

# Aleksy Stocki

## Projekt “Ptaki z Bioseco”

### Spis treści:

1. Baza danych
2. Preprocessing
3. Klasyfikacja
4. Wyniki

### Wstępny opis projektu

Jako pierwszy projekt z przedmiotu Inteligencja Obliczeniowa wybrałem “Ptaki z Bioseco”. Wybrałem ten projekt, ponieważ dane oraz technologia potrzebna do ich interpretacji ma na farmach wiatrowych ważne i realne zastosowanie, co sprawia, że nie oddziela mnie od niego warstwa abstrakcji i braku użyteczności.

Postanowiłem podzielić projekt na dwa modele sieci neuronowej:

- Model rozróżniający zdjęcia chmur, od zdjęć obiektów
- Model rozróżniający zdjęcia samolotów i ptaków

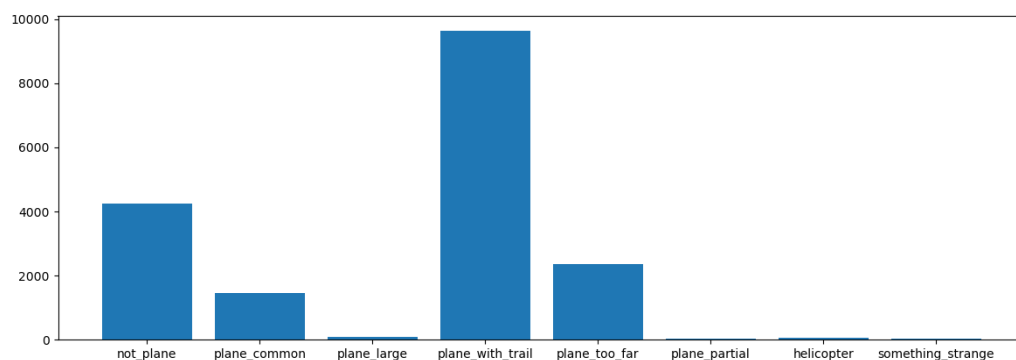
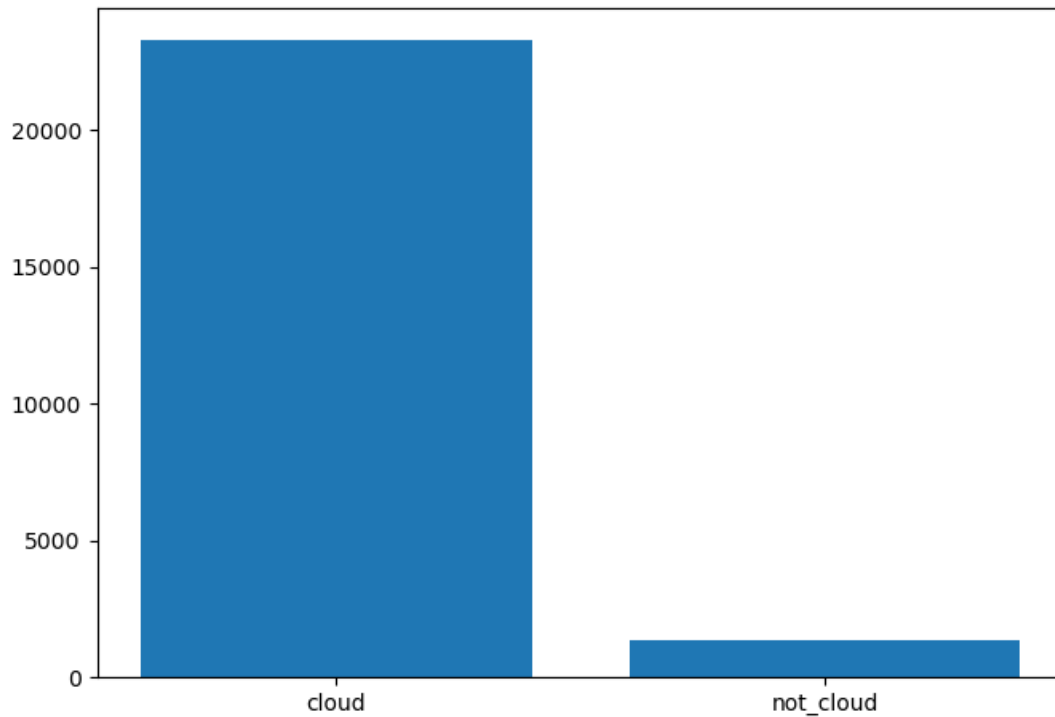
W tym czasie napotkałem wiele przeszkód związanych z procesowaniem danych i uczeniem sieci, które stopniowo udawało mi się pokonywać.

Podczas pracy nad tym projektem zdobyłem wiele dodatkowych i cennych informacji na temat tworzenia tego typu sieci, przetestowałem swoje umiejętności i wyćwiczyłem intuicję oraz poziom zrozumienia potrzebny do rozwijania się w tym dziale nauki.

# Baza Danych

Baza danych ze zdjęciami nieba była oznaczana z pomocą pozostałych studentów i miała kilka zasadniczych wad, które wymagały korekty.

Największym problemem w zbiorze zdjęć była wielka dysproporcja między klasami:



Wysokie i minimalnie niskie słupki doskonale oddają skalę problemu jaki stwarza taka dysproporcja liczbowa w ilości danych do nauki modelu.

Drugim, w mojej subiektywnej skali, zagrożeniem w zbiorze danych był wpływ moich kolegów i koleżanek z roku.

Dla jednego zdjęcia w zbiorze pojawiały się różne oznaczenia, często w tej samej ilości, przez co nie można było wybrać przeważającej odpowiedzi.

file	not_plane	plane_common	plane_large	plane_with_trail	plane_too_far	plane_partial	helicopter	something_strange
0.jpg	0	0	0	1	0	0	0	0
1.jpg	1	0	0	1	0	0	0	0
2.jpg	0	0	0	2	0	0	0	0
3.jpg	0	0	0	2	0	0	0	0
4.jpg	0	0	0	1	0	0	0	0
5.jpg	0	0	0	2	0	0	0	0
6.jpg	0	0	0	2	0	0	0	0
7.jpg	0	0	0	1	0	0	0	0
8.jpg	0	0	0	3	0	0	0	0
9.jpg	0	0	0	3	0	0	0	0
10.jpg	0	0	0	1	0	0	0	0
11.jpg	0	0	0	1	1	0	0	0
12.jpg	0	0	0	2	0	0	0	0
13.jpg	0	0	0	1	1	0	0	0
14.jpg	0	0	0	0	1	0	0	0
15.jpg	0	0	0	0	0	0	2	0
16.jpg	0	0	0	2	0	0	0	0
17.jpg	0	0	0	2	0	0	0	0
18.jpg	0	0	0	2	0	0	0	0
19.jpg	0	0	0	0	1	0	0	0
20.jpg	2	0	0	0	0	0	0	0
21.jpg	2	0	0	0	0	0	0	0
22.jpg	0	0	0	1	0	0	0	0
23.jpg	0	0	0	2	0	0	0	0
24.jpg	0	0	0	0	2	0	0	0
25.jpg	0	0	0	2	0	0	0	0
26.jpg	1	0	0	0	0	0	0	1
27.jpg	0	0	0	2	0	0	0	0
28.jpg	0	0	0	2	0	0	0	0
29.jpg	0	0	0	2	0	0	0	0
30.jpg	2	0	0	0	0	0	0	0

Na powyższym zdjęciu tabeli oznaczeń, pliki 13.jpg i 11.jpg są doskonałym przykładem zaistniałego problemu. Przez to, że takie oznaczenia rozkładały się procentowo jako 50% i 50%, ustawienie thresholdów na to, jakie oznaczenie wybrać, miałyby sens tylko w przypadku gdy takie oznaczenia obliczane byłyby z jakąś wagą, np. Oznaczenie plane\_with\_trail byłoby ważniejsze niż plane\_too\_far.

# Preprocessing

Moja idea preprocessingu dla tego zbioru danych była stosunkowo prosta, mianowicie:

1. Zebrałem wszystkie dane do jednego katalogu.
2. Pozbyłem się zepsutych plików.
3. Obliczyłem statystykę oznaczeń dla każdego zdjęcia.
4. Przeniósłem zdjęcia do oznaczonych katalogów gotowych na generowanie datasetów.

Natomiast pomiędzy zestawami chmur i samolotów, musiałem obrać inne podejścia.

## Chmury

Dla zbioru oznaczeń chmur, procesowanie danych było stosunkowo łatwe.

Najpierw z wypakowanych katalogów z oznaczeniami przeniósłem wszystkie zdjęcia do jednego katalogu (z powtórzeniami), a następnie dla powtarzających się zdjęć zliczyłem ich oznaczenia. W tym przypadku pozwoliłem sobie na swobodę, całkowicie odrzucając zdjęcia dla których oznaczenia mogłyby być niejednoznaczne.

```
30 cloud_array = np.empty([0, 3])
29 print(cloud_array)
28 for file in clouds_files:
27     file_attributes = file.split(".")
26     filename = file
25     label = file_attributes[2]
24     cloud = 0
23     not_cloud = 0
22     if label == "cloud":
21         | cloud = 1
20     elif label == "not_cloud":
19         | not_cloud = 1
18     check = check_for_element_cloud(cloud_array, filename, cloud, not_cloud)
17     if not check:
16         | cloud_array = np.append(cloud_array, [[filename, cloud, not_cloud]], axis=0)
15
14 for arr in cloud_array:
13     file = arr[0].split(".")
12     filename = f"{file[0]}.{file[1]}"
11     if arr[1] > arr[2]:
10         | shutil.copyfile(f"../all_clouds/{arr[0]}", f"../labeled_clouds/{filename}.cloud.{file[-1]}")
9     elif arr[2] > arr[1]:
8         | shutil.copyfile(f"../all_clouds/{arr[0]}", f"../labeled_clouds/{filename}.not_cloud.{file[-1]}")
7     else:
6         | continue
```

Największym problemem, który zwizualizowałem wcześniej, była przepaść między zdjęciami oznaczonymi jako chmury, a zdjęciami nie-chmur.

Żeby zapobiec powstawaniu problemów przy nauce modelu, zwyczajnie przeniósłem wszystkie zdjęcia ze zbioru samolotów i oznaczyłem je jako not\_cloud.

## Samoloty

Zbiór zdjęć samolotów, w porównaniu do zdjęć chmur, okazał się być znacznie trudniejszy w procesowaniu.

Pierwsze podjęte kroki były analogiczne do tych, które podjąłem w przypadku zbioru chmur z jedną różnicą, mianowicie, statystyki oznaczeń zapisywałem do pliku csv.

```
8 df = pd.DataFrame(columns=["file",  
7                             "not_plane",  
6                             "plane_common",  
5                             "plane_large",  
4                             "plane_with_trail",  
3                             "plane_too_far",  
2                             "plane_partial",  
1                             "helicopter",  
74                             "something_strange"])  
1 ind = 0  
2 for arr in planes_array:  
3     print(arr)  
4     file = arr[0].split(".")  
5     filename = f"{file[0]}.{file[1]}"  
6     series = [f"{ind}.jpg",  
7               arr[1],  
8               arr[2],  
9               arr[3],  
10              arr[4],  
11              arr[5],  
12              arr[6],  
13              arr[7],  
14              arr[8]]  
15     df.loc[len(df.index)] = series  
16     shutil.copyfile(  
17         f"..\\all_planes/{arr[0]}", f"..\\labeled_planes/{ind}.{file[-1]}")  
18     ind += 1  
19 print(df)  
20 df.to_csv("../planes.csv", sep=",")
```

Kod sprawdzający, czy plik ze zdjęciem jest zepsuty:

```
25 from struct import unpack
24 from tqdm import tqdm
23 import os
22
21
20 marker_mapping = {
19     0xffd8: "Start of Image",
18     0xffe0: "Application Default Header",
17     0xffdb: "Quantization Table",
16     0xffc0: "Start of Frame",
15     0xffc4: "Define Huffman Table",
14     0xffda: "Start of Scan",
13     0xffd9: "End of Image"
12 }
11
10
9 class JPEG:
8     def __init__(self, image_file):
7         with open(image_file, 'rb') as f:
6             self.img_data = f.read()
5
4     def decode(self):
3         data = self.img_data
2         while (True):
1             marker, = unpack(">H", data[0:2])
26             # print(marker_mapping.get(marker))
1             if marker == 0xffd8:
2                 data = data[2:]
3             elif marker == 0xffd9:
4                 return
5             elif marker == 0xffda:
6                 data = data[-2:]
7             else:
8                 lenchunk, = unpack(">H", data[2:4])
9                 data = data[2+lenchunk:]
10                if len(data) == 0:
11                    break
12
13
14 bads = []
15 images = ["../planes_data/"]
16 for path in images:
17     for folder in tqdm(os.listdir(path)):
18         print(folder)
19         for img in tqdm(os.listdir(f"{path}{folder}")):
20             image = f"{path}{folder}/{img}"
21             image = JPEG(image)
22             try:
23                 image.decode()
24             except: # E722 do not use bare 'except'
25                 bads.append(f"{path}{folder}/{img}")
26
27
28 for name in bads:
29     print(name)
```



Pomimo moich prób obejścia, generowanie datasetu wymagało ode mnie dyskretyzacji oznaczeń samolotów, więc ponownie zliczyłem oznaczenia dla zdjęć, licząc się z ewentualnymi dyskrepancjami między oznaczeniami. Pomędzy dodatkowym zmniejszeniem zbioru danych, a ewentualnymi drobnymi błędami, wybrałem drugą opcję jednocześnie całkowicie pozbywając się danych oznaczonych jako something\_strange.

```
4 def max2(arr):
5     max = arr[0]
6     index = 0
7     for i in range(len(arr)):
8         if arr[i] > max:
9             max = arr[i]
10            index = i
11    return max, index
```

```
19 for file in sorted(os.listdir("../labeled_planes/")):
18     row = label_csv.loc[label_csv["file"] == file]
17     row_labels = row.iloc[:, 2:10]
16     labels = row_labels.values[0]
15     labels_as_arr = [x for x in labels]
14     max, index = max2(labels_as_arr)
13     match index:
12         case 0:
11             shutil.copy(f"../labeled_planes/{file}", f"../planes_data/not_plane/{file}")
10             not_plane.append(file)
9             case 1:
8                 shutil.copy(f"../labeled_planes/{file}", f"../planes_data/plane_common/{file}")
7                 plane_common.append(file)
6                 case 2:
5                     shutil.copy(f"../labeled_planes/{file}", f"../planes_data/plane_large/{file}")
4                     plane_large.append(file)
3                     case 3:
2                         shutil.copy(f"../labeled_planes/{file}", f"../planes_data/plane_with_trail/{file}")
1                         plane_with_trail.append(file)
8                     case 4:
1                         shutil.copy(f"../labeled_planes/{file}", f"../planes_data/plane_too_far/{file}")
2                         plane_too_far.append(file)
3                     case 5:
4                         shutil.copy(f"../labeled_planes/{file}", f"../planes_data/plane_partial/{file}")
5                         plane_partial.append(file)
6                     case 6:
7                         shutil.copy(f"../labeled_planes/{file}", f"../planes_data/helicopter/{file}")
8                         helicopter.append(file)
9                     case 7:
10                         pass
```

Po "pomyślnej" próbie dyskretyzacji danych, postanowiłem wyrównać zbiory oznaczeń. Ze zbioru plane\_with\_trail usunąłem część zdjęć, a dla pozostałych zbiorów sztucznie powieliłem zdjęcia dokonując na nich rotacji. Na koniec tego procesu każde z siedmiu oznaczeń miało w swoim arsenale 4500 zdjęć.

```
3 while len(plane_with_trail) > 4500:
4     random_id = r.randint(0, len(plane_with_trail) - 1)
5     filename = plane_with_trail[random_id]
6     os.remove(f"../planes_data/plane_with_trail/{filename}")
7     plane_with_trail.remove(filename)
8
9 index = 0
10 while len(plane_common) < 4500:
11     index += 1
12     filename = plane_common[r.randint(0, len(plane_common) - 1)]
13     filename_index = filename.split(".")[0]
14     original = Image.open(f"../planes_data/plane_common/{filename}")
15     rotated = original.rotate(rotations_arr[r.randint(0, 2)])
16     saved = rotated.save(f"../planes_data/plane_common/{filename_index}_{index}.jpg")
17     plane_common.append(f"{filename_index}_{index}.jpg")
```



# Klasyfikacja

Klasyfikacja i uczenie modeli przebiegała dosyć burzliwie, ale finalnie osiągnąłem pożądane rezultaty, które były wystarczająco zadowalające.

## Augmentacja danych

Dla obu modeli wykorzystałem bardzo proste zasady manipulowania zdjęciami, tak, aby zachować ich realistyczność i naturę.

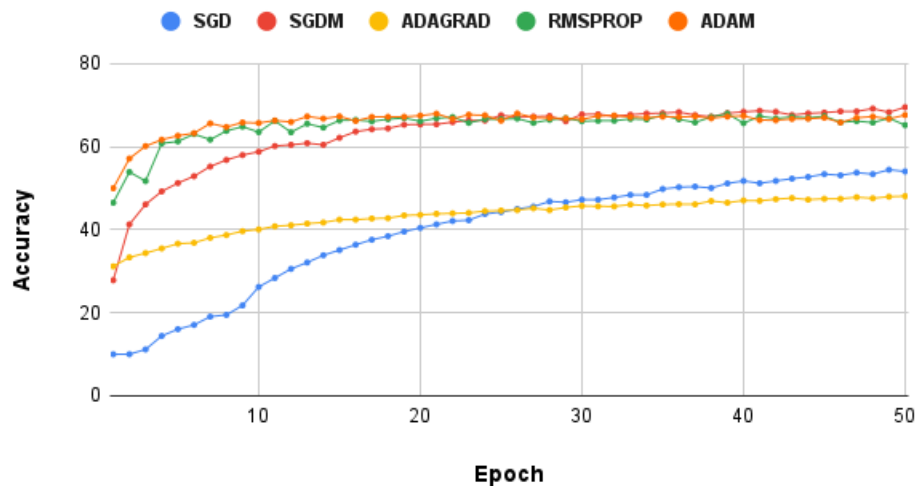
Funkcja augmentacji ma dwie warstwy, jedną jest odbicie zdjęcia w osi pionowej, a drugą, rotacja zdjęcia o drobną wartość. Pozwoliło mi to na dodatkowe poszerzenie zakresu danych, na których uczyły się modele.

```
14 data_augmentation_layers = [  
13     layers.RandomFlip("horizontal"),  
12     layers.RandomRotation(0.25)  
11 ]  
10  
9  
8 def data_augmentation(images):  
7     for layer in data_augmentation_layers:  
6         images = layer(images)  
5     return images
```

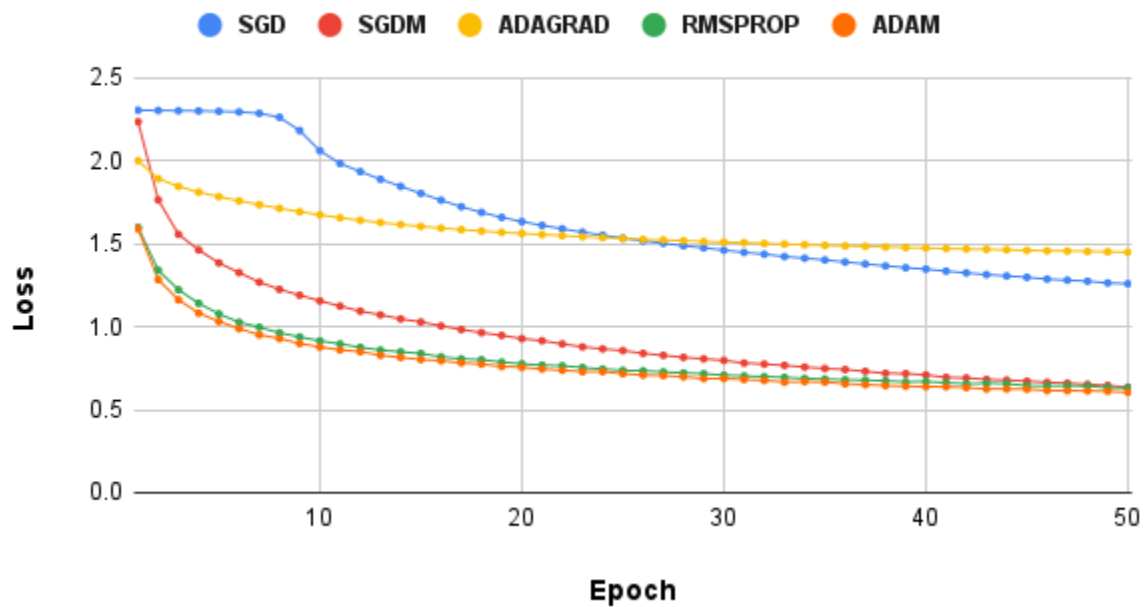
# Optymalizacja

Poszukując informacji na temat optymalizacji natknąłem się na ciekawe informacje odnośnie optymalizacji dla sieci neuronowych badających zdjęcia.

**LENET TEST ACCURACY**



**LENET TRAINING LOSS**



Źródło: <https://towardsai.net/p/l/impact-of-optimizers-in-image-classifiers>

Z informacji z powyższego i kilku innych podobnych artykułów i postów, doszedłem do wniosku, że najlepszym algorytmem optymalizacji do klasyfikacji zdjęć będzie ADAM.

W moim kodzie umieściłem również SGD. Po znalezieniu kilku postów o problemach, które występowały w moim przypadku, kilka razy pojawiła się wzmianka o wykorzystaniu SGD do troubleshootingu sieci. Nie jestem w stanie stwierdzić jak duży miało to wpływ na moją pracę, ale uważam, że lepiej ukazywało pewne założeń i połączenia między występującymi problemami oraz pozwalało mi szybciej przejść przez pierwsze epoki i dojść do sensownych wniosków.

```
12 optimizer = keras.optimizers.Adam(learning_rate=0.00001)
11 optimizer2 = keras.optimizers.SGD(learning_rate=0.01)
```

## Funkcje callback i loss, aktywacja

W przypadku obu modeli korzystałem z tych samych funkcji.

Za callbacki służyły mi ReduceLROnPlateau i ModelCheckpoint, a loss function to probabilistyczne BinaryCrossEntropy dla modelu “clouds” oraz CategoricalCrossEntropy dla modelu “planes”.

```
21 learning_rate_reduction = keras.callbacks.ReduceLROnPlateau(
20     monitor="val_loss",
19     patience=3,
18     verbose=1,
17     factor=0.005,
16     min_lr=0.000001
15 )
14 checkpoint_callback = keras.callbacks.ModelCheckpoint(
13     filepath="./new_data_model/{epoch:02d}_new_data.keras",
12     save_best_only=True,
11     monitor="val_accuracy",
10     mode='max',
9 )
8 callbacks = [learning_rate_reduction, checkpoint_callback]
7 loss = keras.losses.CategoricalCrossentropy(from_logits=True)
```

Przy modelu binarnej klasyfikacji chmur wykorzystałem funkcję aktywacji ReLU, zamieniającą skrajne wartości na zera i jedynki, a w modelu z klasyfikacją kategoriową postawiłem na Sigmoid, by mieć większy zakres wahań w oznaczeniach.

W obu modelach ostatnią warstwę wyjściową pozostawiłem bez funkcji aktywacji, ponieważ wtedy lepiej ukazuje “pewność” modelu przy jego wyborach oznaczeń.

# Modele sieci

## Chmury

W obu modelach zainspirowałem się strukturą kodu z poradnika na oficjalnej stronie Kerasa i dopasowałem warstwy i ich wymiary do swojego problemu.

```
20 inputs = keras.Input(shape=(100, 100, 3))
19 x = data_augmentation(inputs)
18 x = layers.Rescaling(1./255)(x)
17
16 x = layers.Conv2D(32, 3, strides=2, padding="same")(x)
15 x = layers.BatchNormalization()(x)
14 x = layers.Activation("relu")(x)
13
12 previous_block_activation = x # Set aside residual
11
10 for size in [32, 32, 48]:
9     x = layers.Activation("relu")(x)
8     x = layers.Conv2D(size, 3, padding="same")(x)
7     x = layers.BatchNormalization()(x)
6
5     x = layers.Activation("relu")(x)
4     x = layers.Conv2D(size, 3, padding="same")(x)
3     x = layers.BatchNormalization()(x)
2
1     x = layers.MaxPooling2D(3, strides=2, padding="same")(x)
58
1     # Project residual
2     residual = layers.Conv2D(size, 1, strides=2, padding="same")(
3         previous_block_activation
4     )
5     x = layers.add([x, residual]) # Add back residual
6     previous_block_activation = x # Set aside next residual
7
8 x = layers.Conv2D(48, 3, padding="same")(x)
9 x = layers.BatchNormalization()(x)
10 x = layers.Activation("relu")(x)
11
12 x = layers.GlobalAveragePooling2D()(x)
13 num_classes = 2
14 if num_classes == 2:
15     units = 1
16 else:
17     units = num_classes
18
19 x = layers.Dropout(0.25)(x)
20 # We specify activation=None so as to return logits
21 outputs = layers.Dense(units, activation=None)(x)
22
23
24 model = keras.Model(inputs, outputs)
```

W wielu przeprowadzonych testach, na większym modelu sieci neuronowej, głównym problemem był gigantyczny overfitting, sieć zaczynała już z wysoką celnością, a po paru epokach jej jakość drastycznie spadała i zaczynała ona się uczyć danych “na pamięć”.



W celu zwalczenia tego problemu eksperymentowałem z pomniejszeniem złożoności sieci oraz modyfikacją batch size do momentu, aż osiągnie wystarczająco dobry poziom generalizacji cech zdjęć. Końcowym “złotym środkiem” okazał się być rozmiar próbki 64 i widoczny wyżej model. Osiągnął na tyle dobre wyniki, że nie było potrzeby dodatkowo go douczyć.

## Samoloty

```
21 # Entry block
20 inputs = keras.Input(shape=(100, 100, 3))
19 x = data_augmentation(inputs)
18 x = layers.Rescaling(1./255)(x)
17
16 x = layers.Conv2D(32, 3, strides=2, padding="same")(x)
15 x = layers.BatchNormalization()(x)
14 x = layers.Activation("sigmoid")(x)
13
12 previous_block_activation = x # Set aside residual
11
10 for size in [32, 64, 128]:
9     x = layers.Activation("sigmoid")(x)
8     x = layers.Conv2D(size, 3, padding="same")(x)
7     x = layers.BatchNormalization()(x)
6
5     x = layers.Activation("sigmoid")(x)
4     x = layers.Conv2D(size, 3, padding="same")(x)
3     x = layers.BatchNormalization()(x)
2
1     x = layers.MaxPooling2D(3, strides=2, padding="same")(x)
0
1     # Project residual
2     residual = layers.Conv2D(size, 1, strides=2, padding="same")(
3         previous_block_activation
4     )
5     x = layers.add([x, residual]) # Add back residual
6     previous_block_activation = x # Set aside next residual
7
8 x = layers.Conv2D(128, 3, padding="same")(x)
9 x = layers.BatchNormalization()(x)
10 x = layers.Activation("sigmoid")(x)
11
12 x = layers.GlobalAveragePooling2D()(x)
13 num_classes = 7
14 if num_classes == 2:
15     units = 1
16 else:
17     units = num_classes
18
19 #x = layers.Dropout(0.25)(x) # E265 block comment should start
20 # We specify activation=None so as to return logits
21 outputs = layers.Dense(units, activation=None)(x)
22
23
24 model = keras.Model(inputs, outputs)
```



Model sieci rozróżniającej samoloty sprawiał podobne problemy przy trenowaniu, co poprzedni. Trenowanie tego modelu było znacznie ciekawszym wyzwaniem od poprzedniego.

Podjmując wszelkie próby nauczenia go jak najlepszej generalizacji, odrzuciłem warstwę Dropout, manipulowałem batch size, learning rate i ilość neuronów. Niestety sieć zatrzymała się w okolicach 80% celności na zbiorze testowym i żadna zmiana nie pomogła go z tego momentu wyciągnąć wyżej.

Sieć trenowałem w dwóch etapach:

- Pierwszy etap: learning rate 0.00001
- Drugi etap: learning rate 0.000001

Zmiana learning rate w drugim modelu była spowodowana tym, że funkcja callback w pierwszym etapie sama sprowadziła wartość do tej niższej. Przy próbie uczenia modelu na poprzedniej wartości, pierwsze epoki miały bardzo słabe wyniki i spowalniało to cały proces douczania.

Próbowiałem douczyć sieć po raz trzeci, ale wyniki były gorsze od tych z drugiego etapu.

# Wyniki

Dla modelu cloud, wartości od zera w dół oznaczają chmurę, a wartości od zera w górę, nie-chmurę.

Dla modelu planes, indeks najwyższej wartości w tablicy oznacza podaną klasyfikację:

0 - helicopter

1 - not\_plane

2 - plane\_common

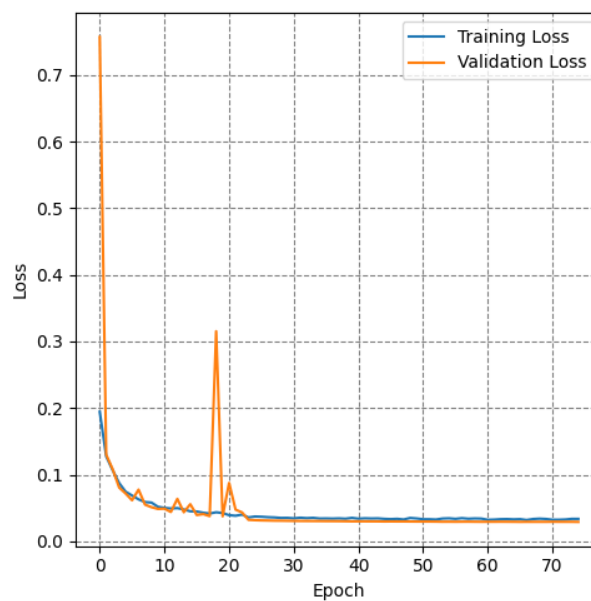
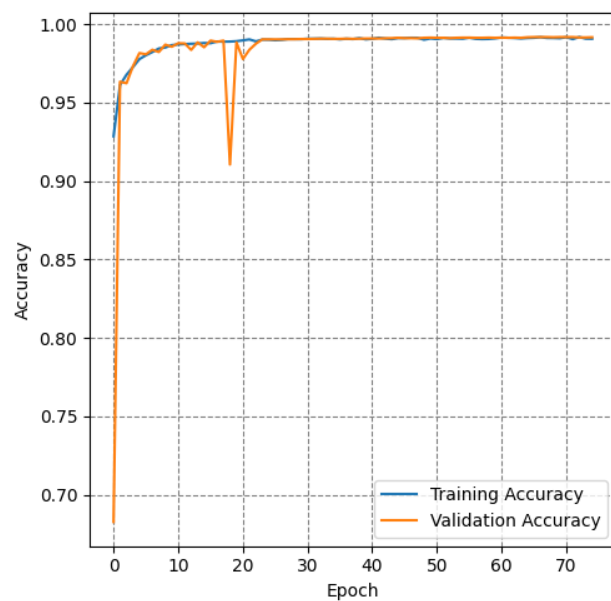
3 - plane\_large

4 - plane\_partial

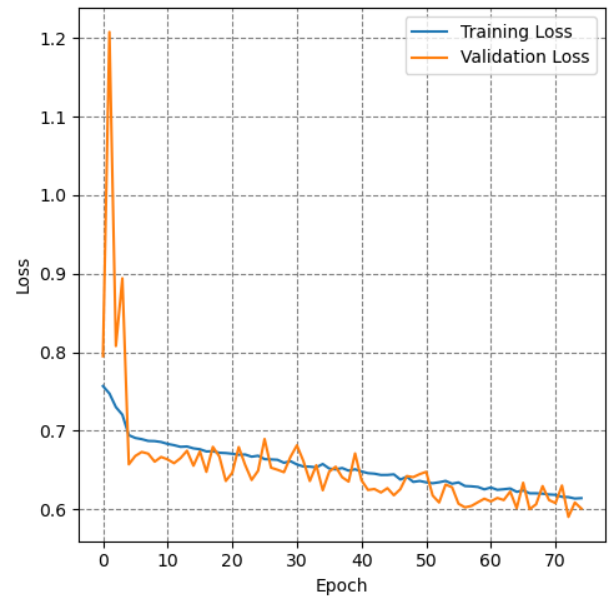
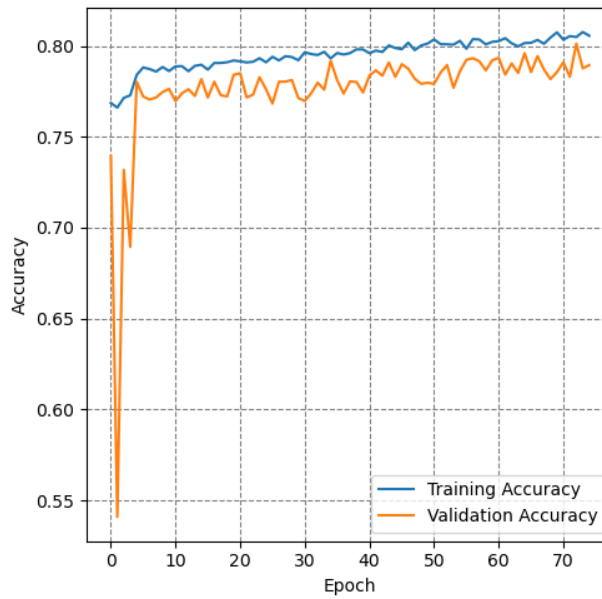
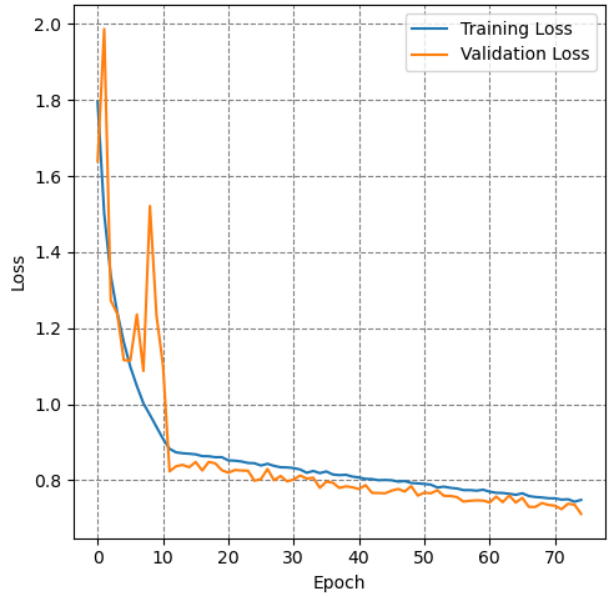
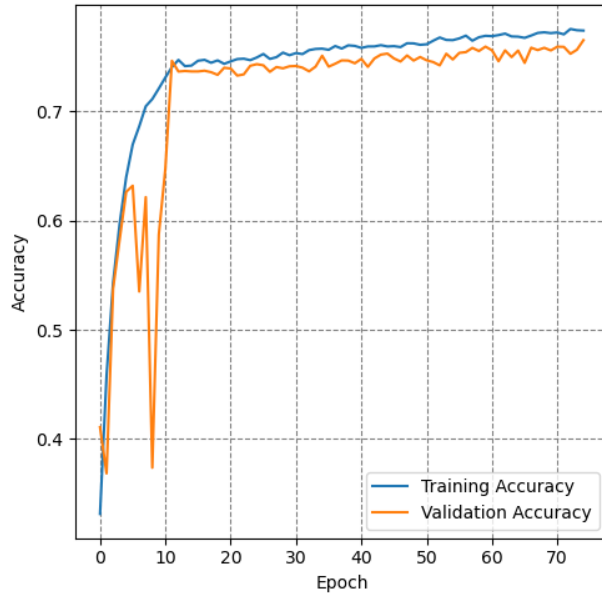
5 - plane\_too\_far

6 - plane\_with\_trail

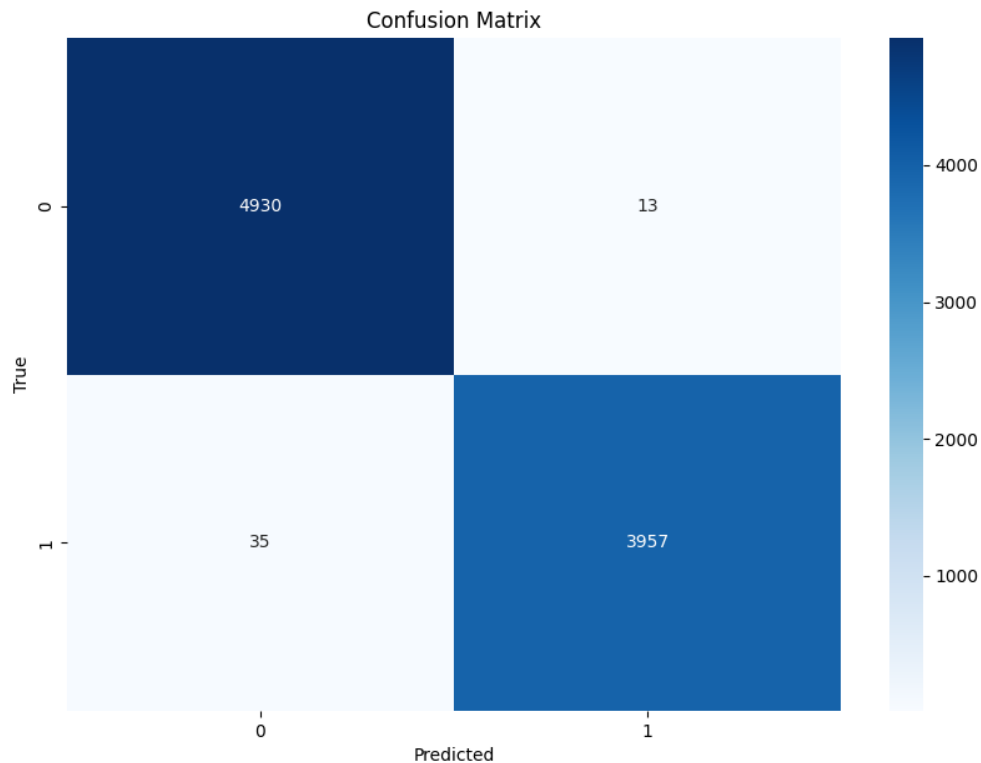
## Wykresy modelu cloud



## Wykresy modelu planes



## Macierze błędu



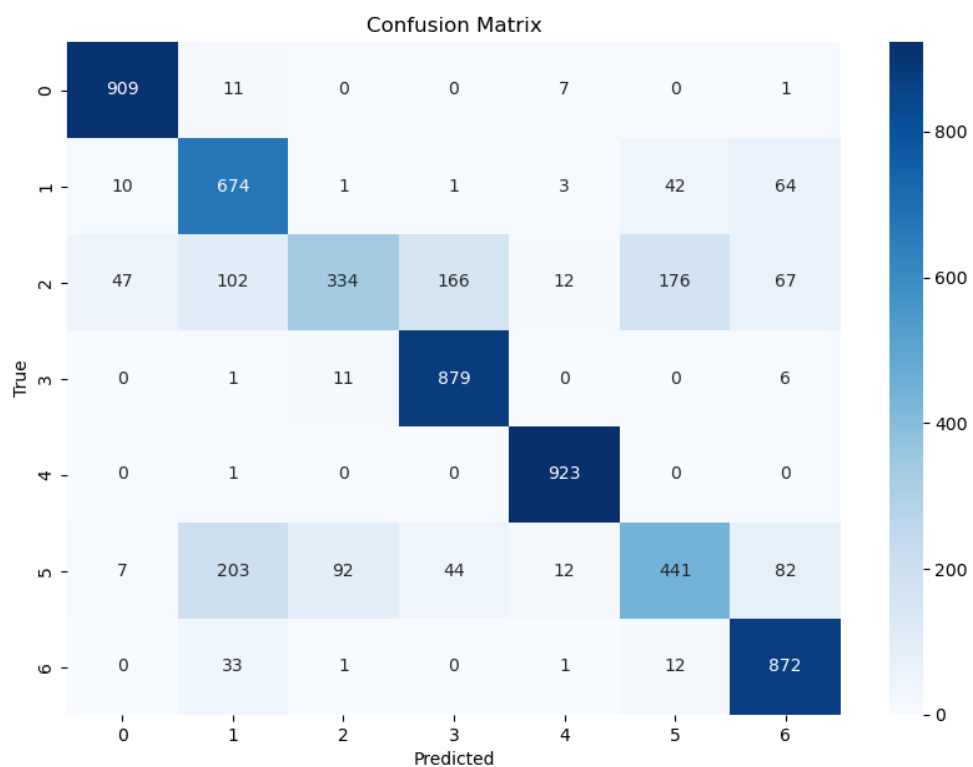
Celność modelu cloud wynosiła ~99%

```
37 # Confusion matrix
38 cm = confusion_matrix(labels, predict_labels)
39 goodsum = 0
40 wholesum = 0
41 goodidx = 0
42 for arr in cm:
43     goodsum += arr[goodidx]
44     goodidx += 1
45     for elem in arr:
46         wholesum += elem
47
48 print("Accuracy from confusion matrix: ", goodsum/wholesum)
```

PROBLEMS 2 DEBUG CONSOLE TERMINAL PORTS OUTPUT COMMENTS

bash - CloudVsNotCloud

Accuracy from confusion matrix: 0.9935370152761457





Celność modelu plane wynosiła ~80%

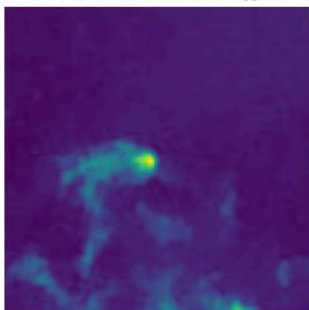
```
1 # Confusion matrix
2 cm = confusion_matrix(labels, predict_labels)
3 print(cm)
4 goodsum = 0
5 wholesum = 0
6 goodidx = 0
7 for arr in cm:
8     goodsum += arr[goodidx]
9     goodidx += 1
10    for elem in arr:
11        wholesum += elem
12
13 print("Accuracy from confusion matrix: ", goodsum/wholesum)
```

NORMAL ▶ main > ~10 -24 > ▲ 2 ▶ TestModel.py

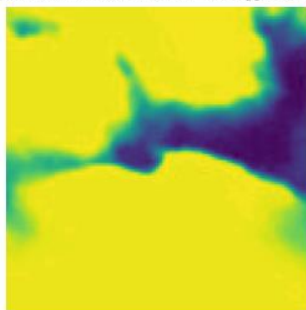
```
[[3760  39   3   0  23   0   6]
 [ 50 3053  23   2  17 183 277]
 [169 429 1475 713  69 649 303]
 [  0  27  48 3725   0   0  26]
 [  0  11   0   0 3831   0   0]
 [ 26 891 435 186  39 1870 364]
 [  0 175   9   0   3  67 3578]]
Accuracy from confusion matrix: 0.8018377645552459
```

Oznaczenie małej próbki zdjęć chmur

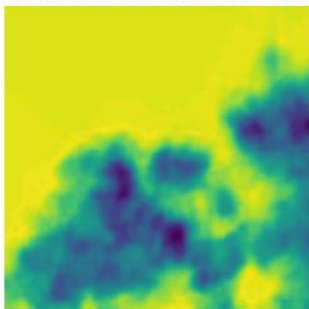
True label: 0, Predicted label: [[-3.996716]]



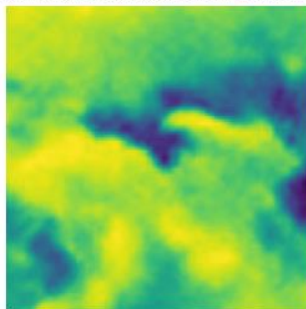
True label: 0, Predicted label: [[-8.172336]]



True label: 0, Predicted label: [[-3.8224797]]



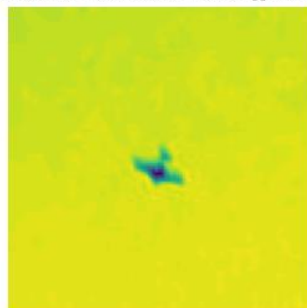
True label: 0, Predicted label: [[-9.651795]]



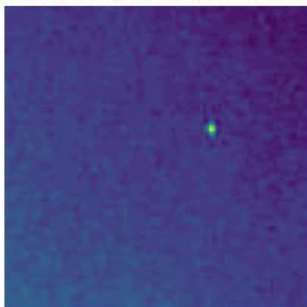
True label: 1, Predicted label: [[6.395374]]



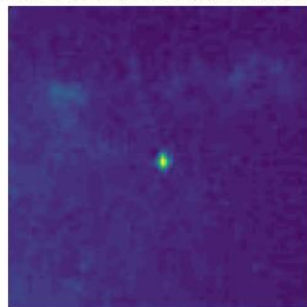
True label: 1, Predicted label: [[2.953097]]



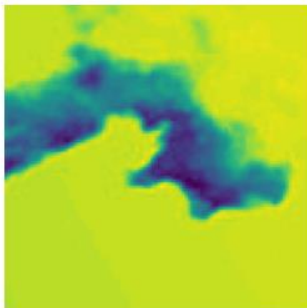
True label: 1, Predicted label: [[6.9782543]]



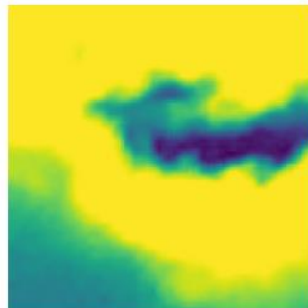
True label: 1, Predicted label: [[5.738247]]



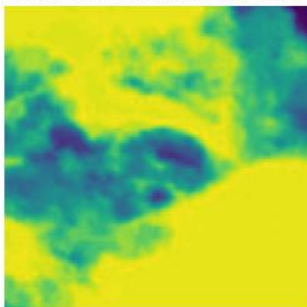
True label: 0, Predicted label:  $[-9.137786]$



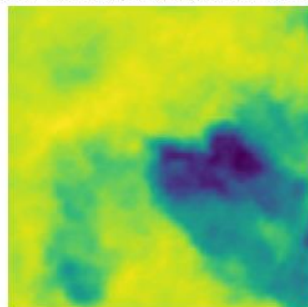
True label: 0, Predicted label:  $[-9.380572]$



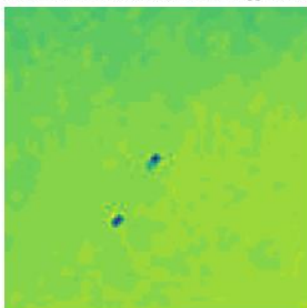
True label: 0, Predicted label:  $[-4.8250737]$



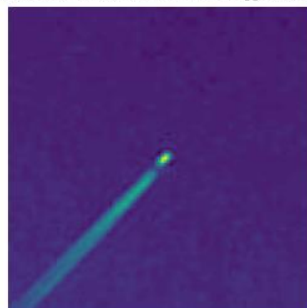
True label: 0, Predicted label:  $[-6.0961185]$



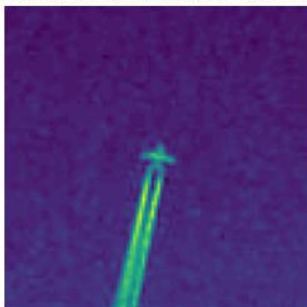
True label: 1, Predicted label: [[4.785901]]



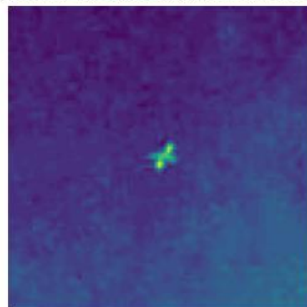
True label: 1, Predicted label: [[8.7201395]]



True label: 1, Predicted label: [[7.5583234]]



True label: 1, Predicted label: [[7.6808915]]



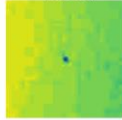
Oznaczenie małej próbki zdjęć samolotów



True label: helicopter,  
Predicted label: [[ 3.589195 -0.9678197 0.1954804 -0.33104768 -2.853521 -1.2543736  
-0.8636088 ]]



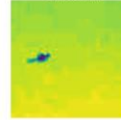
True label: helicopter,  
Predicted label: [[ 2.363579 0.16832751 -0.52276254 0.50117224 -1.6457098 -0.293026  
-2.7972758 ]]



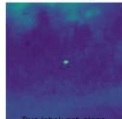
True label: helicopter,  
Predicted label: [[ 3.094601 -1.5469259 0.60222266 -0.48519453 -2.2563426 -0.819826  
-1.3012464 ]]



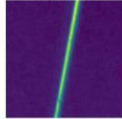
True label: helicopter,  
Predicted label: [[ 1.4553193 -1.0033092 -0.76391464 -0.43410084 0.11085218 -0.6630319  
-1.2105944 ]]



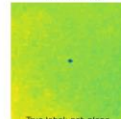
True label: not plane,  
Predicted label: [[ -1.6280715 2.1249099 0.38526115 -2.7883117 -2.6203148 1.6994605  
1.1449527 ]]



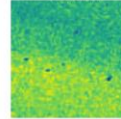
True label: not plane,  
Predicted label: [[ -1.1984447 2.8251224 0.00924805 -2.6229098 -1.4257983 -1.0009857  
1.7791681 ]]



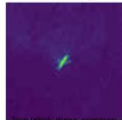
True label: not plane,  
Predicted label: [[ -1.0485699 3.6936994 -0.3908657 -2.3533883 -2.1703908 0.93865377  
0.20165208 ]]



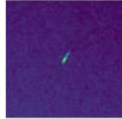
True label: not plane,  
Predicted label: [[ -2.3415902 3.2913015 -1.9562968 -0.41776347 -0.37290153 -1.7409433  
-2.5013444 ]]



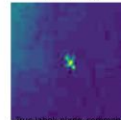
True label: plane common,  
Predicted label: [[ -1.4890596 -1.8207375 2.4252036 1.0391184 -1.8650204 0.2705264  
-1.0043223 ]]



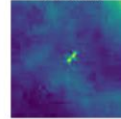
True label: plane common,  
Predicted label: [[ -2.355503 1.4444466 0.6870269 -1.754792 -2.5582576 1.592439  
1.2031771 ]]



True label: plane common,  
Predicted label: [[ -2.4615676 -1.7172413 -2.5338647 0.88327634 -1.8308332 1.3209369  
-1.2611483 ]]



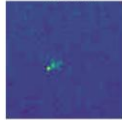
True label: plane common,  
Predicted label: [[ -1.615071 -1.2671894 0.8143229 0.097436 -0.99554616 0.07433464  
0.43664157 ]]



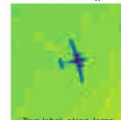
True label: plane large,  
Predicted label: [[ -0.2659021 -1.9236008 0.05002552 3.227797 -1.5093081 -1.0720512  
-0.9898198 ]]



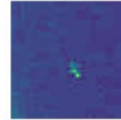
True label: plane large,  
Predicted label: [[ -0.9268974 -0.81872666 -0.2641698 2.0959911 -1.3601089 -0.86763126  
0.01272422 ]]



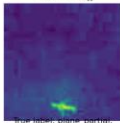
True label: plane large,  
Predicted label: [[ -0.4610931 -2.1619737 -0.10903685 3.9107683 -2.2317317 -1.8083148  
0.39068705 ]]



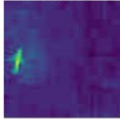
True label: plane large,  
Predicted label: [[ -0.90117365 -1.7339635 -0.0960081 2.8718102 -0.98579866 -1.2339592  
-0.41356146 ]]



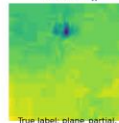
True label: plane partial.  
Predicted label: [[-1.7923412 -0.92543757 -0.30053684 -0.49060634 3.9569983 -1.0161523  
-1.8621442 ]]



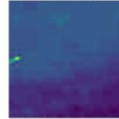
True label: plane partial.  
Predicted label: [[-1.6735551 -0.63635725 -0.53442055 -0.71385556 3.8562376 -1.0360693  
-1.7579803 ]]



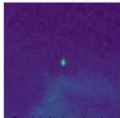
True label: plane partial.  
Predicted label: [[ 0.18741769 -0.75658906 -0.45526543 0.06701161 0.97151315 -1.179343  
-1.2513094 ]]



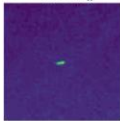
True label: plane partial.  
Predicted label: [[-1.2547337 -0.47013453 -0.50717615 -1.0901011 1.9787298 0.4625989  
-1.6109895 ]]



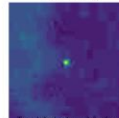
True label: plane too far.  
Predicted label: [[-2.0809293 1.5121368 0.5421531 2.632275 -2.0643015 1.6985513  
1.2262549 ]]



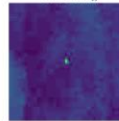
True label: plane too far.  
Predicted label: [[-2.8155224 1.9719305 0.57044923 -2.637564 -2.4070034 2.1061568  
1.4986725 ]]



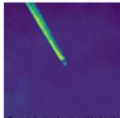
True label: plane too far.  
Predicted label: [[-1.9478343 -1.1167784 1.0759271 -0.6318053 -1.2005824 2.239102  
0.72706705 ]]



True label: plane too far.  
Predicted label: [[-1.1358355 0.16441174 0.30929646 -0.4475709 -1.4076967 1.2923105  
-0.7837049 ]]



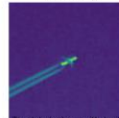
True label: plane with trail.  
Predicted label: [[-1.7356377 0.1164369 -0.07540061 -0.8003004 -2.5865033 -0.47556916  
3.2651842 ]]



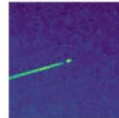
True label: plane with trail.  
Predicted label: [[-2.4758725 -0.06183327 1.0341737 -1.1473464 -2.578204 1.2627494  
1.792184 ]]



True label: plane with trail.  
Predicted label: [[-2.4831338 0.42795825 0.27886242 -0.9819678 -2.3671236 -0.8305579  
3.8853531 ]]



True label: plane with trail.  
Predicted label: [[-2.3860545 1.0961953 -0.8856146 -0.77968735 -2.0684187 -0.6673015  
3.7760038 ]]



# Wnioski

W przypadku obu modeli, na początku uczenia występowały dziwne i ekstremalne skoki, prawdopodobnie z powodu losowej inicjalizacji i różnicy między próbkami.

Model cloud świetnie nauczył się oznaczać oba typy zdjęć, dzięki nie poddanym funkcji aktywacji wynikom, świetnie widać z jaką pewnością model podejmuje decyzje.

Model planes pomimo trudności związanych ze słabym zbiorem danych, nauczył się zadowalająco, przy generowaniu próbek zdjęć, nawet przy swojej zawodności i wielu nieudanych oznaczeniach, udało mu się trafniej oznaczyć obiekt, niż wskazywały by na to wygenerowane przeze mnie oznaczenia.

Podsumowując, projekt i modele na pewno można jeszcze udoskonalić, choć uzyskane wyniki były zaskakująco dobre jak na to, z jakim doświadczeniem rozpocząłem pracę i jakie spotkałem po drodze problemy.

# Źródła

[https://keras.io/examples/vision/image\\_classification\\_from\\_scratch/](https://keras.io/examples/vision/image_classification_from_scratch/)

<https://towardsai.net/p/l/impact-of-optimizers-in-image-classifiers>

<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>

<https://medium.com/geekculture/how-to-use-matplotlib-to-visualise-an-image-from-a-dataset-in-python-1ae816bc78e0>

<https://yasoob.me/posts/understanding-and-writing-jpeg-decoder-in-python/>

Kolega z Francji – Chaton Ge Petê