout(0.2))

# CNN 2
convNN.add(Conv2D(64, kernel_size=(3, 3), strides = (2, 2), activation='relu'))
convNN.add(MaxPooling2D(pool_size=(2, 2)))
convNN.add(Dropout(0.2))

convNN.add(Flatten())

convNN.add(Dense(30))
convNN.add(Activation('relu'))
convNN.add(Dropout(0.2))

convNN.add(Dense(1))
convNN.add(Activation('sigmoid'))

convNN.summary()

convNN.compile(loss='binary_crossentropy',
                  optimizer=opt_adam,
```

```
                  metrics=['accuracy'])

history = convNN.fit(X_train_cv, y_train_cv, batch_size=batch_size, epochs=epochs,
                          validation_data=(X_valid, y_valid), shuffle=True,
                          verbose=1)

# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.grid()
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.grid()
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# Second version of model with Adam optimizer and normalized data
X_nd = X_train
X_nd /= 255
X_train_nd, X_valid_nd, y_train_nd, y_valid_nd = train_test_split(X_nd, target_train

convNN = Sequential()

# CNN 1
convNN.add(Conv2D(64, kernel_size=(3, 3),  \
                  input_shape=(75, 75, 3), \
                  activation='relu'))
convNN.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
convNN.add(Dropout(0.2))

# CNN 2
convNN.add(Conv2D(64, kernel_size=(3, 3), strides = (2, 2), activation='relu'))
convNN.add(MaxPooling2D(pool_size=(2, 2)))
convNN.add(Dropout(0.2))
```

```
convNN.add(Flatten())

convNN.add(Dense(30))
convNN.add(Activation('relu'))
convNN.add(Dropout(0.2))

convNN.add(Dense(1))
convNN.add(Activation('sigmoid'))

convNN.summary()

convNN.compile(loss='binary_crossentropy',
                  optimizer=opt_adam,
                  metrics=['accuracy'])

history = convNN.fit(X_train_nd, y_train_nd, batch_size=batch_size, epochs=epochs,
                         validation_data=(X_valid_nd, y_valid_nd), shuffle=True,
                         verbose=1)

# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.grid()
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.grid()
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# Third version of model with Adam optimizer

convNN = Sequential()
```

```
# CNN 1
convNN.add(Conv2D(64, kernel_size=(3, 3),  \
                  input_shape=(75, 75, 3), \
                  activation='relu'))
convNN.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
convNN.add(Dropout(0.2))

# CNN 2
convNN.add(Conv2D(128, kernel_size=(3, 3), strides = (2, 2), activation='relu'))
convNN.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
convNN.add(Dropout(0.2))

# CNN 3
convNN.add(Conv2D(64, kernel_size=(3, 3), strides = (2, 2), activation='relu'))
convNN.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
convNN.add(Dropout(0.2))

# Flatten the data for the dense layers
convNN.add(Flatten())

# Dense 1
convNN.add(Dense(256))
convNN.add(Activation('relu'))
convNN.add(Dropout(0.2))

# Dense 2
convNN.add(Dense(128))
convNN.add(Activation('relu'))
convNN.add(Dropout(0.2))

# Output
convNN.add(Dense(1))
convNN.add(Activation('sigmoid'))

convNN.summary()

convNN.compile(loss='binary_crossentropy',
                  optimizer=opt_adam,
                  metrics=['accuracy'])

history = convNN.fit(X_train_cv, y_train_cv, batch_size=batch_size, epochs=epochs,
                         validation_data=(X_valid, y_valid), shuffle=True,
                         verbose=1)
```

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.grid()
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.grid()
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()


### StratifiedKFold

X = X_train
Y = target_train

#epochs = 50

kfold = StratifiedKFold(n_splits=4, shuffle=True, random_state=10)
cvscores = []

for train, test in kfold.split(X, Y):
    convNN = Sequential()

    # CNN 1
    convNN.add(Conv2D(64, kernel_size=(3, 3),  \
                      input_shape=(75, 75, 3), \
                      activation='relu'))
    convNN.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
    convNN.add(Dropout(0.2))

    # CNN 2
    convNN.add(Conv2D(64, kernel_size=(3, 3), strides = (2, 2), activation='relu'))
    convNN.add(MaxPooling2D(pool_size=(2, 2)))
```

```
    convNN.add(Dropout(0.2))

    convNN.add(Flatten())

    convNN.add(Dense(30))
    convNN.add(Activation('relu'))
    convNN.add(Dropout(0.2))

    convNN.add(Dense(1))
    convNN.add(Activation('sigmoid'))

    convNN.summary()

    convNN.compile(loss='binary_crossentropy',
                   optimizer=opt_adam,
                   metrics=['accuracy'])

    history = convNN.fit(X[train], Y[train], batch_size=batch_size, epochs=epochs,
                         shuffle=True, verbose=1)
    convNN.fit(X[train], Y[train], batch_size=batch_size, epochs=epochs, shuffle=Tru
    verbose=1)

    scores = convNN.evaluate(X[test], Y[test], verbose=0)
    cvscores.append(scores)

print(cvscores)

accuracy_per_fold = []
for i in range(4):
    accuracy_per_fold.append(cvscores[i][1])

plt.plot(accuracy_per_fold, 'bo')
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.grid()
plt.title('Accuracy per fold')

print(accuracy_per_fold)
```