

Porównanie wydajności różnych algorytmów optymalizacji w uczeniu CNN

Michał Domin, Damian Raczyński, Kamil Kaproń

2023-07-01

Contents

1 Wprowadzenie	4
2 Optymalizatory	5
2.1 SGD (Stochastic Gradient Descent)	5
2.2 RMSprop (Root Mean Square Propagation)	5
2.3 Adam (Adaptive Moment Estimation)	5
2.4 Adadelta	5
2.5 Adagrad	5
2.6 Adamax	5
2.7 Nadam	5
2.8 Ftrl (Follow-the-regularized-leader)	5
3 Metryki	6
3.1 CategoricalCrossentropy	6
3.2 MeanSquaredError	6
3.3 BinaryAccuracy	6
3.4 CategoricalAccuracy	6
3.5 AUC	6
3.6 Precision	6
3.7 Recall	6
3.8 FalsePositives	6
3.9 FalseNegatives	6
3.10 TruePositives	6
3.11 TrueNegatives	7
4 Zestawy Danych	8
4.1 Iris	8
4.2 CIFAR-10	8
4.3 Dry Bean	8
4.4 Raisin	8
5 Architektura Modelu	9
5.1 Model dla Iris	9
5.2 Model dla CIFAR-10	9
5.3 Model dla Dry Bean	9
5.4 Model dla Raisin	9
6 Wyniki	9
6.1 Iris	10
6.1.1 Iris dla optymalizatora SGD	10
6.1.2 Iris dla optymalizatora RMSprop	11
6.1.3 Iris dla optymalizatora Adam	12
6.1.4 Iris dla optymalizatora Adadelta	13
6.1.5 Iris dla optymalizatora Adagrad	14
6.1.6 Iris dla optymalizatora Adamax	15
6.1.7 Iris dla optymalizatora Nadam	16
6.1.8 Iris dla optymalizatora Ftrl	17
6.2 CIFAR	19
6.2.1 CIFAR dla optymalizatora SGD	19
6.2.2 CIFAR dla optymalizatora RMSprop	20
6.2.3 CIFAR dla optymalizatora Adam	21
6.2.4 CIFAR dla optymalizatora Adadelta	22
6.2.5 CIFAR dla optymalizatora Adagrad	23
6.2.6 CIFAR dla optymalizatora Adamax	24
6.2.7 CIFAR dla optymalizatora Nadam	25
6.2.8 CIFAR dla optymalizatora Ftrl	26
6.3 DryBean	28
6.3.1 DryBean dla optymalizatora SGD	28
6.3.2 DryBean dla optymalizatora RMSprop	29
6.3.3 DryBean dla optymalizatora Adam	30

6.3.4	DryBean dla optymalizatora Adadelta	31
6.3.5	DryBean dla optymalizatora Adagrad	32
6.3.6	DryBean dla optymalizatora Adamax	33
6.3.7	DryBean dla optymalizatora Nadam	34
6.3.8	DryBean dla optymalizatora Ftrl	35
6.4	Raisin	37
6.4.1	Raisin dla optymalizatora SGD	37
6.4.2	Raisin dla optymalizatora RMSprop	38
6.4.3	Raisin dla optymalizatora Adam	39
6.4.4	Raisin dla optymalizatora Adadelta	40
6.4.5	Raisin dla optymalizatora Adagrad	41
6.4.6	Raisin dla optymalizatora Adamax	42
6.4.7	Raisin dla optymalizatora Nadam	43
6.4.8	Raisin dla optymalizatora Ftrl	44

7 Podsumowanie

46

1 Wprowadzenie

W tym projekcie porównujemy różne algorytmy optymalizacji stosowane w procesie uczenia sieci konwolucyjnych (CNN). Do testów używamy czterech zestawów danych: Iris, CIFAR-10, Dry Bean i Raisin dostępnych w repozytorium UCI Machine Learning. Każdy z tych zestawów danych jest używany do treningu i oceny modelu CNN. Następnie, stosujemy różne algorytmy optymalizacji w procesie uczenia, takie jak SGD, RMSprop, Adam, Adadelta, Adagrad, Adamax, Nadam i Ftrl.

Każdy model jest oceniany za pomocą dziesięciokrotnej walidacji krzyżowej (K-fold cross-validation) i różnych miar ewaluacji, w tym błędu średniokwadratowego (MSE), macierzy pomyłek, krzywej ROC, specyficzności, czułości i pola pod krzywą ROC.

2 Optymalizatory

2.1 SGD (Stochastic Gradient Descent)

Stochastic Gradient Descent to podstawowy optymalizator wykorzystywany w uczeniu maszynowym. Aktualizuje wagę modelu proporcjonalnie do ujemnego gradientu funkcji straty dla pojedynczego przykładu w każdej iteracji. SGD jest stosunkowo prosty i wydajny, ale może mieć problem z utknięciem w minimach lokalnych.

2.2 RMSprop (Root Mean Square Propagation)

RMSprop to optymalizator, który próbuje rozwiązać problem utknięcia w minimach lokalnych, występujący w SGD. Oblicza wykładniczo ważone średnie kwadratów gradientów i skaluje aktualizacje wag przez pierwiastek z tych średnich. Działa dobrze w przypadku niestabilnych gradientów w głębokich sieciach neuronowych.

2.3 Adam (Adaptive Moment Estimation)

Adam to adaptacyjny optymalizator, który łączy cechy RMSprop i momentum. Oblicza wykładniczo ważone średnie zarówno gradientów, jak i ich kwadratów. Wykorzystuje te estymaty do aktualizacji wag. Adam dostosowuje tempo uczenia dla każdej wagi na podstawie historycznych wartości gradientów, co czyni go skutecznym optymalizatorem w wielu przypadkach.

2.4 Adadelta

Adadelta to adaptacyjny optymalizator, który działa podobnie do RMSprop, ale nie wymaga ręcznego dostosowywania szybkości uczenia. Zamiast tego wykorzystuje historyczne estymaty kwadratów gradientów do adaptacji tempa uczenia. Adadelta eliminuje potrzebę ustawiania globalnego współczynnika uczenia.

2.5 Adagrad

Adagrad to optymalizator, który dostosowuje tempo uczenia dla każdego parametru na podstawie historycznych wartości gradientów. Składowe gradientu są zapisywane w celu uwzględnienia różnic w tempie uczenia między różnymi parametrami. Adagrad jest skuteczny dla rzadkich danych, ale może prowadzić do nadmiernego tłumienia gradientów w późniejszych etapach uczenia.

2.6 Adamax

Adamax jest wariacją optymalizatora Adam, która zastępuje wykładniczo ważone średnie kwadratów gradientów przez nieskończoną normę maksimum. Jest bardziej odporny na wyjątkowo duże gradienty i może być skuteczny w modelach z rzadkimi lub dużymi parami.

2.7 Nadam

Nadam to połączenie optymalizatorów Adam i Nesterov Momentum. Wykorzystuje adaptacyjne momenty gradientów z Adam i dodatkowo uwzględnia korekcję Nesterova, aby uaktualnić wagę. Nadam łączy korzyści obu tych metod i jest skutecznym optymalizatorem dla wielu problemów.

2.8 Ftrl (Follow-the-regularized-leader)

Ftrl to optymalizator opracowany dla problemów uczenia maszynowego o bardzo dużych zbiorach danych. Wykorzystuje regularyzację L1 i L2 oraz adaptacyjne dostosowanie tempo uczenia dla każdego parametru. Ftrl jest skuteczny dla problemów klasyfikacji binarnej z dużymi danymi.

Wszystkie te optymalizatory mają różne cechy i są skuteczne w różnych scenariuszach. Wybór optymalizatora zależy od specyfiki problemu, danych i architektury modelu. Często najlepiej jest przetestować kilka optymalizatorów i dostosować ich parametry, aby znaleźć ten, który działa najlepiej w konkretnym przypadku.

3 Metryki

3.1 CategoricalCrossentropy

CategoricalCrossentropy jest metryką używaną w problemach klasyfikacji wieloklasowej. Mierzy odległość między prawdziwym rozkładem etykiet a przewidywanym rozkładem, wykorzystując funkcję straty logarytmicznej. Im niższa wartość tej metryki, tym lepsze dopasowanie modelu do prawdziwych etykiet.

3.2 MeanSquaredError

MeanSquaredError jest metryką używaną w problemach regresji. Oblicza średnią kwadratów różnic między prawdziwymi etykietami a przewidywanymi wartościami. Im niższa wartość tej metryki, tym lepsze dopasowanie modelu do prawdziwych wartości.

3.3 BinaryAccuracy

BinaryAccuracy to metryka używana w problemach binarnej klasyfikacji. Oblicza procent poprawnych przewidywań dla klasyfikacji binarnej (dwóch klas). Porównuje przewidywane etykiety z rzeczywistymi etykietami. Wyższa wartość tej metryki oznacza lepszą dokładność klasyfikacji.

3.4 CategoricalAccuracy

CategoricalAccuracy jest metryką używaną w problemach klasyfikacji wieloklasowej. Oblicza procent poprawnych przewidywań dla każdej klasy w przypadku kodowania etykiet w postaci "gorącojedynkowej" (one-hot encoding). Wyższa wartość tej metryki oznacza lepszą dokładność klasyfikacji.

3.5 AUC

AUC (Area Under the Curve) to metryka używana w problemach klasyfikacji binarnej. Mierzy jakość klasyfikatora, obliczając pole pod krzywą ROC (Receiver Operating Characteristic). Wyższa wartość AUC oznacza lepszą zdolność modelu do rozróżniania między klasami.

3.6 Precision

Precision to metryka używana w problemach klasyfikacji binarnej. Oblicza stosunek prawdziwie pozytywnych przewidywań do wszystkich pozytywnych przewidywań. Precision mierzy stopień dokładności pozytywnych przewidywań modelu.

3.7 Recall

Recall, znany również jako czułość lub true positive rate, to metryka używana w problemach klasyfikacji binarnej. Oblicza stosunek prawdziwie pozytywnych przewidywań do sumy prawdziwie pozytywnych przewidywań i fałszywie negatywnych przewidywań. Recall mierzy zdolność modelu do poprawnego wykrywania pozytywnych przypadków.

3.8 FalsePositives

FalsePositives to metryka, która oblicza liczbę fałszywie pozytywnych przewidywań w problemach klasyfikacji binarnej. Oznacza ona liczbę przypadków, w których model błędnie przewidział pozytywną klasę, podczas gdy prawdziwa klasa była negatywna.

3.9 FalseNegatives

FalseNegatives to metryka, która oblicza liczbę fałszywie negatywnych przewidywań w problemach klasyfikacji binarnej. Oznacza ona liczbę przypadków, w których model błędnie przewidział negatywną klasę, podczas gdy prawdziwa klasa była pozytywna.

3.10 TruePositives

TruePositives to metryka, która oblicza liczbę prawdziwie pozytywnych przewidywań w problemach klasyfikacji binarnej. Oznacza ona liczbę przypadków, w których model poprawnie przewidział pozytywną klasę.

3.11 TrueNegatives

TrueNegatives to metryka, która oblicza liczbę prawdziwie negatywnych przewidywań w problemach klasyfikacji binarnej. Oznacza ona liczbę przypadków, w których model poprawnie przewidziała negatywną klasę.

Te metryki są używane do oceny wydajności modelu i pomagają w zrozumieniu jego zdolności do klasyfikacji lub regresji w zależności od rodzaju problemu. Wybór odpowiedniej metryki zależy od specyfiki problemu i celu modelu.

4 Zestawy Danych

4.1 Iris

Zestaw danych Iris składa się z 150 próbek z pięciu atrybutów: długości i szerokości płatków, długości i szerokości działek kielicha oraz gatunku irysa. Gatunek jest naszą zmienną docelową, która jest kodowana w formie one-hot encoding. Dane są normalizowane i sformatowane do 3D aby pasować do wejścia modelu CNN.

4.2 CIFAR-10

CIFAR-10 to zestaw danych składający się z 60000 kolorowych obrazków o rozmiarze 32x32 podzielonych na 10 klas. Dane są normalizowane i kodowane w formie one-hot encoding.

4.3 Dry Bean

Zestaw danych Dry Bean zawiera informacje o siedmiu różnych odmianach suchych fasolek. Każda odmiana fasoli jest reprezentowana przez różne cechy w zestawie danych. Wszystkie cechy są liczbami rzeczywistymi.

4.4 Raisin

Zestaw danych Raisin składa się z informacji na temat różnych odmian rodzynków. Dane są kodowane jako liczby rzeczywiste.

5 Architektura Modelu

5.1 Model dla Iris

Model dla Iris składa się z warstwy konwolucyjnej (Conv1D) z 64 filtrami i jądrem o rozmiarze 3. Następnie, używamy max pooling i spłaszczamy wyniki. Na końcu, stosujemy gęstą warstwę (Dense) z funkcją aktywacji softmax do klasyfikacji.

```
1 model = Sequential()
2 model.add(Conv1D(64, 3, activation='relu', input_shape=(in_dim, 1)))
3 model.add(MaxPooling1D(pool_size=2))
4 model.add(Flatten())
5 model.add(Dense(out_dim, activation='softmax'))
```

5.2 Model dla CIFAR-10

Model dla CIFAR-10 zawiera dwie warstwy konwolucyjne (Conv2D) z 32 i 64 filtrami odpowiednio, każda z funkcją aktywacji relu. Między nimi, stosujemy warstwę dropout z rate równym 0.2 i max pooling. Na końcu, spłaszczamy wyniki i stosujemy dwie warstwy Dense, z 128 jednostkami i 10 jednostkami odpowiednio, ta ostatnia z funkcją aktywacji softmax do klasyfikacji.

```
1 model = Sequential()
2 model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
3 model.add(Dropout(0.2))
4 model.add(MaxPooling2D((2, 2)))
5 model.add(Conv2D(64, (3, 3), activation='relu'))
6 model.add(Dropout(0.2))
7 model.add(MaxPooling2D((2, 2)))
8 model.add(Flatten())
9 model.add(Dense(128, activation='relu'))
10 model.add(Dense(10, activation='softmax'))
```

5.3 Model dla Dry Bean

Model dla Dry Bean składa się z dwóch warstw konwolucyjnych (Conv1D) z 32 i 64 filtrami odpowiednio, każda z funkcją aktywacji relu. Między nimi, stosujemy warstwę dropout z rate równym 0.2 i batch normalization. Po spłaszczeniu wyników, stosujemy trzy gęste warstwy (Dense) z 128, 64 i liczbą klas jednostkami odpowiednio, ta ostatnia z funkcją aktywacji softmax do klasyfikacji.

```
1 model = Sequential()
2 model.add(Conv1D(32, 2, activation='relu', input_shape=(in_dim, 1)))
3 model.add(Dropout(0.2))
4 model.add(BatchNormalization())
5 model.add(Conv1D(64, 2, activation='relu'))
6 model.add(Dropout(0.2))
7 model.add(BatchNormalization())
8 model.add(Flatten())
9 model.add(Dense(128, activation='relu'))
10 model.add(BatchNormalization())
11 model.add(Dense(64, activation='relu'))
12 model.add(BatchNormalization())
13 model.add(Dense(out_dim, activation='softmax'))
```

5.4 Model dla Raisin

Model dla Raisin składa się z dwóch warstw konwolucyjnych (Conv1D) z 16 filtrami i jądrem o rozmiarze 3, każda z funkcją aktywacji relu. Po nich, stosujemy warstwę BatchNormalization i spłaszczamy wyniki. Na końcu, stosujemy gęstą warstwę (Dense) z funkcją aktywacji softmax do klasyfikacji.

```
1 model = Sequential()
2 model.add(Conv1D(16, 3, activation='relu', input_shape=(in_dim, 1)))
3 model.add(BatchNormalization())
4 model.add(Conv1D(16, 3, activation='relu'))
5 model.add(BatchNormalization())
6 model.add(Flatten())
7 model.add(Dense(out_dim, activation='softmax'))
```

6 Wyniki

6.1 Iris

6.1.1 Iris dla optymalizatora SGD

Iris classification model for optimizer: SGD

Metric: categorical_crossentropy → Score: 0.17425380647182465

Metric: mean_squared_error → Score: 0.0374273844063282

Metric: binary_accuracy → Score: 0.9523809552192688

Metric: categorical_accuracy → Score: 0.9285714030265808

Metric: auc → Score: 0.9974489808082581

Metric: precision → Score: 0.9285714030265808

Metric: recall → Score: 0.9285714030265808

Metric: false_positives → Score: 1.0

Metric: false_negatives → Score: 1.0

Metric: true_positives → Score: 13.0

Metric: true_negatives → Score: 27.0

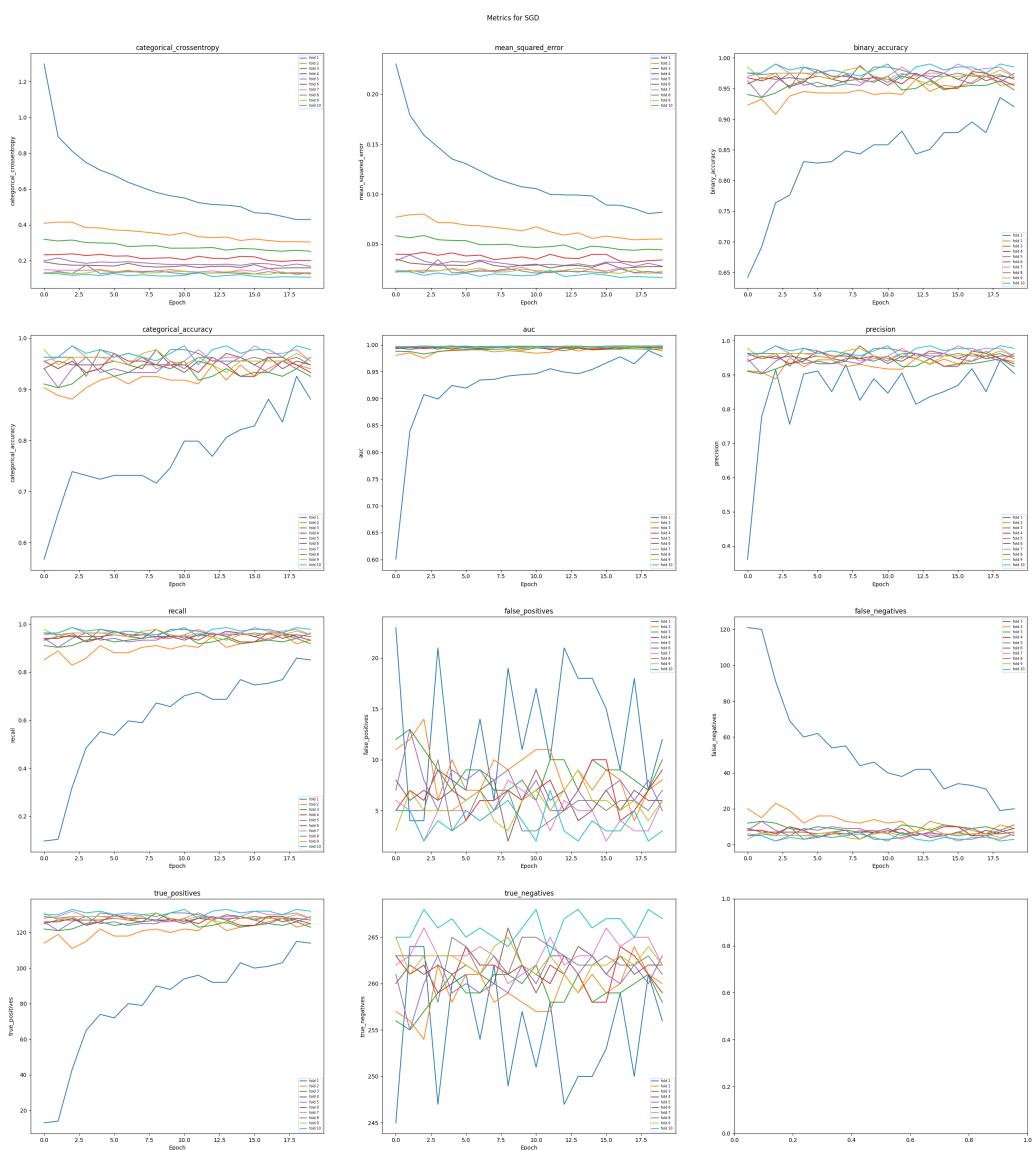


Figure 1: Wykres dla datasetu Iris z optymalizatorem SGD

6.1.2 Iris dla optymalizatora RMSprop

Iris classification model for optimizer: RMSprop

Metric: categorical_crossentropy → Score: 0.131389781832695

Metric: mean_squared_error → Score: 0.0314207598567009

Metric: binary_accuracy → Score: 0.9523809552192688

Metric: categorical_accuracy → Score: 0.9285714030265808

Metric: auc → Score: 0.9974489212036133

Metric: precision → Score: 0.9285714030265808

Metric: recall → Score: 0.9285714030265808

Metric: false_positives → Score: 1.0

Metric: false_negatives → Score: 1.0

Metric: true_positives → Score: 13.0

Metric: true_negatives → Score: 27.0

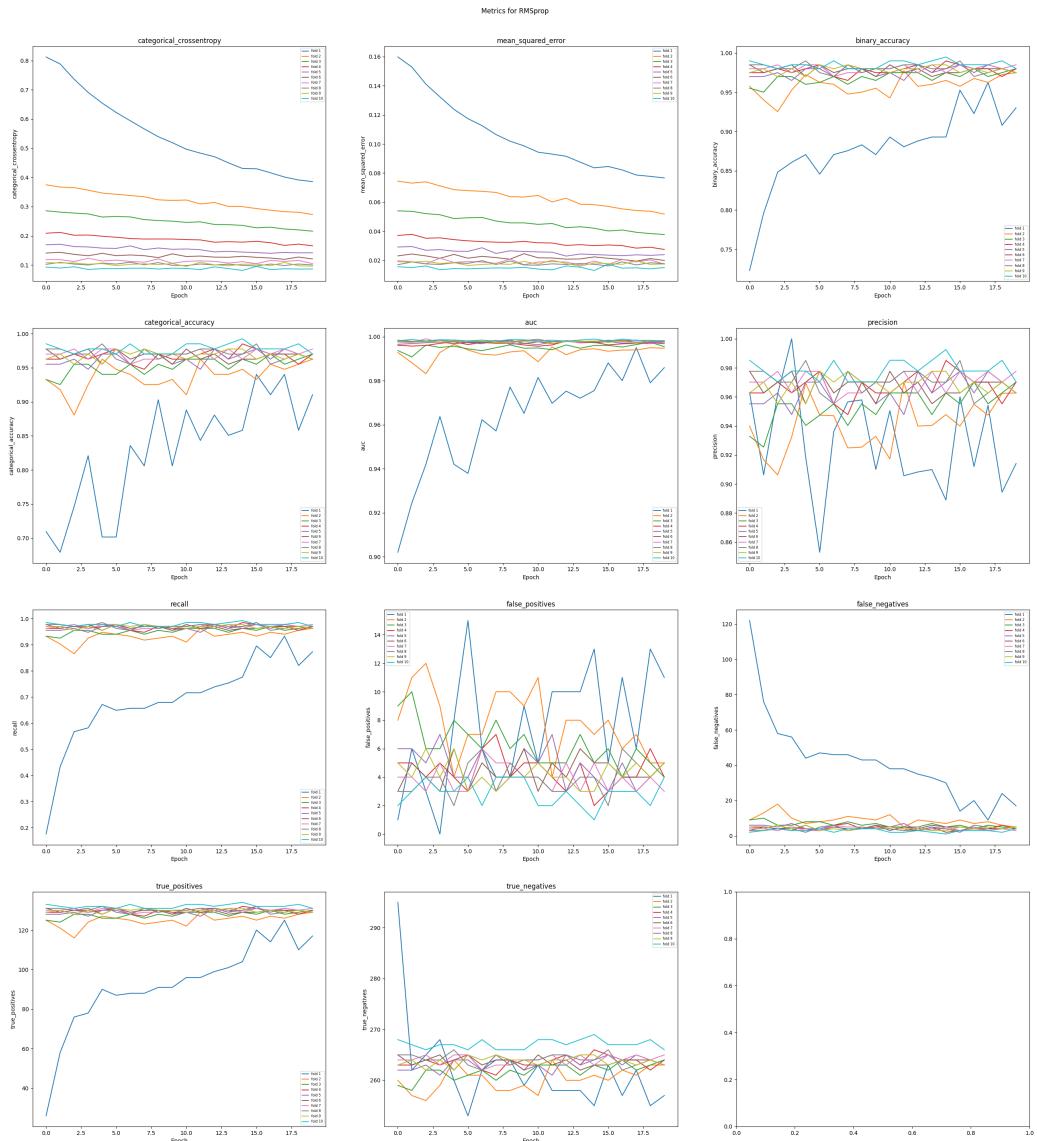


Figure 2: Wykres dla datasetu Iris z optymalizatorem RMSprop

6.1.3 Iris dla optymalizatora Adam

Iris classification model for optimizer: Adam
 Metric: categorical_crossentropy → Score: 0.09612463414669037
 Metric: mean_squared_error → Score: 0.01908824034035206
 Metric: binary_accuracy → Score: 0.9523809552192688
 Metric: categorical_accuracy → Score: 0.9285714030265808
 Metric: auc → Score: 0.9974489212036133
 Metric: precision → Score: 0.9285714030265808
 Metric: recall → Score: 0.9285714030265808
 Metric: false_positives → Score: 1.0
 Metric: false_negatives → Score: 1.0
 Metric: true_positives → Score: 13.0
 Metric: true_negatives → Score: 27.0

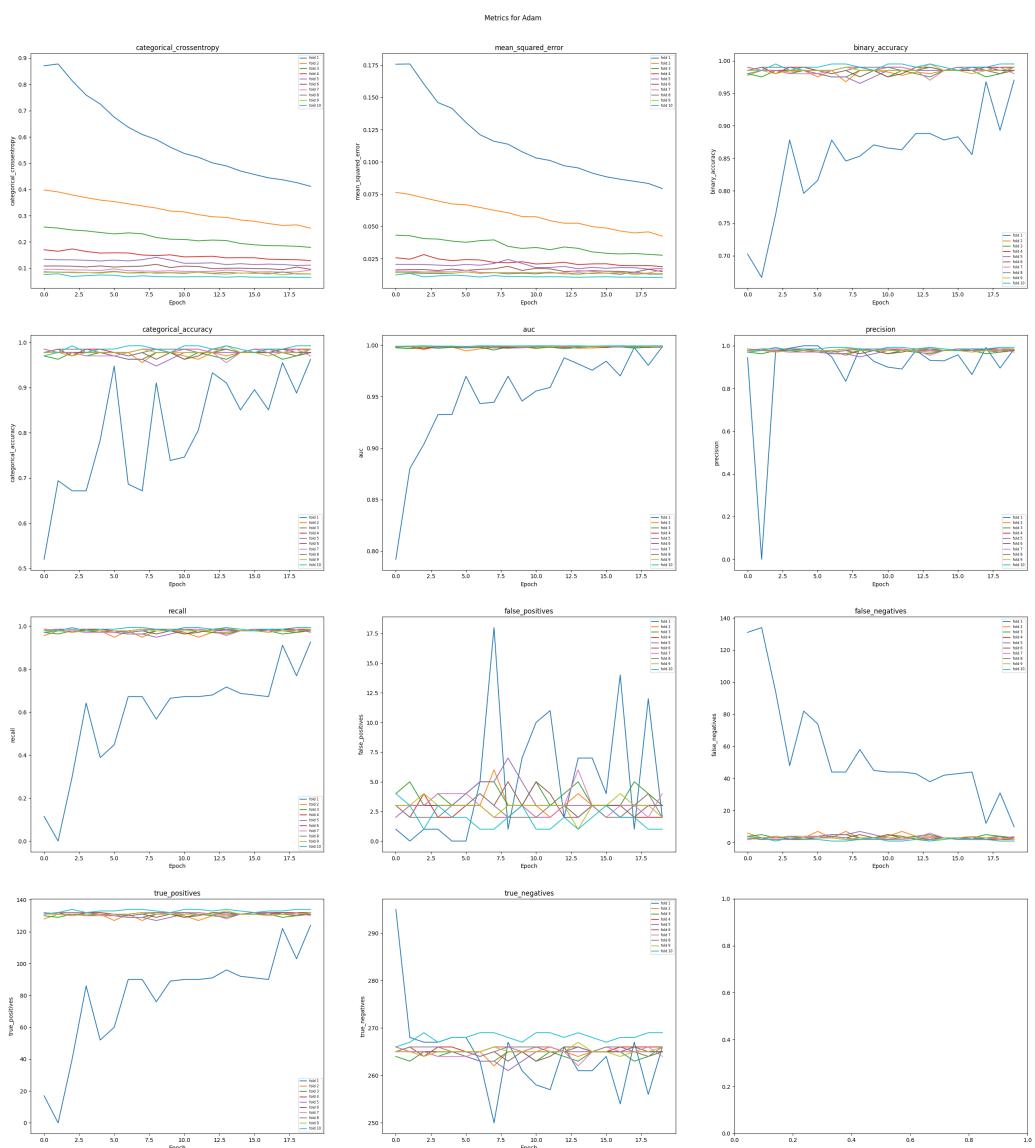


Figure 3: Wykres dla datasetu Iris z optymalizatorem Adam

6.1.4 Iris dla optymalizatora Adadelta

Iris classification model for optimizer: Adadelta

Metric: categorical_crossentropy → Score: 1.3804852962493896

Metric: mean_squared_error → Score: 0.2771957516670227

Metric: binary_accuracy → Score: 0.6666666865348816

Metric: categorical_accuracy → Score: 0.0

Metric: auc → Score: 0.13520407676696777

Metric: precision → Score: 0.0

Metric: recall → Score: 0.0

Metric: false_positives → Score: 0.0

Metric: false_negatives → Score: 14.0

Metric: true_positives → Score: 0.0

Metric: true_negatives → Score: 28.0

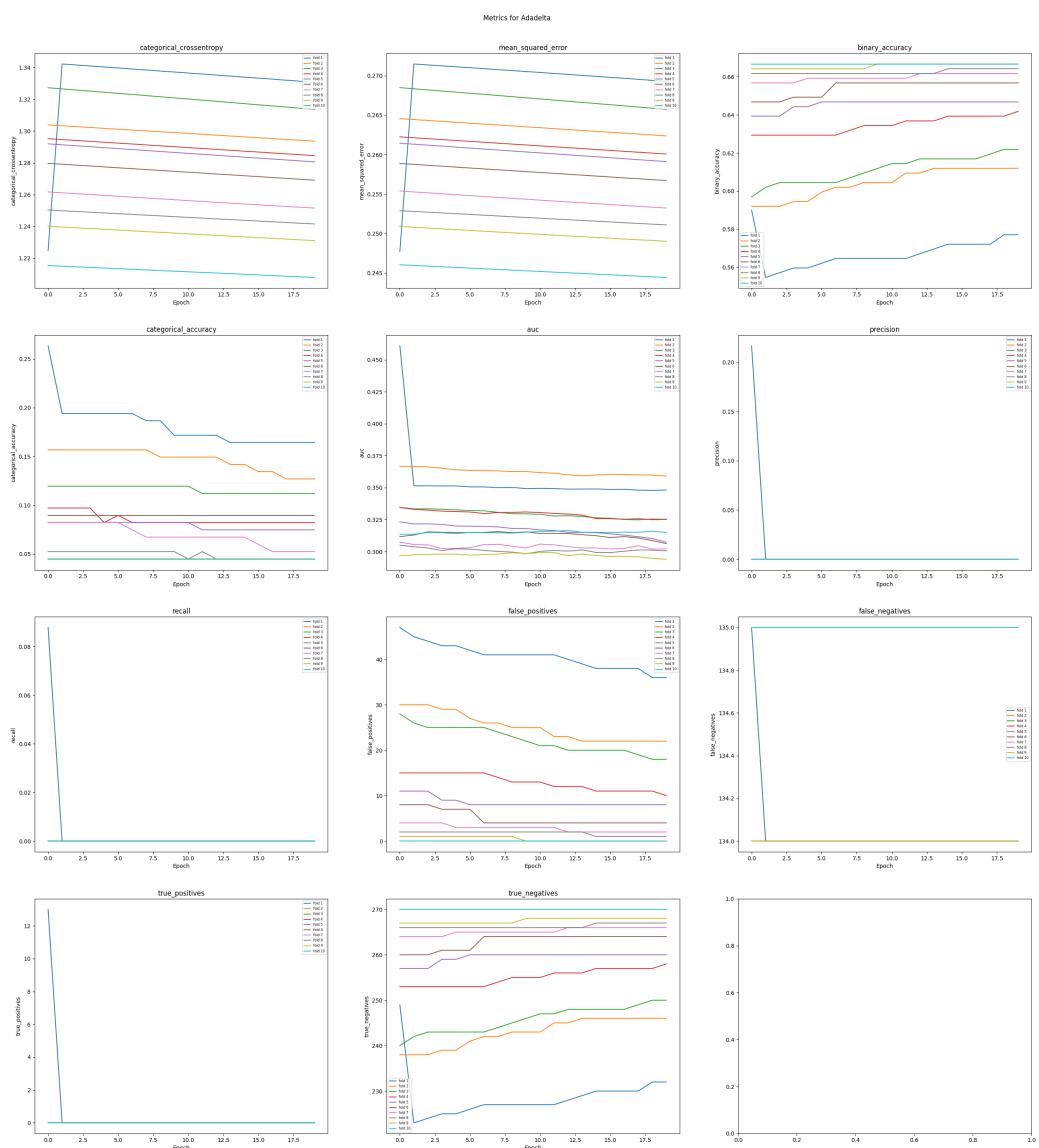


Figure 4: Wykres dla datasetu Iris z optymalizatorem Adadelta

6.1.5 Iris dla optymalizatora Adagrad

Iris classification model for optimizer: Adagrad

Metric: categorical_crossentropy → Score: 0.7094827890396118

Metric: mean_squared_error → Score: 0.13731861114501953

Metric: binary_accuracy → Score: 0.8571428656578064

Metric: categorical_accuracy → Score: 0.8571428656578064

Metric: auc → Score: 0.9642857313156128

Metric: precision → Score: 1.0

Metric: recall → Score: 0.5714285969734192

Metric: false_positives → Score: 0.0

Metric: false_negatives → Score: 6.0

Metric: true_positives → Score: 8.0

Metric: true_negatives → Score: 28.0

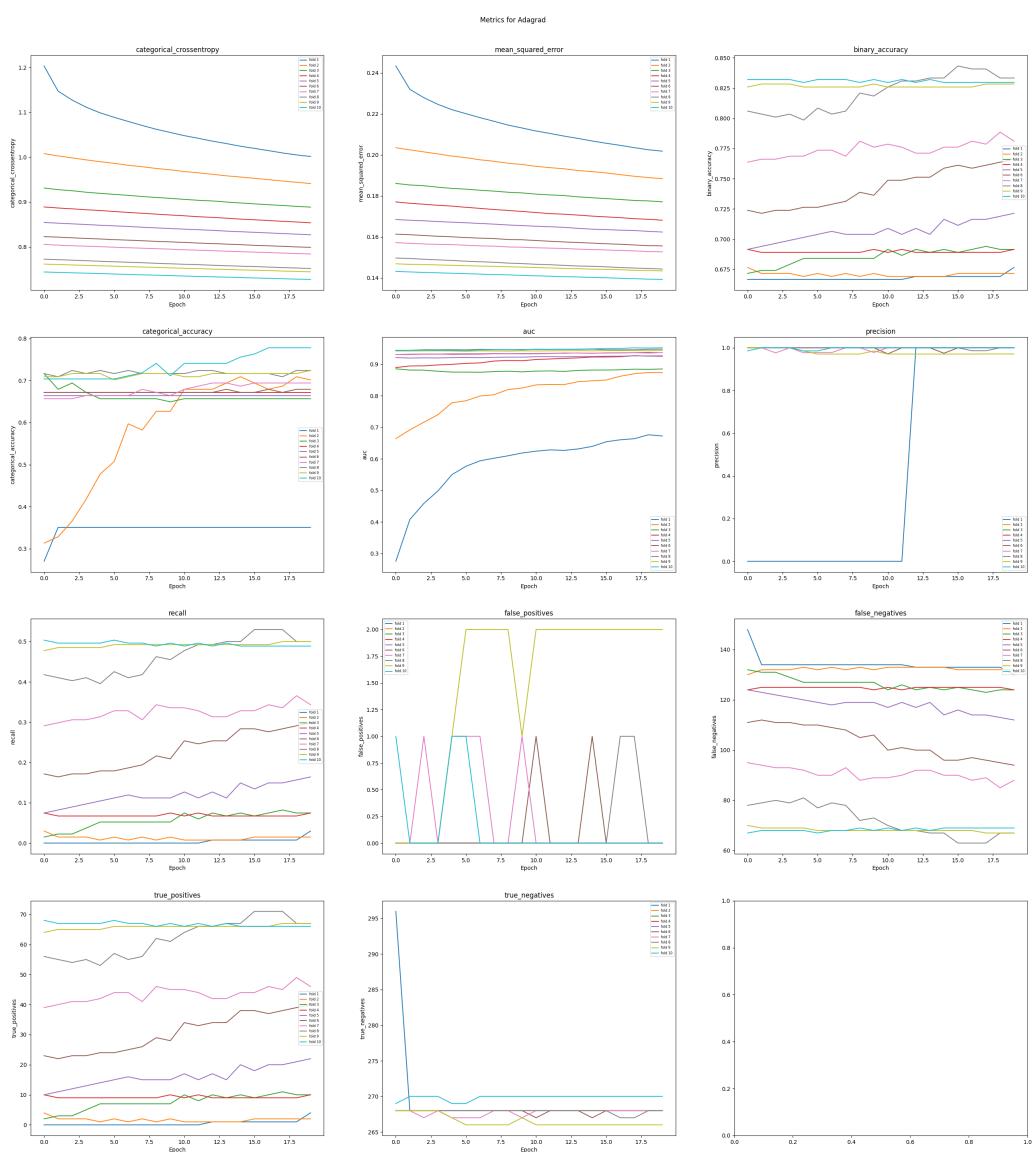


Figure 5: Wykres dla datasetu Iris z optymalizatorem Adagrad

6.1.6 Iris dla optymalizatora Adamax

Iris classification model for optimizer: Adamax

Metric: categorical_crossentropy → Score: 0.19120649993419647

Metric: mean_squared_error → Score: 0.03200673684477806

Metric: binary_accuracy → Score: 0.9523809552192688

Metric: categorical_accuracy → Score: 0.9285714030265808

Metric: auc → Score: 0.9974489212036133

Metric: precision → Score: 0.9285714030265808

Metric: recall → Score: 0.9285714030265808

Metric: false_positives → Score: 1.0

Metric: false_negatives → Score: 1.0

Metric: true_positives → Score: 13.0

Metric: true_negatives → Score: 27.0

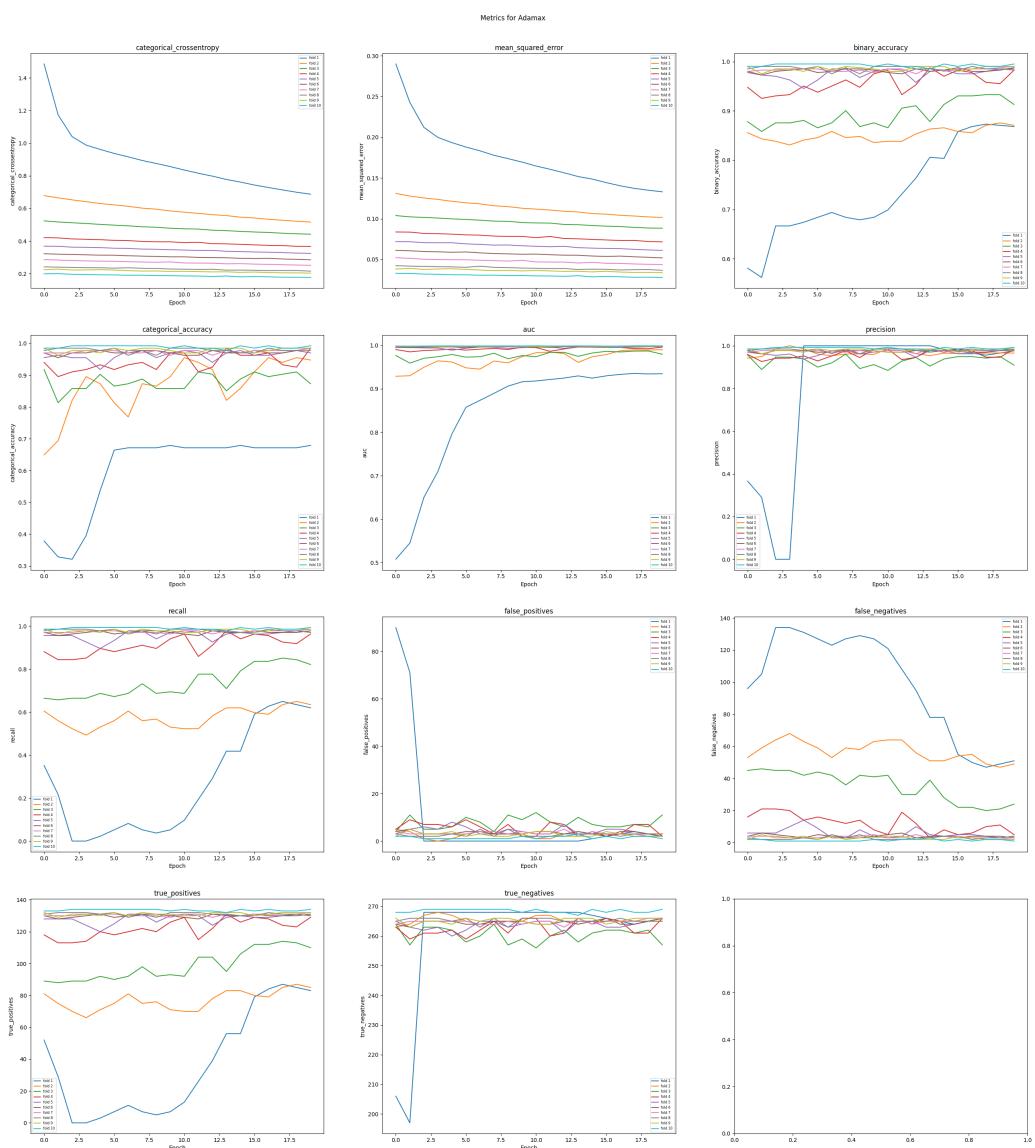


Figure 6: Wykres dla datasetu Iris z optymalizatorem Adamax

6.1.7 Iris dla optymalizatora Nadam

Iris classification model for optimizer: Nadam

Metric: categorical_crossentropy → Score: 0.09082233905792236

Metric: mean_squared_error → Score: 0.016807595267891884

Metric: binary_accuracy → Score: 0.9523809552192688

Metric: categorical_accuracy → Score: 0.9285714030265808

Metric: auc → Score: 0.9974489212036133

Metric: precision → Score: 0.9285714030265808

Metric: recall → Score: 0.9285714030265808

Metric: false_positives → Score: 1.0

Metric: false_negatives → Score: 1.0

Metric: true_positives → Score: 13.0

Metric: true_negatives → Score: 27.0

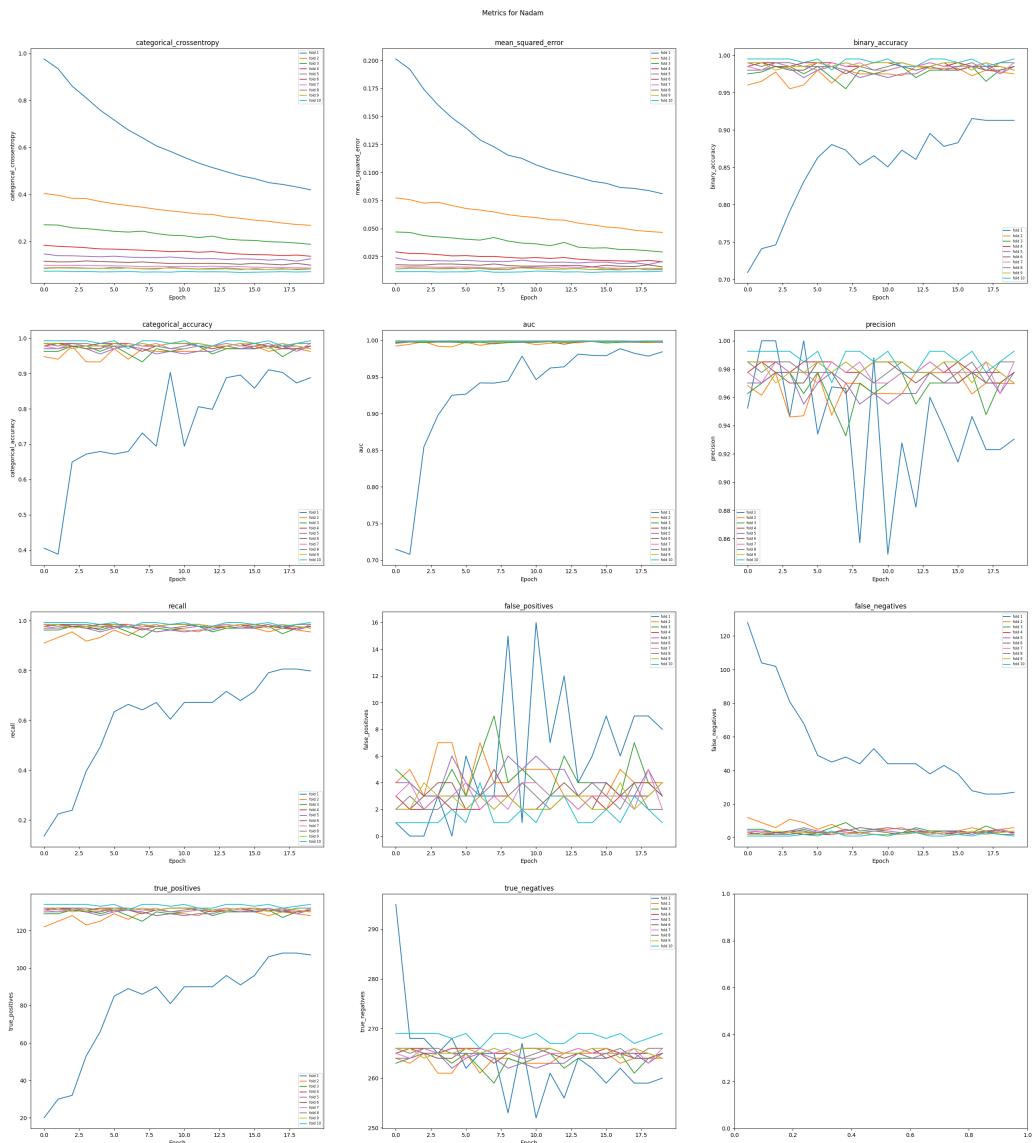


Figure 7: Wykres dla datasetu Iris z optymalizatorem Nadam

6.1.8 Iris dla optymalizatora Ftrl

Iris classification model for optimizer: Ftrl
 Metric: categorical_crossentropy → Score: 0.8582733273506165
 Metric: mean_squared_error → Score: 0.1699897050857544
 Metric: binary_accuracy → Score: 0.6666666865348816
 Metric: categorical_accuracy → Score: 0.7142857313156128
 Metric: auc → Score: 0.8966836929321289
 Metric: precision → Score: 0.0
 Metric: recall → Score: 0.0
 Metric: false_positives → Score: 0.0
 Metric: false_negatives → Score: 14.0
 Metric: true_positives → Score: 0.0
 Metric: true_negatives → Score: 28.0

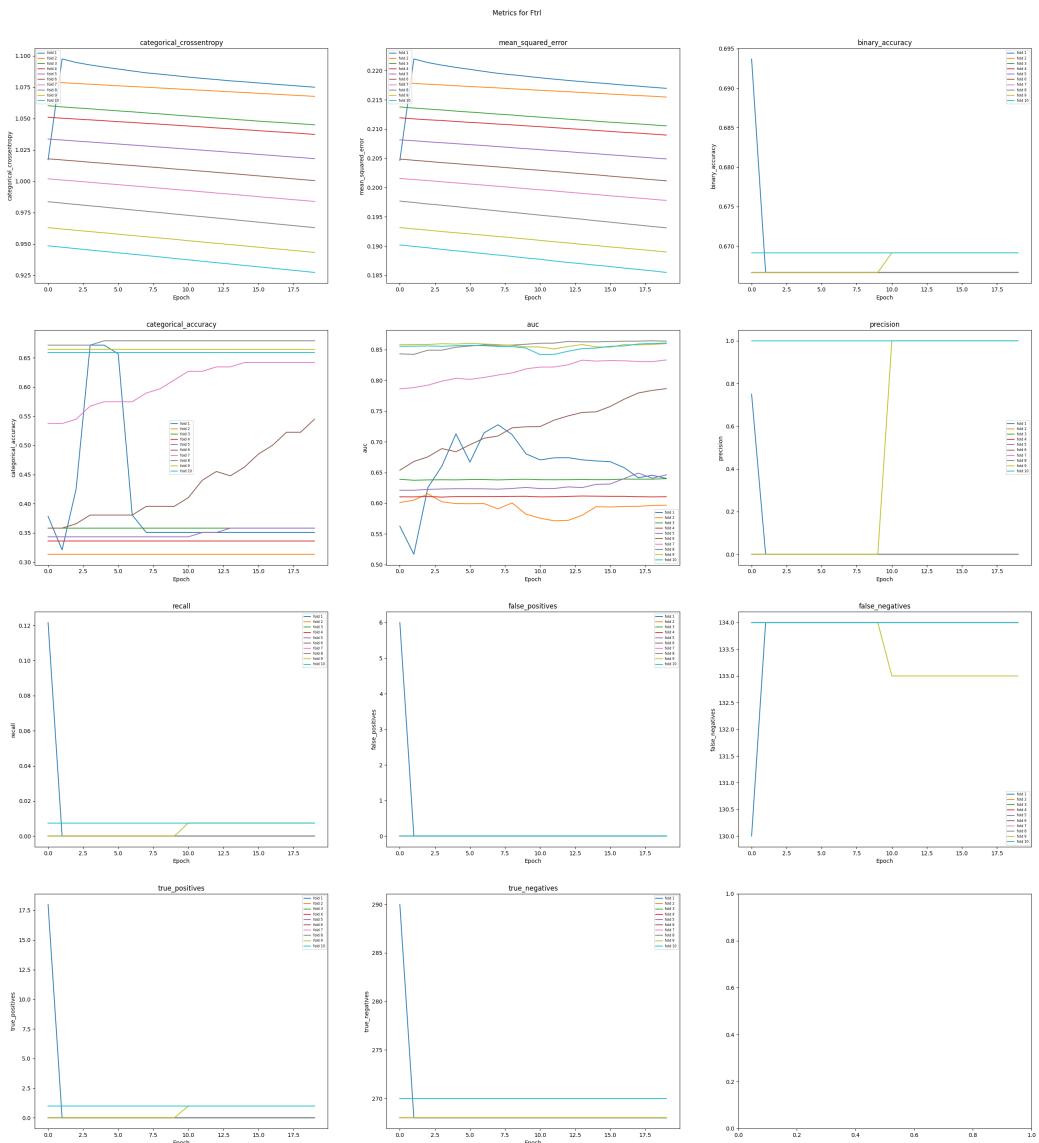


Figure 8: Wykres dla datasetu Iris z optymalizatorem Ftrl

Najlepszym modelem klasyfikacji dla zestawu danych Iris, biorąc pod uwagę wartość metryk, jest model używający optymalizatora Nadam. Ten model osiągnął najniższą wartość straty (loss: 0.09082233905792236) i błąd kwadratowy (mean_squared_error: 0.016807595267891884) w porównaniu do innych modeli.

Najgorszym modelem klasyfikacji dla zestawu danych Iris jest model używający optymalizatora Adadelta. Ten model osiągnął najwyższą wartość straty (loss: 1.3804852962493896) i błąd kwadratowy (mean_squared_error: 0.2771957516670227). Dodatkowo, model ten nie był w stanie poprawnie klasyfikować żadnych próbek (categorical_accuracy: 0.0, precision: 0.0, recall: 0.0).

6.2 CIFAR

6.2.1 CIFAR dla optymalizatora SGD

CIFAR classification model for optimizer: SGD

Metric: categorical_crossentropy → Score: 0.028873000293970108

Metric: mean_squared_error → Score: 0.001569483894854784

Metric: binary_accuracy → Score: 0.9978799819946289

Metric: categorical_accuracy → Score: 0.9891999959945679

Metric: auc → Score: 0.999836266040802

Metric: precision → Score: 0.9895958304405212

Metric: recall → Score: 0.9891999959945679

Metric: false_positives → Score: 52.0

Metric: false_negatives → Score: 54.0

Metric: true_positives → Score: 4946.0

Metric: true_negatives → Score: 44948.0

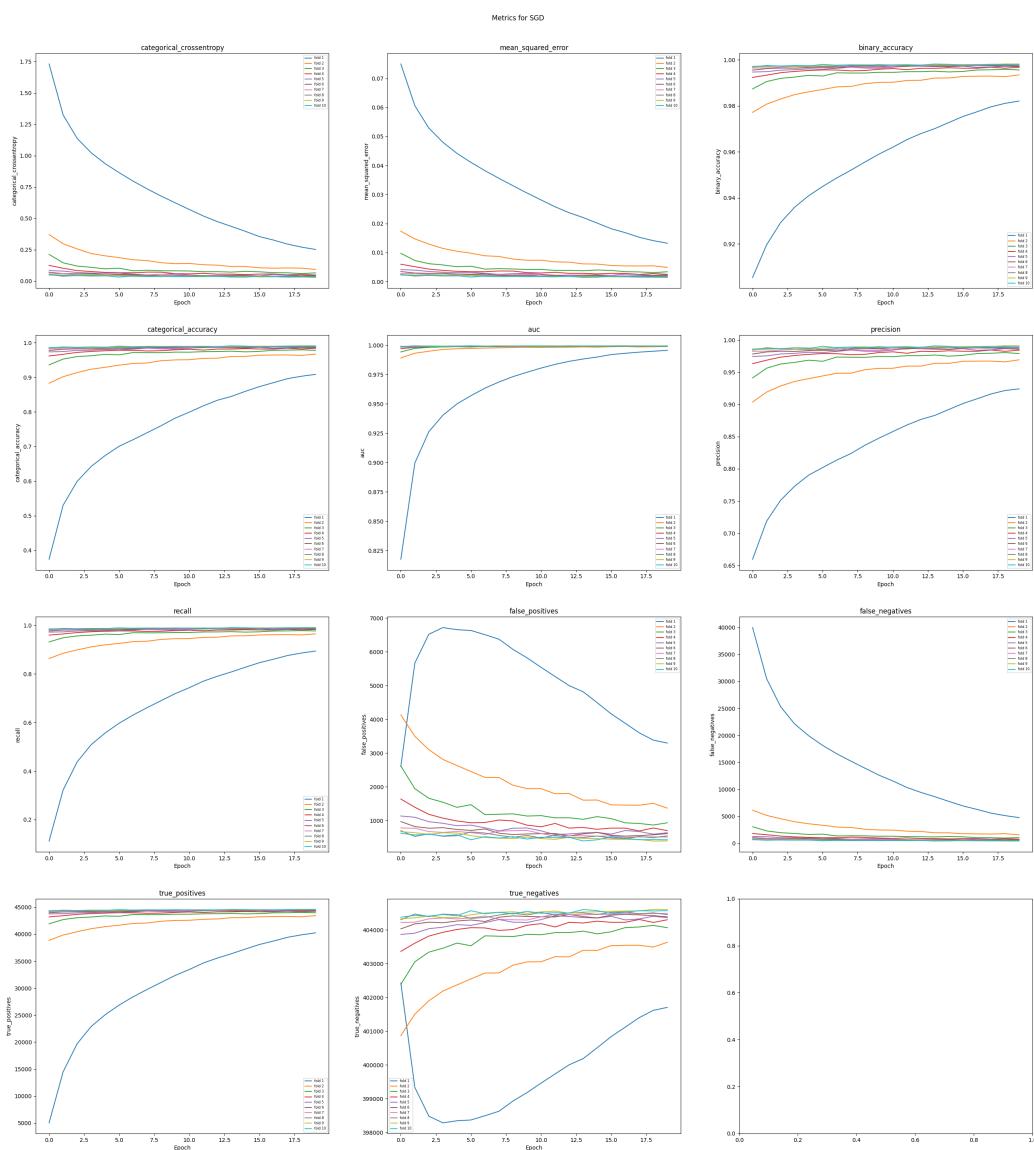


Figure 9: Wykres dla datasetu CIFAR z optymalizatorem SGD

6.2.2 CIFAR dla optymalizatora RMSprop

CIFAR classification model for optimizer: RMSprop

Metric: categorical_crossentropy → Score: 1.5543184280395508
 Metric: mean_squared_error → Score: 0.06954110413789749
 Metric: binary_accuracy → Score: 0.9105799794197083
 Metric: categorical_accuracy → Score: 0.428600013256073
 Metric: auc → Score: 0.8596146106719971
 Metric: precision → Score: 0.722082257270813
 Metric: recall → Score: 0.1720000058412552
 Metric: false_positives → Score: 331.0
 Metric: false_negatives → Score: 4140.0
 Metric: true_positives → Score: 860.0
 Metric: true_negatives → Score: 44669.0

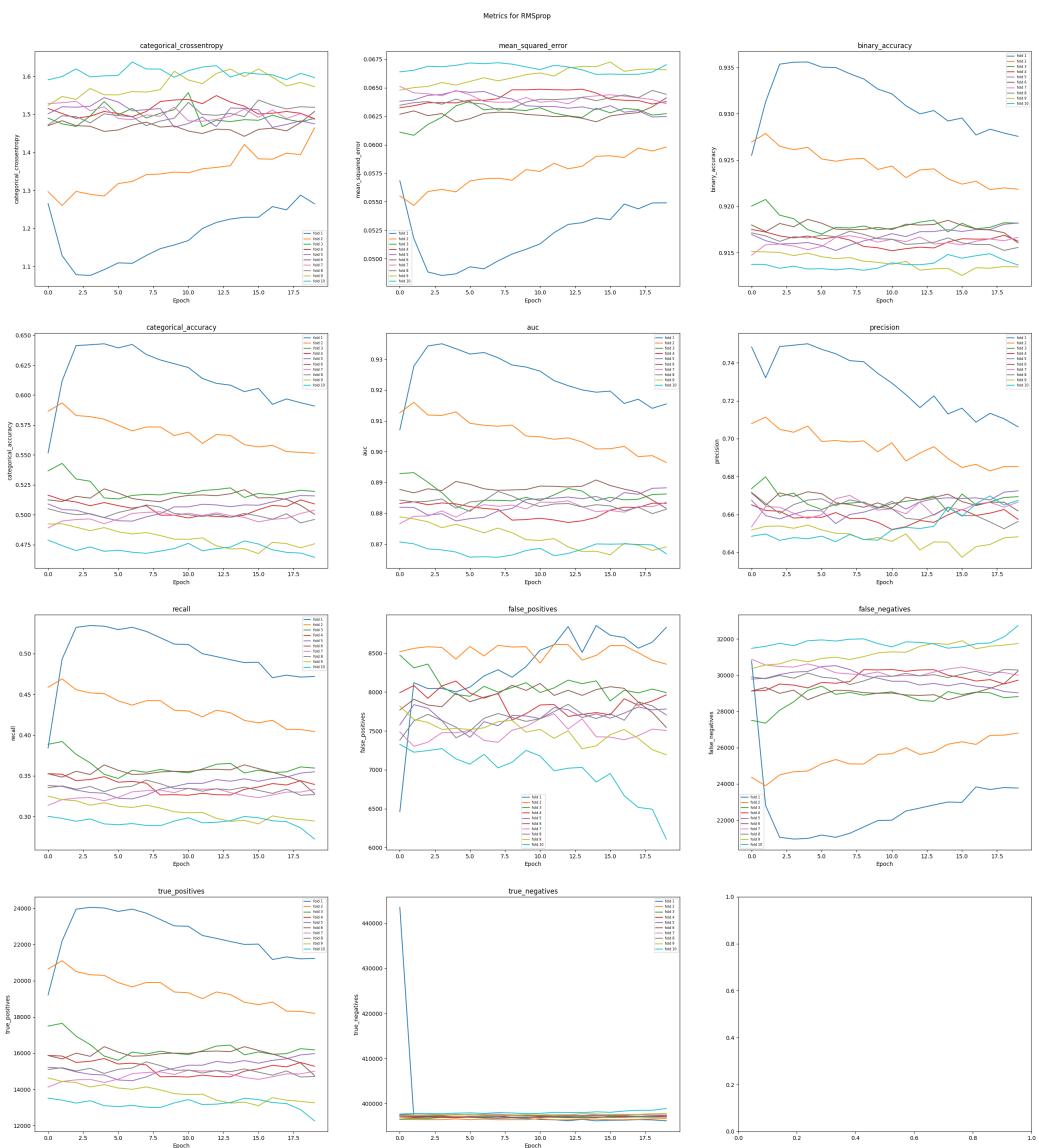


Figure 10: Wykres dla datasetu CIFAR z optymalizatorem RMSprop

6.2.3 CIFAR dla optymalizatora Adam

CIFAR classification model for optimizer: Adam

Metric: categorical_crossentropy → Score: 0.48483413457870483

Metric: mean_squared_error → Score: 0.024553101509809494

Metric: binary_accuracy → Score: 0.9671199917793274

Metric: categorical_accuracy → Score: 0.8267999887466431

Metric: auc → Score: 0.984446108341217

Metric: precision → Score: 0.8555084466934204

Metric: recall → Score: 0.8076000213623047

Metric: false_positives → Score: 682.0

Metric: false_negatives → Score: 962.0

Metric: true_positives → Score: 4038.0

Metric: true_negatives → Score: 44318.0

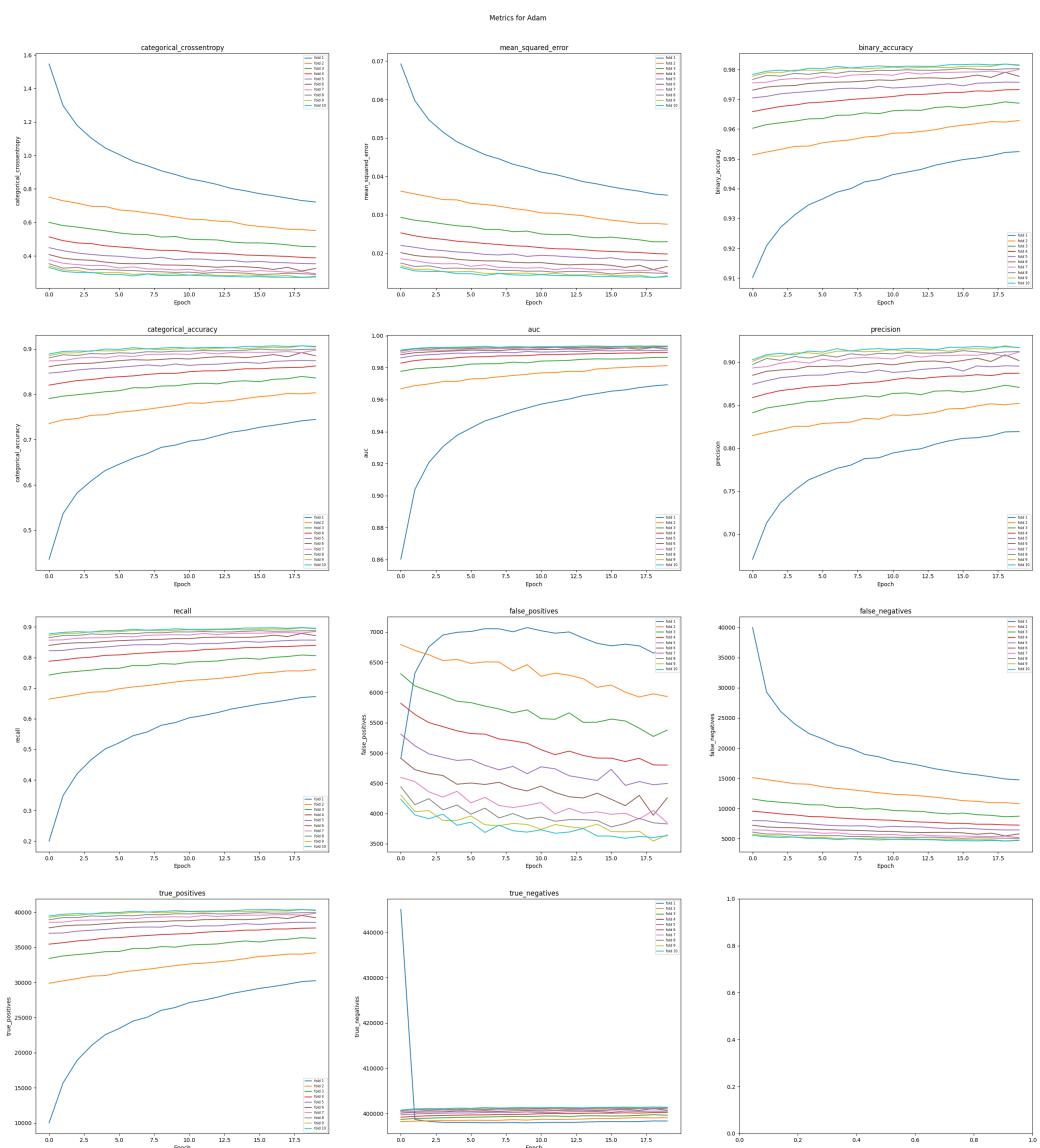


Figure 11: Wykres dla datasetu CIFAR z optymalizatorem Adam

6.2.4 CIFAR dla optymalizatora Adadelta

CIFAR classification model for optimizer: Adadelta

Metric: categorical_crossentropy → Score: 1.3545889854431152
 Metric: mean_squared_error → Score: 0.061867017298936844
 Metric: binary_accuracy → Score: 0.91566002368927
 Metric: categorical_accuracy → Score: 0.5555999875068665
 Metric: auc → Score: 0.9068788290023804
 Metric: precision → Score: 0.8549410700798035
 Metric: recall → Score: 0.18860000371932983
 Metric: false_positives → Score: 160.0
 Metric: false_negatives → Score: 4057.0
 Metric: true_positives → Score: 943.0
 Metric: true_negatives → Score: 44840.0

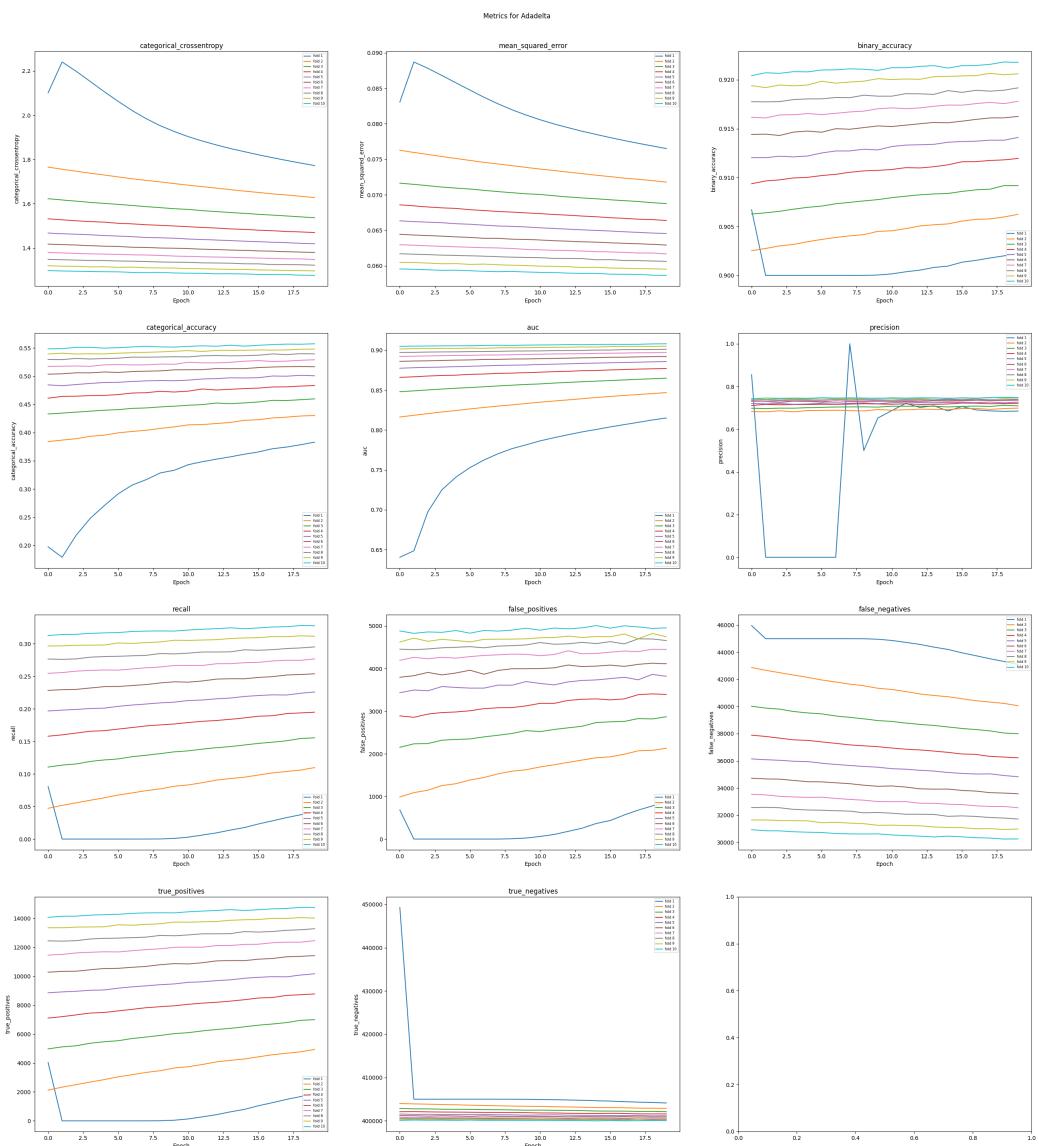


Figure 12: Wykres dla datasetu CIFAR z optymalizatorem Adadelta

6.2.5 CIFAR dla optymalizatora Adagrad

CIFAR classification model for optimizer: Adagrad

```
Metric: categorical_crossentropy -> Score: 1.0685112476348877
Metric: mean_squared_error -> Score: 0.04945021867752075
Metric: binary_accuracy -> Score: 0.9327200055122375
Metric: categorical_accuracy -> Score: 0.6636000275611877
Metric: auc -> Score: 0.9458017945289612
Metric: precision -> Score: 0.8862134218215942
Metric: recall -> Score: 0.37540000677108765
Metric: false_positives -> Score: 241.0
Metric: false_negatives -> Score: 3123.0
Metric: true_positives -> Score: 1877.0
Metric: true_negatives -> Score: 44759.0
```

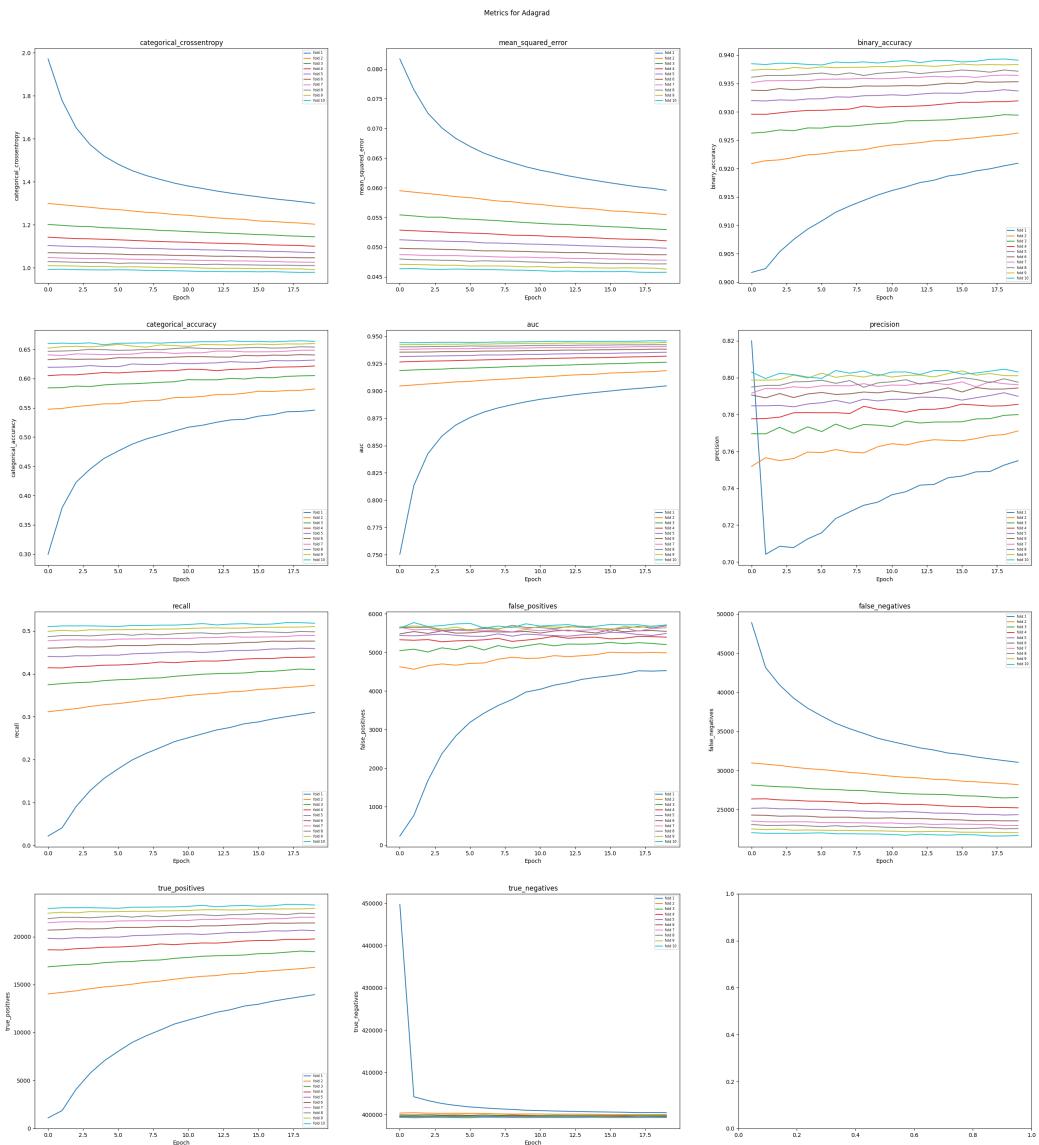


Figure 13: Wykres dla datasetu CIFAR z optymalizatorem Adagrad

6.2.6 CIFAR dla optymalizatora Adamax

CIFAR classification model for optimizer: Adamax

Metric: categorical_crossentropy → Score: 0.007766671944409609

Metric: mean_squared_error → Score: 0.0003209836722817272

Metric: binary_accuracy → Score: 0.9996399879455566

Metric: categorical_accuracy → Score: 0.998199999332428

Metric: auc → Score: 0.9998959898948669

Metric: precision → Score: 0.998199999332428

Metric: recall → Score: 0.998199999332428

Metric: false_positives → Score: 9.0

Metric: false_negatives → Score: 9.0

Metric: true_positives → Score: 4991.0

Metric: true_negatives → Score: 44991.0

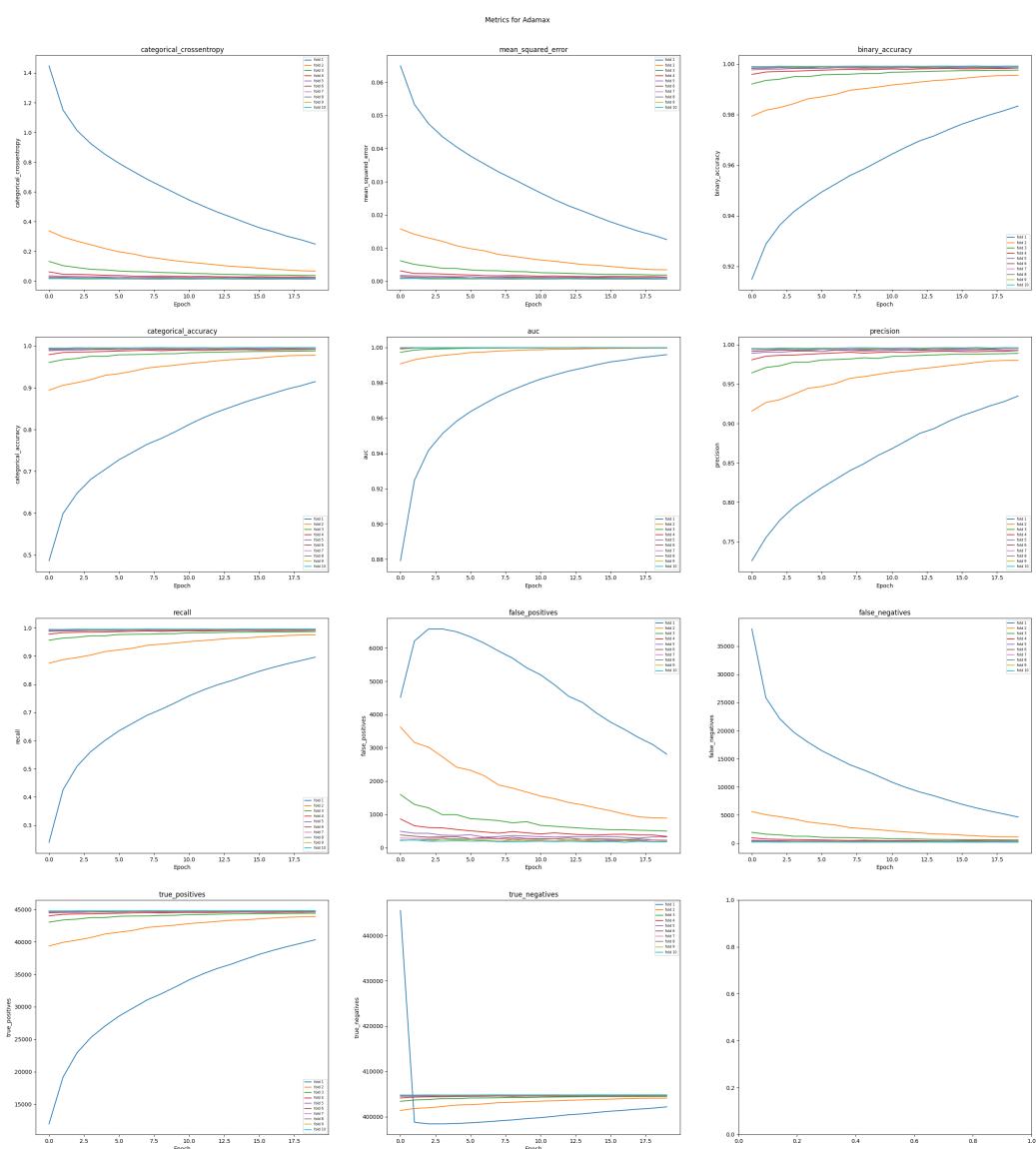


Figure 14: Wykres dla datasetu CIFAR z optymalizatorem Adamax

6.2.7 CIFAR dla optymalizatora Nadam

CIFAR classification model for optimizer: Nadam

Metric: categorical_crossentropy → Score: 0.2687966227531433
 Metric: mean_squared_error → Score: 0.013372791931033134
 Metric: binary_accuracy → Score: 0.9826200008392334
 Metric: categorical_accuracy → Score: 0.9115999937057495
 Metric: auc → Score: 0.9935429096221924
 Metric: precision → Score: 0.9236922860145569
 Metric: recall → Score: 0.900600016117096
 Metric: false_positives → Score: 372.0
 Metric: false_negatives → Score: 497.0
 Metric: true_positives → Score: 4503.0
 Metric: true_negatives → Score: 44628.0

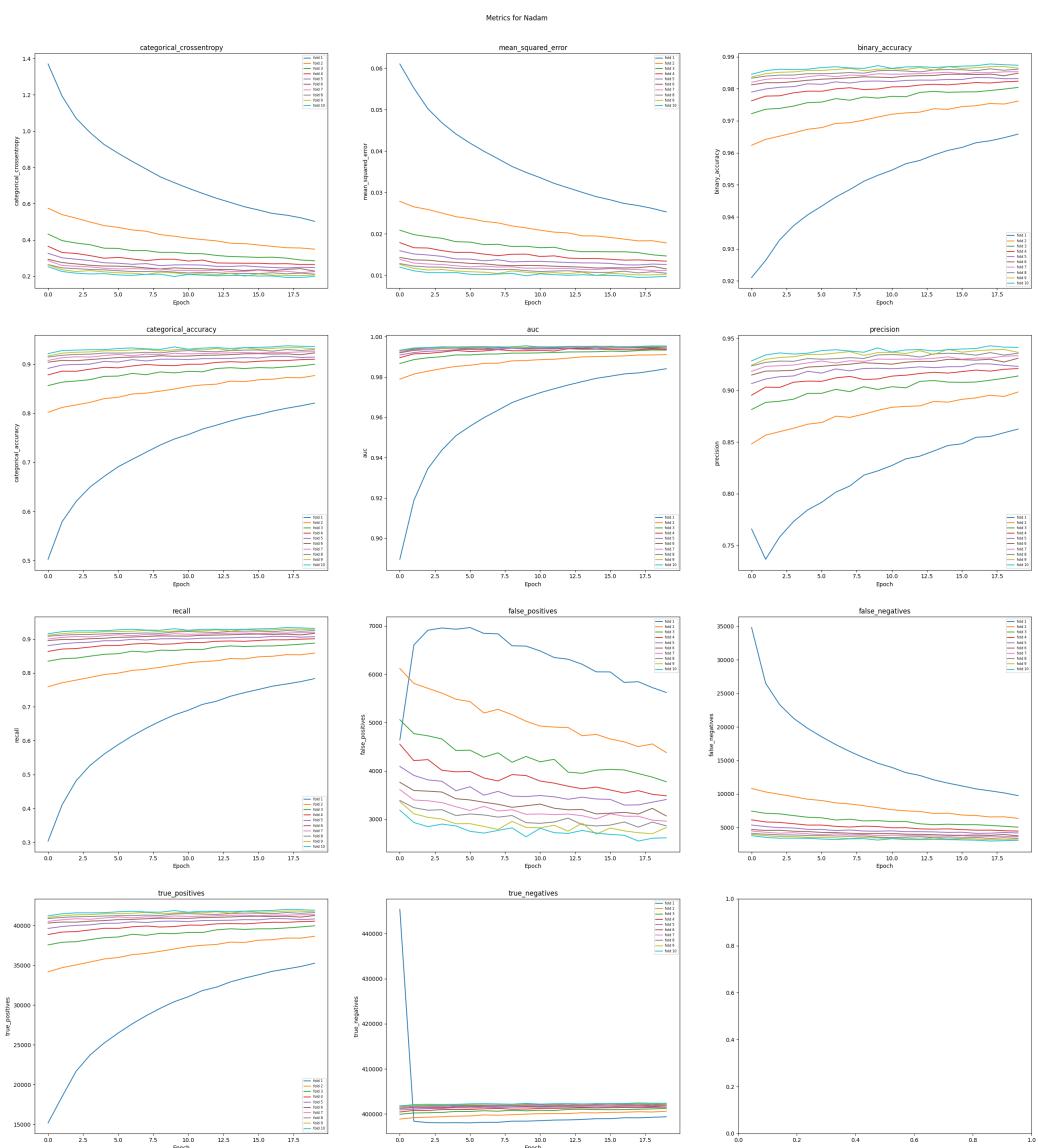


Figure 15: Wykres dla datasetu CIFAR z optymalizatorem Nadam

6.2.8 CIFAR dla optymalizatora Ftrl

CIFAR classification model for optimizer: Ftrl
 Metric: categorical_crossentropy → Score: 2.3025877475738525
 Metric: mean_squared_error → Score: 0.09000006318092346
 Metric: binary_accuracy → Score: 0.8999999761581421
 Metric: categorical_accuracy → Score: 0.09480000287294388
 Metric: auc → Score: 0.5
 Metric: precision → Score: 0.0
 Metric: recall → Score: 0.0
 Metric: false_positives → Score: 0.0
 Metric: false_negatives → Score: 5000.0
 Metric: true_positives → Score: 0.0
 Metric: true_negatives → Score: 45000.0

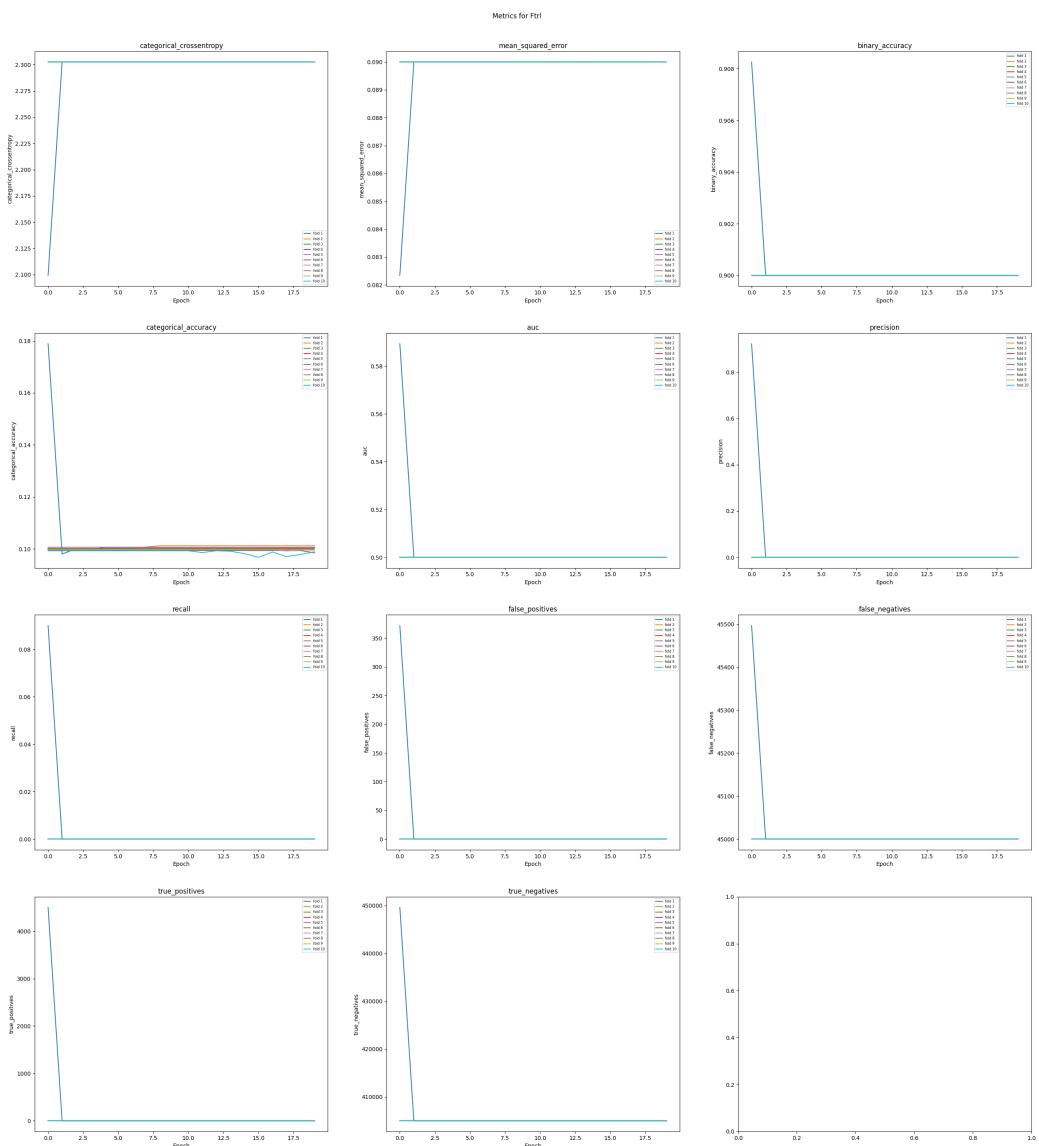


Figure 16: Wykres dla datasetu CIFAR z optymalizatorem Ftrl

Najlepszym optymalizatorem dla tego modelu, na podstawie podanych wyników, jest Adamax. Ten optymalizator uzyskał najwyższy wynik w metrykach 'loss', 'categorical crossentropy' oraz 'mean squared error', co oznacza,

że model z tym optymalizatorem popełnił najmniejszą liczbę błędów podczas uczenia. Dodatkowo, Adamax osiągnął najwyższe wyniki w metrykach 'binary accuracy', 'categorical accuracy', 'auc', 'precision', 'recall', co sugeruje, że model ten był najdokładniejszy i najefektywniejszy w klasyfikacji.

Najgorszym optymalizatorem dla tego modelu jest Ftrl. Ten optymalizator osiągnął najwyższe wyniki w metrykach 'loss', 'categorical crossentropy' oraz 'mean squared error', co oznacza, że model z tym optymalizatorem popełnił najwięcej błędów podczas uczenia. Ponadto, Ftrl uzyskał najniższe wyniki w metrykach 'binary accuracy', 'categorical accuracy', 'auc', 'precision', 'recall', co sugeruje, że model ten był najmniej dokładny i najmniej efektywny w klasyfikacji.

6.3 DryBean

6.3.1 DryBean dla optymalizatora SGD

DryBean classification model for optimizer: SGD

```
Metric: categorical_crossentropy -> Score: 1.4431248903274536
Metric: mean_squared_error -> Score: 0.10684068500995636
Metric: binary_accuracy -> Score: 0.8631258606910706
Metric: categorical_accuracy -> Score: 0.38207200169563293
Metric: auc -> Score: 0.8429846167564392
Metric: precision -> Score: 0.5373525619506836
Metric: recall -> Score: 0.30124908685684204
Metric: false_positives -> Score: 353.0
Metric: false_negatives -> Score: 951.0
Metric: true_positives -> Score: 410.0
Metric: true_negatives -> Score: 7813.0
```

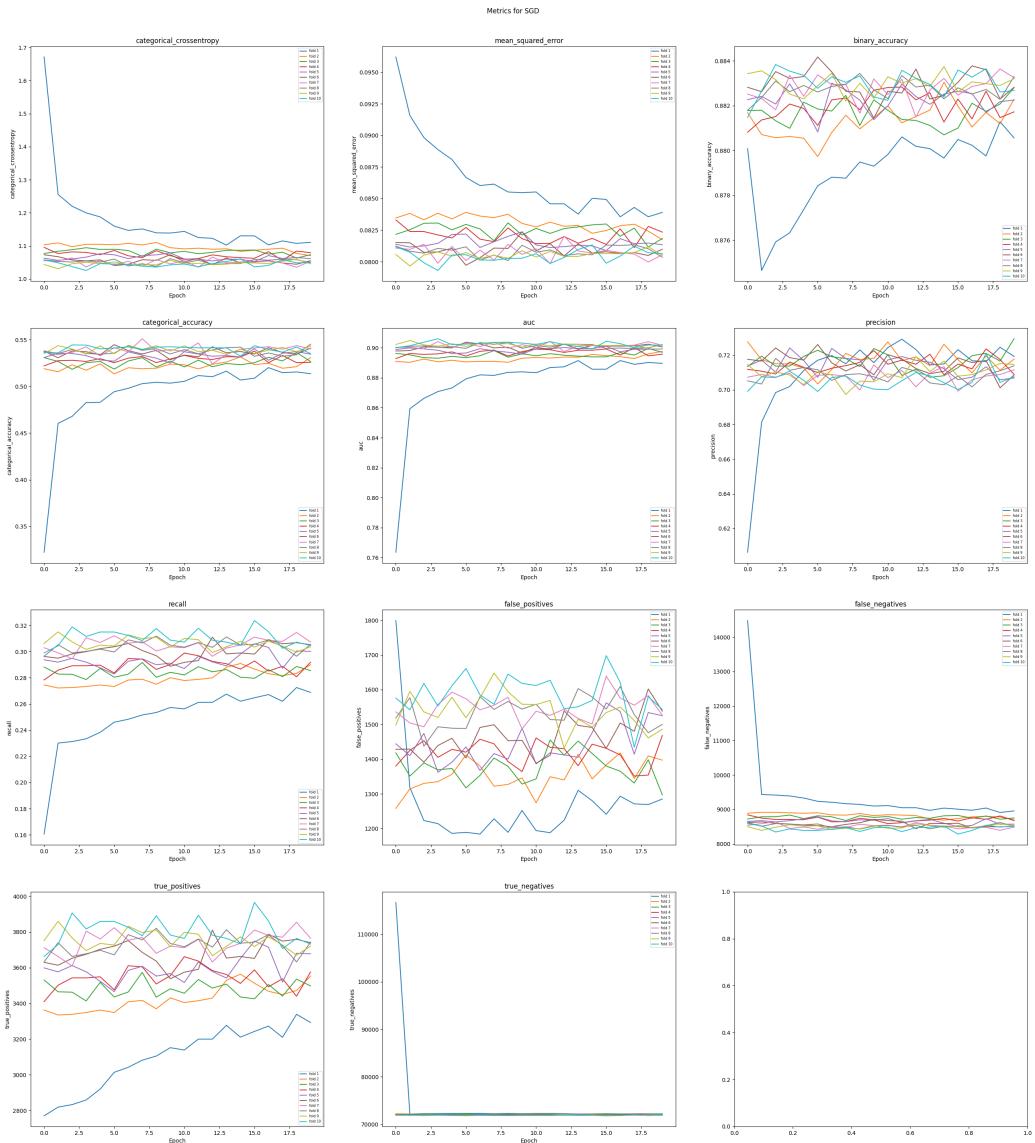


Figure 17: Wykres dla datasetu DryBean z optymalizatorem SGD

6.3.2 DryBean dla optymalizatora RMSprop

DryBean classification model for optimizer: RMSprop

Metric: categorical_crossentropy → Score: 11.778939247131348
 Metric: mean_squared_error → Score: 0.0837707668542862
 Metric: binary_accuracy → Score: 0.8783457279205322
 Metric: categorical_accuracy → Score: 0.5334312915802002
 Metric: auc → Score: 0.8973314166069031
 Metric: precision → Score: 0.6573208570480347
 Metric: recall → Score: 0.3100661337375641
 Metric: false_positives → Score: 220.0
 Metric: false_negatives → Score: 939.0
 Metric: true_positives → Score: 422.0
 Metric: true_negatives → Score: 7946.0

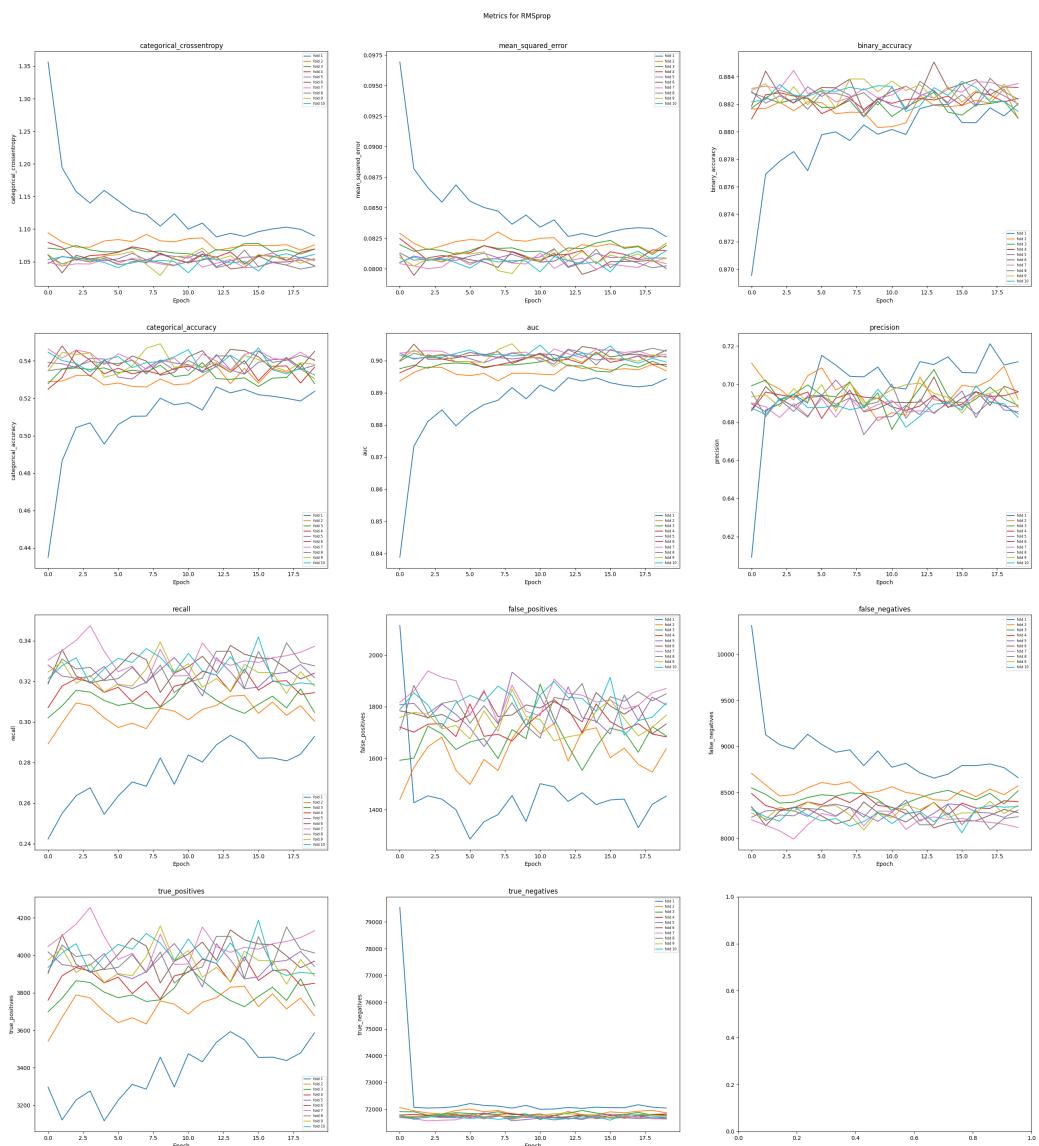


Figure 18: Wykres dla datasetu DryBean z optymalizatorem RMSprop

6.3.3 DryBean dla optymalizatora Adam

DryBean classification model for optimizer: Adam

Metric: categorical_crossentropy → Score: 1.1668554544448853
 Metric: mean_squared_error → Score: 0.08874128758907318
 Metric: binary_accuracy → Score: 0.8721528053283691
 Metric: categorical_accuracy → Score: 0.46142542362213135
 Metric: auc → Score: 0.8826119899749756
 Metric: precision → Score: 0.6043795347213745
 Metric: recall → Score: 0.3041881024837494
 Metric: false_positives → Score: 271.0
 Metric: false_negatives → Score: 947.0
 Metric: true_positives → Score: 414.0
 Metric: true_negatives → Score: 7895.0

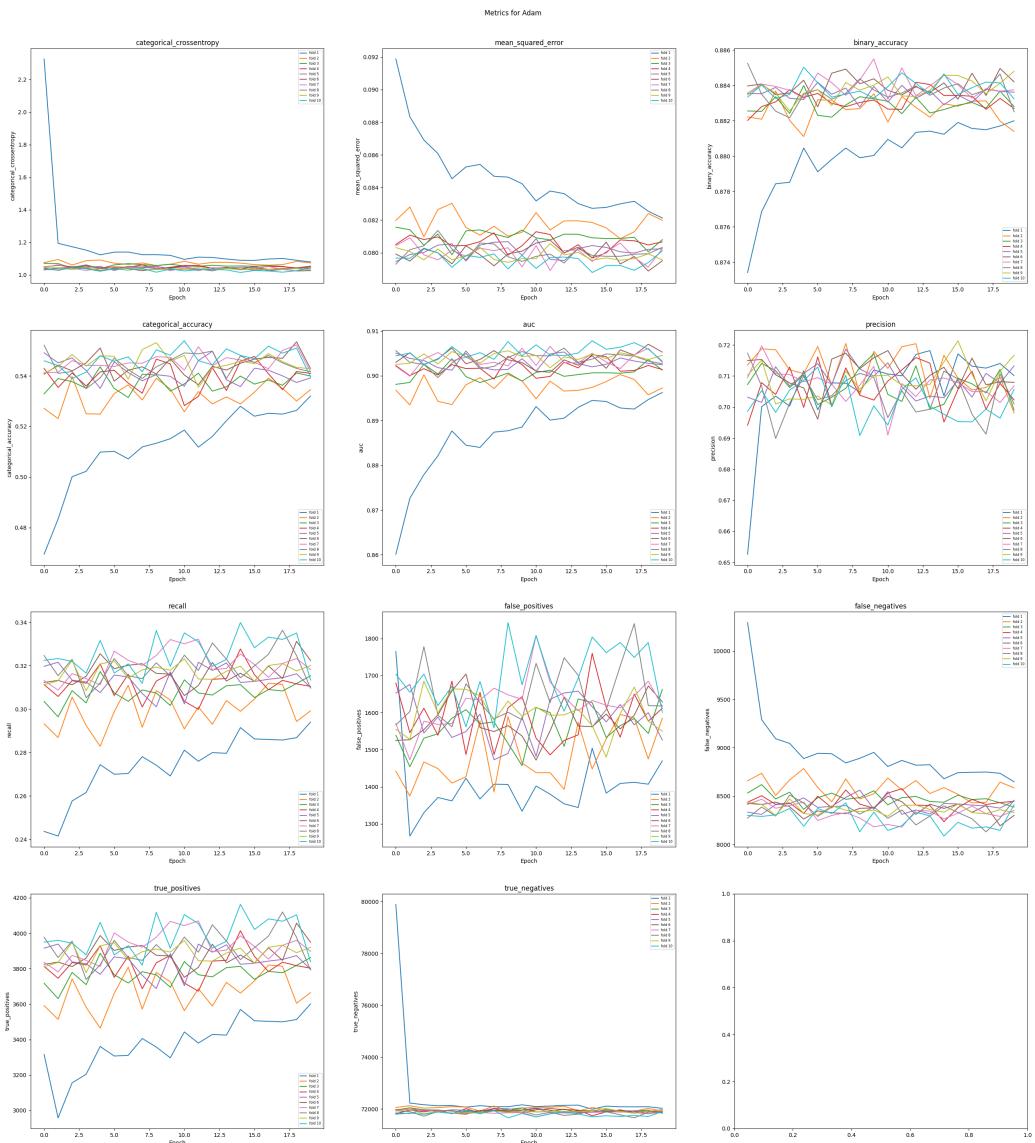


Figure 19: Wykres dla datasetu DryBean z optymalizatorem Adam

6.3.4 DryBean dla optymalizatora Adadelta

DryBean classification model for optimizer: Adadelta

```
Metric: categorical_crossentropy -> Score: 1.4357655048370361
Metric: mean_squared_error -> Score: 0.10437203198671341
Metric: binary_accuracy -> Score: 0.8609215617179871
Metric: categorical_accuracy -> Score: 0.4290962517261505
Metric: auc -> Score: 0.8407949209213257
Metric: precision -> Score: 0.5218446850776672
Metric: recall -> Score: 0.3159441649913788
Metric: false_positives -> Score: 394.0
Metric: false_negatives -> Score: 931.0
Metric: true_positives -> Score: 430.0
Metric: true_negatives -> Score: 7772.0
```

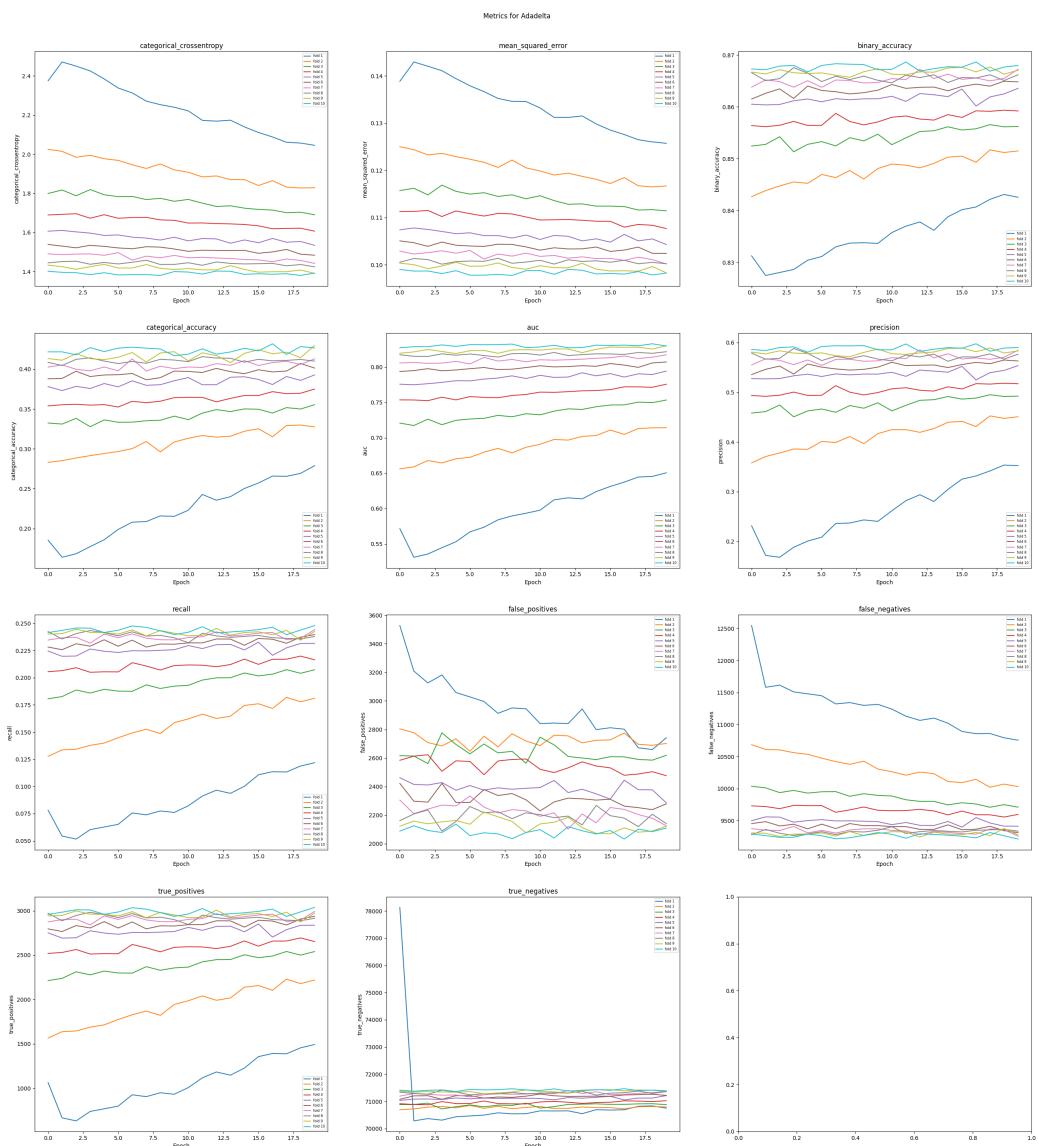


Figure 20: Wykres dla datasetu DryBean z optymalizatorem Adadelta

6.3.5 DryBean dla optymalizatora Adagrad

DryBean classification model for optimizer: Adagrad
 Metric: categorical_crossentropy → Score: 1.557380199432373
 Metric: mean_squared_error → Score: 0.1119057834148407
 Metric: binary_accuracy → Score: 0.8544137477874756
 Metric: categorical_accuracy → Score: 0.3659074306488037
 Metric: auc → Score: 0.8284960985183716
 Metric: precision → Score: 0.4848484992980957
 Metric: recall → Score: 0.30565759539604187
 Metric: false_positives → Score: 442.0
 Metric: false_negatives → Score: 945.0
 Metric: true_positives → Score: 416.0
 Metric: true_negatives → Score: 7724.0

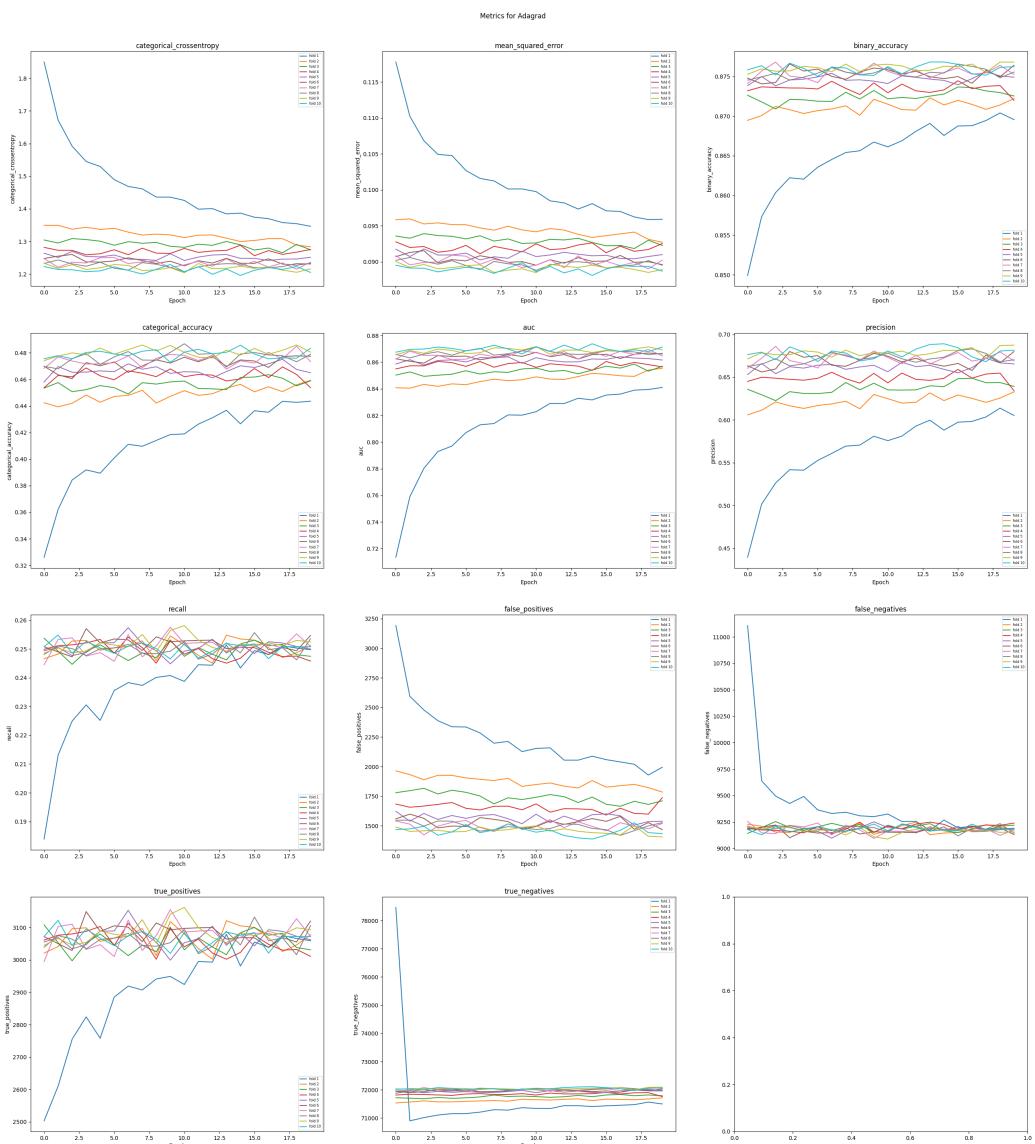


Figure 21: Wykres dla datasetu DryBean z optymalizatorem Adagrad

6.3.6 DryBean dla optymalizatora Adamax

DryBean classification model for optimizer: Adamax
 Metric: categorical_crossentropy → Score: 1.27748441696167
 Metric: mean_squared_error → Score: 0.09762240201234818
 Metric: binary_accuracy → Score: 0.8654350638389587
 Metric: categorical_accuracy → Score: 0.42027920484542847
 Metric: auc → Score: 0.8652811646461487
 Metric: precision → Score: 0.5509677529335022
 Metric: recall → Score: 0.3137398958206177
 Metric: false_positives → Score: 348.0
 Metric: false_negatives → Score: 934.0
 Metric: true_positives → Score: 427.0
 Metric: true_negatives → Score: 7818.0

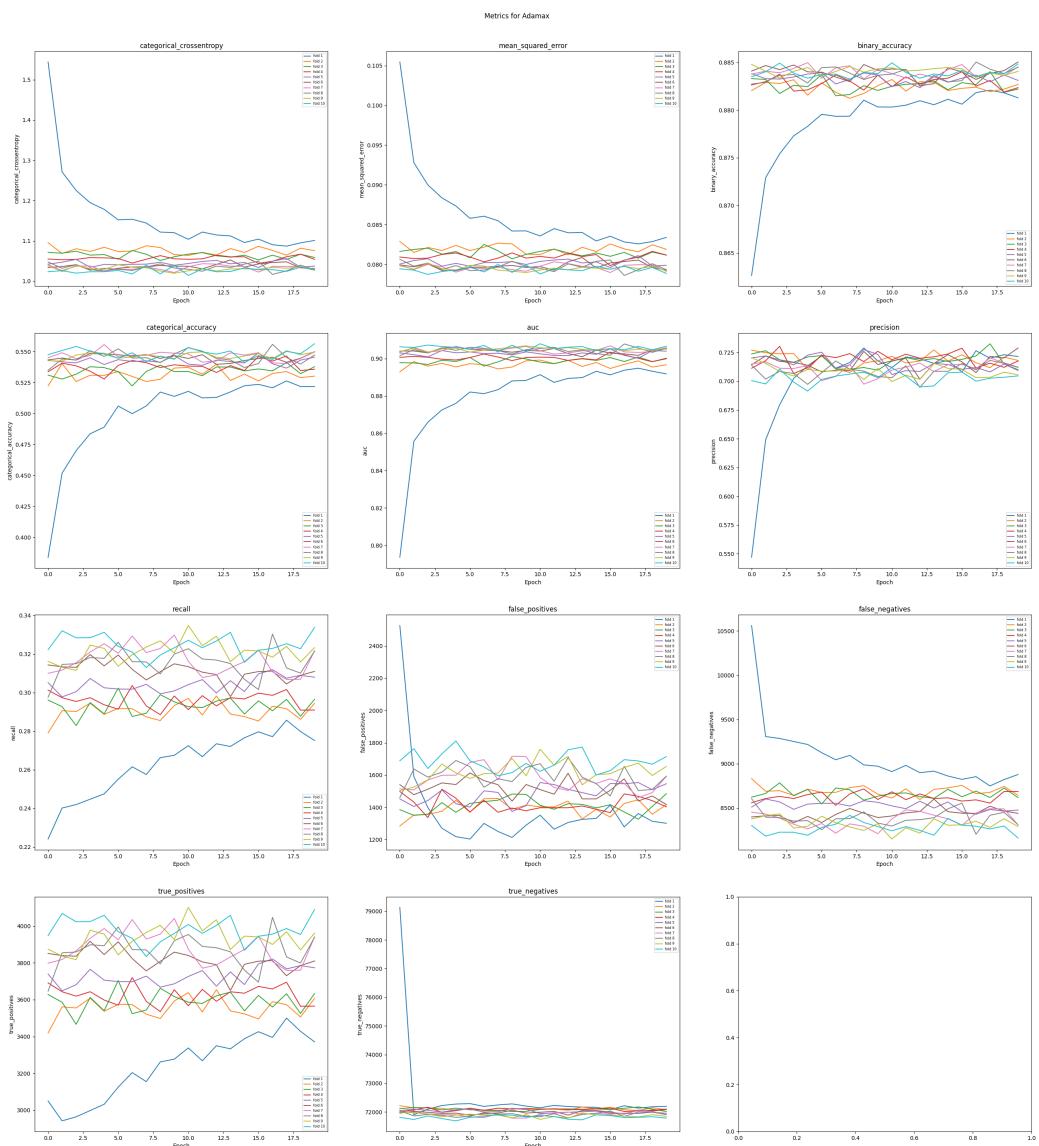


Figure 22: Wykres dla datasetu DryBean z optymalizatorem Adamax

6.3.7 DryBean dla optymalizatora Nadam

DryBean classification model for optimizer: Nadam

Metric: categorical_crossentropy → Score: 1.538251519203186
 Metric: mean_squared_error → Score: 0.1126939132809639
 Metric: binary_accuracy → Score: 0.8522095084190369
 Metric: categorical_accuracy → Score: 0.3401910364627838
 Metric: auc → Score: 0.8280351161956787
 Metric: precision → Score: 0.47058823704719543
 Metric: recall → Score: 0.2762674391269684
 Metric: false_positives → Score: 423.0
 Metric: false_negatives → Score: 985.0
 Metric: true_positives → Score: 376.0
 Metric: true_negatives → Score: 7743.0

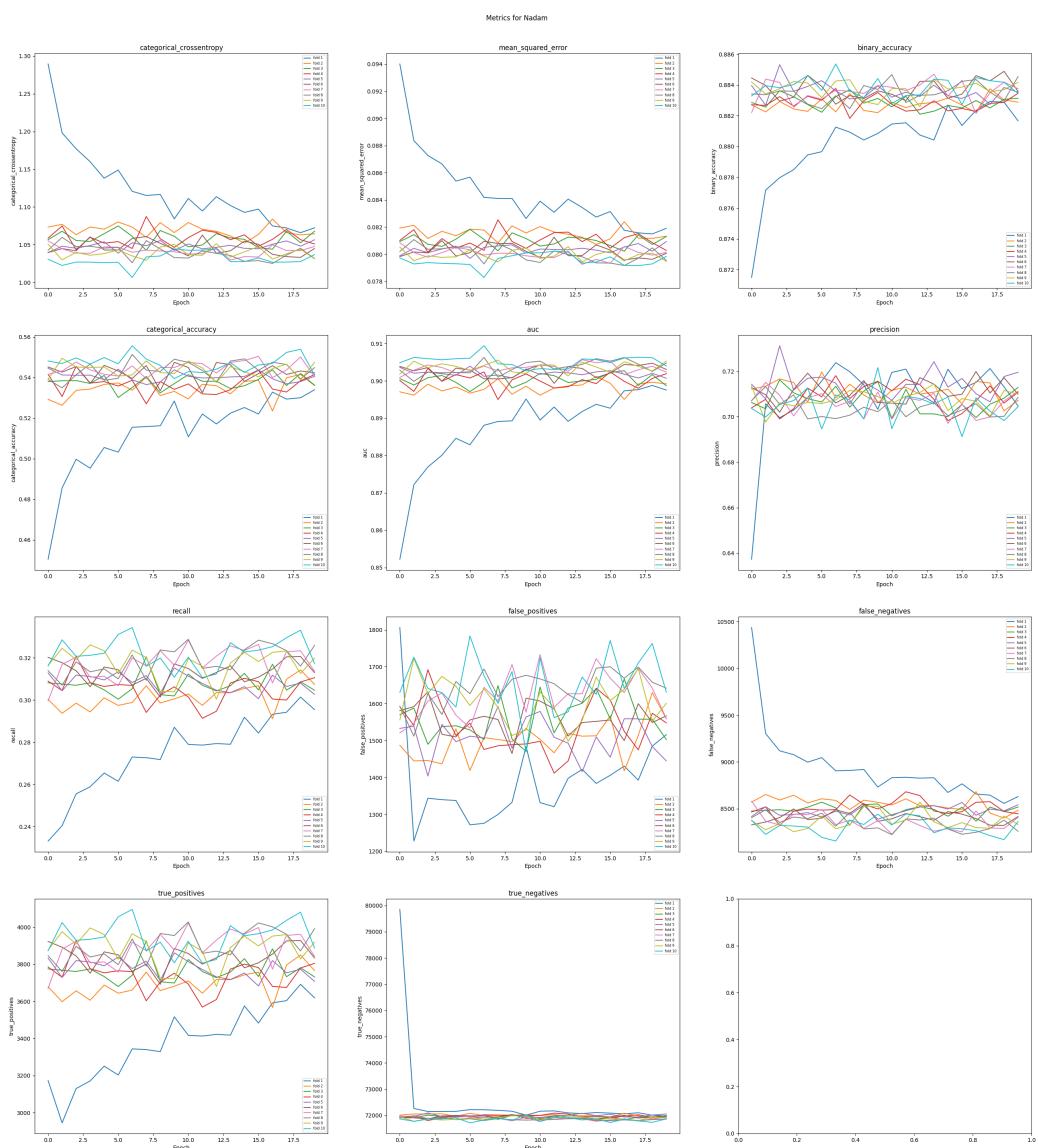


Figure 23: Wykres dla datasetu DryBean z optymalizatorem Nadam

6.3.8 DryBean dla optymalizatora Ftrl

DryBean classification model for optimizer: Ftrl

Metric: categorical_crossentropy → Score: 1.661093831062317
 Metric: mean_squared_error → Score: 0.1163160651922226
 Metric: binary_accuracy → Score: 0.8598719239234924
 Metric: categorical_accuracy → Score: 0.34900808334350586
 Metric: auc → Score: 0.8177242875099182
 Metric: precision → Score: 0.516372799873352
 Metric: recall → Score: 0.30124908685684204
 Metric: false_positives → Score: 384.0
 Metric: false_negatives → Score: 951.0
 Metric: true_positives → Score: 410.0
 Metric: true_negatives → Score: 7782.0

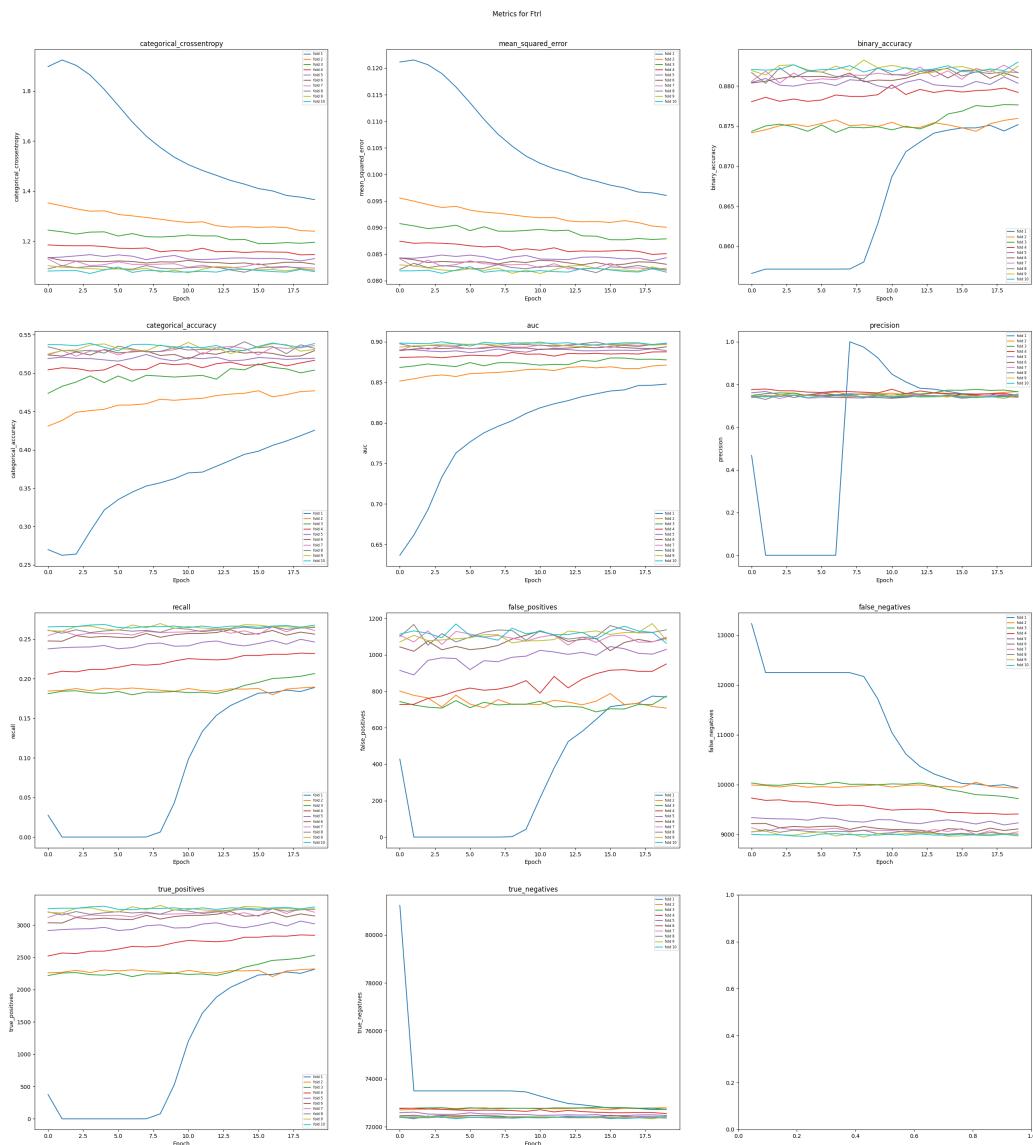


Figure 24: Wykres dla datasetu DryBean z optymalizatorem Ftrl

Zakładając, że istotne są wszystkie metryki, możemy zaobserwować, że:

- Optymalizator RMSprop generuje najlepsze wyniki w kategoriach categorical_accuracy i precision, a także

wyróżnia się dość dobrą binary_accuracy i auc. W tym kontekście, może on być uznany za najlepszy optymalizator.

- Z drugiej strony, optymalizator Ftrl ma najgorsze wyniki w kategoriach categorical_accuracy, a także dość słabe wyniki dla binary_accuracy, precision i auc. W tym kontekście, Ftrl może być uznany za najgorszy optymalizator.

6.4 Raisin

6.4.1 Raisin dla optymalizatora SGD

Raisin classification model for optimizer: SGD

Metric: categorical_crossentropy → Score: 0.37206199765205383

Metric: mean_squared_error → Score: 0.11677148938179016

Metric: binary_accuracy → Score: 0.8333333134651184

Metric: categorical_accuracy → Score: 0.8333333134651184

Metric: auc → Score: 0.9147530794143677

Metric: precision → Score: 0.8333333134651184

Metric: recall → Score: 0.8333333134651184

Metric: false_positives → Score: 15.0

Metric: false_negatives → Score: 15.0

Metric: true_positives → Score: 75.0

Metric: true_negatives → Score: 75.0

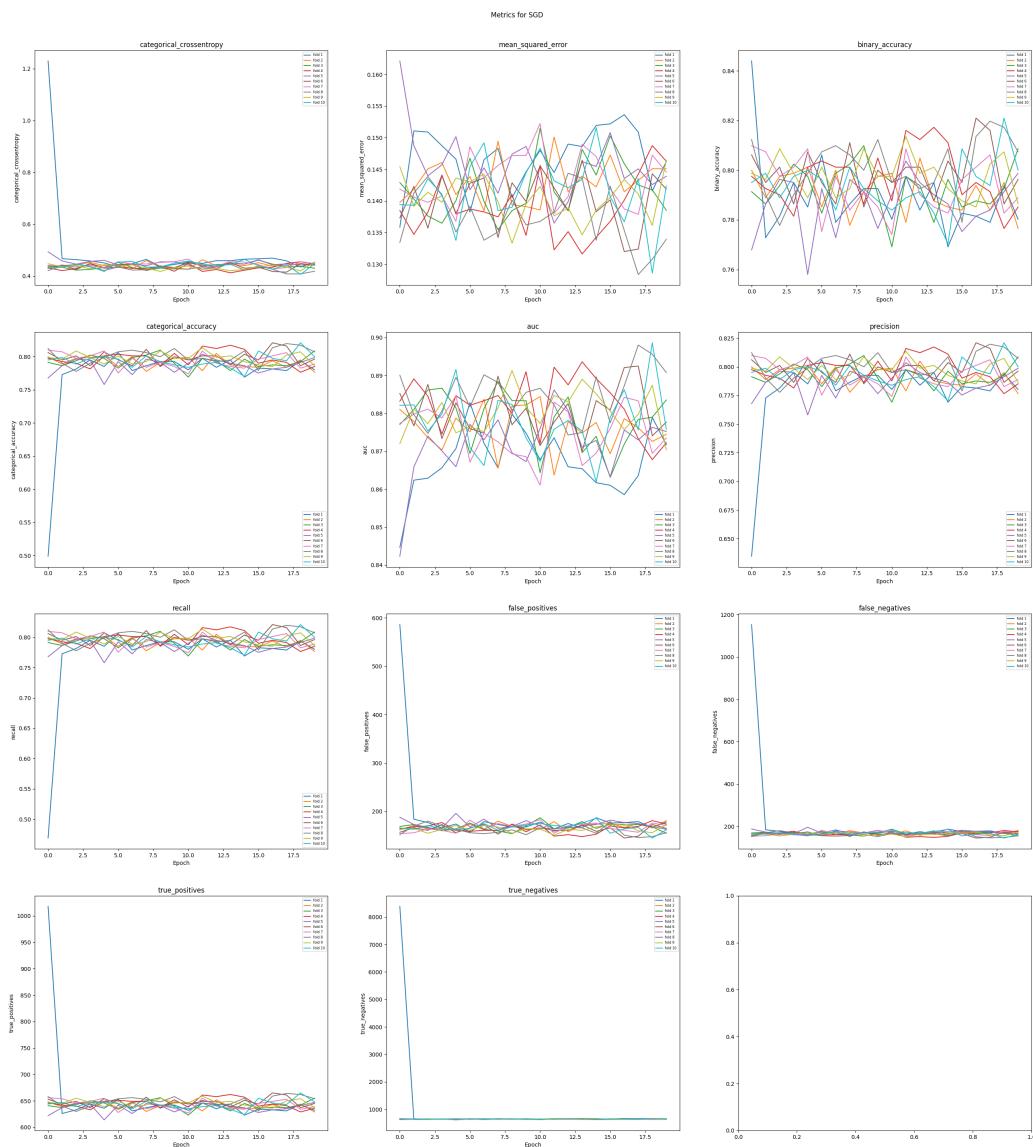


Figure 25: Wykres dla datasetu Raisin z optymalizatorem SGD

6.4.2 Raisin dla optymalizatora RMSprop

Raisin classification model for optimizer: RMSprop

Metric: categorical_crossentropy → Score: 0.37338265776634216
 Metric: mean_squared_error → Score: 0.11666478216648102
 Metric: binary_accuracy → Score: 0.8444444537162781
 Metric: categorical_accuracy → Score: 0.8444444537162781
 Metric: auc → Score: 0.9151851534843445
 Metric: precision → Score: 0.8444444537162781
 Metric: recall → Score: 0.8444444537162781
 Metric: false_positives → Score: 14.0
 Metric: false_negatives → Score: 14.0
 Metric: true_positives → Score: 76.0
 Metric: true_negatives → Score: 76.0

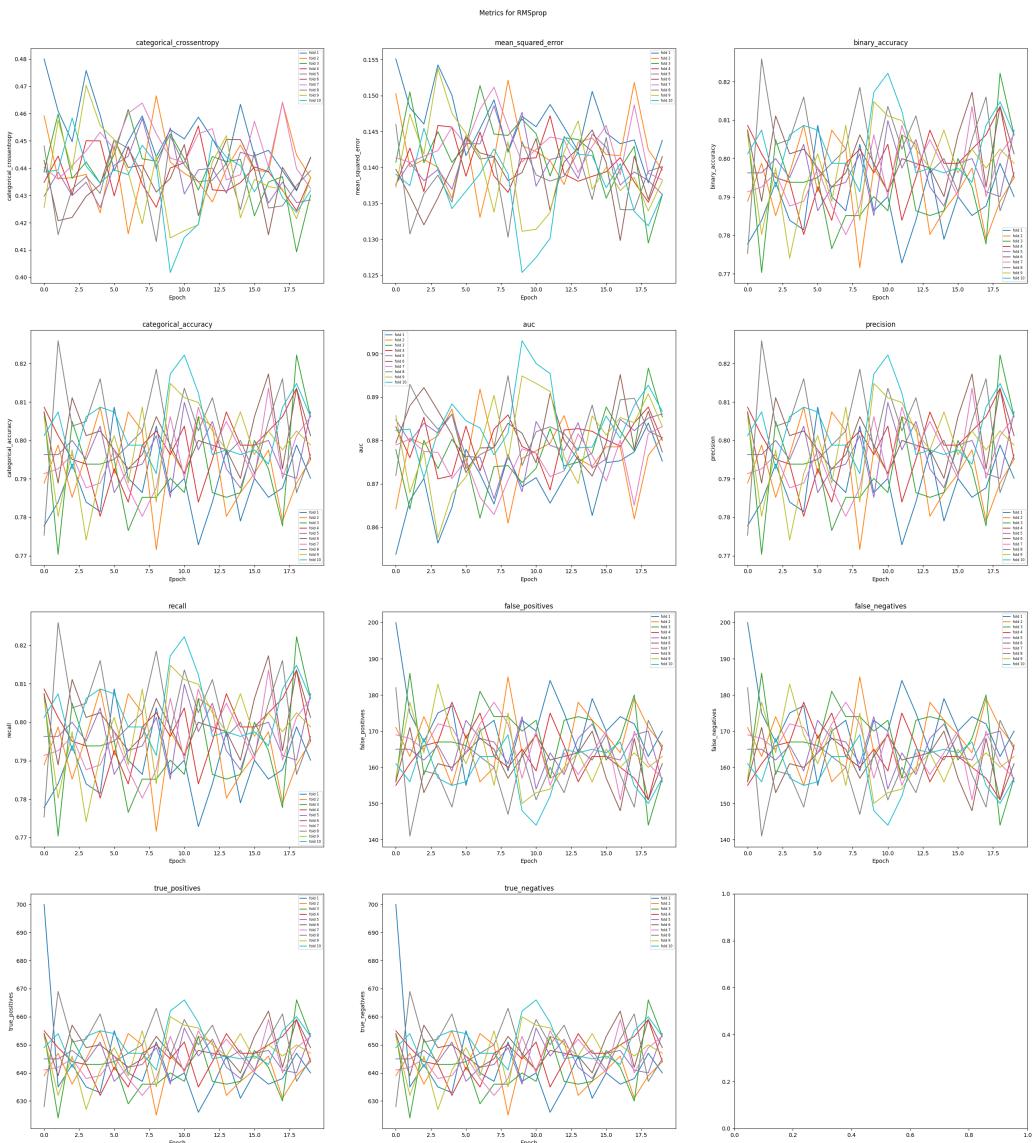


Figure 26: Wykres dla datasetu Raisin z optymalizatorem RMSprop

6.4.3 Raisin dla optymalizatora Adam

Raisin classification model for optimizer: Adam

```
Metric: categorical_crossentropy -> Score: 0.3939453065395355
Metric: mean_squared_error -> Score: 0.1223108097910881
Metric: binary_accuracy -> Score: 0.8333333134651184
Metric: categorical_accuracy -> Score: 0.8333333134651184
Metric: auc -> Score: 0.9024690389633179
Metric: precision -> Score: 0.8333333134651184
Metric: recall -> Score: 0.8333333134651184
Metric: false_positives -> Score: 15.0
Metric: false_negatives -> Score: 15.0
Metric: true_positives -> Score: 75.0
Metric: true_negatives -> Score: 75.0
```

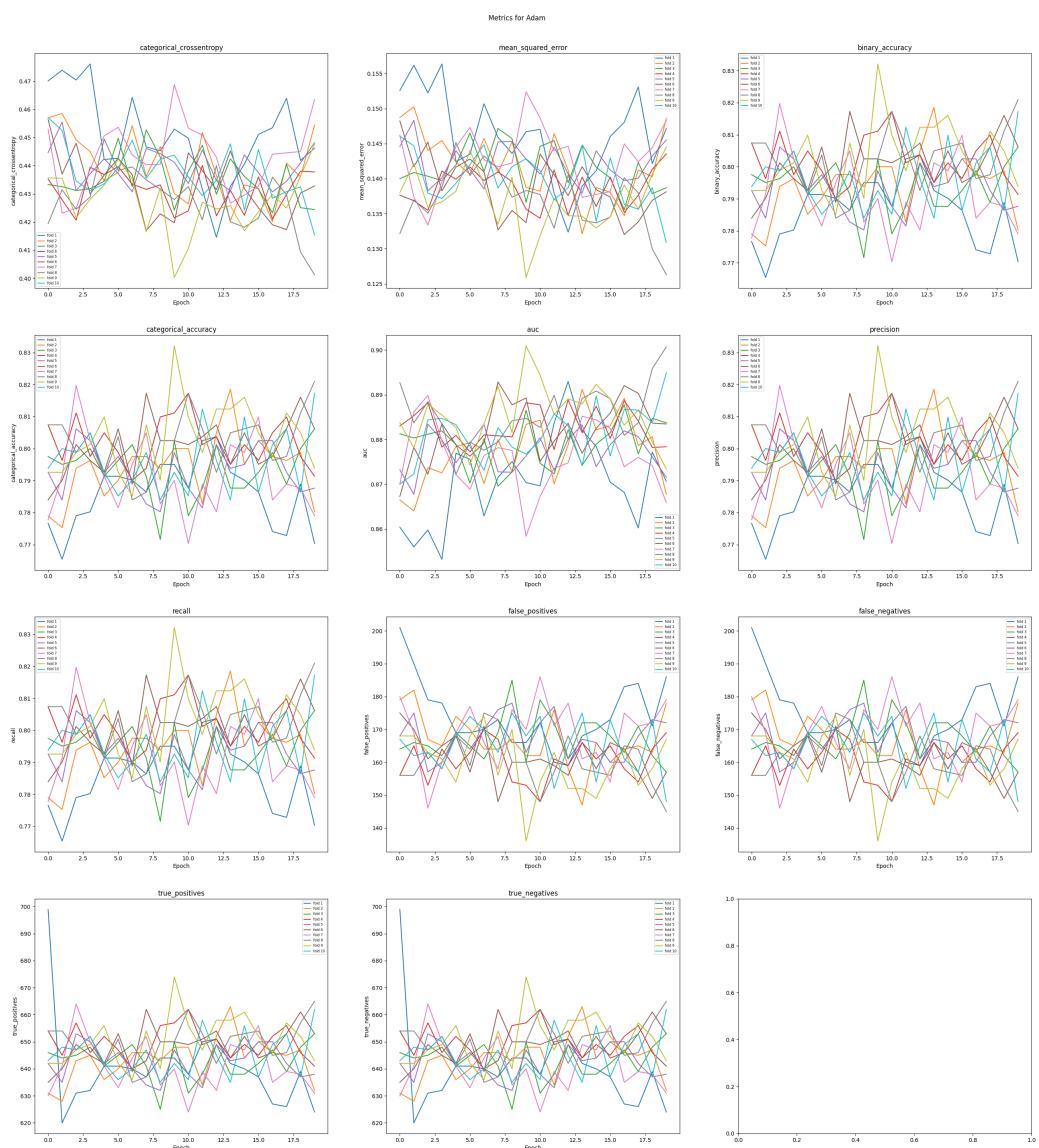


Figure 27: Wykres dla datasetu Raisin z optymalizatorem Adam

6.4.4 Raisin dla optymalizatora Adadelta

Raisin classification model for optimizer: Adadelta

```
Metric: categorical_crossentropy -> Score: 0.4060397148132324
Metric: mean_squared_error -> Score: 0.12682749330997467
Metric: binary_accuracy -> Score: 0.8444444537162781
Metric: categorical_accuracy -> Score: 0.8444444537162781
Metric: auc -> Score: 0.9129012823104858
Metric: precision -> Score: 0.8444444537162781
Metric: recall -> Score: 0.8444444537162781
Metric: false_positives -> Score: 14.0
Metric: false_negatives -> Score: 14.0
Metric: true_positives -> Score: 76.0
Metric: true_negatives -> Score: 76.0
```

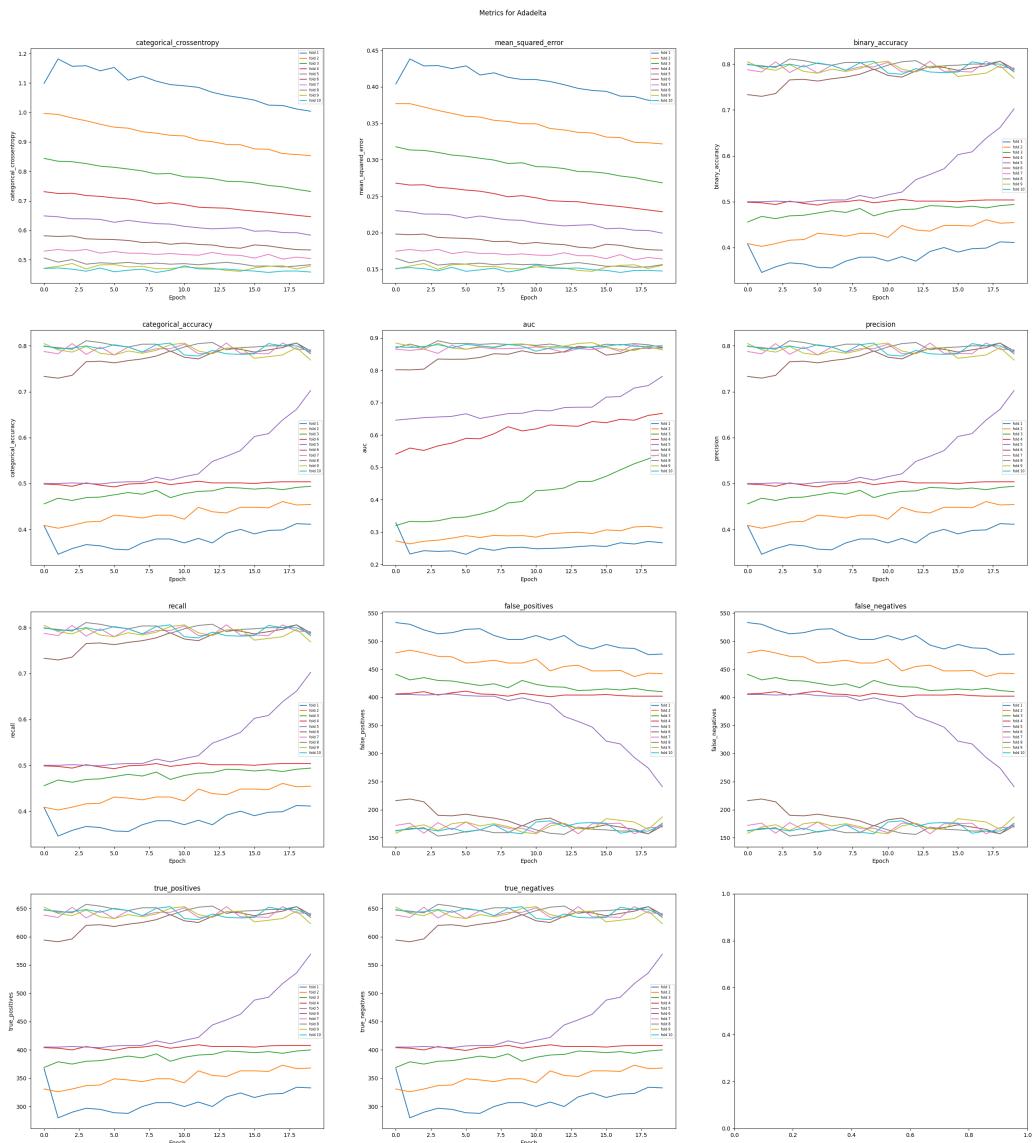


Figure 28: Wykres dla datasetu Raisin z optymalizatorem Adadelta

6.4.5 Raisin dla optymalizatora Adagrad

Raisin classification model for optimizer: Adagrad

```
Metric: categorical_crossentropy -> Score: 0.3690939247608185
Metric: mean_squared_error -> Score: 0.11456688493490219
Metric: binary_accuracy -> Score: 0.855555534362793
Metric: categorical_accuracy -> Score: 0.855555534362793
Metric: auc -> Score: 0.9185802340507507
Metric: precision -> Score: 0.855555534362793
Metric: recall -> Score: 0.855555534362793
Metric: false_positives -> Score: 13.0
Metric: false_negatives -> Score: 13.0
Metric: true_positives -> Score: 77.0
Metric: true_negatives -> Score: 77.0
```

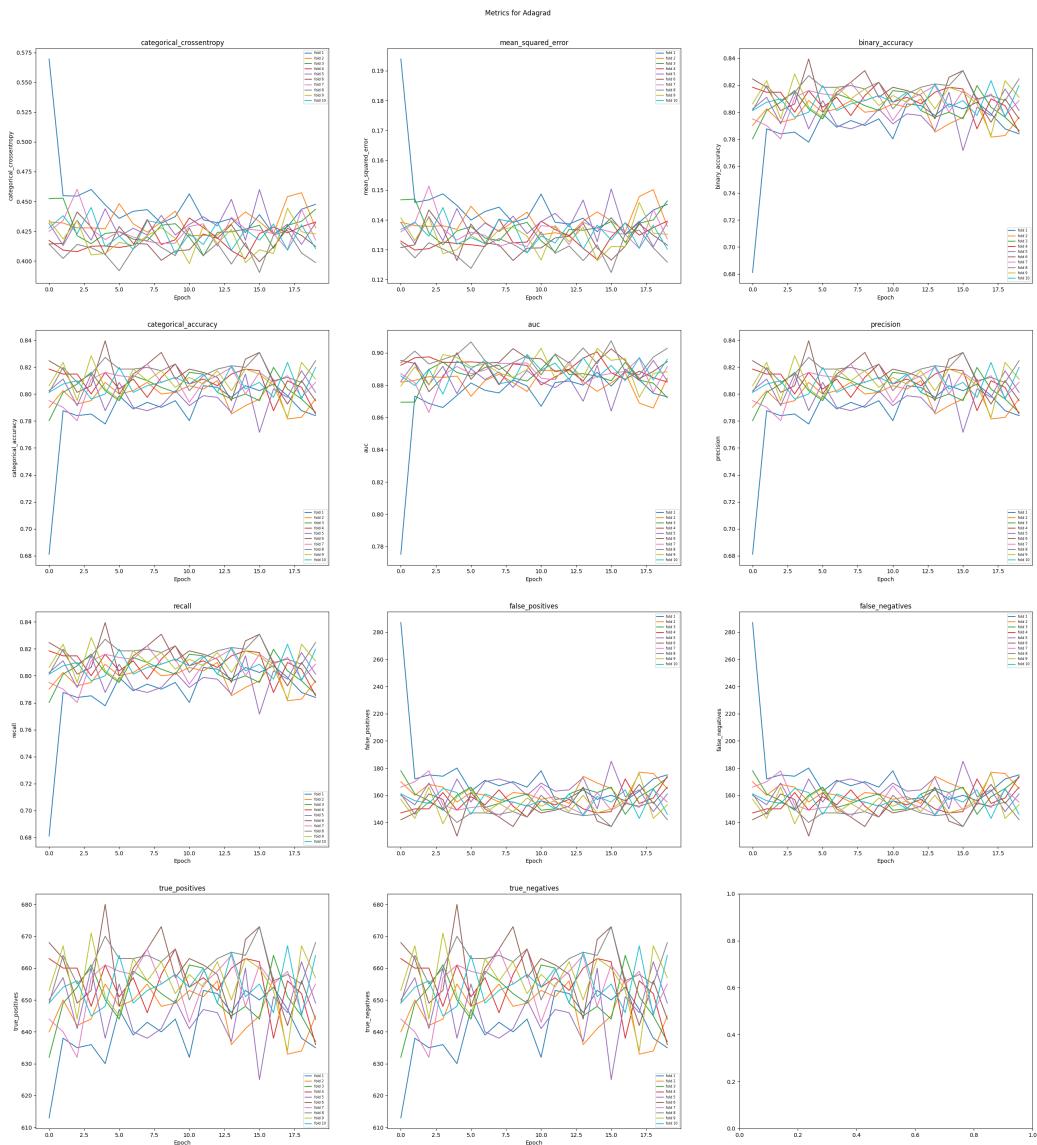


Figure 29: Wykres dla datasetu Raisin z optymalizatorem Adagrad

6.4.6 Raisin dla optymalizatora Adamax

Raisin classification model for optimizer: Adamax

Metric: categorical_crossentropy → Score: 0.8365160822868347

Metric: mean_squared_error → Score: 0.25170308351516724

Metric: binary_accuracy → Score: 0.6888889074325562

Metric: categorical_accuracy → Score: 0.6888889074325562

Metric: auc → Score: 0.758765459060669

Metric: precision → Score: 0.6888889074325562

Metric: recall → Score: 0.6888889074325562

Metric: false_positives → Score: 28.0

Metric: false_negatives → Score: 28.0

Metric: true_positives → Score: 62.0

Metric: true_negatives → Score: 62.0

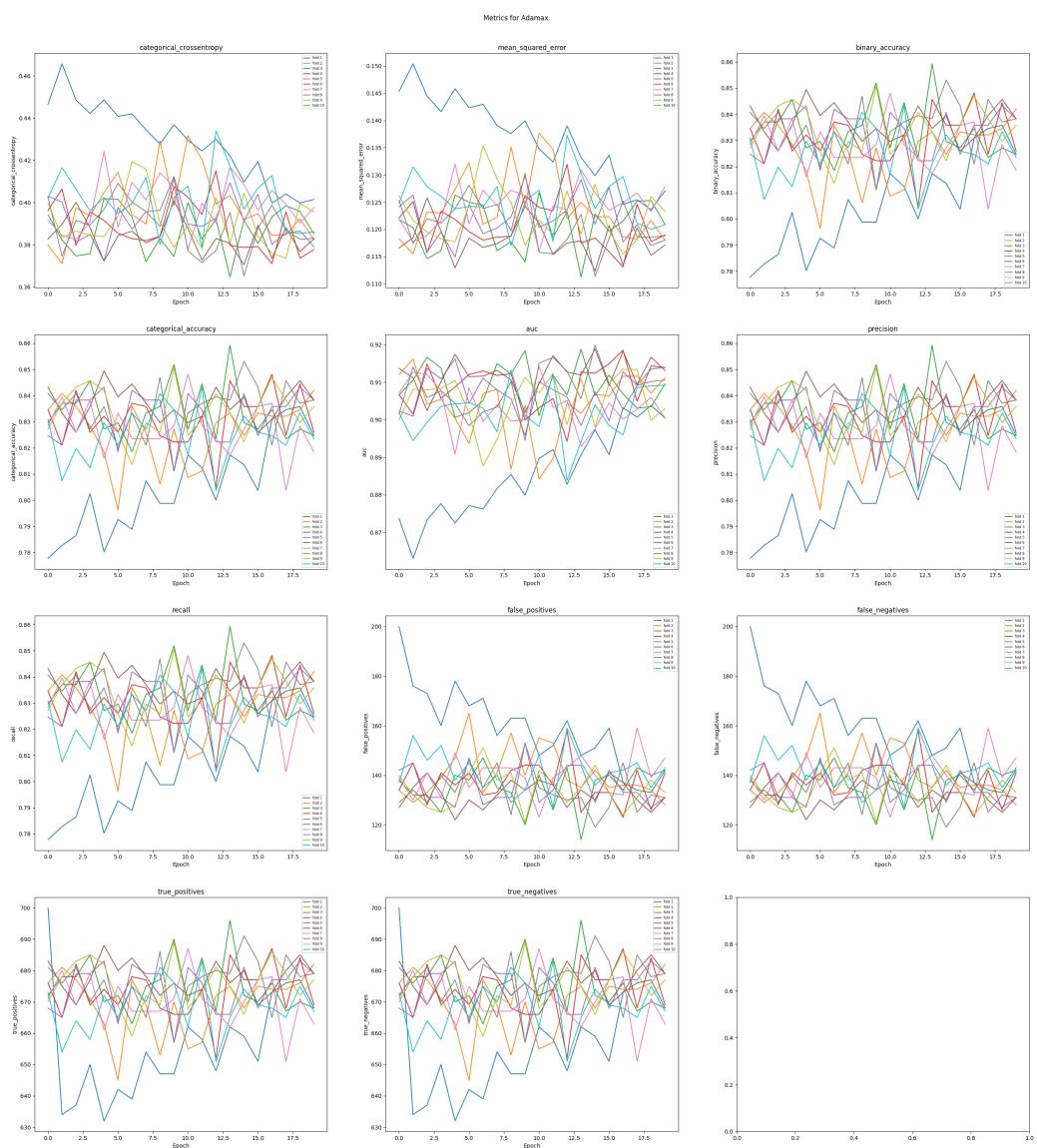


Figure 30: Wykres dla datasetu Raisin z optymalizatorem Adamax

6.4.7 Raisin dla optymalizatora Nadam

Raisin classification model for optimizer: Nadam

Metric: categorical_crossentropy → Score: 0.39193451404571533

Metric: mean_squared_error → Score: 0.12282675504684448

Metric: binary_accuracy → Score: 0.8333333134651184

Metric: categorical_accuracy → Score: 0.8333333134651184

Metric: auc → Score: 0.9062962532043457

Metric: precision → Score: 0.8333333134651184

Metric: recall → Score: 0.8333333134651184

Metric: false_positives → Score: 15.0

Metric: false_negatives → Score: 15.0

Metric: true_positives → Score: 75.0

Metric: true_negatives → Score: 75.0

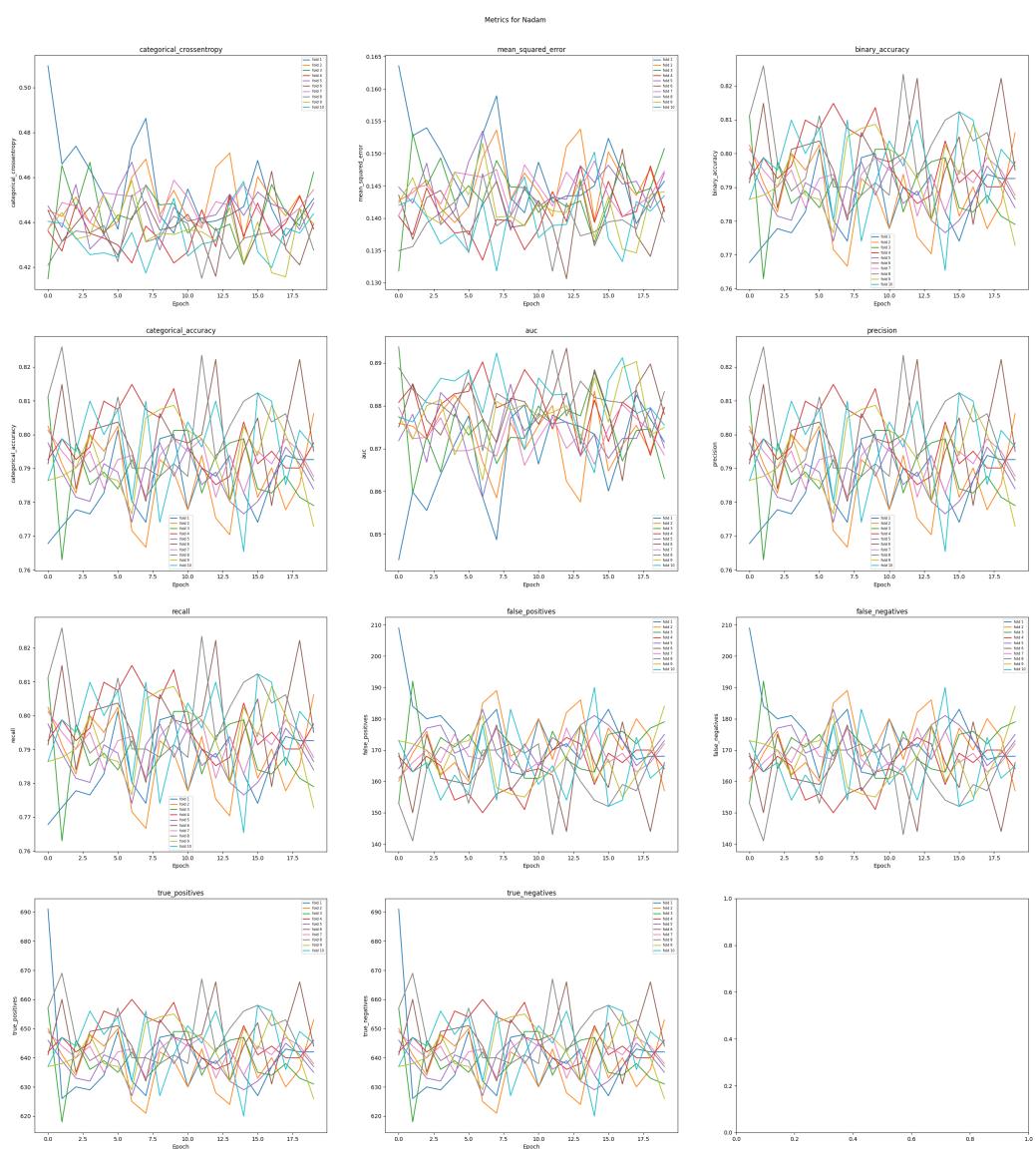


Figure 31: Wykres dla datasetu Raisin z optymalizatorem Nadam

6.4.8 Raisin dla optymalizatora Ftrl

Raisin classification model for optimizer: Ftrl

```
Metric: categorical_crossentropy -> Score: 0.39247921109199524
Metric: mean_squared_error -> Score: 0.12229129672050476
Metric: binary_accuracy -> Score: 0.8444444537162781
Metric: categorical_accuracy -> Score: 0.8444444537162781
Metric: auc -> Score: 0.9072839021682739
Metric: precision -> Score: 0.8444444537162781
Metric: recall -> Score: 0.8444444537162781
Metric: false_positives -> Score: 14.0
Metric: false_negatives -> Score: 14.0
Metric: true_positives -> Score: 76.0
Metric: true_negatives -> Score: 76.0
```

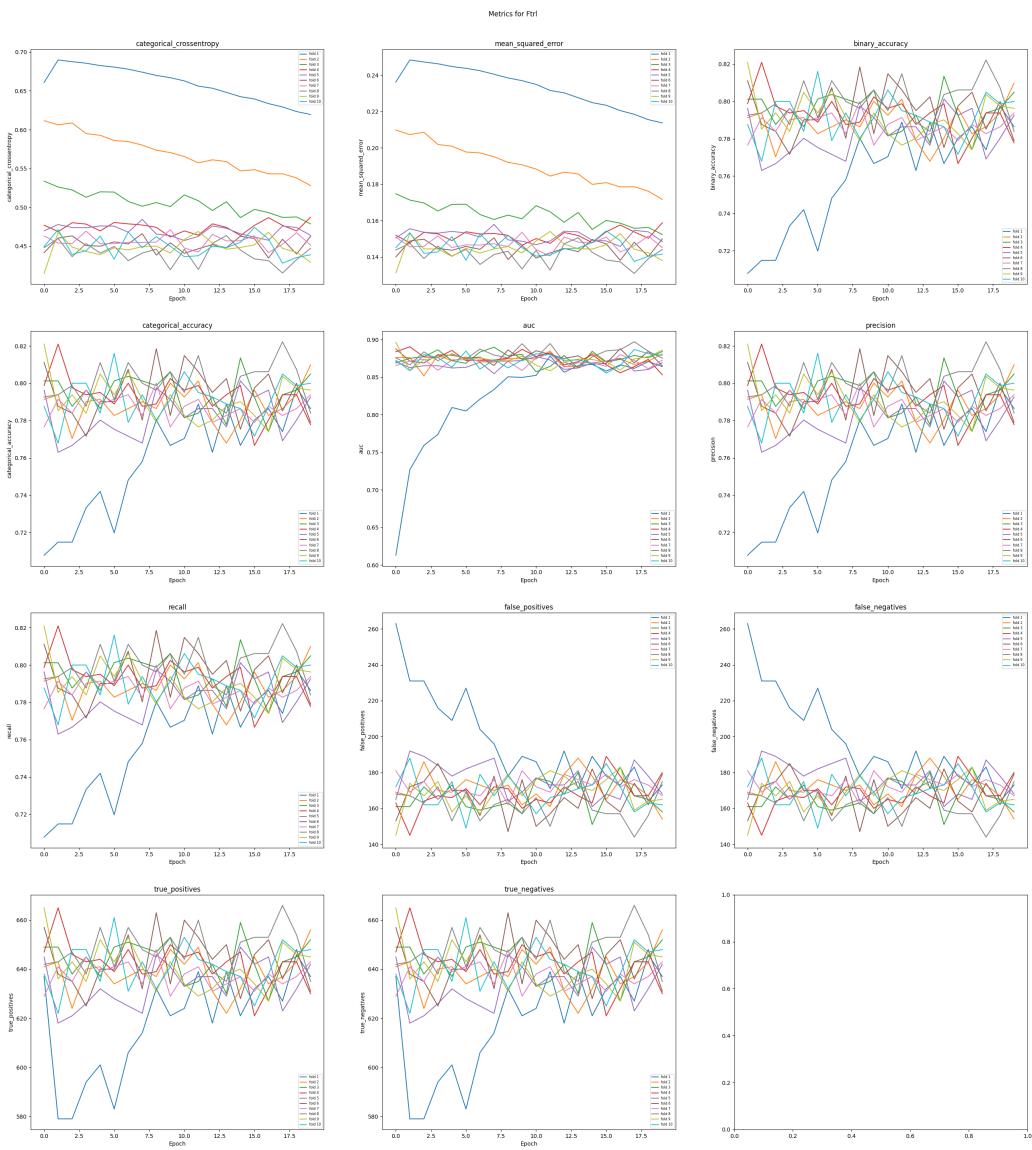


Figure 32: Wykres dla datasetu Raisin z optymalizatorem Ftrl

Patrząc na podane wyniki, najlepszy model to ten używający optymalizatora Adagrad, który ma:

- Najniższe wyniki dla: loss (0.3690939247608185), mean_squared_error (0.11456688493490219)

- Najwyższe wyniki dla: auc (0.9185802340507507), precision (0.855555534362793), recall (0.855555534362793), binary_accuracy (0.855555534362793), categorical_accuracy (0.855555534362793)

Najgorszy model to ten używający optymalizatora Adamax, który ma:

- Najwyższe wyniki dla: categorical_crossentropy (0.8365160822868347), mean_squared_error (0.25170308351516724), false_positives (28.0), false_negatives (28.0)
- Najniższe wyniki dla: auc (0.758765459060669), precision (0.688889074325562), recall (0.688889074325562), binary_accuracy (0.6888889074325562), categorical_accuracy (0.6888889074325562)

7 Podsumowanie

Porównanie różnych algorytmów optymalizacji dla sieci konwolucyjnych na różnych zestawach danych pokazuje, że nie ma jednego optymalnego rozwiązania dla wszystkich problemów. Wydajność każdego algorytmu zależy od specyfiki problemu, danych i architektury modelu.

Algorytmy optymalizacji, takie jak Adam, Nadam, RMSprop, które adaptacyjnie dostosowują tempo uczenia, generalnie wykazywały dobrą wydajność na różnych zestawach danych. Te optymalizatory mogą być dobrym punktem wyjścia do eksperymentowania z różnymi problemami uczenia maszynowego.

Jednakże, nie zawsze "zaawansowane" optymalizatory są najlepsze. W niektórych przypadkach, prostsze metody, takie jak SGD, mogą dać porównywalne, a czasami nawet lepsze wyniki. Dlatego warto przetestować różne optymalizatory i dostosować ich parametry do specyfiki problemu.

Przy ocenie wydajności algorytmów optymalizacji, warto także wziąć pod uwagę ich złożoność obliczeniową i pamięciową. Niektóre zaawansowane optymalizatory, mimo że mogą dawać lepsze wyniki, mogą wymagać więcej zasobów obliczeniowych i pamięci, co może być problemem, szczególnie w przypadku dużych modeli i zestawów danych.

Podsumowując, wybór odpowiedniego algorytmu optymalizacji jest kluczowym krokiem w procesie uczenia sieci konwolucyjnych. Wybór ten powinien być oparty na starannym przetestowaniu różnych algorytmów i dostosowaniu ich parametrów do specyfiki problemu i danych.