Raport CNN klasyfikacja pokemonow

1. Zebranie danych

Found 562 files belonging to 10 classes.

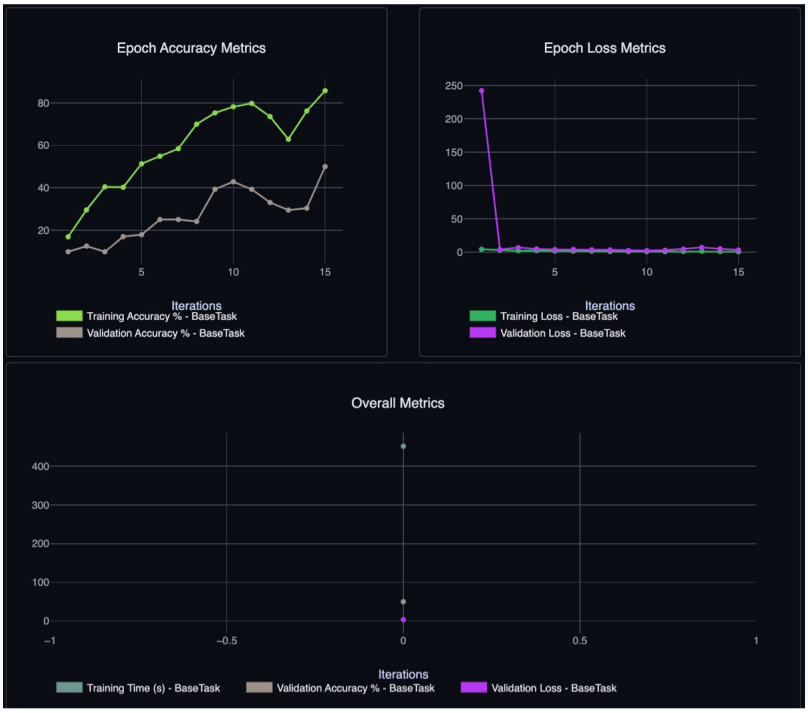
Using 450 files for training.

Found 562 files belonging to 10 classes.

Using 112 files for validation.

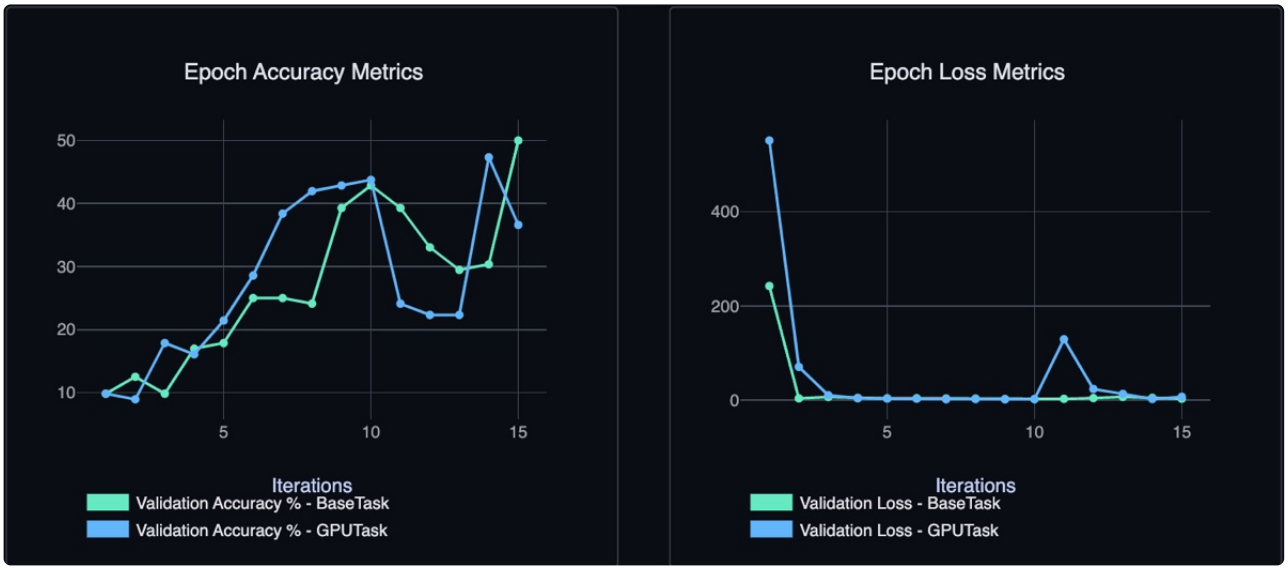
['001Bulbasaur', '004Charmander', '007Squirtle', '025Pikachu', '054Psyduck', '068Machamp', '130Gyarados', '131Lapras', '143Snorlax', '150Mewtwo']

2. Stworzenie podstawowoego modelu za pomoca ResNet50



Widzimy ze model uczył sie 450 sekund oraz mial średni wynik na poziomie 50% accuracy

3a. Porownaie GPU a CPU

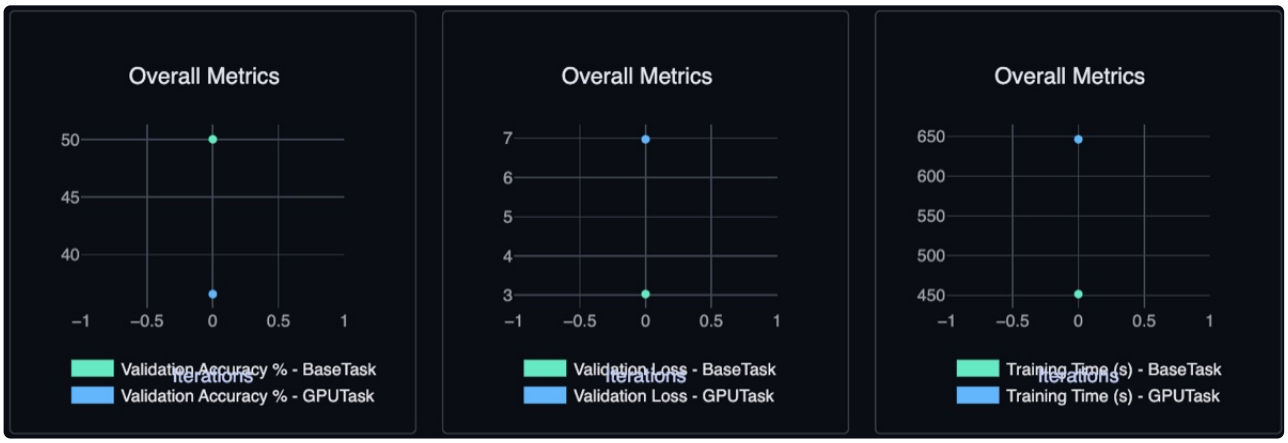


Porównujac wyniki przy uzyciu GPU oraz CPU na macbooku M1 Pro widzimy ze:

- model na GPU wytrenowal sie o 200 sekund wolniej
- dał nam rowniez gorszy wynik bo 36%

najprawdopodobniej spowodowane jest to spowodowane archutektua macbooka ARM i slabo zoptymalizowanego tensorflow w bibliotekach

- tensorflow-macos==2.16.2
- tensorflow-metal==1.2.0
  ktore sa potrzebne dla tej archutektury



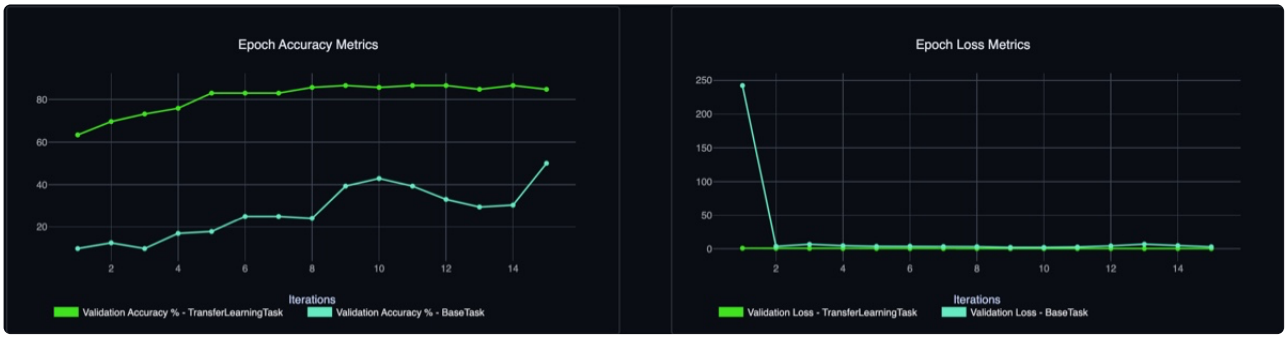
Po przetestowaniu kodu na GPU T4 na Colabie udalo sie uzyskac

- Validation Accuracy: 54.46%
- Total training time: 218.40 seconds

co dobrze pokazuje, że gdybyśmt uczyli model na kartach garficznych Nvidii to uzyskalibyśmy duzo lepszy wynik

3b. Transfer learning

Do reszty modeli został uzyty model zbudowany na CPU czyli BaseTask.



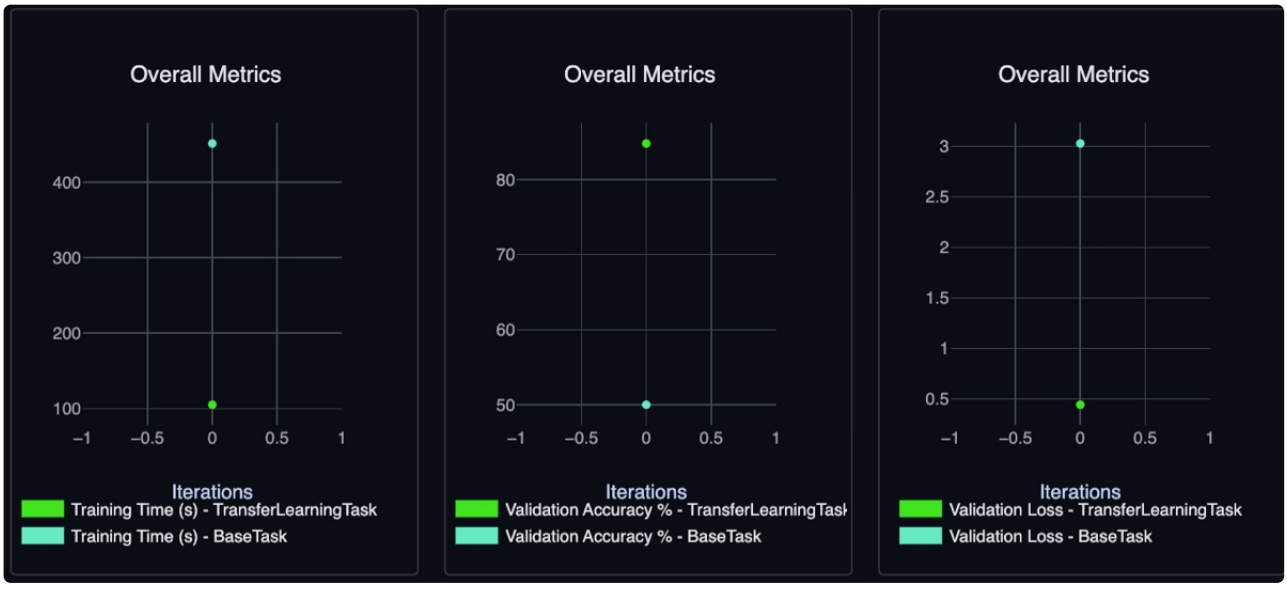
Jak widac model udało sie nauczyc z wynikiem 80% accuracy oraz udało sie go wyuczyc w 100 sekund

- Validation Accuracy: 84.82%
- Total training time: 105.14 seconds

gdzie model bez uzycia transer learningu uczyl sie:

- Validation Accuracy: 50.00%
- Total training time: 451.62 seconds

Jest to spowodowane tym że dajemy modelowi juz przetrenowane wartsty co redukuje przeuczenie w małych zbiorach danych

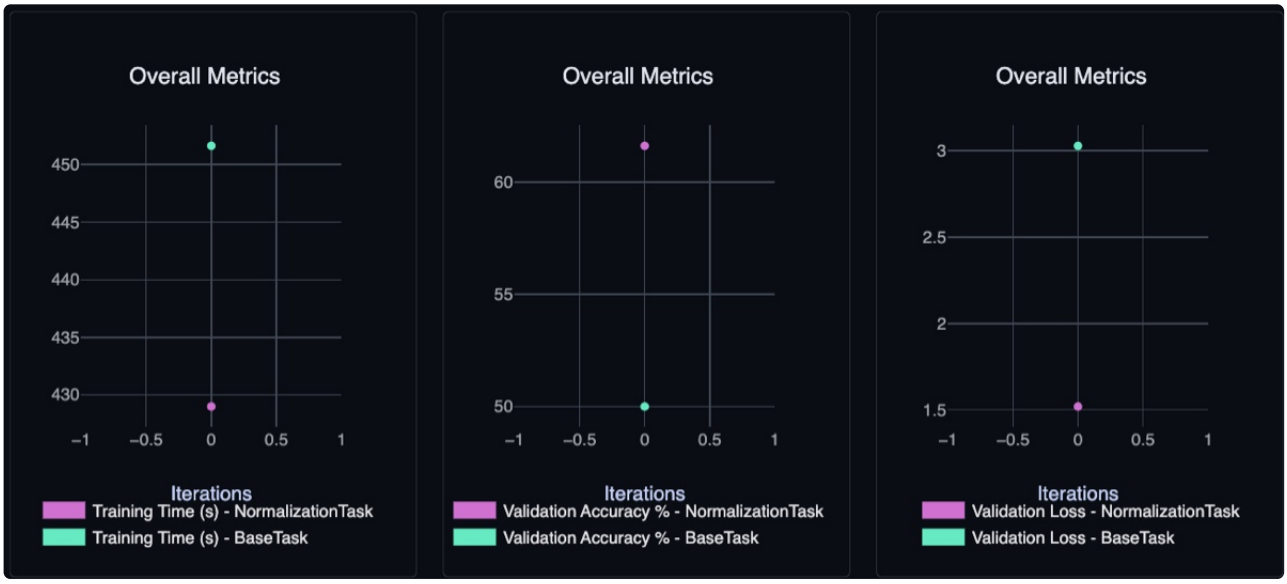


4a. Normalizacja danych

Przechodzimy teraz do normalizacji danych czyli zamiany obrazow z postaci [0,255] do [0,1]

```
normalization_layer = tf.keras.layers.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y))
```

Jak widac model na danych treningowych poradzil sobie podobnie jak podstawowy model. Na danych validacyjnych zdobył lepsze wyniki ponieważ o 11% wiecej accuracy.

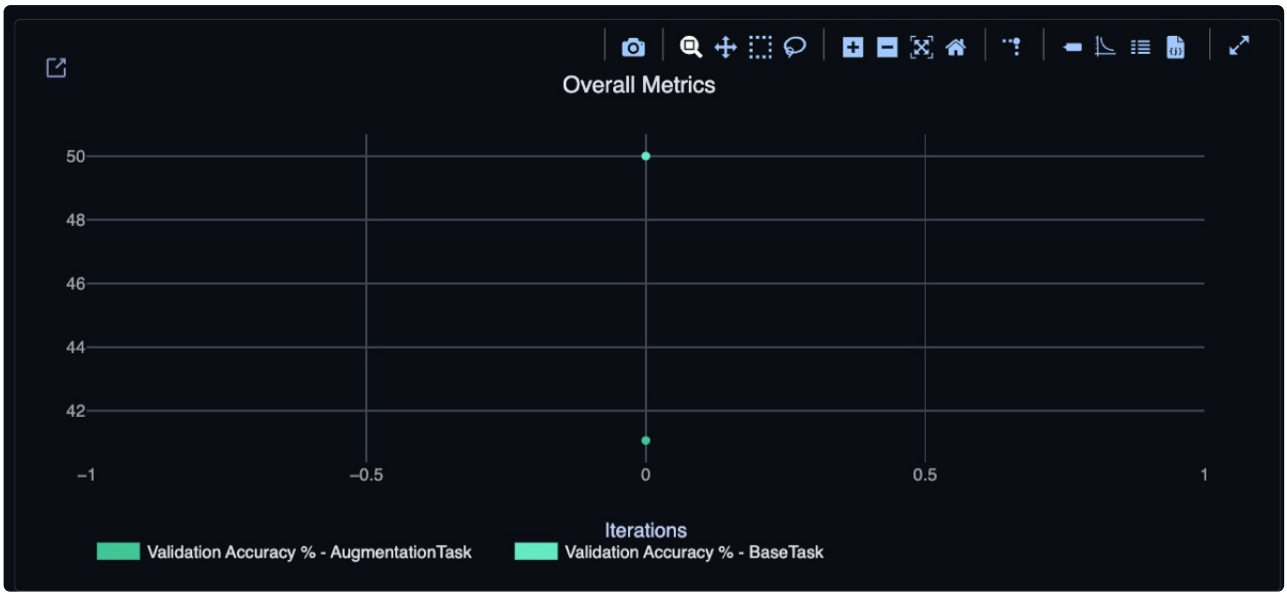


4b. Augmentacja danych

```
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.05),
    layers.RandomZoom(0.1),
])
```

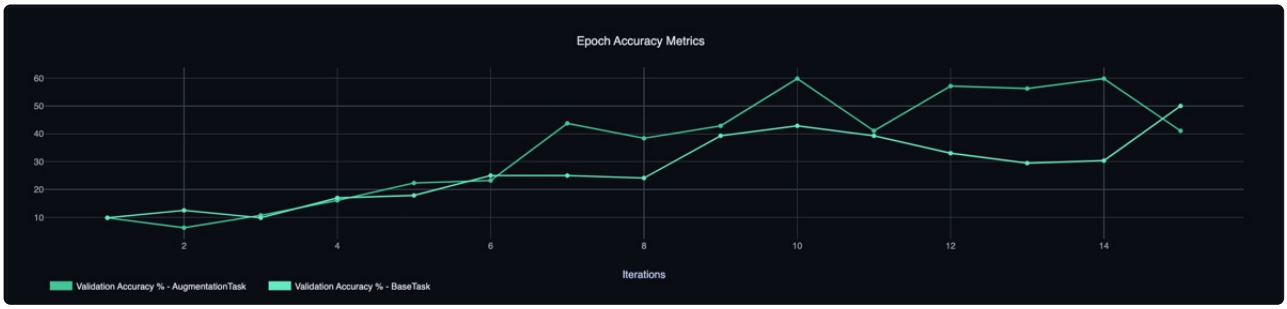
do augmentacji uzyto:

- obracania obrazow horyzontalinie
- obracanie obrazow o okolo 18 stopni
- lekkie zomowanie



Augmentacj przez zmniejszyła poziom accuracy o 9% z powodu zniekształcenia obrazów a na tak małym zbiorze danych może powodować to pogorszenie wyników

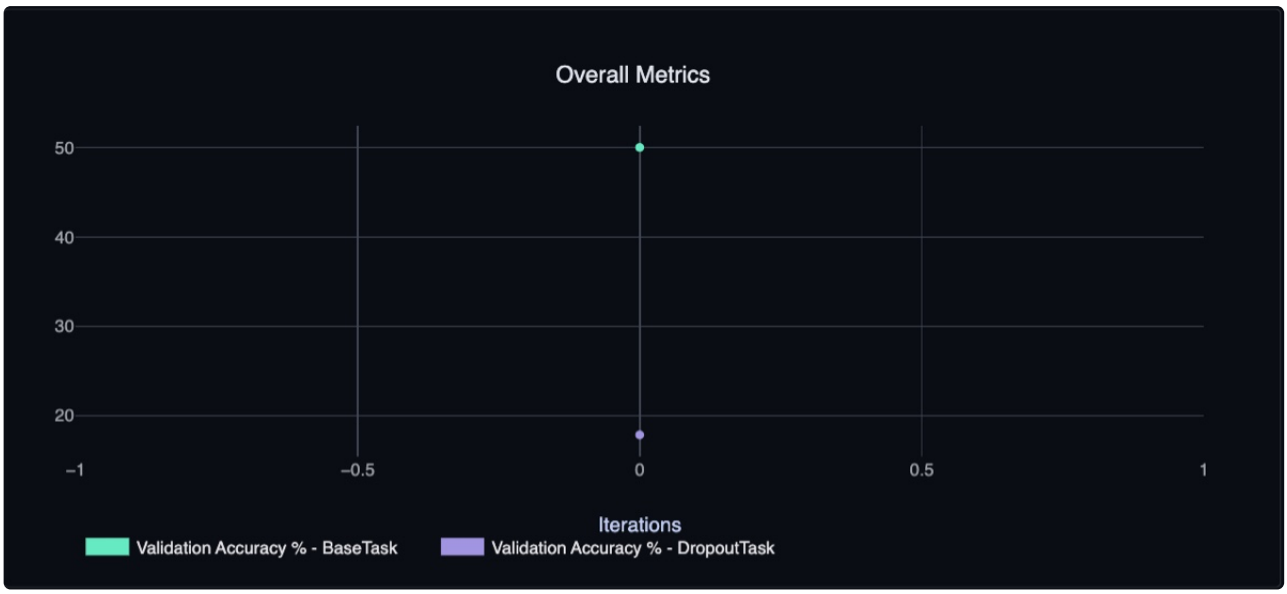
pomimo tego ze na wykresie kazdej epoki widzimy ze wiekszosc czasu byl wyzej niz base model



4c. Dropout

```
x = model.output
 x = layers.GlobalAveragePooling2D()(x)
 x = layers.Dropout(0.5)(x)
 predictions = layers.Dense(class_count, activation='softmax')(x)
```

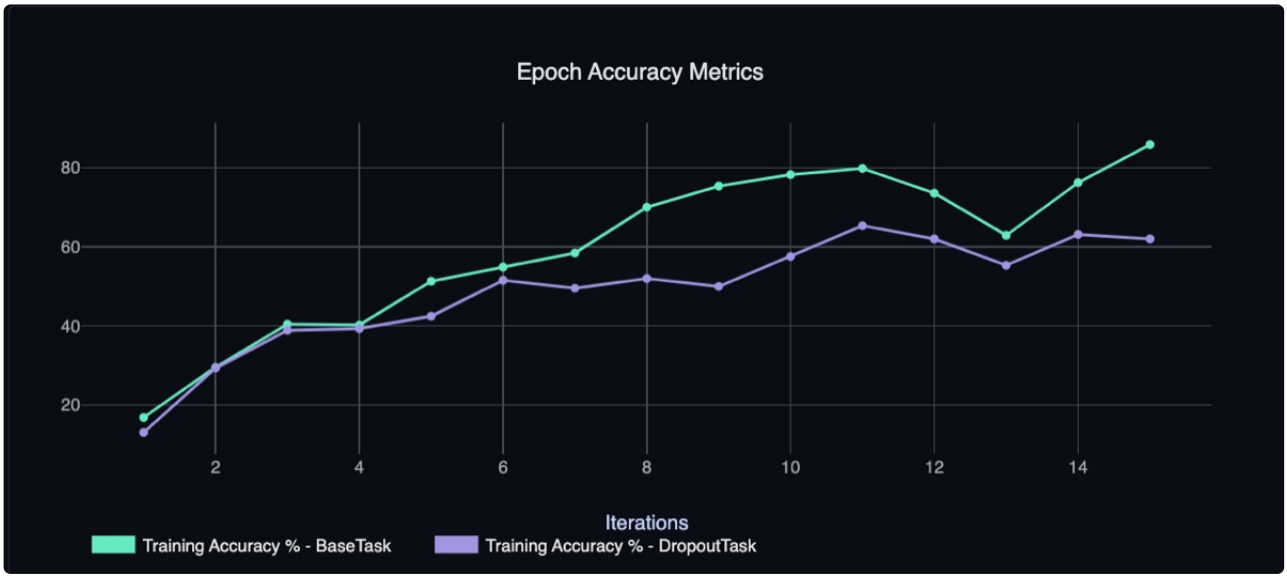
musielismy dodac 2 wartwy konczace aby model mogl klasyfikowac pokemony a zeby Dropout było w srodku modelu



Dropout znaczoca zmniejszył wynik bo z 50% -> 17%. Czas uczenia jest bardzo podobny.

Juz podczas uczenia model gorzej sie uczył o okolo 10%

Dropout dodaje warstwe która wyłacza czesc neuronow na wejsciu co pomaga w przeuczeniu ale rowniez ogranicza czesc informacji o danych.

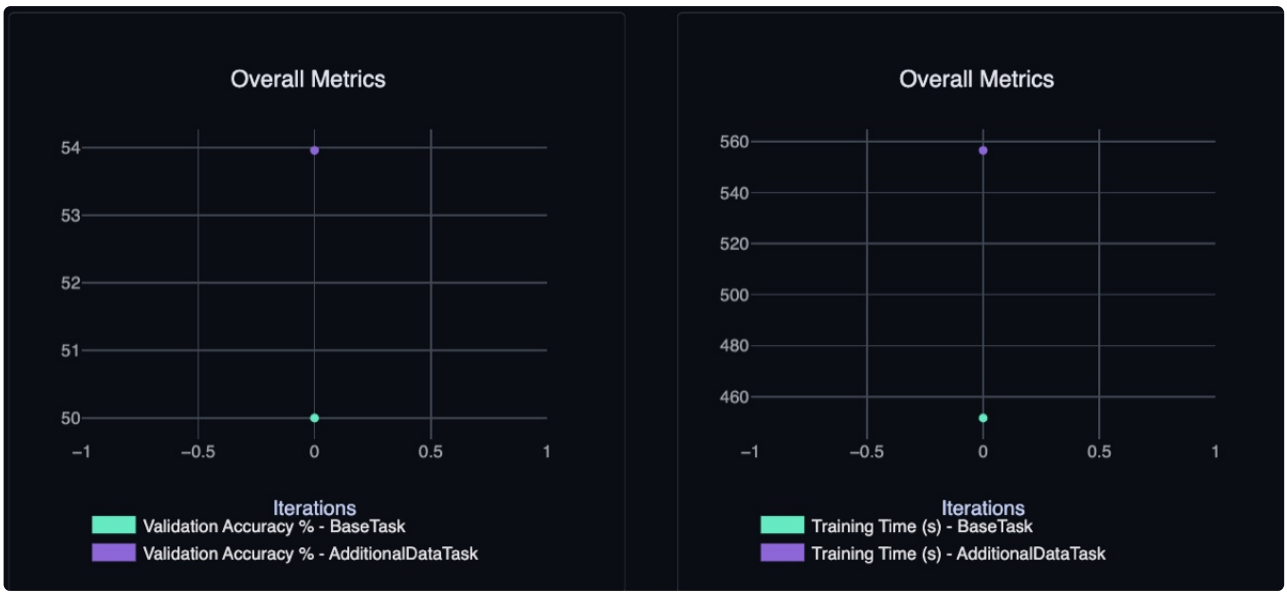


4d. Dodawanie danych

Dodałem 20% danych które wczesniej usunałem z trenowanego modelu. Około 137 dodatkowych zdjec.

Wyniki modelu sie zwiekszył lecz nie znacząco ale to tez zostało dodane tylko 137 zdjec.

Oczywiscie zwiekszył sie czas uczenia modelu

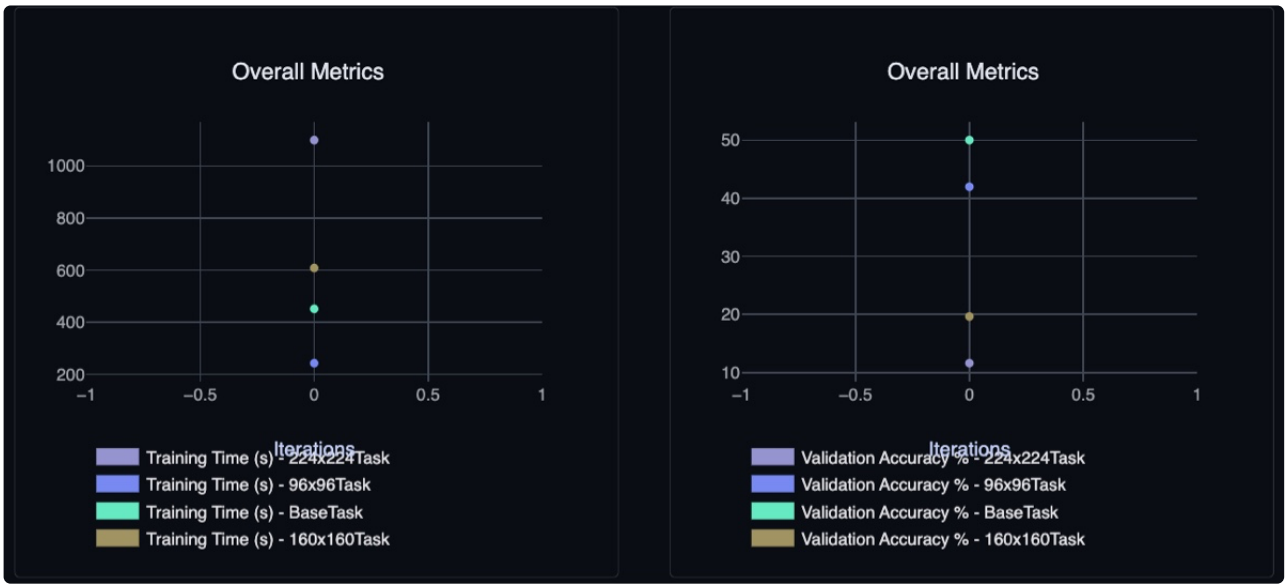


4e. Raport dla roznych wielkosci wejsciowych

Jak widzimy ze zwiekszaniem rozmiaru zdjecia:

- zwieksza sie liniowo czas który potrzebojemy na nauke modelu,
- accuracy z zwikszaniem zdjecia zmniejsza sie ze wzgledu na przeuczenie

Model z wiekszymi zdjeciami dostaje za duzo informacji a co za tym idzie za duzo szumu

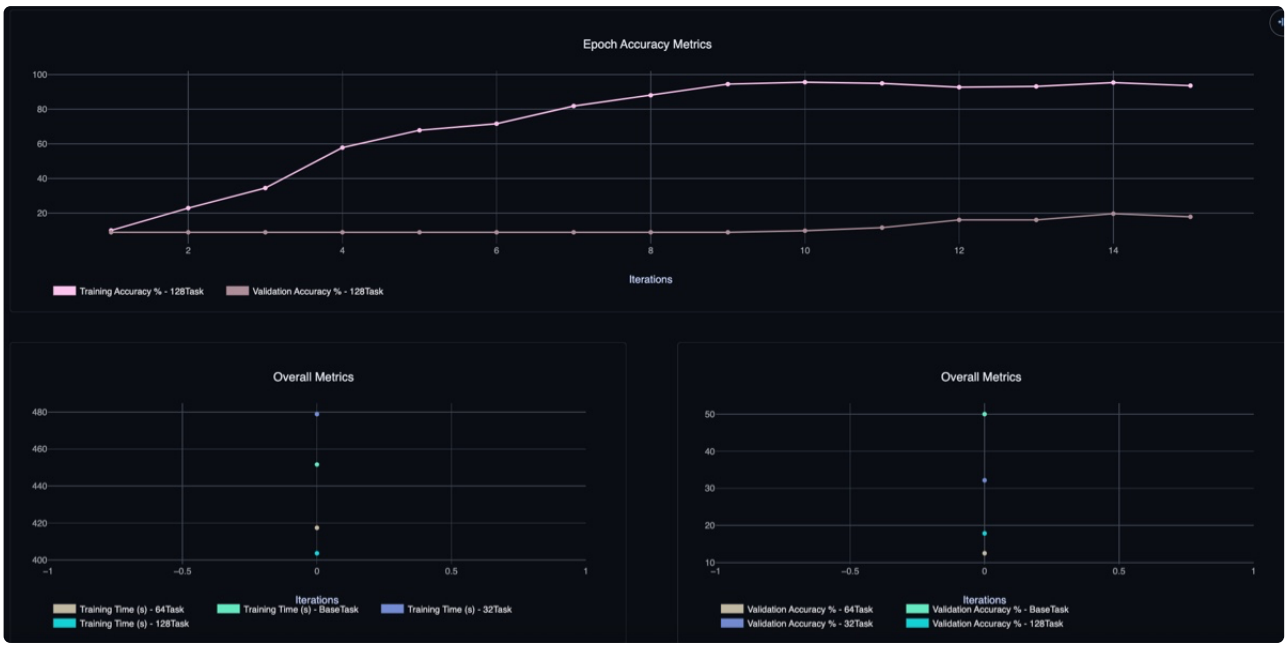


4f. zmiana batch size

Jak widzimy ze zmiana ilosci zdjec wrzucanych naraz pogarsza wyniki.

Batch size w BaseTask to 16 i kazdy kolejny wynik był gorszy.

Model sie przeuczał bo na testing osiagal wynik 90% a na validation okolo 10%



4g. zmiana struktury sieci

porównałem 4 modele:
VGG16, ResNet101, InceptionV3, MobileNet do ResNet50

wszytskie modele były trenowane bez dodatko przetrenowanych wag oraz domyslnych warstw koncowych

z porowanych modeli widzimy ze najlepiej poradzil sobie podstawowy resNet50. InceptionV3 oraz ResNet101 rownież dobrze sobie poradzily

Widzimy rowniez ze VGG prawie nie zaczal w ogole sie uczyc pomimo tego ze zajelo mu "uczenie sie" najdluzej.

Mobilenet sie przeuczył ponieważ na testowych mial całkiem porównywalny wynik do reszty a na validacyjnych nie mógł uzyskac lepszego niz 10%

5. Optymalizacja Hiperparametrow

Nlestety nie udało mi sie przeporwadzic optymalizacji hiperparametrów ze wzgledu na błedy zwiazane z bibliotekami oraz tworzeniem agenta na lokalnym urzadzeniu.