Aleksy Stocki

Projekt “Ptaki z Bioseco”

Spis treści:

Wstępny opis projektu

Jako pierwszy projekt z przedmiotu Inteligencja Obliczeniowa wybralem “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

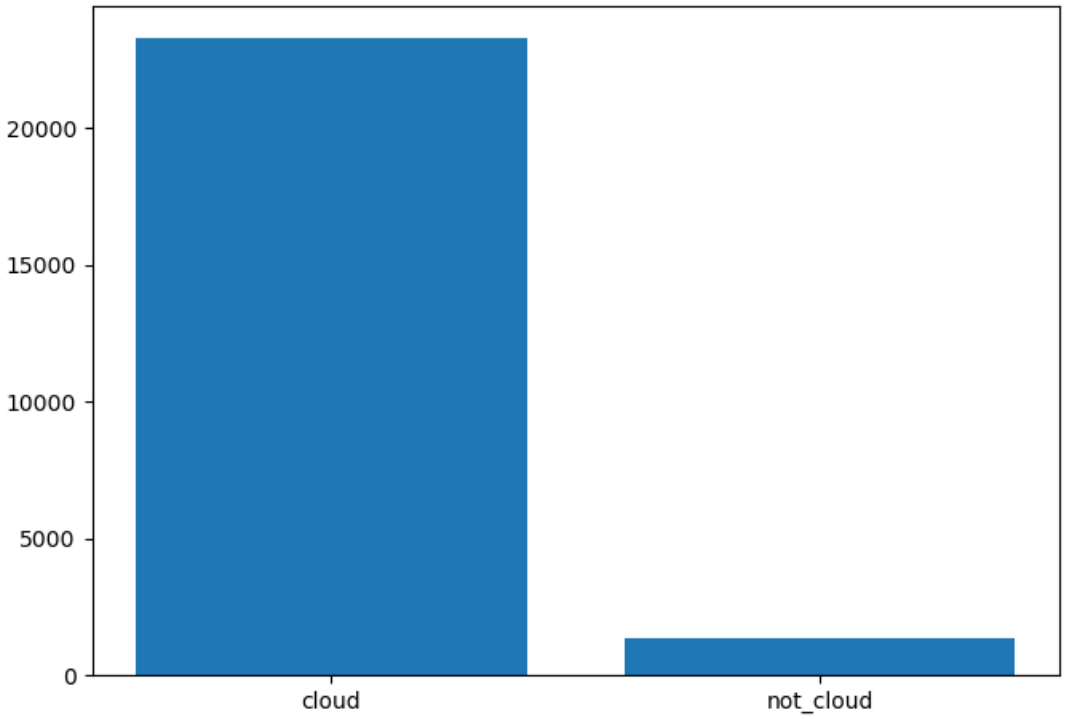
W tym czasie napotkałem wiele przeszkód związanych z procesowaniem danych i uczeniem sieci, które stopniowo udawało mi się pokonywać.

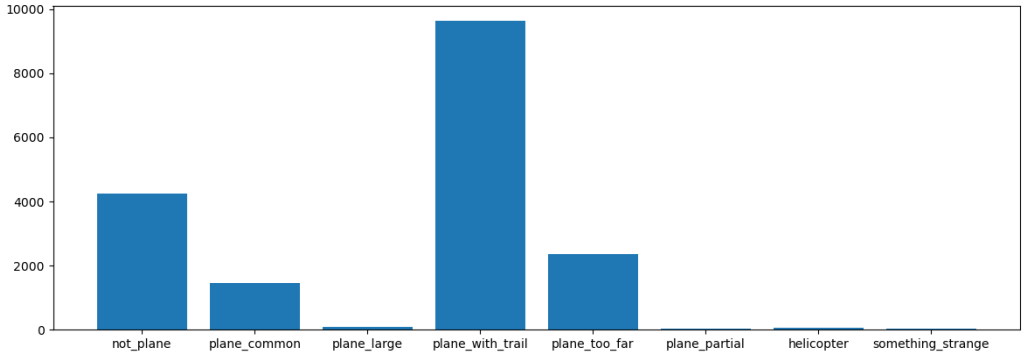
Podczas pracy nad tym projektem zdobyłem wiele dodatkowych i cennych informacji na temat tworzenia tego typu sieci, przetestowałem swoje umiejętności i wyćwiczyłem intuicję oraz poziom zrozumienia potrzebny do rozwijania się w tym dziale nauki.

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, miałoby 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. Przeniosłem 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 przeniosłem 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.

```
cloud_array = np.empty([0, 3])
print(cloud_array)
for file in clouds_files:
    file_attributes = file.split(".")
    filename = file
    label = file_attributes[2]
    cloud = 0
    not_cloud = 0
    if label == "cloud":
        cloud = 1
    elif label == "not_cloud":
        not_cloud = 1
    check = check_for_element_cloud(cloud_array, filename, cloud, not_cloud)
    if not check:
        cloud_array = np.append(cloud_array, [[filename, cloud, not_cloud]], axis=0)     E501 line too

for arr in cloud_array:
    file = arr[0].split(".")
    filename = f"{file[0]}.{file[1]}"
    if arr[1] > arr[2]:
        shutil.copyfile(f"../all_clouds/{arr[0]}", f"../labeled_clouds/{filename}.cloud.{file[-1]}")
    elif arr[2] > arr[1]:
        shutil.copyfile(f"../all_clouds/{arr[0]}", f"../labeled_clouds/{filename}.not_cloud.{file[-1]}")
    else:
        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 przeniosłem wszystkie zdjęcia ze zbioru samolotów i oznaczyłem je jako not_cloud.

Samoloty

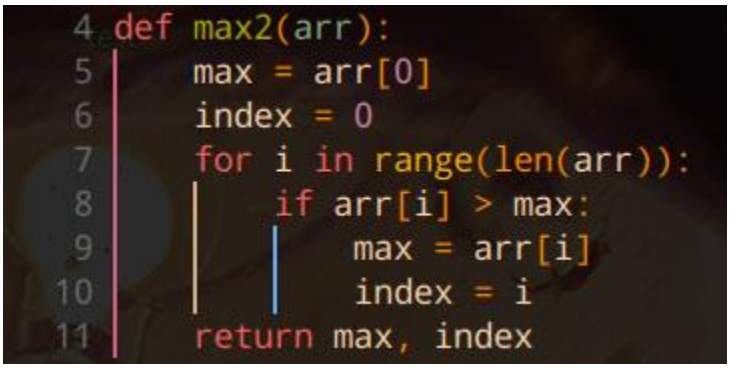
Zbiór zdjęć samolotów , w porównaniu do zdjęć chmur, okazał się być znacznie trudniejszy w procesowaniu.
Pierwsze podjęte kroki były analogiczne do tych, które podjąłem w przypadku zbioru chmur z jedną różnicą, mianowicie, statystyki oznaczeń zapisywałem do pliku csv.

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

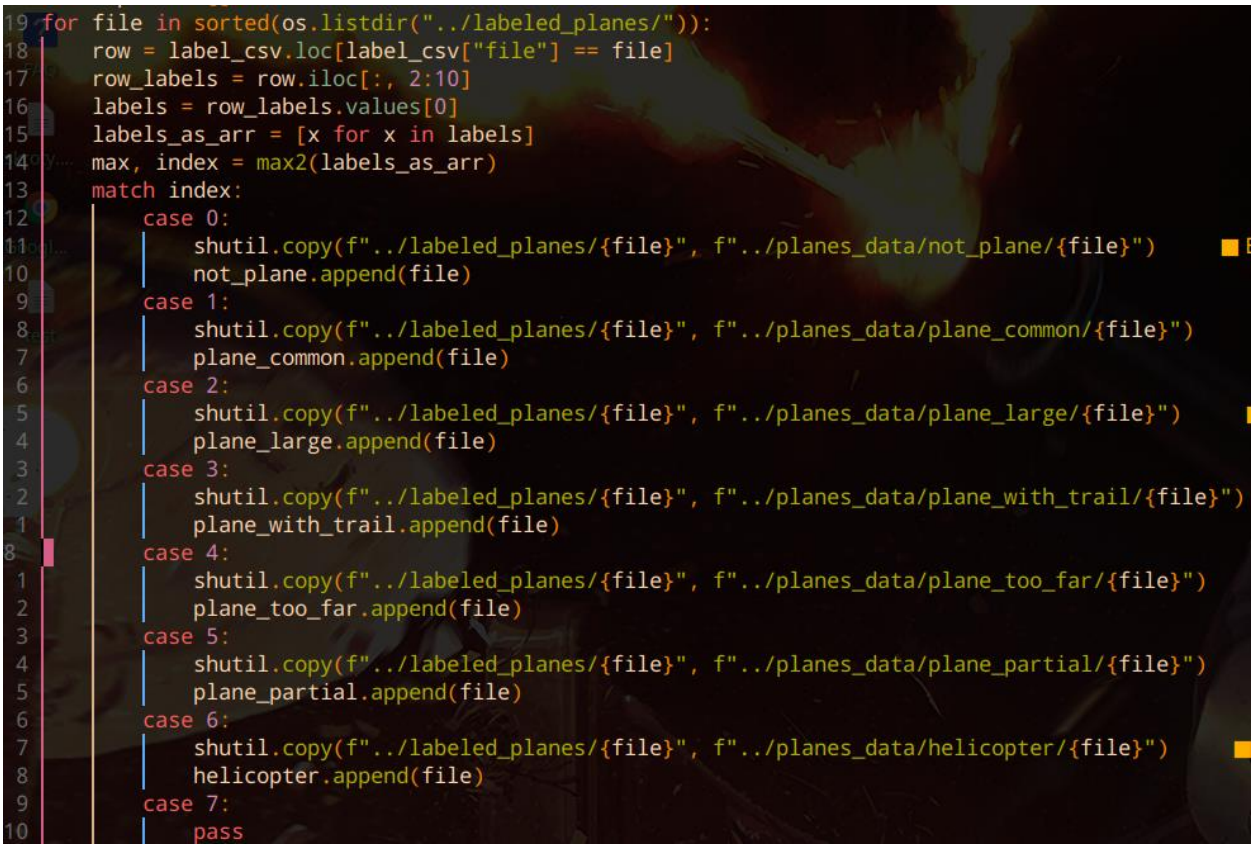
Kod sprawdzający, czy plik ze zdjęciem jest zepsuty:

```
from struct import unpack
from tqdm import tqdm
import os


marker_mapping = {
    0xffd8: "Start of Image",
    0xffe0: "Application Default Header",
    0xffdb: "Quantization Table",
    0xffc0: "Start of Frame",
    0xffc4: "Define Huffman Table",
    0xffda: "Start of Scan",
    0xffd9: "End of Image"
}


class JPEG:
    def __init__(self, image_file):
        with open(image_file, 'rb') as f:
            self.img_data = f.read()

    def decode(self):
        data = self.img_data
        while (True):
            marker, = unpack(">H", data[0:2])
            # print(marker_mapping.get(marker))
            if marker == 0xffd8:
                data = data[2:]
            elif marker == 0xffd9:
                return
            elif marker == 0xffda:
                data = data[-2:]
            else:
                lenchunk, = unpack(">H", data[2:4])
                data = data[2+lenchunk:]
            if len(data) == 0:
                break


bads = []
images = ["../planes_data/"]
for path in images:
    for folder in tqdm(os.listdir(path)):
        print(folder)
        for img in tqdm(os.listdir(f"{path}{folder}")):
            image = f"{path}{folder}/{img}"
            image = JPEG(image)
            try:
                image.decode()
            except:     E722 do not use bare 'except'
                bads.append(f"{path}{folder}/{img}")


for name in bads:
    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. Pomiędzy dodatkowym zmniejszeniem zbioru danych, a ewentualnymi drobnymi błędami, wybrałem drugą opcję jednocześnie całkowicie pozbywając się danych oznaczonych jako something_strange.

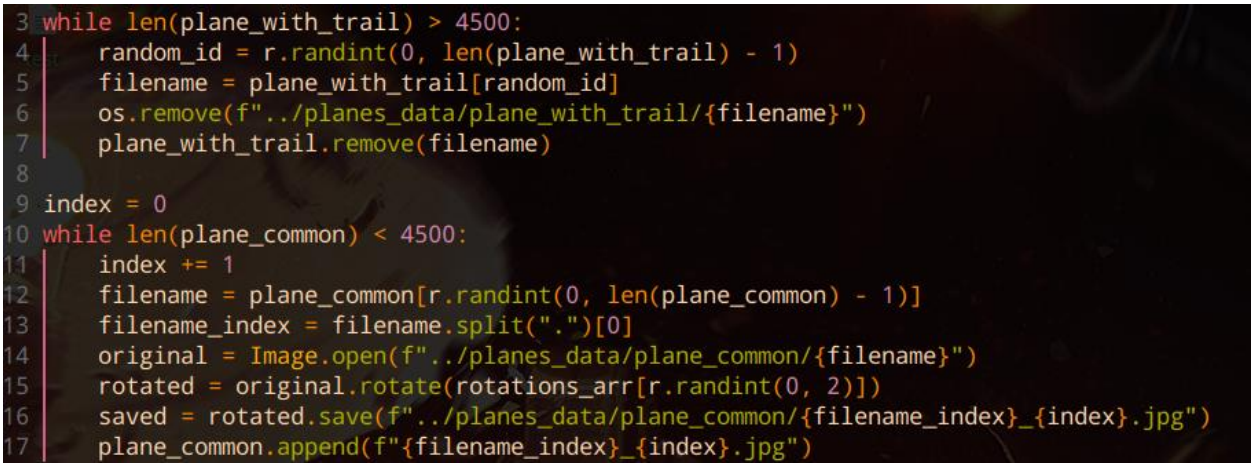


```
def max2(arr):
    max = arr[0]
    index = 0
    for i in range(len(arr)):
        if arr[i] > max:
            max = arr[i]
            index = i
    return max, index
```



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



```
while len(plane_with_trail) > 4500:
    random_id = r.randint(0, len(plane_with_trail) - 1)
    filename = plane_with_trail[random_id]
    os.remove(f"../planes_data/plane_with_trail/{filename}")
    plane_with_trail.remove(filename)

index = 0
while len(plane_common) < 4500:
    index += 1
    filename = plane_common[r.randint(0, len(plane_common) - 1)]
    filename_index = filename.split(".")[0]
    original = Image.open(f"../planes_data/plane_common/{filename}")
    rotated = original.rotate(rotations_arr[r.randint(0, 2)])
    saved = rotated.save(f"../planes_data/plane_common/{filename_index}_{index}.jpg")
    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.

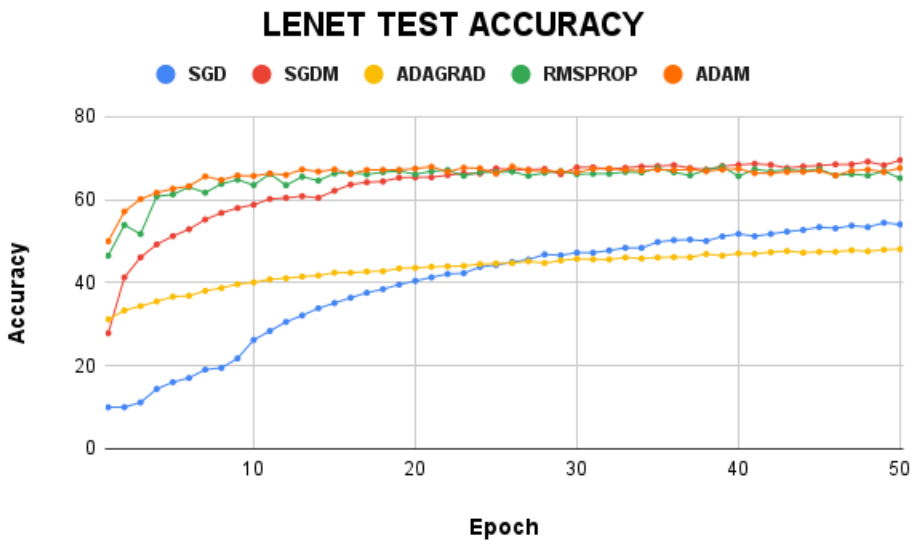
Augmentacja danych

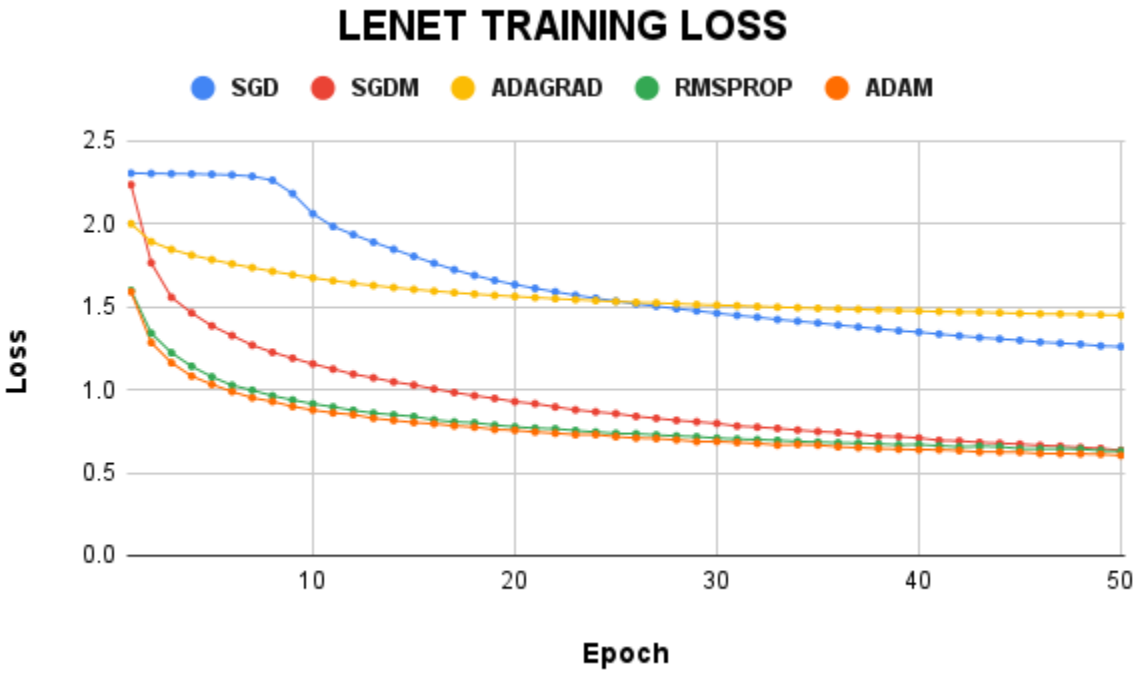
Dla obu modeli wykorzystałem bardzo proste zasady manipulowania zdjęciami, tak, aby zachować ich realistyczność i naturę.

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.

Optymalizacja

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





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 zawiłości 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 kategoryczną 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

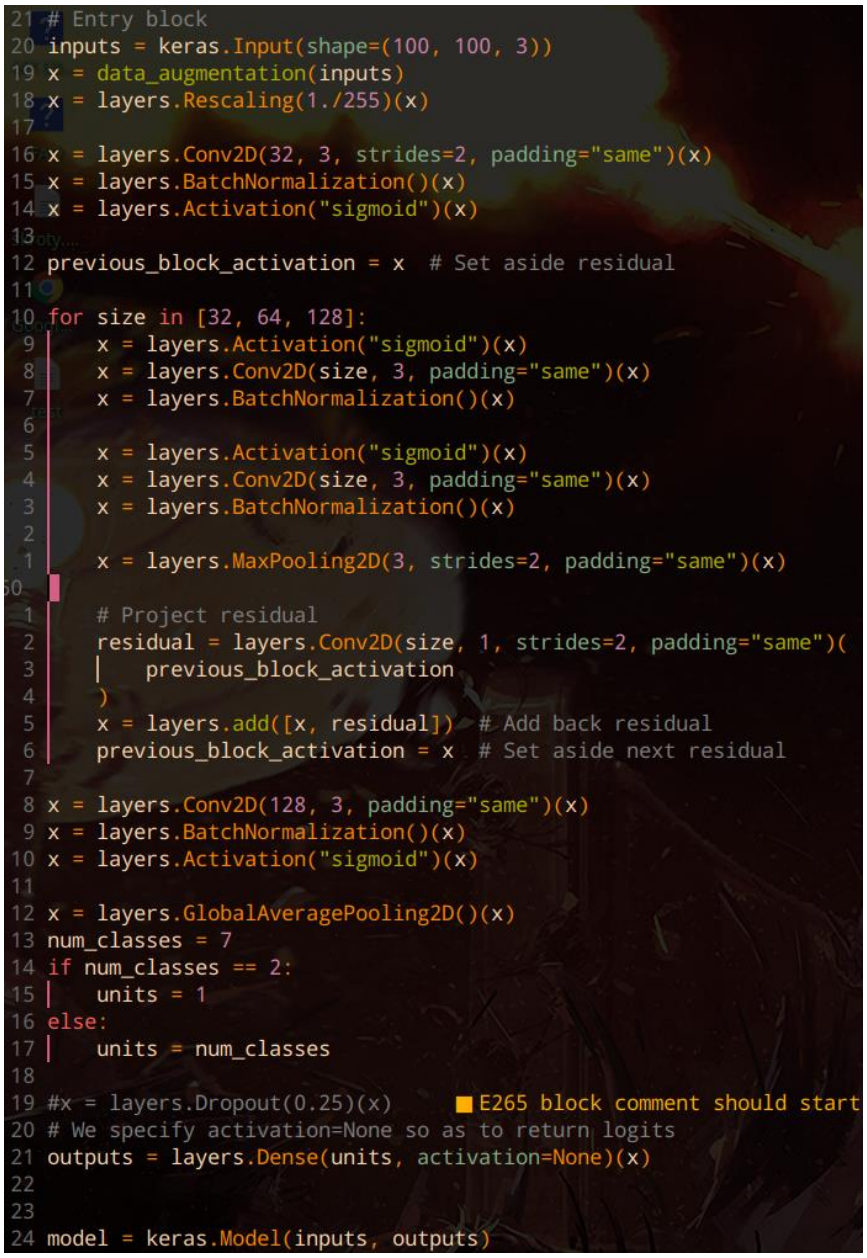
W obu modelach zainspirowałem się strukturą kodu z poradnika na oficjalnej stronie Kerasa i dopasowałem warstwy i ich wymiary do swojego problemu.

```
inputs = keras.Input(shape=(100, 100, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)

x = layers.Conv2D(32, 3, strides=2, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)

previous_block_activation = x  # Set aside residual

for size in [32, 32, 48]:
    x = layers.Activation("relu")(x)
    x = layers.Conv2D(size, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)

    x = layers.Activation("relu")(x)
    x = layers.Conv2D(size, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)

    x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

    # Project residual
    residual = layers.Conv2D(size, 1, strides=2, padding="same")(
        previous_block_activation
    )
    x = layers.add([x, residual])  # Add back residual
    previous_block_activation = x  # Set aside next residual

x = layers.Conv2D(48, 3, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)

x = layers.GlobalAveragePooling2D()(x)
num_classes = 2
if num_classes == 2:
    units = 1
else:
    units = num_classes

x = layers.Dropout(0.25)(x)
# We specify activation=None so as to return logits
outputs = layers.Dense(units, activation=None)(x)


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 douczać.

Samoloty



```
# Entry block
inputs = keras.Input(shape=(100, 100, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)

x = layers.Conv2D(32, 3, strides=2, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("sigmoid")(x)

previous_block_activation = x  # Set aside residual

for size in [32, 64, 128]:
    x = layers.Activation("sigmoid")(x)
    x = layers.Conv2D(size, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)

    x = layers.Activation("sigmoid")(x)
    x = layers.Conv2D(size, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)

    x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

    # Project residual
    residual = layers.Conv2D(size, 1, strides=2, padding="same")(
        previous_block_activation
    )
    x = layers.add([x, residual])  # Add back residual
    previous_block_activation = x  # Set aside next residual

x = layers.Conv2D(128, 3, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("sigmoid")(x)

x = layers.GlobalAveragePooling2D()(x)
num_classes = 7
if num_classes == 2:
    units = 1
else:
    units = num_classes

#x = layers.Dropout(0.25)(x)     E265 block comment should start
# We specify activation=None so as to return logits
outputs = layers.Dense(units, activation=None)(x)


model = keras.Model(inputs, outputs)
```

Model sieci rozróżniającej samoloty sprawiał podobne problemy przy trenowaniu, co poprzedni. Trenowanie tego modelu uświadomiło mi, że mimo wspólnego cierpienia, na studiach znajdą się wrogowie.

Podejmując 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.

Próbowałem 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

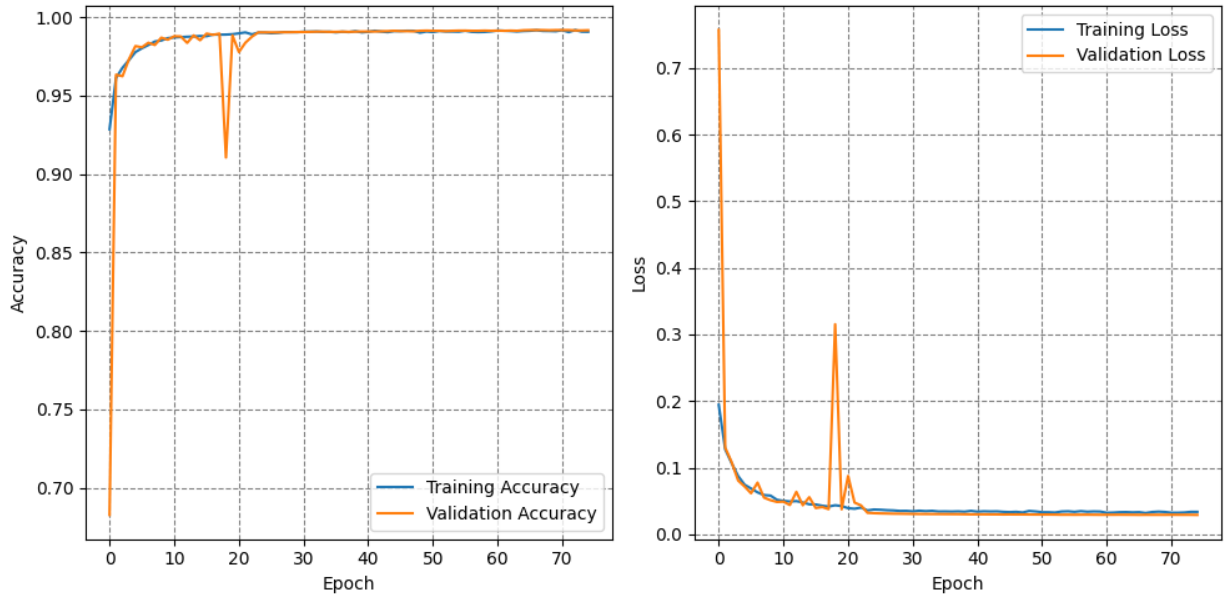
3 – plane_large

4 – plane_partial

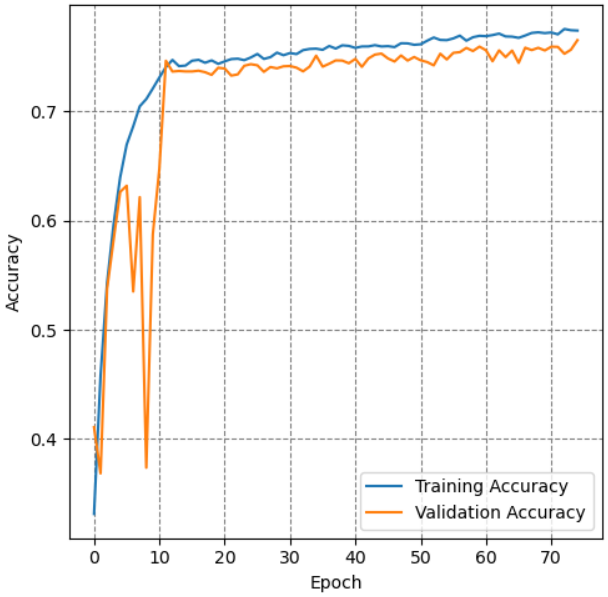
5 – plane_too_far

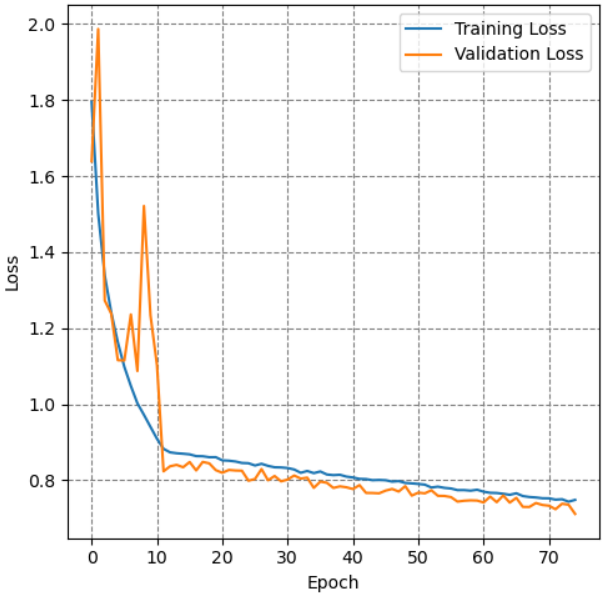
6 – plane_with_trail

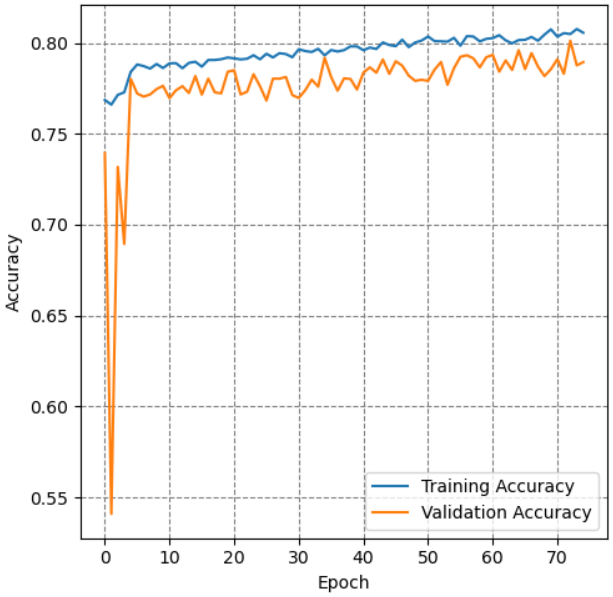
Wykresy modelu cloud

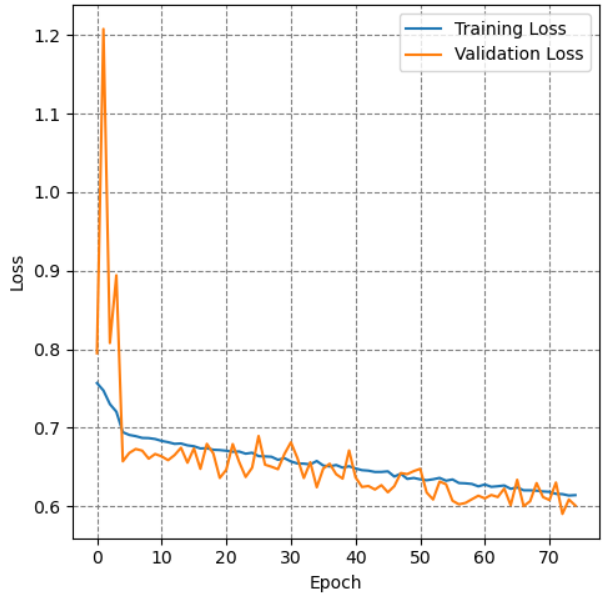


Wykresy modelu planes

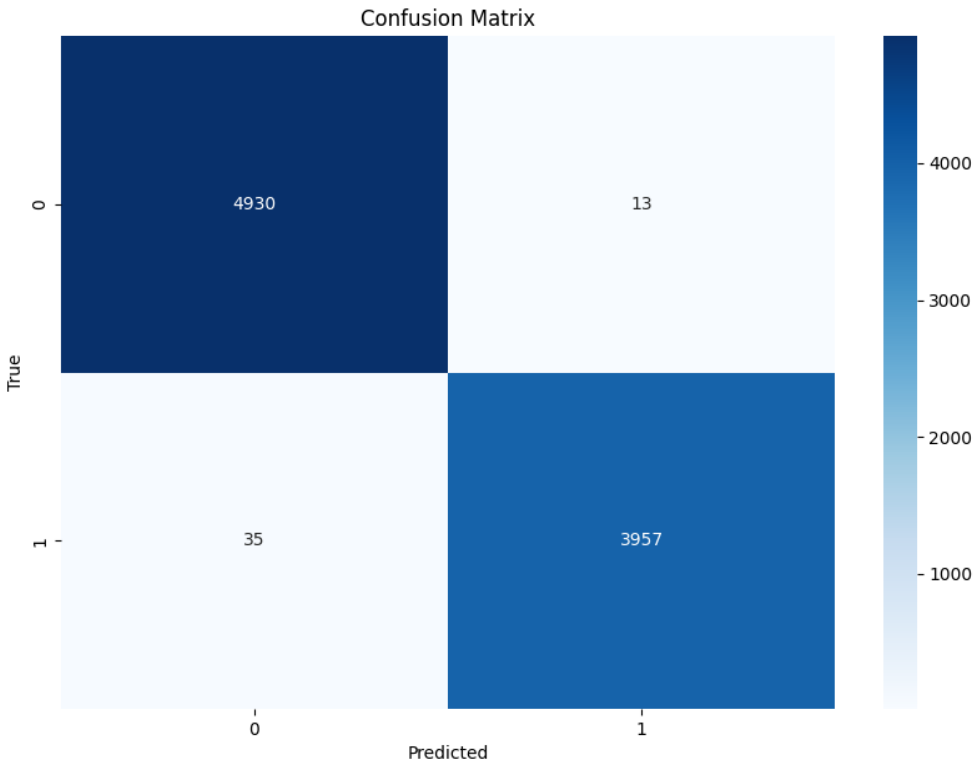


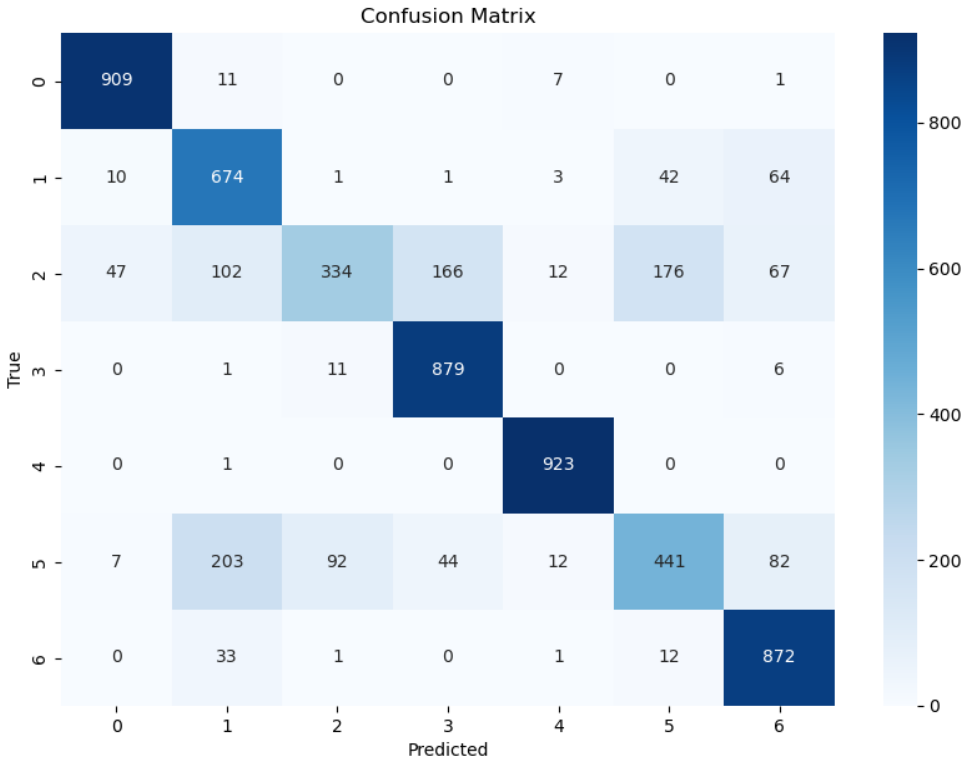






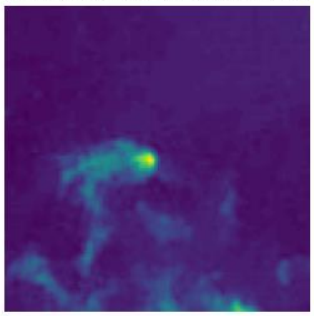
Macierze błędu


Confusion Matrix
4930
13
35
3957
0
1
True
Predicted
4000
3000
2000
1000

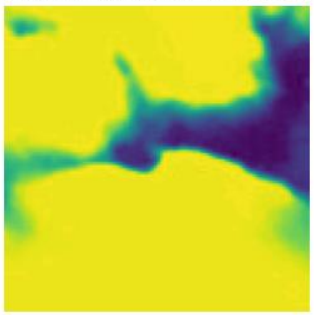


Oznaczenie małej próbki zdjęć chmur

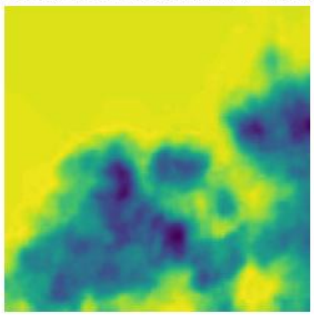
True label: 0, Predicted label: [[-3.996716]]



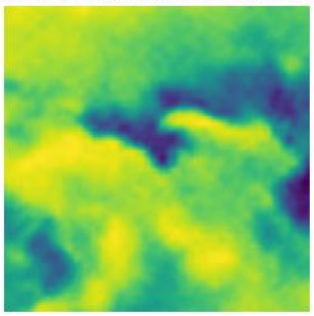
True label: 0, Predicted label: [[-8.172336]]



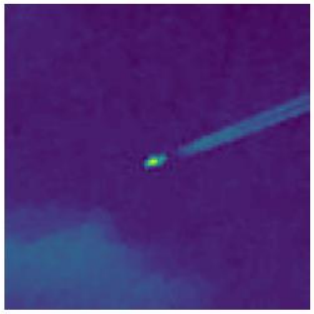
True label: 0, Predicted label: [[-3.8224797]]



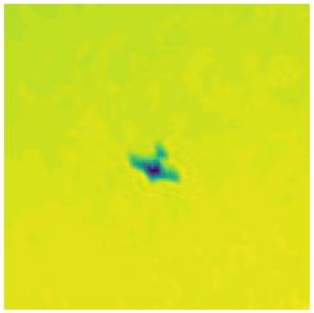
True label: 0, Predicted label: [[-9.651795]]



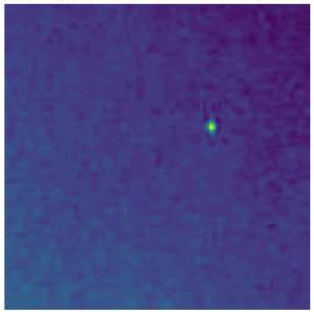
True label: 1, Predicted label: [[6.395374]]



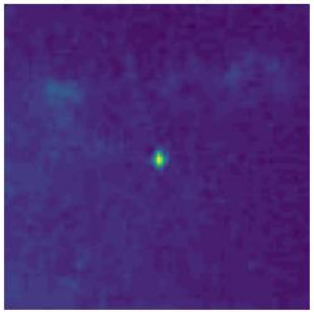
True label: 1, Predicted label: [[2.953097]]



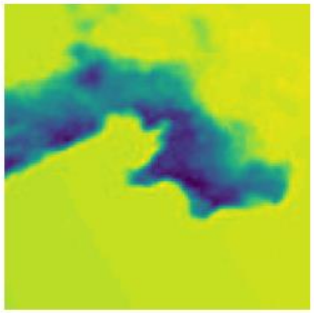
True label: 1, Predicted label: [[6.9782543]]



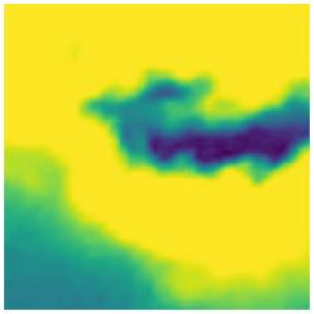
True label: 1, Predicted label: [[5.738247]]



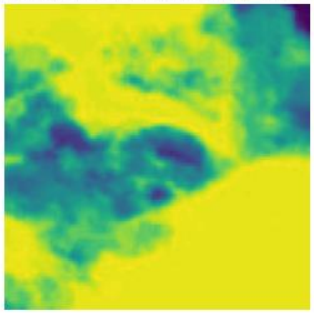
True label: 0, Predicted label: [[-9.137786]]



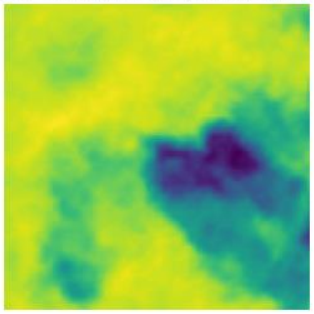
True label: 0, Predicted label: [[-9.380572]]

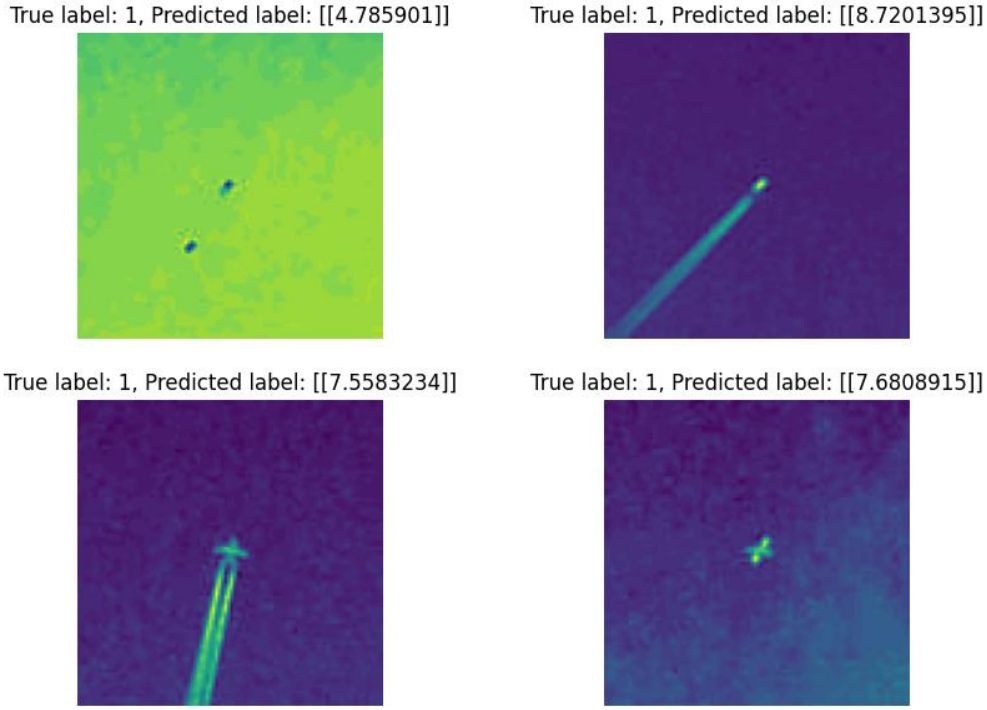


True label: 0, Predicted label: [[-4.8250737]]



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

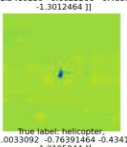




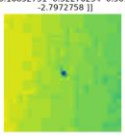
Oznaczenie małej próbki zdjęć samolotów

True label: helicopter,
Predicted label: [[3.589195 -0.9678197   0.1995894 -0.33104768 -2.853521   -1.2543736
 -0.8636088]]

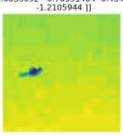
True label: helicopter,
Predicted label: [[3.094601 -1.5469259   0.6022266 -0.48519453 -2.2563426 -0.819826
 -1.3012464]]



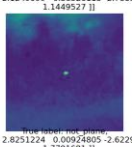
True label: helicopter,
Predicted label: [[2.363579   0.16832751 -0.52276254 0.50117224 -1.6457098 -0.293026
 -2.7972758]]



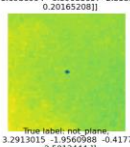
True label: helicopter,
Predicted label: [[1.4553193 -1.0033092 -0.76391464 -0.43410084 0.11085218 -0.6630319
 -1.2105944]]



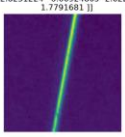
True label: not_plane,
Predicted label: [[-1.6280715   2.1249099   0.38526115 -2.7883117 -2.6203148   1.6994605
  1.1449527]]



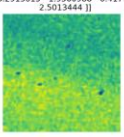
True label: not_plane,
Predicted label: [[-1.0485699   3.6936994 -0.39088657 -2.3533883 -2.1703908   0.93865377
  0.20165208]]



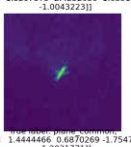
True label: not_plane,
Predicted label: [[-1.1984447   2.8251224   0.00924805 -2.6229098 -1.4257983 -1.0009857
  1.7791681]]



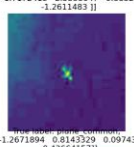
True label: not_plane,
Predicted label: [[-2.3415902   3.2913015 -1.9560988 -0.41776347 -0.37290153 -1.7409433
  2.5013444]]



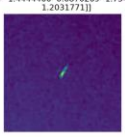
True label: plane_common,
Predicted label: [[-1.4890596 -1.8207375 2.4252036 1.0391184 -1.8650204 0.2705264
  -1.0043223]]



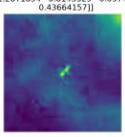
True label: plane_common,
Predicted label: [[-2.4615676 -1.7172413   2.5338647   0.88327634 -1.8308332   1.3209369
  -1.2611483]]



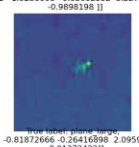
Predicted label: [[-2.355503   1.4444466 0.6870269 -1.754792 -2.5582576 1.592439
  1.2031771]]



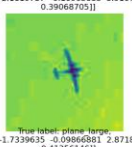
Predicted label: [[-1.615071   -1.2671894   0.8143329   0.097436 -0.99554616 0.07433464
  0.43664157]]



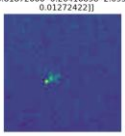
True label: plane_large,
Predicted label: [[-0.2659021 -1.9236008   0.05002552 3.227797   -1.5093081 -1.0720512
  -0.9898198]]



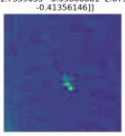
True label: plane_large,
Predicted label: [[-0.4610931 -2.1619737 -0.10903685 3.9107683 -2.2317317 -1.8083148
  0.39068705]]

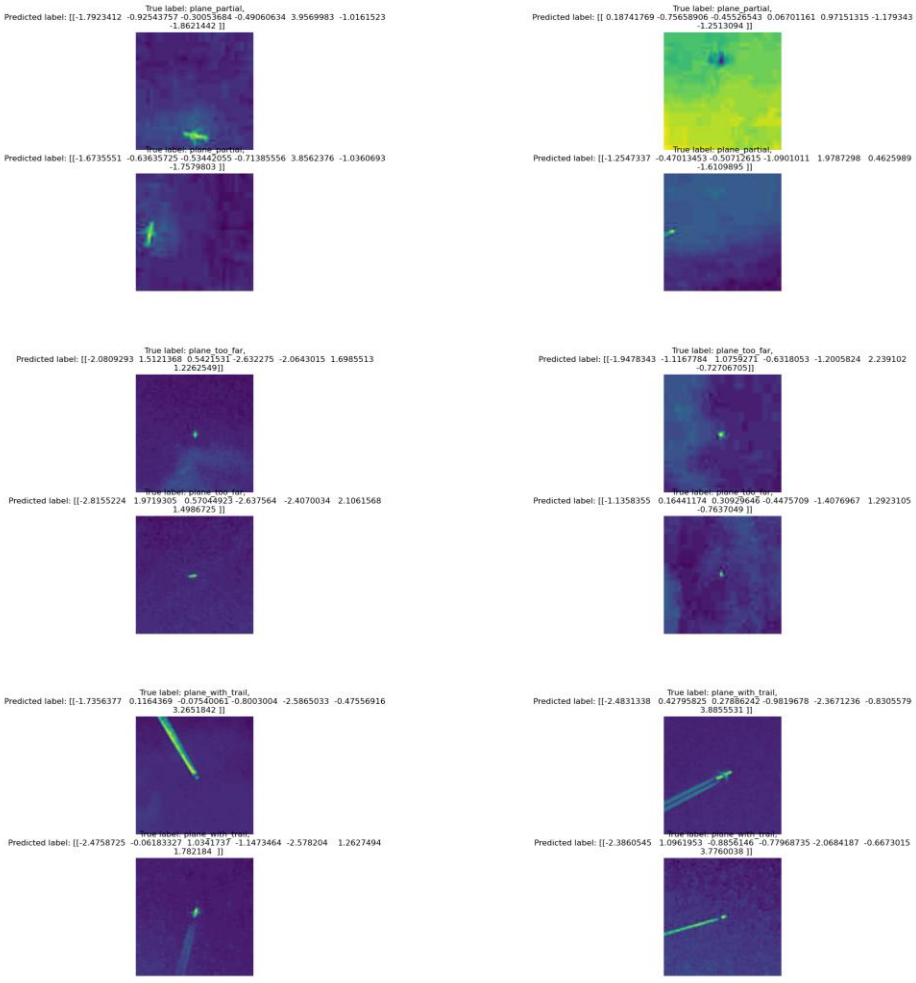


True label: plane_large,
Predicted label: [[-0.9268974 -0.81872666 -0.26416898 2.0959911 -1.3601089 -0.86763126
  0.01272422]]



True label: plane_large,
Predicted label: [[-0.90117365 -1.7339635 -0.09866881 2.8718102 -0.98579866 -1.2339592
  -0.41356146]]





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 nauczyl 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://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