Projektni zadatak

- IZVEŠTAJ -

Leković Matija 0148/22 i Jovanovski Irina 0316/22 | Neuralne Mreže

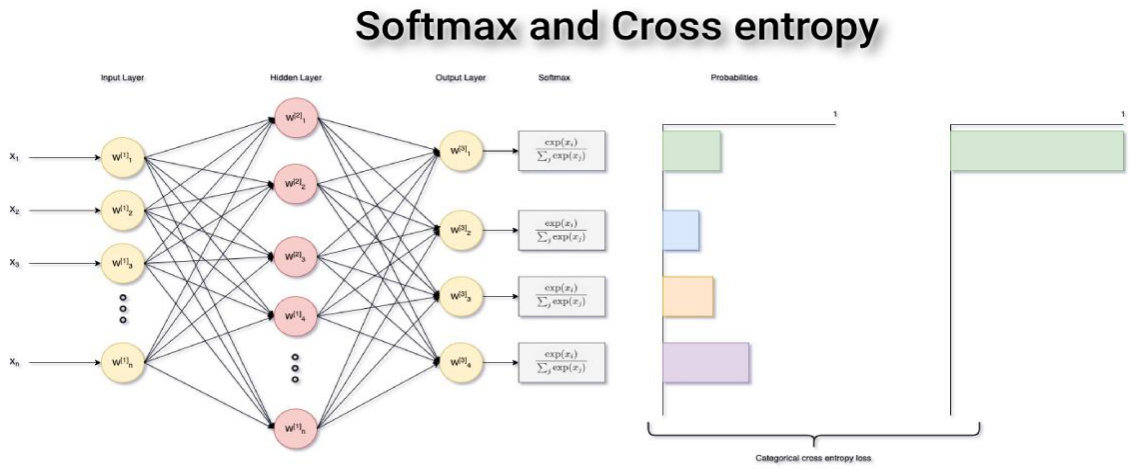
Projektovanje potpuno povezane neuralne mreže

U okviru zadatka bilo je potrebno projektovati višeslojnu potpuno povezanu neuralnu mrežu za klasifikaciju odbiraka u više klasa. Skup podataka nad kojim je projektovana je u našem slučaju bio *Rice.csv*. Skup podataka se sastoji od:

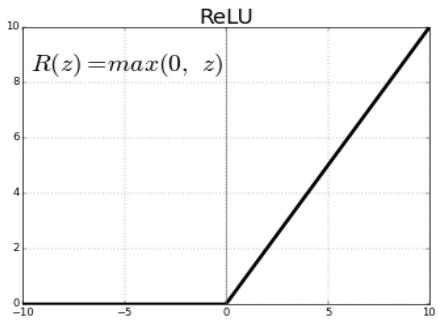
- ✓ **4259** uzoraka
- ✓ **7** atributa tj. obeležja:
  - *area*
  - *perimeter*
  - *majorAxis*
  - *minorAxis*
  - *eccentricity*
  - *convexArea*
  - *extent*
- ✓ **3** klase:
  - *Cammeo*
  - *Osmancik*
  - *Kecimen*

Sama priroda klasifikacionog zadatka se kolokvijalno može definisati kao sortiranje zrna pirinča na osnovu njegovog fizičkog izgleda tj. karakteristika. „Formalna“ priroda se ogleda u tome da je u pitanju višeklasna klasifikacija, gde svaki uzorak pripada tačno jednoj od tri potencijalne klase. Ulazni podaci tj. atributi su numeričkog tipa, a izlazni su takođe numeričkog tipa koji se dobijaju tako što klase koje su tekstualnog (*String*) tipa preslikaju u numeričke labele na osnovu kojih se posle vrši klasifikacija.

Prilikom konstrukcije mreže za kriterijumsku funkciju bila nam je neophodna funkcija koja je pogodna za rad sa višeklasnim problemima, uzima u obzir i dobro kvantifikuje razliku između trenutnih i željenih izlaza, i sa svakom narednom epohom teži da poveća verovatnoću pripadanja klase kojoj uzorak zapravo pripada i smanji verovatnoću pripadanja ostalim – **Categorical Cross-Entropy Loss** (slika ispod) je funkcija koja se dobro uklapa u ove kriterijume.



Za funkciju aktivacije neurona skrivenog sloja odlučili smo se za **ReLU** funkciju (prikazana na slici desno) koja je generalno najpopularnija upravo zbog svoje efikasnosti i jednostavnosti u smislu niske kompleksnosti računanja. Ona takođe rešava problem nestajućeg gradijenta i time sprečava „umiranje“ neurona tokom treninga. S obzirom da se kao kriterijumska funckija koristi Categorical Cross-Entropy Loss koja interno primenjuje **softmax**, na izlazni sloj se ne primenjuje dodatno aktivaciona funkcija.



Za metodu optimizacije korišćene za minimizaciju funkcije greške odabrali smo **mini-batch gradijentni spust** [1]**+ Adam**[2] . Mini-batch gradijentni spust omogućava efikasno obučavanje obradom podskupova podataka balansirajući računsku brzinu i tačnost gradijenta. Umesto da koristi ceo skup podataka ili jedan uzorak za svako ažuriranje, mini batch nalazi optimalnu tačku koja smanjuje računsku složenost uz unošenje korisnog šuma koji pomaže modelima da bolje generalizuju što potom omogućava bržu obuku, stabilnije konvergiranje i poboljšane performanse. Adam optimizator koristi adaptivne momente koji omogućavaju bržu i stabilniju konvergenciju čime se minimizuje rizik zaglavljivanja u lokalnim minimumima i maksimizuje generalizacija modela. Pored ovih metoda optimizacije uzevši u obzir činjenicu da radimo sa čak 7 parametara i u poređenju s tim ne prevelikim datasetom, mislili smo da bi bilo korisno izbeći mogućnost preobučavanja i shodno tome dodali **rano zaustavljanje** kao oblik zaštite od preobučavanja.

Tri hiperparametra koja smo odabrali su:

1. broj neurona u prvom skrivenom sloju:
    a. uloga --> kapacitet sloja za apstrakciju podataka
    b. test vrednosti --> [32, 64, 128]
2. funkcija aktivacije u prvom skrivenom sloju:
    a. uloga --> transformiše ulaz neurona u izlaz
    b. test vrednosti --> [relu, tanh, sigmoid]
3. konstanta obučavanja:
    a. uloga --> kontroliše veličinu koraka tokom ažuriranja težina
    b. test vrednosti --> [0.001, 0.005, 0.01]

---

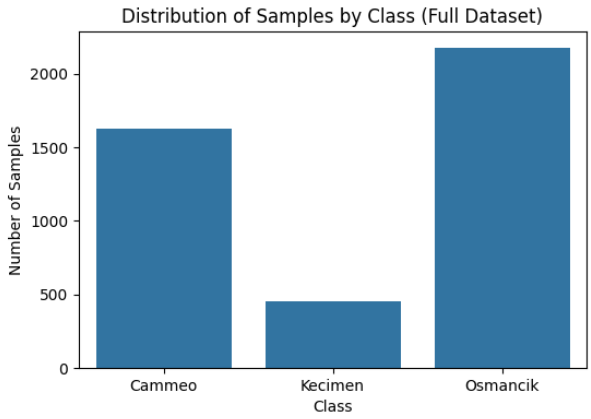
[1] mini-batch gradijentni spust (*gradient descent*) - *SGD* --> isti princip kao obican batch mod, gde se prvo primenjuju svi ulazi na mrezu, pa se onda ceo zbir dodaje za update (npr. pronadjemo δ za prvog, pa za drugog itd.. i dobijemo $\Delta w_{9,7} = \Delta w_{9,7}(1) + \Delta w_{9,7}(2) + \ldots + \Delta w_{9,7}(1000)$; i tek onda radimo jedno azuriranje pojacanja), samo sto se kod SGD-a ne primenjuje svih p ulaza odjednom, vec se ista stvar radi na manjem skupu ulaza (k iteracija)

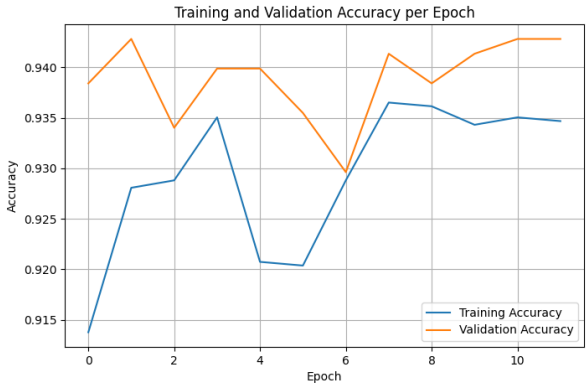
$$E = \frac{1}{2}\sum_{i=1}^{k}\left(d^{(i)} - y^{(i)}\right)^2 \ , k < p$$

[2] Adam, ili *Adaptive Moment Estimation*, je algoritam optimizacije koji kombinuje karakteristike metoda Momentum i RMSProp. Široko se koristi u dubokom učenju za efikasno ažuriranje težina mreže na osnovu podataka za obuku, prilagođavajući brzine učenja za različite parametre kako bi se poboljšale performanse kod različitih problema optimizacije.

$$grad = \frac{\partial J}{\partial \theta}, \quad m_t = \beta_1 m_{t-1} + (1-\beta_1)grad, \quad v_t = \beta_2 v_{t-1} + (1-\beta_2)grad^2, \quad \theta = \theta - \alpha\frac{m_t}{\sqrt{v_t + \epsilon}}$$

Na histogramu desno vidimo distribuciju uzoraka po individualnim klasama. Pre svega bitno je primetiti da je prisutan ***disbalans klasa***; iako postoji razlika u broju uzoraka za *Cammeo* (1629) i *Osmancik* (2180) klase, evidetno je da *Kecimen* sa svojih 450 uzoraka ekstremno odudara od obe što dalje može dovesti do smanjenja tačnosti modela, loše generalizacije, preobučavanja i drugih problema.





Na grafiku levo vidimo krive koje opisuju tačnost tokom 10 epoha. S obzirom da se krive ponašaju u skladu sa očekivanjima (u smislu da tokom epoha tačnost raste za obe krive) jedino relevantno zapažanje bi moglo biti da do većih oscilacija verovatno dolazi usled prethodno spomenutog disbalansa klasa.

$$precision = \frac{True\ Positives^{3}}{True\ Positives + False\ Positives^{4}}$$

$$recall (osetljivost) = \frac{True\ Positives}{True\ Positives + False\ Negatives^{5}}$$

$$f1 - score = 2 \times \frac{precision \times recall}{precision + recall} = \frac{TP}{TP + 0.5(FP + FN)}$$

$$accuracy = \frac{True\ Positives + True\ Negatives^{6}}{Total\ Instances^{7}}$$

```
Classification Report on Test Set:
              precision    recall  f1-score   support

      Cammeo       0.92      0.91      0.92       326
     Kecimen       1.00      1.00      1.00        90
    Osmancik       0.93      0.94      0.94       436

    accuracy                           0.94       852
   macro avg       0.95      0.95      0.95       852
weighted avg       0.94      0.94      0.94       852


Process finished with exit code 0
```

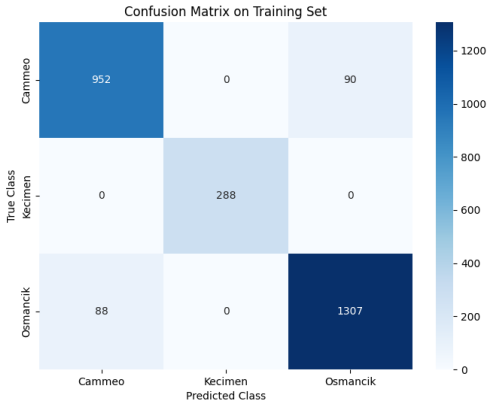
Support = broj stvarnih pojavljivanja uzoraka te klase

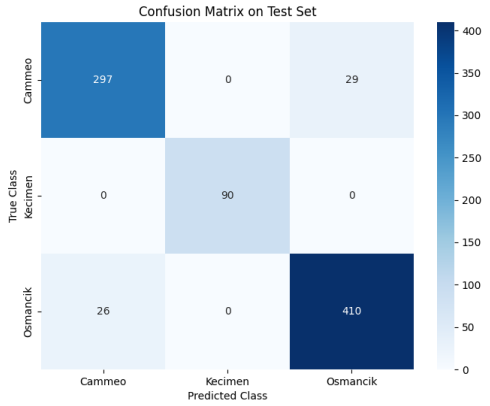
Macro avg je prosek metrika za svaku klasu ako tretiramo sve klase jednako, a weighted avg je prosek metrika za svaku klasu ako ga ponderišemo brojem instanci.

Činjenica da su svi rezultati visoki nam govori da je model dobro istreniran, s tim da kao što možemo primetiti kod *Kecimen* klase svi rezultati su 1.00 i svi uzorci su perfektno rasklasifikovani što može ukazivati na to da je došlo do preobučavanja.

[3] True Positive tj. *TP* – model tačno predviđa da uzorak jeste iz date (predviđene) klase
[4] False Positive tj. *FP* – model netačno predviđa da uzorak jeste iz date (predviđene) klase
[5] False Negative tj. *FN* – model netačno predviđa da uzorak nije iz date (predviđene) klase
[6] True Negative tj. TN – model tačno predviđa da uzorak nije iz date (predviđene) klase
[7] Total Instances = TP + TN + FP + FN

Konfuziona matrica dodatno vizualizuje predikcije modela nasuprot stvarnih klasi kojima pripadaju uzorci.

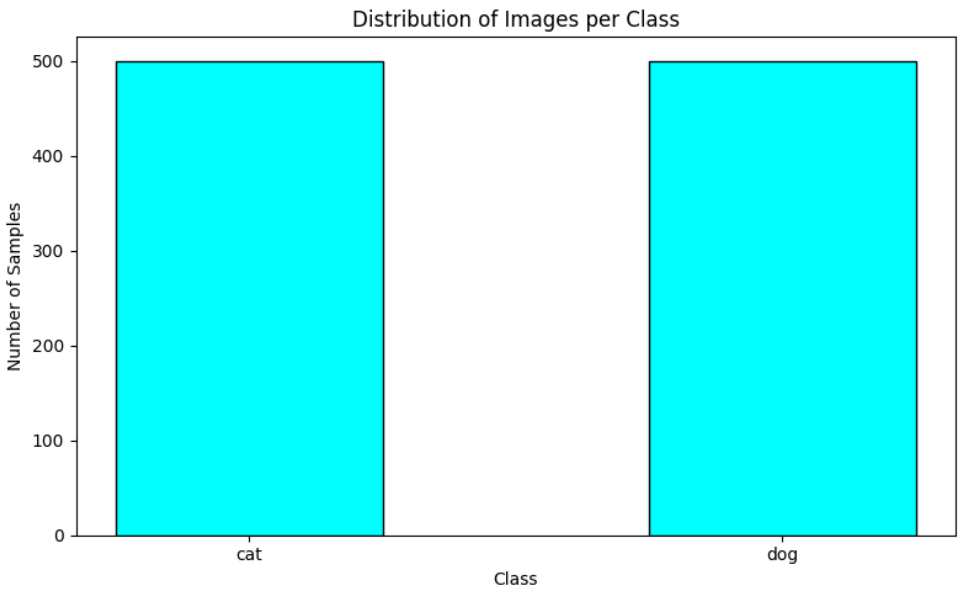




Projektovanje potpuno konvolucione neuralne mreže

U okviru ovog zadatka bilo je potrebno projektovati konvolucionu mrežu (CNN) za klasifikaciju slika u više klasa. Skup podataka (slika) koji smo pronašli se sastoji od 1000 uzoraka (**.jpg fajlovi**) tj. fotografija na kojima su psi ili mačke. Shodno tome svaki uzorak pripada jednoj od **2** klase:

- **pas** (*dog*)
- **mačka** (*cat*)



Od transformacija koje su primenjene imamo:

- ✓ Normalizaciju piksela
  - velike vrednosti piksela mogu otežati učenje pa je za model lakše da se reskaliraju u opseg [0,1]
  - ubrzava konvergenciju tokom treniranja i stabilizuje rad optimizatora

- ✓ Augmentaciju podataka
  - daje veći diverzitet fotografija modifikovanjem onih iz postojećeg skupa da bi model bio „otporniji“ na faktore kao što je ugao, osvetljenje, orijentacija, rotacija fotografije...
  - smanjuje verovatnoću preobučavanja i poboljšava generalizaciju
- ✓ Skaliranje (*resize*) slika
  - svrha je uvođenje uniformne veličine slike
  - omogućava konzistentan ulaz u neuronsku mrežu inače model ne bi mogao da obradi slike različitih dimenzija
- ✓ Shuffle + Batch
  - *shuffle* - da bi se podstakla nezavisnost od redosleda ulaza; *batch* - grupisanje radi efikasnijeg treniranja
  - poboljšava efikasnost treniranja i smanjuje rizik od zaglavljivanja u lokalnom minimumu
- ✓ Regularizacija
  - *Dropout*[8] + *L2 regularizacija*[9] --> smanjuju preobučavanje i poboljšavaju generalizaciju

```
model.compile(
    optimizer=Adam(learning_rate=0.0005),
    loss=SparseCategoricalCrossentropy(),
    metrics=["accuracy"],
)
```

```
model = Sequential([
    data_augmentation,
    layers.Rescaling(1./255),

    layers.Conv2D(32, 3, padding="same", activation="relu"),
    layers.MaxPooling2D(),

    layers.Conv2D(64, 3, padding="same", activation="relu"),
    layers.MaxPooling2D(),

    layers.Conv2D(128, 3, padding="same", activation="relu"),
    layers.MaxPooling2D(),

    layers.Conv2D(256, 3, padding="same", activation="relu"),
    layers.MaxPooling2D(),

    layers.Dropout(0.3),
    layers.Flatten(),
    layers.Dense(256, activation="relu", kernel_regularizer=L2(0.0005)),
    layers.Dropout(0.3),
    layers.Dense(num_classes, activation="softmax"),
```

Kriterijumska funkcija (***SparseCategoricalEntropy***) se dobro ponaša kod problema klasifikacije diskretnih klasa i izbegava potrebu za one-hot enkodiranjem[10] i pojednostavljuje pipeline i štedi memoriju. Od funkcija aktivacije za unutrašnje slojeve smo koristili ***ReLu*** a za izlazni ***softmax*** o čijim prednostima i karakteristikama je detaljnije diskutovano u okviru prethodnog zadatka, a ostaju identični i ovde. Kao metoda optimizacije korišćen je ***Adam*** o kom je takođe bilo više reči u prethodnom zadatku i razlozi ostaju pretežno isti i ovde.

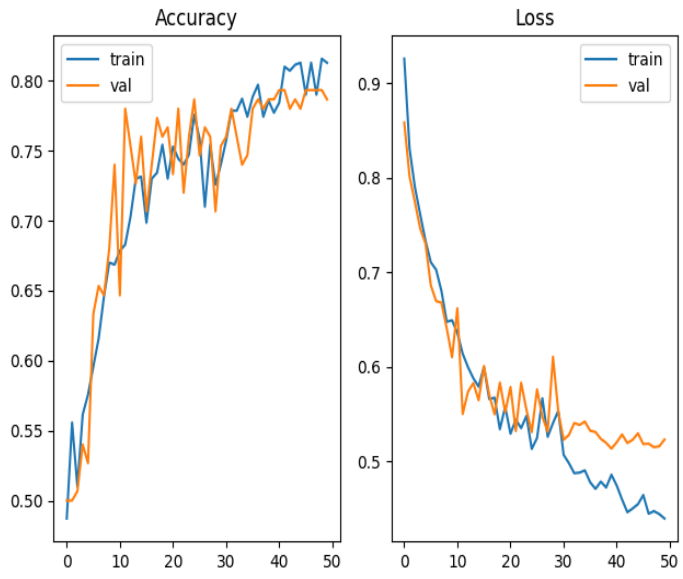
[8] nasumično isključuje neurone tokom treniranja
[9] trudi se da obezbedi ravnomernost pojačanja tj. da se sve osobine uzimaju u obzir
[10] način predstavljanja kategorijskih tj. diskretnih podataka kao binarnih vektora gde svaki element klase ima svoj položaj u vektoru

Model: "sequential_1"

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 160, 160, 3)	0
rescaling (Rescaling)	(None, 160, 160, 3)	0
conv2d (Conv2D)	(None, 160, 160, 32)	896
max_pooling2d (MaxPooling2D)	(None, 80, 80, 32)	0
conv2d_1 (Conv2D)	(None, 80, 80, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 40, 40, 64)	0
conv2d_2 (Conv2D)	(None, 40, 40, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_3 (Conv2D)	(None, 20, 20, 256)	295,168
max_pooling2d_3 (MaxPooling2D)	(None, 10, 10, 256)	0
dropout (Dropout)	(None, 10, 10, 256)	0
flatten (Flatten)	(None, 25600)	0
dense (Dense)	(None, 256)	6,553,856
dropout_1 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 2)	514

```
Total params: 6,942,786 (26.48 MB)
Trainable params: 6,942,786 (26.48 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 12s 897ms/step - accuracy: 0.5357 - loss: 0.9500 -
Epoch 2/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 10s 858ms/step - accuracy: 0.5000 - loss: 0.8436 -
Epoch 3/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 10s 863ms/step - accuracy: 0.4971 - loss: 0.8052 -
Epoch 4/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 10s 890ms/step - accuracy: 0.4957 - loss: 0.7851 -
Epoch 5/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 10s 857ms/step - accuracy: 0.5271 - loss: 0.7682 -
Epoch 6/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 10s 856ms/step - accuracy: 0.5800 - loss: 0.7448 -
Epoch 7/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 9s 828ms/step - accuracy: 0.5929 - loss: 0.7350 -
Epoch 8/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 10s 852ms/step - accuracy: 0.5857 - loss: 0.7365 -
```

Na slikama iznad je prikazana arhitektura mreže prikazana koristeći model.summary().

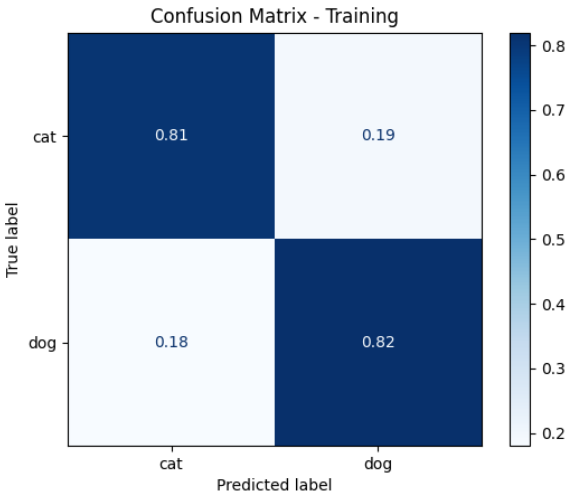


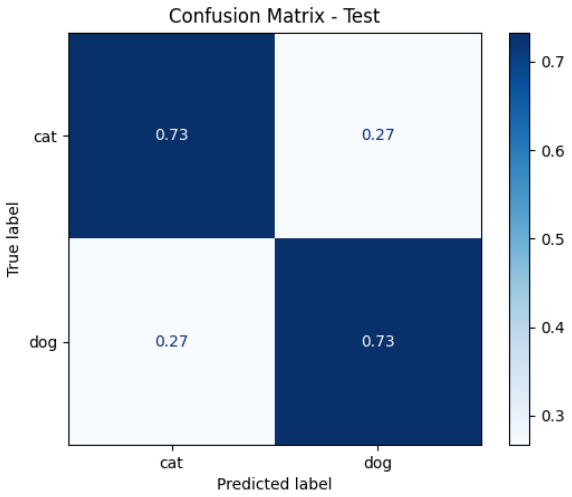

```
Training Accuracy: 79.57%
Training Precision: 79.64%
Training Recall: 79.57%
Training F1: 79.56%

Validation Accuracy: 81.33%
Validation Precision: 81.54%
Validation Recall: 81.33%
Validation F1: 81.30%
Test Accuracy: 72.67%
Test Precision: 72.87%
Test Recall: 72.67%
Test F1: 72.61%
```

Sa grafika iznad vidimo više faktora koji ukazuju na to da model dobro klasifikuje uzorke:

- preciznost na trening podacima raste i dostiže vrednost ~80%
- validaciona kriva takođe raste s tim da pred kraj treniranja vrednost polako stagnira
- validaciona kriva na loss grafiku opada zajedno sa trening krivom, ali oko 30. epohe se već vidi da usporava i do 50. već počinje da raste i tad možemo završiti treniranje inače bi došlo do preobučavanja
- trening kriva na loss grafiku s druge strane nastavlja da opada i približava se nuli što znači da greška kontinuirano opada i do kraja treniranja je praktično eliminisana

Detaljniji prikaz performansi se nalazi na slici desno i na svim setovima performanse su dosledno optimalne.





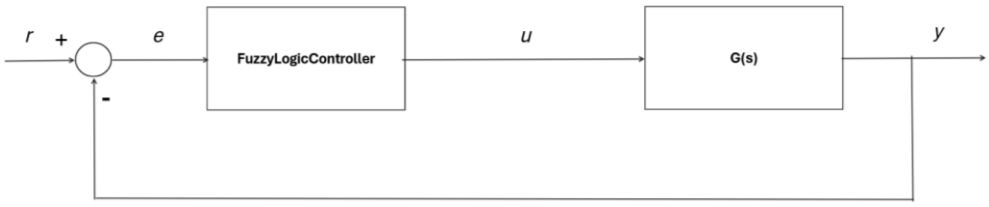
Konfuzione matrice trening i test seta se nalaze na slikama iznad i prikazuju raspodelu predikcija i njihove ispravnosti u formi matrice čije vrednosti se potom koriste (*TP, TN, FP, FN*) za proračunavanje vrednosti performansi klasifikacije kakve smo videli na prethodnoj strani i detaljnije definisali u prvom zadatku.

Na slikama ispod su prikazani primeri dobro i loše sortiranih slika iz skupa podataka.

Projektovanje fuzzy regulatora

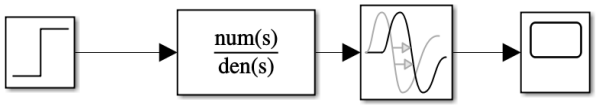
U ovom zadatku smo prvo projektovali jedan sistem fuzzy upravljanja za praćenje referentne vrednosti objekta upravljanja sa zadatim parametrima čije su vrednosti prikazane na slici ispod. Na slici se takođe nalazi i konceptualna šema sistema koja će biti detaljnije razrađena u modelu izgrađenom u MATLAB alatu.

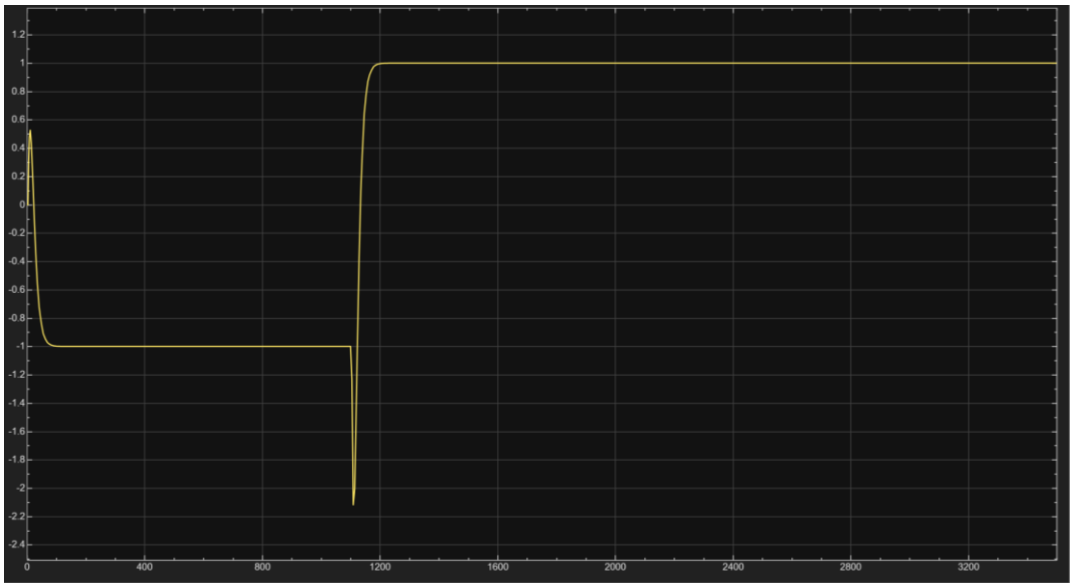
8	$G(s) = \frac{-0.2s+0.01}{(s+0.1)^2} e^{-4s}$	$r \in [-1, +1]$	$u \in [-1.5, +1.5]$



Pre same izrade modela, za pristup projektovanju opredelili smo se za *intuitivni pristup*.

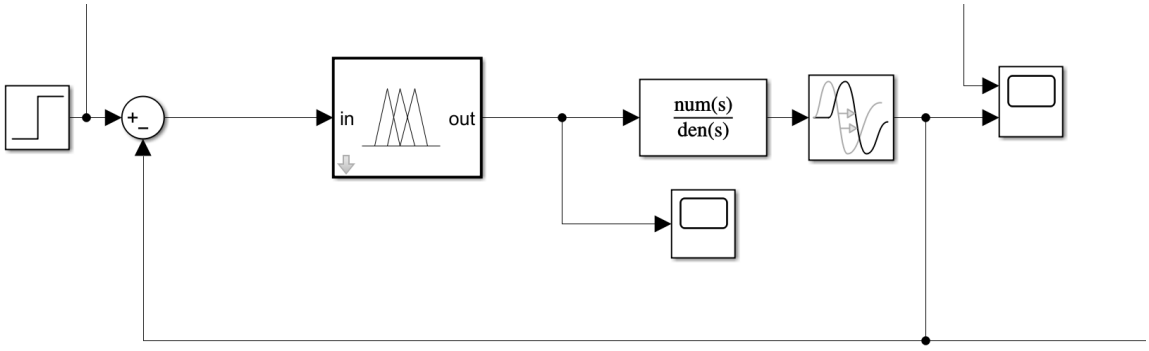
Kao prvi korak postupka povezali smo objekat upravljanja realizovan kao komponenta Transfer Fcn sa parametrima koji su nam dati za G(s) uz dodatak Delay komponente koja se odnosi na deo e^{-4s} na Step blok koji generiše (pobudni) signal koji je na gornjoj slici definisan kao varijabla r. Odavde iz Scope bloka vidimo ponašanje objekta tj. trenutne vrednosti sistema za ulazne vrednosti (pobudni signal).



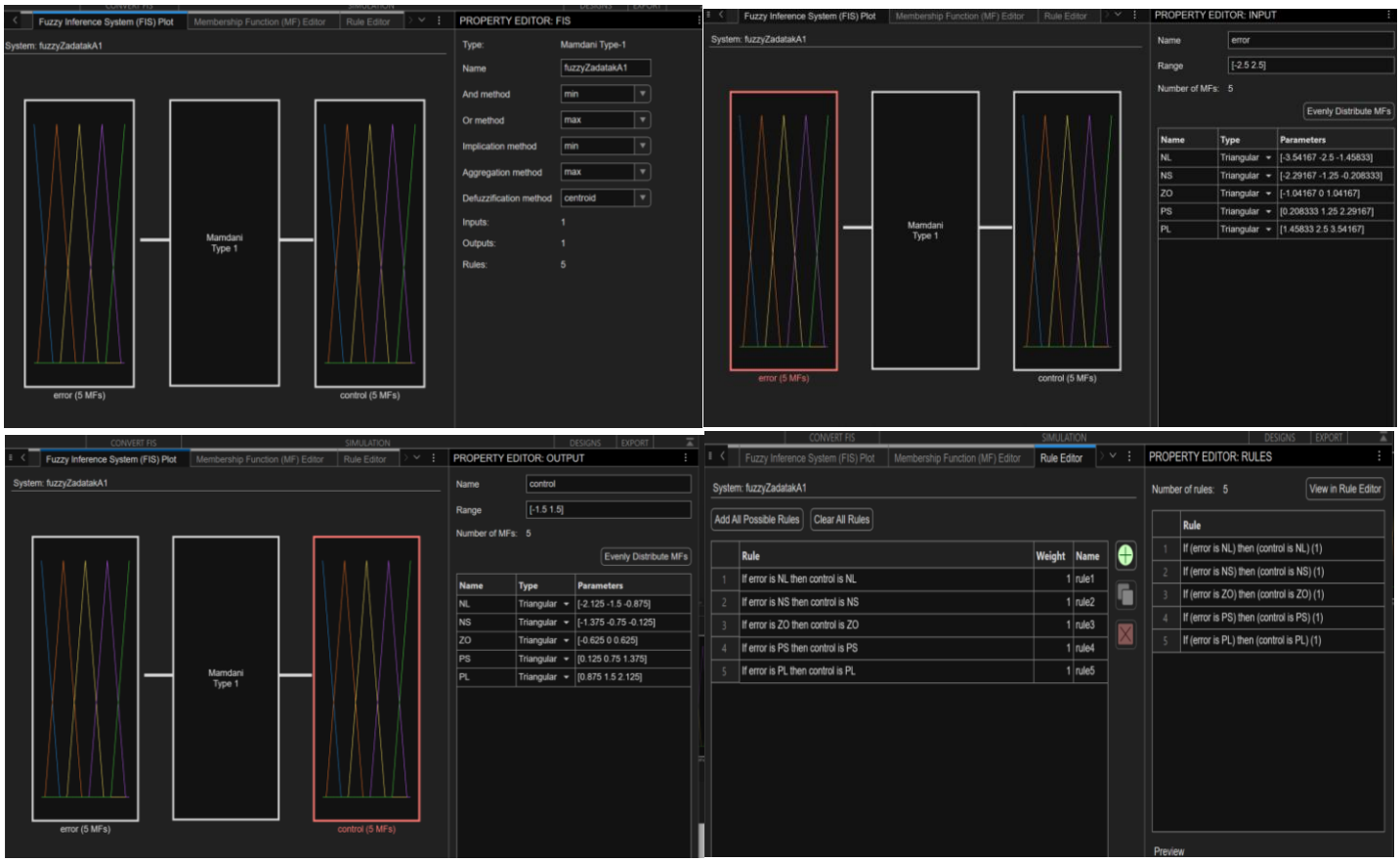


Iz prethodnog koraka vidimo da funkcija već ima željeno ponašanje koje treba samo obezbediti da ostane prisutno u zatvorenoj povratnoj sprezi. Greška u ovom slučaju je **nula** prema s obzirom da je trenutno stanje jednako željenom od početka do kraja.

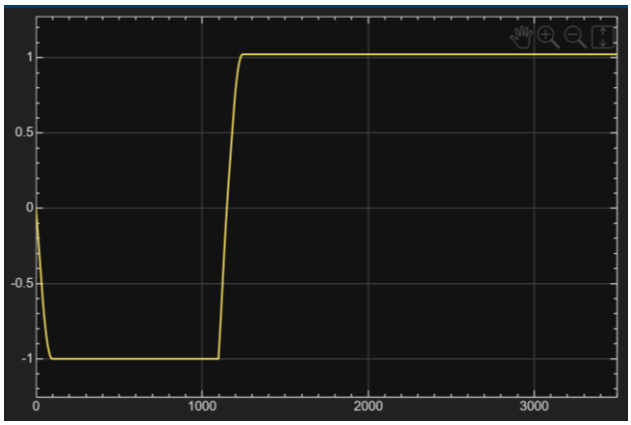
Naredni korak je bio izgradnja modela koji je realizovan u zatvorenoj sprezi (prikazan na slici ispod).

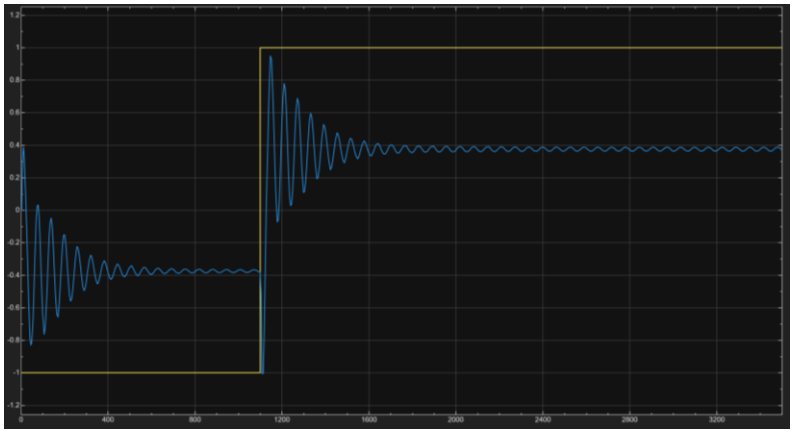


Fuzzy Logic Controller je realizovan na sledeći način:



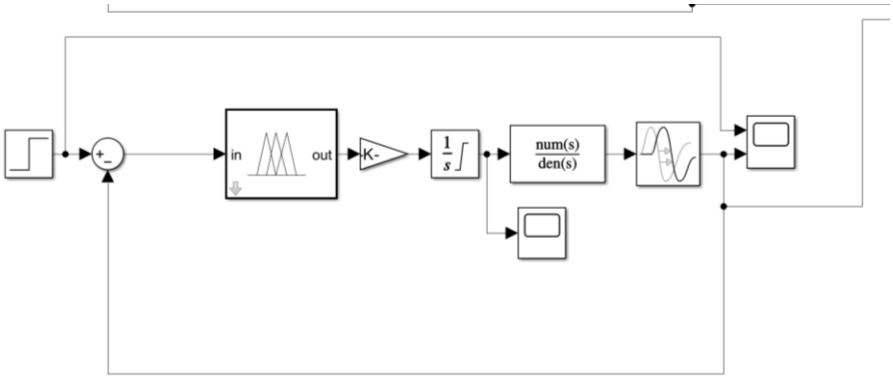
Iz ovog sistema dobijamo sledeće signale upravljanja (prvi grafik) i regulisane varijable (drugi grafik;plavi signal).



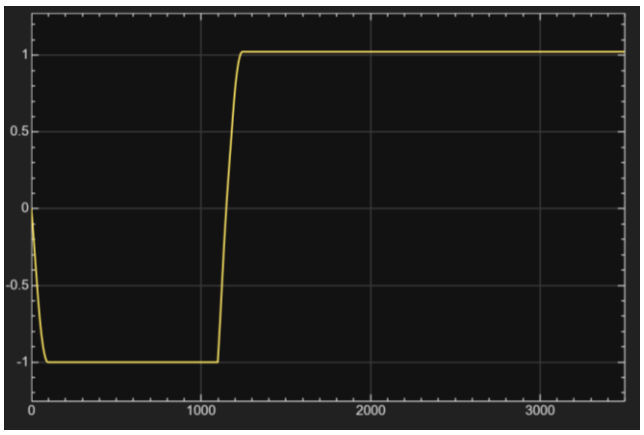


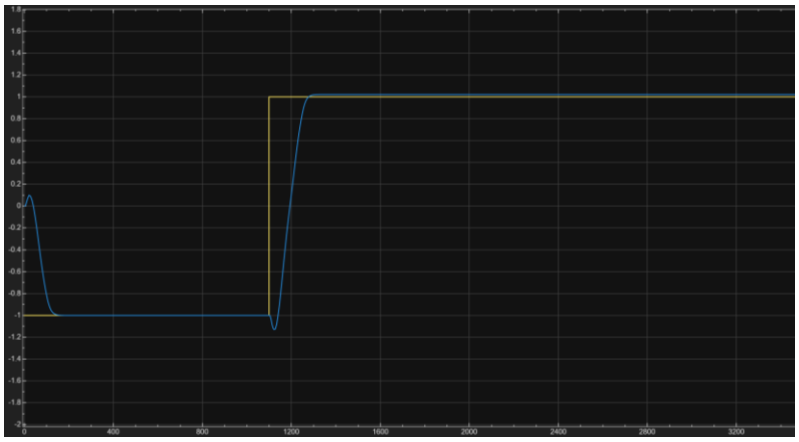
Prvi problem je što se regulisana varijabla manifestuje kao prigušena oscilacija i drugi problem je što oscilira oko vrednosti -0,4 i 0,4 umesto -1 i 1.

Uvodimo Integrator koji akumulira izlaze i time obezbeđuje izbegavanje problema sa potencijalno nultom greškom tj. odstupanjem izlaza od refence. Takođe uvodimo i Gain i testiranjem otkrivamo da je optimalna vrednost (ili bar najbliža optimalnoj) 0,2.



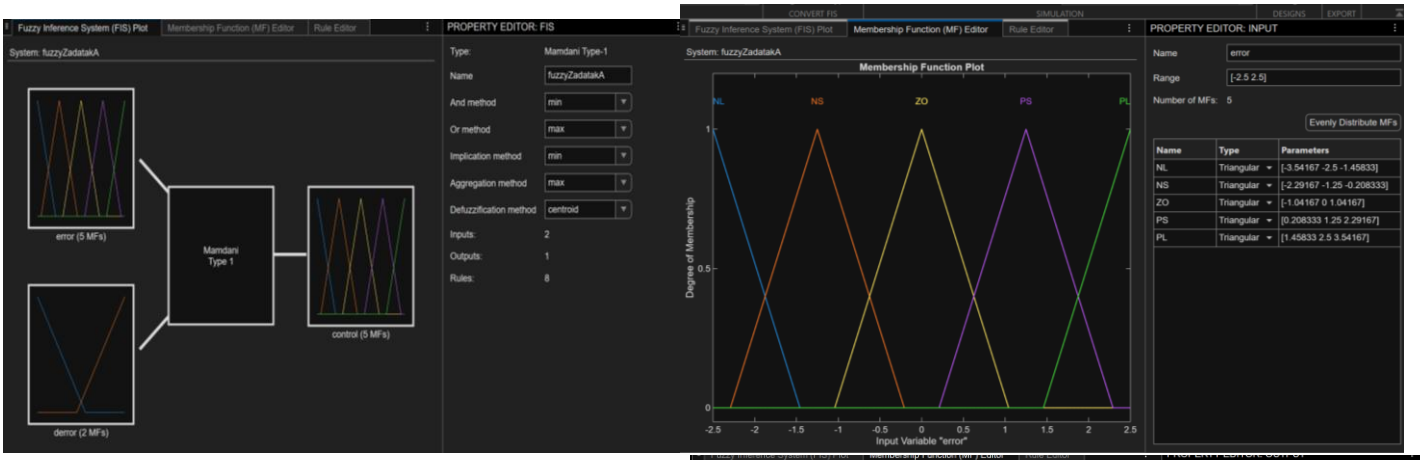
Ovi dodaci nas dovode do sledećih promena na graficima funkcija signala upravljanja i regulisane varijable:

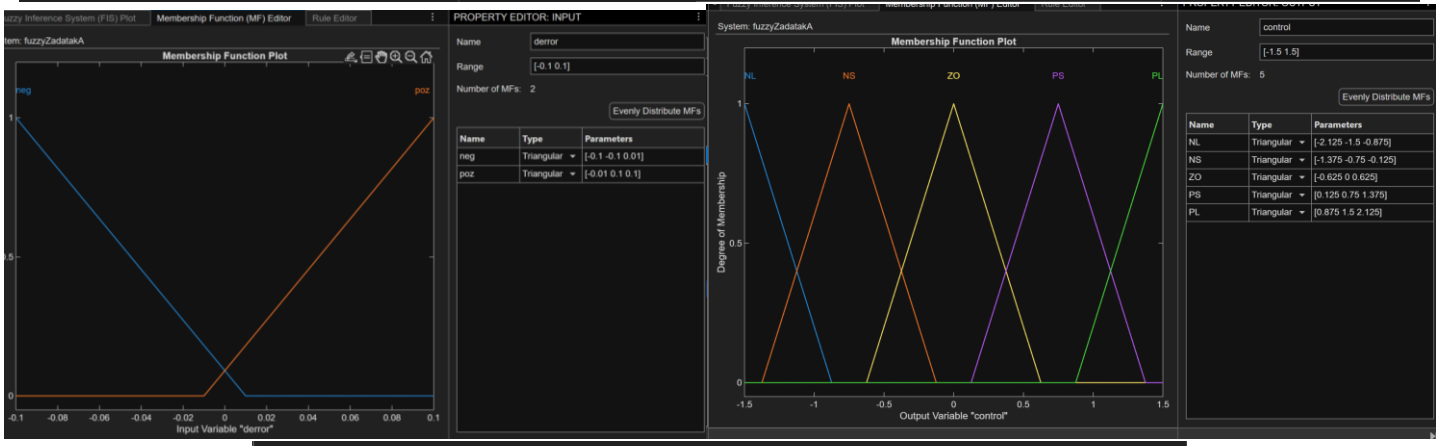




Signal upravljanja se poklapa sa referencom i verovatno bismo ovde mogli stati ali kao što primećujemo odziv na step sa minimalne na maksimalnu vrednost je relativno spor tj. može se unaprediti. Iz tog razloga uvodimo naredne i poslednje promene.

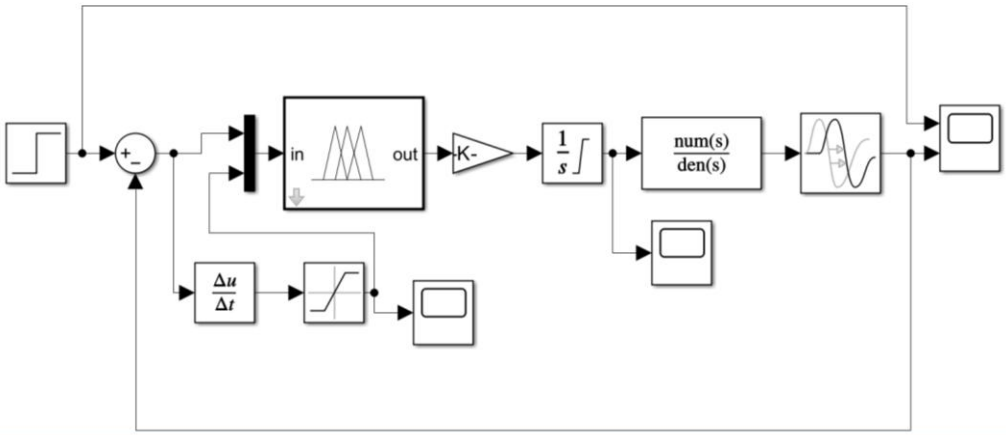
U FIS fajl tj. Fuzzy Logic uvodimo još jedan input – prvi izvod signala greške. Na njega će se dovesti izlaz bloka Derivative sa finalne šeme (koji je prethodno „filtiran“ kroz blok Saturation).



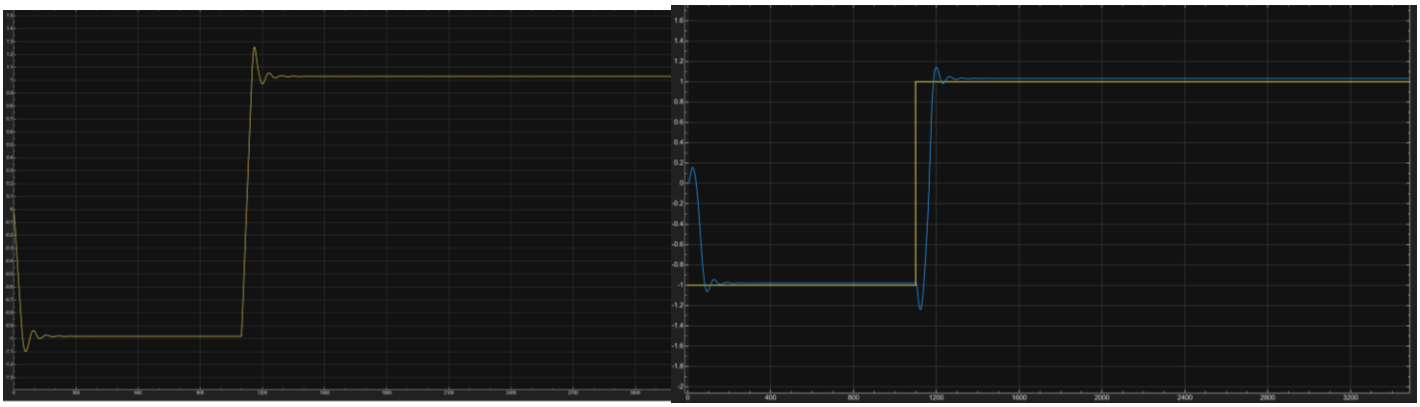


Add All Possible Rules | Clear All Rules

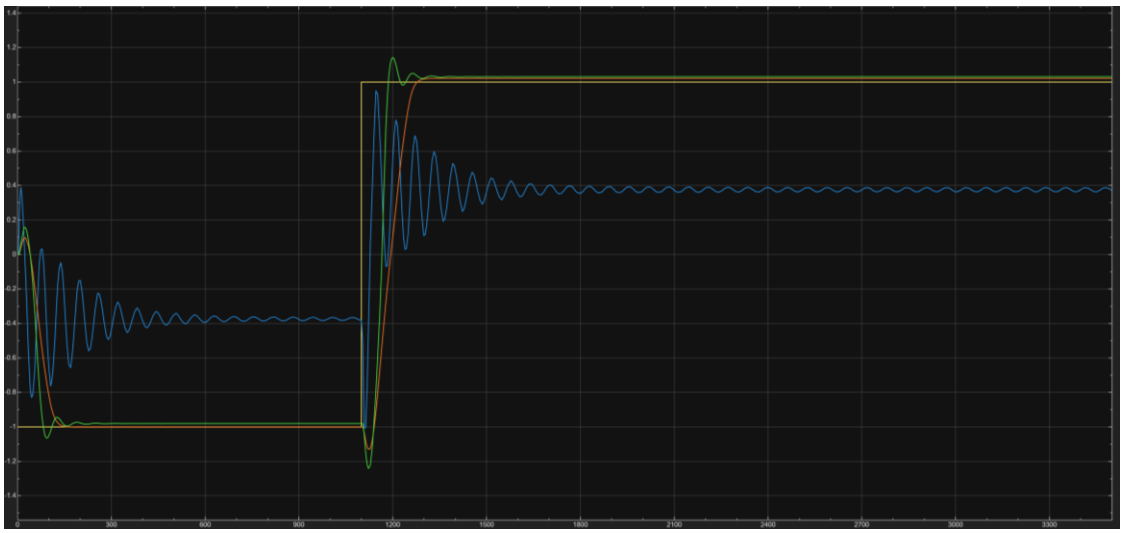
	Rule	Weight	Name
1	If error is NL then control is NL	1	rule1
2	If error is NS and derror is neg then control is NL	1	rule2
3	If error is NS then control is NS	1	rule3
4	If error is ZO and derror is neg then control is NS	1	rule4
5	If error is ZO then control is ZO	1	rule5
6	If error is ZO and derror is poz then control is PS	1	rule6
7	If error is PS and derror is poz then control is PL	1	rule7
8	If error is PL then control is PL	1	rule8



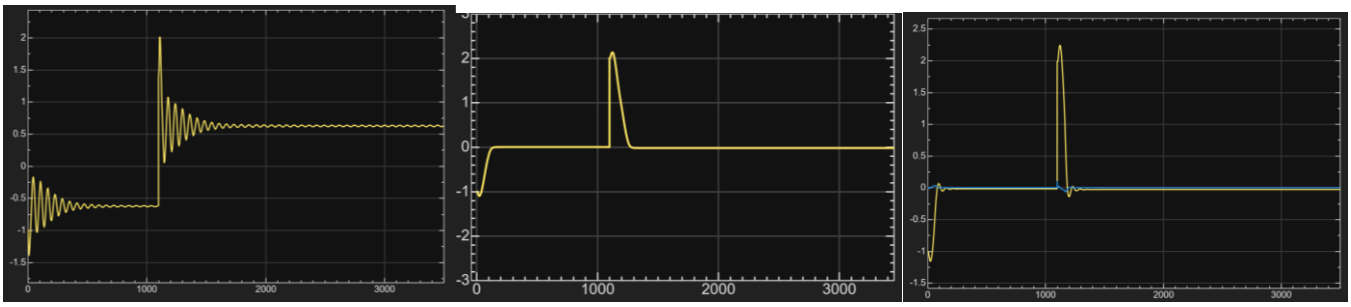
Na ovaj način dolazimo i do finalnog izgleda funkcija signala upravljanja i regulisane varijable:



Svi vremenski oblici direktno upoređeni:



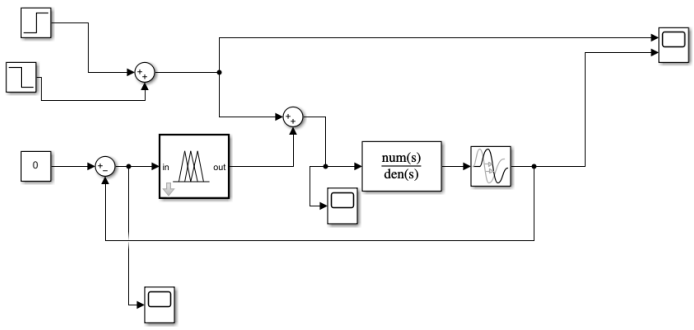
Takođe imamo i vremenske oblike signala na neposrednom ulazu u fuzzy inference sisteme s tim da mislim da smo izdvojili u toku prethodnih analiza pojedinačnih stepena razvoja šeme sve implikacije koje one nose kao što su nulti ulaz i prikaz pravila i izlaza. Prikazana su 3 oblika, po jedan za svaku verziju modela (poslednji je finalni):



U drugom delu zadatka smo projektovali sistem fuzzy upravljanja za potiskivanje nemerljivog poremećaja na ulazu objekta upravljanja s parametrima koji su zadati i prikazani na početku 8. strane izveštaja. Ovde smo se takođe odlučili za intuitivni pristup projektovanju.

Problem možemo dekomponovati na model poput onog iz prethodnog dela tj. zadatka sa adicijom poremećaja. Vrednosti ograničenja upravljanja kao i parametri objekta upravljanja ostaju isti dok se referentna vrednost menja tako što je sad konzistentno 0 (**r=0**) i maksimalna amplituda poremećaja je jednaka polovini maksimalne amplitude upravljačkog signala što je u našem slučaju **-0,75** i **0,75**.

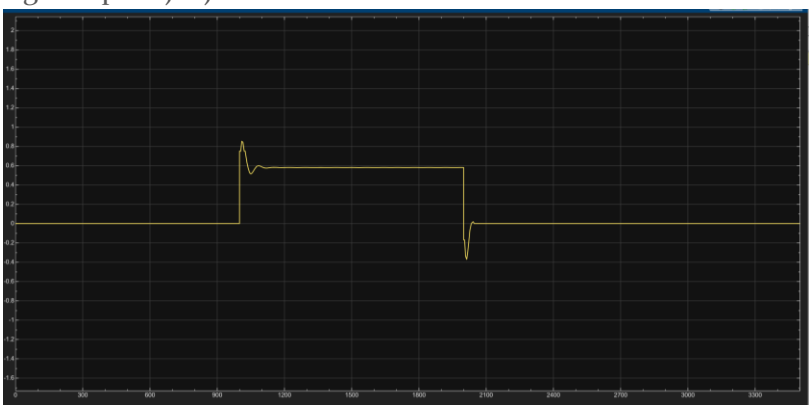
U skladu sa ovim početna šema izgleda ovako:



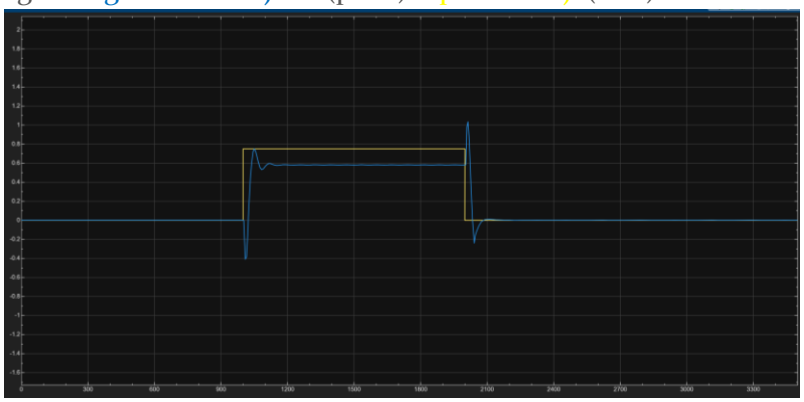
Logika fuzzy kontrolera za ovu šemu je ista kao ona prikazana na 9. strani izveštaja (*fuzzyZadatakA1.fis*) s tim da je output tj. control skaliran na opseg od -0,75 do 0,75. Kao što smo već spomenuli jedina razlika u odnosu na prvu šemu iz prethodnog zadatka je dodatni poremećaj koji se dovodi na izlaz fuzzy kontrolera tj. poremećaj se dodaje na izlaz fuzzy control bloka.

Na slikama ispod vidimo:

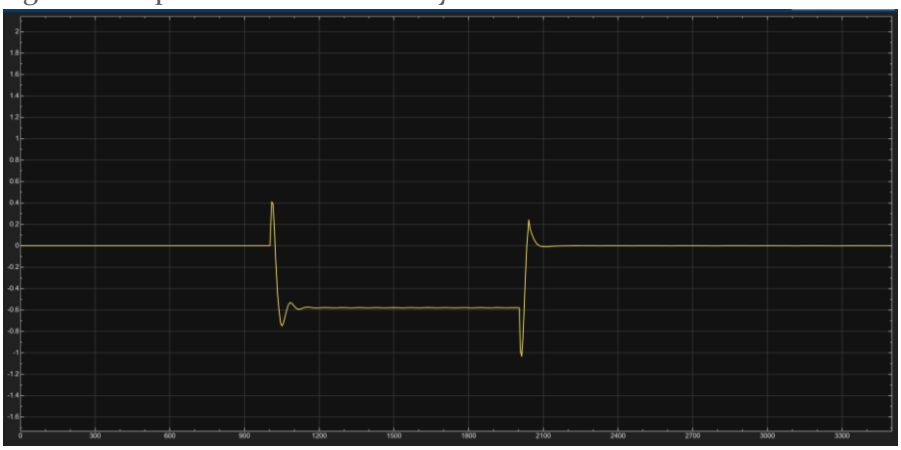
a) signal upravljanja



b) signal regulisane varijable(plavi) + poremećaja(žuti)

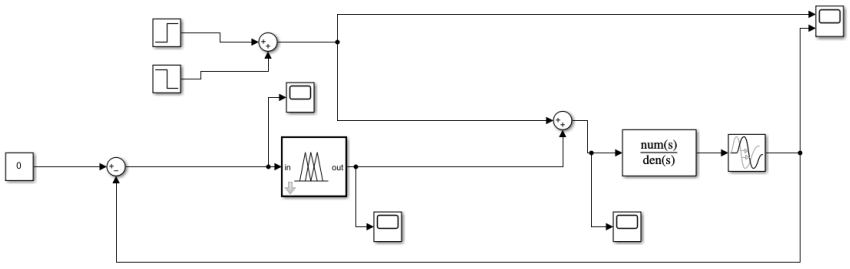


c) signal na neposrednom ulazu fuzzy kontrolera

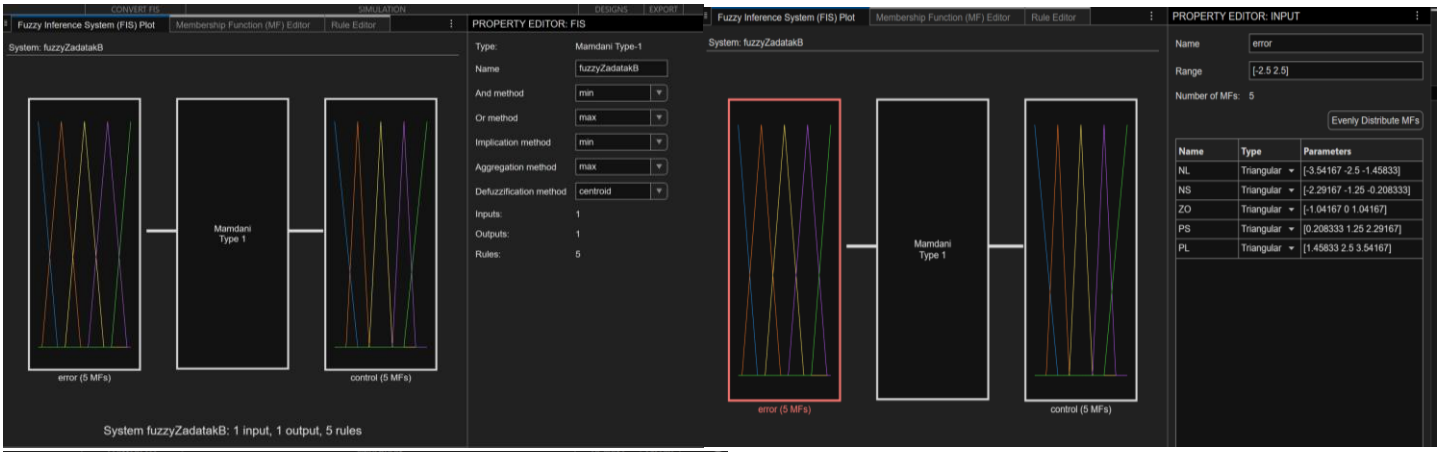


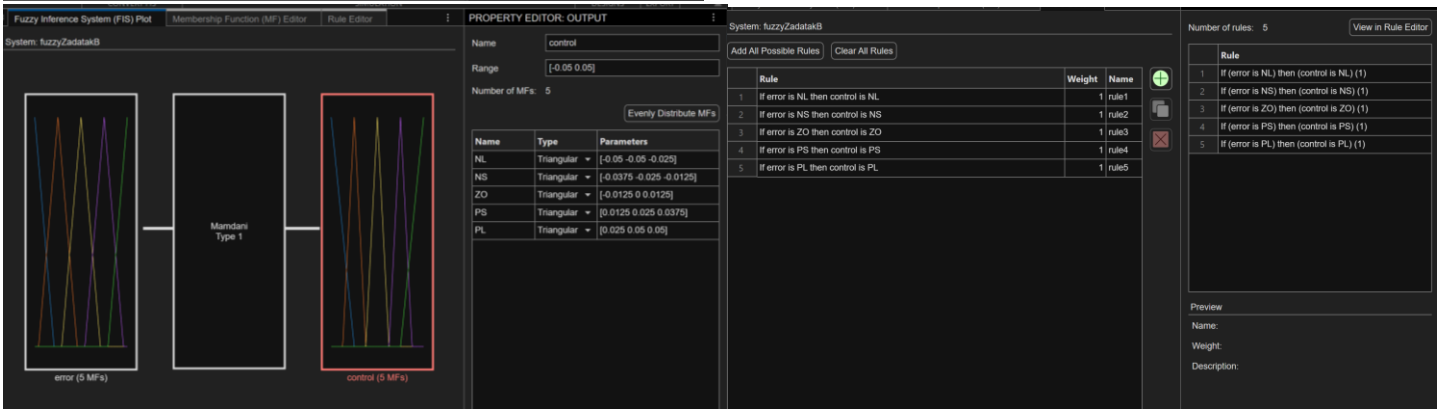
Pošto funkcije imaju već željeno ponašanje u smislu zasićenja i oblika jedino što nam ostaje je prilagođavanje tj. ***skaliranje*** dela funkcije kada se dođe početni poremećaj od +0.75 u trenutku 1000.

Nova šema suštinski ostaje ista osim značajne promene koja na slici nije vidljiva a to je promena u samoj logici tj. šema kontrolnog bloka.



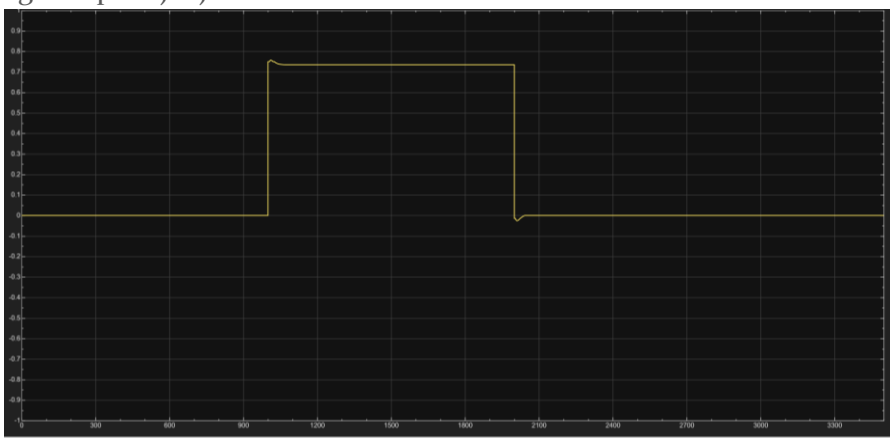
Ona sada izgleda ovako:



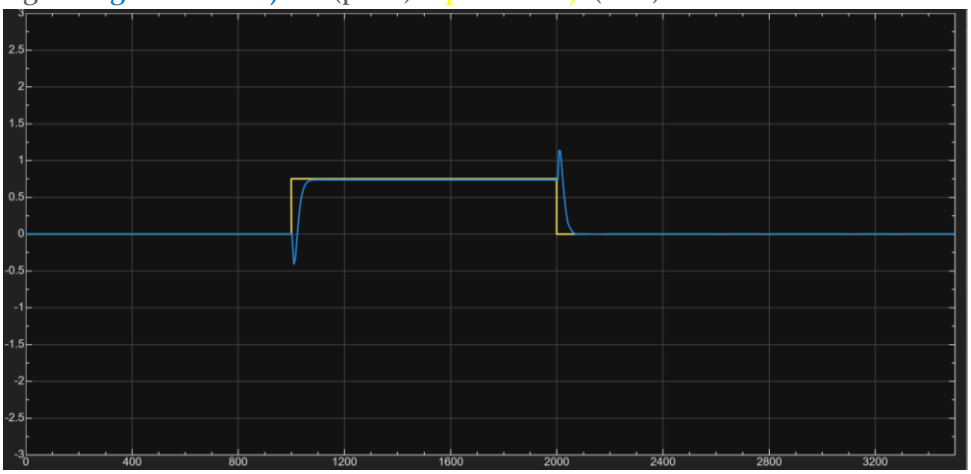


Na slikama vidimo signale koji se dobijaju nakon izmena:

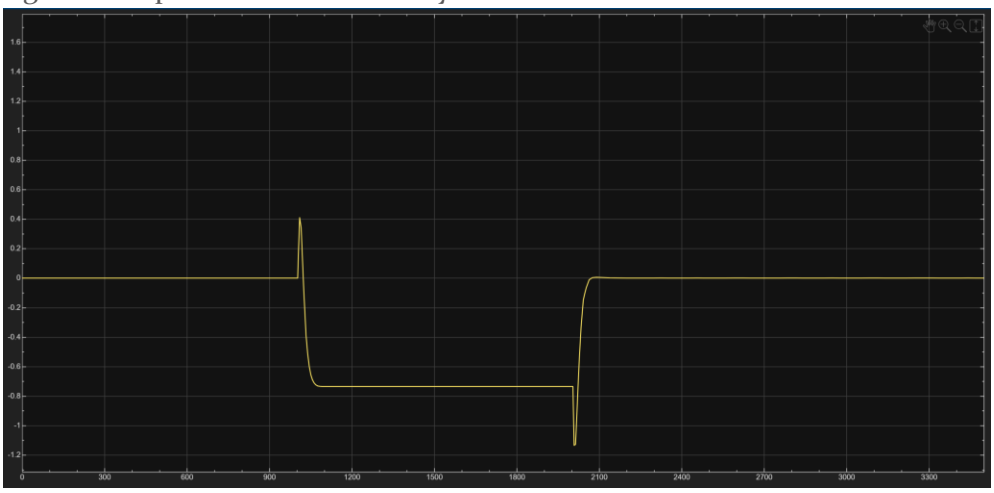
a) signal upravljanja



b) signal regulisane varijable(plavi) + poremećaja(žuti)



c) signal na neposrednom ulazu fuzzy kontrolera



U prethodnom sistemu smo se pretežno bavili razrešavanjem problema usled oscilacija a potom i problemom skaliranja dok je u slučaju poremećaja značajni problem bio skaliranje signala.

U drugom sistemu je akcenat bio na prilagođavanju na promene na izlazu fuzzy kontrolnog bloka dok je u prvom bilo prilagođavanje na promene na ulazu tj. promene upravljačkog signala nasuprot promeni reference.